Summary/Guide

Rebecca Silva

Summer 2019

Dipsea Race Project

Our goal was to build a model to better optimize the head starts in the Dipsea Race so that runners of any age had an equal chance of winning. In the past decade, the winners of the Dipsea Race have been in or around their 60s, and we want a model that makes it possible for runners with no head start to catch up. More concretely, we want to find a 95% confidence interval for the factors at each age, sex, and section. A factor is the ratio between the time at one age to the time at a 'base' age. We define the base ages as 27, 30, and 35 years old; however, my work in this study focuses on factors with base age 30 for the purpose of efficiency. Finding the other factors can be easily outlined and is commented in the code.

Data

We start by reading in the data from all past races (since 1996). The 'Data Reading' folder contains the code for reading in all the data from the website: https://www.dipsea.org/prevresults.php. I first copy and pasted the race results 'by order of finish' into excel and then used the 'readxl' package to read in the data. 'Data_Final.Rmd' contains code to read in and tidy the data, creating the final data set, 'ds', which I use to construct any other data set later in the study. The key variables we use are Sex ("M", "W"), Section ("Invitational", "Runner"), Age (6-80yrs), and Actual_Time (time without head start, mins).

Other important data sets that are used are 'ds_NN' and 'scaled', both created in the 'NN_Model.Rmd' within the 'Modeling' folder.

- 'ds_NN': From 'ds' it selects only the key variables and changes Sex and Section to binary variables.
- 'scaled': From ds_NN, this data set is the data normalization of ds_NN. Each column has values between 0 and 1. I decided to scale instead of normalize after doing some univariate analysis of the continuous variables Age and Actual_Time.

Modeling

My work exploring Generalized Additive Models (GAM) and Neural Networks can be found in the folder 'Modeling'.

General Additive Models (GAM)

The GAM model is an extension of the GLM model. We chose to use a GAM model over a polynomial regression model because gams can better extrapolate, be more flexible with curves as we do not have to specify a functional form, and are less prone to overfitting. All with with gam modeling and adding a model to a bootstrap function can be found in the 'GAM' folder within the 'Modeling' folder.

For gam models we use the 'mgcv' package. The main model we are working with is $ Actual_Time ~ s(Age) + Sex + Section + s(Age, by = as.factor(Sex) + s(Age, by = as. factor(Section)$. My work with gam models and understanding splines and degrees of freedom is in GAM_Explore.Rmd (in the 'Modeling' folder). We can determine the smoothness through adjusting k, the basis dimension, which sets the maximum limit for degrees of freedom. The effective degrees of freedom (edf) is determined by the method for fitting used (GCV, AIC, or REML etc). I used the REML method since it is the default and was recommended by some sights

since REML penalizes over fitting more than GCV. By hand, I adjusted for k by checking if the edf was close to k-1 and if so, I would make k larger to assure the degrees of freedom is large enough. The function gam.check() will give the edf used for each input variable.

We have had issues of overfitting using the gam model. As you will find in 'GAM_Explore.Rmd' even when I keep the basis dimension (k) set to 10 or lower, training 2 to 3 models with the data set 'ds' gives the same prediction with no variation. Although our initial idea was to bootstrap the whole data set and predict on a test set that consisted of every possible combination of age, sex, (and section), I have found that the only way to get enough variation with gam() is to bootstrap a sample of the data each time. As you will find in GAM_Boot.rmd, I sample 85% of the data each time; this percentage can be adjusted as desired.

Gam_Boot and Gam_SexOnly are similar R documents that contain the 'final' code to output a dataframe with a 95% confidence interval for factors at each age, sex, and section and age and sex, respectively. Once one feels confident about the model they are training, the number of bootstraps (B) should be increased to 1,000. Both documents have detailed commenting.

Neural Network Model (NN)

Neural network models train by using 'hidden layers' to generate weights between the input and output layers. The input layers for this model would be Age, Sex, and Section and the output layer is Actual_Time. The hidden layers choose weights of input variables through back propagation. I have read that one hidden layer is usually good enough. The typical rule is that the number of hidden layers should be between the input layer size and output layer, or 2/3 the input layer.
The first step in Neural network models is to normalize the data. I used the scale() function and lapply() to go through the data and scale each column, using $center = min$ and $scale = max \breve{} min$. I worked with the packages neuralnet and nnet to build neural network models. The pros of neuralnet() are that we are able to specify arguments of the model such as number of hidden layers (unlike caret::train). The major con is that it have yet to see if predict times well and takes a while to train 1 model. I can take over an hour to train a model with neuralnet, and the error turns out pretty large. When inputs are just Age and Sex, the hidden layer is set to 1, but with Age, Sex, and Section, I try 2 hidden layers.

I used the train function in the caret package and nnet package to train a neural network with nnet(). In the Caret_Train.rmd I explore using train() with gam and neural network models. I can use train to find the 'best' model by a certain metric and then use nnet() to get train one model with that specific metric. However, I found that just using nnet() to get a model did not work. My predictions came out as zeroes. The other option is that I can use train() within the bootstrap method so that each time it trains a model, it goes through many possible combinations of decay and size. The issue we are finding with train(), is that the model is not accurate. For example, the factor for a 65 year-old, male, invitational runner, should be around 1.28 but instead we get around 1.4. I was not able to build a neural network model that predicted time well; they are seem biased and overestimate Actual_Time. The neural network model is a work in progress.

NNet_Boot and NNet_SexOnly are similar to the Gam_Boot and Gam_SexOnly R documents that contain the 'final' code to output a dataframe with a 95% confidence interval for factors at each age, sex, and section and age and sex, respectively. Once one feels confident about the model they are training, the number of bootstraps (B) should be increased. Both documents have detailed commenting.

US Masters Swimming Data (Top 10)

My work on the project is in the MastersSwimming proj shared on github. The document 'AllTabls.Rmd' contains all my used code to obtain all the data from the website: https://www.usms.org/comp/tt/toptenlist.php. I used the rvest package to scrape the contents. In Google Doc folder shared with you I have uploaded the 3 final data sets: 'USMS_Top10_SCY.csv' (Short Course Yards), 'USMS_Top10_LCM.csv' (Long Course Meters), 'USMS_Top10_SCM.csv' (Short Course Meters).