C++ Implementation and R Package of GuanRank

BIOSTAT 615 Final Project

Kevin Liao Chen Sun Wenjin Gu

December 9, 2018

1 Introduction

Survival analysis is essential for therapeutic decision making and clinical research. There is an urgent need to develop more reliable and robust models for estimating the survival from massive time-to-event data. One of the main challenges of analyzing time-to-event data is the censoring problem, which means the patient status is not fully available due to tracking interruption or time limit of the study. In this case, it is not advisable to use prediction models directly. Many statistical and machine learning applications have been developed to handle censored data. The Cox proportional hazards (PH) model is the most commonly used survival prediction technique and estimates the parameters using the partial likelihood function by assuming a proportional hazards condition.[1] However, the Cox model does not take into consideration the early-censored condition in the calculation of partial likelihood calculation and the proportional hazards condition it assumes may not be met in many situations.

GuanRank is a non-parametric complete hazard ranking model that converts the right-censored data into a linear space. By performing this transformation commonplace machine learning techniques, such as Gaussian process regression and random forests, can then be applied.[2] In addition, GuanRank is the winning algorithm in prediction accuracy in the 2015 ALS Prize4Life DREAM challenge and the 2017 Multiple Myeloma DREAM Challenge.

In this project, we implemented GuanRank in C++ and built an R package based on it. Currently, GuanRank has only been implemented in Python and there exists a need for an R implementation because a lot of survival analysis is conducted in R. Therefore, our project aims to fill this gap by both creating an R package for GuanRank, as well as optimizing the algorithm using the C++ programming language. As a result, our package could be used by a wider array of researchers and benefit the overall field of survival analysis and clinical research. The source code is available at https://github.com/verne91/GuanRank.

2 Methods

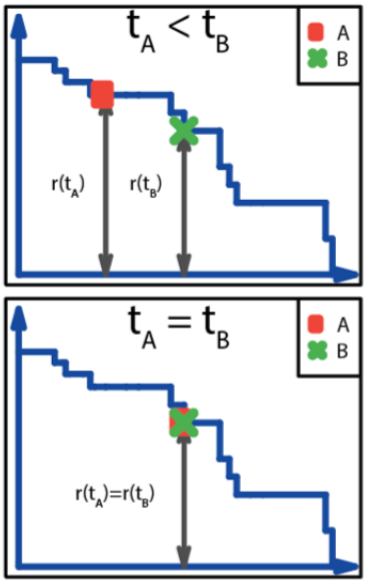
2.1 Implementing GuanRank in C++

There are five main parts in our implementation: 1) create a patient object 2) read the data from csv file 3) calculate the survival probability from Kaplan-Meier curve for each patient 4) calculate GuanRank scores based on pairwise comparisons between patients 5) normalize the GuanRank

scores.

To store the information for each patient, we created a patient class with six attributes: PatientID, survTime, status, rankScore, kmScore and normalizeScore. The values of the first three attributes are input from the CSV file while the values of the last three attributes are initialized as 0.0 when we create this object. For our implementation, all of the patients objects are stored in a vector.

After getting the input data, we calculate the survival probability for each patient in this cohort. This probability is the value of the survival function in the Kaplan-Meier curve. This curve gives an estimation of the survival function across time, which is the proportion of the patients that are still alive at a given time point. We then compare this Kaplan-Meier score between all the patient pairs and compute a GuanRank score for each patient. The detailed calculation process is illustrate in Figure 1.



S_A	S_B	Rank(A) +	Rank(B) +
1	1	1	0
1	0	1	0
0	1	p*	1 - p*
0	0	p*+(1-p*) x 0.5	(1-p*) x 0.5

$$p^* = (r(t_A)-r(t_B))/r(t_B)$$

S_A	S_B	Rank(A) +	Rank(B) +
1(0)	1(0)	0.5	0.5
1	0	1	0

Figure 1: GuanRank

After calculating the GuanRank score for each patient, we then normalize this score into between 0 and 1.

2.2 Speed Comparison

After we finished implementing GuanRank in C++, we next wanted to assess how computationally efficient it was by comparing it to an existing Python implementation. We began by simulating datasets of varying size (n = 500, 1000, 5000, 10000, 50000, 100000) using R to provide as input to both programs. For each dataset, the input variables were simulated in the following way:

- Patient ID - 1 ... n
- Time - Simulated from a Uniform(100, 1000)

- Status - Simulated from a Bernoulli(p = 0.6)

After simulating our data, we then ran both the C++ version and the Python version of GuanRank with each dataset of varying length and computed how long the program took to finish. To quantify the run-time of each program, We ran each program from the command line and used Unix's time command to obtain the user CPU time in seconds and minutes. However, it is important to note that our objective for the speed comparison was simply to get a rough idea of the run-time comparisons. Therefore, we decided to only run each version a single time for each sample size.

2.3 Creating GuanRank R Package

2.3.1 Interfacing C++ Code with R

The first step to build the R Package is to choose the segments of C++ code to be interfaced with R. For this portion, we used the extern "C" function to wrap our code. We create a function called "guanrank" with three arguments. The first is the input file's name while the second and third are vectors storing the patient ID and ranked normalized scores. Essentially, the "guanrank" function just replaces the "main function" in our typical C++ programs. Our C++ code performs all of the calculations in the guanrank function and then store our output in the previously mentioned vectors to easily pass as arguments to R.

2.3.2 The R Wrapper Function

In addition to editing our C++ code to include extern "C", we also needed to write an R script which contains the R wrapper function to call the C++ function.
All arguments used in this step keep consistent with those used in the C++ guanrank function. The input argument of the wrapper function is the file's name. After we use ".C" function to call the compiled C++ codes, the function returns a data frame which contains two columns of patient ID and ranked scores.

3 Results

3.1 Speed Comparison

The computational time for each version of GuanRank is shown below in table 1. We found our C++ implementation to be much faster than the Python implementation for each of our simulated datasets. This trend stays consistent as the sample size increases. In addition, we note that the Python version is vastly slower for larger sample sizes and fails to run for our largest sample size (n=100,000).

Table 1: Run-time between C++ and Python implementations for varying sample sizes

Sample Size	Python	C++
500	1.333s	0.011s
1000	2.026s	0.017s
5000	24.691s	0.342s
10000	1m 34.241s	1.339s
50000	40m 29.579s	32.705s
100000	N/A	2m 9.915s

3.2 GuanRank R Package

Here is an example of the usage of our R package. Our package takes as input a CSV file like Figure 2 below which contains three columns: Patient ID, time, and status.

After importing our GuanRank package in R, we can simply use the function "guanrank" and pass the path of our input file as the argument. Our function then returns a data frame containing patient ID and their GuanRank score.

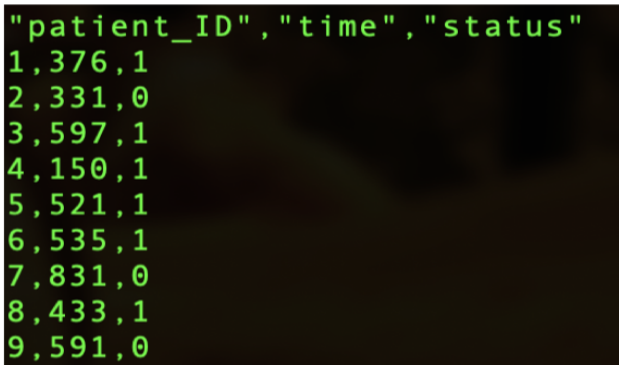


Figure 2: First 10 rows of the input CSV file

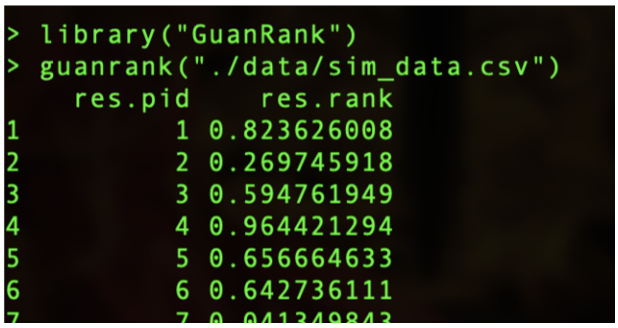


Figure 3: Usage of GuanRank R package

4 Discussion

The field of survival analysis has traditionally implemented parametric models such as the Cox proportional hazards model when analyzing time to event data. GuanRank is a novel non-parametric method that converts right-censored data into a linear space and produces complete hazard rankings for each patient. GuanRank was recently implemented into a Python version by Guan Lab and we found this implementation to be both memory inefficient and time inefficient. Therefore, we leveraged the computational efficiency of C++ to implement the GuanRank algorithm.

Because the actual algorithm is unchanged between the Python version and our C++ implementation, our work focused on improving memory and time efficiency. As seen in table 1, the Python version was comparable to our C++ version in simulated datasets for small sample sizes. However, as the sample sizes grew larger there was a clear divergence in computational speed between the two implementations. Of note is that the Python implementation struggled to accommodate our larger sample sizes of 50,000 and 100,000 whereas our C++ implementation finished within a few minutes. After looking at how GuanRank was implemented in the Python version, we noticed one major difference was how memory was managed. In the Python version, the entire input CSV file was read in to memory at once using Pandas. For larger datasets, this approach is generally not recommended because depending on the computer it may struggle to import the entire dataset. On the other hand, our C++ implementation benefited by parsing the CSV input file line by line to process the sample patient. By doing so, we allocated and managed our memory more efficient in our implementation. In addition to memory management, we expected our C++ implementation to be computationally quicker because of the differences in programming languages. Python is a high level interpreted programming language that is typically easier to code in. However, this ease of implementation comes with the drawback that it needs to be interpreted. On the other hand, C++ code is pre-compiled and inherently faster.

Overall, we were able to accurately implement GuanRank in C++ and build an R package using C++ code. We showed that our C++ version vastly outperformed an existing Python implementation in terms of both memory management and computational time. Our R package "GuanRank" is now publicly available and can be installed using install.packages("GuanRank). By compiling our optimized C++ code into an R package, researchers working in survival analysis can easily implement GuanRank in a common statistical language for their future analysis.

References

[1] D. R. Cox, "Regression models and life-tables," in Breakthroughs in statistics, pp. 527–541, Springer, 1992.

[2] Y. Guan, "Generalizing right-censored data into standard regression problem through complete ranking," 2017.