Genetic Algorithm

Jared Yu

April 20, 2018

```
truefunction<-function(x){ # default
  t <- c(0.1, 0.13, 0.15, 0.23, 0.25, 0.4, 0.44, 0.65, 0.76, 0.78, 0.81)
  h <- c(4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2)
  temp <- 0
  for(i in 1:11) {
    temp <- temp + h[i]/2 * (1 + sign(x - t[i]))
  }
  return(temp)
}
n<-512
x<-(0:(n-1))/n
f<-truefunction(x)
set.seed(0401)
y<-f+rnorm(f)/3
plot(x,y)
lines(x,f)

noisydata = data.frame(x=x,y=y) # data frame made of previous x,y data
rchrom = function(s, rawdata = noisydata){ # random chromosomes of size S
  nobs = nrow(rawdata) # number of observations

  # create a random matrix of 1's and 0's of size nobs*s
  # each row of S1, S2,...,,Sn represents 1 chromosome of length nobs
  return(matrix(sample(x=c(0,1), size=nobs*s, replace = TRUE), ncol = nobs))
}

# determine the number of bins by removing the first, last, summing and adding 1
#Bhat = sum(ch1[c(-1, -length(ch1))]) + 1 # B_hat

nhat = function(chromes) { # n hat
  # change first index to 1, last index to 0
  dummych = chromes # dummy 'variable' for chromosomes
  dummych[1] = 1 # set first index to 1
  dummych[length(dummych)] = 0 # set last index to 0

  # shift index to the left
  pos1 = c(1:length(chromes))[dummych==1] # find indexes of 1's (starting index of eac
h bin)
  dummych = c(dummych[-1],1) # shift left, add a 1 to the right
  pos2 = c(1:length(chromes))[dummych==1] # find '1' index of new vector (ending inde
x of each)
  return((pos2 - pos1) + 1) # number of observations in each bin
}

fhat = function(chromes, rawdata = noisydata) { # f hat
  # same as n hat function
  dummych = chromes
  dummych[1] = 1
  dummych[length(dummych)] = 0
  pos1 = c(1:length(chromes))[dummych==1]
  dummych = c(dummych[-1],1)
  pos2 = c(1:length(chromes))[dummych==1]

  # creates a list y values in each bin
  pieces = sapply(1:length(pos1),
                  function(x) rawdata$y[pos1[x]:pos2[x]])
  return(unlist(lapply(pieces, mean)))
}

mdl = function(ch, rawdata = noisydata){ # MDL
  Bhat = sum(ch[c(-1, -length(ch))]) + 1 # B hat
  nhat = nhat(ch) # n hat
  fhat = fhat(ch) # f hat
  n = nrow(rawdata) # number of observations
  fhatvec = rep(fhat, nhat) # distance between data and f_jhat function
  return(Bhat*log(n) + 0.5*sum(log(nhat)) + (n/2)*log((1/n)*sum((noisydata$y-fhatvec)^
2)))
}

aic = function(ch, rawdata = noisydata) { # AIC
  Bhat = sum(ch[c(-1, -length(ch))]) + 1 # B hat
  nhat = nhat(ch) # n hat
  fhat = fhat(ch) # f hat
  n = nrow(rawdata) # number of observations
  fhatvec = rep(fhat, nhat) # distance between data and f_jhat function
  return(n*log((1/n)*sum((noisydata$y-fhatvec)^2)) + log(n)*2*Bhat)
}

# Step 5
# generate one child
generation = function(method,
                      Pcross = 0.9,
                      Pc = 0.05,
                      ranchrom = ranchrom,
                      rawdata = noisydata) {
  # fitness steps (2,3)
  # Step 2
  if (method == "MDL") {
    # for MDL method
    mdl_val = apply(ranchrom, 1, function(x)
      mdl(x, rawdata)) # find the MDL for each chromosome generated

    # Step 3
    mld_val_rank = rank(mdl_val * -1) # sorts the chromosomes according to greatest MD
L

    if (runif(1) < Pcross) {
      # Step 4
      # create father/mother chromosomes
      parent_ind = sample(
        1:nrow(ranchrom),
        size = 2,
        prob = mld_val_rank / sum(mld_val_rank),
        replace = TRUE
      )
      father = ranchrom[parent_ind[1],]
      mother = ranchrom[parent_ind[2],]

      # crossover
      # father/mother should flip or stay the same
      child = mother # create the child vector from the mother
      child_index = sample(c(0, 1), size = length(father), replace = TRUE) # child ind
ex for switching between parents
      child[child_index == 1] = father[child_index == 1] # substitute indices from fat
her to child
    } else {
      # mutation
      # sample one parent from S chromosomes according to fitness
      parent_ind = sample(
        1:nrow(ranchrom),
        size = 1,
        prob = mld_val_rank / sum(mld_val_rank),
        replace = TRUE
      )
      child = ranchrom[parent_ind,] # create child from one of S chromosomes
      Pc = 0.05 # Probability of change
      # generate a vector with probability 1-Pc that an index is 0, and probability P
c that the index is 1
      mutate_ind  = sample(
        c(0, 1),
        size = length(child),
        prob = c(1 - Pc, Pc),
        replace = TRUE
      )

      # bit inversion
      child[mutate_ind == 1] = abs(child[mutate_ind == 1] - 1) # mutate the index of t
he child according to mutate index
    }
    return(child)
  } else {
    # when AIC is used
    # Step 5
    # generate one child
    # fitness steps (2,3)
    # Step 2
```

```r
    aic_val = apply(ranchrom, 1, function(x)
      aic(x, rawdata)) # find the MDL for each chromosome generated

    # Step 3
    aic_val_rank = rank(aic_val * -1) # sorts the chromosomes according to greatest MD
L

    if (runif(1) < Pcross) {
      # Step 4
      # create father/mother chromosomes
      parent_ind = sample(
        1:nrow(ranchrom),
        size = 2,
        prob = aic_val_rank / sum(aic_val_rank),
        replace = TRUE
      )
      father = ranchrom[parent_ind[1],]
      mother = ranchrom[parent_ind[2],]

      # crossover
      # father/mother should flip or stay the same
      child = mother # create the child vector from the mother
      child_index = sample(c(0, 1), size = length(father), replace = TRUE) # child ind
ex for switching between parents
      child[child_index == 1] = father[child_index == 1] # substitute indices from fat
her to child
    } else {
      # mutation
      # sample one parent from S chromosomes according to fitness
      parent_ind = sample(
        1:nrow(ranchrom),
        size = 1,
        prob = aic_val_rank / sum(aic_val_rank),
        replace = TRUE
      )
      child = ranchrom[parent_ind,] # create child from one of S chromosomes
      Pc = 0.05 # Probability of change
      # generate a vector with probability 1-Pc that an index is 0, and probability P
c that the index is 1
      mutate_ind  = sample(
        c(0, 1),
        size = length(child),
        prob = c(1 - Pc, Pc),
        replace = TRUE
      )

      # bit inversion
      child[mutate_ind == 1] = abs(child[mutate_ind == 1] - 1) # mutate the index of t
he child according to mutate index
```

```r
    }
    return(child)
  }
}

best_chrom = function(ranchrom, rawdata = noisydata, method = "MDL"){ # chromosome fit
ness
  if (method == "MDL"){
    mdl_val = apply(ranchrom, 1, function(x) mdl(x, rawdata))
    return(ranchrom[which.min(mdl_val),])
  } else if (method == "AIC") {
    aic_val = apply(ranchrom, 1, function(x) aic(x, rawdata))
    return(ranchrom[which.min(aic_val),])
  } else {
    return("Either MDL or AIC method allowed")
  }
}

GA = function(rawdata = noisydata, method = "MDL", S = 300, Nsame = 20, Pcross = 0.9,
Pc = 0.05){ # genetic algorithm
  # generate population
  ranchrom = rchrom(S, rawdata)

  # identify best chromosome according to MDL/AIC method
  current_best = best_chrom(ranchrom, rawdata = rawdata, method = method) # determine
the best chromosome per generation

  # create successive generations until Nsame condition reached
  same = 0
  while (same < Nsame) {
    # generate S children for the next genepool population
    next_ranchrom = t(sapply(1:nrow(ranchrom), function(x) generation(method = metho
d, Pcross = Pcross, Pc = Pc, ranchrom = ranchrom, rawdata = rawdata) ))

    # select the most fit chromosome according to MDL or AIC
    next_best = best_chrom(next_ranchrom, rawdata = rawdata, method = method)

    # check if new most fit chromosome is same as previous most fit chromosome
    if (method == "MDL") {
      current_score = mdl(ch = current_best, rawdata = rawdata)
      next_score = mdl(ch = next_best, rawdata = rawdata)
    } else {
      current_score = aic(ch = current_best, rawdata = rawdata)
      next_score = aic(ch = next_best, rawdata = rawdata)
    }

    print(abs(current_score - next_score))
    if (current_score == next_score) {
      # next_best and current_best are the same
      same = same + 1
    } else {
      same = 0
    }
    current_best = next_best
    ranchrom = next_ranchrom
  }

  # plot the best chromm result on a graph

  truefunction<-function(x){
    t <- c(0.1, 0.13, 0.15, 0.23, 0.25, 0.4, 0.44, 0.65, 0.76, 0.78, 0.81)
    h <- c(4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2)
    temp <- 0
    for(i in 1:11) {
      temp <- temp + h[i]/2 * (1 + sign(x - t[i]))
    }
    return(temp)
  }

  n<-512
  x<-(0:(n-1))/n
  f<-truefunction(x)
  set.seed(0401)
  y<-f+rnorm(f)/3
  plot(x,y)
  lines(x,f)

  Bhat = sum(current_best[c(-1, -length(current_best))]) + 1 # B hat
  nhat = nhat(current_best) # n hat
  fhat = fhat(current_best) # f hat
  n = nrow(rawdata) # number of observations
  fhatvec = rep(fhat, nhat) # distance between data and f_jhat function
  lines(x,fhatvec, col="red")

  return(current_best)
}

# run
set.seed(243)
best_AIC = GA(rawdata = noisydata, method = "AIC", S = 300, Nsame = 20)
write.csv(t(t(best_AIC)), "best_AIC.csv", row.names = FALSE)
best_MDL = GA(rawdata = noisydata, method = "MDL", S = 300, Nsame = 20)
write.csv(t(t(best_MDL)), "best_MDL.csv", row.names = FALSE)
```