A Bootstrap Validated Parsimonious Cox PH Regression Model Predicting AIDS Survival

Madison Hobbs & Lathan Liou

May 2, 2019

Introduction

The AIDS epidemic exploded in the early 80s, and the scientific community raced to find curative treatments. One such research group, Hammer et. al., sought to perform a study using a double-blind placebo controlled trial to compare the three-drug regimen of indinavir (IDV), zidovudine (ZDV) or stavudine (d4T) and lamivudine (3TC) with the two-drug regimen of ZDV or d4T and 3TC in HIV-infected patients. Specifically, they were interested in testing the effects of IDV, or as it's more commonly known, Crixivan. The discoverer of Crixivan, Paul Reider (who lectured at Pomona last year as a Robbins Chemistry Fellow!), had high hopes for this drug given its unique 3-D binding capacity to inhibit specific proteins in HIV.

Having obtained the data from the study, our project seeks to understand the factors associated with time of survival until an AIDS defining event or death. We first seek to understand the distributions of our time-to-event variables and the other explanatory variables through an exploratory data analysis. Next, we aim to identify the most important predictors by using a Cox Proportional Hazards model and an XGBoost model. Lastly, we seek to validate our model by a bootstrap analysis. We also have hidden most of our code in this report for readability purposes; however, you can find our full code on our GitHub: https://github.com/latlio/SurvivalAnalysis.

Our analysis illuminates the importance of lower CD4 count and the IDV treatment as predictors associated with lower risk of AIDS diagnosis or death. We also find that Karnofsky score is very indicative of patients' risk of diagnosis or death, as higher Karnofsky scores signal longer survival times.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is, and should be, the *first* step when working with data. Our primary objective is to visualize the distributions of our variables to gain a clearer understanding of how they vary.

A Note About Treatments

According to the variable information table, we note that `txgrp` could have four levels (1: ZDV + 3TC, 2: ZDV + 3TC + IDV, 3: d4T + 3TC, and 4: d4T + 3TC + IDV). However, this dataset contains only two levels of `txgrp` (1: ZDV + 3TC, 2: ZDV + 3TC + IDV).

In fact, since the variable `tx` is supposed to indicate whether the treatment contained IDV, we might assume that `txgrp` and `tx` are redundant information in this particular dataset and that a 1 in `txgrp` is equivalent to a 0 in `tx` while a 2 in `txgrp` is equivalent to a 1 in `tx`. We confirm this hunch below.

The following code says: create a new dataframe by taking all the rows in `data` where `txgrp` is 1 and `tx` is 0 *or* `txgrp` is 2 and `tx` is 1. Now, make sure that new dataframe is identical to the original data frame, and return `TRUE` if this is indeed the case.

```r
# Is it true that for every entry in `aids`
all(
  (aids %>%
    filter(((txgrp == 1 && tx == 0) || (txgrp == 2 && tx == 1))))
  == aids) == TRUE
```

```
## [1] TRUE
```

Exploring Correlation

From an initial pair-wise correlation plot between our explanatory variables in our data (not shown here), we noticed that `cd4` and `strat2` have a correlation coefficient of 0.74, which indicates moderate to strong correlation. This made us realize that `strat2` is the indicator variable for the continuous variable, `cd4`. Additionally, we confirmed that `tx` and `txgrp` are identical because they were perfectly correlated. Lastly, we would like to note that `sex`, `ivdrug`, and `hemophil` are highly unbalanced variables, meaning that one level of the variables is disproportionately represented relative to the other level(s).

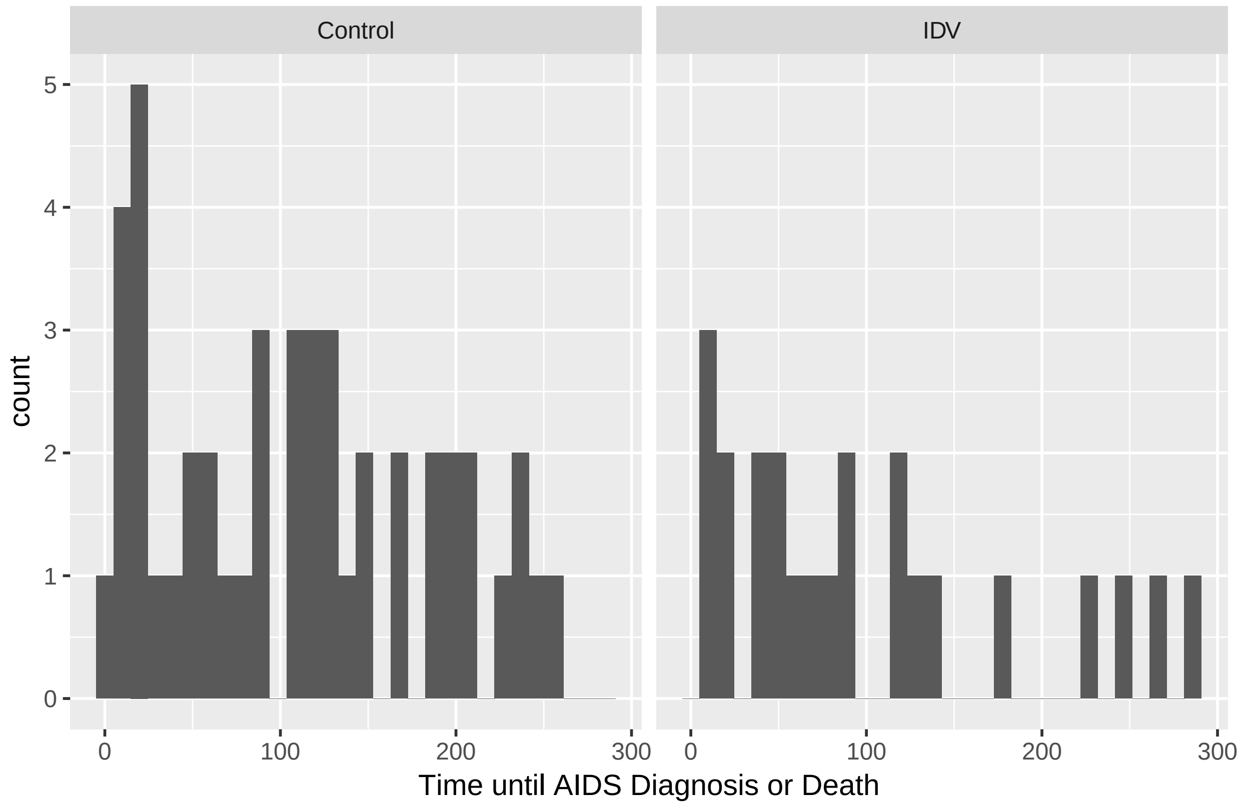
A Note about Censored vs. Non-Censored

It's worth noting that there are, in fact, two censored time-to-event variables. The primary variable of interest is `time` which is time in days to AIDs diagnosis or death, and this is informed by `censor`, which is 1 (true) if an individual was either diagnosed with AIDS *or* died during the course of the study and 0 otherwise. The other censored variable is `time_d` which is the time in days to death alone, governed by `censor_d` which is 1 if the person died during the study and 0 if not.

Since the primary variable of interest is time to AIDs diagnosis or death, we examine the complete (non-censored) individuals - those who were either diagnosed with AIDS *or* who died over the course of the study. The only caveat is that there are only 69 such individuals out of a study of 851 - most of the participants did not die or get diagnosed before the study's end.

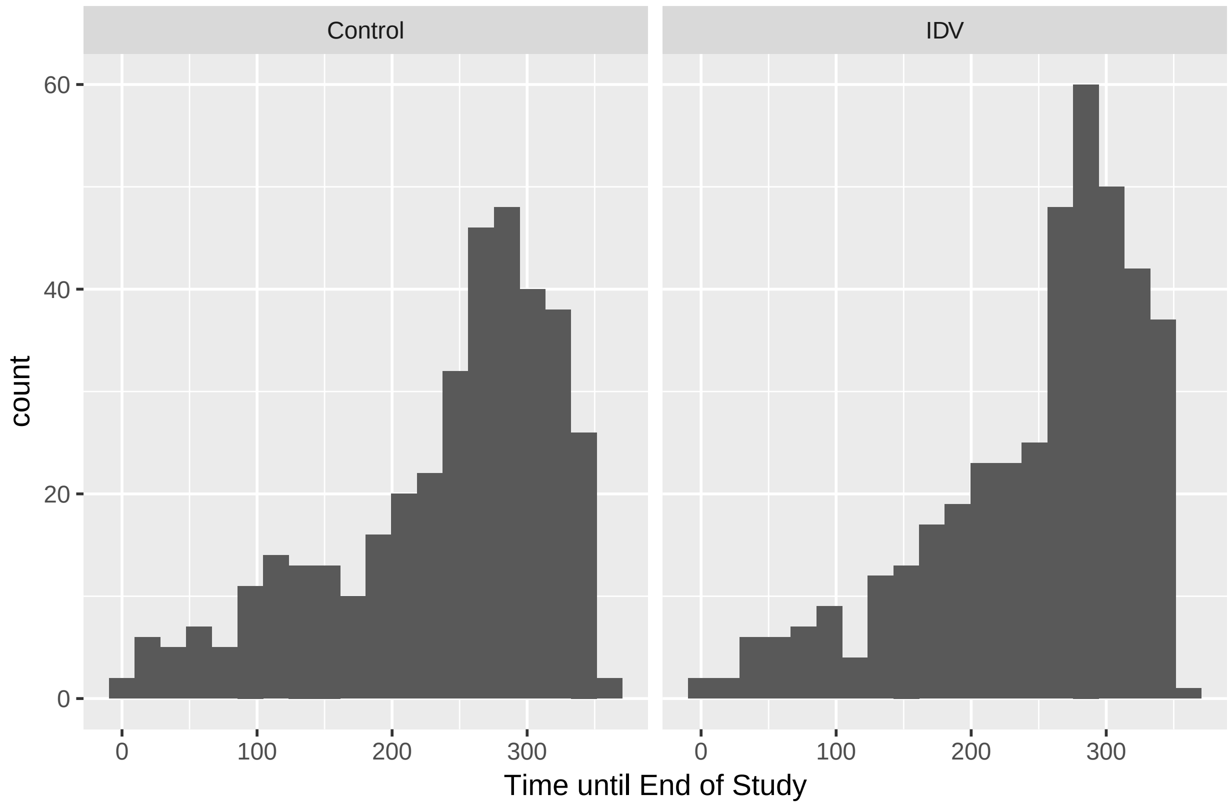
Among those with complete times, we notice from the side-by-side histograms below that both the control and IDV groups are skewed right. This makes sense - for complete observations, it's probably less common for people to last a long time without being diagnosed or dying. The distributions between the control and IDV groups don't look that different, however, especially given the tiny sample sizes.

Figure 1. Comparing the Distribution of Complete Time to Event Observations Between Control and Treatment



When looking at the censored (incomplete) times for diagnosis/death, both control and IDV groups are skewed left in the opposite direction.

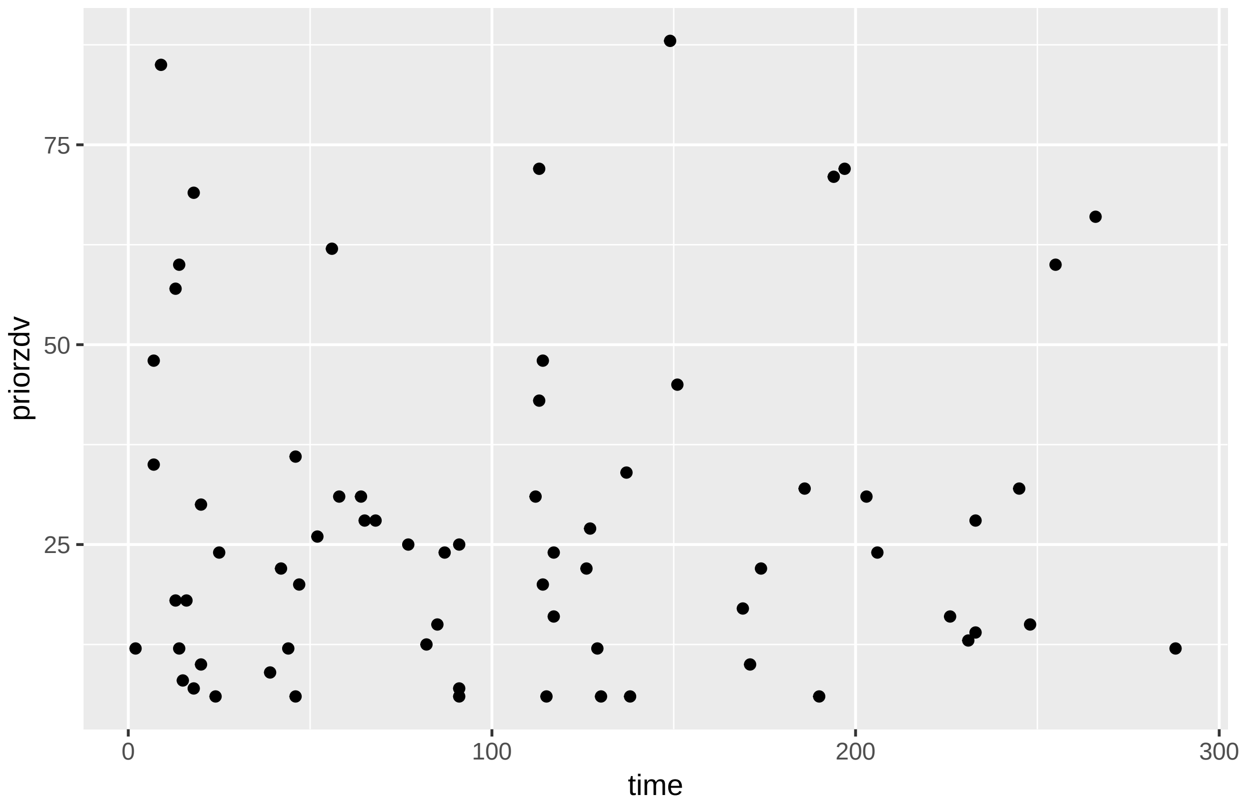
Figure 2. Comparing the Distribution of Incomplete Time to Event Observations Between Control and Treatme



Prior ZDV on Complete Observations

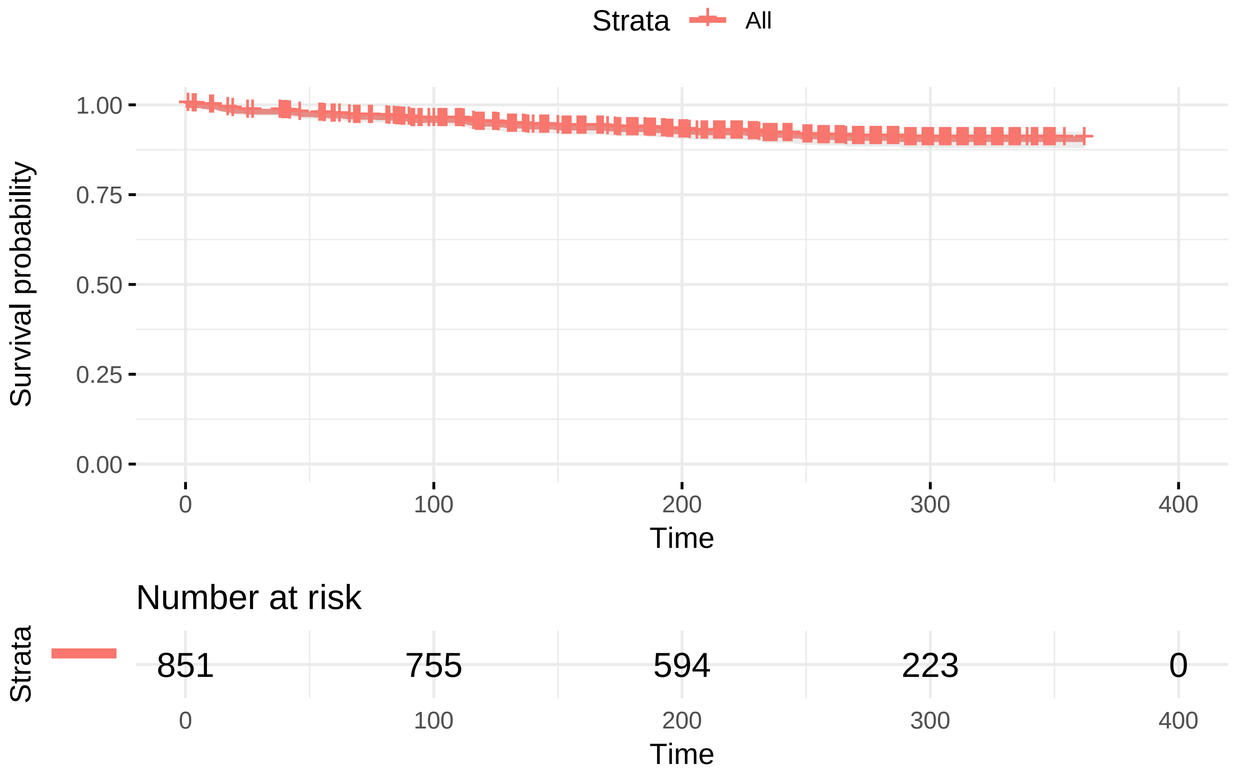
We were also curious about the relationship between time to diagnosis/death and number of months of prior ZDV use for non-censored participants, since ZDV is one of the drugs in both the treatment and control regimen. Interestingly, there appeared to be no relationship whatsoever, as evidenced by the following scatterplot:

Figure 3. Scatterplot of Number of Months of Prior ZDV Treatment vs. Time to Death/Diagnosis

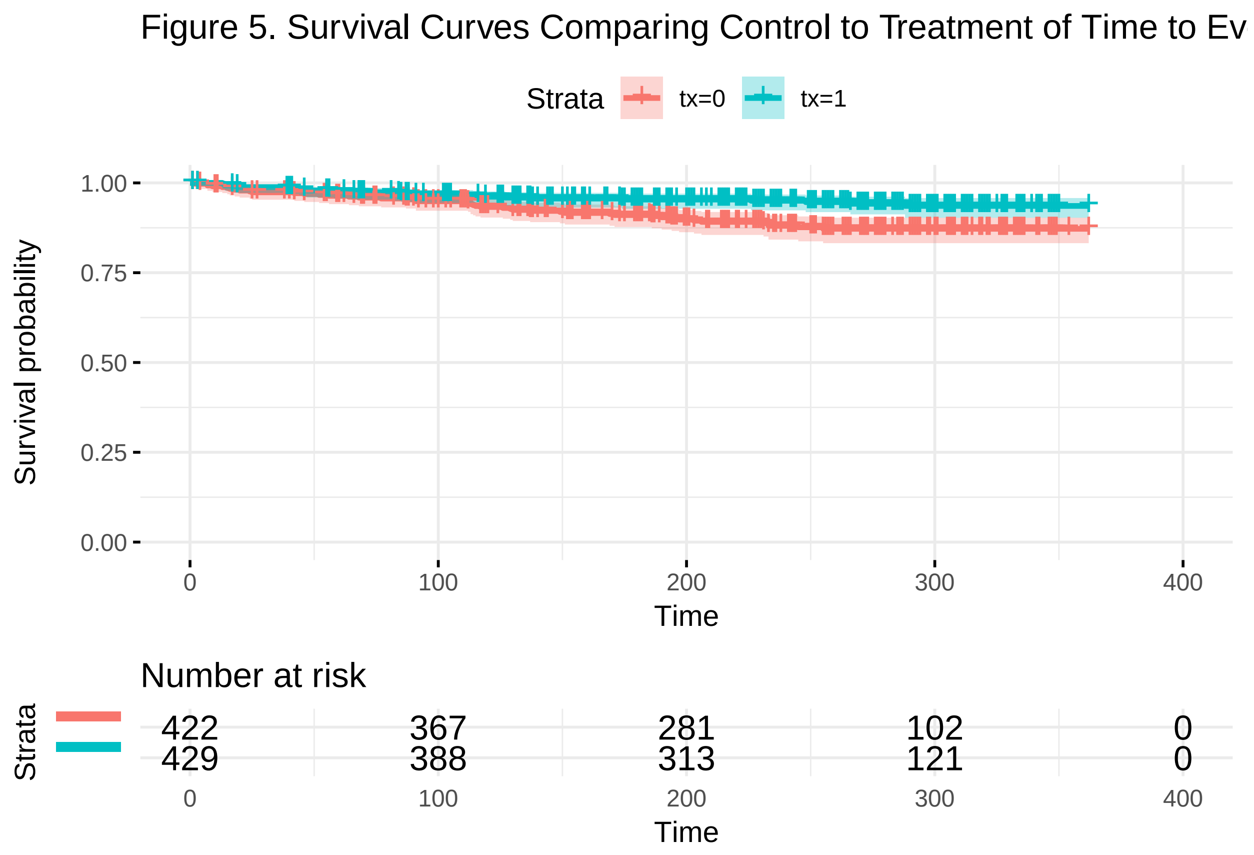


Plotting Survival Curves

Figure 4. Survival Curve of Time to Event Variable



We also fit a survival curve of just the time to death variable. We observe that the overall survival probability of our sample remains relatively high over time and that the last observation is censored. Because treatment is the clinical variable of interest, we next want to see how the survival curves differ between the two treatment groups.



```r
#perform log-rank test
survdiff(Surv(time, censor) ~ tx, data = data, rho = 1)
```

```
## Call:
## survdiff(formula = Surv(time, censor) ~ tx, data = data, rho = 1)
##
##        N Observed Expected (O-E)^2/E (O-E)^2/V
## tx=0 422     44.0     31.9      4.57      9.24
## tx=1 429     22.1     34.2      4.28      9.24
##
##  Chisq= 9.2  on 1 degrees of freedom, p= 0.002
```

We see that the treatment group for which IDV was also administered has a higher survival probability over time compared to the control group. Performing the log-rank test to test our null hypothesis of whether $S_0(t) = S_1(t)$ for all t results in a $\chi^2_1 = 9.2$ with a p-value of 0.002. We thus reject the null hypothesis and conclude that there is evidence that supports that the survival probabilities are significantly different between the treatment groups over some time intervals.

Overall, from our Exploratory Data Analysis, we have several key takeaways that will guide us in our modeling process:

- We will use ‘tx‘ and not ‘txgrp‘ in our model since they are identical in this particular dataset
- We will use ‘cd4‘ and not ‘strat2‘ in our model since ‘strat2‘ is the indicator variable for cd4.
- ‘Priorzdv‘ may not be an important predictor in our later models, e.g. our Cox PH model, whereas Treatment might be.

Cox Proportional Hazards Model

Choosing the Number of Parameters

Our goal is to develop a multivariable survival model for time until death (or diagnosis). In particular, our objective is to build the best predictive model, i.e. we want the highest accuracy on new data. There are 69 deaths (or diagnoses) among 782 patients. The first thing we want to assess is a full additive model. Thus, categorical predictors were expanded using dummy variables. We chose not to include `txgrp` and `strat2` because they were derived (and thus highly correlated with) from other predictor variables.

First, we make sure that the technical condition for proportional hazards is met with the hypothesis test below.

```
cox.zph(coxph(Surv(time,censor) ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + priorzdv + age
```

```
##                rho    chisq      p
## tx1       -0.12102 1.07e+00 0.3009
## sex2      -0.15899 1.86e+00 0.1726
## raceth2    0.19389 2.84e+00 0.0917
## raceth3    0.12322 1.04e+00 0.3086
## raceth4   -0.07600 4.48e-01 0.5032
## raceth5    0.15584 1.34e+00 0.2472
## ivdrug2   -0.18070 7.26e-08 0.9998
## ivdrug3   -0.07277 3.54e-01 0.5516
## hemophil1  0.05969 2.08e-01 0.6486
## karnof80   0.00843 5.08e-03 0.9432
## karnof90   0.02646 5.60e-02 0.8129
## karnof100 -0.06425 2.88e-01 0.5912
## cd4        0.12594 1.02e+00 0.3120
## priorzdv   0.06080 1.78e-01 0.6732
## age        0.15026 1.68e+00 0.1950
## GLOBAL          NA 1.24e+01 0.6448
```

Since no p-values are significant at the $\alpha = 0.05$ level, we conclude that there are no violations of the proportional hazards assumption, so we continue building our CoxPH model. We first build a full additive model.

```
options(scipen = 999)
fit <- coxph(Surv(time, censor) ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + priorzdv + age
fit %>% tidy()
```

```
##         term      estimate      std.error    statistic       p.value
## 1        tx1   -0.692689185    0.259981564 -2.664378094 0.00771308167
## 2       sex2    0.433955810    0.330969192  1.311166782 0.18980142225
## 3    raceth2   -0.266199123    0.306770367 -0.867747189 0.38553274649
## 4    raceth3   -0.133104042    0.350590733 -0.379656474 0.70420043356
## 5    raceth4    0.880199385    0.740160204  1.189201176 0.23436051188
## 6    raceth5    0.159164019    1.062271128  0.149833705 0.88089581703
## 7    ivdrug2 -13.662745199 2217.947151763 -0.006160086 0.99508499357
## 8    ivdrug3   -0.518173596    0.371149088 -1.396133284 0.16267436244
## 9  hemophil1    0.505671649    0.620652558  0.814741907 0.41522005977
## 10  karnof80   -0.690301943    0.429072156 -1.608824839 0.10765464789
## 11  karnof90   -1.349833322    0.428707075 -3.148614520 0.00164046432
## 12 karnof100   -1.810573691    0.475762899 -3.805621862 0.00014144835
## 13       cd4   -0.015164686    0.003190856 -4.752544063 0.00000200873
## 14  priorzdv   -0.002115753    0.004726267 -0.447658398 0.65439974999
## 15       age   0.025738322    0.013850413  1.858307191 0.06312540323
##        conf.low    conf.high
## 1  -1.202243688 -0.183134682
## 2  -0.214731886  1.082643506
## 3  -0.867457993  0.335059747
## 4  -0.820249252  0.554041168
## 5  -0.570487958  2.330886729
## 6  -1.922849135  2.241177172
## 7          -Inf          Inf
## 8  -1.245612442  0.209265250
## 9  -0.710785012  1.722128310
## 10 -1.531267916  0.150664030
## 11 -2.190083750 -0.509582895
## 12 -2.743051839 -0.878095543
## 13 -0.021418649 -0.008910722
## 14 -0.011379067  0.007147561
## 15 -0.001407989  0.052884633
```

After considering whether variables can be mutating into new variables based on our conventional knowledge and finding none, we decided to try shrinkage to reduce our dimensionality. Here, we're using a lasso penalty Cox PH regression model to select our most important features. In brief, lasso essentially imposes an absolute value threshold penalty on each of the β coefficients such that the variables that don't contribute much to explaining the response have their coefficients shrunk to zero.

```r
set.seed(47)
#initialize covariate matrix
x <- model.matrix(Surv(time, censor) ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + priorzdv +
#cross validate lambda
cv.fit <- cv.glmnet(x, Surv(data$time, data$censor), family = "cox", maxit = 1000)

lassofit <- glmnet(x, Surv(data$time, data$censor), family = "cox", maxit = 1000)
#see which coefficients were kept
active.coefs <- predict(lassofit, type = 'coefficients', s = cv.fit$lambda.min)
```

Variable	Lasso-ed β
(Intercept)	0.00
tx1	-0.58
sex2	0.13
raceth2	-0.07
raceth3	0.00*
raceth4	0.64
raceth5	0.00*
ivdrug2	-0.06
ivdrug3	-0.24
hemophil1	0.04
karnof80	0.00*
karnof90	-0.62
karnof100	-0.98
cd4	-0.01
priorzdv	0.00*
age	0.01

We see that the dummy variable for Hispanic and American Indian, the dummy variable for a Karnofsky score of 80 and priorzdv were shrunk to 0. If we rerun our Cox PH model without `priorzdv`, which from our EDA was found to not be highly correlated with time, and conduct a nested model likelihood ratio test comparing the full model to the model without `priorzdv`, we find that `priorzdv` is not needed in the model based on a result of a $\chi^2_1 = 0.2086$ and a p-value of 0.6479

```
fit2 <- coxph(Surv(time, censor) ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + age, data = da
anova(fit, fit2)
```

```
## Analysis of Deviance Table
##  Cox model: response is  Surv(time, censor)
##  Model 1: ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + priorzdv + age
##  Model 2: ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + age
##    loglik  Chisq Df P(>|Chi|)
## 1 -410.34
## 2 -410.45 0.2086  1    0.6479
```

We do think our model can be more parsimonious; however, so as to avoid overspecification, we look back at the Wald's p-values of the full additive model, and we see that treatment, karnof, cd4, and age (slightly above 0.05) are statistically significant. Before proceeding, we acknowledge that there is a fine line between trying not to overspecify our model and missing potential predictor variables that can contribute some explanatory power. We fit a model with only those 4 variables and conduct a likelihood ratio test between this model and the additive model without `priorzdv`.

```
fit3 <- coxph(Surv(time, censor) ~ tx + karnof + cd4 + age, data = data)
anova(fit2, fit3)
```

```
## Analysis of Deviance Table
##  Cox model: response is  Surv(time, censor)
##  Model 1: ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + age
##  Model 2: ~ tx + karnof + cd4 + age
##    loglik  Chisq Df P(>|Chi|)
## 1 -410.45
## 2 -413.81 6.7234  8    0.5667
```

With a $\chi^2_8 = 6.72$ and a p-value of 0.5667, we conclude that none of the other variables in the additive model were needed.

Because age had a borderline p-value, we tried removing it from the model and seeing whether it was important or not.

```
fit4 <- coxph(Surv(time, censor) ~ tx + karnof + cd4, data = data)
#fit5 <- coxph(Surv(time, censor) ~ karnof + cd4 + tx*priorzdv, data = data)
anova(fit3, fit4)
```

```
## Analysis of Deviance Table
##  Cox model: response is  Surv(time, censor)
##  Model 1: ~ tx + karnof + cd4 + age
##  Model 2: ~ tx + karnof + cd4
##    loglik  Chisq Df P(>|Chi|)
## 1 -413.81
## 2 -414.92 2.2308  1    0.1353
```

```
#fit5
#anova(fit4, fit5)
```

It turns out, with a $\chi^2_1 = 2.23$ and a p-value of 0.1353, that age is not needed in the model.

One nagging thought is that marginal variables might have *some* real predictive value even if it's slight. To that end, let's test whether interactions are significant or not. Specifically, because we have reason to believe that there my be interacting effects with treatment group (the clinical variable of interest), we decided to interact treatment with our categorical covariates along with adjusting for cd4, priorzdv, and age.

```
fit.int <- coxph(Surv(time, censor) ~ tx*sex + tx*raceth + tx*ivdrug + tx*hemophil + tx*karnof + cd4 + p
anova(fit, fit.int)
```

```
## Analysis of Deviance Table
##  Cox model: response is  Surv(time, censor)
##  Model 1: ~ tx + sex + raceth + ivdrug + hemophil + karnof + cd4 + priorzdv + age
##  Model 2: ~ tx * sex + tx * raceth + tx * ivdrug + tx * hemophil + tx * karnof + cd4 + priorzdv + age
##    loglik  Chisq Df P(>|Chi|)
## 1 -410.34
## 2 -406.23 8.2238 11    0.6931
```

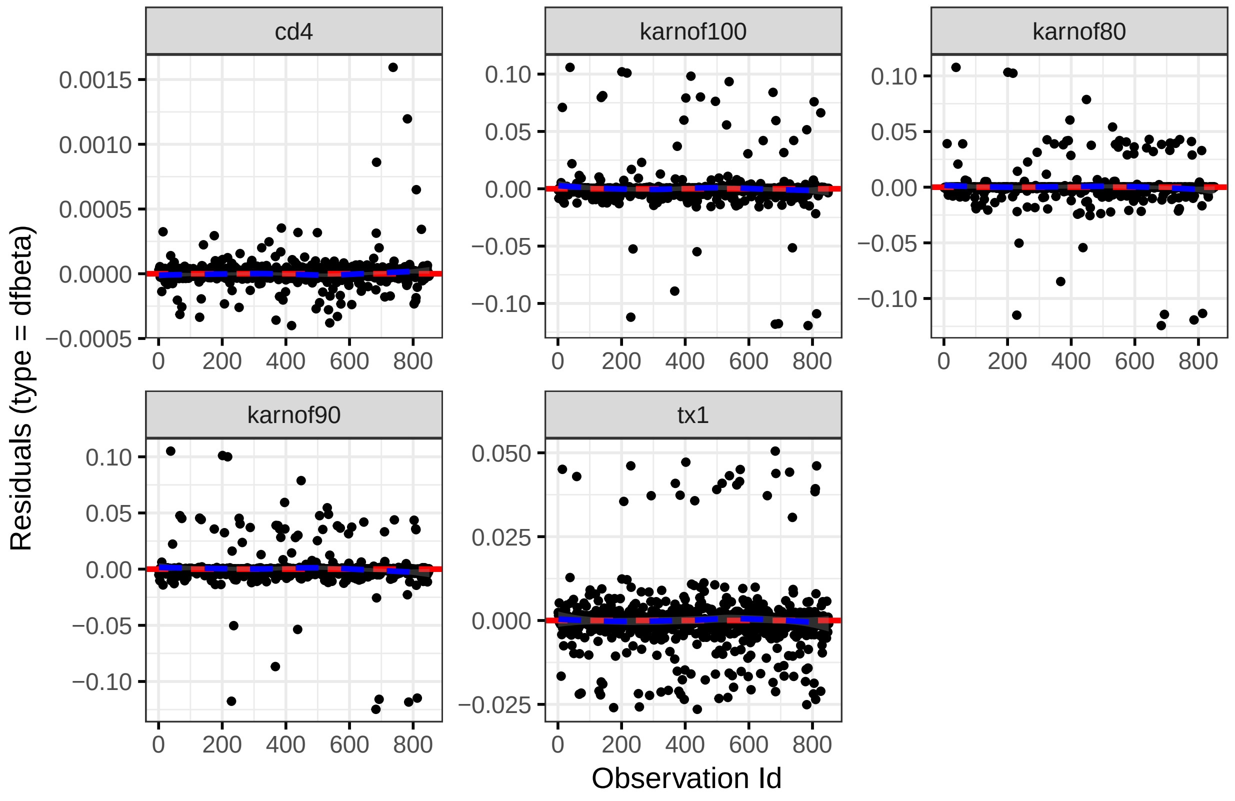
As we might have suspected, none of the interaction terms are needed, so to avoid overfitting, we do not include the interaction terms in our final model.

Influential Observations

In brief, an influential observation is one that is an outlier (unusual time to failure given covariates) and has leverage (an unusual observation in the x-direction). This has the effect of strongly influencing β. To check influence, I'm using dfbeta values which measures the change in β when a purported influential point is removed. Below, we show each feature in a separate plot. On any single feature plot, each point represents an observation and the x-axis simply shows that observation's ID. The residuals on the y-axis show the change in the estimated β coefficient for that feature if that particular observation were removed. For example, in the CD4 plot, if the the observation with the largest residual around 0.0015 were removed, then the estimated β coefficient for CD4 would be 0.0015 higher than it currently is, with that observation included in the model. Belsley et al. (1980) recommend that any observations with the absolute value of `dfbeta` greater than $2/\sqrt{n}$ are to be considered "influential" for that feature. In our case $2/\sqrt{n} = 0.069$.

```
#plot dfbeta plot
ggcoxdiagnostics(fit4, type = "dfbeta",
                 linear.predictions = FALSE, ggtheme = theme_bw(),
                 title = "Figure 6. Plot of Dfbeta values for each explanatory variable")
```

Figure 6. Plot of Dfbeta values for each explanatory variable

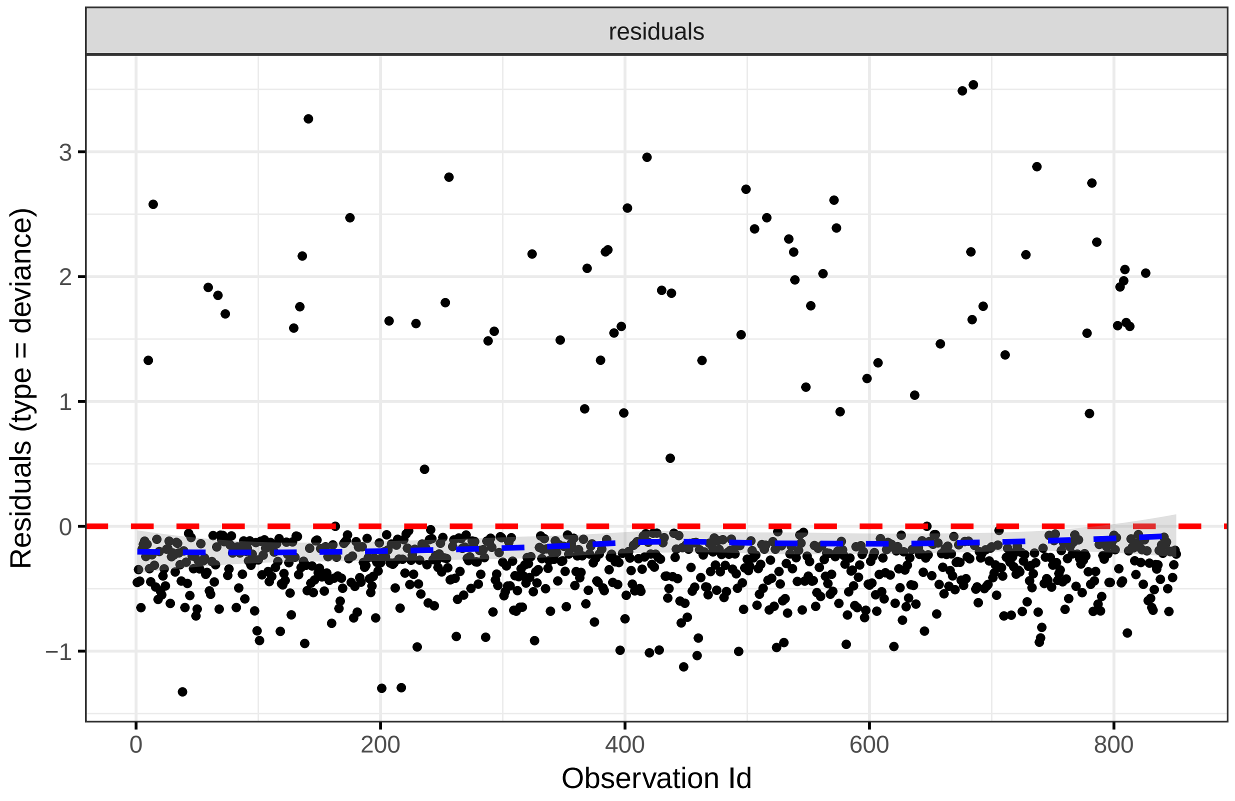


The above index plots show that comparing the magnitudes of the largest dfbeta values to the regression coefficients suggests that none of the observations is super influential, even though some of the dfbeta values for `cd4` and `tx` are large compared with the others. Although there are points with absolute values of `dfbetas` as large as around 0.1, there are no observations which are influential for every single feature, and we consider 0.1 close enough to the cut-off of 0.069 that we would prefer not to remove data. Generally, we should be careful removing influential observations and throwing away data unless there's a *clear* reason we should (e.g. poor data entry), which we didn't find when checking the influential points for `cd4`.

We next look at residuals which measure the difference between the observed number of deaths for the ith individual between time 0 and time t_i and the expected number of deaths based on the fitted CoxPH model. Those residuals are then transformed into deviance residuals which maintain the above interpretation while behaving like OLS residuals (the goal being to observe a random, evenly distributed point cloud). If we look at the deviance residuals for outliers (below), we might initially be concerned because the distribution does not seem symmetric, with many patients having negative residuals that mean they "lived too long".

```r
ggcoxdiagnostics(fit4, type = "deviance",
                 linear.predictions = FALSE, ggtheme = theme_bw(),
                 title = "Figure 7. Plot of Deviance for Final Model")
```

Figure 7. Plot of Deviance for Final Model



But then, we remember that our data was imbalanced between those who lived and died to begin with, because the majority of patients lived until the last time point. Thus, we are not too concerned with influential observations adversely affecting our model.

Checking the Log-Linearity Assumption

In class, we investigated the log-linearity assumption which basically verifies whether a continuous predictor variable is linearly correlated with our log hazard ratio. To do this, we first assume our continuous predictor is categorical, and then we take ratios between the instantaneous relative risks. If the ratios are all roughly the same, then we can assume a linear relationship. In our final model, the only continuous predictor we have is `cd4`, so lets categorize it in the following way:

Level	Desc
0-70	low (0)
71-140	low-medium (1)
141-210	medium (2)
211-280	medium-high (3)
281-350	high (4)

```
#turn cd4 into categorical
data2 <- data %>%
  mutate(cd4_group = ifelse(cd4 <= 70, 0,
                ifelse(cd4 > 71 & cd4 <= 140, 1,
                       ifelse(cd4 > 141 & cd4 <= 210, 2,
                              ifelse(cd4 > 211 & cd4 <= 280, 3,4))))) %>%
  mutate(cd4_group = as.factor(cd4_group))
```

```
#fit our model with the categorical cd4 instead of the continuous one
fit5 <- coxph(Surv(time, censor) ~ tx + karnof + cd4_group, data = data2)
fit5 %>% tidy()
```

```
##         term    estimate    std.error    statistic      p.value  conf.low
## 1        tx1  -0.6800727    0.2572007 -2.644132153 0.0081900673 -1.184177
## 2   karnof80  -0.5807547    0.4120730 -1.409348998 0.1587319970 -1.388403
## 3   karnof90  -1.2995531    0.4090965 -3.176641956 0.0014899085 -2.101368
## 4  karnof100  -1.7502933    0.4599903 -3.805065638 0.0001417666 -2.651858
## 5 cd4_group1  -1.4899999    0.4333253 -3.438525292 0.0005848919 -2.339302
## 6 cd4_group2  -2.8400353    1.0102344 -2.811263621 0.0049347335 -4.820058
## 7 cd4_group3 -17.4248428 3031.6337920 -0.005747674 0.9954140449      -Inf
## 8 cd4_group4  -0.1441065    0.7219043 -0.199619878 0.8417778799 -1.559013
##    conf.high
## 1 -0.1759686
## 2  0.2268936
## 3 -0.4977387
## 4 -0.8487289
## 5 -0.6406980
## 6 -0.8600122
## 7        Inf
## 8  1.2708000
```

Now we do some calculations. We want to see whether $ln(\frac{h_{cd4+100}(t)}{h_{cd4}(t)}) = 70\beta$ turns out to be relative constant. Between group0 and group1 of cd4, the log hazard ratio is $ln(\frac{e^{-1.49}}{e^{0}}) = -1.49$. Between group1 and group2 of cd4, the log hazard ratio is $\frac{e^{-2.84}}{e^{-1.49}} = -1.35$. We can already see that the coefficients for group3 and group4 are whack, which is probably due to a small number of observations within those categories. We conclude that from the similar log hazard ratios calculated above that it's probably safe to say that `cd4` can be modeled as a continuous linear predictor.

Validating our Model by Bootstrapping (Lathan)

##Challenges Personally, my biggest challenge when learning something new is deciding to what degree I'd like to understand the topic. There is a surface understanding of the definition, a more difficult understanding of the mathematics, and an even more difficult understanding of the conceptual applications. In the case of bootstrap, I think I will find challenging understanding the math behind how bootstrap works.

##A brief overview of Bootstrap and its applications to survival analysis Bootstrap relies on sampling with replacement of the sample data and in the case of modelling, it is used to evaluate the performance of the model on the original sample. The estimate of the likely performance of the final model on future data is estimated by the average of all the indices computed on the original sample. If we had an original sample of n elements,X, we resample X m times to get new bootstrap samples $X_i, ...X_m$ each with size n, derive a model in the bootstrap sample, and apply it to the original sample.

Bootstrapping validates the *process* of obtaining our original Cox PH model. It also tends to provide good estimates of the future performance of our final model if the same modeling process was used in our bootstrap samples. According to Efron, who describes an "enhanced" bootstrap, one can estimate the bias due to overfitting, let's call this quantity "optimism". Briefly how it works is (1) calculate C-index for model on original training set (2) Generate bootstrap samples and run model on each sample (3) caclulate difference between each bootstrap C-index and original C-index (4) Average all differences in (3) and call this "optimism" and (5) Subtract "optimism" fom C-index from (1) and one gets the bias-corrected index.

```r
#add data to model fit so bootstrap can re-sample
final.fit <- cph(Surv(time, censor) ~ tx + karnof + cd4, data = data)
g <- update(final.fit, x = TRUE, y = TRUE)
set.seed(47)
#bootstrap validation
validate(g, B = 300)
```

```
##       index.orig training   test optimism index.corrected   n
## Dxy       0.5826   0.5848 0.5689   0.0159          0.5667 300
## R2        0.1295   0.1358 0.1228   0.0129          0.1166 300
## Slope     1.0000   1.0000 0.9452   0.0548          0.9452 300
## D         0.0822   0.0864 0.0777   0.0087          0.0735 300
## U        -0.0022  -0.0022 0.0015  -0.0037          0.0015 300
## Q         0.0844   0.0886 0.0762   0.0124          0.0720 300
## g         1.4159   1.4555 1.3558   0.0997          1.3162 300
```

Training here is defined as the accuracy when evaluated on the bootstrap sample and test is when the model is applied to the original sample. We look at Somers' D statistic to measure the concordance between the ranks of our predicted and observed responses which says two pairs (x_i, y_i) and (x_j, y_j) are concordant if the ranks of both elements agree and discordant otherwise. Then D_{xy} is calculated as the number of pairs that are concordant minus the number of pairs that are discordant over the total number of pairs. Our D_{xy} is 0.5632 which is the difference between the probability of concordance and the probability of discordance of pairs of predicted survival times and pairs of observed survival times. This is essentially a measure of our accuracy of our model on new data.

Validating the SE and the Coefficients via Bootstrap

One idea of validating the standard error and the beta coefficients that we obtained in our original coxph model is to compute our bootstrap standard errors and beta coefficients and view their distributions. In the case of standard errors, if the bootstrap standard errors are close to the original standard errors, then we feel good about the variability of our original estimate. Similarly, if the boostrapped sampling distribution of our betas seems "normal" around our original beta coefficient, then we feel good about our estimate because we are comfortable that our estimated coefficients were close to the bootstrapped average over many new samples.

A little note about the boostrap methodology we used: it was a fairly straightforward sample with replacement without *fixing* the proportion of censored data. In other words, in each bootstrap sample, it could vary from 700 censored events to say, 710 censored events. Future exploratory directions could include trying other bootstrap sampling methods, such as separately bootstrapping censored and event data.

We first compare the standard errors between our bootstrap coefficients and our likelihood coefficients from our coxph model.

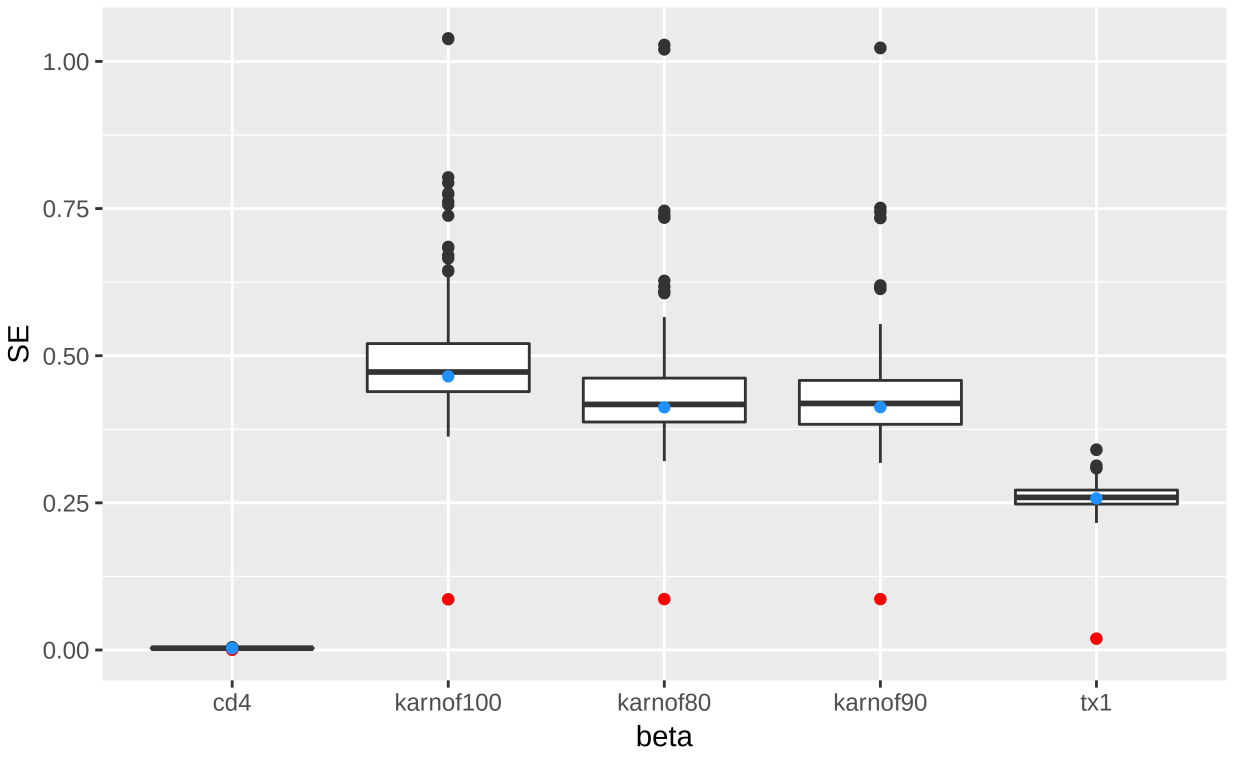
```r
comparison
```

```
## # A tibble: 5 x 3
##    likese bootavgse coef
##     <dbl>     <dbl> <chr>
## 1 0.258     0.261   tx1
## 2 0.412     0.437   karnof80
## 3 0.413     0.436   karnof90
## 4 0.465     0.493   karnof100
## 5 0.00308   0.00313 cd4
```

We see that the standard errors are pretty similar! We also provide a boxplot to visualize how the distribution of bootstrapped standard errors compare to our original standard error estimates. We obtained these boostrapped standard errors by creating 300 boostrapped samples, and for each sample, fitting a model to get β estimates and the standard error of each β estimate, thus generating both a sampling distribution of β estimates and a sampling distribution of β estimate standard errors. In the plot below, we show the sampling distribution of these standard errors. We can see for each explanatory variable that the sampling distribution of standard errors from bootstrapping is centered around the blue dot, representing the standard error from the model fit on the original dataset. The red dots show the standard deviation of the bootstrapped sampling distribution of the standard errors for each β estimate, which are comfortingly low!

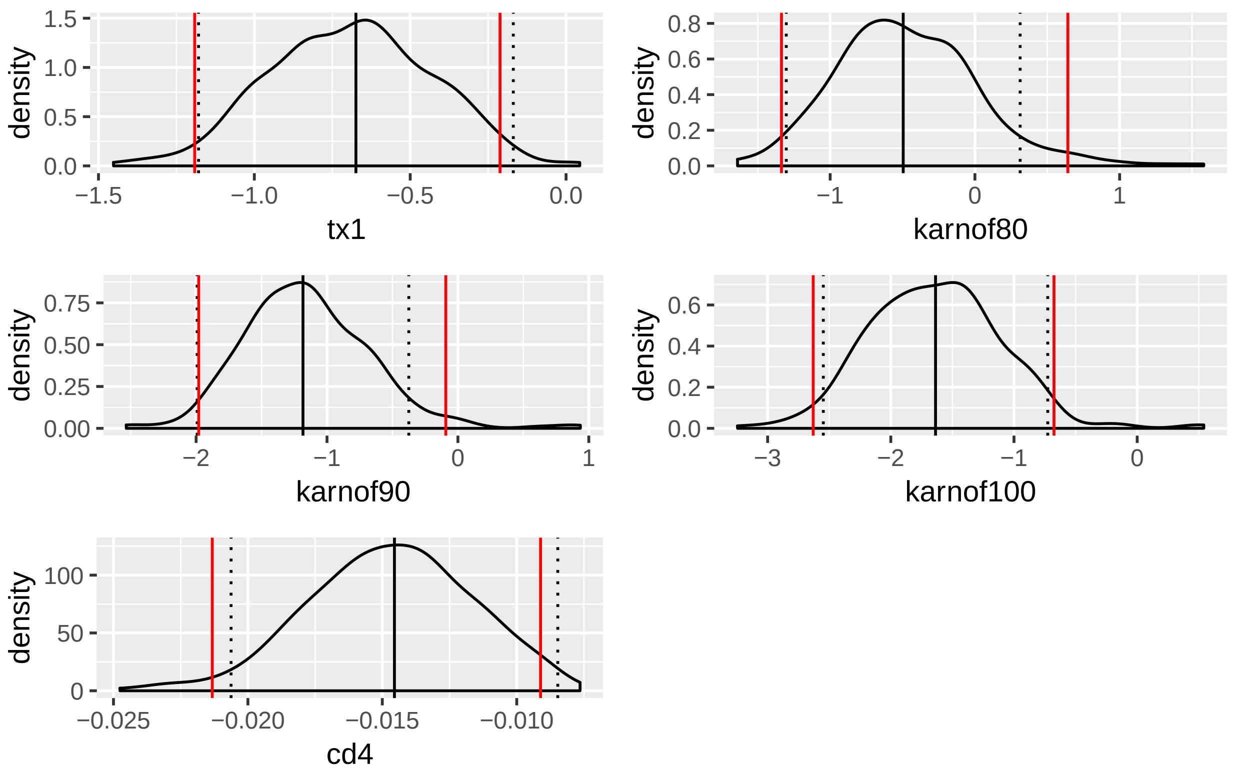
Figure 8. Distribution of Bootstrapped SE

Original SE (blue). Standard deviation of bootstrapped SE (red).



Further, we explore how the bootstrap betas vary compared to our original betas.

Figure 9. Bootstrap Sampling Distribution Beta Coefficients



Solid black line is the original beta. Dotted lines are +/−2*SE of original betas. Red lines are +/−2*SE of bootstrapped betas.

The confidence intervals are fairly similar, with `karnof80` and `karnof90` being slightly right skewed. Thus, we feel comfortable about the original estimates of the coefficients because applying normal theory seems reasonable.

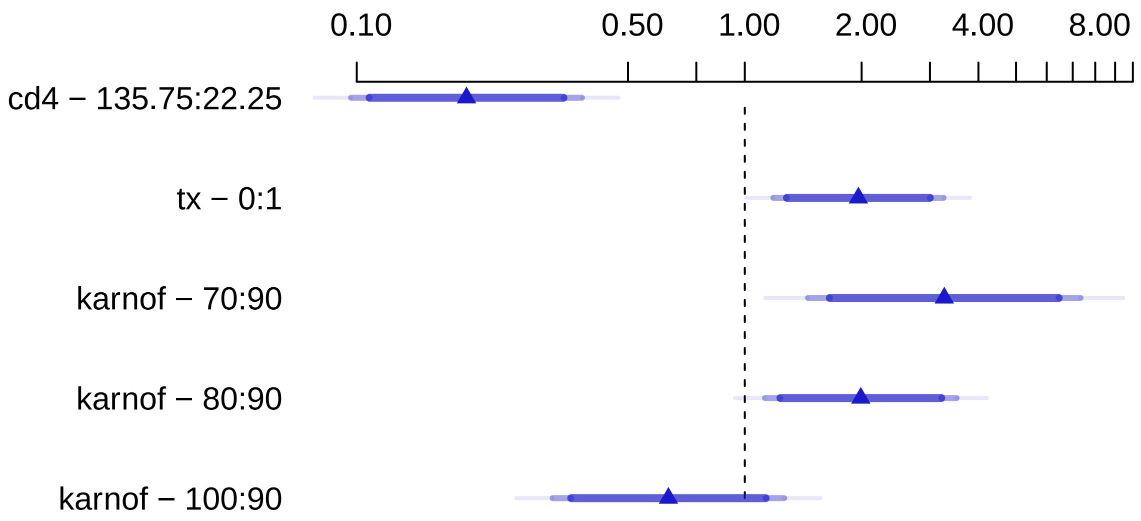
Interpreting our final model

Now that we have validated our final model using Boostrapping, we interpret its output. The final model obtained after selecting features via LASSO was

$$ln(\frac{h_i(t)}{h_0(t)}) = \beta_1 tx1 + \gamma_1 karnof80 + \gamma_2 karnof90 + \gamma_3 karnof100 + \theta_1 cd4$$

. To better understand the effects of each predictor, we provide the following graphics.

Figure 10. The Estimated Hazard Ratios for Each Predictor's Va



This above plot portrays the estimated hazard ratio for different values of each predictor. For the continuous variable `cd4`, it compares the hazard ratio of the 25th percentile of `cd4` values to the the 75th percentile. For the other variables, which are categorical, it compares each level, step by step. Different shaded areas of bar indicate different confidence levels (.9, 0.95, 0.99). We see that increasing `cd4` count from 22.25 to 135.75 increases the hazard (135.75 is treated as the baseline). Or, when the Karnofsky score goes from 70 to 90, we observe a three-fold increase in hazard.

Gradient Boosted Trees (Madison)

In addition to running a traditional CoxPH model, we decided to try running a Cox Proportional Hazards model via a gradient boosted tree.

Background

Gradient boosting machines have gained traction in recent years, popular among Kagglers, researchers, and industry professionals alike. One of the publically availabe algorithms that has fueled this trend is XGBoost (eXtreme Gradient Boosting) developed by Tianqi Chen. XGBoost claims to be a scalable, high-performing, and one of the most computationally efficient implementations of gradient boosting machines out there. It can be used for a variety of regression, classification, and ranking problems.

Gradient boosting is a supervised ensemble method which agglomerates simple, "weak" learners into a more complex whole. In boosting (also called additive training), we start with a constant prediction and iteratively add new functions on top, fixing what we have learned and adding one new model at a time will holding onto functions learned in previous rounds. We fit each model based on the previous prediction, then minimize the loss with the addition of the latest prediction. In doing this, we are actually updating our model each time using gradient descent - hence the name "gradient" boosting! Gradient boosting is possible with almost any simple classifier, but XGBoost in particular uses an ensemble of decision trees. The objective function within XGBoost also incorporates a regularization term.

By default, XGBoost in Python has mean squared error as its loss funciton within its objective function. However, the creators of XGBoost recently added the option to instead use the Cox regression loss function for right-censored survival time data, and that is what we will be using. Predictions are then returned on the hazard ratio scale. The package also includes the negative partial log-likelihood for Cox proportional hazards regression as an evaluation metric.

Cox Proportional Hazards in Generalized Boosted Models

In 2018, Tianqi Chen and other contributors added survival analysis cabability to XGBoost. They use gradient boosting with "the loss function from penalized Cox partial likelihood…, where regularization is explicitly imposed through penalization" ("Boosted nonparametric hazards with time-dependent covariates", Lee et al.). Since the paper associated with XGBoost was published before their survival analysis add-on, we relied on previous publications to give the best explanation of the Cox algorithm underying XGBoost. Their work is summarized below.

But first, to facilitate the understanding of notation downstream, recall the Cox proportional hazards model. For observations $(t_i, \delta_i, x_i), i = 1, \dots, n$ where t_i is the observed time to event for individual i and δ_i is 1 if an event occurred at time t_i and 0 if the observation has been censored, and $x_i = (x_{i1}, \dots, x_{ip})$ is a vector of covariates obtained at time 0. Then the hazard function is $h(t|x_i) = h_0(t)\exp(F(x_i;\beta))$ where $h_0(t)$ is the unspecified baseline hazard and $F(x;\beta)$ is a function of the covariates dependent on a parameter vector β. We like to use a linear predictor of the form $F(x;\beta)$ where each element of the β vector describes the influence of a single covariate. We can obtain an estimate for β by maximizing the partial log-likelihood:

$$l(\beta) = \sum_{i=1}^{n} \delta_i(F(x_i;\beta) - \log(\sum_{j:t_j \geq t_i} \exp(F(x;\beta)))$$

The Cox additions to XGBoost were inspired by the existing Generalized Boosted Models R package `gbm` (Greenwell et al.) which has included the Cox proportional hazard model since 2007. In "Generalized Boosted Models: A guide to the gbm package," Greg Ridgeway details how the developers implementated the Cox proportional hazard model. According to Ridgeway, "The Cox proportional hazard model… is an incredibly useful model and the boosting framework applies quite readily with only slight modification" (Ridgeway 1). Chen et al. detail the mathematics in the context of gradient-boosting trees, summarized below.

The goal of any GBM (gradient boosting machine) is to learn a functional mapping from the data $\{x_i, y_i\}_{i=1}^{n}$ to $y = G(x, \beta)$ where β is the set of parameters of the function G which minimize some cost function $\sum_{i=i}^{n} \Phi(y_i, G(x_i;\beta))$. An important assumption of boosting is that $G(x) = \sum_{m=0}^{M} \rho_m f(x;\tau_m)$ where f is a "weak" learner with a weight ρ and set parameter τ_m (described just below). In other words, this assumption means that $G(x)$ follows an "additive" expansion form, permitting us to perform "additive training" or boosting. Otherwise, there are no functional assumptions made on F, allowing it to be linear, tree-based, or otherwise. Thus, the set of parameters of the function G is exactly $\beta = \{\rho_m, \tau_m\}_{m=1}^{M}$. These are learned in a greedy iterative process to minimize the following cost function, the negative partial log-likelihood:

$$\Phi(y, G) = -\sum_{i=1}^{n} \delta_i(G(x_i) - \log(\sum_{j:t_j \geq t_i} e^{F(x_j)}))$$

.

Inspired by Binder and Schumacher's `CoxBoost` algorithm detailed in "Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models" (2008), the XGBoost developers include a penalization term λ in the cost function Φ. This penalty term λ helps avoid overfitting, as the higher it is, the more cautious the update.

The output of a trained GBM Cox model is the function F. When we "predict" wit a trained GBM model, we get back the $e^{G(x_i)}$ (risk score) for each individual we "predict" on. We can actually compare the hazard ratios between the XGBoost output and the traditional CoxPH model. To demonstrate this, we take the first 10 values output by XGBoost trained on the full dataset (`tx`, `sex`, `raceth`, `ivdrug`, `hemophil`, `karnof`, `cd4`, `priorzdv`, `age`) and compare it to the equivalent cox model's outputs (this was saved as `fit` earlier).

```
xgboost_out <- c( 1.3253996 ,  0.5000791 ,  0.40796974,  0.4254104 ,
                  0.08013923, 0.07688343,  0.03202903,  0.07484766,
                  0.0744646 ,  5.5629425)
coxph_out <- predict(fit, type ="risk")[1:10]
```

```
xgb_ratios <- unlist(Map( {function (x) xgboost_out[1]/x},  xgboost_out))
coxph_ratios <-unlist(Map( {function (x) coxph_out[1]/x},  coxph_out))
compare <- data.frame(coxph_out, xgboost_out, coxph_ratios, xgb_ratios)
compare
```

```
##    coxph_out xgboost_out coxph_ratios xgb_ratios
## 1  4.1795560  1.32539960    1.0000000  1.0000000
## 2  2.5061648  0.50007910    1.6677099  2.6503799
## 3  1.8031462  0.40796974    2.3179240  3.2487694
## 4  2.0812596  0.42541040    2.0081858  3.1155787
## 5  0.2796122  0.08013923   14.9476886 16.5387114
## 6  0.2248243  0.07688343   18.5903232 17.2390800
## 7  0.4268060  0.03202903    9.7926373 41.3811970
## 8  0.6531466  0.07484766    6.3991082 17.7079631
## 9  0.2533244  0.07446460   16.4988274 17.7990562
## 10 7.2425788  5.56294250    0.5770812  0.2382551
```

The last two columns of the above data frame takes the ratio between the first individual's estimated risk and the rest of the individuals considered across the two methods (CoxPH and XGBoost). Although the ratios are not always equal, they are generally similar (within ±1) except for individual number 7 as the ratio in XGBoost is much, much larger.

SHAP Values

Within the past year, SHAP (SHapley Additive exPlanations) has risen in popularity as a way to assess feature importance in tree-based models. We attempt to give a summarised explanation of what these are, but see "A Unified Approach to Interpreting Model Predictions" (2017) and "Consistent Individualized Feature Attribution for Tree Ensembles" (2019) by Lundberg et al. for more detail.

SHAP is actually just an extension of Shapley values from game theory, introduced by Lloyd Shapley in 1953. Imagine that a group of people are playing a cooperative game from which they collectively receive $1000. They wish to fairly divide up this $1000 among themselves based on how much each contributed toward the win. They seek to calculate a value ζ for each player which represents the dollar amount that player will get. They agree that whatever scheme they come up with to distribute the earnings must abide by the following rules, to ensure fairness:

1. The sum of what each player receives should be equal to the total reward.
2. If two people contributed the same amount, they should receive equal proportions of the reward.
3. If any player did not contribute any value, they should receive no part of the reward.
4. If multiple games are played, then an individual's total reward across all games should be equal to the sum of their earnings from each game.

Shapley showed that there is only one method of calculating ζ_i for each player i which respects both rules 1 and 2. Given a set of players P and a reward function r, the amount of money the ith player should receive is:

$$\zeta_i = \sum_{S \subseteq P \setminus \{i\}} \frac{|S|!(|P|-|S|-1)!}{|P|!}(r_{S\cup\{i\}} - r_S)$$

Note that $r_{S\cup\{i\}}$ is the collective reward if the ith player is included while r_S is the reward they will get without the help of player i. Thus, $r_{S\cup\{i\}} - r_S$ represents the contribution of player i. The part, $\sum_{S\subseteq P\setminus\{i\}} \frac{|S|!(|P|-|S|-1)!}{|P|!}$, sums over all possible subsets of players $S \subseteq P \setminus \{i\}$. This must be done because the effect of including a player may depend on which other players are in the game.

As Lundberg et al. note, the player-reward framework translates naturally to a feature-prediction one. The set of players P becomes our set of features. The reward function, which outputs the total collectvie reward, r becomes the machine learning model $r(x)$ for which we wish to derive feature importances, parameterized by a vector of values of explanatory variables (imagine a single row in a tidy dataframe). The collective total reward $r(x)$ becomes the predicted value for a single observation x. To calculate ζ_i in this new context, for each subset of features $S \subseteq P \setminus \{i\}$, we train one model $r_{S\cup\{i\}}(x_{S\cup\{i\}})$ with feature i and another model $r_S(x_S)$ without feature i, where x_S represents the values of the input features from the set S. The predictions of these two models are compared by computing $r_{S\cup\{i\}} - r_S$ and we sum the result over all possible subsets $S \subseteq P \setminus \{i\}$ to yield the importance of feature i for a single observation x:

$$SHAP_i = \sum_{S\subseteq P\setminus\{i\}} \frac{|S|!(|P| - |S| - 1)!}{|P|!}(r_{S\cup\{i\}}(x_{S\cup\{i\}}) - r_S(x_S))$$

The SHAP values uphold the original four rules established in the player-reward example. In particular note that because of rule 1, the sum of the SHAP values across all features is equal to the prediction p given by the full model. That is, $\sum_{i=1}^{|F|} \zeta_i = r(x)$. Essentially, SHAP values decompose the prediction for each individual into a sum of components which correspond to the importance of each feature in the model. We have thus taken a complex, non-linear model and broken it down into individual-specific linear models (linear combinations of features based on their importance).

XGBoost

May 2, 2019

1 Gradient Boosted Trees

```
In [9]: import shap
        import xgboost
        from sklearn.model_selection import train_test_split
        import matplotlib.pylab as pl
        import pandas as pd
        import numpy
```

1.1 Loading Data

As with all prior modeling, we drop `id` because it's just the identification number of each patient We exclude `time_d` and `censor_d` because these seem unfair to include in modeling since they contain information about time and censor We chose not to include `txgrp` and `strat2` because they were derived (and thus highly correlated with) from other predictor variables.

```
In [10]: data = pd.read_csv("AIDSdata.csv")
         data = data.drop(['id', 'time_d', 'censor_d', 'txgrp', 'strat2', 'time_d'], axis=1)
```

We next separate predictors and labels. Since XGBoost only allow one column for y, the censoring information is coded as negative values.

```
In [11]: # predictors
         X = data[['tx', 'sex', 'raceth', 'ivdrug',
                   'hemophil', 'karnof', 'cd4', 'priorzdv', 'age']]

         # labels
         censorL = [c for c in data.censor]
         timeL = [t for t in data.time]
         time_censor = list(zip(censorL, timeL))
         y = [x[1] if x[0] else -x[1] for x in time_censor]
```

We create a train and test split, then train the models. However, for a more ad-hoc approach, we train 80% of the data on a variety of values for `eta` (learning rate or shrinkage parameter), `max_depth` (maximum depth of each tree), `subsample` (fraction of observations to be random samples for each tree), and `lambda` (penalty parameter; the higher it is, the more regularization). From these trials, we chose the values which gave the highest Harrell C-statistic on the remaining held-out test set. The Harrell C-statistic measures how well we can order patients by their survival time, where 1 is a perfect ordering.

```
In [12]: # create a train/test split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

```
In [13]: xgb_full = xgboost.DMatrix(X, label=y)
         xgb_train = xgboost.DMatrix(X_train, label=y_train)
         xgb_test = xgboost.DMatrix(X_test, label=y_test)
```

```
In [15]: # just by messing
         params = {
             "eta": 0.002,
             "max_depth": 2,
             "objective": "survival:cox",
             "subsample": 0.5,
             "lambda": 1

         }
         model_train = xgboost.train(params, xgb_train, 10000,
                                     evals = [(xgb_test, "test")], verbose_eval=0)
```

```
In [16]: def c_statistic_harrell(pred, labels):
             total = 0
             matches = 0
             for i in range(len(labels)):
                 for j in range(len(labels)):
                     if labels[j] > 0 and abs(labels[i]) > labels[j]:
                         total += 1
                         if pred[j] > pred[i]:
                             matches += 1
             return matches/total
```

```
In [17]: # see how well we can order people by survival
         c_statistic_harrell(model_train.predict(xgb_test, ntree_limit=5000), y_test)

Out[17]: 0.8472803347280334
```

The highest C-statistic we have succeeded in obtaining is 0.847. Using these best parameter values, we train a model on the full dataset. This is the model we use from here on out.

```
In [18]: # train final model on the full data set
         params = {
             "eta": 0.002,
             "max_depth": 2,
             "objective": "survival:cox",
             "subsample": 0.5,
             "lambda": 1
         }
         model = xgboost.train(params, xgb_full, 10000, evals = [(xgb_full, "test")], verbose_ev
```

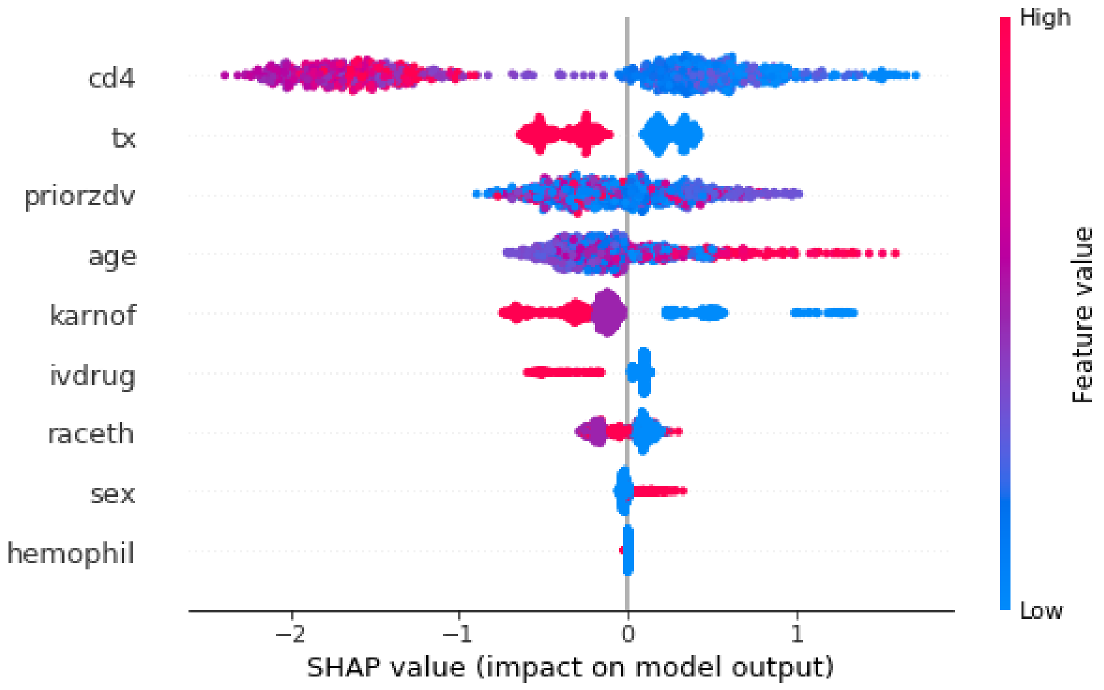
1.2 Inference with SHAP values

To review the the more detailed explanation above, SHAP values are a way to perform inference on tree-based models. Their key advantage is that rather than simply giving an unsigned feature importance value, as was the case traditionally with tree-based models, SHAP values are signed. Thus, we can make inferences such as "higher values of x are associated with lower values of y."

1.2.1 Feature Importance

```
In [19]: shap_values = shap.TreeExplainer(model).shap_values(X)
```

```
In [20]: shap.summary_plot(shap_values, X)
```



In the above SHAP values plot, which replaces the familiar bar chart of feature importance for tree-based models, the predictors are listed from top to bottom in their order of importance in the model. Each dot represents one patient, and the colors represent the range of the effects of each feature within the dataset. These colors enable us to relate how changes in the values of each predictor are associated with the change in hazard. The SHAP values are plotted on the x-axis. Higher SHAP values represent higher risk of death or AIDS diagnosis associated with each explanatory variable.

We can observe that the primary risk factor for death or AIDS diagnosis is Baseline CD4 count (`cd4`) as it is the first predictor listed in the SHAP values plot. The lower the CD4 (bluer dots), the higher the risk of dying or being diagnosed with AIDS.

The next most important risk factor is the IDV treatment, `tx`. Patients who did recieve the IDV treatment (red) have a lower risk of being diagnosed with AIDS or dying as compared to those who did not recieve the IDV treatment.

Another interesting important risk factor is age. Although among younger to middle-aged people the risk of death or diagnosis is around the same, older patients are definitely more likely to die or be diagnosed with age.

A final informative risk factor is Karnofsky Performance Scale (karnof). As expected, the higher the Karnofsky score of a patient, the lower their risk of death or diagnosis.

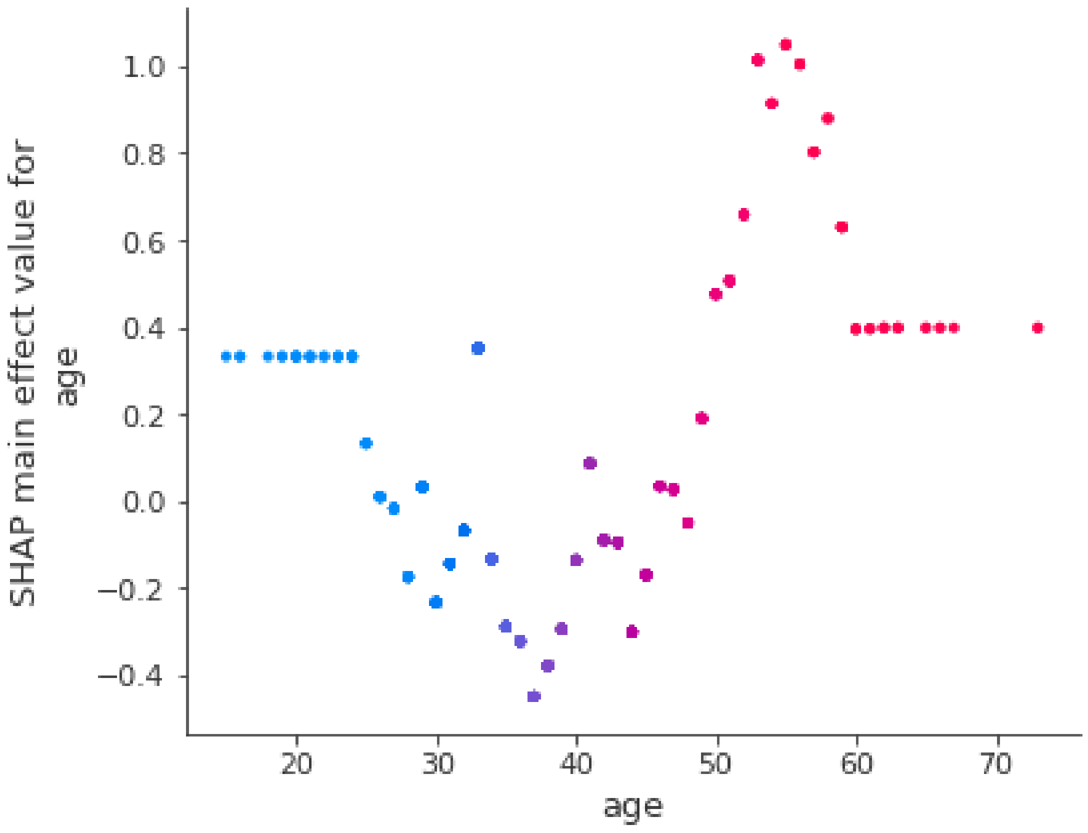
1.2.2 Interactions

We first calculate all SHAP interaction values which may be interpreted as the difference between SHAP values for feature i when feature j is present and the SHAP values for feature i when feature j is absent. Note that SHAP interaction values are commutative because we take the sum of the interaction effects to get the total interaction effect (interaction of i and j written $\iota_{i,j}$ is the same as the interaction of j and i written $\iota_{j,i}$ and the total interaction effect is $\iota_{i,j} + \iota_{j,i}$). The main effect is then defined as the difference between the SHAP values and the SHAP interaction values for a given feature.

```
In [21]: shap_interaction_values = shap.TreeExplainer(model).shap_interaction_values(X)
```

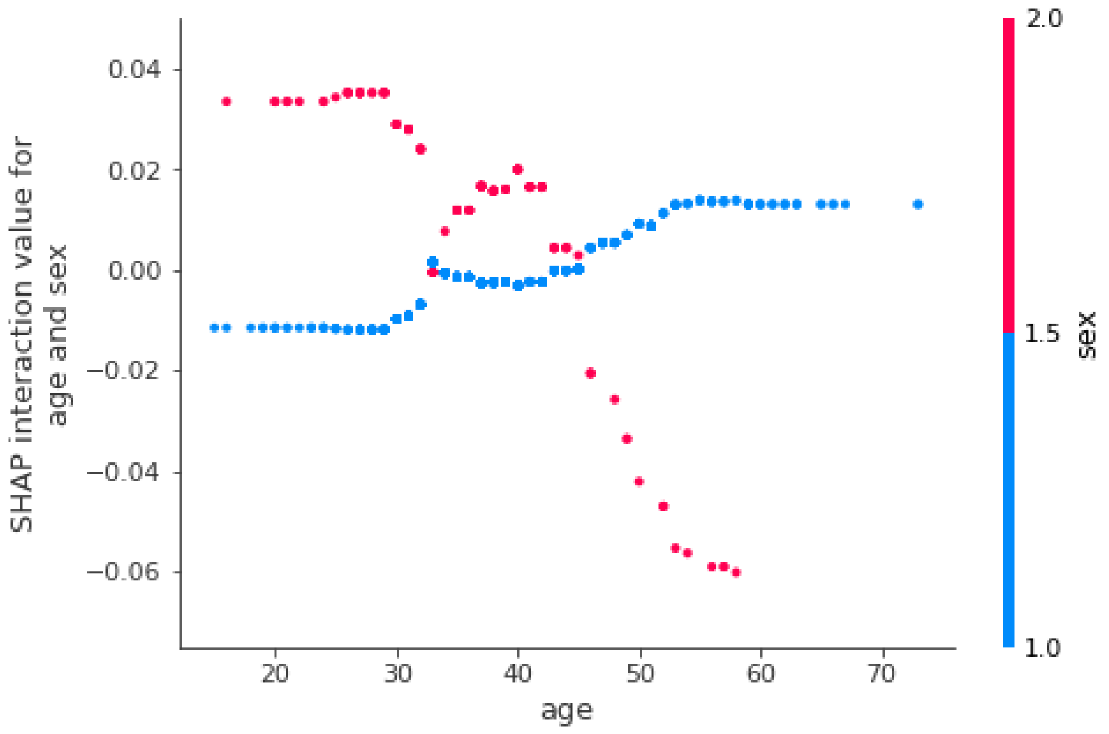
We produce SHAP interaction values which separate the impact of two predictors into main effects and interaction effects. Doing this, we reveal an interesting interaction effect with age and sex.

```
In [22]: shap.dependence_plot(
             ("age", "age"),
             shap_interaction_values, X,
             display_features=X
         )
```



In the plot above, we examine the main effect of age on risk of death or diagnosis. Each dot is a single individual. On the x-axis age is plotted and on the y-axis, the SHAP main effect value for age displayed. Higher SHAP values represent higher risk of death or AIDS diagnosis associated with the the interaction effect of age and sex. In general, older people are at greater risk of death or diagnosis (as alluded to in the preceeding feature importance).

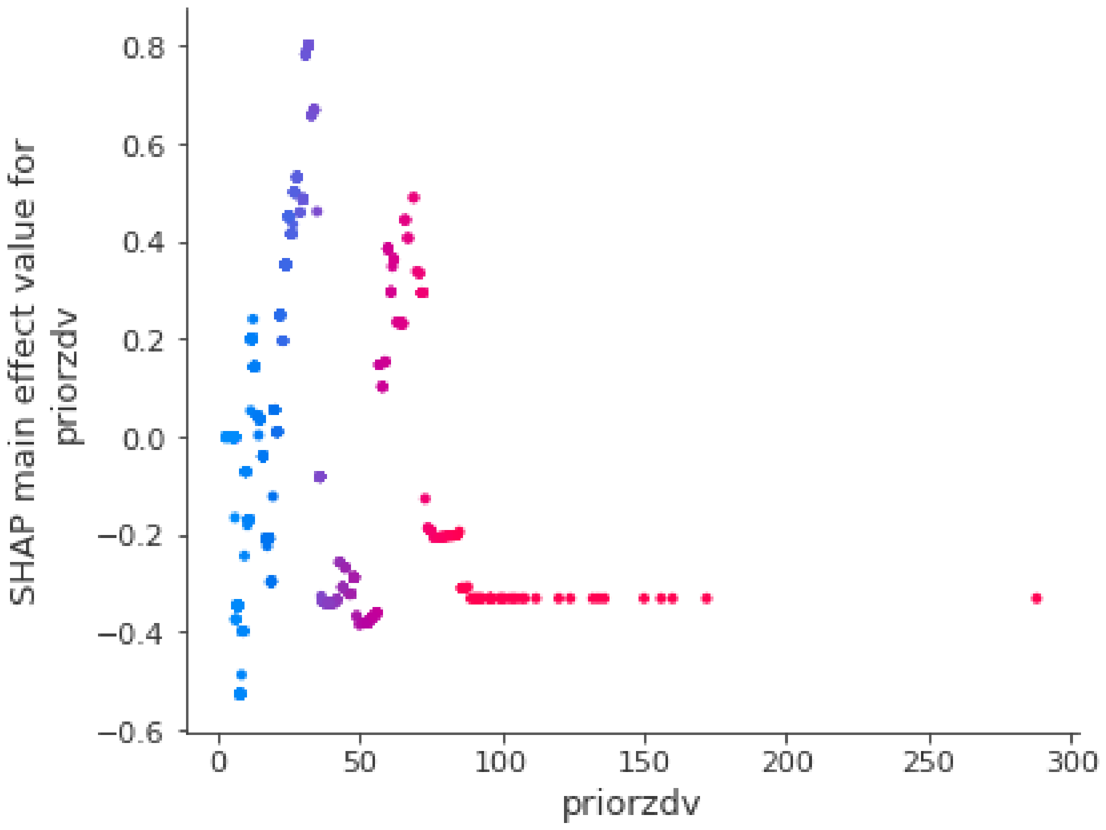
```
In [23]: shap.dependence_plot(
             ("age", "sex"),
             shap_interaction_values, X,
             display_features=X
         )
```



However, looking at the interaction effect of age and sex, we notice an interesting result. In the above plot, each dot is a single individual. On the x-axis, their age is plotted and on the y-axis, the SHAP interaction value for age and sex is displayed. Higher SHAP values represent higher risk of death or AIDS diagnosis associated with the the interaction effect of age and sex. Coloring each dot by age reveals that females are at greater risk of death or diagnosis at younger ages than males, but at older ages, males are more at risk.

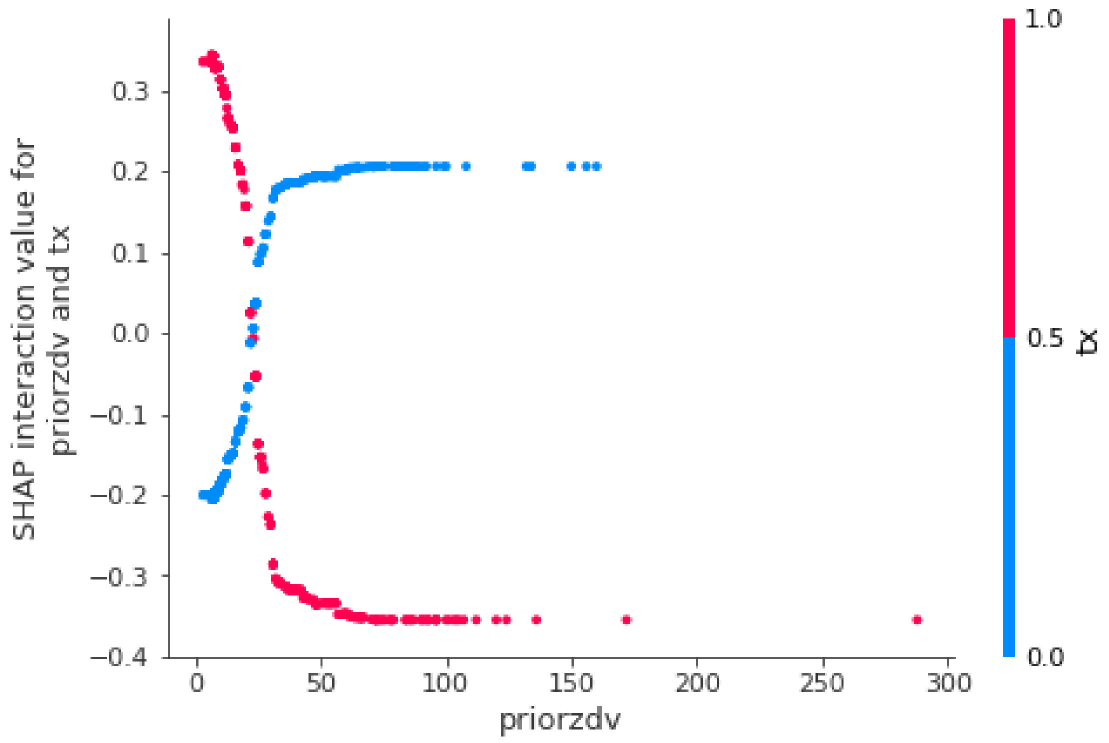
We also wanted to investigate why, although months of prior ZDV ues (`priorzdv`) is supposedly the third-most important feature in the XGBoost model, the feauture importance plot (very first plot) appears to show no real association between `priorzdv` and risk (low values of `priorzdv`, shown in blue, are sometimes associated with higher risk and sometimes associated with lower risk!). By looking at the main effect for `priorzdv` below, we observe this phenomenon again.

```
In [24]: shap.dependence_plot(
             ("priorzdv", "priorzdv"),
             shap_interaction_values, X,
             display_features=X
         )
```



From the above plot, we observe that sometimes low values of `priorzdv` are associated with low SHAP values (lower risk of AIDS diagnosis or death) while for other individuals, low values of `priorzdv` are associated with high SHAP values (high risk of AIDS diagnosis or death).

```
In [25]: shap.dependence_plot(
             ("priorzdv", "tx"),
             shap_interaction_values, X,
             display_features=X
         )
```



However, when observing the interaction effects of number of months of prior ZDV (`priorzdv`) and whether each patient got an IDV treatment (`tx`), we see a clear difference between the treatment groups. For patients who received the IDV treament, more months of prior ZDV use is associated with lower risk of diagnosis or death. However, for patients who did not receive the IDV treatment, more months of prior ZDV use is associated with higher risk of diagnosis or death.

2 Discussion and Concluding Remarks

Overall, our model points to IDV treatment, Karnofsky score, and cd4 levels as important predictors for predicting time to death (or diagnosis). As seen in figure 10, an increase in cd4 is associated with a statistically significant decrease in the log hazard ratio, which is sensible since cd4 cell count is a well-regarded indicator for the presence of HIV – the lower the cd4 count, the higher the activity of the HIV virus killing off these cells. Furthermore, when the Karnofsky score goes from 70 to 90, we observe a signficant three-fold increase in hazard. This result may even serve to help validate the Karnofsky performance metric intrinsically. In addition, Figure 5, our model fit, and Figures 10 both show that there is a significant difference between the control and treatment groups regarding survival, with the treatment group having a higher survival. The XGBoost analysis reinforces the importance of cd4 count, treatment, and Karnofsky score as predictors while also revealing an interesting interaction between prior ZDV use and treatment.

Thus, given that this was an experimental study, we conclude that the treatment containing IDV significantly increases survival after adjusting for cd4 levels and Karnofsky score. We are willing to generalize this claim to patients who have no more than 200 CD4 cells/cubic millimeters, at least 3 months of prior ZDV therapy, and who reside in the United States since those were the inclusion criteria for this study.

3 References

1. https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
2. https://xgboost.readthedocs.io/en/latest/tutorials/model.html
3. https://slundberg.github.io/shap/notebooks/NHANES%20I%20Survival%20Model.html
4. https://statisticalhorizons.com/multicollinearity (Great article on multicollinearity)
5. Harrell, F. (2015) Regression Modeling Strategies. (great textbook on all things survival analysis)
6. https://www.datacamp.com/community/tutorials/bootstrap-r (Overview of bootstrapping)
7. https://stats.stackexchange.com/questions/22017/sample-size-and-cross-validation-methods-for-cox-regression-predictive-models
8. https://stats.stackexchange.com/questions/18084/collinearity-between-categorical-variables (Explanation of VIF for Categorical Variables)
9. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
10. https://stackoverflow.com/questions/53562813/xgboost-cox-survival-time-entry
11. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
12. https://arxiv.org/pdf/1802.03888.pdf
13. http://www.saedsayad.com/docs/gbm2.pdf
14. https://arxiv.org/pdf/1701.07926.pdf
15. https://www.hindawi.com/journals/cmmm/2013/873595/
16. https://medium.com/@gabrieltseng/interpreting-complex-models-with-shap-values-1c187db6ec83
17. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf