R Commands

Data Tasks

1. Vector Manipulation
2. Sort
3. Write to CSV
4. Matrix Manipulation
5. Histogram
6. Saving Objects/Loading Objects
7. Descriptive Statistics
8. Simulating Time Series
9. Simulating Logit Model
10. Simulating Categorical Data
11. Simulating Hierarchical Data
12. Simulating Hierarchical Data-2
13. Plotting two variables
14. Plotting a 2x2 Matrix
15. Replacing Na's with Blanks
16. Character Matrix to Numeric Matrix
17. Bar Chart
18. Scatter Plot
19. DyGraphs
20. Check For Updates
21. Classification Simulation
22. Creating Panel Data
23. Bootstrap

Statistical Models

1. OLS
2. Residual Plots
3. Var test and T test
4. Normality Tests / KS Test
5. Durbin Watson Test
6. IV RANK
7. Non-Linear Least Squares
8. Spectral Analysis
9. Wavelet
10. Dynamic Linear Models
11. Mice

Advanced Statistical Models

1. GMM
2. Hierarchical Linear Models
3. Hierarchical Two Level Linear Model
4. Hierarchical Diagnostics
5. Panel Regression - RE
6. ELO Ratings

Bayesian Models

1. Bayesian Linear Regression + Diagnostics + Model Comparison
2. Bayesian Out of sample Prediction
3. Bayesian AR(1)
4. Bayesian AR(K)
5. Bayesian MA(Q)
6. Bayesian ARMA(1,1) / Bayesian ARMAX / Bayesian ARMA(P,Q)
7. Bayesian Dynamic Panel Regression
8. Bayesian Stochastic Volatility Model
9. Hierarchical Linear Bayesian Model
10. HLM Varying Intercept/Varying slope/Varying Intercept+Slope /Varying w Group Effects / Context Levels
11. HLM Three levels Varying Intercept/Three Levels Varying Intercept + fixed effects/ Three Levels Varying Slope + Intercepts + Fixed effects/
12. HLM +Parallel + Priors + Shiny Stan
13. Heckman Sample Correction
14. Ordered Logit
15. HLM Ordered Logit

Machine Learning Algorithms

1. Text Miner: Simulated Data
2. Text Miner: Twitter Data
3. Text Miner: Speeches
4. ADABOOST
5. Gradient Boosted Trees
6. Ensemble Methods
7. CV Random Forests
8. CV GBM
9. Visualizing Neural Network/Variable Importance
10. Sensitivity Analysis Neural Networks
11. Feature Cleaning and Extraction with Random Forests
12. Data Cleaning
13. PCA Dimension Reduction
14. H20 Deep Learning
15. H20 Random Forests and GBM

Parallel Processing

1. Parallel Processing OLS
2. LMe4 Parallel Processing
3. KMeans Parallel Processing

Vector Algebra

f <- c(1,2,3) - Creating a column Vector
g <- r(1,2,3) - Creating a row vector
dat<- cbind(c(1,2,3),c(1,2,3) - Creating a 3x2 Matrix
dat<- rbind(c(1,2,3),c(1,2,3) - Creating a 2X3 Matrix
F<- t(f) - Transpose
solve(dat) %*% dat - Inverse

Write to CSV

write.csv(MyData, file = "MyData.csv")

Matrix Manipulation

Operator or Function	Description
A * B	Element-wise multiplication
A %*% B	Matrix multiplication
A %o% B	Outer product. **AB'**
crossprod(A,B) **crossprod(A)**	**A'B** and **A'A** respectively.
t(A)	Transpose
diag(x)	Creates diagonal matrix with elements of **x** in the principal diagonal
diag(A)	Returns a vector containing the elements of the principal diagonal
diag(k)	If k is a scalar, this creates a k x k identity matrix. Go figure.
solve(A, b)	Returns vector **x** in the equation **b = Ax** (i.e., $A^{-1}b$)
solve(A)	Inverse of **A** where A is a square matrix.
ginv(A)	Moore-Penrose Generalized Inverse of **A**. ginv(A) requires loading the **MASS** package.
y<-eigen(A)	**y$val** are the eigenvalues of **A** **y$vec** are the eigenvectors of **A**
y<-svd(A)	Single value decomposition of **A**. **y$d** = vector containing the singular values of **A** **y$u** = matrix with columns contain the left singular vectors of **A** **y$v** = matrix with columns contain the right singular vectors of **A**
R <- chol(A)	Choleski factorization of **A**. Returns the upper triangular factor, such that **R'R = A**.
y <- qr(A)	QR decomposition of **A**. **y$qr** has an upper triangle that contains the decomposition and a lower triangle that contains information on the Q decomposition. **y$rank** is the rank of A. **y$qraux** a vector which contains additional information on Q. **y$pivot** contains information on the pivoting strategy used.
cbind(A,B,...)	Combine matrices(vectors) horizontally. Returns a matrix.
rbind(A,B,...)	Combine matrices(vectors) vertically. Returns a matrix.
rowMeans(A)	Returns vector of row means.
rowSums(A)	Returns vector of row sums.
colMeans(A)	Returns vector of column means.
colSums(A)	Returns vector of coumn means.

Histogram

```
m<-mean(DFP$CTR, na.rm=TRUE)
std<-sqrt(var(DFP$CTR, na.rm=TRUE))
hist(DFP$CTR, density=20, prob=TRUE,
    main="Histogram with normal curve")
curve(dnorm(x, mean=m, sd=std), add=TRUE)
max<- max(DFP$CTR)
min<- min(DFP$CTR)
```

Sort

```
# sort by mpg
newdata <- mtcars[order(mpg),]
```

Saving/Loading Objects

```
#Simulating Data
x<- rnorm(100,0,1)
e<- rnorm(100,0,5)
y<- .5*x + e
z<- lm(y~x)

#Saving it
save(z, file='mymodel.rda')
load(file = "mymodel.rda")

#an alternative/better way
saveRDS(z, 'mymodel.rds')
mod2<- readRDS('mymodel.rds')
```

Descriptive Statistics

```
m<-mean(DFP$CTR, na.rm = TRUE)
std<-sqrt(var(DFP$CTR, na.rm = TRUE))
std<-sd(DFP$CTR, na.rm = TRUE)
hist(DFP$CTR, density=20, prob=TRUE,
          main="Histogram with normal curve")
curve(dnorm(x, mean=m, sd=std), add=TRUE)
p<- ecdf(DFP$CTR, na.rm = TRUE)
plot.ecdf(DFP$CTR, na.rm = TRUE)
Call: ecdf(DFP$CTR, na.rm = TRUE)
max<- max(DFP$CTR, na.rm = TRUE)
min<- min(DFP$CTR, na.rm = TRUE)
```

Plotting two variables

```
plot(x,y)
abline(a=0, b=1) # a is the intercept and b is the slope
```

Simulating AR(1) w. Sin

```
x  <- vector(length=500)
e  <- rnorm(500)
x[1]   <- 5
for(i in 2:length(x))
{
  x[i]  <- .4*x[i-1] + e[i]
}
plot(x, type="o", col="blue")
 g<-  rep(c(1,.7,.4,.1,0,0,.5,.2,.1,0),times=20,each=1)
 plot.ts(g)
 spec.pgram(g,taper=.5,log="no")
```

Simulating Logit Model

```
#create data:
x1 = rnorm(1000) # some continuous variables
x2 = rnorm(1000)
z = 1 + 2*x1 + 3*x2 # linear combination with a bias
pr = 1/(1+exp(-z)) # pass through an inv-logit function
y = pr > 0.5 # take as '1' if probability > 0.5
```

Simulating Categorical Model

```
n <- 10000
blah <- character(n)
x1 = rnorm(1000) # some continuous variables
x2 = rnorm(1000)
z = 1 + 2*x1 + 3*x2 + runif(n) # linear combination with a bias
pr = 1/(1+exp(-z))  # pass through an inv-logit function
blah[u<=0.1] <- "A"
blah[u>0.1 & u<=0.3] <- "B"
blah[u>0.3 & u<=0.95] <- "C"
blah[u>0.95] <- "D"
table(blah)
prop.table(summary(as.factor(blah)))
```

Simulating Hierarchical Linear Model

```
# HLM is a common tool used to analyze hierarchically structured data.
# Let's see it in action using a couple of the tools programmed up by HLM researchers.
# First let's generate a data set.
# Imagine we have a two level model, students nested within classrooms.
# Let's say we have 20 classrooms
nclass = 20
# And thirty students per classroom
nstud = 30
```

```
# Let's imagine that we have a classroom effect that varies randomly and is uncorrelated with
#the student level effect.
class.effect = rnorm(nclass)*2
# Imagine also we have a unique outcome per student.
# We have nclass*nstud number of students in total.
student.effect = rnorm(nstud*nclass)*3
# Now we have all the data we need in order to generate our analysis.
data.set = data.frame(class.id = rep(1:nclass, each=nstud),
                      class.effect = rep(class.effect, each=nstud),
                      student.id = 1:(nstud*nclass),
                      student.effect = student.effect)
data.set$outcomes = data.set$class.effect + data.set$student.effect
head(data.set)
lme(outcomes ~ 1, random = ~ 1 | class.id, data=data.set)
```

Creating and Plotting a matrix(2)

```
x<- rnorm(200,1,1)
y<- rnorm(200,1,5)
dat <- matrix(c(x,y),nrow=200, ncol=2)
matplot(dat, type = c("b"),pch=1,col = 1:2)
legend("topleft", legend = 1:2, col=1:2, pch=1)
```

Replacing Na's with Blanks

```
Ajit <- sapply(Ajit, as.character) #Ajit is the name of the data file
Ajit[is.na(Ajit)] <- ""
```

Converting Character Matrix to Numeric

```
d1 <- data.frame(Ajit))
mode(d1[,1])
is.factor(d1[,1])
```

Bar Chart

```
df<- within(Support_Hour, Hour <- factor(Support_Hour$Hour,
                                         levels = c("12AM",
                                                    "1AM",
                                                    "2AM",
                                                    "3AM",
                                                    "4AM",
                                                    "5AM",
                                                    "6AM",
```

```
                                        "7AM",
                                        "8AM",
                                          "9AM",
                                        "10AM",
                                        "11AM",
                                        "12PM",
                                       "1PM",
                                        "2PM",
                                        "3PM",
                                        "4PM",
                                        "5PM",
                                        "6PM",
                                        "7PM",
                                        "8PM",
                                        "9PM",
                                         "10PM",
                                        "11PM"),
                                ordered=TRUE))

temp = aggregate(list(Tickets = df$Tickets), list(Day = factor(df$Day)), mean)  # this
charts the averages#

a<- ggplot(data = temp, aes(x = Day, y = Tickets)) +
  geom_bar(aes(fill=Day),stat="identity")
a<- a + theme(
  panel.grid.minor = element_blank(),
  panel.grid.major = element_line(colour = "Red"),
  panel.background = element_rect(fill = "Black"),
  axis.title.x = element_text(face="bold", colour="Red", size=20),
  axis.title.y = element_text(face="bold", colour="Blue", size=20),
  axis.text.x  = element_text(face="bold", colour="Black", size=10),
  axis.text.y  = element_text(face="bold", colour="Black", size=12)
)
a
```

Scatter plot

```
a<- ggplot(data = temp, aes(x = Hour, y = Tickets))
a <- a + geom_line(colour="Blue") + geom_point(size=2, colour="Red")
a <- a + xlab("Time") + ylab("Tickets") + ggtitle("Tickets over Time")
```

```
a<- a + theme(
  panel.grid.minor = element_blank(),
  panel.grid.major = element_line(colour = "Grey"),
  panel.background = element_rect(fill = "White"),
  axis.title.x = element_text(face="bold", colour="Red", size=20),
  axis.title.y = element_text(face="bold", colour="Blue", size=20),
  axis.text.x  = element_text(face="bold", colour="Black", size=10),
  axis.text.y  = element_text(face="bold", colour="Black", size=12)
)
a
```

DyGraphs

```
y <- read.zoo("Trend.csv", sep = ",", header = TRUE, format = "%m/%d/%Y",
            fill = TRUE, colClasses = rep(NA, 5))
n <- as.xts(y)
f<- cbind(n$Long, n$Longer, n$Longest)
dygraph(f, main = "Trend for Tickets Open Longer than 3 Days") %>%
dyHighlight(highlightCircleSize = 5,
                                                    highlightSeriesBackgroundAlpha =
0.75,
                                                    hideOnMouseOut = TRUE) %>%
dyShading(from = "2015-05-14", to = "2015-06-14", color = "#FFE6E6") %>%
dyShading(from = "2015-03-02", to = "2015-05-14", color = "#CCEBD6") %>%
dySeries("Long", label = "Tickets Open for longer than 3 Days", strokeWidth = 2) %>%
dySeries("Longer", label = "Tickets open long than a Week", strokeWidth = 2)  %>%
dySeries("Longest", label = "Tickets Open longer than 2 Weeks", strokeWidth = 2) %>%
dyOptions(stackedGraph = FALSE) %>% dyAxis("y", label = "%") %>%  dyAxis("y2",
label = "Avg Threads", independentTicks = TRUE) %>% dyRangeSelector()
```

Check for Updates

```
check.for.updates.R(notify_user = TRUE, use_GUI = TRUE,
                page_with_download_url = "http://cran.rstudio.com/bin/windows/base/",
                pat = "R-[0-9.]+-win")
updateR(T, T, T, T, T, T, T)
```

Classification Simulation

```
set.seed(12345)
state<- c()
```

```
county<- c()
republican<- c()
own.status<- c()
incumbent<- c()

for(i in 1:4) {

  republican.s<- runif(1, min=.4, max=.6)
  culture.s<- rnorm(1, mean = 0, sd = .5)

  if(i==1){ c.start<-1}else{c.start<- county[length(county)]}

  for(j in c.start:(c.start + round(runif(1, min=10, max=60)))) {
    republican.c<- runif(1, min=-.4, max=.4)
    culture.c<- rnorm(1, mean=culture.s, sd=.25)
    N= runif(1, min=5, max=100)

    state<- c(state, rep(i,N))
    county<- c(county, rep(j,N))

    republican.T<- rbinom(N, size=1, prob=(republican.s + republican.c))
    own.status.T<- runif(N, min=0, max=100)

    xb<- -2 + culture.c + 1.2*republican.T + .05*own.status.T
    pr.inc<- pnorm(xb)

    incumbent.T<- rbinom(N, size=1, prob=pr.inc)

    republican<- c(republican, republican.T)
    own.status<- c(own.status, own.status.T)
    incumbent<- c(incumbent,incumbent.T)

  }

}

dat.example<- data.frame(incumbent, republican, own.status, state, county)
simple.model<- glm(incumbent~ republican + own.status, family=binomial(link="logit"),data=dat.example)
summary(simple.model)
```

Creating Panel Data

```
library(reshape2)
library(tidyr)
aql <- melt(CRO, id.vars = c("College"))
# if the above doesn't work
tiddy<- gather(CRO, Student)
```

```
v<- aql[order(aql[,1]),] # Sorts it by Column 1
```

Bootstrap

```
sampler <- function(dat, clustervar, replace = TRUE, reps = 1) {
  cid <- unique(dat[, clustervar[1]])
  ncid <- length(cid)
  recid <- sample(cid, size = ncid * reps, replace = TRUE)
  if (replace) {
    rid <- lapply(seq_along(recid), function(i) {
      cbind(NewID = i, RowID = sample(which(dat[, clustervar] == recid[i]),
                                      size = length(which(dat[, clustervar] == recid[i])), replace = TRUE))
    })
  } else {
    rid <- lapply(seq_along(recid), function(i) {
      cbind(NewID = i, RowID = which(dat[, clustervar] == recid[i]))
    })
  }
  dat <- as.data.frame(do.call(rbind, rid))
  dat$Replicate <- factor(cut(dat$NewID, breaks = c(1, ncid * 1:reps), include.lowest = TRUE,
                              labels = FALSE))
  dat$NewID <- factor(dat$NewID)
  return(dat)
}

#Resampling the data set including the rep variables, the "DID" is the grouping variable
set.seed(20)
tmp <- sampler(hdp, "DID", reps = 5)
bigdata <- cbind(tmp, hdp[tmp$RowID, ])
```

Simulating Hierarchical Data_2:

```
# Simulation 1
generate_data = function(
  n # number of units
  , k # number of trials within each condition within each unit
  , noise # measurement noise variance
  , I # population intercept
  , vI # across-units variance of intercepts
  , A # population A effect
  , vA # across-units variance of A effects
```

```
  , rIA # across-units correlation between intercepts and A effects
){
  Sigma = c(
          vI , sqrt(vI*vA)*rIA
          , sqrt(vI*vA)*rIA , vA
  )
  Sigma = matrix(Sigma,2,2)
  means = mvrnorm(n,c(I,A),Sigma)
  temp = expand.grid(A=c('a1','a2'),value=0)
  temp$A = factor(temp$A)
  contrasts(temp$A) = contr.sum
  from_terms = terms(value~A)
  mm = model.matrix(from_terms,temp)
  data = expand.grid(A=c('a1','a2'),unit=1:n,trial=1:k)
  for(i in 1:n){
  data$value[data$unit==i] = as.numeric(mm %*% means[i,]) + rnorm(k*2,0,sqrt(noise))
  }
  data$unit = factor(data$unit)
  data$A = factor(data$A)
  contrasts(data$A) = contr.sum
  return(data)
}

this_data = generate_data(
  n = 20 # number of units
  , k = 10 # number of trials within each condition within each unit
  , noise = 1 # measrurement noise variance
  , I = 2 # population intercept
  , vI = 3 # across-units variance of intercepts
  , A = 4 # population A effect
  , vA = 5 # across-units variance of A effects
  , rIA = .6 # across-units correlation between intercepts and A effects
)

fit = lmer(
  data = this_data
  , formula = value ~ (1+A|unit) + A
)

#Simulation2
# HLM is a common tool used to analysize hierarchically structured data.

# Let's see it in action using a couple of the tools programmed up by HLM researchers.
```

```
# First let's generate a data set.

# Imagine we have a two level model, students nested within classrooms.

# Let's say we have 20 classrooms

nclass = 20

# And thirty students per classroom

nstud = 30

# Let's imagine that we have a classroom effect that varies randomly and is uncorrelated with the
student level effect.

class.effect = rnorm(nclass)*2

# Imagine also we have a unique outcome per student.

# We have nclass*nstud number of students in total.

student.effect = rnorm(nstud*nclass)*3

# Now we have all the data we need in order to generate our analysis.
data.set = data.frame(class.id = rep(1:nclass, each=nstud),
                      class.effect = rep(class.effect, each=nstud),
                      student.id = 1:(nstud*nclass),
                      student.effect = student.effect)

data.set$outcomes = data.set$class.effect + data.set$student.effect

head(data.set)
# Looking good.  Now let's load our HLM package.


dat<- data.frame(Mice1$Order,Mice1$GPA,Mice1$SAT,
Mice1$four,Mice1$six,Mice1$Freshman,
                 Mice1$Admitted,
                 Mice1$ACT,Mice1$Open,Mice1$A,Mice1$Stem,Mice1$E)
colnames(dat) <- c("Order","GPA","SAT","four","six","Freshman",
                   "Admitted","ACT","Open","A","Stem","E")
```

OLS

w<- lm(y ~ x, data = A)

Residual Plots

```
# simulated data, no relationship
df1 <- data.frame(y=rnorm(100), x1=rnorm(100), x2=rnorm(100),
          x3=rnorm(100), x4=rnorm(100))
fit1 <- lm( y ~ ., data=df1 )
#plot(df1$y, fitted(fit1), asp=1)
scatter.smooth(df1$y, fitted(fit1), asp=1)
abline(0,1)
abline(h=mean(fitted(fit1)), col='lightgrey')
plot(df1$y, resid(fit1))
abline(h=0)

plot(fitted(fit1), resid(fit1))
abline(h=0)

# simulated data, relationship
library(MASS)
df2 <- as.data.frame( mvrnorm(100, mu=1:5, Sigma= matrix(.7,5,5)+diag(rep(.3,5))))
names(df2) <- c('y','x1','x2','x3','x4')

fit2 <- lm( y ~ ., data=df2 )
#plot(df2$y, fitted(fit2), asp=1)
scatter.smooth(df2$y, fitted(fit2), asp=1)
abline(0,1)
abline(h=mean(fitted(fit2)), col='lightgrey')
plot(df2$y, resid(fit2))
abline(h=0)

plot(fitted(fit2), resid(fit2))
abline(h=0)

# real data
fit3 <- lm( Murder~Population+Income+Illiteracy+Frost, data=as.data.frame(state.x77))

scatter.smooth( state.x77[,'Murder'], fitted(fit3), asp=1)
abline(0,1)
abline(h=mean(fitted(fit3)), col='lightgrey')
plot(state.x77[,'Murder'], resid(fit3))
abline(h=0)

plot(fitted(fit3), resid(fit3))
abline(h=0)

# Normality of Residuals
```

```
# qq plot for studentized resid
qqPlot(fit, main="QQ Plot")
# distribution of studentized residuals
sresid <- studres(fit)
hist(sresid, freq=FALSE,
    main="Distribution of Studentized Residuals")
xfit<-seq(min(sresid),max(sresid),length=40)
yfit<-dnorm(xfit)
lines(xfit, yfit)
probDist <- pnorm(sresid)
plot(ppoints(length(sresid)), sort(probDist), main = 'PP Plot', xlab = 'Observed Probability', ylab = 'Expected
Probability', col='blue')
abline(0,1, col='red')
```

Variance Test and T Test

```
var.test(a,b)

t.test(Review$Sixup_GPA,Review$Purdue_GPA, var.equal=FALSE, paired=FALSE)
```

Normality Test

```
ad.test(x)
cvm.test(x)
lillie.test(x)
pearson.test(x)
```

KS Two Sample test

```
ks.test(Review$Sixup, Review$Purdue_GPA, alternative = c('two.sided'))
```

Durbin Watson Test

```
## generate two AR(1) error terms with parameter
## rho = 0 (white noise) and rho = 0.9 respectively
err1 <- rnorm(100)
## generate regressor and dependent variable
x <- rep(c(-1,1), 50)
y1 <- 1 + x + err1
plot.ts(y1)
## perform Durbin-Watson test
dwtest(y1 ~ x, alternative = c(‘greater’,’two-sided’,’less’))
err2 <- filter(err1, 0.9, method="recursive")
y2 <- 1 + x + err2
dwtest(y2 ~ x)
```

IV Reg

```
ivreg2 <- function(form,endog,iv,data,digits=3){
  # library(MASS)
  # model setup
  r1 <- lm(form,data)
  y <- r1$fitted.values+r1$resid
```

```
  x <- model.matrix(r1)
  aa <- rbind(endog == colnames(x),1:dim(x)[2])
  z <- cbind(x[,aa[2,aa[1,]==0]],data[,iv])
  colnames(z)[(dim(z)[2]-length(iv)+1):(dim(z)[2])] <- iv
  # iv  and standard errors
  z <- as.matrix(z)
  pz <- z %*% (solve(crossprod(z))) %*% t(z)
  biv <- solve(crossprod(x,pz) %*% x) %*% (crossprod(x,pz) %*% y)
  sigiv <- crossprod((y - x %*% biv),(y - x %*% biv))/(length(y)-length(biv))
  vbiv <- as.numeric(sigiv)*solve(crossprod(x,pz) %*% x)
  res <- cbind(biv,sqrt(diag(vbiv)),biv/sqrt(diag(vbiv)),(1-pnorm(biv/sqrt(diag(vbiv))))*2)
  res <-
matrix(as.numeric(sprintf(paste("%.",paste(digits,"f",sep=""),sep=""),res)),nrow=dim(res)[1])
  rownames(res) <- colnames(x)
  colnames(res) <- c("Coef","S.E.","t-stat","p-val")
  # First-stage F-test
  y1 <- data[,endog]
  z1 <- x[,aa[2,aa[1,]==0]]
  bet1 <- solve(crossprod(z)) %*% crossprod(z,y1)
  bet2 <- solve(crossprod(z1)) %*% crossprod(z1,y1)
  rss1 <- sum((y1 - z %*% bet1)^2)
  rss2 <- sum((y1 - z1 %*% bet2)^2)
  p1 <- length(bet1)
  p2 <- length(bet2)
  n1 <- length(y)
  fs <- abs((rss2-rss1)/(p2-p1))/(rss1/(n1-p1))
  firststage <- c(fs)
  firststage <-
matrix(as.numeric(sprintf(paste("%.",paste(digits,"f",sep=""),sep=""),firststage)),ncol=length(firsts
tage))
  colnames(firststage) <- c("First Stage F-test")
  # Hausman tests
  bols <- solve(crossprod(x)) %*% crossprod(x,y)
  sigols <- crossprod((y - x %*% bols),(y - x %*% bols))/(length(y)-length(bols))
  vbols <- as.numeric(sigols)*solve(crossprod(x))
  sigml <- crossprod((y - x %*% bols),(y - x %*% bols))/(length(y))
  x1 <- x[,!(colnames(x) %in% "(Intercept)")]
  z1 <- z[,!(colnames(z) %in% "(Intercept)")]
  pz1 <- z1 %*% (solve(crossprod(z1))) %*% t(z1)
  biv1 <- biv[!(rownames(biv) %in% "(Intercept)"),]
  bols1 <- bols[!(rownames(bols) %in% "(Intercept)"),]
  # Durbin-Wu-Hausman chi-sq test:
```

```
  # haus <- t(biv1-bols1) %*% ginv(as.numeric(sigml)*(solve(crossprod(x1,pz1) %*%
x1)-solve(crossprod(x1)))) %*% (biv1-bols1)
  # hpvl <- 1-pchisq(haus,df=1)
  # Wu-Hausman F test
  resids <- NULL
  resids <- cbind(resids,y1 - z %*% solve(crossprod(z)) %*% crossprod(z,y1))
  x2 <- cbind(x,resids)
  bet1 <- solve(crossprod(x2)) %*% crossprod(x2,y)
  bet2 <- solve(crossprod(x)) %*% crossprod(x,y)
  rss1 <- sum((y - x2 %*% bet1)^2)
  rss2 <- sum((y - x %*% bet2)^2)
  p1 <- length(bet1)
  p2 <- length(bet2)
  n1 <- length(y)
  fs <- abs((rss2-rss1)/(p2-p1))/(rss1/(n1-p1))
  fpval <- 1-pf(fs, p1-p2, n1-p1)
  #hawu <- c(haus,hpvl,fs,fpval)
  hawu <- c(fs,fpval)
  hawu <-
matrix(as.numeric(sprintf(paste("%.",paste(digits,"f",sep=""),sep=""),hawu)),ncol=length(hawu))
  #colnames(hawu) <- c("Durbin-Wu-Hausman chi-sq test","p-val","Wu-Hausman F-test","p-val")
  colnames(hawu) <- c("Wu-Hausman F-test","p-val")
  # Sargan Over-id test
  ivres <- y - (x %*% biv)
  oid <- solve(crossprod(z)) %*% crossprod(z,ivres)
  sstot <- sum((ivres-mean(ivres))^2)
  sserr <- sum((ivres - (z %*% oid))^2)
  rsq <- 1-(sserr/sstot)
  sargan <- length(ivres)*rsq
  spval <- 1-pchisq(sargan,df=length(iv)-1)
  overid <- c(sargan,spval)
  overid <-
matrix(as.numeric(sprintf(paste("%.",paste(digits,"f",sep=""),sep=""),overid)),ncol=length(overid))
  colnames(overid) <- c("Sargan test of over-identifying restrictions","p-val")
  if(length(iv)-1==0){
    overid <- t(matrix(c("No test performed. Model is just identified")))
          colnames(overid) <- c("Sargan test of over-identifying restrictions")
  }
  full <- list(results=res, weakidtest=firststage, endogeneity=hawu, overid=overid)
  return(full)
}

ivreg2(form= GPA ~ GPA_1 + Credits + DiffCred + Probation + HA,
```

```
      endog="GPA_1",iv=c("Credits_2"),data=na.omit(Cmodel))
```

Non-linear Least Squares

```
A = 0
B = 0
C = 1
D = -1
E = 1
fit = nls(LS ~ A + B*ICPM^C + D*ICPM^E , data = TM, start=list(A=A,B=B, C=C, D=D, E=E),
control = list(maxiter = 5000))
summary(fit)
```

Spectral Analysis

```
x  <- vector(length=200)
e  <- rnorm(200,0,5)
x[1]   <- 0
t<- seq(0, 200 , 1)
for(i in 2:length(x))
{
  x[i]  <- .5*x[i-1] + 2*abs(cos(.05*t[i]*pi)) + e[i]
}
  plot.ts(x)

sp<- spec.pgram(x,c(5,5),taper=0,log="no")
```

Wavelets

```
x  <- vector(length=1000)
e  <- rnorm(1000,0,5)
v  <- rnorm(1000,0,5)
w  <- rnorm(1000,0,5)
x[1]   <- 5
t<- seq(1, 1000, 1)
for(i in 2:length(x))
{
  x[i]  <- .02*(t[i] + v[i]) + .75*(x[i-1] + w[i]) + 5*abs(cos(.02*(t[i]*pi))) + 10*abs(cos(.1*(t[i]*pi))) +
e[i]
}

my.data <- data.frame( x = x)

plot(x, type="o", col="blue")

my.w<- analyze.wavelet(my.data, "x",
                       loess.span=.35,
```

```
                 dt = 1, dj = 1/250,
                lowerPeriod = 8,
                upperPeriod = 128,
                make.pval = T,
                n.sim = 10)

wt.image(my.w, color.key = "quantile", n.levels = 250,
         legend.params = list(lab= "wavelet power levels", mar = 4.7))


reconstruct(my.w, plot.waves = F, lwd = c(1,2) , legend.coords = "bottomleft")
```

Dynamic Linear Models

Yt = Ftθt + vt, vt ~ Nm(0, Vt),
θt = Gtθt−1 + wt, wt ~ Np(0, Wt),
θ0 ~ Np(m0, C0),

#Random Walk w/ constant
Yt = μt + vt, vt ~ N(0, V)
μt = μt−1 + wt, wt ~ N(0, W)
/w Ft = Gt = 1

```
x<- dlm(FF = 1, V = 0.8, GG = 1, W = 0.1, m0 = 0, C0 = 10)
```

#Multiple Random Walks /w constant
Yt = μt + vt, vt ~ N(0, V)
μt = μt−1 + βt−1 + w1,t, w1,t ~ N(0, $\sigma 2_{w1}$)
βt = βt−1 + w2,t, w2,t ~ N(0, $\sigma 2_{w2}$)

θt = (μt βt) , G = (1 1 0 1) , W = ($\sigma 2_{w1}$ 0 0 $\sigma 2_{w2}$) , F = (1 0)

```
lg <- dlm(m0 = rep(0,2), C0 = 10 * diag(2), FF = matrix(c(1,0),nr=1),
         V = 1.4, GG = matrix(c(1,0,1,1),nr=2), W = diag(c(0,0.2)))
```

#Random Walk with time-varying

Yt = Ft*μt + vt, vt ~ N(0, V)
μt = μt−1 + βt−1 + w1,t, w1,t ~ N(0, $\sigma 2_{w1}$)
βt = βt−1 + w2,t, w2,t ~ N(0, $\sigma 2_{w2}$)

```
x <- rnorm(100) # covariates
dlr <- dlm(m0 = rep(0,2), C0 = 10 * diag(2), FF = matrix(c(1,0),nr=1),
          V = 1.3, GG = diag(2), W = diag(c(0.4,0.2)),
          JFF = matrix(c(0,1),nr=1), X = as.matrix(x))
x
```

#Kalman Filter with a random walk time series and SVD(singular value decomposition):

```
mod <- dlmModPoly(order = 1, dV = 15100, dW = 1468) # dv corresponds to Yt series,
dW corresponds the parameters series#

y  <- vector(length=100)
e  <- rnorm(100,0,1)
y[1]   <- 0
for(i in 2:length(y))
{
  y[i]  <- y[i-1] + e[i]
}

mod <- dlmModPoly(order = 1, dV = 15100, dW = 1468)

modFilt <- dlmFilter(y, mod)
str(modFilt,1)

plot.ts(y)
plot.ts(modFilt$m)
```

#Comparing two Kalman Filters with a random walk time series and SVD(singular value
#decomposition):

```
y  <- vector(length=100)
e  <- rnorm(100,0,1)
y[1]   <- 0
for(i in 2:length(y))
{
```

```
  y[i]  <- y[i-1] + e[i]
}

mod1 <- dlmModPoly(order = 1, dV = 15100, dW = 0.5 * 1468)
yFilt1 <- dlmFilter(y, mod1)
plot(window(cbind(y,yFilt1$m[-1]),start=start(y)+1), plot.type='s',
     type='o', col=c("grey","green"), lty=c(1,2), xlab="", ylab="Level")
mod2 <- dlmModPoly(order = 1, dV = 15100, dW = 5 * 1468)
yFilt2 <- dlmFilter(y, mod2)
lines(window(yFilt2$m,start=start(y)+1), type='o', col="red", lty=4)
legend("bottomleft", legend=c("data", "filtered level - model 1",
                              "filtered level - model 2"),
       col=c("grey", "green", "red"), lty=c(1,2,4), pch=1, bty='n')



#Kalman Filter Smoothing with a random walk time series and SVD(singular value
#decomposition):

y  <- vector(length=100)
e  <- rnorm(100,0,1)
y[1]   <- 0
for(i in 2:length(y))
{
  y[i]  <- y[i-1] + e[i]
}

mod <- dlmModPoly(order = 1, dV = 15100, dW = 1468)

modFilt <- dlmFilter(y, mod)
modSmooth <- dlmSmooth(modFilt)
str(modSmooth,1)

plot.ts(modSmooth$s)
plot.ts(y)
plot.ts(modFilt$m)

#Gibbs DLM
x1  <- vector(length=200)
```

```
y <- vector(length=200)
e1 <- rnorm(200,0,1)
e2 <- rnorm(200,0,1)
t<- seq(1, 200,1)
x1[0]=.5
y[0] = 1
for(i in 2:length(x1))
  for(i in 2:length(y))
  {
          x1[i] <- -.4*x1[i-1] + e1[i]
          y[i]<- .2*t[i] + .8*x1[i-1] + e2[i]
          }
set.seed(5)
mcmc <- 100
burn <- 1
outGibbsIRW <- dlmGibbsDIG(y,mod =dlmModPoly(1) + dlmModReg(x1),
                      shape.y = 1e-3, rate.y = 1e-3,
                      shape.theta = 1e-3, rate.theta = 1e-3, n.sample = mcmc + burn)
tail(outGibbsIRW$dV, n=5)
tail(outGibbsIRW$dW, n=5)
dv<- outGibbsIRW$dW[100]
dw<- outGibbsIRW$dW[101,]


mod<- dlm(m0=rep(0,3),C0 = 1000 * diag(3),FF = matrix(c(1,1,1),nr=1),
          V=dv,GG = matrix(c(1,0,0,0,1,0,0,0,1), nrow=3),W = diag(c(dw))
)
smooth <- dlmSmooth(y,mod)
s<- smooth$s[,1]
 plot.ts(s, col = 'blue')
lines(y)
```

MICE - Multiple Imputation For Missing Data

```
rm(list=ls())
set.seed(1222)

x.t<- runif(200)
z.t<- runif(200)
y.t<- x.t + z.t + rnorm(200,sd=.5)
```

```
miss.x<- rbinom(200,1,prob=0.9)
miss.z<- rbinom(200,1,prob=0.9)
miss.y<- rbinom(200,1,prob=0.9)

x<- ifelse(miss.x==1,x.t,NA)
z<- ifelse(miss.z==1,z.t,NA)
y<- ifelse(miss.y==1,y.t,NA)

dat<- data.frame(x,y,z)

mi.dat<- mice(dat, m=20, maxit=50)

fit<- with(mi.dat, lm(y~x + z))
summary(pool(fit))
summary(lm(y~x+z,dat=dat))
```

GMM

```
g1 <- function(tet,x)
  {
    m1 <- (tet[1]-x)
    m2 <- (tet[2]^2 - (x - tet[1])^2)
    m3 <- x^3-tet[1]*(tet[1]^2+3*tet[2]^2)
    f <- cbind(m1,m2,m3)
    return(f)
    }
Dg <- function(tet,x)
{
G <- matrix(c( 1,
               2*(-tet[1]+mean(x)),
               -3*tet[1]^2-3*tet[2]^2,0,
               2*tet[2],-6*tet[1]*tet[2]),
             nrow=3,ncol=2)
return(G)
}

set.seed(123)
n <- 200
x1 <- rnorm(n, mean = 4, sd = 2)

gmm(g = g1, x = x1, t0 = c(mu = 0, sig = 0), gradv = Dg)
```

GMM LOGIT

```
init <- glm(PP ~ Home + GS + Autocorrelation
```

```
        + DD + BACK + FRONT + MID + RED1, family = binomial(link = "logit"), na.action =
na.pass, data=Logit)

summary(native)

logistic <- function(theta, data) {
  return(1/(1 + exp(-data*theta)))
}

dat <- data.matrix(cbind(Logit$PP, 1, Logit$Home, Logit$GS, Logit$Autocorrelation, Logit$DD,
Logit$BACK, Logit$FRONT, Logit$MID,
                    Logit$RED1))

moments <- function(theta, data) {
  y <- as.numeric(data[, 1])
  x <- data.matrix(data[, 2:11])
  m <- x * as.vector((y - logistic(theta, x)))
  return(cbind(m))
}

init <- glm(PP ~ Home + GS + Autocorrelation
        + DD + BACK + FRONT + MID + RED1, family = binomial(link = "logit"), na.action =
na.pass, data=Logit)$


my_gmm <- gmm(moments, x = Logit, t0 = init, type = "iterative", crit = 1e-25, wmatrix =
"optimal", method = "Nelder-Mead", control = list(reltol = 1e-25, maxit = 20000))
summary(my_gmm)
```

Difference between Panel and Mixed Models

Both panel data and mixed effect model data deal with double indexed random variables y_{it}. First index is for group, the second is for individuals within the group. For the panel data the second index is usually time, and it is assumed that we observe individuals over time. When time is second index for mixed effect model the models are called longitudinal models. The

mixed effect model is best understood in terms of 2 level regressions. (For ease of exposition assume only one explanatory variable)

First level regression is the following

$$y_{ij} = a_i + \beta_i x_{ij} + \varepsilon_{ij}$$

This is simply explained as individual regression for each group. The second level regression tries to explain variation in regression :

$$a_i = \gamma_0 + \gamma_1 z_{i1} + u_i$$

$$\beta_i = \delta_0 + \delta_1 z_{i2} + v_i$$

When you substitute the second equation to the first one you get

$$y_{ij} = \gamma_0 + \gamma_1 z_{i1} + \delta_0 x_{ij} + \delta_1 x_{ij} z_{i2} + u_i + v_i x_{ij} + \varepsilon_{ij}$$

The fixed effects are what is fixed, this means $\gamma_0, \gamma_1, \delta_0, \delta_1$. The random effects are u_i and v_i .

Now for panel data the terminology changes, but you still can find common points. The panel data random effects models is the same as mixed effects model with

$$a_i = \gamma_0 + u_i$$

$$\beta_i = \delta_0$$

with model becoming

$$y_{it} = \gamma_0 + \delta_0 x_{it} + u_i + \varepsilon_{it},$$

where u_i are random effects.

The most important difference between mixed effects model and panel data models is the treatment of regressors x_{it}. For mixed effects models they are non-random variables, whereas for panel data models it is always assumed that they are random. This becomes important when stating what is fixed effects model for panel data.

For mixed effect model it is assumed that random effects u_i and v_{it} are independent of ε_{it} and also from x_{it} and z_i, which is always true when x_{it} and z_i are fixed. If we allow for

stochastic x_{it} this becomes important. So the random effects model for panel data assumes that x_{it} is not correlated with u_i. But the fixed effect model which has the same form

$$y_{it} = \gamma_0 + \delta_0 x_{it} + u_i + \varepsilon_{it}$$

allows correlation of x_{it} and u_i. The emphasis then is solely for consistently estimating δ_0. This is done by subtracting the individual means:

$$y_{it} - \bar{y_i} = \delta_0 (x_{it} - \bar{x_i}) + \varepsilon_{it} - \bar{\varepsilon_i},$$

and using simple OLS on resulting regression problem. Algebraically this coincides with least square dummy variable regression problem, where we assume that uiui are fixed parameters. Hence the name fixed effects model.

There is a lot of history behind fixed effects and random effects terminology in panel data econometrics, which I omitted. In my personal opinion these models are best explained in Wooldridge's "Econometric analysis of cross section and panel data". As far as I know there is no such history in mixed effects model, but on the other hand I come from econometrics background, so I might be mistaken.

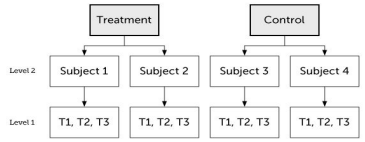
Hierarchical Linear Models - Random Effects

Longitudinal two-level model

#We will begin with the two-level model, where we have repeated measures on individuals in different #treatment groups.

Variables: subjects, tx, therapist, time, y

Subjects = subject id, therapist = cluster of therapist, tx = treatment allocation, y is the outcome variable



#Unconditional Model

Model formulation

$Y_{ij} = \beta_{0j} + R_{ij}$

$\beta_{0j} = \gamma_{00} + U_{0j}$

#With U_{0j} ~N(0, τ_{00}^2) and R_{ij} ~N(0, σ2)

z<- lmer(y ~ 1 + (1 | subjects), data = data)

#Unconditional Growth Model

$Y_{ij} = \beta_{0j} + \beta_{1j}\ t_{ij} + R_{ij}$

$\beta_{0j} = \gamma_{00} + U_{0j}$

$\beta_{1j} = \gamma_{10} + U_{1j}$

#(U_{0j}) ~N(0, τ_{00}^2 , τ_{01}^2)

#(U_{1j})~(0, τ_{01}^2 , τ_{10}^2)

R_{ij} ~N(0, σ2)

z<- lmer(y ~ time + (time | subjects), data = data)

#Conditional growth model

$Y_{ij} = \beta_{0j} + \beta_{1j}\ t_{ij} + R_{ij}$

$\beta_{0j} = \gamma_{00} + \gamma_{10} TX_j + U_{0j}$

$\beta_{1j} = \gamma_{10} + \gamma_{11} TX_j + U_{1j}$

#(U_{0j}) ~(0, τ_{00}^2 , τ_{01}^2)

#(U_{1j})~(0, τ_{01}^2 , τ_{10}^2)

R_{ij} ~(0, σ2)

z<- lmer(y ~ time * tx + (time | subjects), data = data)

#Conditional growth model: dropping random slope

$Y_{ij} = \beta_{0j} + \beta_{1j}\ t_{ij} + R_{ij}$

$\beta_{0j} = \gamma_{00} + \gamma_{10} TX_j + U_{0j}$

$\beta_{1j} = \gamma_{10} + \gamma_{11} TX_j$

U_{0j} ~N(0, τ_{00}^2) and R_{ij} ~N(0, σ2)

z<- lmer(y ~ time * tx + (1 | subjects), data = data)

#Conditional growth model: dropping random intercept

$Y_{ij} = \beta_{0j} + \beta_{1j}\ t_{ij} + R_{ij}$

$\beta_{0j} = \gamma_{00} + \gamma_{10} TX_j$

$\beta_{1j} = \gamma_{10} + \gamma_{11} TX_j + U_{1j}$

U_{0j} ~N(0, τ_{00}^2) and R_{ij} ~N(0, σ2)

z<- lmer(y ~ time * tx + (0 | subjects), data = data)

#Conditional growth model: dropping intercept and slope covariance

$Y_{ij} = \beta_{0j} + \beta_{1j}\ t_{ij} + R_{ij}$

$\beta_{0j} = \gamma_{00} + \gamma_{10} TX_j + U_{0j}$

$\beta_{1j} = \gamma_{10} + \gamma_{11} TX_j + U_{1j}$

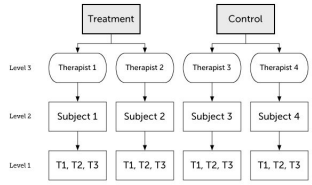
#(U_{0j}) ~(0, τ_{00}^2 , 0)

#(U_{1j})~(0, 0 , τ_{10}^2)

R_{ij} ~N(0, σ2)

z<- lmer(y ~ time * tx + (time | | subjects), data = data)

#Three Level Models



#Conditional three-level growth model

$Y_{ijk} = \beta_{0jk} + \beta_{1kj}\ t_{ijk} + R_{ijk}$

$\beta_{0jk} = \gamma_{00k} + U_{0jk}$

$\beta_{1jk} = \gamma_{10k} + U_{1jk}$

$\gamma_{00k} = \delta_{000} + \delta_{001}\, \mathrm{T}\, X_k + V_{0k}$

$\gamma_{10k} = \delta_{100} + \delta_{101}\, \mathrm{T}\, X_k + V_{1k}$

#(U_{0j}) ~(0, τ_{00}^2 , τ_{01}^2)

#(U_{1j})~(0, τ_{01}^2 , τ_{10}^2)

#(V_{0k}) ~(0, φ_{00}^2 , φ_{01}^2)

#(V_{1k})~(0, φ_{01}^2 , φ_{10}^2)

R_{ijk} ~N(0, σ2)

z<- lmer(y ~ time * tx + (time | therapist/subjects), data = data)

#Heteroskedasticity at level 1

#(R_{ij}| TX = 0) ~N(0, σ_0^2)

#(R_{ij}| TX = 1)~N(0, σ_1^2)

z<- lmer(y ~ time * tx + (time | subjects), data = data, weights = varIdent(form = ~ 1 | tx))

#(R_{ij}| TX = 0) ~ (0, σ_{00}^2, 0, 0)

(0, 0, σ_{01}^2 , 0)

(0, 0, 0, σ_{01}^2)

z<- lmer(y ~ time * tx + (time | subjects), data = data, weights = varIdent(form = ~ 1 | tx*time))

First Order AR(1)

z<- lmer(y ~ time * tx + (time | subjects), data = data, correlation = corAR1())

#Heterogeneous AR(1)

z<- lmer(y ~ time * tx + (time | subjects), data = data, weights = varIdent(form = ~ 1 | tx), correlation = corAR1())

Two level Hierarchical Linear Model

```
require(ggplot2)
require(GGally)
require(reshape2)
require(lme4)
require(compiler)
require(parallel)
require(boot)

# Two level logistic mixed effect
hdp <- read.csv("http://www.ats.ucla.edu/stat/data/hdp.csv")
hdp <- within(hdp, {
  Married <- factor(Married, levels = 0:1, labels = c("no", "yes"))
  DID <- factor(DID)
  HID <- factor(HID)
})

# estimate the model and store results in m

m <- glmer(remission ~ IL6 + CRP + CancerStage + LengthofStay + Experience +
             (1 | DID), data = hdp, family = binomial, control = glmerControl(optimizer = "bobyqa"),
           nAGQ = 10)
```

```
# print the mod results without correlations among fixed effects
print(m,corr=FALSE)

#The first part tells us the estimates are based on an adaptive Gaussian Hermite
#approximation of the likelihood. In particular we used 10 integration points
#As we use more integration points, the approximation becomes more accurate
#converging to the ML estimates; however, more points are more computationally
#demanding and can be extremely slow or even intractable with today's technology.
#To avoid a warning of nonconvergence, we specify a different optimizer with the argument
#control=glmerControl(optimizer="bobyqa").
#Although the model will produce nearly identical
#results without the new argument, we prefer to use models without such warnings.

se <- sqrt(diag(vcov(m)))
# table of estimates with 95% CI
(tab <- cbind(Est = fixef(m), LL = fixef(m) - 1.96 * se, UL = fixef(m) + 1.96 *
              se))

exp(tab)

#Bootstrap Function
sampler <- function(dat, clustervar, replace = TRUE, reps = 1) {
 cid <- unique(dat[, clustervar[1]])
 ncid <- length(cid)
 recid <- sample(cid, size = ncid * reps, replace = TRUE)
 if (replace) {
  rid <- lapply(seq_along(recid), function(i) {
   cbind(NewID = i, RowID = sample(which(dat[, clustervar] == recid[i]),
                                   size = length(which(dat[, clustervar] == recid[i])), replace = TRUE))
  })
 } else {
  rid <- lapply(seq_along(recid), function(i) {
   cbind(NewID = i, RowID = which(dat[, clustervar] == recid[i]))
  })
 }
 dat <- as.data.frame(do.call(rbind, rid))
 dat$Replicate <- factor(cut(dat$NewID, breaks = c(1, ncid * 1:reps), include.lowest = TRUE,
                             labels = FALSE))
 dat$NewID <- factor(dat$NewID)
 return(dat)
```

```
}

#Resampling the data set including the rep variables, the "DID" is the grouping variable
set.seed(20)
tmp <- sampler(hdp, "DID", reps = 5)
bigdata <- cbind(tmp, hdp[tmp$RowID, ])

f <- fixef(m)
r <- getME(m, "theta")
#Next we refit the model on the resampled data.
#First we store the estimates from our original model,
#which we will use as start values for the bootstrap models.
#Then we make a local cluster with 4 nodes (the number of processors on our machine;
#set to the number of processors you have on yours).
#Next, we export the data and load the lme4 package on the cluster.
#Finally, we write a function to fit the model and return the estimates.
#The call to glmer() is wrapped in try because not all models may converge
#on the resampled data. This catches the error and returns it, rather
#than stopping processing.

cl <- makeCluster(4)
clusterExport(cl, c("bigdata", "f", "r"))
clusterEvalQ(cl, require(lme4)

myboot <- function(i) {
 object <- try(glmer(remission ~ IL6 + CRP + CancerStage + LengthofStay +
 Experience + (1 | NewID), data = bigdata, subset = Replicate == i, family = binomial,
 nAGQ = 1, start = list(fixef = f, theta = r)), silent = TRUE)
 if (class(object) == "try-error")
 return(object)
 c(fixef(object), getME(object, "theta"))
}

#Now that we have the data, the local cluster, and the
#fitting function setup, we are ready to actually do the bootstrapping.
#To do this, we use the parLapplyLB function, which loops through every replicate,
#giving them out to each node of the cluster to estimate the models.
#The "LB" stands for load balancing, which means replicates are distributed
#as a node completes its current job. This is valuable because not all
#replicates will converge, and if there is an error and it happens early on,
#one node may be ready for a new job faster than another node.
```

```
#There is some extra communication overhead, but this is small
#compared to the time it takes to fit each model. The results from all
#nodes are aggregated back into a single list, stored in the object res.
#Once that is done, we can shut down the local cluster, which terminates
#the additional R instances and frees memory.

start <- proc.time()
res <- parLapplyLB(cl, X = levels(bigdata$Replicate), fun = myboot)
end <- proc.time()

# shut down the cluster
stopCluster(cl)

#Now that we have the bootstrap results,
#we can summarize them. First, we calculate the number of models that
#successfully converged.
#We do this by checking whether a particular result is numeric or not.
#Errors are not numeric, so they will be skipped.
#We can calculate the mean of the successes to see the proportion of replicates
#that converged and that we have results for.

# calculate proportion of models that successfully converged
success <- sapply(res, is.numeric)
mean(success)

#Next we convert the list of bootstrap results into a matrix,
#and then calculate the 2.5th and 97.5th percentiles for each parameter.
#Finally, we can make a table of the results, including the original estimates and
#standard errors, the mean bootstrap estimate (which is asymptotically equivalent
#to the original results, but may be biased for a small number of replicates,
#as in our case), and the bootstrapped confidence intervals. With these data,
#you could also calculate bias-corrected bootstrap confidence intervals if you
#wanted, although we only show the percentile CIs.

# combine successful results
bigres <- do.call(cbind, res[success])

# calculate 2.5th and 97.5th percentiles for 95% CI
(ci <- t(apply(bigres, 1, quantile, probs = c(0.025, 0.975))))

# All results
```

```
finaltable <- cbind(Est = c(f, r), SE = c(se, NA), BootMean = rowMeans(bigres),ci)
# round and print
round(finaltable, 3)

#Predicted probabilities and graphing

#In a logistic model, the outcome is commonly on one of three scales:
#Log odds (also called logits), which is the linearized scale
#Odds ratios (exponentiated log odds), which are not on a linear scale
#Probabilities, which are also not on a linear scale

#The logit scale is convenient because it is linearized, meaning that
#a 1 unit increase in a predictor results in a coefficient unit increase
#in the outcome and this holds regardless of the levels of the other predictors
#(setting aside interactions for the moment). A downside is the scale is
#not very interpretable. It is hard for readers to have an intuitive understanding
#of logits. Conversely, probabilities are a nice scale to intuitively understand
#the results; however, they are not linear. This means that a one unit increase
#in the predictor, does not equal a constant increase in the probability---
#the change in probability depends on the values chosen for the other predictors.
#In ordinary logistic regression, you could just hold all predictors constant,
#only varying your predictor of interest. However, in mixed effects logistic models,
#the random effects also bear on the results. Thus, if you hold everything constant,
#the change in probability of the outcome over different values of your predictor
#of interest are only true when all covariates are held constant and you are in
#the same group, or a group with the same random effect. The effects are conditional
#on other predictors and group membership, which is quite narrowing.
#An attractive alternative is to get the average marginal probability.
#That is, across all the groups in our sample (which is hopefully representative of
#your population of interest), graph the average change in probability of the
#outcome across the range of some predictor of interest.
# temporary data

tmpdat <- hdp[, c("IL6", "CRP", "CancerStage", "LengthofStay", "Experience",
                  "DID")]

summary(hdp$LengthofStay)

#sampling of values from a variable
jvalues <- with(hdp, seq(from = min(LengthofStay), to = max(LengthofStay), length.out = 100))
```

```
# calculate predicted probabilities and store in a list
pp <- lapply(jvalues, function(j) {
  tmpdat$LengthofStay <- j
  predict(m, newdata = tmpdat, type = "response")
})

# average marginal predicted probability across a few different Lengths of
# Stay
sapply(pp[c(1, 20, 40, 60, 80, 100)], mean)

 # get the means with lower and upper quartiles
plotdat <- t(sapply(pp, function(x) {
  c(M = mean(x), quantile(x, c(0.25, 0.75)))
}))

# add in LengthofStay values and convert to data frame
plotdat <- as.data.frame(cbind(plotdat, jvalues))

# better names and show the first few rows
colnames(plotdat) <- c("PredictedProbability", "Lower", "Upper", "LengthofStay")
head(plotdat)



# plot average marginal predicted probabilities
ggplot(plotdat, aes(x = LengthofStay, y = PredictedProbability)) + geom_line() +
  ylim(c(0, 1))



#We could also add the lower and upper quartiles. This information shows us the
#range in which 50 percent of the predicted probabilities fell.
ggplot(plotdat, aes(x = LengthofStay, y = PredictedProbability)) + geom_linerange(aes(ymin = Lower,
                                                                                      ymax = Upper)) + geom_line(size = 2) + ylim(c(0, 1))

#This is just the beginning of what can be done. For plots,
#it is useful to add more information.
#We could make the same average marginal predicted probabilities,
#but in addition to varying LengthofStay we could do it for each level of
#CancerStage.

# calculate predicted probabilities and store in a list
biprobs <- lapply(levels(hdp$CancerStage), function(stage) {
```

```
  tmpdat$CancerStage[] <- stage
  lapply(jvalues, function(j) {
    tmpdat$LengthofStay <- j
    predict(m, newdata = tmpdat, type = "response")
  })
})

# get means and quartiles for all jvalues for each level of CancerStage
plotdat2 <- lapply(biprobs, function(X) {
  temp <- t(sapply(X, function(x) {
    c(M=mean(x), quantile(x, c(.25, .75)))
  }))
  temp <- as.data.frame(cbind(temp, jvalues))
  colnames(temp) <- c("PredictedProbability", "Lower", "Upper", "LengthofStay")
  return(temp)
})

# collapse to one data frame
plotdat2 <- do.call(rbind, plotdat2)

# add cancer stage
plotdat2$CancerStage <- factor(rep(levels(hdp$CancerStage), each = length(jvalues)))

# show first few rows
head(plotdat2)

# graph it
ggplot(plotdat2, aes(x = LengthofStay, y = PredictedProbability)) +
  geom_ribbon(aes(ymin = Lower, ymax = Upper, fill = CancerStage), alpha = .15) +
  geom_line(aes(colour = CancerStage), size = 2) +
  ylim(c(0, 1)) + facet_wrap(~  CancerStage)

#Things look fairly bleak for the chances of a Stage IV lung cancer patient
#who was in the hospital 10 days having cancer in remission
#(please remember that these are simulated data).
#It also looks like the distribution is skewed.
#We can examine the distribution of predicted probabilities just for that group.

ggplot(data.frame(Probs = biprobs[[4]][[100]]), aes(Probs)) + geom_histogram() +
  scale_x_sqrt(breaks = c(0.01, 0.1, 0.25, 0.5, 0.75))
```

Hierarchical Diagnostics

```
library(mlmRev)
library(HLMdiag)
data(Exam)
head(Exam)


### Sample hierarchical Model

# 1. yi = alpha_i + B*X_i + e_i
# 2. alpha_i = P * Z_i + u_i

(fm1 <- lmer(normexam ~ standLRT + (1 | school), Exam, REML = FALSE))

#This model suggests that students with higher standLRT scores at age 11
#generally scored higher on the GCSE exam at age 16. But is this model appropriate?
#To assess the appropri- ateness of model fm1 we must examine the level-1 and -2 residuals.
#Below we demonstrate using HLMresid() to calculate the LS level-1 residuals from the fitted model.

#To do this we set level = 1 and type = "LS". The standardized level-1 residuals are given by
# ehat_i  = diag(VAR(ehat_i))
#Alter- natively, we can specify standardize = "semi",
#which requests that the semi-standardized residuals (explanation below) be returned.
#For LS level-1 residuals a data frame is returned consisting of the model frame,
#LS residuals, fitted values, and, if requested, standardized residuals.
resid1_fm1 <- HLMresid(fm1, level = 1, type = "LS", standardize = TRUE)
head(resid1_fm1)

qplot(x = standLRT, y = LS.resid, data = resid1_fm1,geom = c("point", "smooth")) + ylab("LS level-1 residuals")

anova(fm1)

fm2 <- lmer(normexam ~ standLRT + I(standLRT^2) + I(standLRT^3) +
            +    (1 | school), Exam, REML = FALSE)

resid1_fm2 <- HLMresid(fm2, level = 1, type = "LS", standardize = "semi")
head(resid1_fm2)
qplot(x = I(standLRT^2), y = semi.std.resid, data = resid1_fm2) +
  geom_smooth(method = "lm") + ylab("semi-standardized residuals") +
  xlab("standLRT2")

Exam$standLRT^2
```

```
ssresid <- na.omit(resid1_fm2$semi.std.resid)

fm3 <- lmer(normexam ~ standLRT +
I(standLRT^2) + I(standLRT^3) + sex + (standLRT | school), Exam,
REML = FALSE)

#To obtain the level-2 EB residuals from model fm3, we use the following code: "EB" as it is the default setting
resid2_fm3 <- HLMresid(object = fm3, level = "school")
head(resid2_fm3)



#Marginal Analysis
#These residuals can be used for diagnostics as they would be in single-level linear models;
#however, as these residuals are the sum of the level-1 and level-2 residuals,
#any problems exhibited must be accompanied by analysis of the other types of residuals
#to pinpoint the source of the problem. One situation in which the marginal residuals are uniquely valuable
#is in assessing the marginal covariance structure, such as in repeated measures and longitudinal data,
#as the marginal residuals, ζi, and observed values, yi, have the same covariance structure.

resid3_fm3<- HLMresid(object = fm3, level = "marginal")

fm4 <- lmer(normexam ~ standLRT + I(standLRT^2) + I(standLRT^3) + sex + schgend + schavg + (standLRT | school), data =
Exam, REML = FALSE)
#Cook's distance
cooksd_fm4  <- cooks.distance(fm4, group = "school")

# MDFFITS
dffits_fm4 <- mdffits(fm4, group = "school")

#Both functions return a vector of diagnostic values and a list of the differences between the original
#and deleted fixed effects parameter vectors (beta_cdd), β□ − β□(i), as an attribute.
#To evaluate diagnostic values, we use dotplots—or a modified version of them.
#The dotplot is modified by grouping all "non-influential" units—as identified by the values of the
#diagnostic— into one group and displaying the influential groups as single cases. For the modified version
#of the dotplot, HLMdiag provides two types of modification for displaying the non-influential units:
#a dotplot or a boxplot. This type of plot allows us to see the overall distribution of the diagnostic
#while focusing on the influential points. Since this should be a commonly used plot,
#we provide the function dotplot_diag() using the plotting tools of ggplot2
```

```
dotplot_diag(x = cooksd_fm4, cutoff = "internal",name = "cooks.distance") + ylab("Cook's distance") + xlab("school")

dotplot_diag(x = dffits_fm4, cutoff = "internal", name = "mdffits", modify = "dotplot") + ylab("mdffits") + xlab("school")


#cutoff = "internal" is specified a name is re- quired, which should be one of the following:
#"cooks.distance", "mdffits", "covratio", "covtrace", "rvc", or "leverage"

#Below, we show how to access the change in the parameter vector associated with the deletion of school 25.
beta_cdd25 <- as.numeric(attr(cooksd_fm4, "beta_cdd")[[25]])
names(beta_cdd25) <- names(fixef(fm4))
beta_cdd25



#The covariance matrix of β□ gives insight into the precision of the parameter estimates. Both the covariance trace
(COVTRACE, Christensen et al. 1992) and the covariance ratio (COV- RATIO) are measures of how precision is effected by the
deletion of unit i
#Again, we make use of a general definition that allows us to examine level-specific dependencies at a later point:
covratio_fm4 <- covratio(fm4, group = "school")
covtrace_fm4 <- covtrace(fm4, group = "school")


#In the case that unit i is not influential, the covariance trace will be close to zero,
#while the covariance ratio is close to one.
covtrace_fm4
covratio_fm4


#Diagnostics for variance components
#relative variance change (RVC)  which measures the change in estimates of the lth variance component,
#θl, with and without unit i.
#RVC is close to zero when unit i does not have a large influence on the variance component.
rvc_fm4 <- rvc(fm4, group = "school")
head(rvc_fm4)

#The command rvc returns a matrix with named columns for each variance component,
#where sigma2 is the residual variance, σ2, and D** denotes the unique entries of D
#where the trailing digits denote the position in the matrix. In this example, D11 is the variance associated with
#the random intercept for schools, D22 is the variance associated with the random slope for standardized LRT score,
#and D21 is the covariance associated with the random slope and random intercept.
```

```
dotplot_diag(x = rvc_fm4[,3], cutoff = "internal", name = "rvc",
modify = "dotplot") + ylab("RVC") + xlab("school")


#Diagnostics for fitted values
#In addition to exploring how subsets of observations directly impact the model parameters,
#it is also of interest to explore whether these observations are unusual with regard to the fitted values
#and explanatory variables. This is done by exploring the leverage of subsets of interest.
#As with linear regression, leverage can be defined as the rate of change in the predicted response
#with respect to the observed response

#To reflect the plurality of statistics that can be defined as "leverage" in a hierarchical model
#leverage returns numorous quantities, the overall leverage (overall, H), the fixed effects leverage (fixef, H1),
#the random effects leverage (ranef, H2), and the unconfounded random effects leverage (ranef.uc, H*2).
#abpve .4 suggests high leverage
leverage_fm4 <- leverage(fm4, level = "school")
head(leverage_fm4)

leverage_fm4
```

Panel Regression Models

```
#pooled regression - ie no transformations
pooled <- plm(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2) + year + probation + honors,
 index = c("name", "time"), data=data)

#fixed effects y_it - ȳ_i
fixed <- plm(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2)
          + year, index = c("name", "time"), data=data, model = 'within')

#fixed effects y_it - ȳ_t  time (group) averages
between<- plm(gpa ~ summer + lag(gpa,1:3) + credits +
         + year, index = c("name", "time"), data=data, model = 'between')

#fixed effects with time and in between effects
```

```
two<- plm(gpa ~ summer + lag(gpa,1:3) + credits +
        + year, index = c("name", "time"), data=data, model = 'between', effect = ‘twoways’)

#random effects y_it − λȳ_i = β(X_it − λX̄_i) + (u_it − λū_i)
random<- plm(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2)
         + year, index = c("name", "time"), data=data, model = 'random')

summary(pooled)
summary(fixed)
summary(random)

# Breusch - Pagan Test for Random Effects
plmtest(pooled,type=c("bp"))

# Hausman Test for Random vs Fixed Effects
phtest(fixed,random)

vif(pooled)

pwtest(gpa ~ summer + lag(gpa,1:3) + credits +
       + year, index = c("name", "time"), data=data)

# Test for serial autocorrelation
pbgtest(fixed)

# Test for whether its Random Effects Causing the error or Serial Correlation
pbsytest(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2)
       + year, index = c("name", "time"), data=data, test="re")

# Test if there is serial correlation with fixed effects models
pwartest(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2)
       + year, index = c("name", "time"), data=data)

# Test if serial correlation is gone with first
pwfdtest(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2)
       + year, index = c("name", "time"), data=data)
```

```
summary(fixed)
summary(random)
summary(two)
phtest(two,pooled)
phtest(gw, gr)
plmtest(pooled, type = c("bp"))

pooltest

summary(pooltest)

#GMM for pooled specification
z1 <- pgmm(gpa ~ summer + lag(gpa,1:3) + credits + lag(credits,1:2)
        + year | lag(gpa,5:8),
        index = c("name", "time"), data = data, model = "onestep")
# GMM for best specification  - one step
z2 <- pgmm(gpa ~ summer + lag(gpa,1) + lag(credits,1:2) + probation + honors| lag(gpa,4:10),
        index = c("name", "time"), data = data, model = "onestep")

# GMM - two step
z3 <- pgmm(gpa ~ summer + lag(gpa,1) + lag(credits,1:2) + probation + honors| lag(gpa,4:10),
        index = c("name", "time"), data = data, model = "twostep")



summary(z1, robust = TRUE)
summary(z2, robust = TRUE)
summary(z3, robust= TRUE)

#Feasible GLS
var<- pggls(gpa ~ summer + lag(gpa,1:3) + credits +
         + year, index = c("name", "time"), data=data, model ='between')
```

ELO Rating System

```
library(PlayerRatings)
PR<- PlayerRatings[,-5]
PR$Player1<- as.character(PlayerRatings$Player1)
PR$Player2<- as.character(PlayerRatings$Player2)
```

```
sobj <- glicko(PR[PR$Week==1,])
for(i in 1:17) sobj <- glicko(PR[PR$Week==i,], sobj$ratings, cval = 25, gamma=3)

summary(sobj)

head(sobj$ratings,30)

sobj <- steph(PR[PR$Week==1,])
for(i in 1:17) sobj <- steph(PR[PR$Week==i,],
   sobj$ratings, cval = 20, hval=20)

head(sobj$ratings,70)
sobj$rati
write.csv(sobj$ratings, file = "player_ratings.csv")

PRs<- predict(sobj, PR)

write.csv(PRs, file = "player_rating.csv")
```

Bayesian Linear Regression + Diagnostics + Model Comparison

```
library(rstan)
library(shinystan)
library(ggplot2)
library(ggmcmc)
library(coda)
library(loo)
height = rnorm(100,5.5,1)
age<- rnorm(100,50,10)
educ<- rnorm(100,14,4)
earn = 10 + .8 * height + .7 * age + 1.1*educ + rnorm(100,0,1)

#function that resolves conflict with coda and rstan
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}

dat = data.frame(height,age, earn, educ)
#converts data frame to rstan data frame
earn_dat <- list(N = 100 ,earn = earn, age = age, height = height, educ = educ)

#model 1 code
earn_code = 'data {
  // First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
vector[N] earn; //Identify our predictor as a vector
vector[N] height;  //Identify our outcome variable as a vector
vector[N] age;
}
parameters {
vector[3] beta; //Our betas are a vector of length 2 (intercept and slope)
real<lower=0> sigma; //error parameter
}
model {
//Priors
beta[1] ~ normal( 0,1000); //intercept
beta[2] ~ normal( 0 , 1000); //slope
beta[3] ~ normal(0,1000);
sigma ~ normal( 0 , 1000); //error
```

```
earn ~ normal(beta[1] + beta[2] * height + beta[3] * age, sigma);
}
generated quantities {
  vector[N] log_lik;
  for (n in 1:N)
    log_lik[n] <- normal_log(earn[n], beta[1] + beta[2] * height[N] + beta[3] * age[N], sigma); // used for fit statistics and model comparisons
}'

# n_eff : effective sample size, a measure of autocorrelation among samples. Higher is better
# rhat: split-chain convergeance diagnostic, >1,1 suggest poor convergeance
fit1 <- stan(model_code = earn_code, data = earn_dat, warmup = 100, thin = 1, iter = 1000, chains = 4)

#extracts the parameter estimates
fit_ss<- rstan::extract(fit1, permuted = TRUE)
#resolves conflict between coda and rstan
fit<- stan2coda(fit1)
summary(fit1)
#puts the betas in a matrix
beta<- fit_ss$beta
#print the output
print(mean(beta[,1]))
print(mean(beta[,2]))
print(mean(beta[,3]))

#shiny plots for diagnostic purposes
my_fit<- launch_shinystan(fit1)

#calculates Leave-One-Out-CV and WAIC
log_lik1 <- extract_log_lik(fit1)
loo1 <- loo(log_lik1)
waic1<- waic(log_lik1)



#model 2 code
earn_code_1 = 'data {
  // First we declare all of our variables in the data block
```

```
int<lower=0> N;// Number of observations
vector[N] earn; //Identify our predictor as a vector
vector[N] height;  //Identify our outcome variable as a vector
vector[N] age;
vector[N] educ;
}
parameters {
vector[4] beta; //Our betas are a vector of length 2 (intercept and slope)
real<lower=0> sigma; //error parameter
}
model {
//Priors
beta[1] ~ normal( 0,1000); //intercept
beta[2] ~ normal( 0 , 1000); //slope
beta[3] ~ normal(0,1000);
beta[4] ~ normal(0,1000);
sigma ~ normal( 0 , 1000); //error
earn ~ normal(beta[1] + beta[2] * height + beta[3] * age + beta[4] * educ, sigma);
}
generated quantities {
vector[N] log_lik;
for (n in 1:N)
log_lik[n] <- normal_log(earn[n], beta[1] + beta[2] * height[N] + beta[3] * age[N] + beta[4] * educ[N], sigma);
}'
fit2 <- stan(model_code = earn_code_1, data = earn_dat, warmup = 100, thin = 3, iter = 1000, chains = 4)
fit_ss1<- rstan::extract(fit2, permuted = TRUE)
beta1<- fit_ss1$beta
print(mean(beta1[,1]))
print(mean(beta1[,2]))
print(mean(beta1[,3]))
print(mean(beta1[,4]))

beta<- fit_ss1$beta
log_lik2 <- extract_log_lik(fit2)
loo2 <- loo(log_lik2)
waic2 <- waic(log_lik2)
```

```
log_lik2<- extract_log_lik(fit2)
loo2<- loo(log_lik2)
waic2<- waic(log_lik2)
waic
waic2
loo1
loo2
#compares the two models, positive number suggests the first model is better, negative suggests the second is better
diff_loo<- compare(loo1,loo2)
diff_waic<- compare(waic,waic2)
diff_waic
diff_loo
```

Bayesian Model with Out of Sample Prediction

```
#the data
height = rnorm(200,5.5,1)
age<- rnorm(200,50,10)
educ<- rnorm(200,14,4)
earn = 10 + .8 * height + .7 * age + 1.1*educ + rnorm(200,0,1)
earns<- earn[1:100]
#splitting the samples
dat1 = data.frame(cbind(intercept= 1, height = height[1:100],age = age[1:100], educ = educ[1:100]))
dat2 = data.frame(cbind(intercept = 1, height = height[101:200],age = age[101:200], educ = educ[101:200]))

model<- 'data {
  int N; //the number of observations
int N2; //the size of the new_X matrix
int K; //the number of columns in the model matrix
real y[N]; //the response
matrix[N,K] X; //the model matrix
matrix[N2,K] new_X; //the matrix for the predicted values
}
parameters {
vector[K] beta; //the regression parameters
real sigma; //the standard deviation
}
```

```
transformed parameters {
vector[N] linpred;
linpred <- X*beta;
}
model {
for(i in 1:K)
beta[i] ~ cauchy(0,2.5);//prior for the slopes following Gelman 2008

y ~ normal(linpred,sigma);
}
generated quantities {
vector[N] y_pred;
y_pred <- new_X*beta; //the y values predicted by the model
}'

#function that resolves conflict with coda and rstan
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}
fit<-stan(model_code=model,data = list(N=100,N2=100,K=4, y=earns,X=dat1,new_X=dat2))
fit_ss<- rstan::extract(fit, permuted = TRUE)
y_pred<- fit_ss$y_pred
for( i in 1:100) {
w[i]<- mean(y_pred[,i])
}
plot(earn[101:200], w)
abline(a=0, b=1)
```

Bayesian AR1

```
#AR1
y  <- vector(length=500)
e  <- rnorm(500,0,1)
y[1]   <- 5
for(i in 2:length(y))
{
  y[i]  <- .4*y[i-1] + e[i]
}
plot(y, type="o", col="blue")
```

```
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}
x<- list(N=500, y = y)
#model 1 code
model = 'data {
// First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
vector[N] y; //Identify our predictor as a vector
}
parameters {
real alpha;
real beta;
real<lower=0> sigma;
}
model {
//Priors
alpha ~ normal( 0,1000); //intercept
beta ~ normal( 0 , 10); //slope
sigma ~ normal( 0 , 1000); //error
for (n in 2:N)
tail(y, N - 1) ~ normal(alpha + beta * head(y, N - 1), sigma);
}'
fit2 <- stan(model_code = model, data = x, warmup = 100, thin = 3, iter = 1000, chains = 4)
print(fit2)
fit_ss1<- rstan::extract(fit2, permuted = TRUE)
print(mean(fit_ss1$alpha))
print(mean(fit_ss1$beta))
```

Bayesian AR(K)

```
#AR(K)
y  <- vector(length=500)
e  <- rnorm(500,0,1)
y[1]   <- 5
for(i in 5:length(y))
{
```

```
  y[i]  <- .4*y[i-1] - .2*y[i-2] + .3*y[i-3] - .5*y[i-4] + e[i]
}
plot(y, type="o", col="blue")
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}
x<- list(N=500, y = y)
#model 1 code
model = 'data {
// First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
int<lower=0> K; // Number of Lag terms
vector[N] y; //Identify our predictor as a vector
}
parameters {
real alpha;
real beta[K];
real<lower=0> sigma;
}
model {
//Priors
alpha ~ normal( 0,1000); //intercept
for(i in 1:K) {
beta[i] ~ normal( 0 , 10); //slope
sigma ~ normal( 0 , 1000); //error
for (n in (K+1):N) {
real mu;
mu<- alpha;
for (k in 1:K)
mu<- mu + beta[k] * y[n-k];
y[n] ~ normal(mu, sigma);
}
}
}'
fit2 <- stan(model_code = model, data = list(x, N=500, K=4, y = y), warmup = 100, thin = 3, iter = 1000, chains = 4)
print(fit2)
fit_ss1<- rstan::extract(fit2, permuted = TRUE)
print(mean(fit_ss1$alpha))
beta<- fit_ss1$beta
```

Bayesian Moving Average(Q)

```
#MA(Q)
y  <- vector(length=500)
e  <- rnorm(500,0,1)
y[1]   <- 5
for(i in 2:length(y))
{
  y[i]  <- .4*y[i-1] + e[i]
}
plot(y, type="o", col="blue")

stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}

x<- list(N=500, y = y)

#model 1 code
model = 'data {
// First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
int<lower=0> Q; // Number of noise lag terms
vector[N] y; //Identify our predictor as a vector
}
parameters {
 real mu;              // mean
real<lower=0> sigma;  // error scale
vector[Q] theta;      // error coeff, lag -t
}
transformed parameters {
      vector[N] epsilon;    // error term at time t
for (t in 1:N) {
epsilon[t] <- y[t] - mu;
for (q in 1:min(t - 1, Q))
epsilon[t] <- epsilon[t] - theta[q] * epsilon[t - q];
}
}
model {
```

```
vector[N] eta;
mu ~ cauchy(0, 2.5);
theta ~ cauchy(0, 2.5);
sigma ~ cauchy(0, 2.5);
for (t in 1:N) {
eta[t] <- mu;
for (q in 1:min(t - 1, Q))
eta[t] <- eta[t] + theta[q] * epsilon[t - q];
}
y ~ normal(eta, sigma);
}'

fit2 <- stan(model_code = model, data = list(x, N=500, Q=4, y = y), warmup = 100, thin = 3, iter = 1000, chains = 4)

print(fit2)

fit_ss1<- rstan::extract(fit2, permuted = TRUE)
theta<- fit_ss1$theta

print(mean(fit_ss1$alpha))
beta<- fit_ss1$beta
```

Bayesian #ARMA(1,1)

```
y  <- vector(length=500)
e  <- rnorm(500,0,1)
y[1]   <- 5
for(i in 2:length(y))
{
  y[i]  <- .4*y[i-1] + e[i]
}
plot(y, type="o", col="blue")

stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}

model<- 'data {
  int<lower=1> T;       // number of observations
```

```
  real y[T];           // observed outputs
}
parameters {
  real mu;             // mean term
  real phi;            // autoregression coeff
  real theta;          // moving avg coeff
  real<lower=0> sigma;  // noise scale
}
model {
  vector[T] nu;        // prediction for time t
  vector[T] err;       // error for time t
  nu[1] <- mu + phi * mu;   // assume err[0] == 0
  err[1] <- y[1] - nu[1];
  for (t in 2:T) {
    nu[t] <- mu + phi * y[t-1] + theta * err[t-1];
    err[t] <- y[t] - nu[t];
  }

  // priors
  mu ~ normal(0,10);
  phi ~ normal(0,2);
  theta ~ normal(0,2);
  sigma ~ cauchy(0,5);

  // likelihood
  y ~ normal(err,sigma);
}'

fit <- stan(model_code = model, data=list(T=500, y=y), iter=200, chains=4);

fit_ss1<- rstan::extract(fit, permuted = TRUE)

phi <- fit_ss1$phi
theta<- fit_ss1$theta
```

Bayesian ARMA(P,Q)

```
#ARMA(P,Q)
```

```
y  <- vector(length=500)
x<- vector
e  <- rnorm(500,0,1)
y[1]   <- 0
for(i in 2:length(y))
{
  y[i]  <- .4*y[i-1] + e[i] + .2*e[i-1]
}

x<- list(N=100,y=y)
plot(y, type="o", col="blue")
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}

#model 1 code
model = 'data {
// First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
int<lower=0> K; // Number of AR Terms
int<lower=0> Q; // Number of MA Terms
vector[N] y; //Identify our predictor as a vector
}
parameters {
real mu;
real beta[K];
real<lower=0> sigma;
real theta[Q]; //error coeff, lag -t
}
transformed parameters {
      vector[N] epsilon;    // error term at time t
for (t in 1:N) {
for(n in (K+1):N)
for(k in 1:K)
epsilon[t] <- beta[k] * y[n-k] - mu;
for (q in 1:min(t - 1, Q))
epsilon[t] <- epsilon[t] - theta[q] * epsilon[t - q];
}
}
```

```
model {
//Priors
vector[N] eta;
mu ~ normal(0,100);
for(i in 1:K) {
beta[i] ~ normal( 0 , 10); //AR
for(j in 1:Q) {
theta[j] ~ normal(0,10); //MA
sigma ~ normal( 0 , 1000); //error
for (t in 1:N) {
eta[t]<- mu;
for(q in 1:min(t-1,Q))
eta[t]<- eta[t] + theta[q] * epsilon[t-q];
}
}
}
y ~ normal(eta, sigma);
}'



print(fit2)
fit_ss1<- rstan::extract(fit2, permuted = TRUE)
print(mean(fit_ss1$alpha))
print(mean(fit_ss1$theta))
```

Bayesian ARMAX

```
y  <- vector(length=100)
e  <- rnorm(100,0,1)
y[1]   <- 5
x<- vector(length=100)
x<- rnorm(100,0,1)
for(i in 1:length(x))
for(i in 2:length(y))
{
  y[i]  <- .4*y[i-1] + .4*x[i-1] + e[i]
}
plot(y, type="o", col="blue")
```

```
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}
z<- list(N=100, y = y,x = x)
#model 1 code
model = 'data {
// First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
int<lower=0> K; // Number of Lag terms
int<lower=0> P; // Number of Lag terms
vector[N] y; //Identify our predictor as a vector
vector[N] x; //Dependent variable
}
parameters {
real alpha;
real phi[K];
real beta[P];
real<lower=0> sigma;
}
model {
//Priors
real mu;
alpha ~ normal( 0,1000); //intercept
for(i in 1:K) {
for(j in 1:P) {
phi[i] ~ normal( 0 , 10); //slope
beta[j] ~ normal(0, 1000); //
sigma ~ normal( 0 , 1000); //error
for (n in (K+1):N) {
for (p in (P+1):N)
mu<- alpha;
for (k in 1:K)
for (p in 1:P)
mu<- mu + phi[k] * y[n-k] + beta[p] * x[n-p];
y[n] ~ normal(mu, sigma);
}
}
}
}'
```

```
fit2 <- stan(model_code = model, data = list(x, N=100, K=1, P = 1, y = y, x = x), warmup = 200, thin = 1, iter = 2000, chains = 6)
print(fit2)
```

Bayesian Dynamic Panel Regression

```
x  <- rnorm(200,0,1)
y<- vector(length=200)
e  <- rnorm(200)
z1<- rep(c(1,0),each=100)
z2<- rep(c(0,1), each=100)
z<- rep(c('a','b'), each = 100)

y[1]   <- 0
for(i in 2:length(y))
{
  y[i]  <- .4*y[i-1] + .5*x[i] -.2*x[i-1]  + 5*z1[i] - 5*z2[i] + e[i]
}

plot.ts(y)

subject_index<- as.integer(as.factor(z))
dat<- data.frame(cbind(y,x,subject_index))
stan2coda <- function(fit) {mcmc.list(lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,])))}

#model 1 code
model = 'data {
// First we declare all of our variables in the data block
int<lower=0> N;// Number of observations
int<lower=0> K; // Number of Lag terms
int<lower=0> P; // Number of Lag terms
vector[N] y; //Identify our predictor as a vector
vector[N] x; //Dependent variable
int subject_index[N];
int N_subject_index;
}
parameters {
real alpha;
real phi[K];
real beta[P];
real<lower=0> sigma;
}
```

```
model {
//Priors
real mu;
alpha ~ normal( 0,1000); //intercept
for(i in 1:K) {
for(j in 1:P) {
phi[i] ~ normal( 0 , 10); //slope
beta[j] ~ normal(0, 1000); //
sigma ~ normal( 0 , 1000); //error
for (n in (K+1):N_subject_index) {
for (p in (P+1):N_subject_index)
mu<- alpha;
for (k in 1:K)
for (p in 1:P)
mu <- mu + phi[k] * y[n-k] + beta[p] * x[n-p];
y[n] ~ normal(mu, sigma);
}
}
}
}'

fit2 <- stan(model_code = model, data = list(x, N=200, K=1, P = 1, y = y, x = x,
subject_index = subject_index, N_subject_index = 2),
warmup = 10, thin = 1, iter = 100, chains = 1)
```

Bayesian Stochastic Volatility Model

Let y = $(y_1, y_2, ..., y_n)^T$ be a vector of returns with mean zero. The intrinsic feature of the SV model is that each observation y_t is assumed to have its "own" contemporaneous variance e^{h_t}, thus relaxing the usual assumption of homoscedasticity. In order to make the estimation of such a model feasible, this variance is not allowed to vary unrestrictedly with time. Rather, its logarithm is assumed to follow an autoregressive process of order one. Note that this feature is fundamentally different to GARCH-type models where the time-varying volatility is assumed to follow a deterministic instead of a stochastic evolution. The SV model can thus be conveniently expressed in hierarchical form. In its centered parameterization, it is given through

1. $y_t|h_t \rightarrow N(0, e^{h_t})$
2. $h_t|h_{t-1} \rightarrow N(\mu + \phi(h_{t-1} - \mu), \sigma_\eta^2)$
3. $h_0|\mu, \phi, \sigma_\eta \rightarrow N(\mu, \sigma_\eta^2/(1 - \phi^2))$

where N denotes denotes the normal distribution with mean μ and variance σ_η^2. We refer to $\theta = (\mu, \phi, \sigma_\eta^2)$ as the vector of parameters: the level of log-variance μ, the persistence of log-variance φ, and the volatility of log-variance σ_η. The process h = $(h_0, h_1, ..., h_n)$ appearing in Equation 2 and Equation 3 is unobserved and usually interpreted as the latent time-varying volatility process (more precisely, the log-variance process). Note that the initial state h0 appearing in Equation 3 is distributed according to the stationary distribution of the autoregressive process of order one.

```
phi <- 0.95;
sigma <- 0.25;
beta <- 0.6;
mu <- 2 * log(beta);
T <- 500;
h <- rep(NA,T);
h[1] <- rnorm(1, mu, sigma / sqrt(1 - phi * phi));
for (t in 2:T)
  h[t] <- rnorm(1, mu + phi * (h[t-1] - mu), sigma);
y <- rep(NA,T);
for (t in 1:T)
  y[t] <- rnorm(1, 0, exp(h[t] / 2));

plot.ts(y)
```

```
stoch<- 'data {
  int<lower=0> T;   // # time points (equally spaced)
    vector[T] y;      // mean corrected return at time t
  }
parameters {
  real mu;                     // mean log volatility
  real<lower=-1,upper=1> phi;  // persistence of volatility
  real<lower=0> sigma;         // white noise shock scale
  vector[T] h_std;             // std log volatility time t
  }
transformed parameters {
  vector[T] h;                 // log volatility at time t
  h <- h_std * sigma;
  h[1] <- h[1] / sqrt(1 - phi * phi);
  h <- h + mu;
  for (t in 2:T)
    h[t] <- h[t] + phi * (h[t-1] - mu);
    }
model {
  sigma ~ cauchy(0,5);
  mu ~ cauchy(0,10);
  h_std ~ normal(0,1);
  y ~ normal(0, exp(h / 2));
  }'
fit <- stan(model_code = stoch, data=list(T=T,y=y), warmup = 100, thin = 2, iter=10000, chains=4, init=0)
elapsed_time <- proc.time() - start_time

fit1<- stan2coda(fit)

fit_ss<- rstan::extract(fit1, permuted = TRUE)

mu<- fit_ss$mu
phi<- fit_ss$phi
print(mean)
print(mean(phi))
```

Hierarchical Bayesian Model

```
#Installing GLMER2Stan
#options(repos=c(getOption('repos'),
#glmer2stan='http://xcelab.net/R'))
#install.packages('glmer2stan',type='source')
```

LMER to RSTAN

```
data(sleepstudy)
 # The average reaction time per day for subjects in a sleep deprivation study.
 # On day 0 the subjects had their normal amount of sleep
 # Starting that night, they were restricted to 3 hours of sleep per night
 # The observation represents the average reaction time on a series of tests
 # given each day to each subject
 m1_lm<- lm(Reaction ~ Days, data = sleepstudy)
 confint(m1_lm)
summary(m1_lm)
 ggplot(sleepstudy, aes(x=Days, y = Reaction)) +
  geom_point() +
  guides(color=F) +
  geom_smooth(method=lm, se = F)
 # with nesting
ggplot(sleepstudy, aes(x=Days, y = Reaction, color=Subject, group = Subject)) +
  geom_point() +
  guides(color=F) +
  geom_smooth(method=lm, se = F)
 # Hierarchical Linear Model, Random slopes and intercepts sets
m1_lme4 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy, REML = FALSE)
 summary(m1_lme4)
confint(m1_lme4)
 AIC(m1_lm,m1_lme4)

# Important annoying fact #2 : STAN doesn't deal with non-numeric variables
 # - factors must be converted to contrast code( glmer2stan does that for you)
 # - grouping variables, however, must be manually converted to integers
 # convert factor of subject ids to sequential integers
 sleepstudy$subject_index<- as.integer(as.factor(sleepstudy$Subject))
 # basic MCMC parameter, should probably be a bit higher but we don't have all day
```

```
 nwarm = 100 # burn in,
niter = 500 # number of steps per chain, more is better(but takes longer)
chains = 4 # number of chains, usually at least 2
 # This Block has to be run to put priors in
m1_g2s_NoSamples <- lmer2stan(Reaction ~ Days + (Days | subject_index), data = sleepstudy,
                        calcWAIC = T,
                        warmup = nwarm,
                        iter = niter,
                        chains = chains,
                        sample=F # when this is NULL the model is built and run like normal#
)

m1_g2s <- lmer2stan(Reaction ~ Days + (Days | subject_index), data = sleepstudy,
                        calcWAIC = T,
                        warmup = nwarm,
                        iter = niter,
                        chains = chains,
                        mymodel=NULL # when this is NULL the model is built and run like normal#
                        )
print(m1_g2s) # standard stan output
stanmer(m1_g2s) # cleaned up stan output
plot(m1_g2s) # looks like shit
traceplot(m1_g2s)

# myglmer2stan
# - fixes bug in formula code which crashes with interactions specified by ':' rather than '*'
# - allows user to use homebrewed model as a function argument

# the problem with glmer2stan is that you can extract the model code no problem, but its not straightforward
# at all to re run the model with edits

# First we'll re run with no edits to the model
 cat(m1_g2s_NoSamples$model) # print the model
sink('example1.txt') # open connection
cat(m1_g2s_NoSamples$model) # any output between these two functions gets written to that file
 sink() # close connection
# This is the end of the code block that enables us to put in priors
```

```
my_g2s_priors = 'data{
  int N;
  real Reaction[N];  # Dependent Variable
  real Days[N]; # Independent Variable
  int subject_index[N];  # Random effects Variable
  int N_subject_index;
}
transformed data{
  vector[2] zeros_subject_index;
  for ( i in 1:2 ) zeros_subject_index[i] <- 0;
}
parameters{
  real Intercept;
  real beta_Days;
  real<lower=0> sigma;
  vector[2] vary_subject_index[N_subject_index];
  cov_matrix[2] Sigma_subject_index;
}
model{
  real vary[N];
  real glm[N];
  // Priors
  Intercept ~ normal( 0 , 100 ); // Hey! Look! An Informed Prior!
  beta_Days ~ normal( 10 , 2 );
  sigma ~ uniform( 0 , 100 );
  // Varying effects
  for ( j in 1:N_subject_index ) vary_subject_index[j] ~ multi_normal( zeros_subject_index , Sigma_subject_index );
  // Fixed effects
  for ( i in 1:N ) {
            vary[i] <- vary_subject_index[subject_index[i],1]
            + vary_subject_index[subject_index[i],2] * Days[i];
            glm[i] <- vary[i] + Intercept
            + beta_Days * Days[i];
  }
  Reaction ~ normal( glm , sigma );
}
 generated quantities{
```

```
  real dev;
  real vary[N];
  real glm[N];
  dev <- 0;
  for ( i in 1:N ) {
            vary[i] <- vary_subject_index[subject_index[i],1]
            + vary_subject_index[subject_index[i],2] * Days[i];
            glm[i] <- vary[i] + Intercept
            + beta_Days * Days[i];
            dev <- dev + (-2) * normal_log( Reaction[i] , glm[i] , sigma );
  }
}'
m1_g2s_final <- lmer2stan(Reaction ~ Days + (Days | subject_index), data = sleepstudy,
                          calcWAIC = T,
                          warmup = nwarm,
                          iter = niter,
                          chains = chains,
                          mymodel= my_g2s_priors
)


data{
    int N;
    real Reaction[N];
    real Days[N];
    int subject_index[N];
    int N_subject_index;
}

parameters{
    real Intercept;
    real beta_Days;
    real<lower=0> sigma;
    real vary_subject_index[N_subject_index];
    real<lower=0> sigma_subject_index;
}
```

```
model{
    real vary[N];
    real glm[N];
    // Priors
    Intercept ~ normal( 0 , 100 );
    beta_Days ~ normal( 0 , 100 );
    sigma_subject_index ~ uniform( 0 , 100 );
    sigma ~ uniform( 0 , 100 );
    // Varying effects
    for ( j in 1:N_subject_index ) vary_subject_index[j] ~ normal( 0 , sigma_subject_index );
    // Fixed effects
    for ( i in 1:N ) {
        vary[i] <- vary_subject_index[subject_index[i]];
        glm[i] <- vary[i] + Intercept
                + beta_Days * Days[i];
    }
    Reaction ~ normal( glm , sigma );
}

generated quantities{
    real dev;
    real vary[N];
    real glm[N];
    dev <- 0;
    for ( i in 1:N ) {
        vary[i] <- vary_subject_index[subject_index[i]];
        glm[i] <- vary[i] + Intercept
                + beta_Days * Days[i];
        dev <- dev + (-2) * normal_log( Reaction[i] , glm[i] , sigma );
    }
}
```

RStan Code Varying Intercept Model

Varying intercept model

This model allows intercepts to vary across county, according to a random effect.

$$y_i = \alpha_{j[i]} + \beta x_i + \varepsilon_i$$

where

$$\varepsilon_i = N(0, \sigma_y^2)$$

and the intercept random effect:

$$\alpha_{j[i]} \sim N(\mu_\alpha, \sigma_\alpha^2)$$

```
varying_intercept = """
data {
  int<lower=0> J;
  int<lower=0> N;
  int<lower=1,upper=J> county[N];
  vector[N] x;
  vector[N] y;
}
parameters {
  vector[J] a;
  real b;
  real mu_a;
  real<lower=0,upper=100> sigma_a;
  real<lower=0,upper=100> sigma_y;
}
transformed parameters {

  vector[N] y_hat;

  for (i in 1:N)
    y_hat[i] <- a[county[i]] + x[i] * b;
}
model {
  sigma_a ~ uniform(0, 100);
  a ~ normal (mu_a, sigma_a);
```

```
  b ~ normal (0, 1);

  sigma_y ~ uniform(0, 100);
  y ~ normal(y_hat, sigma_y);
}
"""
```

```
Varying_intercept_data = (N= len(log_radon), J = len(n_county), county = county, x = floor_measure, y =
log_radon)


Varying_intercept_fit = rstan(model_code = varying_intercept, data= varying_intercept_data, iter=1000, chains=2)
```

RStan Code Varying Slope Model

Varying slope model

This model allows slope to vary across county, according to a random effect.

$$y_i = \alpha + \beta_{j[i]} x_i + \varepsilon_i$$

where

$$\varepsilon_i = N(0, \sigma_y^2)$$

and the slope random effect:

$$\beta_{j[i]} \sim N(\mu_\beta, \sigma_\beta^2)$$

```
varying_slope = """
data {
  int<lower=0> J;
  int<lower=0> N;
  int<lower=1,upper=J> county[N];
  vector[N] x;
  vector[N] y;
}
parameters {
  real a;
  vector[J] b;
  real mu_b;
  real<lower=0,upper=100> sigma_b;
  real<lower=0,upper=100> sigma_y;
}
transformed parameters {

  vector[N] y_hat;

  for (i in 1:N)
    y_hat[i] <- a + x[i] * b[county[i]];
}
model {
  sigma_b ~ uniform(0, 100);
  b ~ normal (mu_b, sigma_b);

  a ~ normal (0, 1);
```

```
  sigma_y ~ uniform(0, 100);
  y ~ normal(y_hat, sigma_y);
}
"""
```

```
Varying__slope_data = (N= len(log_radon), J = len(n_county), county = county, x = floor_measure, y = log_radon)
```

```
Varying_slope_fit = rstan(model_code = varying_slope, data= varying_slope_data, iter=1000, chains=2)
```

Rstan Varying intercept and slope

Varying intercept and slope model

The most general model allows both the intercept and slope to vary by county:

$$y_i = \alpha_{j[i]} + \beta_{j[i]} x_i + \varepsilon_i$$

where

$$\varepsilon_i = N(0, \sigma_y^2)$$

and the slope random effect:

$$\beta_{j[i]} \sim N(\mu_\beta, \sigma_\beta^2)$$

and the intercept random effect:

$$\alpha_{j[i]} \sim N(\mu_\alpha, \sigma_\alpha^2)$$

```
varying_intercept_slope = """
data {
  int<lower=0> N;
  int<lower=0> J;
  vector[N] y;
  vector[N] x;
  int county[N];
}
parameters {
  real<lower=0> sigma;
  real<lower=0> sigma_a;
  real<lower=0> sigma_b;
  vector[J] a;
  vector[J] b;
  real mu_a;
  real mu_b;
}

model {
  mu_a ~ normal(0, 100);
  mu_b ~ normal(0, 100);
```

```
 a ~ normal(mu_a, sigma_a);
 b ~ normal(mu_b, sigma_b);
 y ~ normal(a[county] + b[county]*x, sigma);
}
```

Varying__intercept_slope_data = (N= len(log_radon), J = len(n_county), county = county, x = floor_measure, y = log_radon)

Varying_intercept_slope_fit = rstan(model_code = varying_intercept_slope, data= varying_intercept_slope_data, iter=1000, chains=2)

RStan Group level predictors

Adding group-level predictors

A primary strength of multilevel models is the ability to handle predictors on multiple levels simultaneously. If we consider the varying-intercepts model above:

$$y_i = \alpha_{j[i]} + \beta x_i + \varepsilon_i$$

we may, instead of a simple random effect to describe variation in the expected radon value, specify another regression model with a county-level covariate. Here, we use the county uranium reading ujuj, which is thought to be related to radon levels:

$$\alpha_j = \gamma_0 + \gamma_1 \mu_j + \zeta_j$$

$$\zeta_j \sim N(0, \sigma_\alpha^2)$$

Thus, we are now incorporating a house-level predictor (floor or basement) as well as a county-level predictor (uranium).

Note that the model has both indicator variables for each county, plus a county-level covariate. In classical regression, this would result in collinearity. In a multilevel model, the partial pooling of the intercepts towards the expected value of the group-level linear model avoids this.

Group-level predictors also serve to reduce group-level variation σ_α. An important implication of this is that the group-level estimate induces stronger pooling.

```
hierarchical_intercept = """
data {
 int<lower=0> J;
 int<lower=0> N;
 int<lower=1,upper=J> county[N];
 vector[N] u;
 vector[N] x;
 vector[N] y;
}
parameters {
 vector[J] a;
 vector[2] b;
 real mu_a;
 real<lower=0,upper=100> sigma_a;
 real<lower=0,upper=100> sigma_y;
}
```

```
transformed parameters {
  vector[N] y_hat;
  vector[N] m;

  for (i in 1:N) {
    m[i] <- a[county[i]] + u[i] * b[1];
    y_hat[i] <- m[i] + x[i] * b[2];
  }
}
model {
  mu_a ~ normal(0, 1);
  a ~ normal(mu_a, sigma_a);
  b ~ normal(0, 1);
  y ~ normal(y_hat, sigma_y);
}
"""
hierarchical_intercept_data = (N= len(log_radon), J = len(n_county), county = county, x = floor_measure, y =
log_radon, u = u)

hierarchical_intercept_fit = rstan(model_code = varying_intercept_slope, data= varying_intercept_slope_data,
iter=1000, chains=2)
```

RStan Group level predictors - correlation among levels

Correlations among levels

In some instances, having predictors at multiple levels can reveal correlation between individual-level variables and group residuals. We can account for this by including the average of the individual predictors as a covariate in the model for the group intercept.

A primary strength of multilevel models is the ability to handle predictors on multiple levels simultaneously. If we consider the varying-intercepts model above:

$$y_i = \alpha_{j[i]} + \beta x_i + \varepsilon_i$$

we may, instead of a simple random effect to describe variation in the expected radon value, specify another regression model with a county-level covariate. Here, we use the county uranium reading ujuj, which is thought to be related to radon levels:

$$\alpha_j = \gamma_0 + \gamma_1 \mu_j + \gamma_2 \bar{x} + \zeta_j$$

$$\zeta_j \sim N(0, \sigma_\alpha^2)$$

```
contextual_effect = """
data {
  int<lower=0> J;
  int<lower=0> N;
  int<lower=1,upper=J> county[N];
  vector[N] u;
  vector[N] x;
  vector[N] x_mean;
  vector[N] y;
}
parameters {
  vector[J] a;
  vector[3] b;
  real mu_a;
  real<lower=0,upper=100> sigma_a;
  real<lower=0,upper=100> sigma_y;
}
transformed parameters {
  vector[N] y_hat;

  for (i in 1:N)
    y_hat[i] <- a[county[i]] + u[i]*b[1] + x[i]*b[2] + x_mean[i]*b[3];
}
```

```
model {
  mu_a ~ normal(0, 1);
  a ~ normal(mu_a, sigma_a);
  b ~ normal(0, 1);
  y ~ normal(y_hat, sigma_y);
}
"""
```

```
 contextual_effect_data = (N= len(log_radon), J = len(n_county), county = county, x = floor_measure, y =
 log_radon, u = u, x_mean = mean(x))
contextual_effect_fit = rstan(model_code = contextual_effect, data= contextual_effect_data, iter=1000, chains=2)
```

Rstan three level intercept model

```
#Linear Mixed Effects Model (intercept only)

library(magrittr)
library(rstan)
library(lme4)
library(dplyr)

classroom <- read.csv("http://www-personal.umich.edu/~bwest/classroom.csv")

## Create a vector of school IDs where j-th element gives school ID for class ID j
schoolLookupVec <- unique(classroom[c("classid","schoolid")])[,"schoolid"]



Ni  = length(unique(classroom$childid))
Nj  = length(unique(classroom$classid))
Nk  = length(unique(classroom$schoolid))
## Combine as a stan dataset
dat <- with(classroom,
            list(Ni          = length(unique(childid)),
                 Nj          = length(unique(classid)),
                 Nk          = length(unique(schoolid)),
                 classid     = classid,
                 schoolid    = schoolid,
                 schoolLookup = schoolLookupVec,
                 mathgain    = mathgain))

stan_code<- 'data {
 // Define variables in data
    // Number of level-1 observations (an integer)
    int<lower=0> Ni;
    // Number of level-2 clusters
    int<lower=0> Nj;
    // Number of level-3 clusters
    int<lower=0> Nk;

    // Cluster IDs
```

```
  int<lower=1> classid[Ni];
  int<lower=1> schoolid[Ni];

  // Level 3 look up vector for level 2
  int<lower=1> schoolLookup[Nj];

  // Continuous outcome
  real mathgain[Ni];

  // Continuous predictor
  // real X_1ijk[Ni];
}

parameters {
  // Define parameters to estimate
  // Population intercept (a real number)
  real beta_0;
  // Population slope
  // real beta_1;

  // Level-1 errors
  real<lower=0> sigma_e0;

  // Level-2 random effect
  real u_0jk[Nj];
  real<lower=0> sigma_u0jk;

  // Level-3 random effect
  real u_0k[Nk];
  real<lower=0> sigma_u0k;
}

transformed parameters  {
  // Varying intercepts
  real beta_0jk[Nj];
  real beta_0k[Nk];
```

```
    // Individual mean
    real mu[Ni];

    // Varying intercepts definition
    // Level-3 (10 level-3 random intercepts)
    for (k in 1:Nk) {
      beta_0k[k] <- beta_0 + u_0k[k];
    }
    // Level-2 (100 level-2 random intercepts)
    for (j in 1:Nj) {
      beta_0jk[j] <- beta_0k[schoolLookup[j]] + u_0jk[j];
    }
    // Individual mean
    for (i in 1:Ni) {
      mu[i] <- beta_0jk[classid[i]];
    }
  }

  model {
    // Prior part of Bayesian inference
    // Flat prior for mu (no need to specify if non-informative)

    // Random effects distribution
    u_0k  ~ normal(0, sigma_u0k);
    u_0jk ~ normal(0, sigma_u0jk);

    // Likelihood part of Bayesian inference
    // Outcome model N(mu, sigma^2) (use SD rather than Var)
    for (i in 1:Ni) {
      mathgain[i] ~ normal(mu[i], sigma_e0);
    }
  }'

resStan <- stan(model_code = stan_code, data = dat,
                chains = 4, iter = 10000, warmup = 1000, thin = 10)
traceplot(resStan, pars = c("beta_0","sigma_e0","sigma_u0jk","sigma_u0k"), inc_warmup = FALSE)
print(resStan, pars = c("beta_0","sigma_e0","sigma_u0jk","sigma_u0k"))
```

Rstan Linear Mixed effects Model with Random Intercept & fixed covariates

```
# Linear Mixed Effects Model (random intercepts with fixed effect covariates) ----------------------------

library(magrittr)
library(rstan)
library(lme4)
library(dplyr)

classroom <- read.csv("http://www-personal.umich.edu/~bwest/classroom.csv")

## Create a vector of school IDs where j-th element gives school ID for class ID j
schoolLookupVec <- unique(classroom[c("classid","schoolid")])[,"schoolid"]



Ni  = length(unique(classroom$childid))
Nj  = length(unique(classroom$classid))
Nk  = length(unique(classroom$schoolid))

## Design matrix for model 4.4
desMat <- model.matrix(object = ~ 1 + mathkind + sex + minority + ses + housepov, data = classroom)

## Combine as a stan dataset
dat2 <- with(classroom,
           list(Ni          = length(unique(childid)),
                Nj          = length(unique(classid)),
                Nk          = length(unique(schoolid)),
                p           = ncol(desMat),
                desMat      = desMat,
                classid     = classid,
                schoolid    = schoolid,
                schoolLookup = schoolLookupVec,
                mathgain    = mathgain))
#Linear Mixed Effects Model (random intercepts with fixed effect covariates)

'data {
    // Define variables in data
    // Number of level-1 observations (an integer)
```

```
  int<lower=0> Ni;
  // Number of level-2 clusters
  int<lower=0> Nj;
  // Number of level-3 clusters
  int<lower=0> Nk;
  // Number of fixed effect parameters
  int<lower=0> p;

  // Design matrix
  real desMat[Ni,p];

  // Cluster IDs
  int<lower=1> classid[Ni];
  int<lower=1> schoolid[Ni];

  // Level 3 look up vector for level 2
  int<lower=1> schoolLookup[Nj];

  // Continuous outcome
  real mathgain[Ni];

  // Continuous predictor
  // real X_1ijk[Ni];
}

parameters {
  // Define parameters to estimate
  // Fixed effects
  real beta[p];

  // Level-1 errors
  real<lower=0> sigma_e0;

  // Level-2 random effect
  real u_0jk[Nj];
  real<lower=0> sigma_u0jk;
```

```
  // Level-3 random effect
  real u_0k[Nk];
  real<lower=0> sigma_u0k;
}

transformed parameters  {
  // Varying intercepts
  real beta_0jk[Nj];
  real beta_0k[Nk];

  // Individual mean
  real mu[Ni];

  // Varying intercepts definition
  // Level-3 (10 level-3 random intercepts)
  for (k in 1:Nk) {
    beta_0k[k] <- beta[1] + u_0k[k];
  }
  // Level-2 (100 level-2 random intercepts)
  for (j in 1:Nj) {
    beta_0jk[j] <- beta_0k[schoolLookup[j]] + u_0jk[j];
  }
  // Individual mean
  for (i in 1:Ni) {
    mu[i] <- beta_0jk[classid[i]] +
      desMat[i,2]*beta[2] + desMat[i,3]*beta[3] + desMat[i,4]*beta[4] +
      desMat[i,5]*beta[5] + desMat[i,6]*beta[6];
  }
}

model {
  // Prior part of Bayesian inference
  // Flat prior for mu (no need to specify if non-informative)

  // Random effects distribution
  u_0k  ~ normal(0, sigma_u0k);
  u_0jk ~ normal(0, sigma_u0jk);
```

```
    // Likelihood part of Bayesian inference
    // Outcome model N(mu, sigma^2) (use SD rather than Var)
    for (i in 1:Ni) {
      mathgain[i] ~ normal(mu[i], sigma_e0);
    }
  }'

## Run Stan
resStan <- stan(model_code = stan_code, data = dat2,
                chains = 4, iter = 10000, warmup = 1000, thin = 10)

traceplot(resStan, pars = c("beta","sigma_e0","sigma_u0jk","sigma_u0k"), inc_warmup = FALSE)

print(resStan, pars = c("beta","sigma_e0","sigma_u0jk","sigma_u0k"))
```

HLM_Parrallel

```
library(rstan)
library(devtools)
install_github('nathanvan/rstanmulticore')
library(rstanmulticore)
library(loo)
library(shinystan)
P<- Purdue2
P$Majors<- as.factor(P$Majors)
P$Class<- as.factor(P$Class2)
P$Time<- P$T
P$Majors_index<- as.integer(as.factor(P$Majors))
P$Class2_index <- as.integer(as.factor(P$Class2))
ClassLookupVec <- (unique(P[c("Majors_index","Class2_index")])[,"Class2_index"])
dat <- with(P,
        list(Ni          = length(unique(R)),
             Nk          = length(unique(Majors_index)),
             Nj          = length(unique(Class2_index)),
             Time        = Time,
             Lag_GPA     = L1_Term_GPA,
             Repeated    = Repeated,
             Summer      = Summer,
             Class_Taken  = Class_Taken,
             Class2_index = Class2_index,
             T_Units_Attempted = T_Units_Attempted,
             Unit_Difference =  Unit_Difference,
             EM           = EM,
             MS          =  MS,
             Major     = Majors_index,
             Major_Lookup = unique(ClassLookupVec),
             GPA     = GPA))
#Linear Mixed Effects Model (random intercepts with fixed effect covariates)
stan_code<- 'data {
// Define variables in data
// Number of level-1 observations (an integer)
int<lower=0> Ni;
// Number of level-2 clusters
```

```
int<lower=0> Nj;
// Number of level-3 clusters
int<lower=0> Nk;

// Cluster ID
int<lower=1> Major[Ni];
int<lower=1> Class2_index[Ni];

// Level 3 look up vector for level 2
int<lower=1> Major_Lookup[Nj];

// Continuous outcome
real GPA[Ni];

// Continuous predictor
real Repeated[Ni];
real Summer[Ni];
real Class_Taken[Ni];
real T_Units_Attempted[Ni];
real Unit_Difference[Ni];
real Time[Ni];
real Lag_GPA[Ni];
real MS[Ni];
real EM[Ni];
}

parameters {
// Define parameters to estimate
// Fixed effects
real beta_0;
real beta_1;
real beta_2;
real beta_3;
real beta_4;
real beta_5;
real beta_6;
real beta_7;
```

```
real beta_8;
real beta_9;

// Level-1 errors
real<lower=0> sigma_e0;
// Level-2 random effect
real u_0jk[Nj];
real<lower=0> sigma_u0jk;
// Level-3 random effect
real u_0k[Nk];
real<lower=0> sigma_u0k;
}

transformed parameters  {
// Varying intercepts
real beta_0jk[Nj];
real beta_0k[Nk];

// Individual mean
real mu[Ni];

// Varying intercepts definition
// Level-3 (10 level-3 random intercepts)
for (k in 1:Nk) {
beta_0k[k] <- beta_0 + u_0k[k];
}
// Level-2 (100 level-2 random intercepts)
for (j in 1:Nj) {
beta_0jk[j] <- beta_0k[Major_Lookup[j]] + u_0jk[j];
}
// Individual mean
for (i in 1:Ni) {
mu[i] <- beta_0jk[Class2_index[i]] +
beta_1 * Lag_GPA[i] + beta_2 * Time[i] +
beta_3 * Summer[i] + beta_4 * Class_Taken[i] +
beta_5 * T_Units_Attempted[i] + beta_6 * Unit_Difference[i] +
beta_7 * Repeated[i] + beta_8 * EM[i] + beta_9 * MS[i];
```

```
}
}
model {
// Prior part of Bayesian inference
// Flat prior for mu (no need to specify if non-informative)
beta_0 ~ normal(0, 10);
beta_1 ~ normal(0, 10);
beta_2 ~ normal(0, 10);
beta_3 ~ normal(0, 10);
beta_4 ~ normal(0, 10);
beta_5 ~ normal(0, 10);
beta_6 ~ normal(0, 10);
beta_7 ~ normal(0, 10);
beta_8 ~ normal(0, 10);
beta_9 ~ normal(0, 10);

// Random effects distribution
u_0k  ~ normal(0, sigma_u0k);
u_0jk ~ normal(0, sigma_u0jk);
// Likelihood part of Bayesian inference
// Outcome model N(mu, sigma^2) (use SD rather than Var)
for (i in 1:Ni) {
GPA[i] ~ normal(mu[i], sigma_e0);
}
}
generated quantities {
vector[Ni] log_lik;
vector[Ni] y_pred;
for (i in 1:Ni){
log_lik[i] <- normal_lpdf(GPA[i] | mu[i], sigma_e0); // used for fit statistics and model comparisons
y_pred[i]<- mu[i];
}}'

GPA<- P$GPA

resStan3<- pstan(model_code = stan_code, data = dat, chains = 10, iter = 3000, warmup = 600, thin = 1)
```

Hierarchical Heckman Selection

```
library(MASS)

#--------------------------------------------------------------------------
#                                simulate data
#--------------------------------------------------------------------------

# set-up for observations
N = 100; # number of customers
nT = sample(10:30,N,replace=T); # number of times they are observed
M = sum(nT);
ID = rep(1:N,nT)     # ID must go from 1 to M, links each observation with a customer id.

# numbers of covariates for the two equations
nB1 = 5;
nB2 = 8;
nB = nB1 + nB2;

# covariance of the two equations
rho = 0.2;
sigma = 2;
Cov = matrix(c(1,rho*sigma,rho*sigma,sigma^2),2,2);

# demographics for hete
zC = matrix(c(2,0.6,0.6,2),2,2);
Z = as.matrix(cbind(1,mvrnorm(N,c(0,0),zC)));
nZ = ncol(Z);

#  prior for beta
mu = matrix(runif(nB*nZ,-1,1),nB,);
Lambda = diag(nB);

# covariates
X1 = cbind(1, matrix(rnorm(M*(nB1-1)),,nB1-1));
X2 = cbind(1, matrix(rnorm(M*(nB2-1)),,nB2-1));

# simulate actions and preferences
```

```
visit = array(0,M);
spend = array(0,M);
beta = matrix(0,N,nB);
for( i in 1:N ){
  beta[i,] = mvrnorm(1, mu%*%Z[i,],Lambda);
  beta1 = beta[1:nB1];
  beta2 = beta[(nB1+1):nB];
  for( t in 1:nT[i] ){
    if( i == 1){
      ind = t;
    }else{
      ind = sum(nT[1:(i-1)]) + t;
    }
    error = mvrnorm(1,c(0,0),Cov);
    u_it = X1[ind,]%*%beta1 + error[1];
    if( u_it >0 ){
      visit[ind] = 1;
      spend[ind] = X2[ind,]%*%beta2 + error[2] +
        Cov[1,2]*dnorm(X1[ind,]%*%beta1,0,1)/pnorm(X1[ind,]%*%beta1,0,1);
    }
  }
}



#---------------------------------------------------------------------------
#                              define the model
#---------------------------------------------------------------------------

heckman_hete = "
data {
int<lower=0> N;           // number of customers
int<lower=0> M;           // total number of observations
int<lower=0> nB1;         // number of beta in equation 1
int<lower=0> nB2;         // number of beta in equation 2
int<lower=0> nB;
int<lower=0> nZ;          // number of individual demographic variables plus intercept
int<lower=0> ID[M];       // index for the observations
```

```
matrix[N,nZ] Z;
matrix[M,nB1] X1;
matrix[M,nB2] X2;
vector[M] Y1;
vector[M] Y2;
}
parameters {

matrix[nB,nZ] mu;                       // prior matrix for beta
matrix[nB,N] error_beta;                  // a trick on the variance of beta

// cholesky_factor_corr[nB] L_Omega_beta;   // the correlation matrix of the two equations
vector<lower=0>[nB] sigma_beta;            // the s.t.d of the second equation

real<lower=-1,upper=1> rho_eq;             // the correlation matrix of the two equations
real<lower=0> sigma_eq;                   // the s.t.d of the second equation

}
transformed parameters {

matrix[N,nB] beta;
// beta <- Z*mu' + (diag_pre_multiply(sigma_beta,L_Omega_beta)*error_beta)';
beta <- Z*mu' + (diag_matrix(sigma_beta)*error_beta)';

}
model{

matrix[N,nB1] beta1;
matrix[N,nB2] beta2;

vector[M] lp;
vector[M] value;

//rho_eq ~ uniform(-0.5,0.5);              // correlation matrix of two equations
sigma_eq ~ cauchy(0,2);                   // constrain it to be positive

//L_Omega_beta ~ lkj_corr_cholesky(2);       // correlation matrix of beta vector
```

```
sigma_beta ~ cauchy(0,2);                  // s.t.d. of the covariance matrix (here cause problems)

to_vector(mu) ~ normal(0,3);
to_vector(error_beta) ~ normal(0,1);

// transform beta for two equations for each customer
beta1 <- block(beta,1,1,N,nB1);
beta2 <- block(beta,1,nB1+1,N,nB2);

// calculate the likelihood
for( m in 1:M){
if( Y1[m] == 0 ){
lp[m] <- normal_cdf_log(-X1[m]*beta1[ID[m]]',0,1);
}
else{
value[m] <- (X1[m]*beta1[ID[m]]'+rho_eq/sigma_eq*(Y2[m]-X2[m]*beta2[ID[m]]'))/(1-rho_eq^2)^0.5;
lp[m] <- normal_cdf_log(value[m],0,1) - log(sigma_eq) - (Y2[m]-X2[m]*beta2[ID[m]]')^2/2/sigma_eq^2;
}
}

increment_log_prob( log_sum_exp( lp ) );
}
generated quantities {

vector[nB] m_beta;
vector[nB] sd_beta;

// heterogeneity
for( i in 1:nB ){
m_beta[i] <- mean( col(beta,i) );
sd_beta[i] <- sd( col(beta,i) );
}

}
"
```

```
#------------------------------------------------------------------------------
#                              run the model
#------------------------------------------------------------------------------

#------------------
# initialization
#------------------

data_list = list(N=N, M=M, nB1=nB1, nB2=nB2, nB=nB, nZ=nZ, ID=ID,
                 Z=Z, X1=X1, X2=X2, Y1=visit, Y2=spend);

#-------------------
#  mcmc
#-------------------

fit =
stan(model_code=heckman_hete,data=data_list,warmup=1000,iter=2000,chains=1,control=list(adapt_gamma=0.99,
stepsize=0.005))

#-------------------
#  analysis
#-------------------
print(fit, pars = c("m_beta", "sigma_eq","rho_eq"))

summary(fit)
```

Ordered Logit Model

```
library(rstan)
library(loo)
library(rstanmulticore)

N <- 100  # number of objects
K <- 4  # number of categories
L <- 10  # number of groups
y <- sample(1:K, N, replace = T)
group <- sample(1:L, N, replace = T)
sex <- sample(0:1, N, replace = T)
age <- sample(50:100, N, replace = T)

code <- '  // STAN: hierarchical ordered logistic model
data {
int<lower=0> N;  // number of objects
int<lower=1> K;  // number of categories
int<lower=1> L;  // number of groups
int<lower=1,upper=K> y[N];  // response
vector<lower=0,upper=1>[N] sex;  // predictor
vector<lower=1>[N] age;  // predictor
int<lower=1,upper=L> group[N];  // group level predictor
}
parameters {
vector[L] alpha;  // intercept
real muAlpha;  // mu for intercept
real<lower=0> sigmaAlpha;  // sigma for intercept
real beta1;  // slope for sex
real beta2;  // slope for age
ordered[K-1] c;  // cutpoints for ordered logistic model
}
transformed parameters {
vector[N] y_hat;
for (i in 1:N)
y_hat[i] <- alpha[group[i]] + beta1 * sex[i] + beta2 * age[i];
}
model {
```

```
alpha ~ normal(muAlpha, sigmaAlpha);  // intercept
for (n in 1:N)
y[n] ~ ordered_logistic(y_hat[n], c);
//  y[n] ~ ordered_logistic(x[n] * beta, c);
//  increment_log_prob(ordered_logistic_log(y[n], x[n] * beta, c));
}
generated quantities {
vector[N] log_lik;
for (n in 1:N)
log_lik[n] <- ordered_logistic_log(y[n], y_hat[n], c);
//  log_lik[n] <- ordered_logistic_log(y[n], x[n] * beta, c);
}
'

dataList <- c("N", "K", "L", "y", "group", "sex", "age")
model <- stan_model(model_code = code, model_name='ordered_logitistic', verbose=FALSE)

fit <- sampling(model, data = dataList, iter = 10, chains = 1)
```

HLM Ordered Logit Model

```
P<- Purdue2
P$Majors<- as.factor(P$Majors)
P$Class2<- as.factor(P$Class2)
P$Name<- as.factor(P$Name)
P$Time<- P$T
P$Grade<- Purdue2$Grade
P$Name_index<- as.integer(as.factor(P$Name))
P$Majors_index<- as.integer(as.factor(P$Majors))
P$Class2_index <- as.integer(as.factor(P$Class2))
P$Grade<- as.integer(as.factor(P$Grade))
as.factor(P$Grade)

dat <- with(P,
        list(Ni          = length(unique(R)),
             Nk          = length(unique(Majors_index)),
             K           = length(unique(Grade)),
             Time        = Time,
             Lag_GPA     = L1_Term_GPA,
             Repeated    = Repeated,
             Summer      = Summer,
             Class_Taken  = Class_Taken,
             T_Units_Attempted = T_Units_Attempted,
             Unit_Difference =  Unit_Difference,
             Major         = Majors_index,
             EM            = EM,
             MS            = MS,
             CM            = CM,
             CE            = CE,
             CO            = CO,
             CS            = CS,
             CB          = CB,
             CMS         = CMS,
             Rtimes      = Rtimes,
             Grade     = as.integer(as.factor(Grade))))
```

```
#Linear Mixed Effects Model (random intercepts with fixed effect covariates)

stan_code<- 'data {
// Define variables in data
// Number of level-1 observations (an integer)
int<lower=0> Ni;
// Number of level-2 clusters
int<lower=0> Nk;
// Number of Categories for the independent variable
int<lower=1> K;

// Cluster ID
int<lower=1> Major[Ni];

//Response Variable
int<lower=1,upper=K> Grade[Ni];
// Continuous predictor
// Continuous predictor
real Repeated[Ni];
real Summer[Ni];
real Class_Taken[Ni];
real T_Units_Attempted[Ni];
real Unit_Difference[Ni];
real Time[Ni];
real Lag_GPA[Ni];
real MS[Ni];
real EM[Ni];
real CM[Ni];
real CMS[Ni];
real CS[Ni];
real CB[Ni];
real CE[Ni];
real CO[Ni];
real Rtimes[Ni];
}
parameters {
// Define parameters to estimate
```

```
// Fixed effects
real beta_0;
real beta_1;
real beta_2;
real beta_3;
real beta_4;
real beta_5;
real beta_6;
real beta_7;
real beta_8;
real beta_9;
real beta_10;
real beta_11;
real beta_12;
real beta_13;
real beta_14;
real beta_15;
real beta_16;
// cut points
ordered[K-1] c;
// Level-1 errors
real<lower=0> sigma_e0;
// Level-2 random effect
real u_0k[Nk];
real<lower=0> sigma_u0k;
real u_1k[Nk];
real<lower=0> sigma_u1k;
real u_2k[Nk];
real<lower=0> sigma_u2k;
real u_3k[Nk];
real<lower=0> sigma_u3k;
real u_4k[Nk];
real<lower=0> sigma_u4k;
real u_5k[Nk];
real<lower=0> sigma_u5k;
real u_6k[Nk];
real<lower=0> sigma_u6k;
```

```
}
transformed parameters  {
// Varying intercepts
real beta_0k[Nk];
real beta_1k[Nk];
real beta_2k[Nk];
real beta_3k[Nk];
real beta_4k[Nk];
real beta_5k[Nk];
real beta_6k[Nk];
// Individual mean
real mu[Ni];
for (k in 1:Nk) {
beta_0k[k] <- beta_0 + u_0k[k];
beta_1k[k] <- beta_10 + u_1k[k];
beta_2k[k] <- beta_11 + u_2k[k];
beta_3k[k] <- beta_12 + u_3k[k];
beta_4k[k] <- beta_13 + u_4k[k];
beta_5k[k] <- beta_14 + u_5k[k];
beta_6k[k] <- beta_15 + u_6k[k];
}
// Individual mean
for (i in 1:Ni) {
mu[i] <- beta_0k[Major[i]] +
beta_1 * Lag_GPA[i] + beta_2 * Time[i] +
beta_3 * Summer[i] + beta_4 * Class_Taken[i] +
beta_5 * T_Units_Attempted[i] + beta_6 * Unit_Difference[i] +
beta_7 * Repeated[i] + beta_8 * EM[i] + beta_9 * MS[i] +
beta_1k[Major[i]] * CM[i] + beta_2k[Major[i]] * CB[i] +
beta_3k[Major[i]] * CS[i] + beta_4k[Major[i]] * CO[i] +
beta_5k[Major[i]] * CMS[i] + beta_6k[Major[i]] * CE[i] + beta_16 * Rtimes[i];
}
}
model {
// Prior part of Bayesian inference
// Flat prior for mu (no need to specify if non-informative)
beta_0 ~ normal(0, 10);
```

```
beta_1 ~ normal(0, 10);
beta_2 ~ normal(0, 10);
beta_3 ~ normal(0, 10);
beta_4 ~ normal(0, 10);
beta_5 ~ normal(0, 10);
beta_6 ~ normal(0, 10);
beta_7 ~ normal(0, 10);
beta_8 ~ normal(0, 10);
beta_9 ~ normal(0, 10);
// Random effects distribution
u_0k  ~ normal(0, sigma_u0k);
u_1k  ~ normal(0, sigma_u1k);
u_2k  ~ normal(0, sigma_u2k);
u_3k  ~ normal(0, sigma_u3k);
u_4k  ~ normal(0, sigma_u4k);
u_5k  ~ normal(0, sigma_u5k);
u_6k  ~ normal(0, sigma_u6k);
// Likelihood part of Bayesian inference
// Outcome model N(mu, sigma^2) (use SD rather than Var)
for (i in 1:Ni) {
Grade[i] ~ ordered_logistic(mu[i], c);
}}
generated quantities {
vector[Ni] log_lik;
for (i in 1:Ni){
log_lik[i] <- ordered_logistic_log(Grade[i] , mu[i], c); // used for fit statistics and model comparisons
}}'

resStan2<- pstan(model_code = stan_code, model_name='ordered_logitistic', verbose=FALSE, data = dat,
                 chains = 1, iter = 10, warmup = 1, thin = 1)
```

Text Miner: Simulated Data

```
##########################################
"Sentiment analysis with machine learning"
##########################################
 library(Sentiment)
library(RTextTools)
pos_tweets =  rbind(
  c('I love this car', 'positive'),
  c('This view is amazing', 'positive'),
  c('I feel great this morning', 'positive'),
  c('I am so excited about the concert', 'positive'),
  c('He is my best friend', 'positive'),
  c('Its the bst thing ever', 'positive'),
  c('Dude, its freaken awesome', 'positive'),
  c('Its dope, word son!', 'positive'),
  c('Its a sick apartment, youre so lucky', 'positive'),
  c('You can do anything', 'positive')
)
neg_tweets = rbind(
  c('I do not like this car', 'negative'),
  c('This view is horrible', 'negative'),
  c('I feel tired this morning', 'negative'),
  c('I am not looking forward to the concert', 'negative'),
  c('He is my enemy', 'negative'),
  c('Its an incredibly ugly shirt', 'negative'),
  c('She is a vicious person', 'negative'),
  c('They totally sucked at their jobs', 'negative'),
  c('He is the worst human being ever', 'negative'),
  c('Its horrifying and nauseating', 'negative')
)
test_tweets = rbind(
  c('feel happy this morning', 'positive'),
  c('larry friend', 'positive'),
  c('not like that man', 'negative'),
  c('house not great', 'negative'),
  c('your song annoying', 'negative')
)
```

```
tweets = rbind(pos_tweets, neg_tweets, test_tweets)


matrix= create_matrix(tweets[,1], language="english",
                      removeStopwords=TRUE, removeNumbers=TRUE,  # we can also removeSparseTerms
                      stemWords=FALSE)
# train the model
mat = as.matrix(matrix)
library(e1071)
classifier = naiveBayes(mat[1:20,], as.factor(tweets[1:20,2]) )

predicted = predict(classifier, mat[21:25,])

predicted

table(tweets[21:25, 2], predicted)
recall_accuracy(tweets[21:25, 2], predicted)


container = create_container(matrix, as.numeric(as.factor(tweets[,2])),
                             trainSize=1:20, testSize=21:25,virgin=FALSE)

models = train_models(container, algorithms=c("MAXENT" , "SVM", "RF", "BAGGING", "TREE"))
results = classify_models(container, models)


# recall accuracy
recall_accuracy(as.numeric(as.factor(tweets[21:25, 2])), results[,"FORESTS_LABEL"])
recall_accuracy(as.numeric(as.factor(tweets[21:25, 2])), results[,"MAXENTROPY_LABEL"])
recall_accuracy(as.numeric(as.factor(tweets[21:25, 2])), results[,"TREE_LABEL"])
recall_accuracy(as.numeric(as.factor(tweets[21:25, 2])), results[,"BAGGING_LABEL"])
recall_accuracy(as.numeric(as.factor(tweets[21:25, 2])), results[,"SVM_LABEL"])


# model summary
analytics = create_analytics(container, results)
```

```
summary(analytics)
head(analytics@document_summary)
analytics@ensemble_summary
N=4
set.seed(2014)
cross_validate(container,N,"MAXENT")
cross_validate(container,N,"TREE")
cross_validate(container,N,"SVM")
cross_validate(container,N,"RF")

analytics2<- create_analytics(container)

tweet<- ifelse(tweets[,2] == 'positive',1,0)

tweet

df<- data.frame(cbind(tweet,mat))

k=2
n= floor(nrow(df)/k) #n is the size of each fold
#I rounded down to avoid going out of bounds on the last fold
err.vect = rep(NA,k) #store the error in this vector
#show to partition the first fold
ntrees = 200 #the default is only 100
for ( i in 1:k) {
  s1 = ((i - 1)*n+1) #the start of the subset
  s2 = (i * n)  #the end of the subset
  subset = s1:s2 #the range of the subset

  cv.train = df[-subset,]
  cv.test = df[subset,] #test the model's performance on this data

  #estimates the gbm on the cv.train set
  fit<- gbm.fit(x  = cv.train[,-1], y = cv.train[,1],
                n.tree = ntrees, verbose= FALSE, shrinkage = .00000001,
                interaction.depth =1, n.minobsinnode = 1, bag.fraction =.75,
                distribution = 'adaboost', nTrain = NULL)
```

```
  #use bernouli or adaboost for classification problems
  #make predictions on the test set

  prediction = predict(fit, newdata=cv.test[,-1], n.trees = ntrees)
  err.vect[i] = roc.area(cv.test[,1], prediction)$A
  print(paste("AUC for fold", i, ":", err.vect[i]))

}
print(paste("Average AUC: " , mean(err.vect)))
```

Text Miner = Twitter Data

```
####################
"load data"
####################

#Read data:

happy = readLines("Tweet/happy.txt", encoding = 'UTF-8')
sad = readLines("Tweet/sad.txt")
happy_test = readLines("Tweet/happy_test.txt")
sad_test = readLines("Tweet/sad_test.txt")


tweet = c(happy, sad)
tweet_test= c(happy_test, sad_test)
tweet_all = c(tweet, tweet_test)
sentiment = c(rep("happy", length(happy) ),
              rep("sad", length(sad)))
sentiment_test = c(rep("happy", length(happy_test) ),
                   rep("sad", length(sad_test)))
sentiment_all = as.factor(c(sentiment, sentiment_test))

sentiment_all

#First, try naive Bayes.
```

```
mat= create_matrix(tweet_all, language="english",
                removeStopwords=FALSE, removeNumbers=TRUE,
                stemWords=FALSE, tm::weightTfIdf)

mat = as.matrix(mat)

classifier = naiveBayes(mat[1:160,], as.factor(sentiment_all[1:160]))
predicted = predict(classifier, mat[161:180,]); predicted

table(sentiment_test, predicted)
recall_accuracy(sentiment_test, predicted)



# the other methods

mat= create_matrix(tweet_all, language="english",
                removeStopwords=FALSE, removeNumbers=TRUE,
                stemWords=FALSE, tm::weightTfIdf)
container = create_container(mat, as.numeric(sentiment_all),
                        trainSize=1:160, testSize=161:180,virgin=FALSE) #removeSparseTerms

models = train_models(container, algorithms=c("MAXENT",
                                        "SVM",
                                        #"GLMNET", "BOOSTING",
                                        "SLDA","BAGGING",
                                        "RF", # "NNET",
                                        "TREE"
))

# test the model
results = classify_models(container, models)

table(as.numeric(as.numeric(sentiment_all[161:180])), results[,"FORESTS_LABEL"])

recall_accuracy(as.numeric(as.numeric(sentiment_all[161:180])), results[,"FORESTS_LABEL"])

# formal tests
```

```
analytics = create_analytics(container, results)
summary(analytics)

head(analytics@algorithm_summary)
head(analytics@label_summary)
head(analytics@document_summary)
analytics@ensemble_summary # Ensemble Agreement



# Cross Validation
N=3
cross_SVM = cross_validate(container,N,"SVM")
cross_GLMNET = cross_validate(container,N,"GLMNET")
cross_MAXENT = cross_validate(container,N,"MAXENT")
```

Text Miner: Speeches

```
# set options

options(stringsAsFactors = FALSE)

#set parameters

candidates<- c("Romney", "Obama")
pathname<- "Speeches"

# clean text

cleanCorpus<- function(corpus) {
  corpus.tmp<- tm_map(corpus, removePunctuation)
  corpus.tmp<- tm_map(corpus.tmp, stripWhitespace)
  corpus.tmp<- tm_map(corpus.tmp, content_transformer(tolower))
  corpus.tmp<- tm_map(corpus.tmp, removeWords, stopwords('english'))
  return(corpus.tmp)
}

# build TDM
```

```
generateTDM<- function(cand,path) {
  s.dir<- sprintf("%s/%s",path,cand) # prints the two varibles together as strings - the path with the candidate name
  s.cor<- Corpus(DirSource(directory = s.dir, encoding = "UTF-8"))
  s.cor.cl<- cleanCorpus(s.cor)
  s.tdm<- TermDocumentMatrix(s.cor.cl) #each term speech with words will be quantified with a candidate
  s.tdm<- removeSparseTerms(s.tdm, .7) # removes or filters a portion of the text to speed up analysis
  result<- list(name = cand, tdm = s.tdm) # returns a list
}

tdm<- lapply(candidates, generateTDM, path = pathname)

# attach name

bindcandidatetoTDM<- function(tdm) {
  s.mat<- t(data.matrix(tdm[["tdm"]]))
  s.df<- as.data.frame(s.mat, stringsAsFactors = FALSE) # Converts this to a dataframe
  s.df<- cbind(s.df, rep(tdm[['name']], nrow(s.df)))
  colnames(s.df)[ncol(s.df)] <- 'targetcandidate'
  return(s.df)
}

candTDM<- lapply(tdm,bindcandidatetoTDM)

str(candTDM)
#stack
tdm.stack<- do.call(rbind.fill, candTDM) # it will fill an NA when there are terms in one speech but not in another
tdm.stack[is.na(tdm.stack)] <- 0

head(tdm.stack) # every row represents a speech. Every columnn represents a term
nrow(tdm.stack)

#hold-out

train.idx<- sample(nrow(tdm.stack), ceiling(nrow(tdm.stack) * .7)) # takes 70 percent of the rows as a training set
test.idx<- (1:nrow(tdm.stack))[-train.idx]
```

```
# model - KNN

tdm.cand <- tdm.stack[,"targetcandidate"] # include all rows but just the column targetcandidate, which should be just 0 and 1
tdm.stack.nl<- tdm.stack[, !colnames(tdm.stack) %in% "targetcandidate"] # all of the columns except for the target candidate

knn.pred<- knn(tdm.stack.nl[train.idx,],tdm.stack.nl[test.idx,],tdm.cand[train.idx])

conf.mat<- table('predictions' = knn.pred, Actual = tdm.cand[test.idx])

conf.mat

(accuracy<- sum(diag(conf.mat) / length(test.idx) * 100))
```

Text Miner = Twitter Data

```
###################
"load data"
###################

#Read data:

happy = readLines("Tweet/happy.txt", encoding = 'UTF-8')
sad = readLines("Tweet/sad.txt")
happy_test = readLines("Tweet/happy_test.txt")
sad_test = readLines("Tweet/sad_test.txt")


tweet = c(happy, sad)
tweet_test= c(happy_test, sad_test)
tweet_all = c(tweet, tweet_test)
sentiment = c(rep("happy", length(happy) ),
              rep("sad", length(sad)))
sentiment_test = c(rep("happy", length(happy_test) ),
                   rep("sad", length(sad_test)))
sentiment_all = as.factor(c(sentiment, sentiment_test))
```

```
sentiment_all

#First, try naive Bayes.

mat= create_matrix(tweet_all, language="english",
                removeStopwords=FALSE, removeNumbers=TRUE,
                stemWords=FALSE, tm::weightTfIdf)

mat = as.matrix(mat)

classifier = naiveBayes(mat[1:160,], as.factor(sentiment_all[1:160]))
predicted = predict(classifier, mat[161:180,]); predicted

table(sentiment_test, predicted)
recall_accuracy(sentiment_test, predicted)
# the other methods
mat= create_matrix(tweet_all, language="english",
                removeStopwords=FALSE, removeNumbers=TRUE,
                stemWords=FALSE, tm::weightTfIdf)
container = create_container(mat, as.numeric(sentiment_all),
                        trainSize=1:160, testSize=161:180,virgin=FALSE) #removeSparseTerms
 models = train_models(container, algorithms=c("MAXENT",
                                        "SVM",
                                        #"GLMNET", "BOOSTING",
                                        "SLDA","BAGGING",
                                        "RF", # "NNET",
                                        "TREE"
))
# test the model
results = classify_models(container, models)

table(as.numeric(as.numeric(sentiment_all[161:180])), results[,"FORESTS_LABEL"])

recall_accuracy(as.numeric(as.numeric(sentiment_all[161:180])), results[,"FORESTS_LABEL"])

# formal tests
```

```
analytics = create_analytics(container, results)
summary(analytics)

head(analytics@algorithm_summary)
head(analytics@label_summary)
head(analytics@document_summary)
analytics@ensemble_summary # Ensemble Agreement



# Cross Validation
N=3
cross_SVM = cross_validate(container,N,"SVM")
cross_GLMNET = cross_validate(container,N,"GLMNET")
cross_MAXENT = cross_validate(container,N,"MAXENT")



ADABOOST
rm(list=ls())
set.seed(973487)

load(url('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.sav'))

titanic3<- subset(titanic3, select=c(survived,pclass,sex,age,sibsp))

# separate into training and test data
o<- order(runif(dim(titanic3)[1]))
titanic.train<- titanic3[o[1:655],]
titanic.pred<- titanic3[o[656:1309],]
length(levels(titanic.train$response))
table(as.logical(tmp_train))

#run the model on thr training data
titanic.ada<- ada(survived ~ pclass + sex + age + sibsp, data=titanic.train, verbose=TRUE, na.action=na.rpart,
iter=50)

#... and predict the test data
titanic.ada<- addtest(titanic.ada, test.x=titanic.pred[,-1], test.y=titanic.pred[,1])
```

```
# error reduction for the progressive splits
plot(titanic.ada, test=TRUE)

#variable importance plotting/similar to random forests
varplot(titanic.ada)

#out of sample predictions on the test data set
titanic.oos.predict<- predict(titanic.ada, newdata=titanic.pred, type = 'vector')

# out of sample survivors + deaths correctly predicted
sum(titanic.oos.predict== titanic.pred$survived)/length(titanic.oos.predict)

#survivors only
s<- which(titanic.pred$survived==1)
sum(titanic.oos.predict[s]==titanic.pred$survived[s])/length(titanic.oos.predict[s])

#deaths only
s<- which(titanic.pred$survived==0)
sum(titanic.oos.predict[s]==titanic.pred$survived[s])/length(titanic.oos.predict[s])
```

GBM

```
#close variant(that also incorporates bagging) can also be done using the gbm package. OOB is out of bag
titanic.gbm<- gbm(survived ~ pclass + sex + age + sibsp, data=titanic.train, distribution = 'adaboost', n.trees = 1000, verbose = FALSE)

# this tells us the appropriate number of iterations while also showing us performance characteristics by plotting the training error and the valid predictions error.
#the blue line tells us an approximate number of optimal iterations, but the error suggests its underestimating the total iterations needed
gbm.perf(titanic.gbm,method='OOB')

#marginal effects plots - they average over all of the independent variables
plot.gbm(titanic.gbm,1)
plot.gbm(titanic.gbm,2)
plot.gbm(titanic.gbm,3)
plot.gbm(titanic.gbm,4)
```

```
#out of sample predictions on the test data set
titanic.oos.predict<- ifelse(plogis(predict(titanic.gbm, newdata=titanic.pred, type = 'link', n.trees = 1000))>.5,1,0)

#out of sample survivors plus deaths correctly predicted
sum(titanic.oos.predict==titanic.pred$survived)/length(titanic.oos.predict)

#survivors only
s<- which(titanic.pred$survived==1)
sum(titanic.oos.predict[s]==titanic.pred$survived[s])/length(titanic.oos.predict[s])

#deaths only
s<- which(titanic.pred$survived==0)
sum(titanic.oos.predict[s]==titanic.pred$survived[s])/length(titanic.oos.predict[s])
```

Ensemble Methods

```
#ensemble models: flexibile mixture modeling
x<- runif(100)
class<- rbinom(100,size=1, prob=.5)

#creates two classes
y<- class*(2-2*x) + (1-class)*(-2 + 2*x) + rnorm(100,sd=.5)
plot(y~x, col=class+2)

require(flexmix)

#k has to be told explicitly
flex.mod<- flexmix(y~x, k=2)

# tells us the fit statistics plus how many were classified in each class
summary(flex.mod)

#provides the summary of each
parameters(flex.mod)

#used to predict the lines and the classifications for each data point
```

```
#creating a sequence of x as an order of points
newdata=data.frame(x=seq(from=0, to=1, by=.1))
pred.line<- predict(flex.mod,newdata)
pred.class<- ifelse(flex.mod@posterior$scaled[,1]>flex.mod@posterior$scaled[,2],'red','green')

plot(y~x,col=pred.class)
lines(pred.line[[1]]~newdata$x, col='red', lty=2)
lines(pred.line[[2]]~newdata$x, col='green',lty=2)



# more complex example
x<- runif(1000,min=0, max=10)
y<- ifelse(x<5, sin(pi*0.5*x), 2*x-10) +rnorm(1000,sd=1)
plot(y~x)

flex.mod<- flexmix(y~x, k=4)
summary(flex.mod)
parameters(flex.mod)

newdata<- data.frame(x=seq(from=0, to=10, by =.1))
pred.line<- predict(flex.mod,newdata)
preds<- flex.mod@posterior$scaled

maxpost<- apply(X=preds, FUN=max, MARGIN=1)
for(i in 1:length(maxpost)){
  pred.class[i]<- which(preds[i,]==maxpost[i])
}
color<- c("black", "red", "blue", "green")

pred.class.col<- color[as.numeric(pred.class)]



plot(y~x,col=pred.class.col)

lines(pred.line[[1]]~newdata$x, col='black', lty=2, lwd=2)
lines(pred.line[[2]]~newdata$x, col='red', lty=2, lwd=2)
```

```
lines(pred.line[[3]]~newdata$x, col='blue', lty=2, lwd=2)
lines(pred.line[[4]]~newdata$x, col='green', lty=2, lwd=2)
```

Cross Validation Random Forest

```
#create data:
x1 = rnorm(1000)           # some continuous variables
x2 = rnorm(1000)
z = 1 + 2*x1 + 3*x2        # linear combination with a bias
pr = 1/(1+exp(-z))         # pass through an inv-logit function
y = y = rbinom(1000,1,pr)  # bernoulli response variable
df = data.frame(y=y,x1=x1,x2=x2)
train<- df
# For K fold cv, you create k different positions in the data
# and I am already assuming my data is in a random order
k=10
n= floor(nrow(train)/k) #n is the size of each fold
#I rounded down to avoid going out of bounds on the last fold
 err.vect = rep(NA,k) #store the error in this vector
 #show to partition the first fold

#k-fold Cv allows us to use all of the data for the final model
#but still have realistic model performance estimates

#next, move to the second fold:
#need to loop over each of the folds
for(i in 1:k) {
  s1 = ((i -1 ) * n + 1)
  s2 = (i * n)
  subset = s1:s2
    cv.train = train[-subset,]
  cv.test = train[subset,]
   #run the random forest as a train set
```

```
  fit <- randomForest(x = cv.train[,-1], y = as.factor(cv.train[,1]))
  #make predictions on the test set
  prediction <- predict(fit, newdata= cv.test[,-1], type = 'prob')[,2]
  err.vect[i] = roc.area(cv.test[,1], prediction)$A
  print(paste("AUC for fold", i, ":", err.vect[i]))
}
print(paste("Average AUC: ", mean(err.vect)))



print(paste("Average AUC: ", mean(err.vect)))
```

#Cross validation for Generalized Boosted Regression Models

```
ntrees = 1000 #the default is only 100
for ( i in 1:k) {
  s1 = ((i - 1)*n+1) #the start of the subset
  s2 = (i * n)  #the end of the subset
  subset = s1:s2 #the range of the subset

  cv.train = train[-subset,]
  cv.test = train[subset,] #test the model's performance on this data

  #estimates the gbm on the cv.train set
  fit<- gbm.fit(x  = cv.train[,-1], y = cv.train[,1],
            n.tree = ntrees, verbose= FALSE, shrinkage = .005,
            interaction.depth = 20, n.minobsinnode = 5, distribution = ‘adaboost’)
  #use bernouli or adaboost for classification problems
  #make predictions on the test set

  prediction = predict(fit, newdata=cv.test[,-1], n.trees = ntrees)
  err.vect[i] = roc.area(cv.test[,1], prediction)$A
  print(paste("AUC for fold", i, ":", err.vect[i]))

}

print(paste("Average AUC: " , mean(err.vect)))
```

Visual_NN

##Visualizing NN

```
library(clusterGeneration)

seed.val<-2
set.seed(seed.val)

num.vars<-8
num.obs<-1000

#input variables
cov.mat<-genPositiveDefMat(num.vars,covMethod=c("unifcorrmat"))$Sigma
rand.vars<-mvrnorm(num.obs,rep(0,num.vars),Sigma=cov.mat)

#output variables
parms<-runif(num.vars,-10,10)
y1<-rand.vars %*% matrix(parms) + rnorm(num.obs,sd=20)
parms2<-runif(num.vars,-10,10)
y2<-rand.vars %*% matrix(parms2) + rnorm(num.obs,sd=20)

#final datasets
rand.vars<-data.frame(rand.vars)
resp<-data.frame(y1,y2)
names(resp)<-c('Y1','Y2')
dat.in<-data.frame(resp,rand.vars)

#nnet function from nnet package. NNet package can take separate(or combined) x and y inputs as data frames or as a formula
library(nnet)
set.seed(seed.val)
mod1<-nnet(rand.vars,resp,data=dat.in,size=10,linout=T)



#neuralnet function from neuralnet package, notice use of only one response. NeuralNet can only use a formula as input
library(neuralnet)
form.in<-as.formula('Y1~X1+X2+X3+X4+X5+X6+X7+X8')
set.seed(seed.val)
mod2<-neuralnet(form.in,data=dat.in,hidden=10)

#mlp function from RSNNS package. Can only take a data frame as a combined or separate variables as input
```

```
library(RSNNS)
set.seed(seed.val)
mod3<-mlp(rand.vars, resp, size=10,linOut=T)

#Plotting Function for NN's
plot.nnet <- function(mod.in,nid=T,all.out=T,all.in=T,bias=T,wts.only=F,rel.rsc=5,circle.cex=5,
                      node.labs=T,var.labs=T,x.lab=NULL,y.lab=NULL,line.stag=NULL,struct=NULL,cex.val=1,
                      alpha.val=1,circle.col='lightblue',pos.col='black',neg.col='grey', max.sp = F, ...){

  require(scales)

  #sanity checks
  if('mlp' %in% class(mod.in)) warning('Bias layer not applicable for rsnns object')
  if('numeric' %in% class(mod.in)){
    if(is.null(struct)) stop('Three-element vector required for struct')
    if(length(mod.in) != ((struct[1]*struct[2]+struct[2]*struct[3])+(struct[3]+struct[2])))
      stop('Incorrect length of weight matrix for given network structure')
  }
  if('train' %in% class(mod.in)){
    if('nnet' %in% class(mod.in$finalModel)){
      mod.in<-mod.in$finalModel
      warning('Using best nnet model from train output')
    }
    else stop('Only nnet method can be used with train object')
  }

  #gets weights for neural network, output is list
  #if rescaled argument is true, weights are returned but rescaled based on abs value
  nnet.vals<-function(mod.in,nid,rel.rsc,struct.out=struct){

    require(scales)
    require(reshape)

    if('numeric' %in% class(mod.in)){
      struct.out<-struct
      wts<-mod.in
    }

    #neuralnet package
    if('nn' %in% class(mod.in)){
      struct.out<-unlist(lapply(mod.in$weights[[1]],ncol))
```

```
  struct.out<-struct.out[-length(struct.out)]
  struct.out<-c(
    length(mod.in$model.list$variables),
    struct.out,
    length(mod.in$model.list$response)
  )
  wts<-unlist(mod.in$weights[[1]])
}

#nnet package
if('nnet' %in% class(mod.in)){
  struct.out<-mod.in$n
  wts<-mod.in$wts
}

#RSNNS package
if('mlp' %in% class(mod.in)){
  struct.out<-c(mod.in$nInputs,mod.in$archParams$size,mod.in$nOutputs)
  hid.num<-length(struct.out)-2
  wts<-mod.in$snnsObject$getCompleteWeightMatrix()

  #get all input-hidden and hidden-hidden wts
  inps<-wts[grep('Input',row.names(wts)),grep('Hidden_2',colnames(wts)),drop=F]
  inps<-melt(rbind(rep(NA,ncol(inps)),inps))$value
  uni.hids<-paste0('Hidden_',1+seq(1,hid.num))
  for(i in 1:length(uni.hids)){
    if(is.na(uni.hids[i+1])) break
    tmp<-wts[grep(uni.hids[i],rownames(wts)),grep(uni.hids[i+1],colnames(wts)),drop=F]
    inps<-c(inps,melt(rbind(rep(NA,ncol(tmp)),tmp))$value)
  }

  #get connections from last hidden to output layers
  outs<-wts[grep(paste0('Hidden_',hid.num+1),row.names(wts)),grep('Output',colnames(wts)),drop=F]
  outs<-rbind(rep(NA,ncol(outs)),outs)

  #weight vector for all
  wts<-c(inps,melt(outs)$value)
  assign('bias',F,envir=environment(nnet.vals))
}

if(nid) wts<-rescale(abs(wts),c(1,rel.rsc))
```

```
  #convert wts to list with appropriate names
  hid.struct<-struct.out[-c(length(struct.out))]
  row.nms<-NULL
  for(i in 1:length(hid.struct)){
    if(is.na(hid.struct[i+1])) break
    row.nms<-c(row.nms,rep(paste('hidden',i,seq(1:hid.struct[i+1])),each=1+hid.struct[i]))
  }
  row.nms<-c(
    row.nms,
    rep(paste('out',seq(1:struct.out[length(struct.out)])),each=1+struct.out[length(struct.out)-1])
  )
  out.ls<-data.frame(wts,row.nms)
  out.ls$row.nms<-factor(row.nms,levels=unique(row.nms),labels=unique(row.nms))
  out.ls<-split(out.ls$wts,f=out.ls$row.nms)

  assign('struct',struct.out,envir=environment(nnet.vals))

  out.ls

}

wts<-nnet.vals(mod.in,nid=F)

if(wts.only) return(wts)

#circle colors for input, if desired, must be two-vector list, first vector is for input layer
if(is.list(circle.col)){
  circle.col.inp<-circle.col[[1]]
  circle.col<-circle.col[[2]]
}
else circle.col.inp<-circle.col

#initiate plotting
x.range<-c(0,100)
y.range<-c(0,100)
#these are all proportions from 0-1
if(is.null(line.stag)) line.stag<-0.011*circle.cex/2
layer.x<-seq(0.17,0.9,length=length(struct))
bias.x<-layer.x[-length(layer.x)]+diff(layer.x)/2
bias.y<-0.95
```

```
circle.cex<-circle.cex

#get variable names from mod.in object
#change to user input if supplied
if('numeric' %in% class(mod.in)){
  x.names<-paste0(rep('X',struct[1]),seq(1:struct[1]))
  y.names<-paste0(rep('Y',struct[3]),seq(1:struct[3]))
}
if('mlp' %in% class(mod.in)){
  all.names<-mod.in$snnsObject$getUnitDefinitions()
  x.names<-all.names[grep('Input',all.names$unitName),'unitName']
  y.names<-all.names[grep('Output',all.names$unitName),'unitName']
}
if('nn' %in% class(mod.in)){
  x.names<-mod.in$model.list$variables
  y.names<-mod.in$model.list$respons
}
if('xNames' %in% names(mod.in)){
  x.names<-mod.in$xNames
  y.names<-attr(terms(mod.in),'factor')
  y.names<-row.names(y.names)[!row.names(y.names) %in% x.names]
}
if(!'xNames' %in% names(mod.in) & 'nnet' %in% class(mod.in)){
  if(is.null(mod.in$call$formula)){
    x.names<-colnames(eval(mod.in$call$x))
    y.names<-colnames(eval(mod.in$call$y))
  }
  else{
    forms<-eval(mod.in$call$formula)
    x.names<-mod.in$coefnames
    facts<-attr(terms(mod.in),'factors')
    y.check<-mod.in$fitted
    if(ncol(y.check)>1) y.names<-colnames(y.check)
    else y.names<-as.character(forms)[2]
  }
}
#change variables names to user sub
if(!is.null(x.lab)){
  if(length(x.names) != length(x.lab)) stop('x.lab length not equal to number of input variables')
  else x.names<-x.lab
}
```

```
if(!is.null(y.lab)){
 if(length(y.names) != length(y.lab)) stop('y.lab length not equal to number of output variables')
 else y.names<-y.lab
}

#initiate plot
plot(x.range,y.range,type='n',axes=F,ylab='',xlab='',...)

#function for getting y locations for input, hidden, output layers
#input is integer value from 'struct'
get.ys<-function(lyr, max_space = max.sp){
 if(max_space){
  spacing <- diff(c(0*diff(y.range),0.9*diff(y.range)))/lyr
 } else {
  spacing<-diff(c(0*diff(y.range),0.9*diff(y.range)))/max(struct)
 }

 seq(0.5*(diff(y.range)+spacing*(lyr-1)),0.5*(diff(y.range)-spacing*(lyr-1)),
    length=lyr)
}

#function for plotting nodes
#'layer' specifies which layer, integer from 'struct'
#'x.loc' indicates x location for layer, integer from 'layer.x'
#'layer.name' is string indicating text to put in node
layer.points<-function(layer,x.loc,layer.name,cex=cex.val){
 x<-rep(x.loc*diff(x.range),layer)
 y<-get.ys(layer)
 points(x,y,pch=21,cex=circle.cex,col=in.col,bg=bord.col)
 if(node.labs) text(x,y,paste(layer.name,1:layer,sep=''),cex=cex.val)
 if(layer.name=='I' & var.labs) text(x-line.stag*diff(x.range),y,x.names,pos=2,cex=cex.val)
 if(layer.name=='O' & var.labs) text(x+line.stag*diff(x.range),y,y.names,pos=4,cex=cex.val)
}

#function for plotting bias points
#'bias.x' is vector of values for x locations
#'bias.y' is vector for y location
#'layer.name' is  string indicating text to put in node
bias.points<-function(bias.x,bias.y,layer.name,cex,...){
 for(val in 1:length(bias.x)){
  points(
```

```
    diff(x.range)*bias.x[val],
    bias.y*diff(y.range),
    pch=21,col=in.col,bg=bord.col,cex=circle.cex
   )
   if(node.labs)
    text(
     diff(x.range)*bias.x[val],
     bias.y*diff(y.range),
     paste(layer.name,val,sep=''),
     cex=cex.val
    )
  }
}

#function creates lines colored by direction and width as proportion of magnitude
#use 'all.in' argument if you want to plot connection lines for only a single input node
layer.lines<-function(mod.in,h.layer,layer1=1,layer2=2,out.layer=F,nid,rel.rsc,all.in,pos.col,
                      neg.col,...){

  x0<-rep(layer.x[layer1]*diff(x.range)+line.stag*diff(x.range),struct[layer1])
  x1<-rep(layer.x[layer2]*diff(x.range)-line.stag*diff(x.range),struct[layer1])

  if(out.layer==T){

   y0<-get.ys(struct[layer1])
   y1<-rep(get.ys(struct[layer2])[h.layer],struct[layer1])
   src.str<-paste('out',h.layer)

   wts<-nnet.vals(mod.in,nid=F,rel.rsc)
   wts<-wts[grep(src.str,names(wts))][[1]][-1]
   wts.rs<-nnet.vals(mod.in,nid=T,rel.rsc)
   wts.rs<-wts.rs[grep(src.str,names(wts.rs))][[1]][-1]

   cols<-rep(pos.col,struct[layer1])
   cols[wts<0]<-neg.col

   if(nid) segments(x0,y0,x1,y1,col=cols,lwd=wts.rs)
   else segments(x0,y0,x1,y1)

  }
```

```
  else{

    if(is.logical(all.in)) all.in<-h.layer
    else all.in<-which(x.names==all.in)

    y0<-rep(get.ys(struct[layer1])[all.in],struct[2])
    y1<-get.ys(struct[layer2])
    src.str<-paste('hidden',layer1)

    wts<-nnet.vals(mod.in,nid=F,rel.rsc)
    wts<-unlist(lapply(wts[grep(src.str,names(wts))],function(x) x[all.in+1]))
    wts.rs<-nnet.vals(mod.in,nid=T,rel.rsc)
    wts.rs<-unlist(lapply(wts.rs[grep(src.str,names(wts.rs))],function(x) x[all.in+1]))

    cols<-rep(pos.col,struct[layer2])
    cols[wts<0]<-neg.col

    if(nid) segments(x0,y0,x1,y1,col=cols,lwd=wts.rs)
    else segments(x0,y0,x1,y1)

  }

}

bias.lines<-function(bias.x,mod.in,nid,rel.rsc,all.out,pos.col,neg.col,...){

  if(is.logical(all.out)) all.out<-1:struct[length(struct)]
  else all.out<-which(y.names==all.out)

  for(val in 1:length(bias.x)){

    wts<-nnet.vals(mod.in,nid=F,rel.rsc)
    wts.rs<-nnet.vals(mod.in,nid=T,rel.rsc)

    if(val != length(bias.x)){
      wts<-wts[grep('out',names(wts),invert=T)]
      wts.rs<-wts.rs[grep('out',names(wts.rs),invert=T)]
      sel.val<-grep(val,substr(names(wts.rs),8,8))
      wts<-wts[sel.val]
      wts.rs<-wts.rs[sel.val]
    }
```

```
    else{
      wts<-wts[grep('out',names(wts))]
      wts.rs<-wts.rs[grep('out',names(wts.rs))]
    }

    cols<-rep(pos.col,length(wts))
    cols[unlist(lapply(wts,function(x) x[1]))<0]<-neg.col
    wts.rs<-unlist(lapply(wts.rs,function(x) x[1]))

    if(nid==F){
      wts.rs<-rep(1,struct[val+1])
      cols<-rep('black',struct[val+1])
    }

    if(val != length(bias.x)){
      segments(
        rep(diff(x.range)*bias.x[val]+diff(x.range)*line.stag,struct[val+1]),
        rep(bias.y*diff(y.range),struct[val+1]),
        rep(diff(x.range)*layer.x[val+1]-diff(x.range)*line.stag,struct[val+1]),
        get.ys(struct[val+1]),
        lwd=wts.rs,
        col=cols
      )
    }

    else{
      segments(
        rep(diff(x.range)*bias.x[val]+diff(x.range)*line.stag,struct[val+1]),
        rep(bias.y*diff(y.range),struct[val+1]),
        rep(diff(x.range)*layer.x[val+1]-diff(x.range)*line.stag,struct[val+1]),
        get.ys(struct[val+1])[all.out],
        lwd=wts.rs[all.out],
        col=cols[all.out]
      )
    }

  }
}

#use functions to plot connections between layers
```

```
#bias lines
if(bias) bias.lines(bias.x,mod.in,nid=nid,rel.rsc=rel.rsc,all.out=all.out,pos.col=alpha(pos.col,alpha.val),
                    neg.col=alpha(neg.col,alpha.val))

#layer lines, makes use of arguments to plot all or for individual layers
#starts with input-hidden
#uses 'all.in' argument to plot connection lines for all input nodes or a single node
if(is.logical(all.in)){
 mapply(
  function(x) layer.lines(mod.in,x,layer1=1,layer2=2,nid=nid,rel.rsc=rel.rsc,
                          all.in=all.in,pos.col=alpha(pos.col,alpha.val),neg.col=alpha(neg.col,alpha.val)),
  1:struct[1]
 )
}
else{
 node.in<-which(x.names==all.in)
 layer.lines(mod.in,node.in,layer1=1,layer2=2,nid=nid,rel.rsc=rel.rsc,all.in=all.in,
             pos.col=alpha(pos.col,alpha.val),neg.col=alpha(neg.col,alpha.val))
}
#connections between hidden layers
lays<-split(c(1,rep(2:(length(struct)-1),each=2),length(struct)),
            f=rep(1:(length(struct)-1),each=2))
lays<-lays[-c(1,(length(struct)-1))]
for(lay in lays){
 for(node in 1:struct[lay[1]]){
  layer.lines(mod.in,node,layer1=lay[1],layer2=lay[2],nid=nid,rel.rsc=rel.rsc,all.in=T,
              pos.col=alpha(pos.col,alpha.val),neg.col=alpha(neg.col,alpha.val))
 }
}
#lines for hidden-output
#uses 'all.out' argument to plot connection lines for all output nodes or a single node
if(is.logical(all.out))
 mapply(
  function(x) layer.lines(mod.in,x,layer1=length(struct)-1,layer2=length(struct),out.layer=T,nid=nid,rel.rsc=rel.rsc,
                          all.in=all.in,pos.col=alpha(pos.col,alpha.val),neg.col=alpha(neg.col,alpha.val)),
  1:struct[length(struct)]
 )
else{
 node.in<-which(y.names==all.out)
 layer.lines(mod.in,node.in,layer1=length(struct)-1,layer2=length(struct),out.layer=T,nid=nid,rel.rsc=rel.rsc,
             pos.col=pos.col,neg.col=neg.col,all.out=all.out)
```

```
  }

  #use functions to plot nodes
  for(i in 1:length(struct)){
    in.col<-bord.col<-circle.col
    layer.name<-'H'
    if(i==1) { layer.name<-'I'; in.col<-bord.col<-circle.col.inp}
    if(i==length(struct)) layer.name<-'O'
    layer.points(struct[i],layer.x[i],layer.name)
  }

  if(bias) bias.points(bias.x,bias.y,'B')

}

plot.nnet(mod3)

#neural net with three hidden layers, 9, 11, and 8 nodes in each
mod<-mlp(rand.vars, resp, size=c(9,11,8),linOut=T)
par(mar=numeric(4),family='serif')
plot.nnet(mod)

##Variable Importance


#An obvious difference between a neural network and a regression model is that the number of weights is excessive
#in the former case. This characteristic is advantageous in that it makes neural networks very flexible for modeling
#non-linear functions with multiple interactions, although interpretation of the effects of specific variables is
#of course challenging.

#Garson 19912 (also Goh 19953) identifies the relative importance of explanatory variables for specific response variables
#in a supervised neural network by deconstructing the model weights. The basic idea is that the relative importance
#(or strength of association) of a specific explanatory variable for a specific response variable can be determined by
#identifying all weighted connections between the nodes of interest. That is, all weights connecting the specific input
#node that pass through the hidden layer to the specific response variable are identified. This is repeated for all other
#explanatory variables until the analyst has a list of all weights that are specific to each input variable.
#The connections are tallied for each input node and scaled relative to all other inputs.
#A single value is obtained for each explanatory variable that describes the relationship with response variable
#in the model (see the appendix in Goh 1995 for a more detailed description).
#The original algorithm presented in Garson 1991 indicated relative importance as the absolute magnitude from zero
#to one such the direction of the response could not be determined. I modified the approach to preserve the sign,
```

```
#as you'll see below.
# install package from GitHub
install.packages('devtools')
library(devtools)
install_github('fawda123/NeuralNetTools')
library(NeuralNetTools)

# run the examples for the garson function
example(garson)

require(clusterGeneration)
require(RSNNS)

#define number of variables and observations
set.seed(2)
num.vars<-8
num.obs<-10000

#define correlation matrix for explanatory variables
#define actual parameter values
cov.mat<-genPositiveDefMat(num.vars,covMethod=c("unifcorrmat"))$Sigma
rand.vars<-mvrnorm(num.obs,rep(0,num.vars),Sigma=cov.mat)
parms<-runif(num.vars,-10,10)
y<-rand.vars %*% matrix(parms) + rnorm(num.obs,sd=20)

#prep data and create neural network
y<-data.frame((y-min(y))/(max(y)-min(y)))
names(y)<-'y'
rand.vars<-data.frame(rand.vars)
mod<-mlp(rand.vars, resp[,1], size=5,linOut=T)

cols <- heat.colors(5)
garson(mod) +
  scale_y_continuous('Rel. Importance', limits = c(-1, 1)) +
  scale_fill_gradientn(colours = cols) +
  scale_colour_gradientn(colours = cols)
```

SA_NN

```
#define number of variables and observations

require(clusterGeneration)
require(nnet)
set.seed(2)
num.vars<-8
num.obs<-10000

#define correlation matrix for explanatory variables
#define actual parameter values
cov.mat<-genPositiveDefMat(num.vars,covMethod=c("unifcorrmat"))$Sigma
rand.vars<-mvrnorm(num.obs,rep(0,num.vars),Sigma=cov.mat)
parms1<-runif(num.vars,-10,10)
y1<-rand.vars %*% matrix(parms1) + rnorm(num.obs,sd=20)
parms2<-runif(num.vars,-10,10)
y2<-rand.vars %*% matrix(parms2) + rnorm(num.obs,sd=20)

#prep data and create neural network
rand.vars<-data.frame(rand.vars)
resp<-apply(cbind(y1,y2),2, function(y) (y-min(y))/(max(y)-min(y)))
resp<-data.frame(resp)
names(resp)<-c('Y1','Y2')
mod1<-nnet(rand.vars,resp,size=8,linout=T)

lek.fun<-function(mod.in,var.sens=NULL,resp.name=NULL,exp.in=NULL,steps=100,split.vals=seq(0,1,by=0.2),val.out=F){

 require(ggplot2)
 require(reshape)

 ##
 #sort out exp and resp names based on object type of call to mod.in
 #get matrix for exp vars

 #for nnet
 if('nnet' %in% class(mod.in) | !'mlp' %in% class(mod.in)){
  if(is.null(mod.in$call$formula)){
    if(is.null(resp.name)) resp.name<-colnames(eval(mod.in$call$y))
    if(is.null(var.sens)) var.sens<-colnames(eval(mod.in$call$x))
    mat.in<-eval(mod.in$call$x)
```

```
  }
  else{
    forms<-eval(mod.in$call$formula)
    dat.names<-model.frame(forms,data=eval(mod.in$call$data))
    if(is.null(resp.name)) resp.name<-as.character(forms)[2]
    if(is.null(var.sens))
      var.sens<-names(dat.names)[!names(dat.names) %in% as.character(forms)[2]]
    mat.in<-dat.names[,!names(dat.names) %in% as.character(forms)[2]]
  }
}

#for rsnns
if('mlp' %in% class(mod.in)){
  if(is.null(exp.in)) stop('Must include matrix or data frame of input variables')

  if(is.null(resp.name)) resp.name<-paste0('Y',seq(1,mod.in$nOutputs))
  mat.in<-data.frame(exp.in)
  names(mat.in)<-paste0('X',seq(1,mod.in$nInputs))
  if(is.null(var.sens)) var.sens<-names(mat.in)

}

##
#gets predicted output for nnet based on matrix of explanatory variables
#selected explanatory variable is sequenced across range of values
#all other explanatory variables are held constant at value specified by 'fun.in'
pred.sens<-function(mat.in,mod.in,var.sel,step.val,fun.in,resp.name){

  mat.out<-matrix(nrow=step.val,ncol=ncol(mat.in),dimnames=list(c(1:step.val),colnames(mat.in)))

  mat.cons<-mat.in[,!names(mat.in) %in% var.sel]
  mat.cons<-apply(mat.cons,2,fun.in)
  mat.out[,!names(mat.in) %in% var.sel]<-t(sapply(1:step.val,function(x) mat.cons))

  mat.out[,var.sel]<-seq(min(mat.in[,var.sel]),max(mat.in[,var.sel]),length=step.val)

  out<-data.frame(predict(mod.in,new=as.data.frame(mat.out)))
  names(out)<-paste0('Y',seq(1,ncol(out)))
  out<-out[,resp.name,drop=F]
  x.vars<-mat.out[,var.sel]
  data.frame(out,x.vars)
```

```
}

#use 'pred.fun' to get pred vals of response across range of vals for an exp vars
#loops over all explanatory variables of interest and all split values
lek.vals<-sapply(
  var.sens,
  function(vars){
    sapply(
      split.vals,
      function(splits){
        pred.sens(
              mat.in,
              mod.in,
              vars,
              steps,
              function(val) quantile(val,probs=splits),
              resp.name
        )
      },
      simplify=F
    )
  },
  simplify=F
)

#melt lek.val list for use with ggplot
lek.vals<-melt.list(lek.vals,id.vars='x.vars')
lek.vals$L2<-factor(lek.vals$L2,labels=split.vals)
names(lek.vals)<-c('Explanatory','resp.name','Response','Splits','exp.name')

#return only values if val.out = T
if(val.out) return(lek.vals)

#ggplot object
p<-ggplot(lek.vals,aes(x=Explanatory,y=Response,group=Splits)) +
  geom_line(aes(colour=Splits,linetype=Splits,size=Splits)) +
  facet_grid(resp.name~exp.name) +
  scale_linetype_manual(values=rep('solid',length(split.vals))) +
  scale_size_manual(values=rep(1,length(split.vals)))
```

```
  return(p)

}

lek.fun(mod1)

lek.fun(mod1,var.sens=c('X2','X5'),split.vals=seq(0,1,by=0.05))

head(lek.fun(mod1,val.out=T))

mod2<-lm(Y1~.,data=cbind(resp[,'Y1',drop=F],rand.vars))
lek.fun(mod2)
```

Titanic Random Forest with feature extraction

```
# Load packages
library('ggplot2') # visualization
library('ggthemes') # visualization
library('scales') # visualization
library('dplyr') # data manipulation
library('mice') # imputation
library('randomForest') # classification algorithm

#Now that our packages are loaded, let’s read in and take a peek at the data.

train <- read.csv('train.csv', stringsAsFactors = F)
test  <- read.csv('test.csv', stringsAsFactors = F)



full  <- bind_rows(train, test) # bind training & test data

# check data
str(full)

#Feature Engineering
full$Title <- gsub('(.*, )|(\\..*)', '', full$Name)

head(full$Title)
```

```
rare_title <- c('Dona', 'Lady', 'the Countess','Capt', 'Col', 'Don',
                'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer')

# Also reassign mlle, ms, and mme accordingly
full$Title[full$Title == 'Mlle']        <- 'Miss'
full$Title[full$Title == 'Ms']          <- 'Miss'
full$Title[full$Title == 'Mme']         <- 'Mrs'

full$Title[full$Title %in% rare_title]  <- 'Rare Title'

# Show title counts by sex again
table(full$Sex, full$Title)

str(full$Name)

full$Surname <- sapply(full$Name,
                       function(x) strsplit(x, split = '[,.]')[[1]][1])

head(full$Surname)
head(full$Name)

#Do families sink or swim together?

# Create a family size variable including the passenger themselves
full$Fsize <- full$SibSp + full$Parch + 1

# Create a family variable
full$Family <- paste(full$Surname, full$Fsize, sep='_')

ggplot(full[1:891,], aes(x = Fsize, fill = factor(Survived))) +
  geom_bar(stat = "bin", position='dodge') +
  scale_x_continuous(breaks=c(1:11)) +
  labs(x = 'Family Size')

# Discretize family size
full$FsizeD[full$Fsize == 1] <- 'singleton'
full$FsizeD[full$Fsize < 5 & full$Fsize > 1] <- 'small'
```

```
full$FsizeD[full$Fsize > 4] <- 'large'

# Show family size by survival using a mosaic plot
mosaicplot(table(full$FsizeD, full$Survived), main='Family Size by Survival', shade=TRUE)

table(full$Fsize)

#Treat a few more variables …
full$Cabin

# Create a Deck variable. Get passenger deck A - F:
full$Deck<-factor(sapply(full$Cabin, function(x) strsplit(x, NULL)[[1]][1]))

#Missingness -Sensible value imputation

# Passengers 62 and 830 are missing Embarkment
full[c(62, 830), 'Embarked']

cat(paste('We will infer their values for **embarkment** based on present data that we can imagine may be relevant:
        **passenger class** and **fare**. We see that they paid<b> $', full[c(62, 830), 'Fare'][[1]][1], '</b>and<b> $', full[c(62,
830), 'Fare'][[1]][2],
        '</b>respectively and their classes are<b>', full[c(62, 830), 'Pclass'][[1]][1],
        '</b>and<b>', full[c(62, 830), 'Pclass'][[1]][2], '</b>. So from where did they embark?'))

# Get rid of our missing passenger IDs
embark_fare <- full %>%
  filter(PassengerId != 62 & PassengerId != 830)
# Use ggplot2 to visualize embarkment, passenger class, & median fare
ggplot(embark_fare, aes(x = Embarked, y = Fare, fill = factor(Pclass))) +
  geom_boxplot() +
  geom_hline(aes(yintercept=80),
             colour='red', linetype='dashed', lwd=2) +
  scale_y_continuous(labels=dollar_format())

#Voilà! The median fare for a first class passenger departing from Charbourg (‘C’) coincides nicely with the $80 paid by our embarkment-deficient passengers. I think we can safely replace the NA values with ‘C’.

full$Embarked[c(62, 830)] <- 'C'
```

```
full[1044, ]

ggplot(full[full$Pclass == '3' & full$Embarked == 'S', ],
       aes(x = Fare)) +
  geom_density(fill = '#99d6ff', alpha=0.4) +
  geom_vline(aes(xintercept=median(Fare, na.rm=T)),
             colour='red', linetype='dashed', lwd=1) +
  scale_x_continuous(labels=dollar_format())

# Replace missing fare value with median fare for class/embarkment
full$Fare[1044] <- median(full[full$Pclass == '3' & full$Embarked == 'S', ]$Fare, na.rm = TRUE)

#Predictive imputation
# Show number of missing Age values
sum(is.na(full$Age))

#We could definitely use rpart (recursive partitioning for regression) to predict missing ages, but I'm going to use the  mice package for this
#task just for something different. You can read more about multiple imputation using chained equations in r here (PDF). Since we haven't done it yet,
#I'll first factorize the factor variables and then perform mice imputation.



# Make variables factors into factors
factor_vars <- c('PassengerId','Pclass','Sex','Embarked',
                 'Title','Surname','Family','FsizeD')
full[factor_vars] <- lapply(full[factor_vars], function(x) as.factor(x))

# Set a random seed
set.seed(129)

# Perform mice imputation, excluding certain less-than-useful variables:
mice_mod <- mice(full[, !names(full) %in% c('PassengerId','Name','Ticket','Cabin','Family','Surname','Survived')], method='rf')



mice_output <- complete(mice_mod)
```

```
# Plot age distributions
par(mfrow=c(1,2))
hist(full$Age, freq=F, main='Age: Original Data',
    col='darkgreen', ylim=c(0,0.04))
hist(mice_output$Age, freq=F, main='Age: MICE Output',
    col='lightgreen', ylim=c(0,0.04))

# Replace Age variable from the mice model.
full$Age <- mice_output$Age

# Show new number of missing Age values
sum(is.na(full$Age1))

#Feature Engineering: Round 2
#Now that we know everyone's age, we can create a couple of new age-dependent variables: Child and Mother.
#A child will simply be someone under 18 years of age and a mother is a passenger who is 1) female, 2) is over 18, 3) has more than 0 children (no kidding!), and 4) does not have the title 'Miss'.

# First we'll look at the relationship between age & survival
ggplot(full[1:891,], aes(Age, fill = factor(Survived))) +
 geom_histogram() +
 # I include Sex since we know (a priori) it's a significant predictor
 facet_grid(.~Sex) +
 theme_few()

# Create the column child, and indicate whether child or adult
full$Child[full$Age < 18] <- 'Child'
full$Child[full$Age >= 18] <- 'Adult'

# Show counts
table(full$Child, full$Survived)

#Looks like being a child doesn't hurt, but it's not going to necessarily save you either!
#We will finish off our feature engineering by creating the Mother variable. Maybe we can hope that mothers are more likely to have survived on the Titanic.
```

```
# Adding Mother variable
full$Mother <- 'Not Mother'
full$Mother[full$Sex == 'female' & full$Parch > 0 & full$Age > 18 & full$Title != 'Miss'] <- 'Mother'

# Show counts
table(full$Mother, full$Survived)

# Finish by factorizing our two new factor variables
full$Child  <- factor(full$Child)
full$Mother <- factor(full$Mother)

md.pattern(full)
str(full)

#Prediction - Split into training & test sets
# Split the data back into a train set and a test set
train <- full[1:891,]
test <- full[892:1309,]

rf_model <- randomForest(factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch +
                           Fare + Embarked + Title +
                           FsizeD + Child + Mother,
                         data = train)

# Show model error
plot(rf_model, ylim=c(0,0.36))
legend('topright', colnames(rf_model$err.rate), col=1:3, fill=1:3)
```

#The black line shows the overall error rate which falls below 20%.

#The red and green lines show the error rate for ‘died’ and ‘survived’ respectively.

#We can see that right now we’re much more successful predicting death than we are survival. What does that say about me, I wonder?

```
#Variable importance
```

```
# Get importance
importance    <- importance(rf_model)
varImportance <- data.frame(Variables = row.names(importance),
                            Importance = round(importance[ ,'MeanDecreaseGini'],2))

# Create a rank variable based on importance
rankImportance <- varImportance %>%
  mutate(Rank = paste0('#',dense_rank(desc(Importance))))

# Use ggplot2 to visualize the relative importance of variables
ggplot(rankImportance, aes(x = reorder(Variables, Importance),
                           y = Importance, fill = Importance)) +
  geom_bar(stat='identity') +
  geom_text(aes(x = Variables, y = 0.5, label = Rank),
            hjust=0, vjust=0.55, size = 4, colour = 'red') +
  labs(x = 'Variables') +
  coord_flip()

#Whoa, glad we made our title variable! It has the highest relative importance out of all of our predictor variables.
#I think I'm most surprised to see that passenger class fell to #5, but maybe that's just bias coming from watching the movie Titanic too many times as a kid.

#Prediction!
#We're ready for the final step — making our prediction! When we finish here, we could iterate through the preceding steps making tweaks as we go or fit the data using
#different models or use different combinations of variables to achieve better predictions. But this is a good starting (and stopping) point for me now.

# Predict using the test set
prediction <- predict(rf_model, test)

# Save the solution to a dataframe with two columns: PassengerId and Survived (prediction)
solution <- data.frame(PassengerID = test$PassengerId, Survived = prediction)

# Write the solution to file
write.csv(solution, file = 'rf_mod_Solution.csv', row.names = F)
```

Data Cleaning

```
library(stringr)
total<- total[,-1]



total1 <- total[!duplicated(total),]
write.csv(total2, file = "total2.csv")
total1 <- total1[order(total1$Team, total1$QTR, -total1$Time),]

total1$PLTYPE<- ifelse(total1$PLTYPE == '2pt pass', 'pass', ifelse(total1$PLTYPE == '2pt rush', 'rushed', total1$PLTYPE))
total1$PLTYPE<- ifelse(total1$PLTYPE == 'post-play fumble', 'fumble', ifelse(total1$PLTYPE == 'pre-play fumble', 'fumble', ifelse(total1$PLTYPE == 'pre-punt fumble', 'fumble', ifelse(total1$PLTYPE == 'rush', 'rushed', total1$PLTYPE))))
total1$PLTYPE<- ifelse(total1$PLTYPE == 'penalties offsetting', 'penalty', ifelse(total1$PLTYPE == 'penalty declined', 'penalty', ifelse(total1$PLTYPE == 'penalty superseded', 'penalty', ifelse(total1$PLTYPE == 'post-abort pass', 'pass', total1$PLTYPE))))
head(total1)
table(total1$PLTYPE)
total1 <- total1[order(total1$Year, total1$WEEK, total1$HOME, total1$QTR, -total1$Time),]
total1<- total1[!(total1$PLTYPE=='penalty'| total1$PLTYPE=='fumble'),]



table(total1$PLTYPE)

total1<- total1[,-39]
head(total1)
table(total1$PP)
total1$PP[is.na(total1$PP)] <- 0

for (i in 2:length(total1$R)) {
  total1$Rush[i] = ifelse( total1$Team[i] == total1$Team[i-1] & total1$PLTYPE[i-1] == 'rushed',1, 0)
}



for (i in 2:length(total1$R)) {
  total1$Pass[i] = ifelse( total1$Team[i] == total1$Team[i-1] & total1$PLTYPE[i-1] == 'pass',1, 0)
}

total1$YARDS[is.na(total1$YARDS)] <- 4
```

```
for (i in 2:length(total1$R)) {
  total1$Yardage[i] = ifelse( total1$Team[i] == total1$Team[i-1], total1$YARDS[i-1], 4)
}


for (i in 2:length(total1$R)) {
  total1$Autocorrelation[i] = ifelse( total1$Team[i] == total1$Team[i-1] & total1$PP[i-1] == 1, 1, 0)
}

total1$Plays<- 0
for (i in 2:length(total1$R)) {
  total1$Plays[i] = ifelse( total1$Team[i] == total1$Team[i-1], total1$Plays[i-1] + 1, 0)
}

total2<- total1[!(total1$PLTYPE=='rushed'),]


head(total1$Plays)
sum(is.na(total1$Plays))


total1$Pass[is.na(total1$Pass)] <- 0
total1$Rush[is.na(total1$Rush)] <- 0
total1$Yardage[is.na(total1$Yardage)] <- 4
total1$PP[is.na(total1$PP)] <- 0
total1$Autocorrelation[is.na(total1$Autocorrelation)] <- 0
head(total1)


A<- data.frame(cbind(PR$R, PR$Times))
colnames(A)<- c('R', 'Times')

total3 <- merge(total2,A,by="R")
head(total3)
```

```
total3$Times_1<- ifelse(total3$QTR == 1, total3$Times + 45,ifelse(total3$QTR == 2,total3$Times + 30, ifelse( total3$QTR == 3, total3$Times + 15, total3$Times )))

total3$Scores<- ifelse(total3$Score > 0, log(total3$Score), ifelse(total3$Score < 0, -log(abs(total3$Score)), 0))
total3$GAPS<- ifelse(total3$GAP > 0, log(total3$GAP), ifelse(total3$GAP < 0, -log(abs(total3$GAP)), 0))

total3$WP<- (1-pnorm((-3*total3$GAPS + 0.5) * 2.5*log(60/total3$Times_1),-1.15*total3$Scores * (total3$Times_1 / 60), 13.45/(sqrt(60/total3$Times_1)), lower.tail = TRUE)) + .5 * (pnorm((-3*total3$GAPS + 0.5) * 2.5*log(60/total3$Times_1),
-1.15*total3$Scores * (total3$Times_1 / 60), 13.45/(sqrt(60/total3$Times_1)), lower.tail = TRUE)
- pnorm((-3*total3$GAPS + 0.5) * 2.5*log(60/total3$Times_1),-1.15*total3$Scores * (total3$Times_1 / 60), 13.45/(sqrt(60/total3$Times_1)), lower.tail = TRUE))

sum(is.na(total3$WP))

total3$OFFENSE<- as.character(total3$OFFENSE)
total3$DEFENSE<- as.character(total3$DEFENSE)
total3$HOME<- as.character(total3$HOME)
total3$OFFENSE<- ifelse(total3$OFFENSE == 'STL', 'LARM', ifelse(total3$OFFENSE == 'SD', 'LAC', total3$OFFENSE))
total3$DEFENSE<- ifelse(total3$DEFENSE == 'STL', 'LARM', ifelse(total3$DEFENSE == 'SD', 'LAC', total3$DEFENSE))
total3$HOME<- ifelse(total3$HOME == 'STL', 'LARM', ifelse(total3$HOME == 'SD', 'LAC', total3$HOME))
table(total3$OFFENSE)

sum(is.na(total3$PLAYER))
total3$PLAYER<- as.character(total3$PLAYER)
sort(table(total3$PLAYER),decreasing=F)
total3$PLAYER[total3$PLAYER=="5-D.Carr"]<-"5-Da.Carr"
total3$PLAYER[total3$PLAYER=="8-D.Carr"]<-"8-Da.Carr"
sort(table(total3$PLAYER))

x<- str_split_fixed(total3$PLAYER, "-", 2)
total3$QBR<- x[,2]

sort(table(total3$QBR), decreasing = F)

total3$QBR<- as.character(total3$QBR)
total3$QBR[total3$QBR=="Ma.Moore"]<-"M.Moore"
total3$QBR[total3$QBR=="Matt.Moore"]<-"M.Moore"
```

```
total3$QBR[total3$QBR=="Sh.Hill"]<-"S.Hill"
total3$QBR[total3$QBR=="Jo.Freeman"]<-"J.Freeman"
total3<- total3[,-1]
sort(table(total3$QBR), decreasing = F)
table(total3$RECEPT)
total3<- total3[!(total3$RECEPT=='aborted snap'),]

sort(table(total3$QBR), decreasing = F)
total3$QBR<- factor(total3$QBR)

levels(total3$QBR)[rank(table(total3$QBR)) < 183] <- "Other"
total3$QBR<- as.character(total3$QBR)

sort(table(total3$QBR), decreasing = F)

total3$QBR[total3$QBR=="R.Brown"]<-"Other"
total3$QBR[total3$QBR=="J.Cribbs"]<-"Other"

total3$QBR<- as.character(total3$QBR)
sort(table(total3$QBR), decreasing = F)

total3$QBR<- factor(total3$QBR)

levels(total3$QBR)[rank(table(total3$QBR)) < 42] <- "bench"
total3$QBR<- as.character(total3$QBR)
total3$QBR[total3$QBR=="J.Webb"]<-"bench"

sort(table(total3$QBR), decreasing = F)

total3$QBR<- factor(total3$QBR)

levels(total3$QBR)[rank(table(total3$QBR)) < 29] <- "backup"
total3$QBR<- as.character(total3$QBR)
sort(table(total3$QBR), decreasing = F)

table(total3$RECEPT)
```

```
total3$QBR<- factor(total3$QBR)
total3 <- total3[order(total3$Year,total3$WEEK,total3$OFFENSE, total3$QTR, total3$R),]
total3$Year<- as.numeric(total3$Year)
total3$WEEK<- as.numeric(total3$WEEK)
total3$OFFENSE<- as.character(total3$OFFENSE)
total3$DEFENSE<- as.character(total3$DEFENSE)
#for (i in 2:length(PP$R)) {
#PP$Autocorrelation[i]<-ifelse(PP$Year[i]==PP$Year[i-1] & PP$WEEK[i]==PP$WEEK[i-1] &
#PP$OFFENSE[i]==PP$OFFENSE[i-1] & PP$PP[i-1]== 1, 1, 0)
#}
#PP$Autocorrelation[is.na(PP$Autocorrelation)] <- 0



head(total3)
total3$Distance<- ifelse(total3$TOGO <4, 'Short', ifelse(total3$TOGO >3 & total3$TOGO < 7, 'Medium', ifelse(total3$TOGO >
6 & total3$TOGO < 11, 'Long', ifelse(
  total3$TOGO > 10 & total3$TOGO < 16, 'Longer', ifelse(total3$TOGO > 15 & total3$TOGO < 21, 'Very Long',
ifelse(total3$TOGO > 20 & total3$TOGO <26, 'Too Long','Forever')
  )))))


total3$DOWN<- as.factor(total3$DOWN)


table(total3$OFFENSE)
total3$H<- ifelse(total3$Year == 2007 & (total3$OFFENSE == 'MIA' | total3$OFFENSE == 'NYG'), 0,
           ifelse(total3$Year == 2008 & (total3$OFFENSE == 'LAC' | total3$OFFENSE == 'NO'),0,
                ifelse(total3$Year == 2009 & (total3$OFFENSE == 'NE'|  total3$OFFENSE == 'TB'),0,
                     ifelse(total3$Year == 2010 & (total3$OFFENSE == 'DEN'|  total3$OFFENSE == 'SF'),0,
                          ifelse(total3$Year == 2011 & (total3$OFFENSE == 'CHI'|  total3$OFFENSE == 'TB'),0,
                               ifelse(total3$Year == 2012 & (total3$OFFENSE == 'NE'|  total3$OFFENSE == 'LARM'),0,
                                    ifelse(total3$Year == 2013 & (total3$OFFENSE == 'PIT' | total3$OFFENSE == 'MIN'),0,
                                         ifelse(total3$Year == 2013 & (total3$OFFENSE == 'SF' | total3$OFFENSE == 'JAC'),0,
                                              ifelse(total3$Year == 2014 & (total3$OFFENSE == 'MIA' | total3$OFFENSE ==
'OAK'),0,
                                                   ifelse(total3$Year == 2014 & (total3$OFFENSE == 'DET' | total3$OFFENSE ==
'ATL'),0,
                                                        ifelse(total3$Year == 2014 & (total3$OFFENSE == 'DAL' | total3$OFFENSE
== 'JAC'),0,
```

```
                                                    ifelse(total3$Year == 2015 & (total3$OFFENSE == 'NYJ' |
total3$OFFENSE == 'MIA'),0,
                                                        ifelse(total3$Year == 2015 & (total3$OFFENSE == 'BUF' |
total3$OFFENSE == 'JAC'),0,
                                                            ifelse(total3$Year == 2015 & (total3$OFFENSE == 'DET' |
total3$OFFENSE == 'KC'),0,
                                                                ifelse(total3$OFFENSE == total3$HOME,1,0)))))))))))))))
sort(table(total3$QBR), decreasing = FALSE)

total3$Z<- runif(203177,0,10000)
total3 <- total3[order(total3$Z),]
total3$PP<- as.factor(as.numeric(total3$PP))
total3$T<- ifelse(total3$Year==2006,1,ifelse(total3$Year==2007,2,
                                  ifelse(total3$Year==2008,3,
                                      ifelse(total3$Year==2009,4,
                                          ifelse(total3$Year==2010,5,
                                              ifelse(total3$Year==2011,6,
                                                  ifelse(total3$Year==2012,7,
                                                      ifelse(total3$Year==2013,8,
                                                          ifelse(total3$Year==2014,9,10)))))))))



total3$`XTRA NOTE`<- as.character(total3$`XTRA NOTE`)
head(total3)
table(total3$`XTRA NOTE`)



total3$Gadget<- ifelse(total3$`XTRA NOTE` == '(6-P.White QB)'| total3$`XTRA NOTE` =='(7-M.Vick QB)'|
total3$`XTRA NOTE` =='(7-M.Vick QB)' | total3$`XTRA NOTE` == '(Field Goal formation)' |
   total3$`XTRA NOTE` =='(Punt formation)' | total3$`XTRA NOTE` =='(Shotgun, 15-T.Tebow QB)'| total3$`XTRA NOTE`==
'(Shotgun, 7-M.Vick QB)'|total3$`XTRA NOTE`=='(Wildcat)' |total3$`XTRA NOTE`=='(Wildcat, 7-M.Vick QB)' |
     total3$`XTRA NOTE`=='(Wildcat)/Flea Flicker', 1, 0)



total3$Flea_Flicker<- ifelse(total3$`XTRA NOTE` == 'Flea flicker'| total3$`XTRA NOTE` =='Flea Flicker'|
                    total3$`XTRA NOTE` =='(No Huddle)/Flea Flicker' | total3$`XTRA NOTE` == '(Shotgun, Flea Flicker)' |
                    total3$`XTRA NOTE` =='(Shotgun)/Flea Flicker' | total3$`XTRA NOTE` =='(Wildcat)/Flea Flicker', 1, 0)
```

```
total3$Shotgun<- ifelse(total3$`XTRA NOTE` == ' (No Huddle, Shotgun)'| total3$`XTRA NOTE` =='(Shotgun)'|
                        total3$`XTRA NOTE` =='(Shotgun, 15-T.Tebow QB)' | total3$`XTRA NOTE` == '(Shotgun, Flea Flicker)' |
                        total3$`XTRA NOTE` =='(Shotgun)/Flea Flicker' | total3$`XTRA NOTE` =='(Shotgun, 6-P.White QB)' |
total3$`XTRA NOTE` =='(Shotgun, 7-M.Vick QB)', 1, 0)

total3$NoHuddle<- ifelse(total3$`XTRA NOTE` == ' (No Huddle, Shotgun)'| total3$`XTRA NOTE` =='(No Huddle)'|
                         total3$`XTRA NOTE` =='(No Huddle)/Flea Flicker', 1, 0)
head(total3$Shotgun)

total3$Other<- ifelse(total3$QBR == 'Other',1, 0)
table(total3$Other)
total3<- total3[!total3$Other == 1, ]
table(total3$Other)
total3<- total3[-201461,-56]
sort(table(total3$QBR))

total3[201460,]

summary(total3$WP)



total3$Gadget[is.na(total3$Gadget)]<- 0
total3$Shotgun[is.na(total3$Shotgun)]<- 0
total3$NoHuddle[is.na(total3$NoHuddle)]<- 0
total3$Flea_Flicker[is.na(total3$Flea_Flicker)]<- 0

mean(total3$RB, na.rm = TRUE)
mean(total3$WR, na.rm = TRUE)
mean(total3$TE, na.rm = TRUE)

total3$RB[is.na(total3$RB)]<- 2
total3$WR[is.na(total3$WR)]<- 2
total3$TE[is.na(total3$TE)]<- 1
```

```
colnames(total3)[colSums(is.na(total3)) > 0]

sum(is.na(total3$WP))

head(total3$Shotgun)
sort(table(total3$QBR))

table(total3$Shotgun)
total3$WP<- as.numeric(total3$WP)
total3$Times_1<- as.numeric(as.character(total3$Times_1))
sort(table(total3$QBR))
train<- data.frame(cbind(as.numeric(total3$PP),total3$T, total3$Gadget, total3$Flea_Flicker, total3$Shotgun, total3$NoHuddle, total3$WEEK, total3$QTR,
                         total3$ZONE, total3$DOWN, total3$TOGO, total3$RB, total3$WR, total3$TE, total3$Autocorrelation, total3$Offense, total3$Defense,
                         total3$WP, total3$Cold, total3$Hot, total3$QBR, total3$Rush, total3$Pass, total3$Yardage, total3$Distance, total3$H, total3$Year, total3$Times_1))



head(train)

colnames(train)<- c('PP','T','Gadget', 'Flea_Flicker', 'Shotgun', 'NoHuddle', 'WEEK', 'QTR', 'ZONE','DOWN',
                    'TOGO','RB','WR','TE','Autocorrelation', 'Offense', 'Defense', 'WP', 'Cold', 'Hot',
                    'QBR','Rush','Pass','Yardage', 'Distance', 'H', 'Year', 'Times')

train$Times<- as.numeric(as.character(train$Times))

str(train)

train$WP<- as.numeric(as.character(train$WP))
summary(train$WP)
head(train)

head(total3)

train$PP<- ifelse(train$PP==1,0,1)
train$QTR<- ifelse(train$QTR==5,4,train$QTR)
table(train$QTR)
```

```
train$QTR<- as.factor(as.character(train$QTR))
train$ZONE<- as.factor(train$ZONE)
train$Distance<- as.factor(train$Distance)
train$DOWN<- as.factor(train$DOWN)
train$QBR<- as.factor(train$QBR)
train$Year<- as.factor(as.character(train$Year))

A<- model.matrix(PP ~ ZONE + Distance + DOWN + WEEK +  QBR, train)
B<- model.matrix(PP ~ Year + QTR, train)
train<- data.frame(cbind(train, A))
head(train)
str(train)
train<- train[,-29]
train<- data.frame(cbind(train,))
head(train)
head(train)

train$T<- as.numeric(train$T)
train$WR<- as.numeric(train$WR)
train$Offense<- as.numeric(train$Offense)
train$Defense<- as.numeric(train$Defense)
train$TOGO<- as.numeric(train$TOGO)

str(train)

summary(train$Times)
Z<- train
Z$Yardage<- as.numeric(as.character(Z$Yardage))
Z$RB<- as.numeric(as.character(Z$RB))
Z$TE<- as.numeric(as.character(Z$TE))
str(Z)
Z<- Z[,-1]
Z<- Z[,-7]
Z<- Z[,-7]
Z<- Z[,-7]
Z<- Z[,-17]
Z<- Z[,-20]
```

```
Z<- Z[,-2]

head(Z)

normalize<- function(x) {
  return ((x - min(x))/ (max(x) -min(x)))
}

str(Z)

Z$TOGO<-normalize(Z$TOGO)
Z$RB<-normalize(Z$RB)
Z$WR<-normalize(Z$WR)
Z$TE<-normalize(Z$TE)
Z$Offense<-normalize(Z$Offense)
Z$Defense<-normalize(Z$Defense)
Z$WP<-normalize(Z$WP)
Z$Yardage<-normalize(Z$Yardage)
str(Z)
```

PCA Dimension Reduction

```
library(RCurl) # download https data
library(hydroGOF)
library(xgboost)
library(Metrics)
urlfile <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/gisette/GISETTE/gisette_train.data'
x <- getURL(urlfile, ssl.verifypeer = FALSE)
gisetteRaw <- read.table(textConnection(x), sep = '', header = FALSE, stringsAsFactors = FALSE)

urlfile <- "https://archive.ics.uci.edu/ml/machine-learning-databases/gisette/GISETTE/gisette_train.labels"
x <- getURL(urlfile, ssl.verifypeer = FALSE)
g_labels <- read.table(textConnection(x), sep = '', header = FALSE, stringsAsFactors = FALSE)
g_labels[1:5,]
print(dim(gisetteRaw))
```

#The gisetteRaw data frame has 5001 columns and that's the kind of size we're looking for.

#Before we can start the PCA transformation process, we need to remove the extreme near-zero variance as it won't help us much and risks crashing the script.

#We load the caret package and call nearZeroVar function with saveMetrics parameter set to true. This will return a data frame with the degree of zero variance for each feature:

```
library(caret)
nzv <- nearZeroVar(gisetteRaw, saveMetrics = TRUE)
print(paste('Range:',range(nzv$percentUnique)))
print(head(nzv))

#We remove features with less than 0.1% variance:
print(paste('Column count before cutoff:',ncol(gisetteRaw)))

gisette_nzv <- gisetteRaw[c(rownames(nzv[nzv$percentUnique > 0.1,])) ]
print(paste('Column count after cutoff:',ncol(gisette_nzv)))

#The data is cleaned up and ready to go. Let's see how well it performs without any PCA transformation. We bind the labels (response/outcome variables) to the set:

dfEvaluate <- cbind(as.data.frame(sapply(gisette_nzv, as.numeric)), cluster=g_labels$V1)
EvaluateAUC <- function(dfEvaluate) {
  require(xgboost)
  require(Metrics)
  CVs <- 5
  cvDivider <- floor(nrow(dfEvaluate) / (CVs+1))
  indexCount <- 1
  outcomeName <- c('cluster')
  predictors <- names(dfEvaluate)[!names(dfEvaluate) %in% outcomeName]
  lsErr <- c()
  lsAUC <- c()
  for (cv in seq(1:CVs)) {
    print(paste('cv',cv))
    dataTestIndex <- c((cv * cvDivider):(cv * cvDivider + cvDivider))
    dataTest <- dfEvaluate[dataTestIndex,]
```

```
  dataTrain <- dfEvaluate[-dataTestIndex,]
  bst <- xgboost(data = as.matrix(dataTrain[,predictors]),
                 label = dataTrain[,outcomeName],
                 max.depth=6, eta = 1, verbose=0,
                 nround=5, nthread=4,
                 objective = "reg:linear")

  predictions <- predict(bst, as.matrix(dataTest[,predictors]), outputmargin=TRUE)
  err <- rmse(dataTest[,outcomeName], predictions)
  auc <- auc(dataTest[,outcomeName],predictions)

  lsErr <- c(lsErr, err)
  lsAUC <- c(lsAUC, auc)
  gc()
 }
 print(paste('Mean Error:',mean(lsErr)))
 print(paste('Mean AUC:',mean(lsAUC)))
}
```

#We're going to feed the data into the following cross-validation function using the zxgboost model. This is a fast model and does great with large data sets.

#The repeated cross-validation will run the data 5 times, each time assigning a new chunk of data as training and testing.

#This not only allows us to use all the data as both train and test sets, but also stabilizes our AUC (Area Under the Curve) score.

```
EvaluateAUC(dfEvaluate)
```

#This yields a great AUC score of 0.9659 (remember, AUC of 0.5 is random, and 1.0 is perfect).

#But we don't really care how well the model did; we just want to use that AUC score as a basis of comparison against the transformed PCA variables.

#So, let's use the same data and run it through prcomp.

#This will transform all the related variables that account for most of the variation - meaning that the first component variable will be the most powerful variable

#(Warning: this can be a very slow to process depending on your machine - it took 20 minutes on my MacBook - so do it once and store the resulting data set for later use):

```
pmatrix <- scale(gisette_nzv)
princ <- prcomp(pmatrix)

#Let's start by running the same cross-validation code with just the first PCA component (remember, this holds most of the variation of our data set).
#We need to use our princ result set and call the predict function to get our data.frame:

nComp <- 1
dfComponents <- predict(princ, newdata=pmatrix)[,1:nComp]

dfEvaluate <- cbind(as.data.frame(dfComponents),
                    cluster=g_labels$V1)

EvaluateAUC(dfEvaluate)

#The resulting AUC of 0.719 isn't that good compared to the orginal, non-transformed data set. But we have to remember that this is one variable against almost 5000!! Let's try this again with 10 components:
nComp <- 10
dfComponents <- predict(princ, newdata=pmatrix)[,1:nComp]

dfEvaluate <- cbind(as.data.frame(dfComponents),
                    cluster=g_labels$V1)

EvaluateAUC(dfEvaluate)

#Hmmm, going back down… Let's stop here and stick with the first 10 PCA components.
#So, 10 PCA columns versus 4639 columns - not bad, right?
#Keep in mind that you should be able to get closer to the AUC of the original data set
#by adding more PCA components as prcomp accounts for all variations in the data. On the other hand, by following the steps in this walkthrough,
#you can get a great AUC score with very little effort and an absurdly smaller resulting data set.
```

H20 Deep Learning

```
getOption("repos")
library(readr)
library(parallel)
library(foreach)
library(doParallel)
library(randomForest)
library(verification)
library(ParallelForest)
library(doMC)
registerDoMC()
numCores<- detectCores()
numCores
library(stringr)
library(plyr)
library(h2o)
## Classification and Regression with H2O Deep Learning
#
#* Introduction
#  * Installation and Startup
#  * Decision Boundaries
#* Cover Type Dataset
#  * Exploratory Data Analysis
#  * Deep Learning Model
#  * Hyper-Parameter Search
#  * Checkpointing
#  * Cross-Validation
#  * Model Save & Load
#* Regression and Binary Classification
#* Deep Learning Tips & Tricks
#
### Introduction
#This tutorial shows how a H2O [Deep Learning](http://en.wikipedia.org/wiki/Deep_learning) model can be used to do supervised classification and regression. A great tutorial about Deep Learning is given by Quoc Le [here](http://cs.stanford.edu/~quoc/tutorial1.pdf) and [here](http://cs.stanford.edu/~quoc/tutorial2.pdf). This tutorial covers usage of H2O from R. A python version of this tutorial will be available as well in a separate document. This file is available in plain R, R markdown and regular markdown formats, and the plots are available as PDF files. All documents are available [on Github](https://github.com/h2oai/h2o-tutorials/tree/master/tutorials/deeplearning/).
```

#

#If run from plain R, execute R in the directory of this script. If run from RStudio, be sure to setwd() to the location of this script. h2o.init() starts H2O in R's current working directory. h2o.importFile() looks for files from the perspective of where H2O was started.

#

#More examples and explanations can be found in our [H2O Deep Learning booklet](http://h2o.ai/resources/) and on our [H2O Github Repository](http://github.com/h2oai/h2o-3/). The PDF slide deck can be found [on Github](https://github.com/h2oai/h2o-tutorials/tree/master/tutorials/deeplearning/H2ODeepLearning.pdf).

#

H2O R Package

#

#Load the H2O R package:

#

R installation instructions are at http://h2o.ai/download

library(h2o)

#

Start H2O

#Start up a 1-node H2O server on your local machine, and allow it to use all CPU cores and up to 2GB of memory:

#

localH2O <- h2o.init(nthreads = -1, startH2O = TRUE, max_mem_size='2g', ip = '127.0.0.1', port=54321)

df <- as.h2o(Z)

splits <- h2o.splitFrame(df, c(0.6,0.2), seed=1000)

train <- h2o.assign(splits[[1]], "train.hex") # 60%

valid <- h2o.assign(splits[[2]], "valid.hex") # 20%

test   <- h2o.assign(splits[[3]], "test.hex") # 20%

First Run of H2O Deep Learning

#Let's run our first Deep Learning model on the covtype dataset.

#We want to predict the `Cover_Type` column, a categorical feature with 7 levels, and the Deep Learning model will be tasked to perform (multi-class) classification. It uses the other 12 predictors of the dataset, of which 10 are numerical, and 2 are categorical with a total of 44 levels. We can expect the Deep Learning model to have 56 input neurons (after automatic one-hot encoding).

#

response <- "PP"

predictors <- setdiff(names(df), response)

predictors

#

train[,response] <-as.factor(train[,response])

valid[,response] <-as.factor(valid[,response])

test[,response] <-as.factor(test[,response])

```
m4 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train,
  validation_frame=valid,   ## validation dataset: used for scoring and early stopping
  x=predictors,
  y=response,
  distribution = "multinomial",
    activation="RectifierWithDropout",  ## default
  hidden=c(400),       ## default: 2 hidden layers with 200 neurons each
  l1=1e-5,                     ## add some L1/L2 regularization
  l2=1e-5,
  sparse = TRUE,
  epochs=500,
  score_validation_samples=10000,
  max_w2=1,
  score_duty_cycle=0.025,         ## don't score more than 2.5% of the wall time
  adaptive_rate=F,                ## manually tuned learning rate
  rate=0.01,
  nfolds = 4
)
summary(m4)
#
#Let's compare the training error with the validation and test set errors
#
h2o.performance(m4, train=T)          ## sampled training data (from model building)
h2o.performance(m4, valid=T)          ## sampled validation data (from model building)
h2o.performance(m4, newdata=train)    ## full training data
h2o.performance(m4, newdata=valid)    ## full validation data
h2o.performance(m4, newdata=test)     ## full test data
#
#To confirm that the reported confusion matrix on the validation set (here, the test set) was correct, we make a prediction on the
test set and compare the confusion matrices explicitly:
#
pred <- h2o.predict(m4, test)
pred
test$Accuracy <- pred$predict == test$PP
1-mean(test$Accuracy)
```

H20 Random Forests & GBM

```
x1 = rnorm(10000) # some continuous variables
x2 = rnorm(10000)
x3 = rnorm(10000)
x4 = rnorm(10000)
z = 1 + 2*x1 + 3*x2 + x3 + x2 * x3 + tanh(x1) + cosh(x2)^2 + 5*(sin(x1*x2) /sinh(x4*x2))  + x2 ** (12) + rnorm(1000,0,10)  # linear combination with a bias
pr = 1/(1+exp(-z)) # pass through an inv-logit function
y = rbinom(1000,1,pr) # bernoulli response variable
df<- data.frame(cbind(blah,x1,x2, x3, x4))
#############################################################################################
### Goal: demonstrate usage of H2O's Random Forest and GBM algorithms
### Task: Predicting forest cover type from cartographic variables only
###       The actual forest cover type for a given observation
###       (30 x 30 meter cell) was determined from the US Forest Service (USFS).
### Note: If run from plain R, execute R in the directory of this script. If run from RStudio,
### be sure to setwd() to the location of this script. h2o.init() starts H2O in R's current
### working directory. h2o.importFile() looks for files from the perspective of where H2O was
### started.
## H2O is an R package
library(h2o)
## Create an H2O cloud
h2o.init(
  nthreads=-1,            ## -1: use all available threads
  max_mem_size = "2G")    ## specify the memory size for the H2O cloud
h2o.removeAll() # Clean slate - just in case the cluster was already running
#prosPath <- system.file("extdata", "prostate.csv", package="h2o")
#df <- h2o.uploadFile(prosPath)
#df[,2] <- as.factor(df[,2])
#df[1:50,]
## Load a file from disk
df <- as.h2o(df)

## First, we will create three splits for train/test/valid independent data sets.
## We will train a data set on one set and use the others to test the validity
##  of model by ensuring that it can predict accurately on data the model has not
##  been shown.
```

```
## The second set will be used for validation most of the time. The third set will
##  be withheld until the end, to ensure that our validation accuracy is consistent
##  with data we have never seen during the iterative process.
splits <- h2o.splitFrame(
  df,           ##  splitting the H2O frame we read above
  c(0.6,0.2),   ##  create splits of 60% and 20%;
  ##  H2O will create one more split of 1-(sum of these parameters)
  ##  so we will get 0.6 / 0.2 / 1 - (0.6+0.2) = 0.6/0.2/0.2
  seed=1234)    ##  setting a seed will ensure reproducible results (not R's seed)

train <- h2o.assign(splits[[1]], "train.hex")
## assign the first result the R variable train
## and the H2O name train.hex
valid <- h2o.assign(splits[[2]], "valid.hex")   ## R valid, H2O valid.hex
test <- h2o.assign(splits[[3]], "test.hex")     ## R test, H2O test.hex

## take a look at the first few rows of the data set
train[1:10,]   ## rows 1-5, all columns

## run our first predictive model
rf1 <- h2o.randomForest(         ## h2o.randomForest function
  training_frame = train,        ## the H2O frame for training
  validation_frame = valid,      ## the H2O frame for validation (not required)
  x=2:5,                         ## the predictor columns, by column index
  y=1,                           ## the target index (what we are predicting)
  model_id = "rf_covType_v1",    ## name the model in H2O
  ##   not required, but helps use Flow
  ntrees = 200,                  ## use a maximum of 200 trees to create the
  ##  random forest model. The default is 50.
  ##  I have increased it because I will let
  ##  the early stopping criteria decide when
  ##  the random forest is sufficiently accurate
  stopping_rounds = 10,          ## Stop fitting new trees when the 2-tree
  ##  average is wihin 0.001 (default) of
  ##  the prior two 2-tree averages.
  ##  Can be thought of as a convergence setting
  score_each_iteration = T,      ## Predict against training and validation for
```

```
  ##  each tree. Default will skip several.
  seed = 100)               ## Set the random seed so that this can be
##  reproduced.
################################################################################
summary(rf1)                    ## View information about the model.
## Keys to look for are validation performance
##  and variable importance

rf1@model$validation_metrics     ## A more direct way to access the validation
##  metrics. Performance metrics depend on
##  the type of model being built. With a
##  multinomial classification, we will primarily
##  look at the confusion matrix, and overall
##  accuracy via hit_ratio @ k=1.
h2o.performance(rf1, valid = TRUE)
#h2o.confusionMatrix(rf1, valid = TRUE)
#h2o.confusionMatrix(rf1, test, valid = FALSE)
#h2o.hit_ratio_table(rf1,valid = TRUE)
## Even more directly, the hit_ratio @ k=1

pred <- h2o.predict(rf1, test)
pred
test$Accuracy <- pred$predict == test$y
1-mean(test$Accuracy)
################################################################################

## Now we will try GBM.
## First we will use all default settings, and then make some changes,
##  where the parameters and defaults are described.

gbm1 <- h2o.gbm(
  training_frame = train,        ## the H2O frame for training
  validation_frame = valid,      ## the H2O frame for validation (not required)
  x=2:3,                         ## the predictor columns, by column index
  y=1,                          ## the target index (what we are predicting)
  model_id = "gbm_covType1",     ## name the model in H2O
  seed = 2000000)                ## Set the random seed for reproducability
```

```
################################################################################
summary(gbm1)                  ## View information about the model.

#h2o.hit_ratio_table(gbm1,valid = T)[1,2]
## Overall accuracy.
pred <- h2o.predict(gbm1, test)
h2o.confusionMatrix(gbm1,valid=T)
pred
test$Accuracy <- pred$predict == test$y
1-mean(test$Accuracy)
## This default GBM is much worse than our original random forest.
## The GBM is far from converging, so there are three primary knobs to adjust
##  to get our performance up if we want to keep a similar run time.
## 1: Adding trees will help. The default is 50.
## 2: Increasing the learning rate will also help. The contribution of each
##  tree will be stronger, so the model will move further away from the
##  overall mean.
## 3: Increasing the depth will help. This is the parameter that is the least
##  straightforward. Tuning trees and learning rate both have direct impact
##  that is easy to understand. Changing the depth means you are adjusting
##  the "weakness" of each learner. Adding depth makes each tree fit the data
##  closer.
##
## The first configuration will attack depth the most, since we've seen the
##  random forest focus on a continuous variable (elevation) and 40-class factor
##  (soil type) the most.
##
## Also we will take a look at how to review a model while it is running.

################################################################################
gbm2 <- h2o.gbm(
  training_frame = train,     ##
  validation_frame = valid,   ##
  x=2:5,                      ##
  y=1,                       ##
  ntrees = 20,                ## decrease the trees, mostly to allow for run time
```

```
  ##  (from 50)
  learn_rate = 0.2,          ## increase the learning rate (from 0.1)
  max_depth = 10,            ## increase the depth (from 5)
  stopping_rounds = 2,       ##
  stopping_tolerance = 0.01,  ##
  score_each_iteration = T,   ##
  model_id = "gbm_covType2",  ##
  seed = 2000000)            ##

#### While this is running, we can actually look at the model.
#### To do this we simply need a new connection to H2O.
#### This R console will run the model, so we need either another R console
####   or the web browser (or python, etc.).
#### In the demo, we will use Flow in our web browser
####  http://localhost:54321
#### And the focus will be to look at model performance, since we are using R to
####  control H2O. So we can simply type in:
####  getModel "gbm_covType2"
################################################################################

summary(gbm2)
#h2o.hit_ratio_table(gbm1,valid = T)[1,2]    ## review the first model's accuracy
#h2o.hit_ratio_table(gbm2,valid = T)[1,2]    ## review the new model's accuracy

pred
test$Accuracy <- pred$predict == test$y
1-mean(test$Accuracy)
################################################################################

## This has moved us in the right direction, but still lower accuracy
##  than the random forest.
## And it still has not converged, so we can make it more aggressive.
## We can now add the stochastic nature of random forest into the GBM
##  using some of the new H2O settings. This will help generalize
##  and also provide a quicker runtime, so we can add a few more trees.

gbm3 <- h2o.gbm(
```

```
  training_frame = train,     ##
  validation_frame = valid,   ##
  x=2:5,                      ##
  y=1,                       ##
  ntrees = 30,               ## add a few trees (from 20, though default is 50)
  learn_rate = 0.3,          ## increase the learning rate even further
  max_depth = 10,            ##
  sample_rate = 0.7,         ## use a random 70% of the rows to fit each tree
  col_sample_rate = 0.7,       ## use 70% of the columns to fit each tree
  stopping_rounds = 2,       ##
  stopping_tolerance = 0.01,  ##
  score_each_iteration = T,   ##
  model_id = "gbm_covType3",  ##
  seed = 2000000)            ##
##########################################################################

summary(gbm3)
pred
test$Accuracy <- pred$predict == test$y
1-mean(test$Accuracy)
#h2o.hit_ratio_table(rf1,valid = T)[1,2]     ## review the random forest accuracy
#h2o.hit_ratio_table(gbm1,valid = T)[1,2]    ## review the first model's accuracy
#h2o.hit_ratio_table(gbm2,valid = T)[1,2]    ## review the second model's accuracy
#h2o.hit_ratio_table(gbm3,valid = T)[1,2]    ## review the newest model's accuracy
##########################################################################

## Now the GBM is close to the initial random forest.
## However, we used a default random forest.
## Random forest's primary strength is how well it runs with standard
##  parameters. And while there are only a few parameters to tune, we can
##  experiment with those to see if it will make a difference.
## The main parameters to tune are the tree depth and the mtries, which
##  is the number of predictors to use.
## The default depth of trees is 20. It is common to increase this number,
##  to the point that in some implementations, the depth is unlimited.
##  We will increase ours from 20 to 30.
## Note that the default mtries depends on whether classification or regression
```

```
##  is being run. The default for classification is one-third of the columns.
##  The default for regression is the square root of the number of columns.

rf2 <- h2o.randomForest(        ##
  training_frame = train,       ##
  validation_frame = valid,     ##
  x=2:5,                        ##
  y=1,                          ##
  model_id = "rf_covType2",     ##
  ntrees = 200,                 ##
  max_depth = 30,               ## Increase depth, from 20
  stopping_rounds = 25,         ##
  stopping_tolerance = 1e-2,    ##
  score_each_iteration = T,     ##
  seed=3000000)                 ##
################################################################################
summary(rf2)
pred
test$Accuracy <- pred$predict == test$y
1-mean(test$Accuracy)
#h2o.hit_ratio_table(gbm3,valid = T)[1,2]    ## review the newest GBM accuracy
#h2o.hit_ratio_table(rf1,valid = T)[1,2]     ## original random forest accuracy
#h2o.hit_ratio_table(rf2,valid = T)[1,2]     ## newest random forest accuracy
################################################################################

## So we now have our accuracy up beyond 95%.
## We have witheld an extra test set to ensure that after all the parameter
##  tuning we have done, repeatedly applied to the validation data, that our
##  model produces similar results against the third data set.

## Create predictions using our latest RF model against the test set.
finalRf_predictions<-h2o.predict(
  object = rf2
  ,newdata = test)

## Glance at what that prediction set looks like
## We see a final prediction in the "predict" column,
```

```
##  and then the predicted probabilities per class.
finalRf_predictions

## Compare these predictions to the accuracy we got from our experimentation
h2o.hit_ratio_table(rf2,valid = T)[1,2]             ## validation set accuracy
mean(finalRf_predictions$predict==test$y)  ## test set accuracy

## We have very similar error rates on both sets, so it would not seem
##  that we have overfit the validation set through our experimentation.
##
## This concludes the demo, but what might we try next, if we were to continue?
##
## We could further experiment with deeper trees or a higher percentage of
##  columns used (mtries).
## Also we could experiment with the nbins and nbins_cats settings to control
##  the H2O splitting.
## The general guidance is to lower the number to increase generalization
##  (avoid overfitting), increase to better fit the distribution.
## A good example of adjusting this value is for nbins_cats to be increased
##  to match the number of values in a category. Though usually unnecessary,
##  if a problem has a very important categorical predictor, this can
##  improve performance.
##
## Also, we can tune our GBM more and sur
```

R Parallel Processing

```
library(parallel)
library(foreach)
library(doParallel)
library(doMC)
registerDoMC()
numCores<- detectCores()
numCores
system.time({
  r<- foreach(icount(trials), .combine=cbind) %dopar% {
    ind<- sample(100,100, replace=TRUE)
    result1<- glm(x[ind,2] ~ x[ind,1], family = "binomial")
    (result1)
  }
})
e<- rnorm(5000,0,1)
x1<- rnorm(5000,0,1)
x2<- rnorm(5000,0,1)
y<- 5 + 1.5*x1 + e
n<- 5000
w<- data.frame(cbind(x1,y))
system.time( {
z<- foreach(i=0:1, .combine = 'cbind') %dopar%{
  lm(y~ x1 ,data= w[(i*2500+1):(2500*i+2500),])$
}
})

require(dplyr)
system.time( {
  z<- foreach::foreach(i=1:2, .combine = 'cbind') %dopar%{
    w<- predict(lm(y1 ~ x1,data= dat1[i:100,]), dat2[i:100,])
  }
})
system.time( {
  z<- foreach::foreach(i=1:2, .combine = 'cbind') %dopar%{
    w<- predict(lmer(y1 ~ x2 | x3 ,data= dat1[i:100,]), dat2[i:100,])
  }
})
```

Parallel Processing - LME4

```
system.time( {
  z<- foreach::foreach(i=seq(0,4000,1000), .combine = 'cbind') %dopar%{
   w<- predict(lmer(y1 ~ x2 | x3 ,data= dat1[(1+i):(1000+1)])
  }
})
```

Parallel Processing Kmeans

```
result<- kmeans(x=dat, centers = 4, nstart=100)
print(result)

results <- foreach::foreach( i = 1:2 ) %dopar% {
  lm(y ~ x1 + x2, data=dat[i:100,])
}
```