Psychometric and Machine Learning Approaches to Reduce the Length of Scales

Supplementary Materials

Empirical Example

Below is R code to carry out several psychometric and machine learning approaches to reduce the length of scales. The empirical example for this illustration is a curated dataset with 20 items from the Short Self-regulation Questionaire (SSRQ), described in the accompanying manuscript. Below are the item stems from the measure:

- SSRQ01. I don't notice the effects of my actions until it's too late.
- SSRQ02. I put off making decisions.
- SSRQ03. It's hard for me to notice when I've 'had enough' (alcohol, food, sweets).
- SSRQ04. I have trouble following through with things once I've made up my mind to do something.
- SSRQ05. I don't seem to learn from my mistakes.
- SSRQ06. I usually only have to make a mistake one time in order to learn from it.
- SSRQ07. I can usually find several different possibilities when I want to change something.
- SSRQ08. Often I don't notice what I'm doing until someone calls it to my attention.
- SSRQ09. I usually think before I act.
- SSRQ10. I learn from my mistakes.
- SSRQ11. I give up quickly.
- SSRQ12. I usually keep track of my progress toward my goals.
- SSRQ13. I am able to accomplish goals for myself.
- SSRQ14. I have personal standards, and try to live up to them.
- SSRQ15. As soon as I see a problem or challenge, I start looking for possible solutions.
- SSRQ16. I have a hard time setting goals for myself.
- SSRQ17. When I'm trying to change something, I pay a lot of attention to how I'm doing.
- SSRQ18. I have trouble making plans to help me reach my goals.
- SSRQ19. I set goals for myself and keep track of my progress.
- SSRQ20. If I make a resolution to change something, I pay a lot of attention to how I'm doing.

- SSRQ21. I know how I want to be.
- SSRQ22. I have trouble making up my mind about things.
- SSRQ23. When it comes to deciding about a change, I feel overwhelmed by the choices.
- SSRQ24. Most of the time I don't pay attention to what I'm doing.
- SSRQ25. Once I have a goal, I can usually plan how to reach it.
- SSRQ26. If I wanted to change, I am confident that I could do it.
- SSRQ27. I can stick to a plan that's working well.
- SSRQ28. I have a lot of willpower.
- SSRQ29. I am able to resist temptation.

Suppose that we need to select less items to administer during an intervention study. Researchers could take several approaches to select items. They could create a short-form with items that everybody would take, or they administer the items as in a tailored test, conditional on participants' previous responses. Both approaches could be carried out with a focus on estimating precise scores or with a focus on predicting the score on the overall measure. Here, the methods illustrated are IRT (item response theory) to build static short-forms, computerized adaptive testing (CAT), the genetic algorithm to build static short-forms, and regression trees for tailored testing. The dataset consists of 522 Mturk participants who completed the questionnaire, divided in a training dataset (N=400) and a testing dataset (N=122).

Before starting, let's load the training and the testing datasets:

```
#setwd("../datasets")
trdata1= read.csv('psyml_trdata.csv')
tedata1= read.csv('psyml_tedata.csv')
```

The procedures are largely carried out in two steps. For the psychometric models, the training dataset is used to estimate the item parameters for an IRT model, and the testing dataset is used to estimate the scores. For the machine learning models, the training dataset is used to estimate the models, and the testing dataset is used to evaluate the predictions of the models.

Static Short-forms using IRT

Largely, item response theory can build a short-form by manually choosing items that maximize item information across the range of the latent variable measured. Also, researchers could have the option of trading some items that maximize information for some items that might be relevant for the definition of the construct they intend to measure.

Training dataset

First, the required packages are loaded

```
library(mirt)  #load mirt package
```

```
## Loading required package: stats4
```

```
## Loading required package: lattice
```

```
library(plyr) #load package plyr
```

Then, the IRT graded response model is fitted and the item parameters are printed:

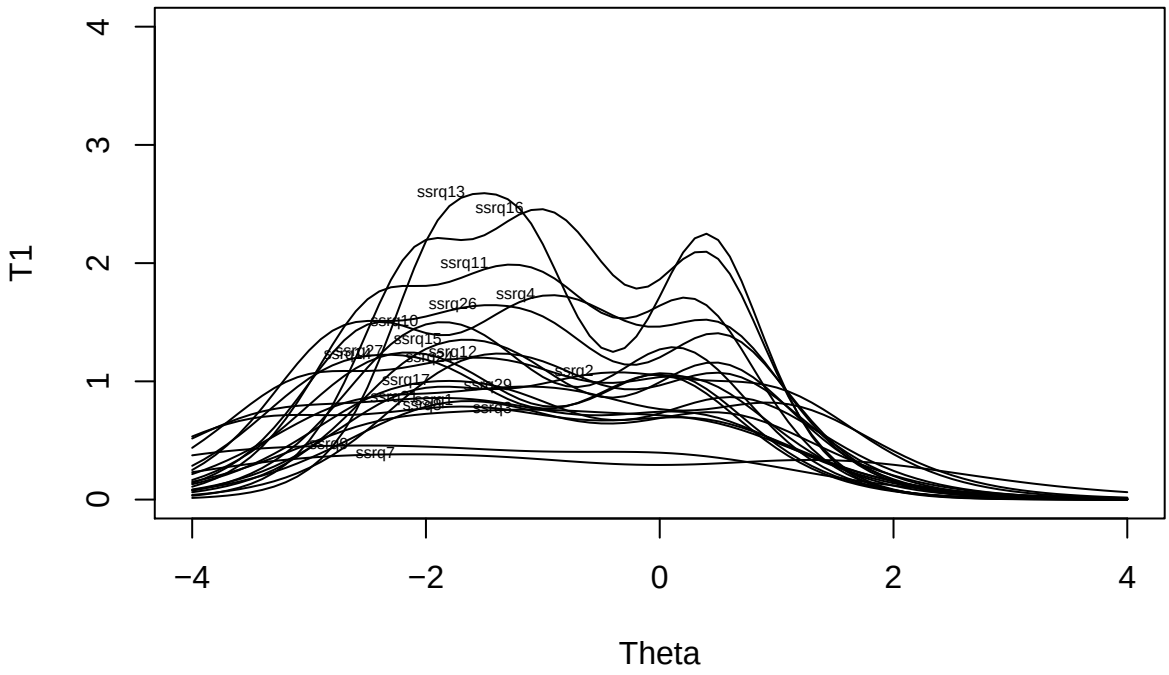
```
n2=mirt(trdata1,1,verbose=FALSE,quietly=TRUE)  #fit mirt model
n3=coef(n2,IRTpars=T)
n3[[21]]<-NULL
n3=lapply(n3,data.frame)
n4<-do.call(rbind.fill,n3)
n4 #item parameters
```

```
##           a        b1         b2          b3         b4
## 1  1.689406 -1.929322 -1.4445765  0.53401209         NA
## 2  1.885151 -2.046266 -0.6285995 -0.22360239 0.95704794
## 3  1.574638 -2.303865 -1.1838492 -0.91933290 0.33007132
## 4  2.392420 -2.469682 -1.1975098 -0.67541669 0.50236444
## 5  1.137118 -2.934825 -1.4886487  1.43070906         NA
## 6  1.617344 -3.416979 -1.8936208 -1.40378762 0.33412711
## 7  1.212198 -3.662102 -2.4688234 -1.91724269 0.08006628
## 8  2.248814 -2.143726 -1.6018185  0.15541595         NA
## 9  2.560639 -2.413066 -1.4773376 -0.95545673 0.29459428
## 10 2.035134 -1.723528 -1.0825057  0.56128053         NA
## 11 2.986951 -1.835224 -1.2020374  0.40422233         NA
## 12 2.037458 -2.761643 -1.9473755  0.01468906         NA
## 13 2.132896 -2.000093 -1.3423376  0.50003164         NA
## 14 2.852665 -2.066558 -1.1790229 -0.79777773 0.40851947
## 15 1.841015 -2.271759 -1.3993085  0.65637597         NA
## 16 1.675033 -3.311051 -2.0883189 -1.42995154 0.26639261
## 17 1.987086 -3.028240 -1.8192829 -1.20466566 0.26317984
## 18 2.341211 -2.723297 -1.7035714 -1.02167533 0.52806666
## 19 2.048237 -2.461712 -1.8789868  0.06058681         NA
## 20 1.769711 -2.574044 -1.3633401 -0.69957169 1.04373961
```

Next, item information is extracted and functions are plotted:

```
Theta=matrix(seq(-4,4,by=.1))  #Theta range
T1=iteminfo(extract.item(n2,1),Theta) #extract information
plot(Theta,T1,type='l',ylim=c(0,4)) #plot first infomration function
l1=which(T1==max(T1))
text(x=mean(Theta[l1]), y=max(T1), pos=2, labels=paste0('ssrq',1),cex=.5)

#plot rest of information functions in the same plot as above
for(i in 2:20){
  T2=iteminfo(extract.item(n2,i),Theta)
  l1=which(T2==max(T2))
  lines(Theta,T2)
  text(x=mean(Theta[l1]),y=max(T2),pos=2,labels = names(trdata1)[i],cex=.5)
}
```



It is difficult to see in the plot, but six items that both maximize item information (tallest functions) and that extend to lower and higher ranges of θ are items 2, 11, 13, 15, 16, and 29. Also, item 3 and item 9 are included because they might be important for the definition of the self-regulation construct. Those items correspond to the 2nd, 3rd, 7th, 9th, 11th, 13th, 14th, and 20th columns in the training dataset. Below are the items for the short-form:

- SSRQ02. I put off making decisions.
- SSRQ03. It's hard for me to notice when I've 'had enough' (alcohol, food, sweets).
- SSRQ09. I usually think before I act.
- SSRQ11. I give up quickly.
- SSRQ13. I am able to accomplish goals for myself.
- SSRQ15. As soon as I see a problem or challenge, I start looking for possible solutions.
- SSRQ16. I have a hard time setting goals for myself.
- SSRQ29. I am able to resist temptation.

Next, we will add the item information functions for the original item set and for the item set we have just selected to define their corresponding test information function, and then those would be plotted.

```
#Extract information for TIF of original item set
tif1=list(NULL)
for(i in 1:20){
  tif1[[i]]=iteminfo(extract.item(n2,i),Theta)
}

tif2=do.call(rbind,tif1)
tif3=colSums(tif2)
```

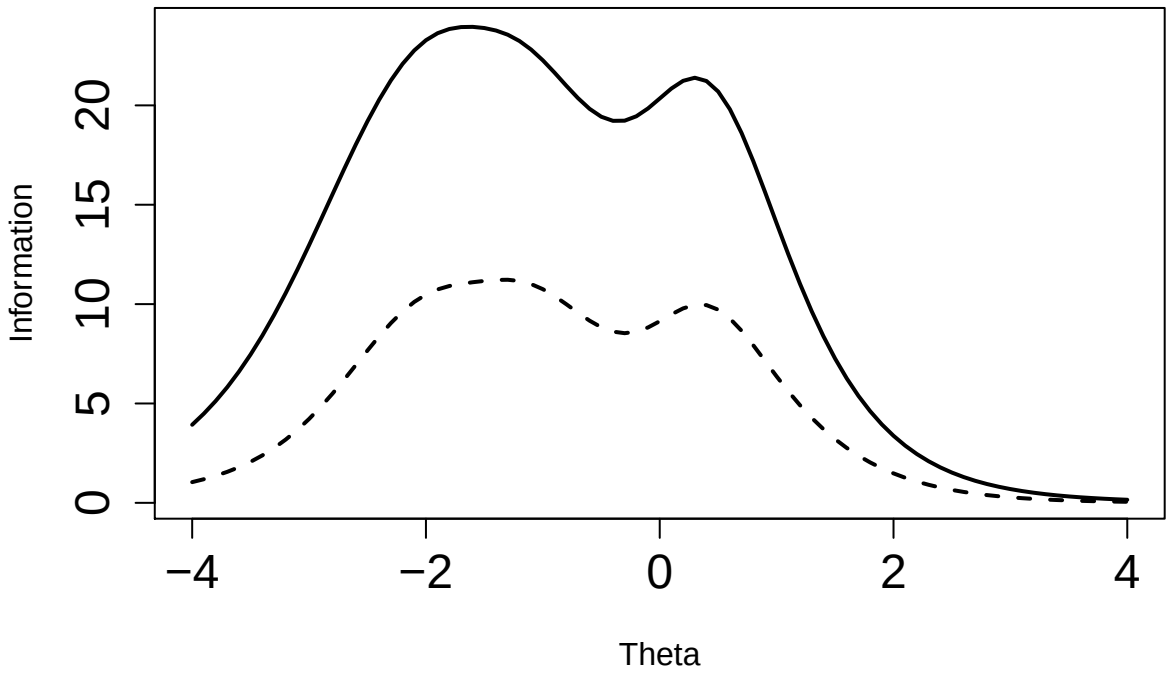
```r
#Extract information for TIF of reduced item set
## Items selected: 2,3,7,9,11,13,14,20

li=list(NULL)
for(i in c(2,3,7,9,11,13,14,20)){
  T2=iteminfo(extract.item(n2,i),Theta)
  li[[i]]=T2
}

li2=do.call(rbind,li)
li3=colSums(li2)

#Plot TIF's of original item set and short-form in the same plot

plot(Theta,tif3,type='l',ylab='Information',lwd=2,cex.axis=1.5)
lines(Theta,li3,type='l',lty=2,lwd=2)
```



Finally, we need to build a custom scoring model to estimate IRT scores only from the reduced set of items. This scoring model would allow us to estimate scores on the short-form from the response patterns in the testing dataset.

```r
#Building a custom scoring mirt object

tr1=mirt(trdata1[,c(2,3,7,9,11,13,14,20)],1,pars='values')

n5=n4[c(2,3,7,9,11,13,14,20),] #item parameters

tr1$value[tr1$name == 'a1'] <- n5[,1]
tr1$value[tr1$name == 'd1'] <- n5[,2]*-n5[,1]
tr1$value[tr1$name == 'd2'] <- n5[,3]*-n5[,1]
tr1$value[tr1$name == 'd3'] <- n5[,4]*-n5[,1]
tr1$value[tr1$name == 'd4'] <- n5[c(1:4,7:8),5]*-n5[c(1:4,7:8),1]
```

```
tr1$est=FALSE
```

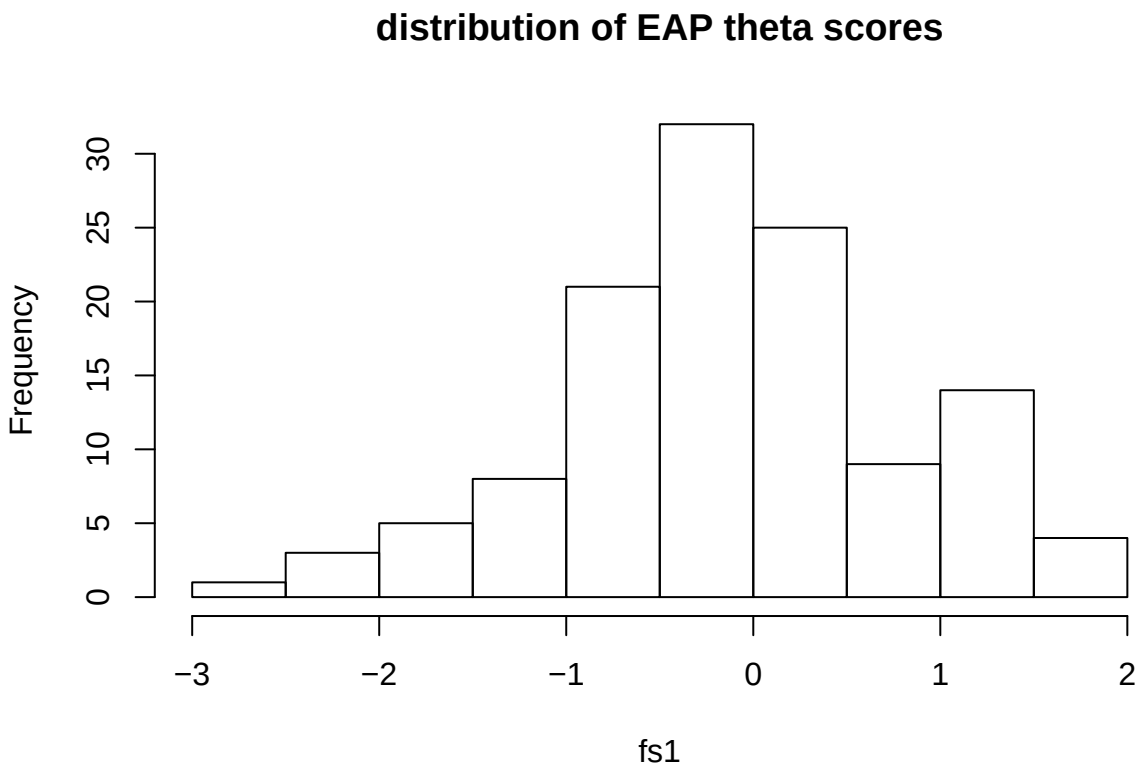
Testing Dataset

To estimate scores in the derived short-form for the participants in the testing dataset:

```
#score items from testing data using custom model

mod <- mirt(tedata1[,c(2,3,7,9,11,13,14,20)], 1, pars = tr1)
```

```
fs1=fscores(mod) #IRT EAP scores for the testing dataset
hist(fs1,main='distribution of EAP theta scores')
```




```
#End
```

Computerized Adaptive Testing

The goal of CAT is to build a test that is tailored to the participant. In essence, CAT uses an algorithm to select items conditional on the participants previous responses. For this illustration, items are selected to maximize item information at the participant's current standing on the latent variable.

Training Dataset

First, the required packages are loaded:

```
library(catR)
library(mirt)
library(plyr)
```

Then, the IRT model is fitted and the item parameters (also known as the item bank) are saved to a data frame. This step is exactly the same step as the one for IRT short-forms discussed above.

```
n2=mirt(trdata1,1)  #fit mirt model
n3=coef(n2,IRTpars=T)
n3[[21]]<-NULL
n3=lapply(n3,data.frame)
n4<-do.call(rbind.fill,n3)
```

Testing dataset

In this step, the CAT characteristics are specified, such as using the item parameters from the training dataset as the item bank, start item administration by administering the item most informative at the mean of theta, select items based on maximizing item information (MFI), estimate scores using EAP scoring, stop item administration once the EAP[θ] have a .30 SEM, and estimate EAP[θ] scores at the end of item administration. Finally, we also need to specify the EAP[θ] scores for the testing dataset that should be recovered by the CAT.

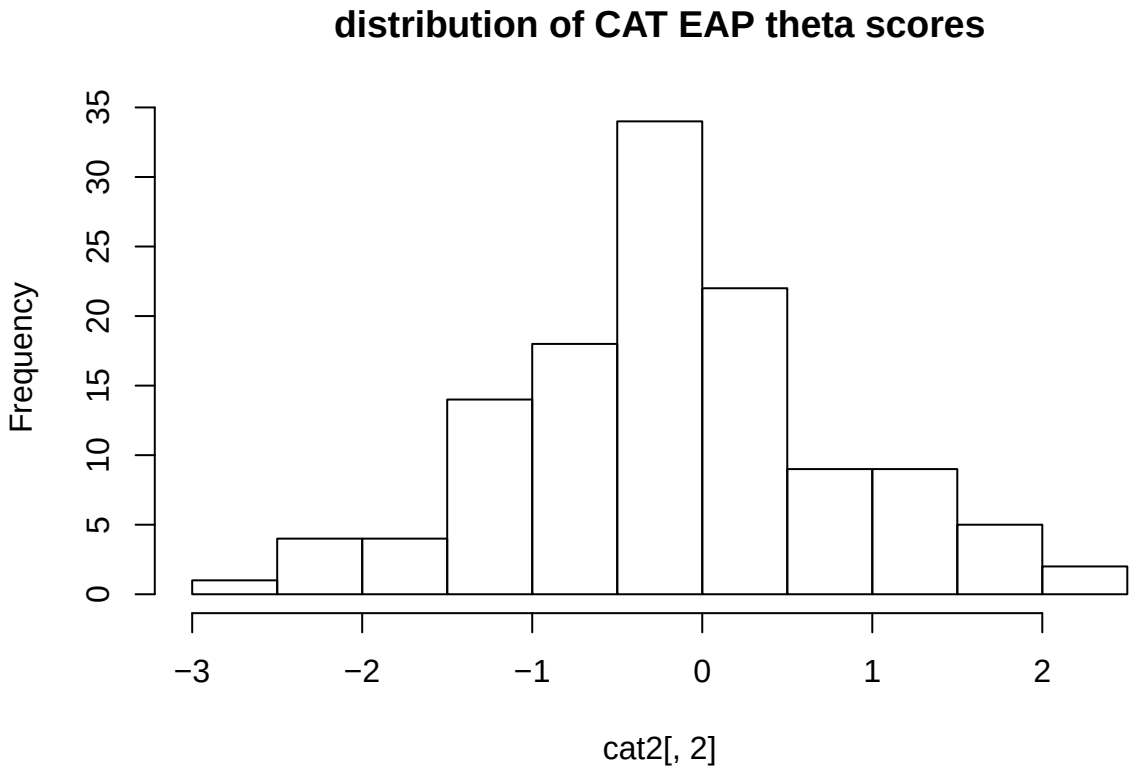
```
bank <- n4   #item bank
start <- list(nrItems = 1, theta = 0)  #start
test <- list(itemSelect = "MFI", method = "EAP") #item selection
stop <- list(rule = "precision", thr = 0.3) #stopping rule
final <- list(method = "EAP") #final score estimate
s4<-fscores(n2,response.pattern=tedata1) #EAP scores for tedata1
s5=s4[,"F1"] #save scores that CAT should recover
```

After specifying the CAT characteristics, then the CAT simulation can start, and basic information about the CAT simulation can be extracted:

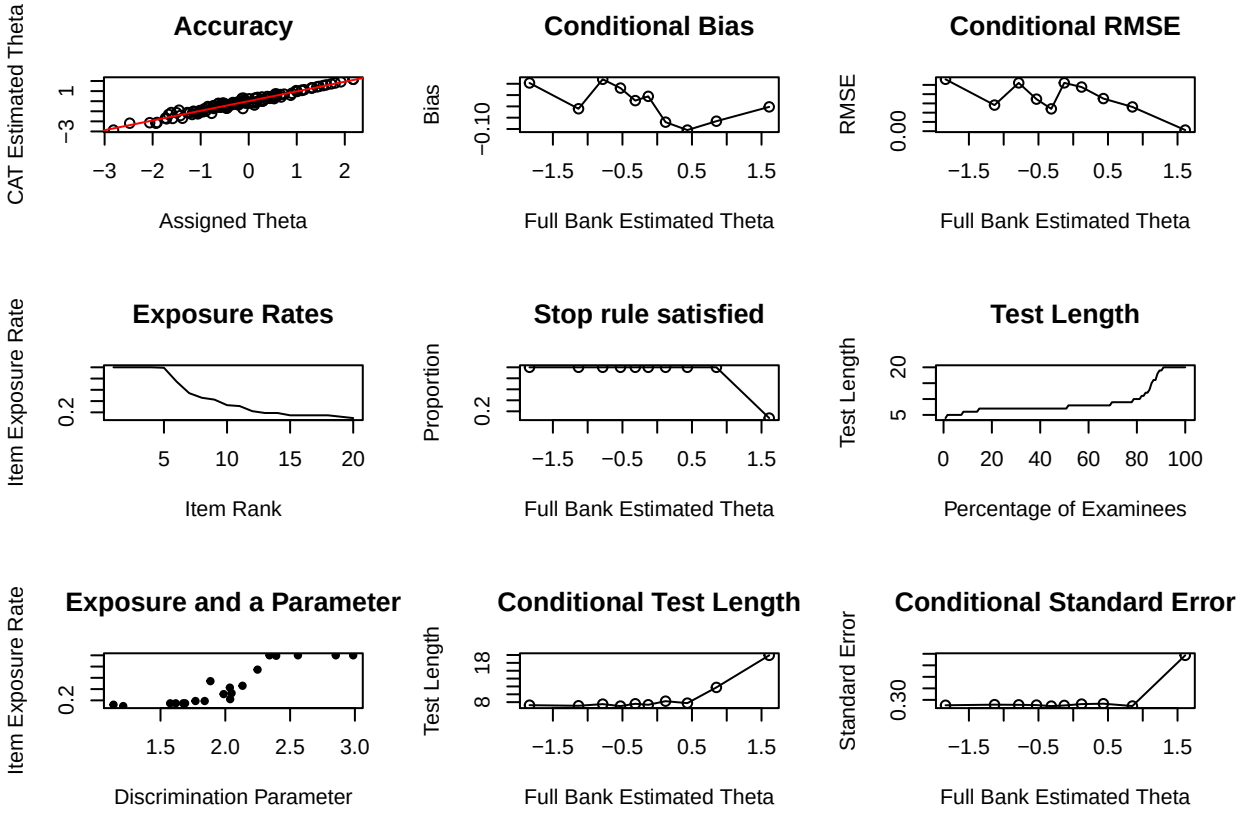
```
#Simulate the CAT
res4 <- simulateRespondents(thetas = s5, responsesMatrix=tedata1,
itemBank = bank, model = "GRM", start = start, test = test,
stop = stop, final = final)
```

```
## Simulation process:  0 %
## Simulation process:  10 %
## Simulation process:  20 %
## Simulation process:  30 %
## Simulation process:  40 %
## Simulation process:  50 %
## Simulation process:  60 %
## Simulation process:  70 %
## Simulation process:  80 %
## Simulation process:  90 %
## Simulation process:  100 %
```

```
#Save CAT values
cat2=res4$final.values.df
hist(cat2[,2],main='distribution of CAT EAP theta scores')
```




```
plot(res4)
```




```
## The plot was not captured!
```

```
#End
```

From the plots above, we could understand a bit about the CAT performance. Most participants who had an EAP[θ] of 0.5 or less received around 8 items to estimate an EAP[θ] score with a reliability of around .90 (SEM = .30). Participants who had an EAP[θ] higher than 1.0 received all of the items, and they could not meet the stopping rule of SEM < .30. Also, it appears that the EAP[θ] score from the CAT largely recovers the EAP[θ] of the whole item set.

Static Short-forms with the Genetic Algorithm

If there are 20 items in the original scales, there are over a million possible short-forms that could be built using these items. The genetic algorithm provides a guided search procedure to explore possible solutions to find the best short-form using an iterative approach inspired in natural evolution. The optimization goal of this approach is to predict the summed score of the original item set with the least number of items. ### Training Dataset First, we load the packages that are needed:

```
library(GAabbreviate)
```

```
## 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: psych
```

Second, we save our training dataset into a matrix and estimate the summed score for all of the 20 items.

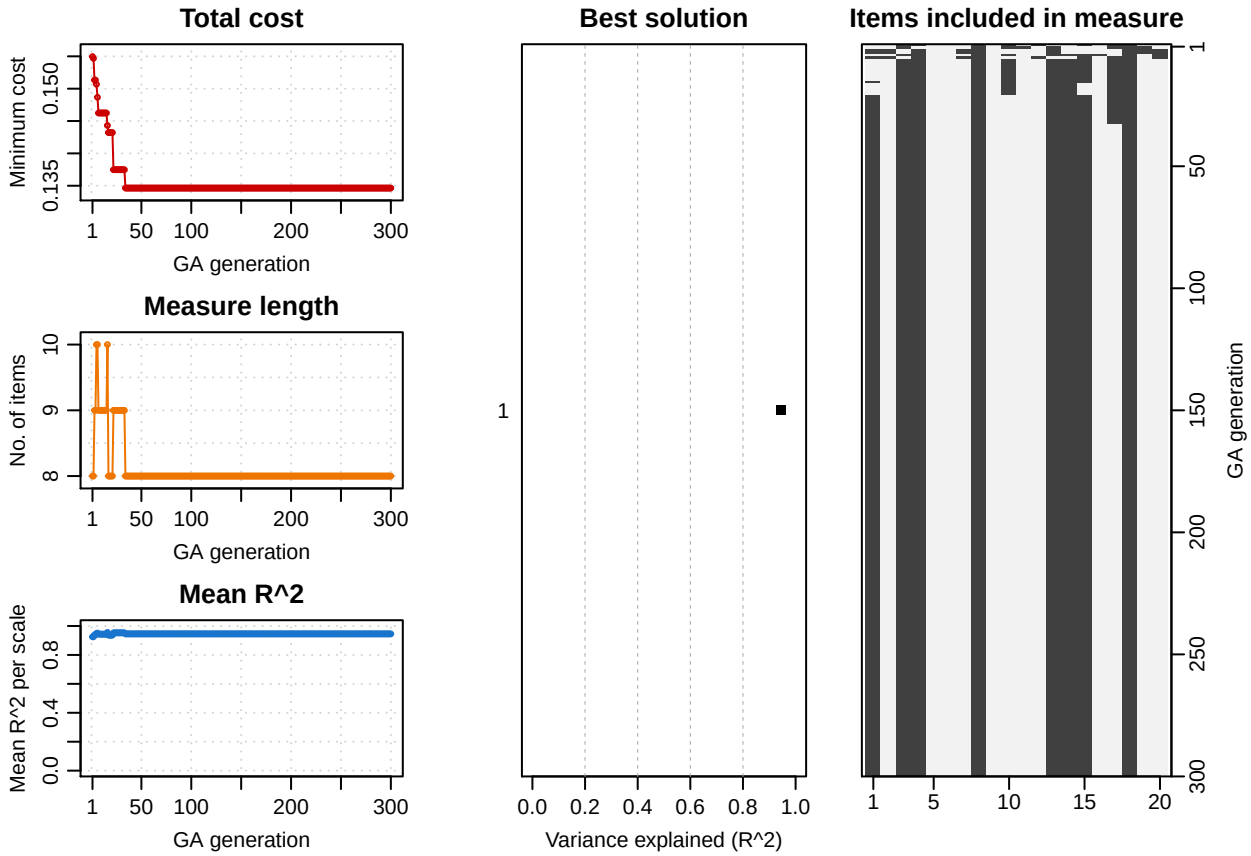
```
items = matrix(sapply(trdata1,as.numeric),nrow=nrow(trdata1),ncol=ncol(trdata1)) #summed scores
scales = cbind(rowSums(items)) #summed scores
```

The genetic algorithm is then carried out using the fit function presented in the paper (Eq. 5) with the following specifications: item cost of .01, as many as ten items in the short-form, population size of 10 chromosomes, 300 generations, and no internal cross-validation (we will do an external cross-validation – we have a testing dataset).

```
set.seed(2018)
GAA = GAabbreviate(items=items, scales=scales, itemCost = 0.01,
maxItems = 10, popSize = 10, maxiter = 300,crossVal=FALSE,verbose=FALSE)
```

The solution by the genetic algorithm is then inspected and the items that the genetic algorithm chose are extracted. After that, the items extracted were used to predict the summed score of the training dataset using regression.

```
plot(GAA)
```




```
GAA$best #min fit function
```

```
## $cost
## [1] 0.134639
##
## $fit
## [1] 0.054639
```

```
GAA$measure$items #items chosen by column number
```

```
##  x1  x3  x4  x8 x13 x14 x15 x18
##   1   3   4   8  13  14  15  18
```

```
cln=colnames(trdata1)[GAA$measure$items]
cln #names of items chosen from dataset
```

```
## [1] "ssrq1"  "ssrq3"  "ssrq4"  "ssrq10" "ssrq15" "ssrq16" "ssrq17" "ssrq26"
```

```
galm=lm(scales~.,data=data.frame(trdata1[,GAA$measure$items])) #fit reg model
summary(galm)
```

```
##
## Call:
## lm(formula = scales ~ ., data = data.frame(trdata1[, GAA$measure$items]))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.3260  -1.7804   0.0366   1.9004   8.4440
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.8155     0.5836  11.678  < 2e-16 ***
## ssrq1         2.1551     0.2039  10.569  < 2e-16 ***
## ssrq3         1.8349     0.1389  13.210  < 2e-16 ***
## ssrq4         2.3137     0.1777  13.021  < 2e-16 ***
## ssrq10        2.8363     0.2365  11.991  < 2e-16 ***
## ssrq15        2.5477     0.2386  10.678  < 2e-16 ***
## ssrq16        2.0600     0.1906  10.808  < 2e-16 ***
## ssrq17        1.9907     0.2371   8.398 8.49e-16 ***
## ssrq26        2.2150     0.2202  10.057  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.89 on 391 degrees of freedom
## Multiple R-squared:  0.9475, Adjusted R-squared:  0.9464
## F-statistic: 882.2 on 8 and 391 DF,  p-value: < 2.2e-16
```

Below are the items that the genetic algorithm chose for the short-form:

* SSRQ01. I don't notice the effects of my actions until it's too late.
* SSRQ03. It's hard for me to notice when I've 'had enough' (alcohol, food, sweets).
* SSRQ04. I have trouble following through with things once I've made up my mind to do something.
* SSRQ10. I learn from my mistakes.
* SSRQ15. As soon as I see a problem or challenge, I start looking for possible solutions.
* SSRQ16. I have a hard time setting goals for myself.
* SSRQ17. When I'm trying to change something, I pay a lot of attention to how I'm doing.
* SSRQ26. If I wanted to change, I am confident that I could do it.

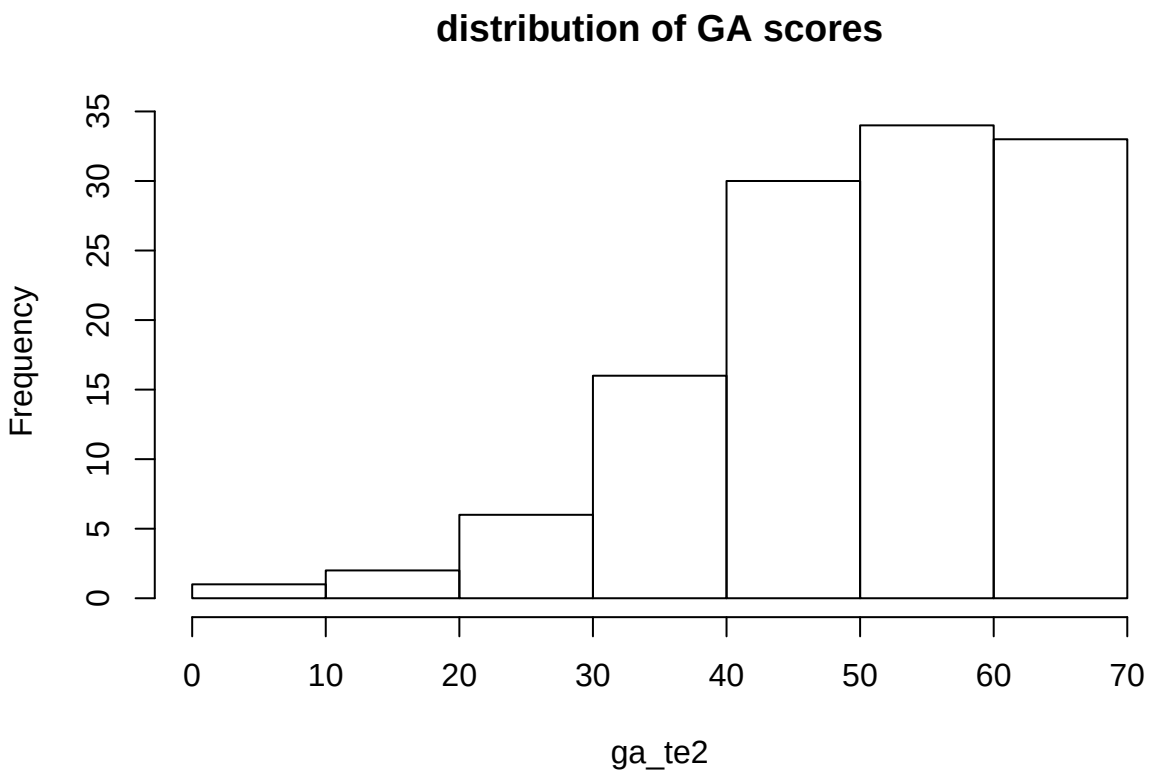
The predicted scores for the scores on the genetic algorithm short-form would be determined by the following equation:

$$GA_{score} = 6.81 + 2.16 ssrq_1 + 1.83 ssrq_3 + 2.31 ssrq_4 + 2.84 ssrq_{10} + 2.54 ssrq_{15} + 2.06 ssrq_{16} + 1.99 ssrq_{17} + 2.22 ssrq_{26}$$

Testing dataset

Finally, the model from the training dataset was used to estimate the predicted summed score in the testing dataset

```
ga_te=tedata1[,cln]
ga_te2=predict(galm,newdata=as.data.frame(ga_te))
hist(ga_te2,main='distribution of GA scores')
```




```
#End
```

Regression Trees for Tailored Testing

Regression trees have been previously proposed as an option for adaptive testing when the dataset does not meet the psychometric assumptions of CAT. In this case, regression trees could be built using the conditional inference framework to derive a tree-like structure that yield a series of rules in how a set of items could be administered. The tree is built in a training dataset where all of the items have been administered for respondents. However, once the tree is built, new respondents would receive items depending on the hierarchy of splits shown in the tree. The optimization goal of the tree is to predict the summed score on the whole set of items.

Training dataset

First, we load the packages that are needed:

```
library(partykit)
```

```
## Loading required package: grid
```

```
## Loading required package: libcoin
```

```
## Loading required package: mvtnorm
```

Then, the summed score for all of the items is estimated and then a tree is grown to predict the summed score for individual items. The p-value threshold was around .05.The tree is long, so only the rules are presented:

```
trsum=rowSums(trdata1)
tree1=ctree(trsum~.,data=as.data.frame(trdata1))
tree1
```

```
##
## Model formula:
## trsum ~ ssrq1 + ssrq2 + ssrq3 + ssrq4 + ssrq7 + ssrq8 + ssrq9 +
##     ssrq10 + ssrq11 + ssrq12 + ssrq13 + ssrq14 + ssrq15 + ssrq16 +
##     ssrq17 + ssrq21 + ssrq24 + ssrq26 + ssrq27 + ssrq29
##
## Fitted party:
## [1] root
## |   [2] ssrq16 <= 3
## |   |   [3] ssrq11 <= 1
## |   |   |   [4] ssrq13 <= 0: 20.100 (n = 10, err = 504.9)
## |   |   |   [5] ssrq13 > 0
## |   |   |   |   [6] ssrq1 <= 0: 25.778 (n = 9, err = 521.6)
## |   |   |   |   [7] ssrq1 > 0: 37.143 (n = 21, err = 694.6)
## |   |   [8] ssrq11 > 1
## |   |   |   [9] ssrq13 <= 1
## |   |   |   |   [10] ssrq9 <= 2: 31.222 (n = 9, err = 355.6)
## |   |   |   |   [11] ssrq9 > 2: 42.654 (n = 26, err = 893.9)
## |   |   |   [12] ssrq13 > 1
## |   |   |   |   [13] ssrq4 <= 3
## |   |   |   |   |   [14] ssrq1 <= 1
## |   |   |   |   |   |   [15] ssrq16 <= 2: 37.417 (n = 12, err = 104.9)
## |   |   |   |   |   |   [16] ssrq16 > 2: 46.000 (n = 13, err = 212.0)
## |   |   |   |   |   [17] ssrq1 > 1
## |   |   |   |   |   |   [18] ssrq11 <= 3
## |   |   |   |   |   |   |   [19] ssrq24 <= 2: 43.947 (n = 19, err = 342.9)
## |   |   |   |   |   |   |   [20] ssrq24 > 2
## |   |   |   |   |   |   |   |   [21] ssrq2 <= 2
## |   |   |   |   |   |   |   |   |   [22] ssrq26 <= 3
## |   |   |   |   |   |   |   |   |   |   [23] ssrq4 <= 1: 44.000 (n = 7, err = 136.0)
## |   |   |   |   |   |   |   |   |   |   [24] ssrq4 > 1: 48.931 (n = 29, err = 251.9)
## |   |   |   |   |   |   |   |   |   [25] ssrq26 > 3: 52.000 (n = 8, err = 76.0)
## |   |   |   |   |   |   |   |   [26] ssrq2 > 2
## |   |   |   |   |   |   |   |   |   [27] ssrq14 <= 2: 51.458 (n = 24, err = 120.0)
## |   |   |   |   |   |   |   |   |   [28] ssrq14 > 2: 55.875 (n = 16, err = 155.8)
## |   |   |   |   |   |   |   [29] ssrq11 > 3: 56.000 (n = 18, err = 406.0)
## |   |   |   |   [30] ssrq4 > 3
## |   |   |   |   |   [31] ssrq27 <= 2: 54.450 (n = 20, err = 274.9)
## |   |   |   |   |   [32] ssrq27 > 2: 60.200 (n = 20, err = 389.2)
## |   [33] ssrq16 > 3
## |   |   [34] ssrq4 <= 2: 49.364 (n = 11, err = 578.5)
## |   |   [35] ssrq4 > 2
## |   |   |   [36] ssrq2 <= 3
## |   |   |   |   [37] ssrq10 <= 2: 56.611 (n = 18, err = 416.3)
## |   |   |   |   [38] ssrq10 > 2
## |   |   |   |   |   [39] ssrq29 <= 3: 60.762 (n = 21, err = 191.8)
## |   |   |   |   |   [40] ssrq29 > 3: 64.706 (n = 17, err = 149.5)
## |   |   |   [41] ssrq2 > 3
## |   |   |   |   [42] ssrq29 <= 3
## |   |   |   |   |   [43] ssrq17 <= 2: 61.667 (n = 12, err = 178.7)
## |   |   |   |   |   [44] ssrq17 > 2: 65.737 (n = 19, err = 153.7)
## |   |   |   |   [45] ssrq29 > 3
## |   |   |   |   |   [46] ssrq17 <= 2: 65.143 (n = 7, err = 32.9)
## |   |   |   |   |   [47] ssrq17 > 2: 69.206 (n = 34, err = 151.6)
```

```
##
## Number of inner nodes:    23
## Number of terminal nodes: 24
```

Below are the unique items that CART used for adaptive testing:

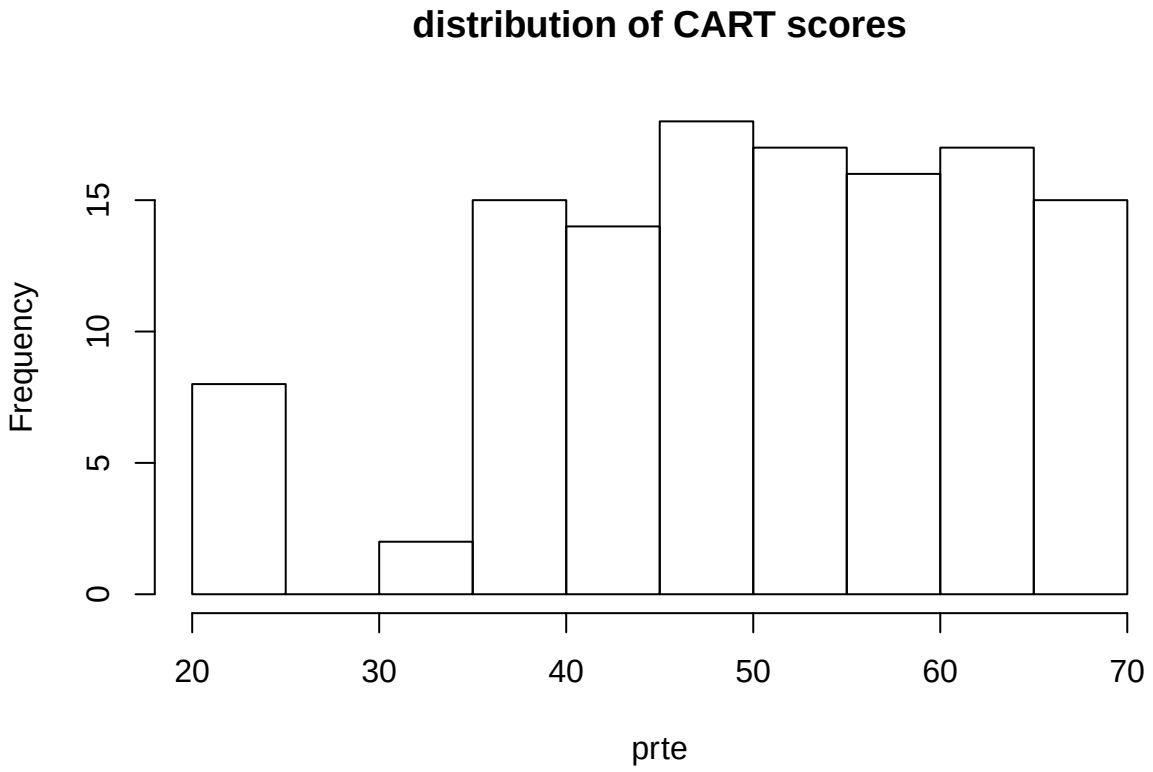
- SSRQ01. I don't notice the effects of my actions until it's too late.
- SSRQ02. I put off making decisions.
- SSRQ04. I have trouble following through with things once I've made up my mind to do something.
- SSRQ09. I usually think before I act.
- SSRQ10. I learn from my mistakes.
- SSRQ11. I give up quickly.
- SSRQ13. I am able to accomplish goals for myself.
- SSRQ14. I have personal standards, and try to live up to them.
- SSRQ16. I have a hard time setting goals for myself.
- SSRQ17. When I'm trying to change something, I pay a lot of attention to how I'm doing.
- SSRQ24. Most of the time I don't pay attention to what I'm doing.
- SSRQ26. If I wanted to change, I am confident that I could do it.
- SSRQ27. I can stick to a plan that's working well.
- SSRQ29. I am able to resist temptation.

According to the tree, in theory, participants from the testing dataset would start by receiving item ssrq16. If participants respond with a 3 or less, then the next item administered would be item ssrq11. If the participants endorsed ssrq16 with a 4 or more, then the participant would be administered item ssrq04. Participants would continue to make their way down the tree until they land on a leaf, which would provide the predicted value in the whole set of 20 items.

Testing Dataset

Finally, the model from the training dataset was used to estimate the predicted summed score in the testing dataset.

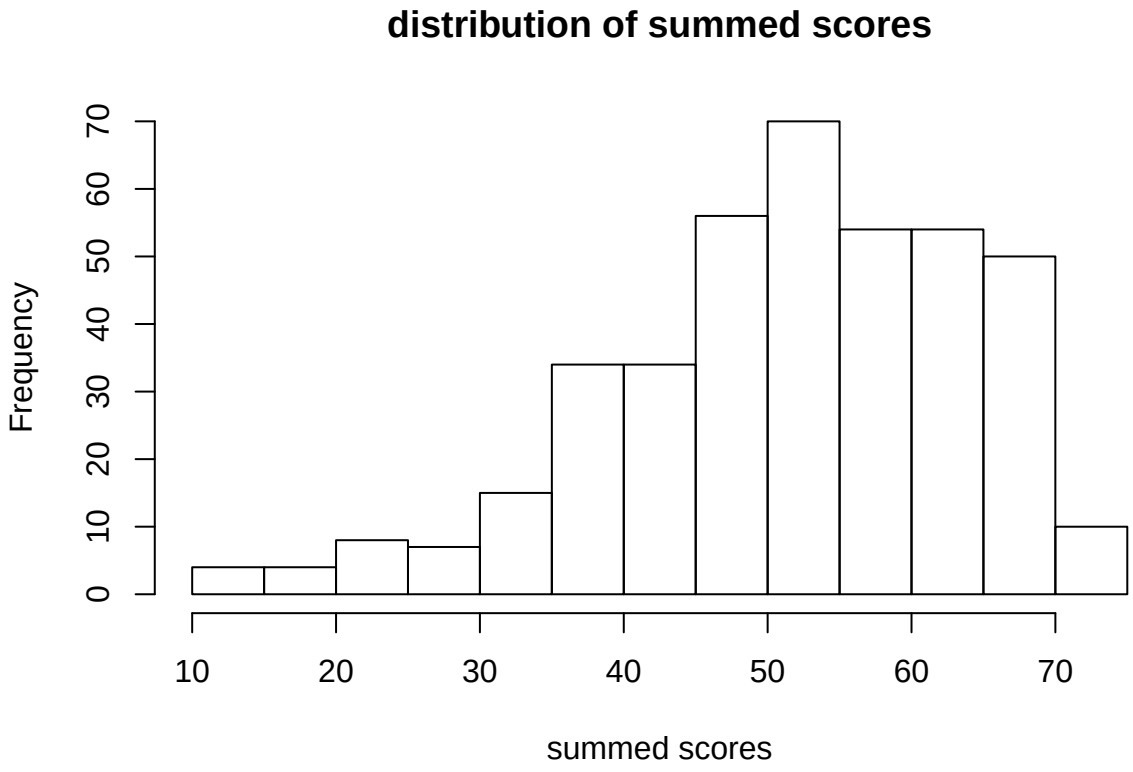
```
prte=predict(tree1,newdata=as.data.frame(tedata1))
hist(prte,main='distribution of CART scores')
```



Benchmark for comparison

For comparision, below is the distribution of the summed scores for the 20 SSRQ items in the testing dataset and the distribution of the IRT scores when the 20 SSRQ items from the testing dataset are administered.

```
hist(scales,main='distribution of summed scores',xlab='summed scores')
```




```
hist(s5,main='distribution of IRT EAP theta scores',xlab='theta scores')
```