ML for Time-to-Event Predictions

Noah Simon

Jan, 2021

Overview for Today

We will

- Relate predictive modeling with time-to-event outcome with ML techniques for continuous/binary outcome
- Discuss several contemporary methods (and software)
- Discuss validation and selection of tuning parameters
- Talk about "calibrating risk scores"

A few motivating problems

Suppose I would like to develop a prognostic biomarker signature for people with a certain type of cancer

I might measure gene expression profiles from tumor biopsies...

and try to use that to build a predictive model for time to progression (but some patients leave the study before progressing)

Generally I have many more candidate features (genes) than observations (people)

How can we do this?

A few motivating problems

Suppose I have a bunch of patients with macular degeneration (a disease of the eye).

I take a retinal image when they enter the study...

and follow up on them until their disease progresses

There is a moment of clear structural change that we can observe

How can I learn, from the image, how to differentiate patients at higher vs lower risk of progressing?

Additionally, how can I predict time to progression?

Time to an event!

Known as "time-to-event" outcome.

Statistical toolset for this often known as *survival analysis*.

Incredibly common in biomedical studies!

Also used by tech companies to, eg., identify factors that influence people to discontinue service/quit their job

---

Time to an event!

Known as "time-to-event" outcome.

Statistical toolset for this often known as *survival analysis*.

Incredibly common in biomedical studies!

Also used by tech companies to, eg., identify factors that influence people to discontinue service/quit their job

---

In what follows we will use x to refer to the measurements/features that we intend to use to predict time to event.

Often x is only measured at baseline.

Time to an event!

Time to event outcome is a bit tricky!

Generally modeled as a Poisson process:

Each pt has a biased coin w/ $P(heads)$ determined by their risk

Every second[1], nature flips that coin and if it turns up heads, then they have the event!

We are trying to learn each patient's probability of heads!

[1] I have discretized things for ease of understanding

Time to an event!

There is an added complication:

Even for a given patient, their prob of heads will change over time!

So now we need to try and learn...

$\pi(t, x)$ the prob of heads...

at time t, for a patient with feature-values equal to x.

Once we have $\pi(t, x)$ for a patient (over a large range of t), can get things like survival functions, median survival time, etc...

Baseline vs Time-varying covariates

Engaging with time-varying covariates is quite difficult!

For the majority of this session I will assume covariates are measured at some baseline [though with left truncation allowed].

I will try to mention when methods permit time-varying covariates

Predictive Modeling

Before moving further with time-to-event outcome...

let's return back to simpler predictive modeling problems.

We will look at these from both statistical and ML viewpoints.

This will help motivate tools we can use for time-to-event prediction!

Predictive Modeling

Often have features $\mathbf{x}_i$, and outcome y_i measured on pts $i = 1, \ldots, n$.

Our goal is generally twofold:

- To build a predictive function $f(\cdot)$, so that $f(\mathbf{x})$ is maximally associated with y.
- For each value of $f(\mathbf{x})$ to identify likely values of y.

Statistical Modeling

In statistical modeling...

Generally have a probabilistic model for our data

With some unknown parameters that we need to estimate

Those parameters are estimated to make our model most consistent with our data

Statistical Modeling

More generally Suppose $\mathbf{x} \in \mathbb{R}^p$ is a numeric vector

and y is a simple numeric outcome.

The modeler/statistician assumes

$$y_i = x_i^\top \beta^* + \epsilon$$

and fits a maximum likelihood estimate

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^\top \beta \right)^2$$

Predictive Modeling

However, we could easily divorce this from a probabilistic model...

Predictive Modeling

More generally Suppose $\mathbf{x} \in \mathbb{R}^p$ is a numeric vector

and y is a simple numeric outcome.

The predictive modeler wants to find β to minimize avg error between outcome and prediction

$$\ell(y, \eta) = (y - \eta)^2$$

and fits a model to minimize the empirical error

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^\top \beta \right)^2$$

Predictive Modeling

We can play this game for many types of outcome/statistical models!

Inferential Modeling - II

More generally:

Suppose $\mathbf{x} \in \mathbb{R}^p$ is a numeric vector

and y is a symmetric binary (1 or -1).

The modeler/statistician assumes

$$P(y_i = 1) = \frac{e^{x_i^\top \beta}}{1 + e^{x_i^\top \beta}}$$

and fits a maximum likelihood estimate

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \log \left(1 + e^{-y_i (x_i^\top \beta)}\right)$$

Predictive Modeling

Again, we could easily divorce this from a probabilistic model...

Predictive Modeling - II

More generally:

Suppose $\mathbf{x} \in \mathbb{R}^p$ is a numeric vector

and y is a symmetric binary (1 or -1).

The predictive modeler wants to find β to minimize avg error between outcomes and predictions

$$\ell(y, \eta) = \log\left(1 + e^{-y\eta}\right)$$

and fits a model to minimize the empirical error

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y_i(x_i^\top \beta)}\right)$$

Predictive Modeling - II

More generally:

Suppose $\mathbf{x} \in \mathbb{R}^p$ is a numeric vector

and y is a symmetric binary (1 or -1).

Or perhaps, the predictive modeler uses hinge loss (for SVM)

$$\ell(y, \eta) = [1 - y\eta]_+$$

and fits a model to minimize the empirical error

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \left[1 - y_i(x_i^\top \beta)\right]_+$$

Predictive Modeling

In predictive modeling, there are 2 things we can change:

- The measure of loss $\ell(y, \eta)$
- The set of candidate models we consider:

  perhaps something more interesting than $f_\beta(x) = x^\top \beta$...

Predictive Modeling

Rather than using $f_\beta(x) = x^\top \beta$...

can use more interesting/complex model, eg...

- Function with interactions
- Polynomial (/piecewise polynomial) function
- Neural Networks

In all these cases, we [approximately] solve

$$\hat{\beta} = \text{argmin}_{\beta \in B} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f_\beta(x_i)\right)$$

and set $\hat{f} = f_{\hat{\beta}}$

Machine Learning in this Lens

Almost all supervised learning procedures can be seen as heuristics for trying to solve

$$\hat{\beta} = \text{argmin}_{\beta \in B} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f_\beta(x_i)\right)$$

over an appropriate B, with some specified f_β, and ℓ.

Some examples:

- Trees
- Boosting/Bagging
- Neural Networks/Deep Learning
- Support Vector Machines
- Lasso

Machine Learning in this Lens

Sometimes we add penalties[2]

$$\hat{\beta} = \text{argmin}_{\beta \in B} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f_\beta(x_i)\right) + \lambda P(\beta)$$

This allows us to reduce the complexity of our model (by increasing the value of the penalty parameter λ).

[2] These could equivalently be given as constraints on B

Reduction to Optimization

$$\hat{\beta} = \text{argmin}_{\beta \in B} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f_\beta(x_i)\right)$$

This reduces a large piece of predictive modeling to an optimization problem.

Survival Data?

How does this fit with time-to-event/Survival data?

There are 2 main approaches:

1. Try to estimate an entire conditional survival curve
2. Try to do risk stratification (identify a univariate risk score that summarizes the features)

This second approach most directly engages with those ML ideas

The most common tools generally engage with the second approach.

Risk Stratification - I

In risk stratification, we rank survival curves as a function of x

(often done without actually estimating the curves!)

If the shape of the survival curve highly depends on x...

and you want to understand that shape better

Risk stratification (ranking) may not be the right approach!

Risk Stratification - II

In risk stratification, we rank survival curves as a function of x

(often done without actually estimating the curves!)

In high dimensional/complex problems with limited data, even ranking is hard!

In my mind trying to do something more subtle (learning differences in curves between similarly ranked people) may be asking too much!

I am going to focus on risk stratification (and how it connects to proportional hazards regression and ML)

Risk Stratification - III

Try to do risk stratification

(identify a univariate risk score that summarizes the features)

We will primarily focus on methods for this task

These scores can later be calibrated to give survival curves.

To build a risk score, we need a method to evaluate the goodness of fit of a given $f_\beta(x)$ with the observed data!

Survival Data

For each patient we measure:

- Values of covariates, x
- and T the time that an event of interest happens

There are often 2 added complications:

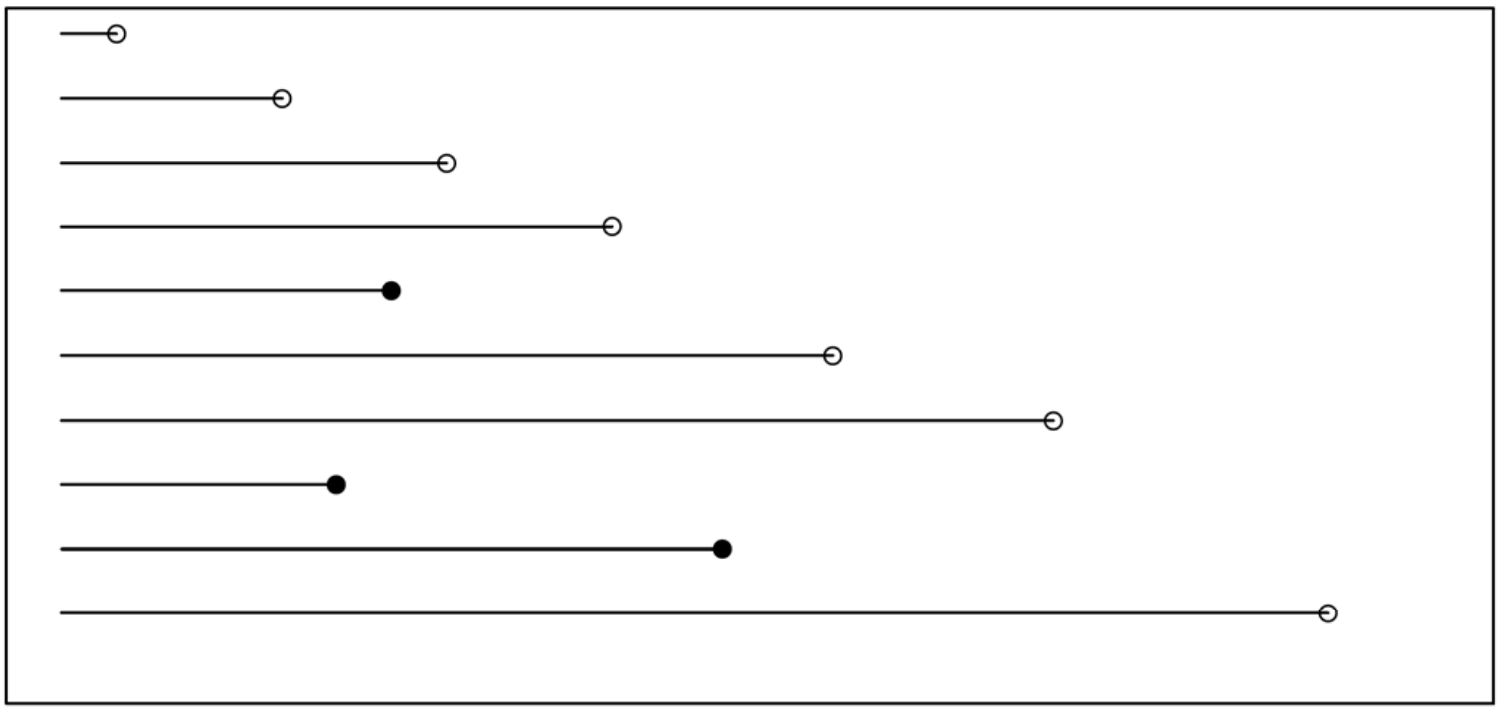
- Not every patient experiences an event — some leave the study first

  (right censoring)

- Not everyone enters the study at the same "time"

  (left truncation)

Survival Data, picture



Survival time (all start at zero)

Cox Model (statistician's view)

As a reminder, we want to understand

$$\pi(t, x)$$

The prob of having event at t-th time increment; for a patient with feature-value x (conditional on having not yet experienced the event)...[3]

In statistical modeling, a simplifying assumption is often made:

Proportionality of hazards

This is sometimes known as the *Cox model*

[3] I have discretized things here, again for ease of exposition

Cox Model (statistician's view)

In Cox model, we assume[4]

$$\pi(t, x) = \pi(t) e^{f_\beta(x)}$$

We imagine there are two parts:

- The baseline rate $\pi(t)$ that is covariate independent
- The covariate-specific modifier $e^{f_\beta(x)}$

We generally primarily care about f_β

[4] Should formally be done in continuous time with hazards

Cox Model (statistician's view)

Often primarily interested in *risk stratification*...

Identifying which patients are particularly high risk of event

In that case, $e^{f_\beta(x)}$ is target of inference...

It is the only thing that changes between patients

Fitting the model

To estimate $e^{f_\beta(x)}$

Could try to write out likelihood of data and maximize it...

This ends up being bit of a bear!

would also require estimating $\pi(t)$

---

proportional hazards allows us to employ a nice trick!

Stepping back

Before continuing to talk about the cox model...

consider a seemingly completely different approach

Ignoring our model for a second...

Want a function $f_\beta(x)$ that "stratifies risk".

Ideally...

Larger $f_\beta(x) \to$ earlier event

Smaller $f_\beta(x) \to$ later event

Stepping back

Let's try to build a scoring function ℓ based on this...

to quantify how well a given f_β fits our data

This is tricky! Because also need $\pi(t)$ to predict event times...

Instead Cox had a clever idea!

Measure goodness of fit based on concordance!

A comparative measure!

Fitting the Model

Consider a pair of patients (Zoe and Greg)

Suppose $T_{Zoe} < T_{Greg}$

A "concordant" score would assign higher risk to Zoe than Greg

---

When fitting our model, we want to maximize [average] concordance [5]

[5] Idea of concordance for comparisons of censored data discussed as early as 1962, by Gehan and Gilbert (separately)

Concordance Measure - I

Simplest concordance measure is

$$I\{f_\beta(x_{Zoe}) > f_\beta(x_{Greg})\} I\{T_{Zoe} < T_{Greg}\}$$
$$+ I\{f_\beta(x_{Zoe}) < f_\beta(x_{Greg})\} I\{T_{Zoe} > T_{Greg}\}$$

This is just a binary indicator that the higher risk pt experienced the event first.

This is related to the AUC for binary outcome.

Concordance Measure - II

Another simple, and important, concordance measure is

$$\log \left[\frac{e^{f_\beta(x_{Zoe})}}{e^{f_\beta(x_{Zoe})} + e^{f_\beta(x_{Greg})}} I\left\{T_{Zoe} < T_{Greg}\right\} + \frac{e^{f_\beta(x_{Greg})}}{e^{f_\beta(x_{Zoe})} + e^{f_\beta(x_{Greg})}} I\left\{T_{Zoe} > T_{Greg}\right\} \right]$$

This is the [log] "share of the risk" of the pt with the first event.

Just a smoothed version of the previous measure.

Partial Likelihood

Cox noticed that this was related to the "Cox model"

In particular, he proposed estimating f_β by maximizing the [log] partial likelihood

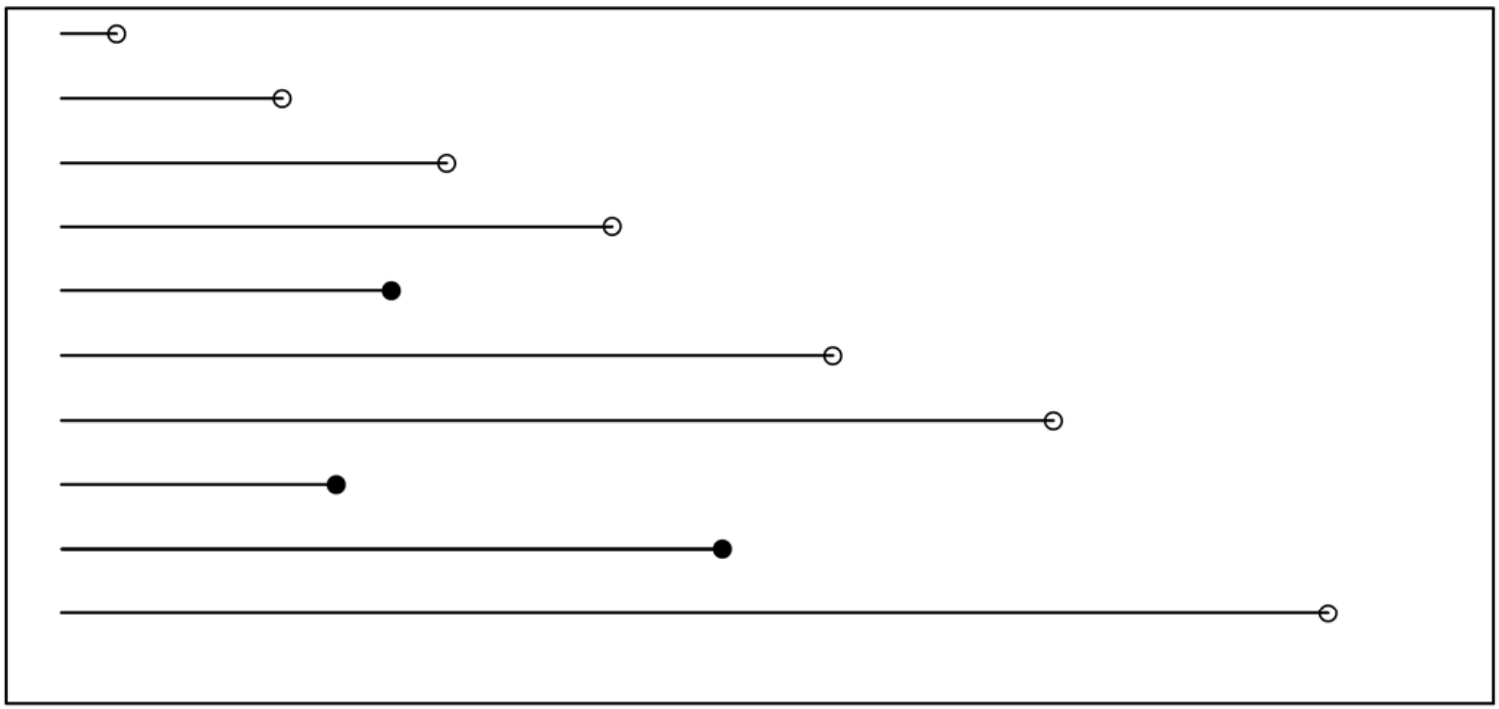
This is an extension of the previous concordance measure that applies to 2+ observations

---

$$L_n(\beta) = \sum_{i \in \text{event set}} \log \left(\frac{e^{f_\beta(x_i)}}{\sum_{j \in R_i} e^{f_\beta(x_j)}} \right)$$

Here R_i is the set of pts in the study, who have not yet experienced the event before time T_i.

Example



Survival time (all start at zero)

Partial Likelihood

Consider the *partial likelihood* in context of horse racing...

As each horse finishes the race, we compare the score of the finishing horse, to all horses still running!

Want to construct scoring function f_β, so that each time a horse finishes... it has the largest score of its remaining competitors.

Partial Likelihood

Consider the *partial likelihood* in context of horse racing...

Equivalently, want the order of the scores to be in agreement with the order of the horses finishing.

The partial likelihood is a *smooth* way of quantifying the level of that agreement

Horse Racing Segue

My understanding is that the approach above was, at one point, state-of-the-art for betting on horse racing [6]

It's actually essentially a version of "conditional logistic regression" there

As we move forward we can think about how ML could be applied using the *partial likelihood* or *conditional logistic* measure of goodness-of-fit.

[6] probably 25 years ago... given secrecy around these things!

Coarsening the Time

We have removed much of the information about time here...

Only retain ordering of the event times when fitting the model

Returning to our model

Just discussed an intuitive but ad-hoc way to identify a function $\hat{f}_\beta$ for risk stratification.

In fact, in our model, if proportional hazards holds

$$\pi(t, x) = \pi(t) e^{f^*_\beta(x)}$$

Cox's approach is statistically well-founded[7] for estimating $f^*_\beta(x)$

[7] in fact optimal

Returning to our model

Just discussed an intuitive but ad-hoc way to identify a function $\hat{f}_\beta$ for risk stratification.

In fact, in our model, if proportional hazards holds

$$\pi(t, x) = \pi(t) e^{f_\beta^*(x)}$$

Cox's approach is statistically well-founded[7] for estimating $f_\beta^*(x)$

eg. $\hat{f}_\beta \to f_\beta^*$ as the number of observed pts grows

[7] in fact optimal

Returning to our model

In our probabilistic model, if proportional hazards holds

$$\pi(t, x) = \pi(t) e^{f_\beta^*(x)}$$

Cox's approach is statistically well-founded for estimating $e^{f_\beta^*(x)}$

And we managed to avoid estimating $\pi(t)$

Returning to our model

In fact, when you fit a [linear] Cox PH regression model, you are solving:

$$\hat{\beta} \leftarrow \text{argmin}_{\beta} \sum_{i \in \text{event set}} \log \left(\frac{e^{x_i^\top \beta}}{\sum_{j \in R_i} e^{x_j^\top \beta}} \right)$$

This is what, eg., the `coxph` function in `R` does.

Often this is done for inferential reasons...

but $x_i^\top \hat{\beta}$ could be taken as our risk score

and, in many cases, performs as well as fancier methods!

Machine Learning Version

This roughly fits into our ML paradigm...

We have a "loss" $-L(\beta)$ which characterizes goodness-of-fit

We can modify our function class f_β to generate different estimators

Again, this allows some ML-ish extensions

- Survival Trees/Forests
- Cox-Lasso
- Cox-Neural Networks

Machine Learning Version

Note, our loss $-L(\beta)$ has a few minor issues:

- It doesn't separate simply over our observations
- Resulting score doesn't immediately give a survival time prediction

This issues can be overcome!

Proportional Hazards

For inferential survival modeling, often concerned about proportional hazards (potentially for good reason)

Here, I don't really worry about proportional hazards.

Partial likelihood is just a way to measure concordance between a candidate risk score and observed event times (goodness of fit).

Will want to do an independent (and likely non-parametric) evaluation of our risk score afterwards...

If proportional hazards is strongly violated, we might get a poor risk score... But probably only when risk stratification is impossible (eg. because of the shape of the survival curves)

ML methods in this context

We will engage with a few ML methods for survival data in this context...

- Penalized Cox Regression
- Survival Trees
- Survival Tree Ensembles

  (bagging/random forests + boosting)
- Deep Learning for Survival Models

Cox Regression

For Cox regression... We try to identify a linear combination of features that maximizes *concordance* between predicted scores and failure times

Here *concordance* is measured using the partial likelihood

When the # of features (p) is $>>$ # of obs (n) this model will *overfit* the data.

Penalized regression is a framework for fitting less complex models than standard cox regression.

Penalized Cox Regression

We choose a penalty $P(\cdot)$ and a tuning parameter $\lambda > 0$ and solve

$$\min - \sum_{i \in \text{event set}} \log \left(\frac{e^{x_i^\top \beta}}{\sum_{j \in R_i} e^{x_j^\top \beta}} \right) + \lambda P(\beta)$$

We have

- the negative log partial likelihood which measures poorness-of-fit
- the penalty function, which encourages simpler fits
- the tuning parameter, that determines the tradeoff between goodness-of-fit and model complexity

Penalized Cox Regression

Two common penalties are...

Ridge regression

$$P(\beta) = \sum_{j=1}^{p} \beta_j^2$$

The Lasso

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|$$

Both of these encourage coefficients to be small...

The lasso results in estimates with coefficients equal to *exactly* zero.

Choosing λ - I

λ here is known as a "tuning parameter"

(aka hyper-parameter, meta-parameter)

To employ penalized regression, we need to identify the "right" value for λ.

How do we do this?

Validating a Model - I

This is related to the general idea of validating a model

In general in stats/ML, to validate the performance of a model, one needs a dataset NOT used in constructing the model

The dataset used to construct the model is called the training data; the dataset used for evaluating is called the validation data.

When one has only a single dataset, it is common to randomly split the dataset to form training/validation data (or repeatedly do this using cross-validation)

Validating a Model - II

In addition to a validation dataset, one needs a way of measuring the predictive performance of a candidate model

For continuous outcome, often mean-squared error (between predictions and outcome) is used

For binary outcome, binomial deviance, misclassification rate, and AUC are common

---

As a reminder, when training our model we selected a measure of goodness/poorness of fit. The measure doesn't need to match between training and validation.

Validating a Model - III

With time-to-event outcome, and a set risk score, $f(\cdot)$, it is common to evaluate the predictive performance of the model on a validation dataset using some form of concordance.

One common choice is the predictive partial likelihood. There we calculate the partial likelihood using our fixed risk score f with only observations in the validation dataset

$$\sum_{i \in \text{val event set}} \log\left(\frac{e^{f(x_i)}}{\sum_{j \in R_i} e^{f(x_j)}}\right)$$

Another common choice is the c-index: There we just count the proportion of pairs of observations where predicted risk has the same order as the event times.

Choosing λ - II

What does this mean for choosing λ?

We generally use the following recipe:

1. Split our data [randomly] into training and validation sets
2. Build penalized regression models with many different candidate λ-values on the training data
3. Calculate the *concordance* for each of those models on the validation data
4. Select the model (λ-value) with the maximum concordance

Software for Penalized Cox Regression

Luckily, you generally don't need to do this yourself!

The `glmnet` package in `R` does all of this for you!

`cv.glmnet` function would be particularly useful here.

(predictive partial likelihood is used for model selection)

Flexible Penalized Cox Regression

Adding user derived features is a common trick to make penalized regression more flexible...

For example, with numeric features, if you want to fit an additive model you might create a new set of k features from each original feature, using, eg. polynomial transformations

$$\begin{pmatrix} x \end{pmatrix} \rightarrow \left(\begin{array}{c|c|c|c} x & x^2 & \cdots & x^k \end{array}\right)$$

Could additionally/alternatively include interactions

Recommendation

For $p > n$ scenarios I find cox regression with a lasso penalty very hard to beat!

In theory, there are strong restrictions on the selected models (eg. linearity, no interactions)

In practice, it is *very* hard to learn complex things from data with many features and few obs.

Survival Trees (and Ensembles)

If you are OK with a more black box model, combinations (ensembles) of trees can sometimes improve performance.

Before discussing ensembles of trees, let's quickly discuss how trees work

Survival Trees

[single] Tree-based methods partition the predictor space into a pre-specified # of rectangles...

And fit a piecewise constant fcn within those rectangles to minimize a loss

They try to optimally identify the shape of the rectangles and the value of the fcn within each rectangle to minimize the loss.

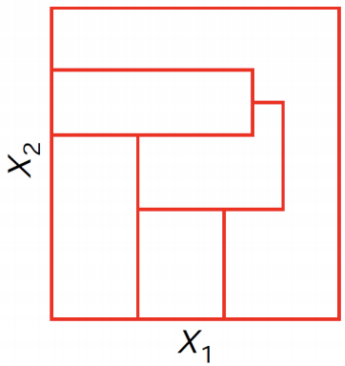
Classification and regression trees (CART) generally use MSE or binomial deviance for their loss

Survival trees often use loss based on the partial likelihood.[8]

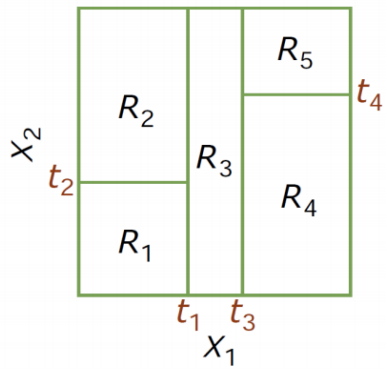
[8] This is not how they are commonly discussed, but they can be seen in this way!

Survival Trees

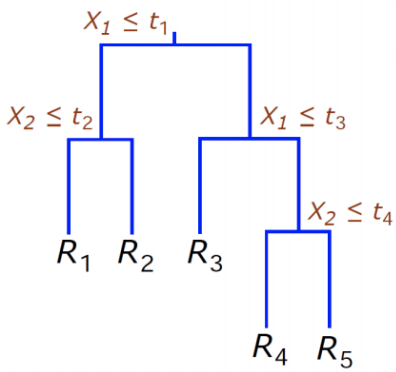
Here is a visual schematic of the risk score $f(\cdot)$ identified by a tree[9]



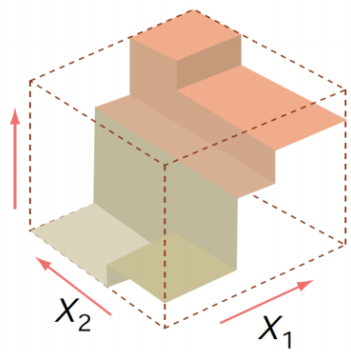
(a) General partition that cannot be obtained from recursive binary splitting.



(b) Partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data.



(c) Tree corresponding to the partition in the top right panel.



(d) A perspective plot of the prediction surface.

[9] borrowed from http://www.stats.ox.ac.uk/∼flaxman/HT17_lecture13.pdf

Survival Trees - Tuning

For survival trees, main tuning parameter is the number of splits/rectangles

Can again use training/validation sets to identify the best number of splits to use

---

`R` Packages `rpart`, and `MST` seem to be the most popular implementations for fitting single survival trees (they are slightly to somewhat different in the details, but its unclear to me if that makes much practical difference)

Ensembling Survival Trees - I

A single tree does not generally have better performance than [penalized] cox regression

However, by combining many trees, can get improved performance

In my mind, these tree ensembles are generally among the most competitive methods available

Ensembling Survival Trees - II

Two ways to ensemble trees:

- Bagging (/random forests)
- Gradient Boosting

I generally find boosting outperforms bagging

Though, Boosting is [a bit] slower to fit and cannot easily be parallelized

Bagging Trees and Random Survival Forests

We know that, in general, averaging reduces variance...

Bagging tries to take advantage of this

The recipe looks like this:

1. Create 1000+ bootstrapped copies of your dataset
2. On each one, fit a survival tree
3. To predict risk for future pts, use the average of the risk predictions from those 1000+ trees

Bagging Trees and Random Survival Forests

This simple strategy is extremely useful for improving the performance of trees!

Random survival forests add an additional component (randomly reducing candidate features considered at each step). This is also generally helpful (or at least not harmful)

RSF have two tuning parameters

- Number of splits for each tree

  (should generally be quite large!)

- Number of features to consider at each split

  (not used for simple bagging)

Random Survival Forests

There is, again, a bunch of software out there for this!

I most like the `ranger` package in `R` — it is very fast and seems quite robust

Boosted Survival Trees

Boosting is an alternative way to ensemble trees

Rather than fitting trees in parallel [to bootstrapped data]...

- Boosting sequentially fits small trees
- Each new tree is fit to the current "residuals" of the model

In this way, each new tree tries to improve predictions where the current model is performing poorly.

Final risk predictions are given by summing the risk predictions of all the trees

Boosted Survival Trees

I think boosting is one of the most important statistical/ML ideas for predictive modeling of the last 40 years.

Boosted tree algorithms are incredibly hard to beat for tabular data

Boosted Trees have two tuning parameters

- Number of splits for each tree

  (should generally be extremely small! 1-5 perhaps)

- Number of trees

There is also something called a "learning rate" — this should be small! (0.01, or 0.001). Smaller is better, but there are strongly diminishing returns (and computationally takes longer because more trees are required for smaller learning rates).

Boosted Survival Trees

There is, again, a bunch of software out there for this!

The `gbm` package in `R` is pretty easy to use (and works well)

The `xgboost` package in `R` and `python` is fancier and has more options (and is industry standard in tech, but it's not totally clear to me that it will gain much in practice)

Also some interesting new work that does not use partial likelihood loss, and allows for time-varying covariates (`BoXHED` python package).

Tuning the Models

As discussed before, tuning takes the following steps:

1. Split into training and validation datasets...
2. Fit models with various values of tuning parameters on training data...
3. Evaluate some form of concordance on validation data

Risk Scores vs Survival Curves

We have so far talked about building risk scores...

How do we use these scores to construct survival curves?

- I consider this similar to calibration for binary prediction models

From Risk Scores to Survival Curves - I

Let's imagine we have 2 datasets:

1. Model Construction dataset
2. Calibration dataset

Suppose we used the first data to construct a scoring function

How can we use the second dataset to estimate *personalized* survival curves?

From Risk Scores to Survival Curves - II

For ppl with a given score $f(x) = z_0$, would like to estimate their survival function

What we do is quite simple (and classical)

1. On calibration dataset, identify all patients with score z_i close to z_0

2. Form a Kaplan-Meier curve using only those patients with similar scores

In practice use weighted KM

(higher weights for people with z_i closer to z_0)

From Risk Scores to Survival Curves - III

Calibration could potentially be done with validation data

Can use cross-validation (/pre-validation) for this with limited samples

This can be used to predict eg. median survival time

Summary

Building risk prediction models with time-to-event data

- Related this to minimizing a [concordance-based] loss function
- Talked about parameter tuning using concordance on a validation set
- Discussed penalized cox regression, tree-based-estimation, and ensembling methods (bagging + boosting)

Predicting survival curves from risk scores

- Can use kernel-weight KM with risk scores as our covariate (on independent data)
- Similar idea to calibration with probabilistic classification

Deep Learning and Neural Networks

Deep learning has been used for a number of things...

One major application has been predictive modeling.

It has been particularly useful when features...

- are images/audio files or
- are sequences of objects like words or
- have other strong structure

(often spatial or sequential structure)

Deep Learning and Neural Networks

What is deep learning/what are NNs?

Neural networks are just a particularly flexible type of function $f_\beta(\cdot)$ where...

- β is a finite set of continuous parameters that modify how f_β maps its inputs to its output(s)
- For each x, $f_\beta(x)$ is a smooth [differentiable] function of β

A hardcore ML person might say that $f_\beta(x) = x^\top \beta$ is a NN

Deep Learning and Neural Networks

Neural networks have a bunch of "tuning parameters" that determine how high dimensional the parameter vector, β, is.

They also determine specifics of the functional form of f_β.

Higher dimensional β-vector $\rightarrow$ more flexible model

(and the more prone we are to overfitting!)

Essentially NN models with higher dimensional β are known as *deep learning* models[10]

[10] This is a slight oversimplification

Deep Learning for Survival Data

This fits into our paradigm from before...

When we use NNs for predictive modeling with continuous/binary outcome, we are trying to solve:

$$\hat{\beta} = \text{argmin}_\beta \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f_\beta(x_i)\right)$$

where β are just the parameters in our NN.

With time-to-event outcome we could just switch to

$$\hat{\beta} = \text{argmin}_\beta \sum_{i \in \text{event set}} \log\left(\frac{e^{f_\beta(x_i)}}{\sum_{j \in R_i} e^{f_\beta(x_j)}}\right)$$

In practice, for large datasets we may need to modify this for computational reasons

Deep Learning for Survival Data in Practice - I

If you don't have...

- *very* structured input data
- A lot of obs, at least a few thousand I would think

I wouldn't worry about using NNs/Deep learning. I would be very surprised if NNs would improve on gradient boosting results there.

Deep Learning for Survival Data in Practice - II

If you want to try out NNs, there is a bit of a learning curve:

You need to use a package to "specify" your neural network and how you want to fit it

Common packages for this are `pytorch` or `tensorflow`

There is a somewhat easier way to engage with these using a package called `keras`. All of these require a bit of scripting.

There is a nice python package `PySurvival` that connects to tensorflow under the hood[11]

[11] it has several options — DeepSurv and MTLR are good NN options

Deep Learning for Survival Summary

In this last part we...

- (very) briefly mentioned how NNs and DL for survival data fit into our previous framework
- talked about when NNs/DL might be useful to try (and when not)
- touched on software for "training" those networks with time-to-event outcome

---

For more discussion on NNs (as well as a python tutorial) I like the free book `https://d2l.ai/`. Additionally, `https://www.deeplearningbook.org/` is also a good free book.