Non-Linear Parametric Modeling in R using Genetic Algorithm

In solving a business problem, depending on the context of the problem, we may or may not be required to explain the predictions from a statistical model. When we need to understand and explain the predictions, we tend to use parametric or equation based models. When accuracy of prediction is all that matters, we prefer machine learning based models as these algorithms tend to have higher accuracy than parametric models for a given amount of invested effort. There are other advantages to using machine learning based models like ability to handle high dimensionality (Random Forest vs Linear Regression). With recent advancements through "interpretable machine learning" effort, we have also got partial dependency plots and shapely plots that mitigate "black-box" drawback of machine learning.

Inspite of such advantages to machine learning based models, when it comes to explainability of predictions, nothing works as good as an equation based model. Businesses tend to believe and buy into ideas that they can understand, interpret and fiddle with to see if its in line with what is expected. Apart from this, speaking from personal experience, equation based models are more reliable when they have to predict using data outside of training data range, and they react in a anticipated fashion in such scenarios. As such, we feel more confident about equation based models. But, "no free lunch theorem" (https://en.wikipedia.org/wiki/No_free_lunch_theorem) might resonate with many of us and an individual's call ("artistic bend") to choose between either of the approaches (parametric or non-parametric) should be respected.

If you have decided to use the parametric or equation based approach, and the equation being fit to the data is not linear in nature, this article might be of help to you. Through this post, we will see how we can use Genetic Algorithm to fit non-linear equations to a dataset.

In our early career in data science, most of us must have seen and used linear regression. While practicing with linear regression, if an explanatory variable would have a non-linear trend with the dependent variable, we would try out different transformations (log(x), 1/x, x^2, etc.) to straighten out the non-linearity and use the transformed explanatory variable in modeling the dependent variable. In fitting many non-linear equations, this workaround can still be helpful. For example, if we are trying to fit y = a*e^bx (here a and b are being solved for), we can take log on both the sides and model the value of log(y) = log(a) + bx. In this, log(y) can treated as y1, log(a) can be treated as x1 and equation becomes y1 = x1 + bx. Using linear regression this can be solved and transformations can be reversed for solving a and b to get back original equation.

There are non-linear equations where we need to solve the them in their original form due to their complexity. There are also advantages like avoiding local minima that are part of the algorithms that we are going to discuss, which could offer a motivation to solve the equations in their original form. In this article, we will see how we can solve such equations using Genetic Algorithm.

We have heard that human population has increased exponentially over the last many centuries. Although recent research (https://blog.ucsusa.org/doug-boucher/world-population-growth-exponential) suggests that it has started to become linear in last 50 odd years, for our problem we will consider it to be exponential and fit an exponential equation to the world population. The equation would be:

P = Po*e^(rt)

where: P = world population at time t, Po = world population at initial time 1515 AD, t = time in years after initial year, r = rate of growth of population

We will be solving for "r" through GA

Genetic Algorithm

Genetic algorithm belongs to the class of evolutionary computing algorithms. This algorithms mimics the process of evolution to perform optimization - creates a population of initial solutions, applies mutation, crossover, selection and other evolutionary functions to take out best solutions, then repeats the steps over generations/iterations till all the iterations are exhausted or no significant improvement is noticed. There are many good resources online to learn GA and interesting applications people have used it in. In its basic form, GA is an optimization algorithm and can be used in solving equations, traveling salesman problem, etc. What I appreciate about GA is that there are so many parameters in it that you can manage and specify, all these helpful in avoiding local minima and finding a good solution.

We start with reading in the required libraries and dataset. Dataset has been downloaded from https://ourworldindata.org/world-population-growth.

```r
libs <- c('GA', 'dplyr', 'ggplot2','doParallel' )
lapply(libs, require, character.only = T)
```

```
## Loading required package: GA

## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.2
## Type 'citation("GA")' for citing this R package in publications.

##
## Attaching package: 'GA'

## The following object is masked from 'package:utils':
##
##     de

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Loading required package: doParallel

## Loading required package: parallel

## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] TRUE
```

```
df <- read.csv("A:/Projects/Non-Linear Parametric Modelling/WorldPopulationAnnual12000years_interpolate
head(df)
```

```
##     year World.Population..Spline.Interpolation.until.1950.
## 1 -10000                                            2431214
## 2  -9999                                            2432196
## 3  -9998                                            2433179
## 4  -9997                                            2434162
## 5  -9996                                            2435145
## 6  -9995                                            2436129
```

As we can see from the second column's name, a spline interpolation technique has been used to generate world population for years before 1950. I am not too sure about the numbers way too much into the past. For solving our equation we will use, only last one hundred years of data, i.e, between 1915 and 2015. We then We set the year 1515 to be 0 (i.e. To) and each year as numerical distance from this year. We also rename the columns to match the equation we have listed

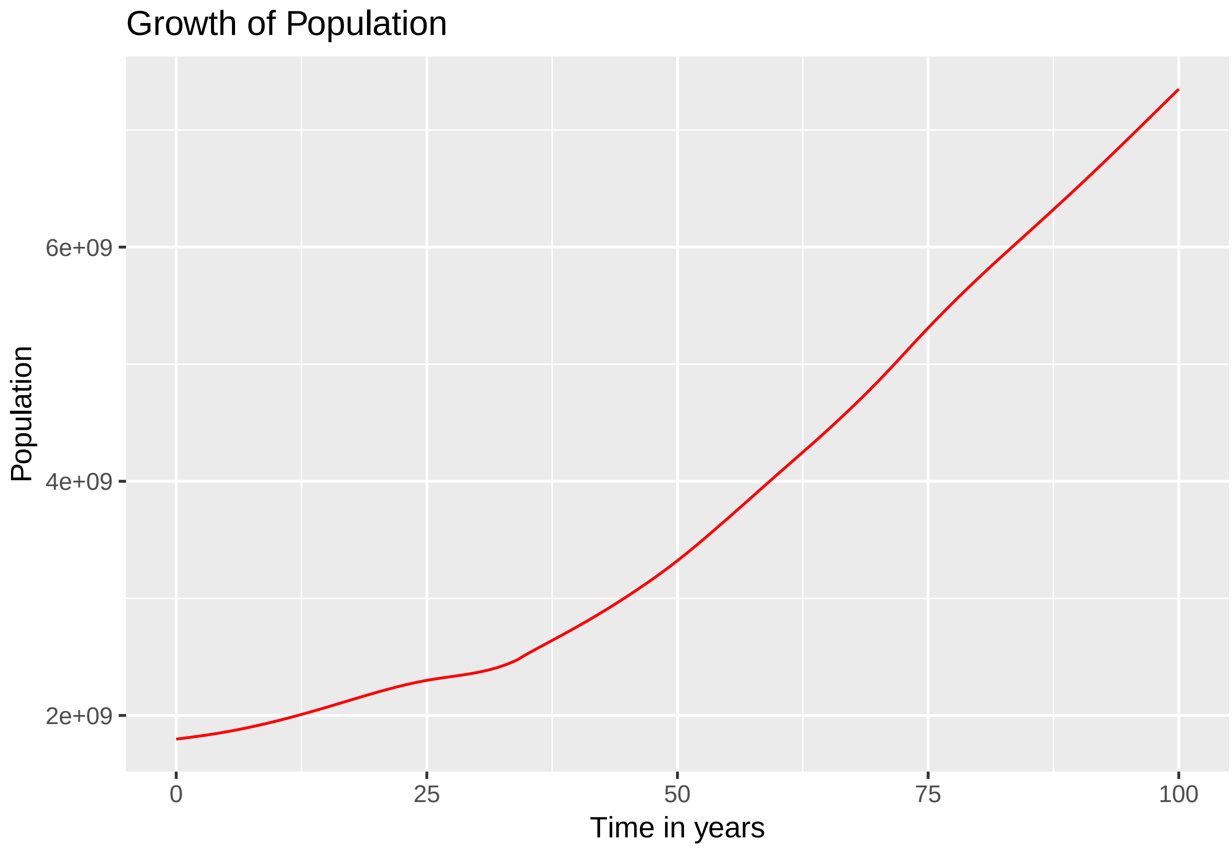
```
df <- df[(nrow(df)- 100):nrow(df),]
df$year <- df$year - min(df$year)
colnames(df)[1] <- "t"
colnames(df)[2] <- "P"
```

We now assign Po, which is the population in the first year of the dataset

```
Po <- df$P[df$t == 0]
```

Let's visualize how does the population growth looks like:

```
ggplot2::ggplot(df, aes(t, P)) + geom_line(color = "red") + xlab("Time in years") + ylab("Population")
```



Well, the trend does appear to be exponential in the first half but seems to become linear in the second half. For our demonstration of demonstration of GA, we will continue to consider the curve to be exponential

There are four esstential steps in the process: 1. Declare the name of the variable being solved for. This step ensures readability of the output from GA when many variables are being solved for. 2. Defining the upper limit and lower limit of the search space in which solution of the variables will be looked for 3. Defining the fitness function, using which GA will evaluate the quality of the solution 4. Executing the GA iteration with all the parameters specified. This step is where we define how the GA will evolve solutions over generations

We perform steps 1 to 3 here:

```
gaNames <- "rate of growth of population" #declaring the name of the variable we are solving for

# Defining the fitness function, i.e., a function which will evaluate how close the actual values are t
fitnessFunction <- function(P, Po, r, t){
  Pfit <- Po*exp(r*t)
  error <- (P - Pfit)^2
  fitness <- -sum(error) # GA is a maximization function, as we want to reduce the error, we introduce
}

# We declate the search space, i.e. the space in which the values of r can be found. As we know that po
upperLimitOfSearchSpace = 0.05 # These limits should be a vector of same length as gaNames and in same
lowerLimitOfSearchSpace = 0.0001
```

Next we initiate the GA to search for the solution:

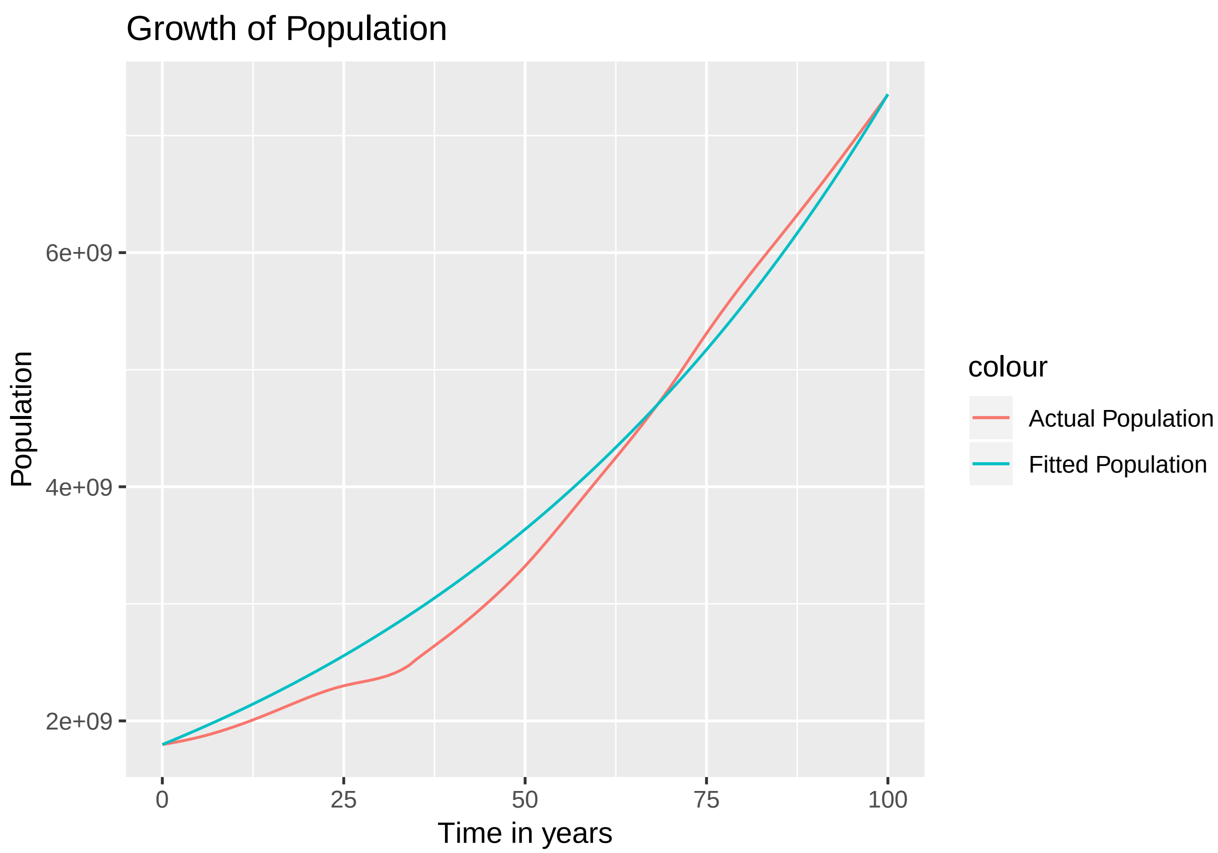
```r
rGA <- ga( #the function is called as ga
  type = "real-valued", # solving for a real number, not rank or combination
  fitness = fitnessFunction, # passing the fitness function to evaluate quality of solution
  P = df$P, # We pass the variables needed to estimate the fitness function
  Po = df$P[df$t == 0],
  t = df$t,
  lower = lowerLimitOfSearchSpace,
  upper = upperLimitOfSearchSpace,
  pcrossover = 0.9, #crossover, mutation, etc. these are the paramterers that lend GA it's advantage of
  pmutation = 0.10,
  elitism = 10,
  popSize = 1000,
  population = "gareal_Population",
  selection = "gareal_nlrSelection",
  crossover = "gareal_laCrossover",
  mutation = "gareal_nraMutation",
  # suggestions = matrix(guess,nrow=sample(1:pop, size=1),ncol=length(guess),byrow=TRUE),
  maxiter = 100,
  run = 40,
  parallel = TRUE,
  # monitor = monitor,
  names = gaNames
)

cat("\nBest value of rate of growth of population:",rGA@solution,"\n")
```

```
##
## Best value of rate of growth of population: 0.01408498
```

Let's plot the population fit line

```r
df$pFit <- df$P[df$t == 0]*exp(as.numeric(rGA@solution)*df$t)
ggplot(df, aes(t)) +
  geom_line(aes(y = P, colour = "Actual Population")) +
  geom_line(aes(y = pFit, colour = "Fitted Population")) +
  xlab("Time in years") + ylab("Population") + ggtitle("Growth of Population")
```


Growth of Population
Population
6e+09
4e+09
2e+09
0
25
50
75
100
Time in years
colour
Actual Population
Fitted Population