Grade Regressions and Analysis

Evan Green

3/10/2017

```
create_data_for_model <- function(spec_graph,grade_info = grades_3, hw_num=1){
  members <- V(spec_graph)$name
  df <- grade_info
  rownames(df) <- grade_info$code
  df <- df[members,]
  df$isMale <- df$Sex == "m"
  df$isFemale <- df$Sex == "f"
  df$Eigen <- eigen_centrality(spec_graph)$vector
  df$Betweenness <- betweenness(spec_graph)
  df$Constraint <- constraint(spec_graph)
  df$Constraint[is.nan(df$Constraint)] <- 1.5
  comp <- components(spec_graph)
  df$Component_Size <- comp$csize[comp$membership]
  df$Big_Component <- df$Component_Size > 50
  df$Out_Degree <- degree(spec_graph, mode = "out")
  df$In_Degree <- degree(spec_graph, mode = "in")
  #compute average score of neighbors
  if(TRUE){
    df$Ave_Score_of_Helpers <- 0
    df$Ave_Score_of_Helpees <- 0
    df$Ave_Score_of_Recip <- 0
    df$Recip_Degree <- 0
    df$Helpers <- ""
    df$Helpees <- ""
    df$Recips <- ""
    for(mem in members){
      neigh_in <- neighbors(spec_graph,mem,mode = "in")
      neigh_out <- neighbors(spec_graph,mem,mode = "out")
      neigh_recip <- base::intersect(neigh_in, neigh_out)
      exclude_recip <- F
      if(exclude_recip){
        neigh_in <- setdiff(neigh_in,neigh_recip)
        neigh_out <- setdiff(neigh_out,neigh_recip)
      }

      df$Recip_Degree[df$code == mem]  <- length(neigh_recip)

      neigh_cols <-  c("Ave_Score_of_Helpers","Ave_Score_of_Helpees","Ave_Score_of_Recip")
      identity_cols <- c("Helpers","Helpees","Recips")
      neigh_vec <- list(neigh_in,neigh_out,neigh_recip)
      names(neigh_vec) <- neigh_cols
      for(num in 1:length(neigh_vec)){
        if(length(neigh_vec[[num]]) > 0){
          df[df$code == mem,names(neigh_vec)[num]] <- mean(df[neigh_vec[[num]],
                                                    paste("hw", hw_num,sep="")]) - mean(df[,pa
          df[df$code == mem,identity_cols[num]] <- paste(paste(members[neigh_vec[[num]]],sep=""),collaps
```

```
          }
        }
      }
    }
    #remove the people who dropped the class at some point
    df <- df[!df$did_drop_after4,]

    df$Target <- df[,paste("hw",hw_num,sep = "")]
    df <- df[,!(colnames(df) %in% c("Sex","did_drop_after4",
                                    "did_drop_immediately",
                                    paste("hw",hw_num,sep = "")))]
    return(df)
}

graph_data_sets <- list()
for(i in 1:k_num_psets){
    graph_data_sets[[i]] <- create_data_for_model(graphs[[i]], hw_num = i)
}

mse <- function(y , model, new_data = NULL){
    if(is.null(new_data)){
        return(mean((y - predict(model)) ** 2))
    }else{
        return(mean((y - predict(model,new_data)) ** 2))
    }
}
```

```
full_data_set <- grades_3
stats_we_want <- c("Ave_Score_of_Helpers","Constraint","Eigen","Betweenness",
                   "In_Degree","Out_Degree","Ave_Score_of_Helpees","Recip_Degree",
                   "Ave_Score_of_Recip","Helpers","Helpees","Recips","Big_Component",
                   "Component_Size","isMale","isFemale")
for(i in 1:k_num_psets){
    full_data_set <- merge(full_data_set, graph_data_sets[[i]][,c("code",stats_we_want)])
    colnames(full_data_set)[colnames(full_data_set) %in% stats_we_want] <- paste(stats_we_want,i,sep = "_
}

full_data_set[,paste("test",1:2,sep = "")] <- full_data_set[,paste("test",1:2,sep = "")] /2
```

Flattening the problem

```
columns <- c("Code","Grade",stats_we_want)
flattened <- data.frame(matrix(0,ncol = length(columns),nrow = 9 * nrow(full_data_set)))
colnames(flattened) <- columns
current_row <- 1
for(i in 1:k_num_psets){
    for(j in 1:nrow(full_data_set)){
        flattened[current_row,] <- full_data_set[j,c("code",paste("hw",i,sep=""),
                                                  paste(columns[3:ncol(flattened)],i,sep = "_"))]
        current_row <- current_row + 1
    }
}
for(i in 1:2){
    for(j in 1:nrow(full_data_set)){
```

```r
    flattened[current_row,1:2] <- full_data_set[j,c("code",paste("test",i,sep=""))]
    flattened[current_row,"Constraint"] <- 1.5  #Thats what constraint is when there is no help
    current_row <- current_row + 1
  }
}
flattened$Code <- as.factor(flattened$Code)

thing_for_subsetting <- 0:(9*110-1) %/% 110
mean(flattened$Out_Degree[flattened$In_Degree==0 & thing_for_subsetting < 7]==0)
```

```r
#Problems with this prediction is that each problem set has a different mean,
#which makes it hard to know if you aren't including that in the sample.
#Additionally, the mean is probably more indicative of the assignment or grading
#than differential performance by students


flattened_mean  <- flattened
for(i in 0:8){
  flattened_mean[thing_for_subsetting==i,"Grade"] <-  (flattened_mean[thing_for_subsetting==i,"Grade"] -
}
```

```r
p <- ggplot(flattened_mean[thing_for_subsetting<7 & flattened_mean$In_Degree<4,],
            aes(factor(In_Degree), Grade))
p <- p + geom_violin(color = "grey50") +
  stat_summary(fun.y=mean, geom="point", size=2, colour="blue") +
  labs(x="In-Degree",y="Grade") + theme(legend.position="none") +
  ggtitle(expression(atop("Distribution of Grades by In-Degree",
                          atop(italic("Means visualized as points"), ""))))+
  theme(plot.title = element_text(hjust = 0.5,size = 22),
        axis.text=element_text(size=16),
        axis.title=element_text(size=20))

ggsave("Grades_by_In_Degree.png", plot = p,
       width = 7, height = 7, units = "in")



table(flattened_mean$Out_Degree)
table(flattened_mean$In_Degree)
flattened_mean$Code[flattened_mean$Out_Degree>=10]
for_plot <- flattened_mean
for_plot$Out_Degree[for_plot$Out_Degree>3] <- "4+"

p <- ggplot(for_plot[thing_for_subsetting<7,], aes(factor(Out_Degree), Grade))

p <- p + geom_violin(color = "grey50") +
  stat_summary(fun.y=mean, geom="point", size=2, colour="blue") +
  labs(x="Out-Degree",y="Grade") + theme(legend.position="none") +
  ggtitle(expression(atop("Distribution of Grades by Out-Degree",
                          atop(italic("Means visualized as points"), ""))))+
    theme(plot.title = element_text(hjust = 0.5,size = 22),
        axis.text=element_text(size=16),
        axis.title=element_text(size=20))
p
```

```r
ggsave("Grades_by_Out_Degree.png", plot = p,
       width = 7, height = 7, units = "in")
```

```r
#Difference from the means makes a huge differnce
thing_for_subsetting <- 0:(9*110-1) %/% 110
lm_list <- list()

error_dif <- 0
error_lm_total <- 0

#stepAIC(lm(data = flattened_mean, Grade ~ .  -isMale - isFemale - Time - Subsetting - Helpers - Helpees


formula <- Grade ~ Code +Out_Degree+Ave_Score_of_Helpees + In_Degree + Ave_Score_of_Helpers +
  (In_Degree== 0)

formula_2 <- Grade ~ Code + Constraint  + Recip_Degree + Big_Component +
  Out_Degree+Ave_Score_of_Helpees + In_Degree + Ave_Score_of_Helpers
formula_3 <- Grade ~ Code + Constraint  + Recip_Degree + Big_Component

formula_no_skip <- Grade ~ Code  + Ave_Score_of_Helpees+Out_Degree
formula_skip <- Grade ~ Code  + Ave_Score_of_Helpees+Recip_Degree + Constraint



skip_test <- F

formula_to_use <- formula_no_skip
if(skip_test){
  formula_to_use <- formula_skip
}


for(i in 1:k_num_psets){
  thing_to_avoid <- i-1
  if(skip_test){
    thing_to_avoid <- c(thing_to_avoid, 7,8)
  }
  subset <- ! thing_for_subsetting %in% thing_to_avoid
  lm1 <- lm(data = flattened_mean,
            formula_to_use,
            subset = subset)
  lm_list[[i]] <- lm1
  mu1 <- predict(lm1,newdata = flattened_mean[thing_for_subsetting ==(i-1),])
  ability_0 <- tapply(flattened_mean$Grade[subset],flattened_mean$Code[subset],mean)
  print(i)
  error_lm <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - mu1)**2)
  error_lm_total <- error_lm + error_lm_total
  error_ave <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - ability_0)**2)
  error_dif <- error_dif - error_lm + error_ave
  print(paste(error_lm,error_ave,sep = " | "))
}
```

```r
if(!skip_test){
  print("test")
  for(i in 1:2){
    lm1 <- lm(data = flattened_mean,
              formula_to_use,
              subset = thing_for_subsetting != (i+6))
    lm_list[[i+k_num_psets]] <- lm1
    mu1 <- predict(lm1,newdata = flattened_mean[thing_for_subsetting ==(i+6),])
    ability_0 <- tapply(flattened_mean$Grade[thing_for_subsetting != (i+6)],
                        flattened_mean$Code[thing_for_subsetting != (i+6)],mean)
    print(i)
    error_lm <-mean((flattened_mean[thing_for_subsetting ==(i+6),"Grade"] - mu1)**2)
    error_lm_total <- error_lm + error_lm_total
    error_ave <- mean((flattened_mean[thing_for_subsetting ==(i+6),"Grade"] - ability_0)**2)
    error_dif <- error_dif - error_lm + error_ave

    print(paste(error_lm,error_ave,sep = " | "))
  }
}
print('hopefully positive')
print(error_dif)

#now fit it on the whole data set so we get the best estimates

subset <- rep(T,nrow(flattened_mean))
if(skip_test){
  subset <- !thing_for_subsetting %in%  c(7,8)
}

lm1 <- lm(data = flattened_mean, formula_to_use,
          subset = subset)
mu1 <- c(lm1$coefficients[1],lm1$coefficients[1] +
           lm1$coefficients[grepl("Code",names(lm1$coefficients))])
names(mu1)[1] <- 1
names(mu1) <- gsub("Code","",names(mu1))
barplot(mu1)
mu1 <- as.data.frame(mu1)
mu1$Code <- rownames(mu1)

barplot(c(error_lm_total/ifelse(skip_test,7,9),
          (error_lm_total + error_dif)/ifelse(skip_test,7,9)),
        col=nice_colors)

error_lm_total / (error_lm_total + error_dif)
#the model is only a 3 percent improvement but its the best that I have gotten so far
#it is at least not overfit since its not grained on any data its predicting
visreg(lm1)
summary(lm1)
lm2 <- stepAIC(lm1,trace = 0)
summary(lm2)
```

```r
stargazer(lm2)
```

```
mean(lm1$residuals**2)
mean((flattened_mean$Grade-rep(tapply(flattened_mean$Grade,flattened_mean$Code,mean),9))**2)
#big improvment when you get the full sample too


flattened_plus_gender <- merge(flattened_mean[thing_for_subsetting < ifelse(skip_test,7,9),],
                               grades,by.x = "Code", by.y = "code")
flattened_plus_gender <- merge(flattened_plus_gender,mu1)
flattened_plus_gender %>%
  group_by(Sex) %>%
  summarise(mean_dev = mean(Grade), mean_coef = mean(mu1),
            median_dev = median(Grade), median_coef = mean(mu1),
            in_d = mean(In_Degree), out_d = mean(Out_Degree),
            good_grades = mean(Grade > median(flattened_plus_gender$Grade)),
            good_students = mean(mu1 > median(flattened_plus_gender$mu1)),
            good_grades_mean = mean(Grade > mean(flattened_plus_gender$Grade)),
            good_students_mean = mean(mu1 > mean(flattened_plus_gender$mu1)),
            isolates = mean(In_Degree==0 & Out_Degree == 0),
            no_in = mean(In_Degree == 0), no_out = mean(Out_Degree==0),
            help = mean(Ave_Score_of_Helpers), eigen = mean(Eigen))


a <- as.numeric(as.character(flattened_plus_gender$Code[!duplicated(flattened_plus_gender$Code)]))

to_plot <- flattened_plus_gender$mu1[!duplicated(flattened_plus_gender$Code)][order(a)]
barplot(to_plot[order(to_plot)],
        col = nice_colors[as.numeric(as.factor(flattened_plus_gender$Sex[!duplicated(flattened_plus_gen
qqnorm(to_plot)
#not normally distributed


grades_by_code <- flattened_plus_gender %>%
  group_by(Code) %>%
  summarise(average_grade = mean(Grade))



coef_percentiles <- ecdf(to_plot)(to_plot)
grades_by_code_percentile <- ecdf(grades_by_code$average_grade)(grades_by_code$average_grade)

percentile_df <- data.frame(matrix(NA,nrow = 110,ncol = 3))
colnames(percentile_df) <- c("Coef","Grade","Gender")
percentile_df$Coef <- coef_percentiles
percentile_df$Grade <- grades_by_code_percentile
percentile_df$Gender <- flattened_plus_gender$Sex[!duplicated(flattened_plus_gender$Code)][order(a)]

plot(percentile_df$Grade ~ percentile_df$Coef,
     col =  nice_colors[as.numeric(as.factor(percentile_df$Gender ))],
     pch = 16,
     ylab = "Grade Percentile",
     xlab = "Ability Percentile",
     main = "")
abline(b=1,a=0)
```

```r
legend("topleft",col = nice_colors[c(3,2,1)],pch = 16,
       c("Men","Women","Non-Binary"))


percentile_df$Helped <- percentile_df$Grade > percentile_df$Coef
percentile_df$Helped_Amount <- (percentile_df$Grade - percentile_df$Coef)
expectation <- mean(percentile_df$Helped)

t.test(percentile_df$Helped[percentile_df$Gender != "?"] ~ percentile_df$Gender[percentile_df$Gender !=

t.test(percentile_df$Helped_Amount[percentile_df$Gender != "?"] ~ percentile_df$Gender[percentile_df$Gen
t.test(percentile_df$Coef[percentile_df$Gender != "?"] ~ percentile_df$Gender[percentile_df$Gender != "
t.test(percentile_df$Grade[percentile_df$Gender != "?"] ~ percentile_df$Gender[percentile_df$Gender !=


percentile_df %>%
  group_by(Gender) %>%
  summarise(pct_helped = mean(Helped)/expectation)




#So this is pretty surprising.

#DIFFERENCE IS NOT SIGNIFICANT
#It seems that in this model, women are being advantaged by the collaboration.
#selection Bias towards more connected women?
```

```r
#One thing I still need to work out is how do I add the do students
#learn more feature because right now I dont really allow for that.
visreg(lm1,gg=TRUE)

i<-7

V(graphs[[i]])$ability_0 <-  tapply(flattened_mean$Grade[thing_for_subsetting != (8)],
                                    flattened_mean$Code[thing_for_subsetting != (8)],
                                    mean)
if(i==6){
  V(graphs[[i]])$ability_0 <- V(graphs[[i]])$hw6 -
    V(graphs[[i]])$ability_0 - mean(V(graphs[[i]])$hw6)
}else if(i==7){
  V(graphs[[i]])$ability_0 <- V(graphs[[i]])$hw7 -
    V(graphs[[i]])$ability_0 - mean(V(graphs[[i]])$hw7)
}


V(graphs[[i]])$resid <- lm1$residuals[thing_for_subsetting[thing_for_subsetting!=8]==(i-1)]
set.seed(1)
plot(graphs[[i]],
     edge.arrow.size=.2,
     edge.width = .4,
     vertex.frame.color=nice_colors[as.numeric(as.factor(V(graphs[[i]])$Sex))],
     vertex.size=abs(V(graphs[[i]])$resid) ,
     vertex.color=nice_colors[(V(graphs[[i]])$resid>0)+1],
     vertex.label="",
```

```
     main = i)
legend("topright",col =nice_colors[1:2],
       legend = c("Under Predicting","Over Predicting"),pch= 16,cex=.8,
       bty="n")
set.seed(1)
plot(graphs[[i]],
     edge.arrow.size=.2,
     edge.width = .4,
     vertex.frame.color=nice_colors[as.numeric(as.factor(V(graphs[[i]])$Sex))],
     vertex.size=abs(V(graphs[[i]])$ability_0) ,
     vertex.color=nice_colors[(V(graphs[[i]])$ability_0>0)+1],
     vertex.label="",
     main = i)
legend("topright",col =nice_colors[1:2],
       legend = c("Under Predicting","Over Predicting"),pch= 16,cex=.8,
       bty="n")
```

```
a<-tapply(flattened_mean$Grade,
          flattened_mean$Constraint,mean)
plot(names(a),
     a)
lm_con <- lm(a ~ as.numeric(names(a)))
abline(lm_con)
```

Expectation Maximization

```
ff <- Grade~.
z <- data.frame(matrix(0, nrow=nrow(flattened_mean),
                       ncol=nrow(graph_data_sets[[7]])))
colnames(z) <- sort(unique(flattened_mean$Code))

identity_cols <- c("Helpers","Helpees","Recips")
x <- lapply( identity_cols,function(x)
  data.frame(matrix(0,nrow=nrow(flattened_mean),ncol=nrow(graph_data_sets[[7]]))))
for(i in 1:length(x)){
  colnames(x[[i]]) <-  sort(unique(flattened_mean$Code))
}
for(i in 1:nrow(flattened_mean)){
  z[i, colnames(z) == flattened_mean$Code[i]] <- 1
  for(ii in 1:length(identity_cols)){
    people <- strsplit(flattened_mean[i,identity_cols[ii]] ,"|",fixed = T)[[1]]
    if(length(people)>0){
      for(person in people){
        #I am over writing this for things that are reciprocal
        #TODO: MAKE IT SO THAT PEOPLE ARE ONLY LISTED ONCE
        #Done
        x[[ii]][i, colnames(z) == person] <- 1
      }
    }
  }
}
x[[4]] <-z
```

```
###https://www.r-bloggers.com/fitting-a-model-by-maximum-likelihood/
```

```
ability_vec <- tapply(flattened_mean$Grade,flattened_mean$Code,mean)
names(ability_vec) <- paste("Code", unique(flattened_mean$Code),sep="_")
other_params_names <- c(identity_cols,"Sigma")
other_params <- rep(1,length(other_params_names))
names(other_params) <- other_params_names

params_vec <- c(ability_vec,other_params)

for_mle <- paste(paste(names(params_vec), "=",round(params_vec,3),sep=" "),collapse = ", ")
for_to_fit <- paste(names(params_vec),collapse = ", ")

x<-lapply(x,as.matrix)

LL <- function(Code_1 = as.double(spar), membership=x,grades=flattened_mean$Grade) {
  # Find residuals
  #
  components <-lapply(membership, function(y) y %*% Code_1[1:110])
  # Calculate the likelihood for the residuals (with mu and sigma as parameters)
  #
  ret_val <- -sum(suppressWarnings(dnorm(grades - components[[1]] * Code_1[111] - components[[2]] * Code
  return(ifelse(is.finite(ret_val),ret_val,10 ** 10))
}

spar <- list(Code_1 = 3.382, Code_2 = 3.104, Code_3 = 1.104, Code_6 = -1.952, Code_7 = -7.007, Code_8 =

#spar <- c(rep(0,113),3.2)

a<-optim(fn = LL,
         par = as.double(spar),
         method = "L-BFGS-B", lower = c(rep(-30,110),-1,-1,-1,0),
         upper = c(rep(30,110),1,1,1,10),control = list(maxit = 1))

system.time(LL())
ptm <- proc.time()
# Loop to time
for (i in 1:100){
  LL()
}
# Stop the clock
proc.time() - ptm
a$convergence
a$message
a$value
```

```
spar<-as.double(spar)
components <- lapply(x, function(y) y %*% spar[1:110])
errors <- flattened_mean$Grade - components[[1]] * spar[111] -
  components[[2]] * spar[112] - components[[3]] * spar[113] - components[[4]]
mean(errors ** 2)
spar<-as.double(spar)
components <- lapply(x, function(y) y %*% a$par[1:110])
errors <- flattened_mean$Grade - components[[1]] * a$par[111] -
  components[[2]] * a$par[112] - components[[3]] * a$par[113] - components[[4]]
mean(errors ** 2)
```

```
errors_baseline <- flattened_mean$Grade - rep(tapply(flattened_mean$Grade,
                                                     flattened_mean$Code,mean),9)
mean(errors_baseline ** 2)
```

comparing predictions with network vs. no information

```
#exclude tests because they don't have network information

thing_for_subsetting <- 0:(9*110-1) %/% 110
lm_list <- list()

error_dif <- 0
error_lm_total <- 0
error_rf_total <- 0
ability_0 <- rowMeans(full_data_set[,grepl("hw|test",colnames(full_data_set))])


flattened_mean$True_In <- flattened_mean$In_Degree - flattened_mean$Recip_Degree
flattened_mean$True_Out <- flattened_mean$Out_Degree - flattened_mean$Recip_Degree
for(i in 1:k_num_psets){
  lm1 <- lm(data = flattened_mean,
            Grade  ~ Constraint  + (Recip_Degree) + Big_Component +
              (In_Degree==0) + (Out_Degree ==0),
            subset = thing_for_subsetting != (i-1) & thing_for_subsetting <7)
  lm_list[[i]] <- lm1
  mu1 <- predict(lm1,newdata = flattened_mean[thing_for_subsetting ==(i-1) &
                                                thing_for_subsetting <7,])
  rf1 <-  randomForest(data = flattened_mean,
                       Grade  ~ Constraint  + Recip_Degree + Big_Component  +In_Degree + Out_Degree,
                       subset = thing_for_subsetting != (i-1) & thing_for_subsetting <7)
  mu2 <- predict(rf1,newdata = flattened_mean[thing_for_subsetting ==(i-1) &
                                                thing_for_subsetting <7,])

  print(i)
  error_lm <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - mu1)**2)
  error_lm_total <- error_lm + error_lm_total
  error_rf <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - mu2)**2)
  error_rf_total <- error_rf + error_rf_total
  error_ave <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - 0)**2)
  error_dif <- error_dif - error_lm + error_ave
  print(paste(error_lm,error_ave,sep = " | "))
}
barplot(c(error_lm_total/7,error_rf_total/7,
          (error_lm_total + error_dif)/7),col=nice_colors)

print(error_rf_total/7)
print((error_lm_total + error_dif)/7)

(error_lm_total/(error_lm_total + error_dif))
(error_rf_total/(error_lm_total + error_dif))

lm1 <- lm(data = flattened_mean,
          Grade  ~ Constraint  + (Recip_Degree) + Big_Component
          + (In_Degree==0) + (Out_Degree ==0),
```

```
          subset = thing_for_subsetting <7)

rf1 <-  randomForest(data = flattened_mean,
                     Grade  ~ Constraint  + Recip_Degree + Big_Component
                     +In_Degree + Out_Degree,
                     subset = thing_for_subsetting <7)

visreg(lm1)
visreg(rf1)
barplot(rf1$importance,beside = T,
        names.arg =rownames(rf1$importance),
        las =2,cex.names =.7,
        col = nice_colors)
```

```
thing_for_subsetting <- 0:(9*110-1) %/% 110
lm_list <- list()

error_dif <- 0
error_lm_total <- 0
error_rf_total <- 0
ability_0 <- rowMeans(full_data_set[,grepl("hw|test",colnames(full_data_set))])

flattened_mean$True_In <- flattened_mean$In_Degree - flattened_mean$Recip_Degree
flattened_mean$True_Out <- flattened_mean$Out_Degree - flattened_mean$Recip_Degree

rf_formula <- Grade  ~ Constraint  + Recip_Degree  +In_Degree +
  Out_Degree  + Ave_Score_of_Helpers + Ave_Score_of_Helpees+Ave_Score_of_Recip
lm_formula <- Grade ~ Constraint + Recip_Degree +
  Ave_Score_of_Helpers + Ave_Score_of_Helpees
rf_formula<-lm_formula
for(i in 1:k_num_psets){
  lm1 <- lm(data = flattened_mean,
            formula = lm_formula,
            subset = thing_for_subsetting != (i-1) & thing_for_subsetting <7)
  lm_list[[i]] <- lm1
  mu1 <- predict(lm1,newdata = flattened_mean[thing_for_subsetting ==(i-1) &
                                               thing_for_subsetting <7,])
  rf1 <-  randomForest(data = flattened_mean,
                       rf_formula,
                       subset = thing_for_subsetting != (i-1) & thing_for_subsetting <7,
                       ntree = 500)
  mu2 <- predict(rf1,newdata = flattened_mean[thing_for_subsetting ==(i-1) &
                                               thing_for_subsetting <7,])

  print(i)
  error_lm <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - mu1)**2)
  error_lm_total <- error_lm + error_lm_total
  error_rf <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - mu2)**2)
  error_rf_total <- error_rf + error_rf_total
  error_ave <- mean((flattened_mean[thing_for_subsetting ==(i-1),"Grade"] - 0)**2)
  error_dif <- error_dif - error_lm + error_ave
  print(paste(error_lm,error_ave,sep = " | "))
}
barplot(c(error_lm_total/7,error_rf_total/7,
```

```
          (error_lm_total + error_dif)/7),col=nice_colors)

(error_lm_total/(error_lm_total + error_dif))
(error_rf_total/(error_lm_total + error_dif))

print(error_rf_total / 7 )

lm1 <- lm(data = flattened_mean,
          lm_formula,
          subset = thing_for_subsetting <7)

rf1 <-  randomForest(data = flattened_mean,
                     rf_formula,
                     subset = thing_for_subsetting <7)

visreg(lm1)
visreg(rf1)
barplot(rf1$importance,beside = T,
        names.arg =rownames(rf1$importance),
        las = 2,
        cex.names =  .7,
        col = nice_colors)
```

```
to_plot <- c((error_lm_total + error_dif)/7, 22.55918, error_rf_total / 7,
             13.5,12.6)
names(to_plot) <- c("Baseline:\n No Information","Random Forest:\n Only Network",
                    "Random Forest:\nNetwork & Others' Grades",
                    "Baseline:\nStudent's Average",
                    "Linear Model:\nAll Information")
barplot(to_plot,cex.names = .8,
        main = "Mean Squared Error for Different Methods",
        col = brewer.pal(5,"RdYlBu"),
        ylab = "Mean Squared Error",
        ylim = c(0,30))
```

New data set for learning on tests

```
#Can Get 15% improvement of MSE
students <- sample(unique(flattened_mean$Code))
tests <- flattened_mean[thing_for_subsetting>6,1:2]
n_folds <- 10
folds <- sample(rep(1:n_folds,length.out = nrow(tests)))

for(i in 3:ncol(flattened_mean)){
  if(is.numeric(flattened_mean[,i])){
    tests[,colnames(flattened_mean)[i]] <- rep(tapply(flattened_mean[thing_for_subsetting < 7 ,i],
                                                      flattened_mean$Code[thing_for_subsetting < 7],
                                                      mean),2)
  }
}
tests$Ave_Grade <- rep(tapply(flattened_mean[thing_for_subsetting < 7,"Grade"],
                              flattened_mean$Code[thing_for_subsetting < 7],
                              mean),2)
```

```r
lm_t <- stepAIC(lm(tests,
                   formula = Grade ~ . - Code - isMale - isFemale -
                     Grade-Eigen-Betweenness - Component_Size))
formula_test <- Grade ~ Ave_Score_of_Helpers + In_Degree+ Ave_Score_of_Helpees +
  Recip_Degree + Ave_Grade
error_dif <- 0
error_baseline <- numeric(n_folds)
error_lm <- numeric(n_folds)
error_rf <- numeric(n_folds)

for(i in 1:n_folds){
  subset = tests$Code %in% students[((i-1) * 11+1):(i*11)]
  lm1 <- lm(data = tests,
            formula =formula_test,
            subset = !subset)
  rf1 <- randomForest(data = tests,
                      formula_test,
                      subset = !subset)
  mu1 <- predict(lm1, tests[subset, ])
  error_baseline[i] <- mean((tests$Grade[subset] - tests$Ave_Grade[subset])**2)
  error_lm[i] <- mean((tests$Grade[subset] - mu1)**2)
  error_rf[i] <- mean((tests$Grade[subset] - predict(rf1, tests[subset, ]))**2)
}
sum(error_lm) / sum(error_baseline)
sum(error_rf) / sum(error_baseline)


summary(lm_t)
visreg(lm_t)
```

```r
stargazer(lm_t)
```

simulating data to test for chance that this was just random

```r
empirical_sd <- 3.2 #this is from the residual standard error of a couple different models
just_hw <- thing_for_subsetting < 7
empirical_sd_no_person <- sd(flattened_mean$Grade[just_hw])
n_trials <- 100

ability_0 <- tapply(flattened_mean$Grade,flattened_mean$Code,mean)

ability_just_hw <- tapply(flattened_mean$Grade[just_hw],
                          flattened_mean$Code[just_hw],mean)

results_mat <- matrix(0,ncol = (6+2+2),nrow  = n_trials)
colnames(results_mat) <- c("MSE_just_net","MSE_net_collab",
                           "Ave_Score_of_Helpees_hw",
                           "Out_Degree_hw","(Intercept)_tests",
                           "Ave_Score_of_Helpers_tests",
                           "In_Degree_tests","Ave_Score_of_Helpees_tests",
                           "Recip_Degree_tests","Ave_Grade_tests")
```

```r
for(i in 1:n_trials){
  #flattened_mean$Trial_Just_HW <- rep(ability_just_hw,9) + rnorm(n = nrow(flattened_mean),
  #                                                                sd = empirical_sd_no_person)
  flattened_mean$Trial <- rep(ability_0,9) + rnorm(n = nrow(flattened_mean),
                                                   sd = empirical_sd)
  tests$Trial <- rep(ability_just_hw*.75,2) + rnorm(n = nrow(tests),
                                                    sd = empirical_sd)

  rf_just_network <- randomForest(data = flattened_mean,
                                  Trial ~ Constraint  + Recip_Degree +
                                    Big_Component  +In_Degree + Out_Degree,
                                  subset = just_hw)
  results_mat[i,1] <- mse(flattened_mean$Trial[just_hw],
                          rf_just_network) / var(flattened_mean$Trial[just_hw])


  flattened_mean$Ave_Score_of_Helpers_Trial <- 0
  flattened_mean$Ave_Score_of_Helpees_Trial <- 0
  flattened_mean$Ave_Score_of_Recip_Trial <- 0
  helpers <- strsplit(flattened_mean$Helpers, split = "|", fixed = T)
  helpees <- strsplit(flattened_mean$Helpees, split = "|", fixed = T)
  recips <- strsplit(flattened_mean$Recips, split = "|", fixed = T)
  for(j in 1:770){
    a <- ifelse(length(helpers[[j]])>0,
                flattened_mean$Trial[flattened$Code %in% helpers[[j]] &
                                       thing_for_subsetting == (j-1) %/% 110],0)
    b <- ifelse(length(helpees[[j]])>0,
                flattened_mean$Trial[flattened$Code %in% helpees[[j]] &
                                       thing_for_subsetting == (j-1) %/% 110],0)
    d <- ifelse(length(helpees[[j]])>0,
                flattened_mean$Trial[flattened$Code %in% recips[[j]] &
                                       thing_for_subsetting == (j-1) %/% 110],0)

    flattened_mean$Ave_Score_of_Helpers_Trial[j] <- ifelse(length(a)>0 &!any(is.na(a)),mean(a),0)
    flattened_mean$Ave_Score_of_Helpees_Trial[j] <- ifelse(length(b)>0 &!any(is.na(b)),mean(b),0)
    flattened_mean$Ave_Score_of_Recip_Trial[j] <- ifelse(length(d)>0 & !any(is.na(d)),mean(d),0)
  }

  rf_network_and_collab <- randomForest(data = flattened_mean,
                                        Trial  ~ Constraint  + Recip_Degree  +In_Degree
                                        + Out_Degree  + Ave_Score_of_Helpers_Trial +
                                          Ave_Score_of_Helpees_Trial +Ave_Score_of_Recip_Trial,
                                        subset = just_hw)
  results_mat[i,2] <- mse(flattened_mean$Trial[just_hw],
                          rf_network_and_collab) / var(flattened_mean$Trial[just_hw])

  lm_everything <- lm(data = flattened_mean,
                      Trial ~ Code  + Ave_Score_of_Helpees_Trial + Out_Degree)
  results_mat[i,3:4] <-  lm_everything$coefficients[111:112]

  lm_tests <- lm(data = tests,
                 Trial ~ Ave_Score_of_Helpers + In_Degree+ Ave_Score_of_Helpees +
                   Recip_Degree + Ave_Grade)
```

```
  results_mat[i,5:ncol(results_mat)] <- lm_tests$coefficients
}

mean(results_mat[,1] < results_mat[,2])

for(i in 1:ncol(results_mat)){
  hist(results_mat[,i],
       main = colnames(results_mat)[i])
  abline(v = quantile(results_mat[,i], probs = c(0.025,.975)),
         col = "red")
  print(colnames(results_mat)[i])
  print(quantile(results_mat[,i], probs = c(0.025,.975)))
}

apply(results_mat,2,function(x) round(mean(x),2))
```

```
empirical_sd <- 3.2 #this is from the residual standard error of a couple different models
just_hw <- thing_for_subsetting < 7
empirical_sd_no_person <- sd(flattened_mean$Grade[just_hw])
n_trials <- 100
ability_0 <- tapply(flattened_mean$Grade,flattened_mean$Code,mean)

ability_just_hw <- tapply(flattened_mean$Grade[just_hw],flattened_mean$Code[just_hw],mean)

results_mat <- matrix(0,ncol = (6+2+2),nrow  = n_trials)
colnames(results_mat) <- c("MSE_just_net","MSE_net_collab",
                           "Ave_Score_of_Helpees_hw",
                           "Out_Degree_hw","(Intercept)_tests",
                           "Ave_Score_of_Helpers_tests",
                           "In_Degree_tests","Ave_Score_of_Helpees_tests",
                           "Recip_Degree_tests","Ave_Grade_tests")



for(i in 1:n_trials){
  flattened_mean$Trial <- rep(ability_0,9) + rnorm(n = nrow(flattened_mean), sd = empirical_sd,
                                                    mean = mean(c(flattened_mean$In_Degree - mean(flatte
  tests$Trial <- rep(ability_just_hw*.75,2) + rnorm(n = nrow(tests), sd = empirical_sd,
                                                     mean = (tests$Out_Degree - tests$In_Degree))
  flattened_mean$Ave_Score_of_Helpers_Trial <- 0
  flattened_mean$Ave_Score_of_Helpees_Trial <- 0
  flattened_mean$Ave_Score_of_Recip_Trial <- 0
  helpers <- strsplit(flattened_mean$Helpers, split = "|", fixed = T)
  helpees <- strsplit(flattened_mean$Helpees, split = "|", fixed = T)
  recips <- strsplit(flattened_mean$Recips, split = "|", fixed = T)
  for(j in 1:770){
    a <- ifelse(length(helpers[[j]])>0,
                flattened_mean$Trial[flattened$Code %in% helpers[[j]] &
                                       thing_for_subsetting == (j-1) %/% 110],0)
    b <- ifelse(length(helpees[[j]])>0,
                flattened_mean$Trial[flattened$Code %in% helpees[[j]] &
                                       thing_for_subsetting == (j-1) %/% 110],0)
    d <- ifelse(length(helpees[[j]])>0,
                flattened_mean$Trial[flattened$Code %in% recips[[j]] &
                                       thing_for_subsetting == (j-1) %/% 110],0)
```

```
  flattened_mean$Ave_Score_of_Helpers_Trial[j] <- ifelse(length(a)>0 &!any(is.na(a)),mean(a),0)
  flattened_mean$Ave_Score_of_Helpees_Trial[j] <- ifelse(length(b)>0 &!any(is.na(b)),mean(b),0)
  flattened_mean$Ave_Score_of_Recip_Trial[j] <- ifelse(length(d)>0 & !any(is.na(d)),mean(d),0)
}

for(j in 1:770){
  if(flattened_mean$Ave_Score_of_Helpees_Trial[j] ==0 & flattened_mean$Ave_Score_of_Helpers_Trial[j] =
    flattened_mean$Trial[j] <- flattened_mean$Trial[j]
  }else if(flattened_mean$Ave_Score_of_Helpees_Trial[j] ==0){
    flattened_mean$Trial[j] <- mean(flattened_mean$Trial[j],
                                    flattened_mean$Ave_Score_of_Helpers_Trial[j])
  }else if(flattened_mean$Ave_Score_of_Helpers_Trial[j] ==0){
    flattened_mean$Trial[j] <- mean(flattened_mean$Trial[j],
                                    flattened_mean$Ave_Score_of_Helpers_Trial[j],
                                    flattened_mean$Ave_Score_of_Helpees_Trial[j])
  }
}
for(j in 1:770){
  a <- ifelse(length(helpers[[j]])>0,
              flattened_mean$Trial[flattened$Code %in% helpers[[j]] &
                                   thing_for_subsetting == (j-1) %/% 110],0)
  b <- ifelse(length(helpees[[j]])>0,
              flattened_mean$Trial[flattened$Code %in% helpees[[j]] &
                                   thing_for_subsetting == (j-1) %/% 110],0)
  d <- ifelse(length(helpees[[j]])>0,
              flattened_mean$Trial[flattened$Code %in% recips[[j]] &
                                   thing_for_subsetting == (j-1) %/% 110],0)

  flattened_mean$Ave_Score_of_Helpers_Trial[j] <- ifelse(length(a)>0 &
                                                     !any(is.na(a)),mean(a),0)
  flattened_mean$Ave_Score_of_Helpees_Trial[j] <- ifelse(length(b)>0 &
                                                     !any(is.na(b)),mean(b),0)
  flattened_mean$Ave_Score_of_Recip_Trial[j] <- ifelse(length(d)>0 &
                                                   !any(is.na(d)),mean(d),0)
}




rf_just_network <- randomForest(data = flattened_mean,
                                Trial ~ Constraint  + Recip_Degree
                                + Big_Component  +In_Degree + Out_Degree, subset = just_hw)
results_mat[i,1] <- mse(flattened_mean$Trial[just_hw],
                        rf_just_network) / var(flattened_mean$Trial[just_hw])

rf_network_and_collab <- randomForest(data = flattened_mean,
                                      Trial  ~ Constraint  + Recip_Degree  +In_Degree
                                      + Out_Degree  + Ave_Score_of_Helpers_Trial +
                                        Ave_Score_of_Helpees_Trial +Ave_Score_of_Recip_Trial,
                                      subset = just_hw)
results_mat[i,2] <- mse(flattened_mean$Trial[just_hw]
                        ,rf_network_and_collab) / var(flattened_mean$Trial[just_hw])
```

```r
  lm_everything <- lm(data = flattened_mean,
                      Trial ~ Code  + Ave_Score_of_Helpees_Trial + Out_Degree)
  results_mat[i,3:4] <-  lm_everything$coefficients[111:112]

  lm_tests <- lm(data = tests,
                 Trial ~ Ave_Score_of_Helpers + In_Degree+ Ave_Score_of_Helpees +
                   Recip_Degree + Ave_Grade)
  results_mat[i,5:ncol(results_mat)] <- lm_tests$coefficients
}
results_mat <- results_mat[1:i,]

mean(results_mat[,1] < results_mat[,2])

for(col in 1:ncol(results_mat)){
  hist(results_mat[,col],
       main = colnames(results_mat)[col])
  abline(v = quantile(results_mat[,col], probs = c(0.025,.975)),
         col = "red")
  print(colnames(results_mat)[col])
  print(quantile(results_mat[,col], probs = c(0.025,.975)))
}
apply(results_mat,2,function(x) round(mean(x),2))
```