Customer Churn Analysis: Genetic Testing Service Attrition Using Proportional Hazard Models in R

Faris Sbahi

July 8, 2016

Abstract

Here, we seek to assess the effectiveness of a unique statistical discipline, survivor analysis coupled with the paradigm of Random Forests in its application to customer attrition. We'll develop a proprietary algorithm to provide a sensitivity-optimized risk score to accounts. Customer attrition analysis is often used as a key business metric due to the generally lower cost of retaining a customer as opposed to acquiring a new one. Historically, hazard models have been applied to biological systems to analyze the expected duration of time to death. Here, we redefine this time to event to be time to account loss. Conventional tools in retention analysis, such as the logistic regression or even a Cox PH model, cannot compensate for censoring, provide both whether and when an account will leave, and account for time-varying covariates while assessing variable importance and accounting for non-linearities. Overall, we exposed a revealing picture and were able to make sound predictions given certain parameters.

1 Theoretical Introduction

Survivor analysis is a powerful discipline in statistics that allows for a deeper understanding of data plagued by censoring and the dilemma of time-varying covariates. Particularly, in the context of sales, we notice the issue of right-censoring. Furthermore, we may be interested in the question of *when* an account is likely to be lost.

In the following section, we will introduce the original problem of attrition, a common response variable that data scientists in sales are interested in modeling. Furthermore, we will compare potential methods that can be used to analyze the question at hand, and ultimately why we chose the Random Survival Forest model over a more classical Cox PH model.

1.1 Customer Churn

Customer churn or attrition has important implications for both the growth and profit of a company. For GTS and other similarly structured companies, retention of customers is of utmost importance, as recruitment of new customers is more difficult than holding on to existing customers. Hence, a nominal price of time and money exists that businesses are willing to pay to understand why and when a customer may leave. This has been explored in the context of insurance attrition with the use of a classical proportional hazards model.

With this knowledge of risk and reason, the problem can be targeted and ameliorated effectively. However, these insights have been the traditional target of simple classifiers like the logistic regression. What we seek to gain is deeper information into when an account is likely to leave and a sound treatment of time-varying covariates.

In GTS, physicians prescribe tests, for which we process and deliver results. In this competitive market where several competitors are attempting to take customers from one another, it is imperative that physicians continue to prescribe our tests. While customer loyalty is relatively low in our business, there are significant barriers to switching from one competitor to another. Physicians would have to change their workflow in the office, educate their staff and prepare for another competitors process flow; therefore, this switch is gradual and can be detected by a slight tapering in the ordering patterns. This tapering will prove to be an important indicator in our model and identify customer churn.

This has huge implications not just in the genetic testing industry but also in pharmaceutical industry which is also dependent on the number of drugs a physician prescribes for patients.

1.2 Logistic Regression

When dealing with the issue of attrition, we are concerned with understanding the probability that an account will be lost. Hence, a logistic regression may seem like a plausible choice. We may choose the logistic model under the assumption that there are two discrete outcomes: account lost and account retained.

This is problematic for a number of reasons. One may immediately observe, we make the bold assumption that there only exists two independent account characteristics: healthy and unhealthy. Therefore, no intermediate accounts can be considered in our analysis. Hence, we reduce our dataset to what we deem as two independent subsets.

Even more problematic, is the simple fact that our subset of "healthy" accounts likely include or are exclusively accounts that remain open at the time of collection. Therefore, we stumble upon the issue of right-

censoring, in which we only have the information that an account has lasted up until a point in time, t. Hence, some of these accounts may leave within hours while others stay open for years. Clearly, we have potentially discretized our dataset inappropriately in a situation as such.

The logistic regression is ubiquitous in nearly every field of research nowadays. It's mathematical form is trivial but is useful for estimating probability of binary responses (and can even be extended with multinomial logits).

Regardless, we'll build mathematical intuition of the logistic function, odds ratios, and logit to ease our transition to more complex models.

The logistic function $\sigma(t)$ is defined as

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{1}$$

Where we treat t as a linear function of the explanatory variable x. Hence, it takes the form

$$t = \beta_0 + \beta_1 x \tag{2}$$

Here, we can make a new definition by taking the inverse of the logistic function. This is called the logit, or log odds.

$$\beta_0 + \beta x = ln(\frac{\sigma(t)}{1 - \sigma(t)}) \tag{3}$$

Now, we can easily define the odds of the dependent variable equaling a case, given a set of predictors. Odds are the equivalent to the exponential function of the linear regression expression. It is determined by the ratio of the probability of success and the probability of failure. Note, how the logit is an intuitive linking function. This is due to its mathematical form: $ln\frac{p}{1-p}$. Hence, we can find odds by exponentiating both sides.

We thereby define odds as

$$odds = e^{\beta_0 + \beta_1 x} \tag{4}$$

Then, we can easily define a continuous metric: the odds ratio

$$\frac{P(Y|x_i + 1)/(1 - P(Y|x_i + 1))}{P(Y|x_i)/(1 - P(Y|x_i))} = e^{\beta_i} \tag{5}$$

In other words, the effect on the odds after a one unit increase in x with all other variables held constant. If there are multiple explanatory variables, the odds ratio can be calculated for each x_i. In effect, the

result is an understanding of each variable's effect on the odds of success vs. failure, which is easily calculated as the *exp* of the variable's coefficient.

1.3 Survivor Analysis and Cox Proportional Hazards

Interestingly enough, the logistic regression was first developed by statistician David Cox in 1958, the namesake of the Cox PH model. Survival models provide the association between the time that passes before an event occurs and one or more covariates. Traditionally, this event is death and hence this model has bene used extensively in the medical field. However, in proportional hazard models the unique effect of a unit increase in a covariate is multiplicative on the hazard rate. This is particularly problematic in scenarios where the covariates may be acting nonlinearly.

Evidently, in the field of GTS it should be expected that variables like volume and age will not act proportionally on the hazard rate. Mathematically, however, the model description is fairly straightforward. For this reason, its use has remained extensive so long as the proportional hazard assumption isn't too poor of an assumption to make.

However, in our unique situation we found that we needed a more precise model that could expose more about our datasets while providing considerably more accurate predictions. Hence, the Cox PH model did not suit our needs for its downfalls.

As such, we still ultimately incorporate survival analysis into our mod

1.4 Decision Trees and Random Forests

Decision tree learning has the goal of predicting the value of a target given a number of inputs. It's agile nature has led it to become a common tool in data mining.

As seen in figure 1, each node represents an input variable. Each outcome is referred to as a "leaf". Note that each leaf represents a value of the target variable given the value of the input variables.

A decision tree is a great utility for its ability to represent classifications intuitively with a flow-chart like structure. However, the method of "learning" a tree is a touch more elusive, at least at first glance.

There exists a few notable methods, however of interest is an algorithm known as "CART", or Classification and Regression Trees–a common tool of predictive analytics. The algorithm is a non-parametric decision learning tree technique that is capable of producing either classification or regression trees, dependent on the variables involved.

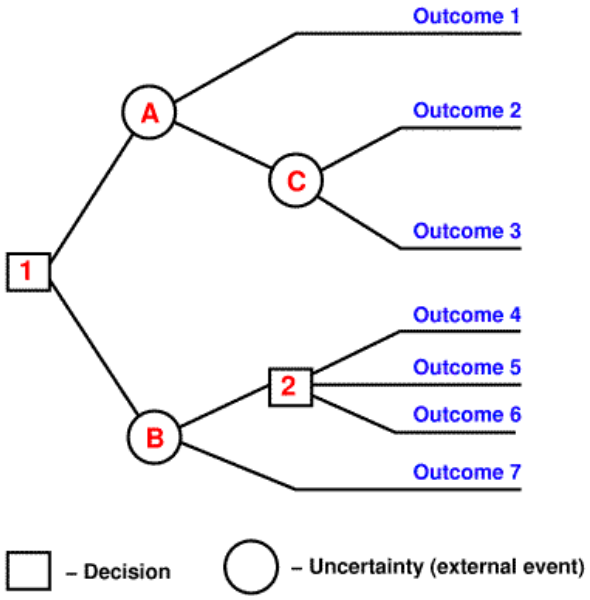

Figure 1: A visual of decision tree learning

As you may be familiar, the term non-parametric implies that no assumptions are made about the underlying probability distributions. In essence, the parameters are determined by the training data, not the model–it's *non-parametric*.

The CART algorithm involves (1) creating rules based on variables values to get the best split to differentiate observations based on the dependent variable (2) applying the same process to the child node recursively (3) ceasing to split after the algorithm detects that no further gain can be made.

Now, in the paradigm of random decision forests again, a multitude of these decision trees are "grown" at training time. The output is the class that is the mode of the classification, the mean prediction. In effect, random decision forests correct for decision trees' habit of overfitting the training set.

The original formulation of random decision forests was created using the random subspace method. In essence, the random subspace algorithm randomly draws a subset of attributes from the original attribute space in order to determine the splitting attribute for each node splitting event of a decision tree.

However, it was when Breiman extended this algorithm with the idea of bagging that random forests were popularized. Bagging, an abbreviation of bootstrap aggregating, generates new training data

iteratively by repeatedly selecting a random sample with replacement for the training set and fitting decision trees to these samples.

In effect, this leads to better model performance and decreased variance without increasing bias.

1.5 Random Survival Forests

With the theoretical basis of Survival analysis and random forests in mind, we move to the model of interest. Random Survival Forests are a natural extension

2 Analysis

We begin with a dataset that includes 56 metrics over 746,659 cases. Each case corresponds to a distinct order by an account. Dealing with such a large dataset and aiming to do the aforementioned analyses, we choose to use the Microsoft Open R toolkit, an enhanced R distribution. The R package "survival" includes functions to reduce the complexity of our survival analysis.

A note on memory management:
R manages your data by storing everything in RAM. Therefore, if you receive warning messages on memory allocation it is a good idea to delete objects no longer in use and run garbage collection **gc()**.

2.1 Pre-processing

First, we aim to simplify our data in a fashion to perform as simple of a regression as possible. Hence, from the dataset we extract looks like:

Table 1: Initial Variables

Variable	Description
Kit_ID	unique test order ID
clinic_id	unique account ID
Product_Group	type of test ordered
Region_Group	type of order (Direct/IVF/Int'l)
Received_Week	week test was received

In R, we perform this operation with

```
> vars <- c("Kit_ID", "clinic_id", "Product_Group", "Region_Group",
"Received_Week")
> dfvars <- df[vars]
```

where df is our original, imported dataset.

We are interested in the tests NPT and HCS, so we remove those that don't match from the dataset. Furthermore, we choose to focus on direct sales so we act accordingly.

We will use the package "dplyr" often for our filtering of data.

```
> install.packages("dplyr")
> library(dplyr)
> df = filter(dfvars, Region_Group == "Direct")
> df = filter(df, Product_Group == "NPT" | Product_Group = "HCS")
```

By using the *summary* function included in R we may visualize a series of metrics about our new dataset.

```
> summary(df)
```

First, it's important to ensure that R has read in our csv data in the correct format.

```
> str(df)
```

Our analysis required the following commands, as clinic_id and Received_Week had been stored as categorical variables (known as factors in R).

```
> df$Received_Week = as.Date(df$Received_Week, format = "%m/%d/%y")
> df$clinic_id = as.integer(as.character(df$clinic_id))
```

Now, running the structure command again, we see that R has read in our kits, case, and clinic IDs as integers, product and region groups as categorical variables, and dates as dates.

We have no interest in "dabblers" of the market as they may hinder our model. Hence, we remove clients that have ordered less than 5 tests per month.

```
> by_clinid <- group_by(df, clinic_id)
> df <- mutate(by_clinid, total_days = max(Received_Week)
-min(Received_Week))
> by_clinid <- group_by(df, clinic_id)
> df <- mutate(by_clinid, total_volume = n())
> by_clinid <- group_by(df, clinic_id)
> df <- mutate(by_clinid, volmo = total_volume/((as.integer
(total_days))/30+0.25))
> df <- filter(df, volmo > 5)
```

Now, we want to extract an important covariate: volume at each time point. At the time being we have a dataset that consists of a unique row for each test order, the week an order was placed, the clinic ID, the total days an account was open, and volume/month.

2.2 Data from the Viewpoint of Cox

Important to understand is that survival models attempt to extract the underlying hazard function in a given system. Of course, this makes the assumption that such a function *exists* or can be approximated. This is a reasonable assumption in our situation, as it can be expected that each account is subject to a similar pattern of increasing/decreasing hazards over time. However, you may object that there are variables that are more important to the likelihood of attrition than time. Perhaps, you may even find that the hazard sees some sort of multiplicative or **proportional** increase or decrease depending on these covariates. If you've managed to reason such, you've identified the underlying basis of the Cox **Proportional** Hazards model.

Of course, our assumption that the covariates are multiplicative to the hazard is core to the model. Hence, we will test this using a one-line R command.

2.3 Covariate Extraction

With an understanding of the Cox model we reason that alongside our approximated hazard function, volume should be an important covariate that will affect the likelihood of attrition.

Note that from Table 2, if we were to group clinic IDs by time point, the number of rows would correspond to the orders received at that time point. Hence, we will calculate the metric exactly as such

Table 2: Metrics

Variable	Description	Filtration
Product_Group	type of test ordered	NPT and HCS
Region_Group	region of order (Direct/IVF/Int'l)	Direct
Received_Week	week test was received	
clinic_id	corresponding to a unique account	
total_days	total days an account is open	
total_volume	total tests ordered over entire period	
volmo	volume/month	> 5

```
> by_clingroup <- group_by(df, clinic_id, Received_Week)
> df <- mutate(by_clingroup, volume=n())
```

We need age, to serve as a covariate and so we can extract long an account has "survived" in time in model building.

```
> by_clinid <- group_by(df, clinic_id)
> df <- mutate(by_clinid, age = Received_Week-min(Received_Week))
```

Finally, we need the status of an account. We will use the definition that if an account hasn't placed an order in our last week of data collection, then they were lost at their maximum time point. However, we will only test this assumption on dates of at least one month prior to our date of collection. This inherently filtration ensures that a lost account hasn't returned within four weeks (ponder and convince yourself of this). Otherwise, they remain open or were right-censored.

```
> by_clinid <- group_by(df, clinic_id)
> df <- mutate(by_clinid, max_week = max(Received_Week)
> by_clingroup <- group_by(df, clinic_id, Received_Week)
> df <- mutate(by_clingroup, status = ifelse(Received_Week == max_week &
Received_Week < as.Date("2016-07-03"), 0, 1))
```

Now, when an account is lost there is a 0 for status. The data is now processed for analysis. We may export to a csv for archival purposes.

```
write.csv(df, "cox_processed.csv")
```

Let's remove the non-unique rows and save the columns we're interested in.

```
data <- distinct(df, clinic_id, Received_Week)
data = data[,c("clinic_id","volume","age","status", "Received_Week")]
```

2.4 Cox PH Analysis

Now, we have interpretable data so we can test assumptions, look for significance, and create a model. We will need the R "survival" package.

```
> install.packages("survival")
> library(survival)
```

We subset the data into 3 training segments in order to assess its simulation and predictive capabilities.

Table 3: Cox PH Metrics

Variable	Description	Utility
clinic_id	corresponding to a unique account	unique accounts can be tracked to estimate underlying hazard function
volume	total days an account is open	initial covariate of interest
age	total tests ordered over entire period	tracks time from account open
status	1: alive 0: dead	pinpoint "death" to uncover predictors signifying arrival of "event"
Received_Week	week test was received	subset data for training/testing

Table 4: Simulations

	Training	Test
1.	1/5/15 to 12/15/15	1/5/16 to 6/5/16
2.	1/5/15 to 4/5/16	4/6/16 to 6/5/16
3.	1/5/14 to 12/15/14	1/5/15 to 6/5/15

These correspond to 1 quarterly and 2 biquarterly simulations. In effect, we are assessing whether we can predict the probability of attrition accurately over the next 3 or 6 months given the past 12-15 months. Our third simulation during a time of more predictable company dynamics serves as a good comparison to our first and second simulations. This approach is anecdotal, however it is OK for our initial purposes. We eliminate the last two weeks of December when possible as the holiday season poses unusual circumstances.

Our model will be training to the training set. Then, it can make predictions as to when/which open accounts will be lost in the near future. This information can be analytically compared to the true test set. Furthermore, the model can make predictions upon the test set.

2.5 Model Building

We begin with simulation 1.

First, we subset the training and test sets.

```
> training = filter(data, Received_Week > as.Date("2015-01-04"), Received_Week <
as.Date("2015-12-16"))
```

```
> test = filter(data, Received_Week > as.Date("2016-01-04"), Received_Week <
as.Date("2016-06-06"))
```

Our analysis needs a baseline. Hence, we may create this variable now that we have selected our time-segment of interest.

```
> by_clinid <- group_by(training, clinic_id)
> training = mutate(by_clinid, start = age - min(age))
> by_clinid <- group_by(test, clinic_id)
> test = mutate(by_clinid, start = age - min(age))
```

Furthermore, in the treatment of time-varying covariates it is necessary to define intervals of time for our coxph function. Of course, our intervals are weekly so we simply add 7 days to our start variable.

```
> training = mutate(training, stop = start + 7)
> test = mutate(test, stop = start + 7)
```

Now, we can use the R *coxph* function with our covariate being volume. We may need to convert the baseline variables to a numeric type if it is stored as a date difference.

```
training$start = as.integer(training$start)
training$stop = as.integer(training$stop)
trfit <- coxph(Surv(start, stop, status==0) ~ volume + age, data = training)
```

Using the summary function we can come to various conclusions.

```
 > summary(trfit)
```

The variable "n" corresponds to the sum of distinct accounts placing orders on a given week, with "number of events" corresponding to the number of closings of accounts.

We may be interested in testing the assumption our proportional hazards model is reliant upon, that our covariate is multiplicative to the hazard.

```
> cox.zph(trfit)
```

A significant p-value would imply that our assumption is not a good one. Our model initially assigned a p-value on the order of 10^{-7} to the covariate of volume.

Hence, it is clear that the proportional hazard assumption is not a good one with the case of volume. Therefore, we may consider changing models altogether: an accelerated failure time model avoids the linearity assumption of the Cox PH model.

However, an alternative is reasoning intuitively why this assumption failed. Of course, volume is closely tied to the time of measure. The effect obviously varies depending on time in a non-linear fashion.

Therefore, we reasoned, perhaps, the inclusion of the interaction term between volume and start time will address our issue.

```
> trfit <- coxph(Surv(start, stop, status==0) ~ volume + baseline +
 volume:baseline, data = training)
> cox.zph(trfit)
```

The cox.zph function failed to reject the null hypothesis. Hence, linearity is no longer a bad assumption.

Ultimately, we found multiple useful covariates and interacting terms.

```
#repeat for test
> by_clinid <- group_by(training, clinic_id)
> training <- mutate(by_clinid, cumvol = cumsum(volume))

> trfit<- coxph(formula=Surv(start, stop, status==0) ~ volume + volume:baseline
 + baseline + cumvol + age + baseline:age + age:cumvol, data = training)
```

Now, we extract the baseline hazard function.

```
> bhaz_tr <- survfit(trfit)
> plot(bhaz_tr)
> plot(bhaz_tr$time, bhaz_tr$cumhaz, xlab = "Time from account open (days)",
ylab = "Cumulative Hazard", main = "Hazard Rate")
```

The first plot shows us the proportion surviving at each time point. Simply, it is showing the proportion of accounts still open. This plot is particularly important, as the integral or area under the curve of the survival function shown is the expected value of life. Hence, we will use this value to estimate the time an account will stay open.

However, the second plot is also of interest. Shown is the cumulative hazard against time in days. The cumulative hazard function is **not** a probability. However, it is indeed a measure of risk. The greater the cumulative hazard, the greater risk of failure at time t.

Note the shape of the cumulative hazard function. The cumulative hazard is the integrated hazard function, which is greater than 0 at each point, so it cannot decrease. However, you may be interested in the acceleration of the cumhaz function (hazard velocity) as when our function slows our hazard is decreasing past this point.

We may even examine the effects of changes in variables.

```
plot(survfit(trfit, newdata = data.frame(volume = 1, start = 200)
```

Plotted will be the estimated survival function of an account of age 500 that orders 1 test in a given week. Our plot indicated a visual estimation of just over 500 days upon integration. Hence, we can guess

that an account like this would close soon thereafter, as predicted by our model.

The same methods can be used for the other 2 training sets.

2.6 Adding Covariates

2.7 Predictions

Now, that we have built our models for all 3 training sets we seek to test its prediction accuracy on our test sets.

```
> pred <- predict(trfit, newdata = test, type ="risk")
> pred <- as.data.frame(pred)
> test <- data.frame(pred , test)
```

For each row in our test set we now have a "risk" value. This value is defined as the risk score exp(lp), where lp is the linear predictor. Hence, this value indicates the hazard ratio, or how likely the account is to be lost compared to an average client. For example, a value of 2 implies that the account is twice as likely to be lost than the average. Evidently, this data is rather meaningful. Let's see what information we can extract to assess the effectiveness.

```
> filt_test <- filter(test, status == 0)
```

We begin our analysis by filtering our data for client that were actually lost at the time of close to visualize what our model had predicted in terms of risk score. Next, we seek some visualizations.

```
> hist(filt_test$pred, xlab = "Risk Score", main = "Risk Score at time of Loss")
> summary(filt_test$pred)
```

Now, let's create a table and export a csv with the information described in 1.

```
> filt_test = mutate(filt_test, predicted = ifelse(pred>1.5, 1, 0))
> write.csv(filt_test, "test.csv")
> summary(filt_test$predicted)
> sum(filt_test$predicted)
```

This is great, if our interest is to rank the likelihood of an account being lost. However, in this context risk analysis is tasked with putting a price on retaining a customer. It may not always be true that we target the customer that is most likely to leave first. In practice, often times we will take a slightly less risky customer that is a higher volume consumer and attempt to maintain their business.

Therefore, a key reason we selected a proportional hazard model in the first place and a key advantage over a generalized linear model like the logistic regression is its ability to estimate survival time.

Of course, the integral of the survival density function multiplied by time from 0 to ∞ will give us the expected survival time value. However, this poses a difficult calculation as although the survival function goes to zero in the $\lim_{t\to\infty}$ it may require an excessively large interval. Hence, we use an approximation of the maximum baseline time considered in our training set model as our upper bound.

The survfit function shown earlier gives two vectors of survival probability density and time values. Hence, we do not have a function to integrate analytically. Therefore, we will use the trapezoidal method of approximation.

It would be trivial to write our own function to do this. However, the "kulife" package already has one written for us.

```
> install.packages("kulife")
> library(kulife)
# create null object
> surv_est = 0
# for training
> bhaz_tr <- summary(survfit(trfit, newdata = training))
> for(i in 1:32845){surv_est[i] <- auc(bhaz_tr$time, bhaz_tr$surv[1:44,i],
 from=0, yleft=1)
> surv_est = as.data.frame(surv_est)
> training <- data.frame(surv_est, training)
> training <- mutate(training, est = surv_est - baseline)
# for test
> bhaz_tr <- summary(survfit(trfit, newdata = test))
> for(i in 1:72894){surv_est[i] <- auc(surv_test$time, surv_test$surv[1:74,i],
 from=0, yleft=1)
> surv_est = as.data.frame(surv_est)
> test <- data.frame(surv_est, test)
> test <- mutate(test, est = surv_est - baseline)
```

Now, we have a full estimator for survival.

3 Discussion

4 Conclusions

Although a more simple, linear model would've saved a great amount of time, in predictive analytics it is obviously crucial to create an interpretable and accurate model that can help locate important metrics to follow and perhaps signal restructuring.

4.1 Cox PH as an Effective Model in Sales

Evidently, the Cox PH model can predict, to some good estimation, the likelihood that an account will be lost.

4.2 Final Thoughts

Overall, when any model is selected it is gravely important to test the respective assumptions. Hence, a reasonable evaluation of alternatives would involve whether this model relies on less sound assumptions than the other.

5 Shiny Setup

First, we launch an EC2 instance on Amazon AWS. We choose a compute-optimized server with 16 cores to suit our computationally intensive needs (c4.4xlarge).

Next, we SSH into the server. We must install all relevant dependencies.

```
    # first we install R
sudo sh -c 'echo "deb http://cran.rstudio.com/bin/linux/ubuntu trusty/" >>
 /etc/apt/sources.list'
gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
    gpg -a --export E084DAB9 | sudo apt-key add -
    sudo apt-get update
    sudo apt-get -y install r-base
    # then Shiny
    sudo su - -c "R -e \"install.packages('shiny', repos=
    'https://cran.rstudio.com/')\""
$ sudo apt-get install gdebi-core
$ wget https://download3.rstudio.org/ubuntu-12.04/x86_64/shiny-server
-1.4.2.786-amd64.deb
$ sudo gdebi shiny-server-1.4.2.786-amd64.deb

sudo /opt/shiny-server/bin/deploy-example default
```

Now, we can modify the app.R code with code we've tested in RStudio by launching on the localhost. Hence, we can access the server's public IP address from our browser and see our Shiny app. Alternatively, we can associate it with a domain for a more accessible endpoint.

6 Common App Usage

Here, we'll address in a condensed matter the common uses of the Shiny app alongside the interpretation of the plots and statistics provided.

6.1 Accessing the Shiny Endpoint

The software is hosted on an EC2 compute-optimized instance. I've written code to manually parallel process information across each of the 16 cores. This is particularly useful when forming the survival forest, as each core can actively build a tree. Hence, the complete model can be built in approximately 10 minutes.

To start the server it is required to log on to the company's AWS account, change the region to East, and then select EC2 instances where you'll be able to start and stop the server. When the server is running you can check the README for the URL access endpoint and associated username and password.

6.2 Making Predictions

A Kits Received CSV can be uploaded with essentially no processing. All of the filtration will happen automatically. However, it's crucial that the columns Product_Group, Region_Group, clinic_id and Received_Week retain the same name throughout time, otherwise R will not be able to search for these columns. Adding or deleting other columns, however, will not affect model performance. It is imperative that the uploaded CSV has the start date of at minimum one year prior to the first date included in the training dataset.

In example, if one uploads a CSV that begins in 2015 and has a training range that begins on that same date, the data will be biased toward short-lived accounts, because all deaths in the training set will have been from accounts that opened and closed in 2015. Hence, the model will falsely predict age or other variables collinear with time, as overwhelmingly good predictors.

6.3 Adding Covariates

The app was built for the intelligent addition of covariates, with no expectation of format or variable type. Hence, almost any type of supporting data can be included in your model. This works by virtue of an array of well-defined processing algorithms. Essentially, an additional CSV can be uploaded using the corresponding button so long as it fits in the following category:

1. The data identifies the LIMS ID in each row
2. Each row corresponds to a unique event (e.g. sales call)
3. Each row contains a date that is associated with the event in the format specified in the sidebar. This date does not have to be in the same kits received weekly format. The software will determine automatically which week the event belongs to.

Note that when working with relational databases, almost all data is stored in this manner. Hence, in this case we're not interested in any of the columns that aren't uniform across each data point in the CSV itself. We're primarily interested in the raw number of rows that exist for each clinic-week data point. An example would be sales calls. While none of the columns in the Salesforce CSV may be of interest, we are interested in how many sales calls actually took place between each clinic on a given week. Therefore, we'd like to aggregate the data and calculate this sum and see if it is a legitimate predictor. Note that if a column exists that is a characteristic of the account at all or a particular time point, it may be of interest, as it may describe the account itself.

The data will automatically be aggregated and associated to the clinic and the kit received week. Generally, it's expected that the column describing the LIMS id and the date will be named differently in each CSV. Hence, you'll select which column corresponds to each variable in the sidebar. Furthermore, dates come in a range of formats so you'll tell the app which format your CSV has stored dates.

The recommended progression when working with both an additional covariate CSV and kits received is as follows.

1. Upload additional covariate CSV at the bottom of the sidebar, ensuring that the CSV spans the same range of the training dataset and has values for the time point of prediction
2. Select which columns correspond to clinic id and date
3. Select the date format
4. Check the "Additional Data" tabset at the bottom to ensure LIMS ID and week of event were processed correctly
5. Upload the kits received CSV at the top of the sidebar, ensuring the first date in the uploaded file precedes the training dataset by at least one year
6. Select add_data.density to be included in core data from the sidebar
7. Select add_data.density to be included in the group of regressors from the sidebar