Group Search pt.1

The goal of this demonstration is to look at techniques that go beyond linear regression for testing measurment invariance and searching for groups.

One of the main methods we will be covering is SEM Trees, see this Psych Methods publication: http://psycnet.apa.org/journals/met/18/1/71/
Firest, lets load some of the important packages:

```
library(lavaan) # SEM
library(OpenMx) # SEM -- need for semtree -- can't install through R
# source('http://openmx.psyc.virginia.edu/getOpenMx.R')
library(semtree) # SEM Trees & Forests -- can't install through R
#source('http://www.brandmaier.de/semtree/getsemtree.R')
library(pROC) # for roc
```

Neither the OpenMx nor semtree packages are on CRAN, have to download from their respective websites. (see links above)

Note: to install semtree, OpenMx has to already be installed!

SEM Trees Prereq's

We will use the Holzinger-Swineford 1939 data from the lavaan package

```
HS <- HolzingerSwineford1939[,-c(1,4)] # make shorter dataset name
colnames(HS)
```

```
##  [1] "sex"    "ageyr"  "school" "grade"  "x1"     "x2"     "x3"
##  [8] "x4"     "x5"     "x6"     "x7"     "x8"     "x9"
```

So we have 9 variables that will comprise the CFA model (x1-x9), and five covariates ("sex","ageyr","school","grade"). Note, we could combine "ageyr" and "agemo", but this is just a demo.

For the semtree package, you can specify any SEM that either OpenMx or lavaan can fit. The demonstration on the website uses a Latent Growth Curve model:
http://brandmaier.de/semtree/user-guide/using-openmx-with-semtree/

For our purposes, we will just use a simple one-factor CFA model. Note– the proper factor structure is 3 factors, but the more misfit, the more potential to split on covariates.

At the time this script was created, the semtree package works with both OpenMx and lavaan, but with varying degrees of effectiveness. It tends to work across a wider number of conditions with OpenMx (yet much much slower than lavaan), so therefore we will use OpenMx here.

One factor model in lavaan just for demonstration

```
model1.lav <- '
F1 =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
'
lav.fit <- cfa(model1.lav, HS,meanstructure=T)
#summary(lav.fit,fit=T)
```

OpenMx
Note – OpenMx is a little touchy when it comes to starting values, as they are more important to properly specify than lavaan or Mplus in my experience. Therefore, I usually run the model first in lavaan and just copy the parameter estimates. If you have a larger model and don't want to type parameters, you could make it quicker and pull directly from lavaan:

```
# just factor loadings
lav.load <- parameterestimates(lav.fit)$est[parameterestimates(lav.fit)$op == "=~"]
lav.load
```

```
## [1] 1.0000000 0.5075919 0.4928142 1.9301032 2.1230068 1.7959249 0.3846012
## [8] 0.3984500 0.6055726
```

```
# just covariances (residual)
lav.resids <- parameterestimates(lav.fit)$est[parameterestimates(lav.fit)$op == "~~"]
lav.resids
```

```
##  [1] 1.0978452 1.3146610 1.2115889 0.3801255 0.4855457 0.3560649 1.1446042
##  [8] 0.9806184 0.9194656 0.2605274
```

Could assign vector to an object, and specify the object directly in the mxModel()
OpenMx

```
model1.openmx <- mxModel("Model 1",
                         type="RAM",
                         mxData(
                           observed=HS,
                           type="raw"
                         ),
                         manifestVars=c("x1","x2","x3","x4","x5","x6","x7","x8","x9"),
                         latentVars="F1",
                         # residual variances
                         mxPath(
                           from=c("x1","x2","x3","x4","x5","x6","x7","x8","x9"),
                           arrows=2,
                           free=TRUE,
                           values=c(1.1,1.3,1.2,.4,.5,.35,1.1,1,.9),
                           labels=c("e1","e2","e3","e4","e5","e6","e7","e8","e9")
                         ),
                         # latent variance
                         mxPath(
                           from="F1",
                           arrows=2,
                           free=TRUE,
                           values=0.26,
                           labels ="varF1"
```

```
                      ),
                      # factor loadings
                      mxPath(
                        from="F1",
                        to=c("x1","x2","x3","x4","x5","x6","x7","x8","x9"),
                        arrows=1,
                        free=c(FALSE,T,T,T,T,T,T,T,T),
                        values=c(1,0.5,0.5,1.9,2.1,1.8,0.4,0.4,0.6),
                        labels =c("11","12","13","14","15","16","17","18","19")
                      ),
                      # means
                      mxPath(
                        from="one",
                        to=c("x1","x2","x3","x4","x5","x6","x7","x8","x9","F1"),
                        arrows=1,
                        free=c(TRUE,TRUE,TRUE,TRUE,T,T,T,T,T,FALSE),
                        values=c(4.93,6,2.2,3,4.3,2.1,4.1,5.5,5.3,0),
                        labels =c("meanx1","meanx2","meanx3","meanx4","meanx5",
                                  "meanx6","meanx7","meanx8","meanx9",NA)
                     ),
                    #mxExpectationNormal(),
                    mxFitFunctionML()#,
                    #mxTryHard()
)


fit.openmx <- mxRun(model1.openmx)

factorSat <- mxRefModels(fit.openmx,run=T)
#summary(fit.openmx,refModels=factorSat)

fit.openmx$output$est
```

```
##        12        13        14        15        16        17        18
## 0.5075915 0.4928121 1.9300901 2.1229996 1.7959205 0.3845915 0.3984575
##        19        e1        e2        e3        e4        e5        e6
## 0.6055671 1.0978431 1.3146576 1.2115908 0.3801267 0.4855452 0.3560638
##        e7        e8        e9     varF1    meanx1    meanx2    meanx3
## 1.1446037 0.9806182 0.9194641 0.2605292 4.9357682 6.0880390 2.2504143
##    meanx4    meanx5    meanx6    meanx7    meanx8    meanx9
## 3.0609059 4.3405292 2.1855698 4.1859013 5.5270756 5.3741223
```

Couple observations from OpenMx –
1. warnings are generally not a problem – will still run (except RED)
2. covariates not in the factor model will still affect run time. It is this fact that causes the much slower run time of OpenMx with semtree in comparison to lavaan w/ semtree

Traditional Multiple Groups

Important thing to remember is that it can only really handle categorical grouping variables * If you want an example of how to specify model in OpenMx let me know * Note that this corresponds to configural invariance

```
model1.lav <- '
F1 =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
'
lav.Group1 <- cfa(model1.lav, HS,meanstructure=T,group="sex")
anova(lav.fit,lav.Group1)
```

```
## Chi Square Difference Test
##
##            Df    AIC    BIC  Chisq Chisq diff Df diff Pr(>Chisq)
## lav.fit    27 7756.4 7856.5 312.26
## lav.Group1 54 7747.8 7947.9 347.92     35.658      27      0.123
```

```
#summary(lav.Group1,fit=T)

# Metric
#lav.Group2 <- cfa(model1.lav, HS,meanstructure=T,group="sex",
#                  group.equal="loadings")
# Strong
#lav.Group3 <- cfa(model1.lav, HS,meanstructure=T,group="sex",
#                  group.equal="loadings,intercepts")
# Strict
#lav.Group4 <- cfa(model1.lav, HS,meanstructure=T,group="sex",
#                  group.equal="loadings,intercepts,residuals")

# also could use measurementInvariance() from semTools package
```

We could continue this process by constraining the loadings to be equal (Metric), intercepts (Strong), and finally residuals (Strict).

But this only really works for categorical variables.

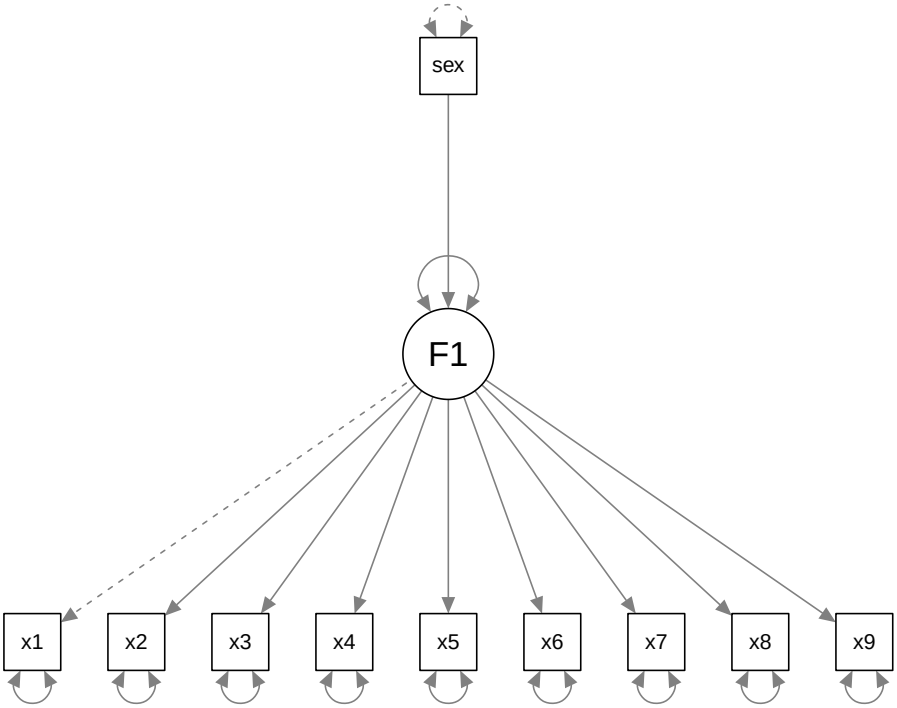
Using covariates

```
library(semPlot)
model11.lav <- '
F1 =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
F1 ~ sex
'
lav.Cov1 <- cfa(model11.lav, HS,meanstructure=T)
#summary(lav.Cov1,fit=T)
parameterEstimates(lav.Cov1)[10,]
```

```
##   lhs op rhs   est    se     z pvalue ci.lower ci.upper
## 1  F1  ~ sex 0.062 0.063 0.984  0.325   -0.061    0.185
```

What does this model look like?

```
semPaths(lav.Cov1,intercepts=F)
```



So now with adding a covariate, we are no longer only affecting the likelihood, we are changing the parameter estimates (loadings).

Are they nested? I don't think so...

This procedure should be somewhat similar to using linear regression with an estimated factor score.

```
# create factor score
fscor.lav = predict(lav.fit)
lm(fscor.lav ~ HS$sex)
```

```
##
## Call:
## lm(formula = fscor.lav ~ HS$sex)
##
## Coefficients:
## (Intercept)       HS$sex
##    -0.08244      0.05442
```

Pretty close estimates, but now the covariate has no relationship to factor loadings residuals etc...

How about summed score?

```
# first scale variables
indicators <- scale(HS[,5:13])
sum <- rowSums(indicators)
lm(sum ~ HS$sex)
```

```
##
## Call:
## lm(formula = sum ~ HS$sex)
##
## Coefficients:
## (Intercept)       HS$sex
##      0.2217      -0.1463
```

Very different estimates, which is to be expected. Creating factor scores can be though of as a weighted form of summed scores, where the weight corresponds to the factor loading.

We can do the same things with continuous covariates, but we can use continuous covariates in the sense of assessing invariance. Further, if we enter a continuous variable to assess invariance, it splits the sample into groups based on each value of the covariate. This is infeasible unless there are a large number of cases (people) per value of the covariate.

Later, we will look at if the covariate is ordinal. Traditional multiple groups treat them as nominal, not ordinal.

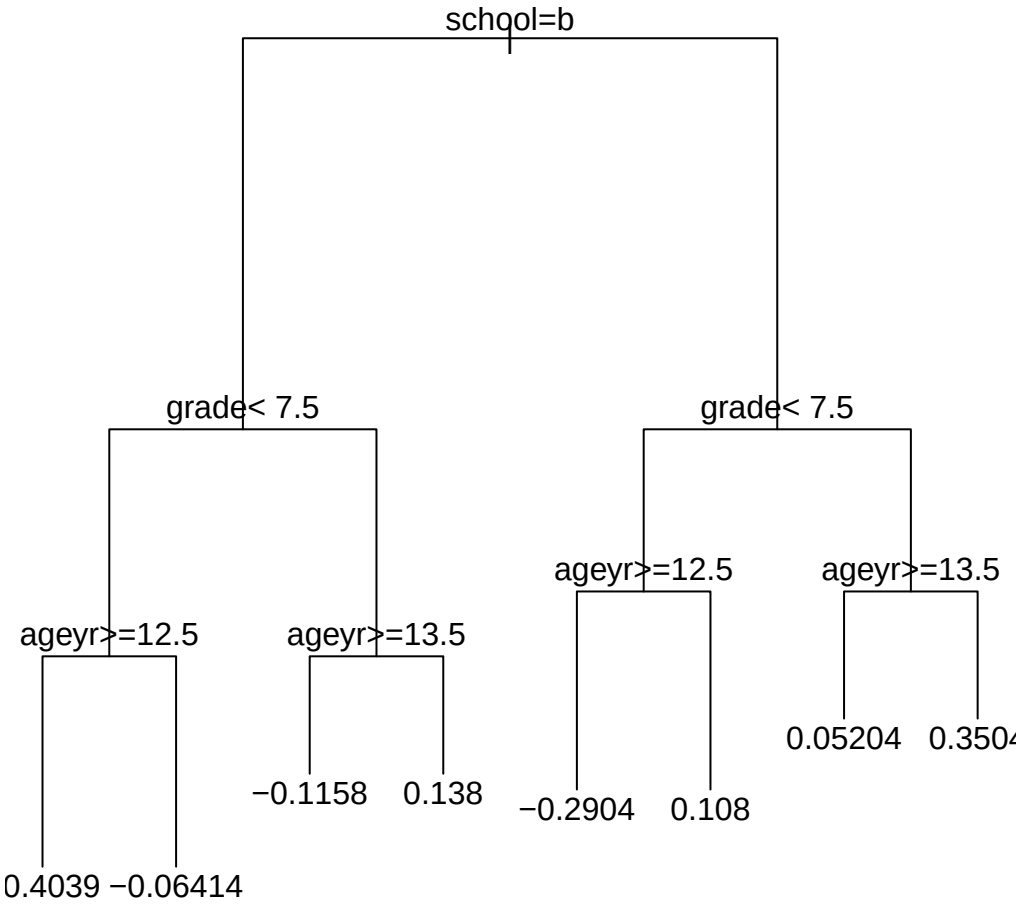
SEM Trees

http://brandmaier.de/semtree/
Now that we have this model, we can use the model object for semtree()

But before we try out SEM Trees, lets compare it to two alternative, but very similar methods: The use of Decision Trees with factor scores and summed scores. In some situations, it may be interesting to compare the resulting tree structure across methods.

Factor scores:

```
# append to HS
HS.run <- HS
HS.run$fscor <- fscor.lav
rpart.fscor <- rpart(fscor ~ sex + ageyr + school + grade,data=HS.run)
plot(rpart.fscor,compress=T);text(rpart.fscor)
```



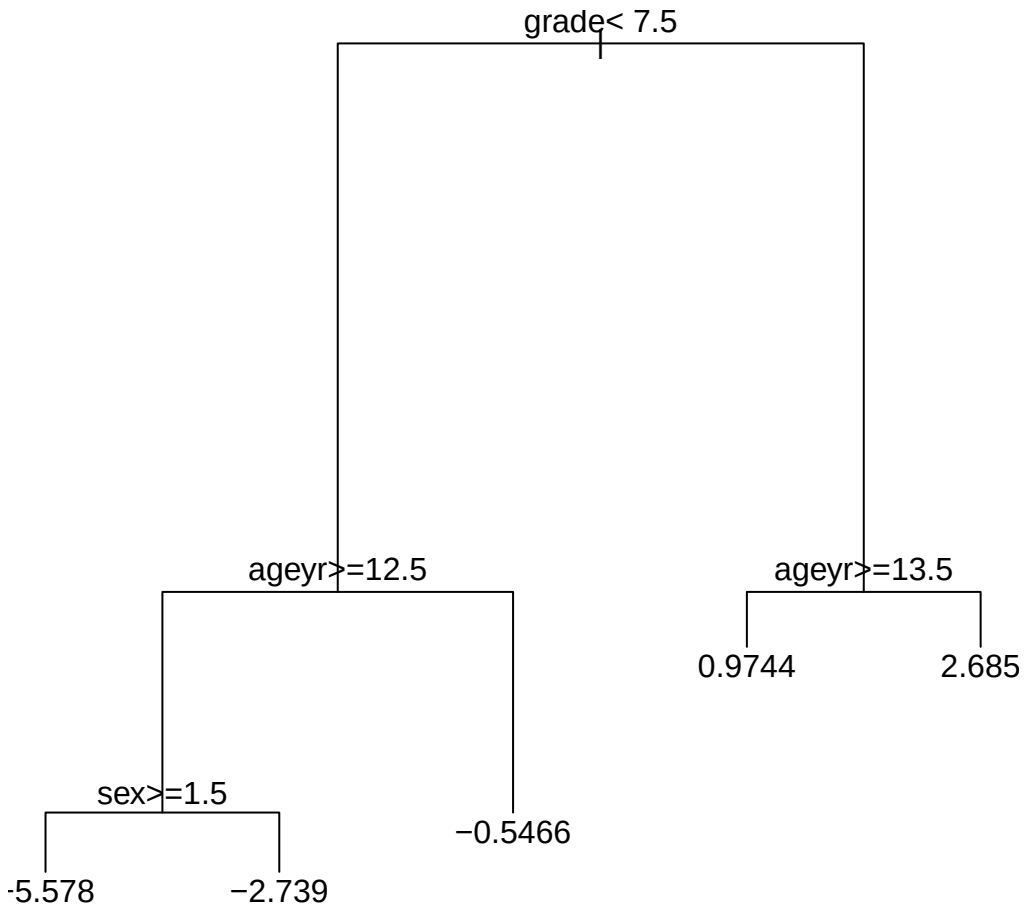

```
rpart.fscor # first split is school
```

```
## n= 301
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 301 69.559770 -1.635827e-16
##    2) school=Pasteur 156 33.592340 -1.136381e-01
##      4) grade< 7.5 78 11.634410 -2.383839e-01
##        8) ageyr>=12.5 40  5.454160 -4.039130e-01 *
##        9) ageyr< 12.5 38  3.930578 -6.414269e-02 *
##      5) grade>=7.5 78 19.530330  1.110760e-02
##       10) ageyr>=13.5 39 10.128040 -1.157839e-01 *
##       11) ageyr< 13.5 39  8.146379  1.379991e-01 *
##    3) school=Grant-White 145 31.785560  1.222590e-01
##      6) grade< 7.5 79 18.999410  2.230835e-02
##       12) ageyr>=12.5 17  3.143776 -2.903905e-01 *
##       13) ageyr< 12.5 62 13.737580  1.080484e-01 *
##      7) grade>=7.5 66 11.052250  2.418968e-01
##       14) ageyr>=13.5 24  4.145247  5.204243e-02 *
##       15) ageyr< 13.5 42  5.547606  3.503850e-01 *
```

Summed Scores:

```
# note, if you have missing data and want to impute it
# one of the easiest procedures I know of uses caret
# uses k-nearest neighbors imputation -- quick and easy
library(caret)
sub1 = HS.run[,5:13]
HS.run$sum <- sum
proc = preProcess(sub1,method=c("center","scale","knnImpute"))
model1.proc = predict(proc,sub1)
# HS$sum <- rowSums(model1.proc)

rpart.sumscor <- rpart(sum ~ sex + ageyr + school + grade,data=HS.run)
plot(rpart.sumscor);text(rpart.sumscor)
```




```
rpart.sumscor # first split is grade
```

```
## n= 301
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 301 8331.0980 -9.530951e-16
```

```
##   2) grade< 7.5 157 3801.0330 -1.776429e+00
##     4) ageyr>=12.5 57 1250.2310 -3.934017e+00
##       8) sex>=1.5 24  441.2898 -5.577617e+00 *
##       9) sex< 1.5 33  696.9555 -2.738673e+00 *
##     5) ageyr< 12.5 100 2134.2090 -5.466037e-01 *
##   3) grade>=7.5 144 3494.4470  1.936801e+00
##     6) ageyr>=13.5 63 1558.1790  9.744001e-01 *
##     7) ageyr< 13.5 81 1832.5320  2.685335e+00 *
```

Interesting that there are very different structure based on whether factor scores or summed scores are used.

In this case, what do we gain using DT's over just linear regression?

```
lm.out1 <- lm(fscor ~ sex + ageyr + school + grade,data=HS.run)
summary(lm.out1)
```

```
##
## Call:
## lm(formula = fscor ~ sex + ageyr + school + grade, data = HS.run)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.14217 -0.28923 -0.01292  0.23975  1.30340
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -0.81175    0.41294  -1.966  0.05026 .
## sex            0.01296    0.05054   0.256  0.79781
## ageyr         -0.17060    0.02899  -5.885 1.08e-08 ***
## schoolPasteur -0.16413    0.05166  -3.177  0.00164 **
## grade          0.41373    0.05822   7.106 9.02e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4297 on 295 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.215,  Adjusted R-squared:  0.2044
## F-statistic:  20.2 on 4 and 295 DF,  p-value: 1.016e-14
```

```
lm.out2 <- lm(sum ~ sex + ageyr + school + grade,data=HS.run)
summary(lm.out2)
```

```
##
## Call:
## lm(formula = sum ~ sex + ageyr + school + grade, data = HS.run)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.531  -3.005  -0.180   2.647  19.634
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)   -19.38280    4.59902  -4.215 3.33e-05 ***
## sex            -0.49610    0.56291  -0.881    0.379
## ageyr          -1.50044    0.32287  -4.647 5.08e-06 ***
## schoolPasteur  -0.01643    0.57532  -0.029    0.977
## grade           5.30065    0.64846   8.174 8.89e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.785 on 295 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.1876, Adjusted R-squared:  0.1765
## F-statistic: 17.03 on 4 and 295 DF,  p-value: 1.42e-12
```

Gain a little predictive power using factor scores. But is the R-squared higher in DT?

```
cor(predict(rpart.fscor),HS.run$fscor)**2
```

```
##             F1
## [1,] 0.2203343
```

```
cor(predict(rpart.sumscor),HS.run$sum)**2
```

```
## [1] 0.2002056
```

Almost identical

How about for which predictors are important/significant?

```
summary(lm.out1)
```

```
##
## Call:
## lm(formula = fscor ~ sex + ageyr + school + grade, data = HS.run)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.14217 -0.28923 -0.01292  0.23975  1.30340
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -0.81175    0.41294  -1.966  0.05026 .
## sex            0.01296    0.05054   0.256  0.79781
## ageyr         -0.17060    0.02899  -5.885 1.08e-08 ***
## schoolPasteur -0.16413    0.05166  -3.177  0.00164 **
## grade          0.41373    0.05822   7.106 9.02e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4297 on 295 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.215,  Adjusted R-squared:  0.2044
## F-statistic:  20.2 on 4 and 295 DF,  p-value: 1.016e-14
```

```
summary(lm.out2)
```

```
##
## Call:
## lm(formula = sum ~ sex + ageyr + school + grade, data = HS.run)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.531  -3.005  -0.180   2.647  19.634
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -19.38280    4.59902  -4.215 3.33e-05 ***
## sex            -0.49610    0.56291  -0.881    0.379
## ageyr          -1.50044    0.32287  -4.647 5.08e-06 ***
## schoolPasteur  -0.01643    0.57532  -0.029    0.977
## grade           5.30065    0.64846   8.174 8.89e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.785 on 295 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.1876, Adjusted R-squared:  0.1765
## F-statistic: 17.03 on 4 and 295 DF,  p-value: 1.42e-12
```

For both, “ageyr” and “grade” seem to be the best predictors

First split in rpart with factor score was ’school“

First split in rpart with sum score is”grade“

All methods agree that”sex" isn’t important

The most important distinction is the DT or CART “automatically” search for interactsion, where linear regression you have to manually create interaction variables.

SEM Trees Run

It is worth noting that there are inherent problems with estimating factor scores – better to not have to estimate and keep as part of SEM. This is exactly what SEM Trees do – no explicit estimation of factor scores.

SEM Trees – just with defaults

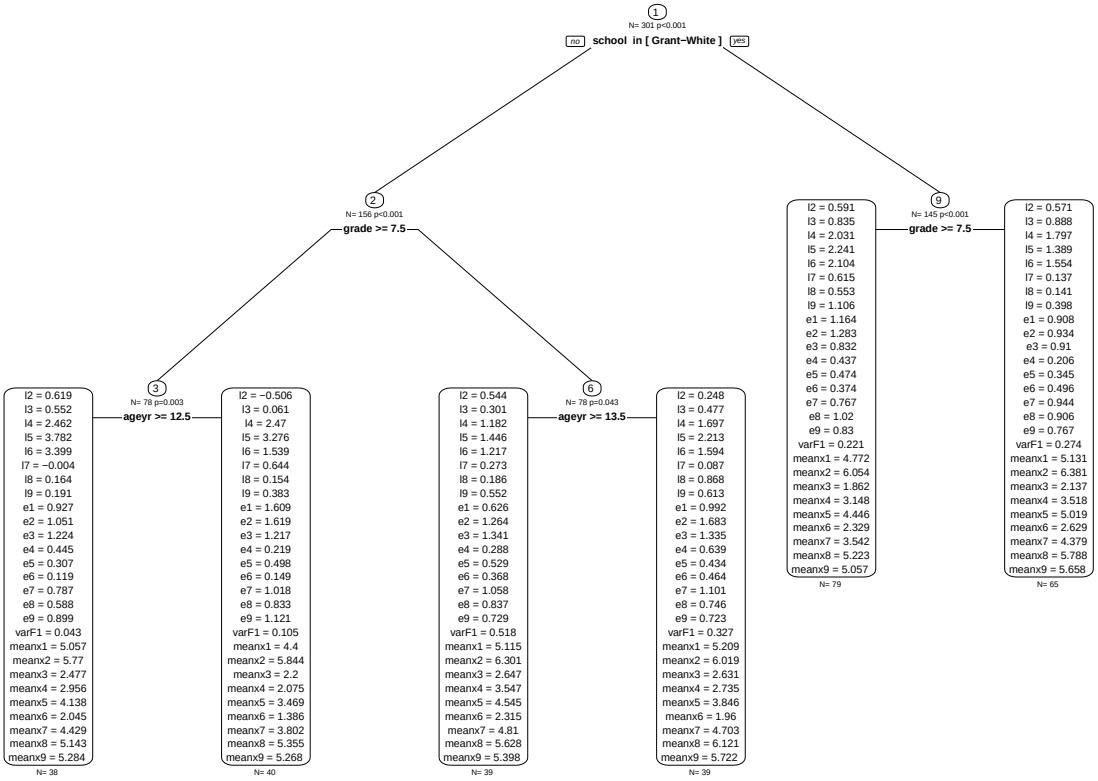
```
#control <- semtree.control()
#control$method <- "naive"
system.time(semtree_model1 <- semtree(fit.openmx))
```

```
##    user  system elapsed
##  24.885   3.077  14.561
```

```
#warnings()
semtree_model1
```

```
## SEMtree with numbered nodes
##  |-[1] school  in [ Grant-White ] [N=301 LR=121.42, df=27]
##  |   |-[2] grade >= 7.5 [N=156 LR=71.94, df=27]
##  |   |   |-[3] ageyr >= 12.5 [N=78 LR=51.91, df=27]
##  |   |   |   |-[4] TERMINAL [N=38]
##  |   |   |   |-[5] TERMINAL [N=40]
##  |   |   |-[6] ageyr >= 13.5 [N=78 LR=40.76, df=27]
##  |   |   |   |-[7] TERMINAL [N=39]
##  |   |   |   |-[8] TERMINAL [N=39]
##  |   |-[9] grade >= 7.5 [N=145 LR=67.31, df=27]
##  |   |   |-[10] TERMINAL [N=79]
##  |   |   |-[11] TERMINAL [N=65]
```

```
plot(semtree_model1)
```




```
# lavaan doesn't work
#system.time(semtree_model11 <- semtree(lav.fit,HS)) # still problem with lavaan
```

So same first split as with factor scores, but not near the depth. Now the Decision Tree algorithm may be slightly different in rpart and semtree, but the bigger difference is in the splitting criterion. semtree uses the likelihood ratio test, which you can think of in relation to a multiple group model. For each split, the data is split into two subgroups. The fit of subgroup1 + subgroup2 is compared to the fit of no groups (all cases). It searches for subgroups with similar model parameters, that are more similar to each other and disimilar from other groups.

Now, there are many different options for modifying the default semtree. These are changed in the semtree.control() function.
Example:

```
control <- semtree.control()

# get all options and see what they need to be called
str(control)
```

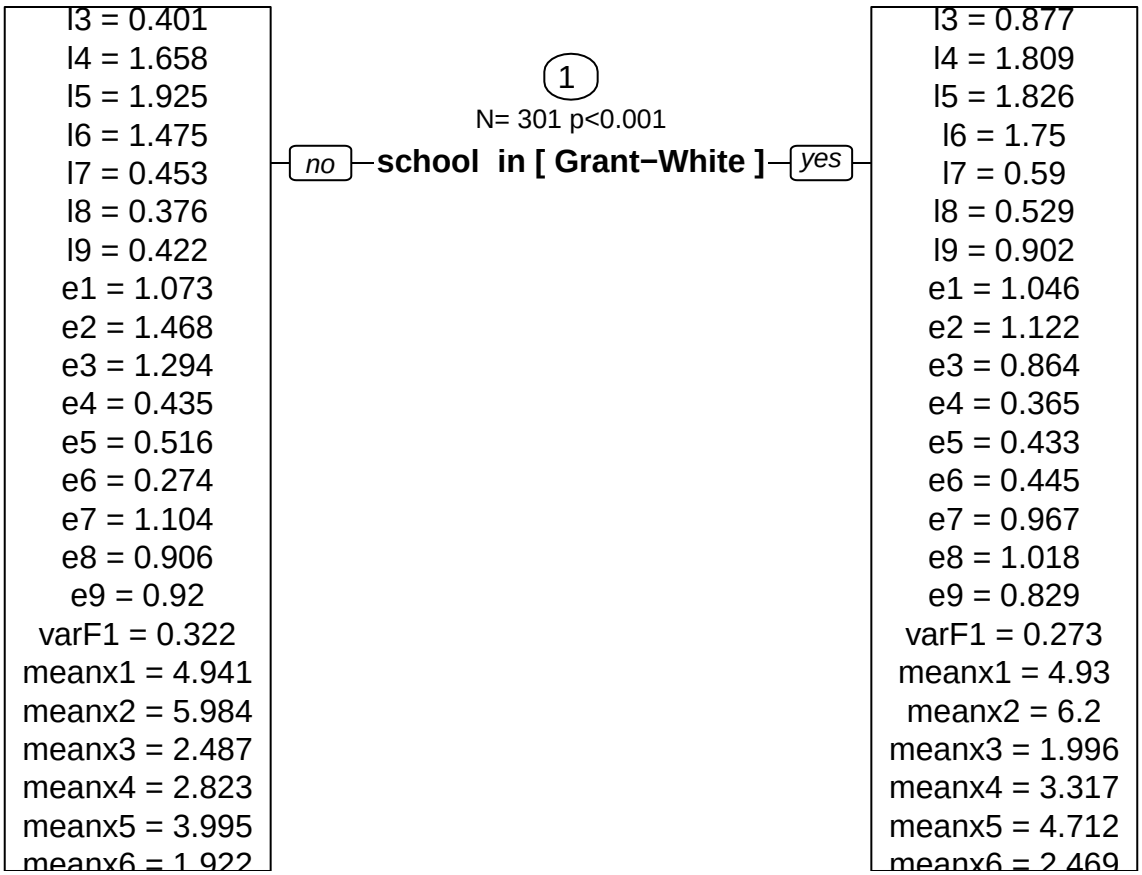
```
## List of 16
##  $ verbose             : logi FALSE
##  $ num.folds           : num 5
##  $ exclude.heywood     : logi TRUE
##  $ min.N               : num 20
##  $ method              : chr "naive"
##  $ max.depth           : logi NA
##  $ alpha               : num 0.05
##  $ alpha.invariance    : logi NA
##  $ progress.bar        : logi TRUE
##  $ bonferroni          : logi FALSE
##  $ use.all             : logi FALSE
##  $ exclude.code        : logi NA
##  $ seed                : logi NA
##  $ mtry                : logi NA
##  $ custom.stopping.rule: logi NA
##  $ report.level        : num 0
##  - attr(*, "class")= chr "semtree.control"
```

```
control$min.N = 50 # important for numerical stability
control$method <- "fair"

control # shows all options
```

```
## SEM-Tree control:
## -----------------
## Splitting Method:  fair
## Alpha Level:  0.05
## Bonferroni Correction: FALSE
## Minimum Number of Cases:  50
## Maximum Tree Depth:  NA
## Number of CV Folds:  5
## Exclude Heywood Cases:  TRUE
## Test Invariance Alpha Level:  NA
## Use all Cases:  FALSE
## Verbosity:  FALSE
## Progress Bar:  TRUE
## Seed:  NA
```

```
semtree_model2 <- semtree(fit.openmx,control=control)
plot(semtree_model2)
```



In this, we changed the splitting criterion method. The default is "naive" which has no correction for the number of response options for each covariate. The "fair" option is very similar to that in the ctree() from party, where it is an unbiased splitting criterion. By unbiased I am referring to the default CART programs give an implicit preference to splitting on covariates that have a large number of response options. By chance, these covariates are more likely to be chosen for splitting simply due to the large number of response options.

the method options are "naive", "fair", and "cv"

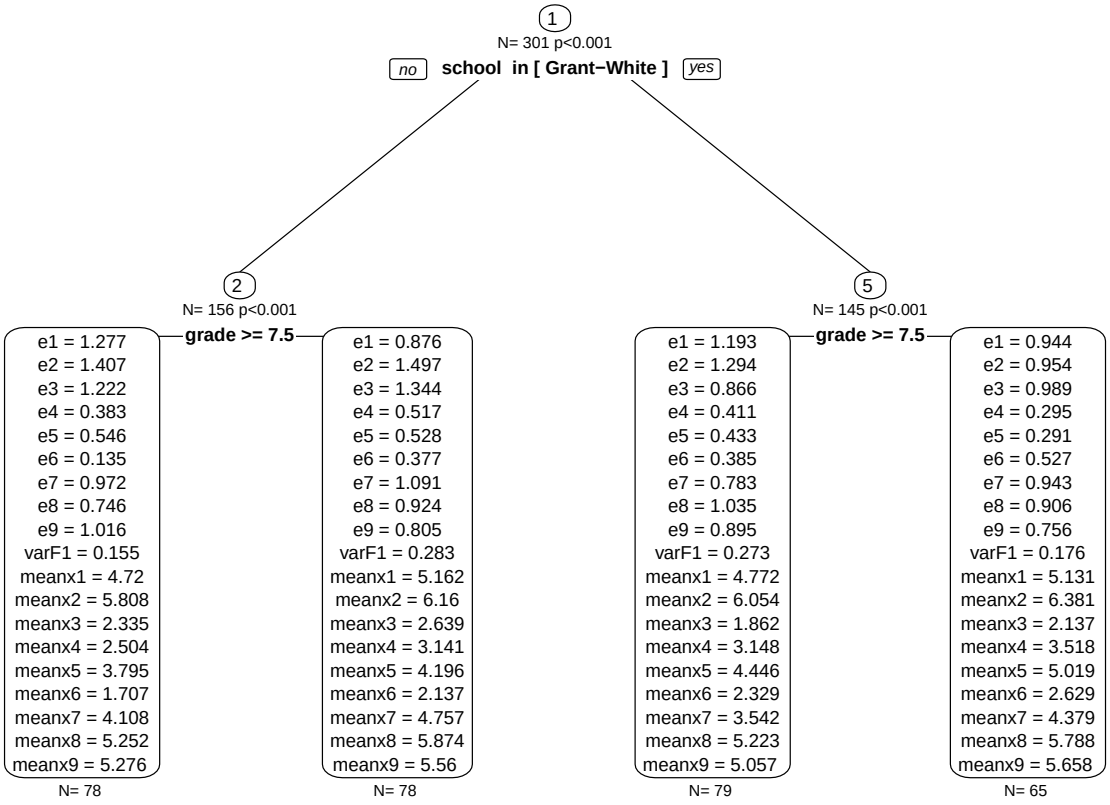
An easy way to ensure measurement invariance. Comparing subgroups without constraining the model and testing for measurement invariance does not ensure that you are measuring the same thing in each potential subgroup that results from a semtree model.

SEM Trees makes it very easy to add invariance constraints w/o having to go back and modify the OpenMx or lavaan models.

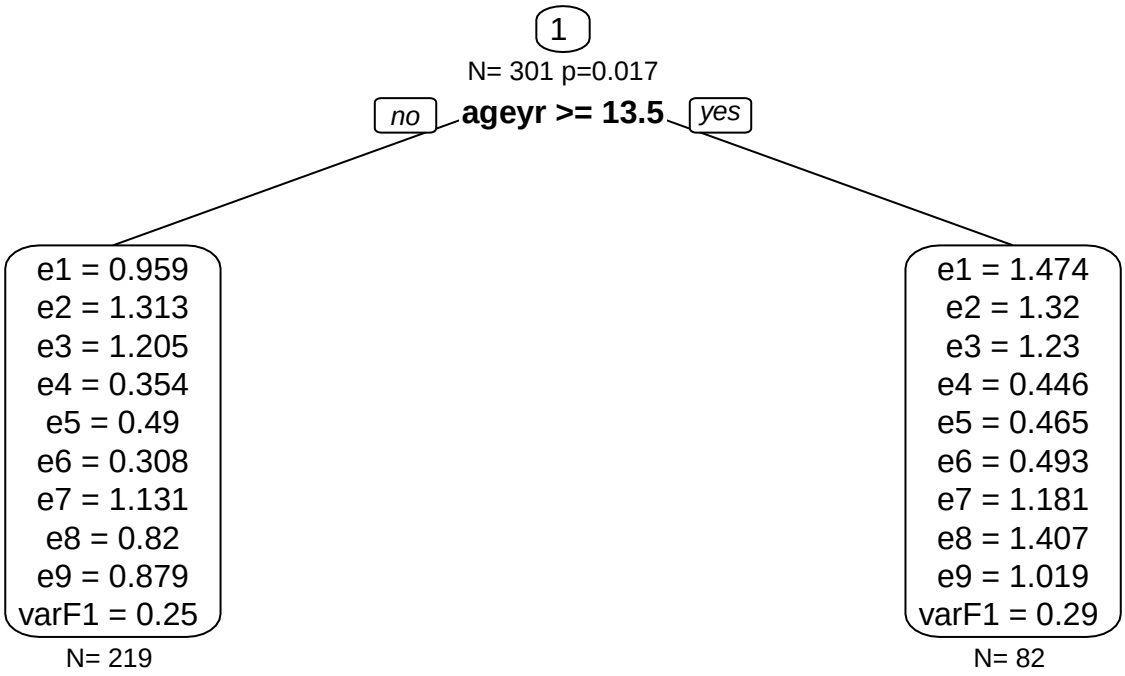
Note:invariance only implemented for "naive"
In this model, I am forcing the loadings to be the exact same across each tested subgroup. Note that this can result in very different model results.

```
# Metric Invariance
control$alpha.invariance <- 0.01
control$method <- "naive"
loads <- c("l2","l3","l4","l5","l6","l7","l8","l9") # "l1" already the same

semtree_invar <- semtree(fit.openmx,control=control,global.constraints=loads)
plot(semtree_invar)
```




```
#same can be done for strong invariance
intercepts <- c("meanx1","meanx2","meanx3","meanx4","meanx5",
                "meanx6","meanx7","meanx8","meanx9")
semtree_invar2 <- semtree(fit.openmx,control=control,global.constraints=c(loads,intercepts))
plot(semtree_invar2)
```



```
# change LRT alpha parameter -- default = 0.05
```

Note: if doing with lavaan, make sure meanstructure=T is used, otherwise hard to get intercept (mean) parameters for each indicator variable.

Other options:

```
control$max.depth <- 5 # how many levels to tree structure
control$min.N <- 50 # minimum number of cases per leaf node
control$num.folds <- 10 # number of cross-validation folds
control$bonferroni <- T # corrects for multiple-testing
control$verbose <- F # show what split is it on
control$progress.bar <- FALSE # annoying messages for parallelizing
```

SEM Forests

Not out in publication yet, but available through the semtree package.

SEM Forests compared to SEM Trees is analogous to Decision Trees compared to Random Forests. With SEM Forests, we are losing interpretability: **no single tree, instead hundreds or thousands**. Even though we lose our ability to explicitly see the relationship between the covariates and the SEM model, we gain much higher predictive power. By using hundreds (thousands) of trees and choosing a subset of predictors to create each tree, we are able to fit the relationship between covariates and SEM to a much higher degree.

With having to create hundreds or thousands of trees it is predictably much much slower than creating a single tree. So similar to using random forests, it becomes advantageous to set up your R session and models to be run in parallel.

Important: different packages in R using different packages for parallelization, and these different packages work differently across Macs and Windows OS's (and Linux)

```
library(snowfall)
sfInit(parallel=T,cpus=4) # cluster of 4 CPUs created, in parallel
```

```
## R Version:  R version 3.1.2 (2014-10-31)
```

```
# not sure how many on your machine?
detectCores()
```

```
## [1] 4
```

```
# Sys.getenv('NUMBER_OF_PROCESSORS') # Windows


sfLibrary(OpenMx)# using OpenMx package in the cluster
```

```
## Library OpenMx loaded.
```

```
sfLibrary(semtree) # using semtree package in the cluster
```

```
## Library semtree loaded.
```

Now that we are maxing out our computer, we are ready to run SEM Forests using the same OpenMx model as before.

```
control <- semforest.control()

# needed for transferring same controls from semtree() to semforest()
control$semtree.control <- semtree.control()


control$num.trees <- 50 # number of trees to fit
control$sampling <- "subsample" # sampling process
#control$sampling <- "bootstrap"
control$mtry <- 2 # number of variables compared per node
print(control)
```

```
## $num.trees
## [1] 50
##
## $sampling
## [1] "subsample"
##
## $premtry
## [1] 0
##
## $mtry
## [1] 2
##
## $semtree.control
## SEM-Tree control:
## -----------------
## Splitting Method:  naive
## Alpha Level:  0.05
## Bonferroni Correction: FALSE
## Minimum Number of Cases:  20
## Maximum Tree Depth:  NA
## Number of CV Folds:  5
## Exclude Heywood Cases:  TRUE
## Test Invariance Alpha Level:  NA
## Use all Cases:  FALSE
## Verbosity:  FALSE
## Progress Bar:  TRUE
## Seed:  NA
##
## attr(,"class")
## [1] "semforest.control"
```

Note: subsampling seems to be preferred: http://www.biomedcentral.com/1471-2105/8/25
Let's run it

```
forest.subsample <- semforest(fit.openmx,HS,control)
```

```
## Job  1 : 1 -> 4  jobs: 4
## Job  2 : 1 -> 4  jobs: 4
## Job  3 : 1 -> 4  jobs: 4
## Job  4 : 1 -> 4  jobs: 4
## Job  1 : 5 -> 8  jobs: 4
## Job  2 : 5 -> 8  jobs: 4
## Job  3 : 5 -> 8  jobs: 4
## Job  4 : 5 -> 8  jobs: 4
## Job  1 : 9 -> 12  jobs: 4
## Job  2 : 9 -> 12  jobs: 4
## Job  3 : 9 -> 12  jobs: 4
## Job  4 : 9 -> 12  jobs: 4
## Job  1 : 13 -> 16  jobs: 4
## Job  2 : 13 -> 16  jobs: 4
## Job  3 : 13 -> 16  jobs: 4
## Job  4 : 13 -> 16  jobs: 4
## Job  1 : 17 -> 20  jobs: 4
## Job  2 : 17 -> 20  jobs: 4
## Job  3 : 17 -> 20  jobs: 4
## Job  4 : 17 -> 20  jobs: 4
## Job  1 : 21 -> 24  jobs: 4
## Job  2 : 21 -> 24  jobs: 4
## Job  3 : 21 -> 24  jobs: 4
## Job  4 : 21 -> 24  jobs: 4
## Job  1 : 25 -> 28  jobs: 4
## Job  2 : 25 -> 28  jobs: 4
## Job  3 : 25 -> 28  jobs: 4
## Job  4 : 25 -> 28  jobs: 4
## Job  1 : 29 -> 32  jobs: 4
## Job  2 : 29 -> 32  jobs: 4
## Job  3 : 29 -> 32  jobs: 4
## Job  4 : 29 -> 32  jobs: 4
## Job  1 : 33 -> 36  jobs: 4
## Job  2 : 33 -> 36  jobs: 4
## Job  3 : 33 -> 36  jobs: 4
## Job  4 : 33 -> 36  jobs: 4
## Job  1 : 37 -> 40  jobs: 4
## Job  2 : 37 -> 40  jobs: 4
## Job  3 : 37 -> 40  jobs: 4
## Job  4 : 37 -> 40  jobs: 4
## Job  1 : 41 -> 44  jobs: 4
## Job  2 : 41 -> 44  jobs: 4
## Job  3 : 41 -> 44  jobs: 4
## Job  4 : 41 -> 44  jobs: 4
## Job  1 : 45 -> 48  jobs: 4
## Job  2 : 45 -> 48  jobs: 4
## Job  3 : 45 -> 48  jobs: 4
## Job  4 : 45 -> 48  jobs: 4
## Job  1 : 49 -> 50  jobs: 2
## Job  2 : 49 -> 50  jobs: 2
```

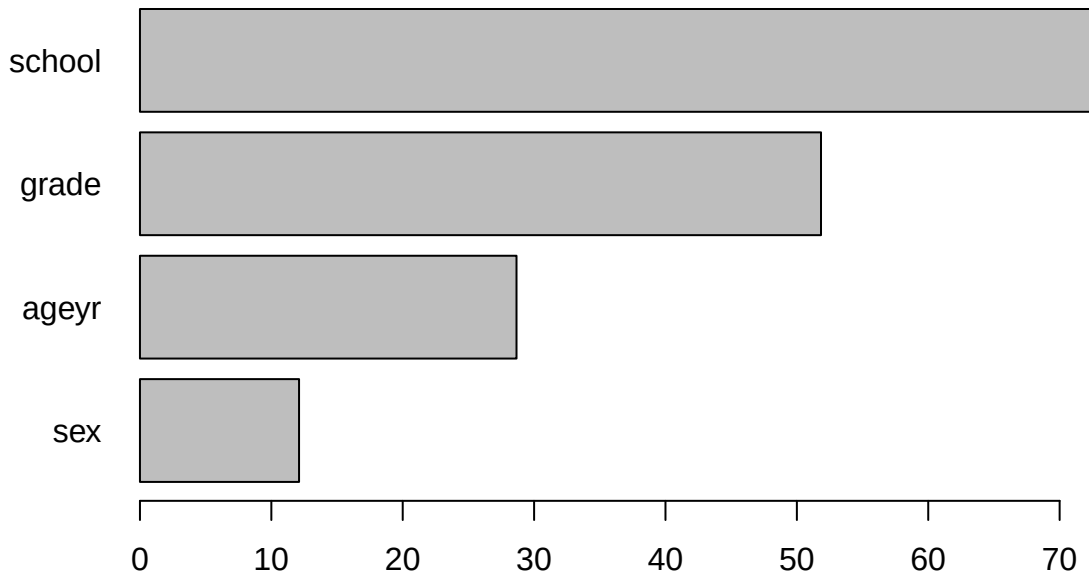
Not much for summary() or print. Instead, other functions for extracting results.
Variable importance is the one of the best ways to get a feel for which variables impacted the prediction the most. This is when there isn't a single tree to visualize.

```
set.seed(1234)
vim.subsample <- varimp(forest.subsample,parallel=T)
```

```
## Job  1 : 1 -> 4  jobs: 4
## Job  2 : 1 -> 4  jobs: 4
## Job  3 : 1 -> 4  jobs: 4
## Job  4 : 1 -> 4  jobs: 4
## Job  1 : 5 -> 8  jobs: 4
## Job  2 : 5 -> 8  jobs: 4
## Job  3 : 5 -> 8  jobs: 4
## Job  4 : 5 -> 8  jobs: 4
## Job  1 : 9 -> 12  jobs: 4
## Job  2 : 9 -> 12  jobs: 4
## Job  3 : 9 -> 12  jobs: 4
## Job  4 : 9 -> 12  jobs: 4
## Job  1 : 13 -> 16  jobs: 4
## Job  2 : 13 -> 16  jobs: 4
## Job  3 : 13 -> 16  jobs: 4
## Job  4 : 13 -> 16  jobs: 4
## Job  1 : 17 -> 20  jobs: 4
## Job  2 : 17 -> 20  jobs: 4
## Job  3 : 17 -> 20  jobs: 4
## Job  4 : 17 -> 20  jobs: 4
## Job  1 : 21 -> 24  jobs: 4
## Job  2 : 21 -> 24  jobs: 4
## Job  3 : 21 -> 24  jobs: 4
## Job  4 : 21 -> 24  jobs: 4
## Job  1 : 25 -> 28  jobs: 4
## Job  2 : 25 -> 28  jobs: 4
## Job  3 : 25 -> 28  jobs: 4
## Job  4 : 25 -> 28  jobs: 4
## Job  1 : 29 -> 32  jobs: 4
## Job  2 : 29 -> 32  jobs: 4
## Job  3 : 29 -> 32  jobs: 4
## Job  4 : 29 -> 32  jobs: 4
## Job  1 : 33 -> 36  jobs: 4
## Job  2 : 33 -> 36  jobs: 4
## Job  3 : 33 -> 36  jobs: 4
## Job  4 : 33 -> 36  jobs: 4
## Job  1 : 37 -> 40  jobs: 4
## Job  2 : 37 -> 40  jobs: 4
## Job  3 : 37 -> 40  jobs: 4
## Job  4 : 37 -> 40  jobs: 4
## Job  1 : 41 -> 44  jobs: 4
## Job  2 : 41 -> 44  jobs: 4
## Job  3 : 41 -> 44  jobs: 4
## Job  4 : 41 -> 44  jobs: 4
## Job  1 : 45 -> 48  jobs: 4
## Job  2 : 45 -> 48  jobs: 4
## Job  3 : 45 -> 48  jobs: 4
```

```
## Job  4 : 45 -> 48  jobs: 4
## Job  1 : 49 -> 50  jobs: 2
## Job  2 : 49 -> 50  jobs: 2
```

```
plot(vim.subsample)
```



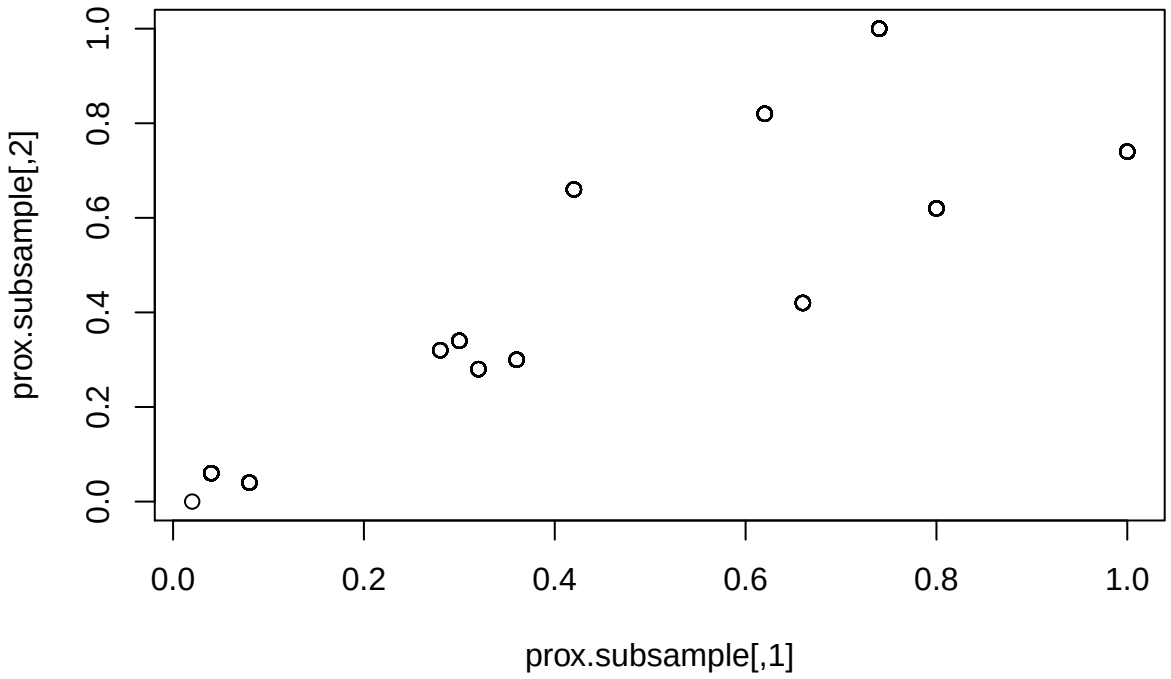
Similar to the semtree results, school seems to be most important (first split)

Variable importance can be thought of the number of times each variable was chosen to be split on across all of the trees that were created. Gives an idea of which variables are necessary if you wanted to run the study again in the future and had to make a choice at which variables to keep from one study to the future study.

Case Proximity Plots. This is a way to a form of clustering of cases (individuals) that tend to have similar values or fit across trees.

analogous to randomForest(. . . ,proximity=TRUE)$proximity from randomForest package

```
prox.subsample <- proximity(forest.subsample)
plot(prox.subsample)
```

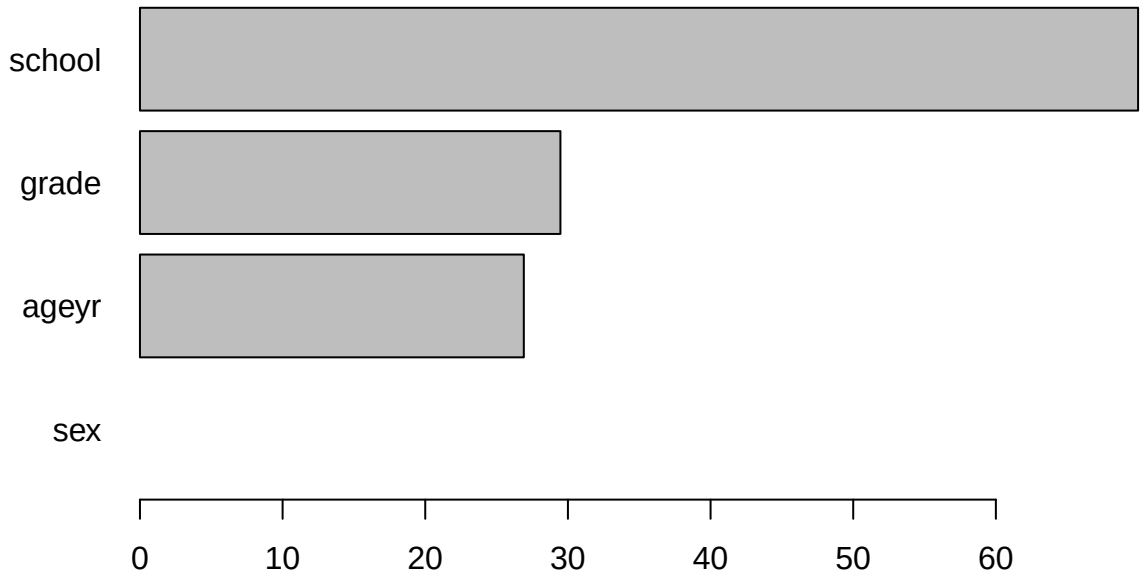


Pruning.
Once a forest is created, you can go back and prune the individual trees in an attempt to make the results more generalizeable.

```
pforest.subsample <- prune.semforest(forest.subsample, max.depth=1)
plot(varimp(pforest.subsample))
```

```
## Job  1 : 1 -> 4  jobs: 4
## Job  2 : 1 -> 4  jobs: 4
## Job  3 : 1 -> 4  jobs: 4
## Job  4 : 1 -> 4  jobs: 4
## Job  1 : 5 -> 8  jobs: 4
## Job  2 : 5 -> 8  jobs: 4
## Job  3 : 5 -> 8  jobs: 4
## Job  4 : 5 -> 8  jobs: 4
## Job  1 : 9 -> 12  jobs: 4
## Job  2 : 9 -> 12  jobs: 4
## Job  3 : 9 -> 12  jobs: 4
## Job  4 : 9 -> 12  jobs: 4
## Job  1 : 13 -> 16  jobs: 4
## Job  2 : 13 -> 16  jobs: 4
## Job  3 : 13 -> 16  jobs: 4
## Job  4 : 13 -> 16  jobs: 4
## Job  1 : 17 -> 20  jobs: 4
## Job  2 : 17 -> 20  jobs: 4
## Job  3 : 17 -> 20  jobs: 4
## Job  4 : 17 -> 20  jobs: 4
## Job  1 : 21 -> 24  jobs: 4
## Job  2 : 21 -> 24  jobs: 4
## Job  3 : 21 -> 24  jobs: 4
## Job  4 : 21 -> 24  jobs: 4
## Job  1 : 25 -> 28  jobs: 4
```

```
## Job  2 : 25 -> 28  jobs: 4
## Job  3 : 25 -> 28  jobs: 4
## Job  4 : 25 -> 28  jobs: 4
## Job  1 : 29 -> 32  jobs: 4
## Job  2 : 29 -> 32  jobs: 4
## Job  3 : 29 -> 32  jobs: 4
## Job  4 : 29 -> 32  jobs: 4
## Job  1 : 33 -> 36  jobs: 4
## Job  2 : 33 -> 36  jobs: 4
## Job  3 : 33 -> 36  jobs: 4
## Job  4 : 33 -> 36  jobs: 4
## Job  1 : 37 -> 40  jobs: 4
## Job  2 : 37 -> 40  jobs: 4
## Job  3 : 37 -> 40  jobs: 4
## Job  4 : 37 -> 40  jobs: 4
## Job  1 : 41 -> 44  jobs: 4
## Job  2 : 41 -> 44  jobs: 4
## Job  3 : 41 -> 44  jobs: 4
## Job  4 : 41 -> 44  jobs: 4
## Job  1 : 45 -> 48  jobs: 4
## Job  2 : 45 -> 48  jobs: 4
## Job  3 : 45 -> 48  jobs: 4
## Job  4 : 45 -> 48  jobs: 4
## Job  1 : 49 -> 50  jobs: 2
## Job  2 : 49 -> 50  jobs: 2
```

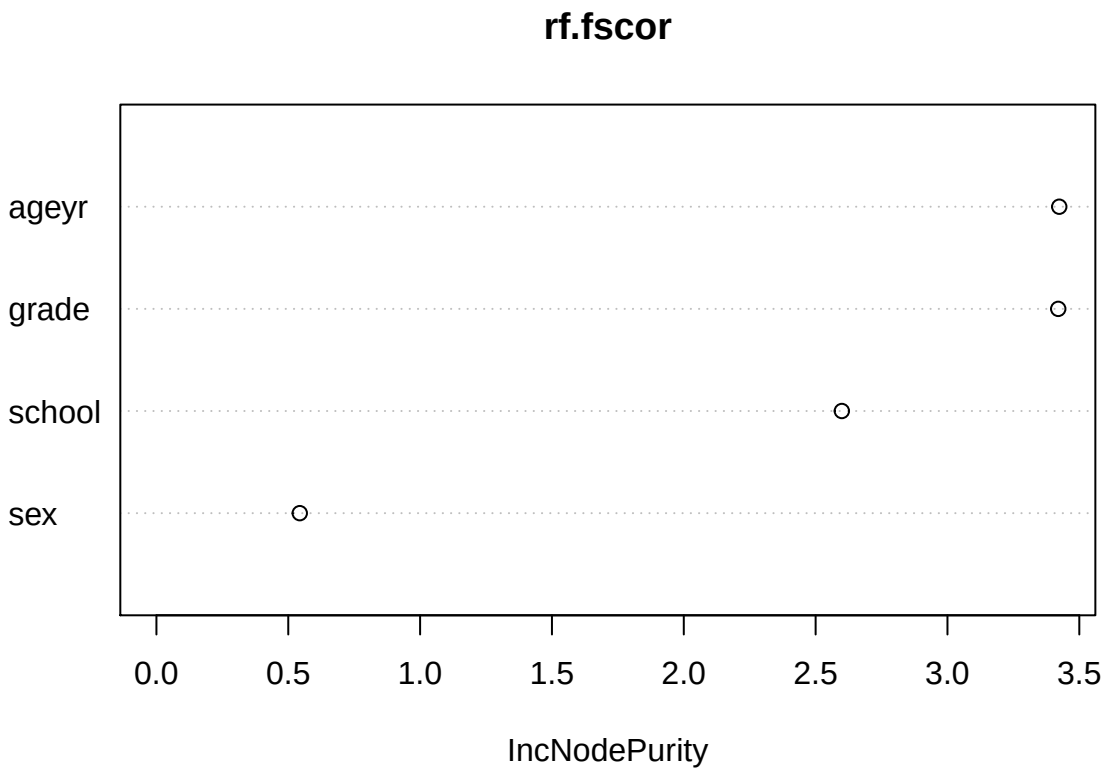


This can change our interpretation of variable importance. Note that grade now becomes the most important variable instead of school.
So would we get similar results if we used randomForest() with factor scores?
Note: randomForest has to have complete cases
– we could impute, but for example we just use remove cases

```
library(psych)
#describe(HS.run)
```

```
HS.comp <- HS.run[complete.cases(HS.run),]

library(randomForest)
rf.fscor <- randomForest(fscor ~ sex + ageyr + school + grade,data=HS.comp)
varImpPlot(rf.fscor)
```



Oddly, now ageyr becomes the most important