R Notebook

```r
#install.packages("softImpute")
#install.packages("ranger")


library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.
```

```r
library(ranger)
```

```
##
## Attaching package: 'ranger'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v stringr 1.4.0
## v tidyr   1.0.0     v forcats 0.4.0
## v readr   1.3.1
```

```
## -- Conflicts ------------------------------------------------- tidyverse_conflicts() --
## x dplyr::combine()  masks randomForest::combine()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x ggplot2::margin() masks randomForest::margin()
```

```r
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
library(softImpute)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded softImpute 1.4

##
## Attaching package: 'softImpute'

## The following object is masked from 'package:tidyr':
##
##     complete
```

```r
OSR2 <- function(predictions, train, test) {
  SSE <- sum((test - predictions)^2)
  SST <- sum((test - mean(train))^2)
  r2 <- 1 - SSE/SST
  return(r2)
}

ratings <- read_csv("C:/Users/Murtz.Kizilbash/Desktop/ieor142/hw5/MusicRatings.csv")
```

```
## Parsed with column specification:
## cols(
##   userID = col_double(),
##   songID = col_double(),
##   rating = col_double()
## )
```

```r
View(ratings)
songs <- read.csv("C:/Users/Murtz.Kizilbash/Desktop/ieor142/hw5/Songs.csv")
users <- read.csv("C:/Users/Murtz.Kizilbash/Desktop/ieor142/hw5/Users.csv")
mergedratings <- merge(ratings, songs, by.x = "songID")

# clean up
range<-range(mergedratings$rating, na.rm = FALSE)
#ratings$userID <- as.factor(ratings$userID)
#ratings$songID <- as.factor(ratings$songID)
#ratings$rating <- as.factor(ratings$rating)
#users$userID <- as.factor(users$userID)
#songs$songID <- as.factor(songs$songID)
#songs$songName <- as.factor(songs$songName)
#songs$year <- as.factor(songs$year)
#songs$artist <- as.factor(songs$artist)
#songs$genre <- as.factor(songs$genre)


set.seed(345)
train.ids <- sample(nrow(mergedratings), 0.92*nrow(mergedratings))
test <- mergedratings[-train.ids,]
train <- mergedratings[train.ids,]

# split training into real training and validation set
# for hyperparameter search
val1.ids <- sample(nrow(train), (4/92)*nrow(train))
val1 <- train[val1.ids,]
train <- train[-val1.ids,]

# for blending
val2.ids <- sample(nrow(train), (4/92)*nrow(train))
val2 <- train[val2.ids,]
train <- train[-val2.ids,]

# First try CF
mat.train <- Incomplete(train$userID, train$songID, train$rating)
summary(train)
```

```
##      songID          userID         rating
##  Min.   :  1.0   Min.   :   1   Min.   :1.000
##  1st Qu.:191.0   1st Qu.: 599   1st Qu.:1.000
##  Median :400.0   Median :1211   Median :1.000
##  Mean   :398.4   Mean   :1209   Mean   :1.196
##  3rd Qu.:608.0   3rd Qu.:1816   3rd Qu.:1.301
##  Max.   :807.0   Max.   :2421   Max.   :3.433
##
##                   songName          year
##  Halo                   :  1400   Min.   :1975
##  They Might Follow You:  1310   1st Qu.:2003
##  Mia                    :  1222   Median :2007
##  Use Somebody           :  1218   Mean   :2005
##  Party In The U.S.A.    :  1217   3rd Qu.:2008
##  Clocks                 :  1184   Max.   :2010
```

```
##  (Other)                :236056
##                         artist              genre
##  Coldplay                  : 12683   Country   :  2282
##  The Killers               :  9710   Electronic: 24336
##  The New Pornographers     :  8918   Folk      :  7415
##  Kings Of Leon             :  8431   Pop       : 56710
##  Miley Cyrus               :  7754   Rap       :  8099
##  The All-American Rejects:  6756   RnB       : 12818
##  (Other)                   :189355   Rock      :131947
```

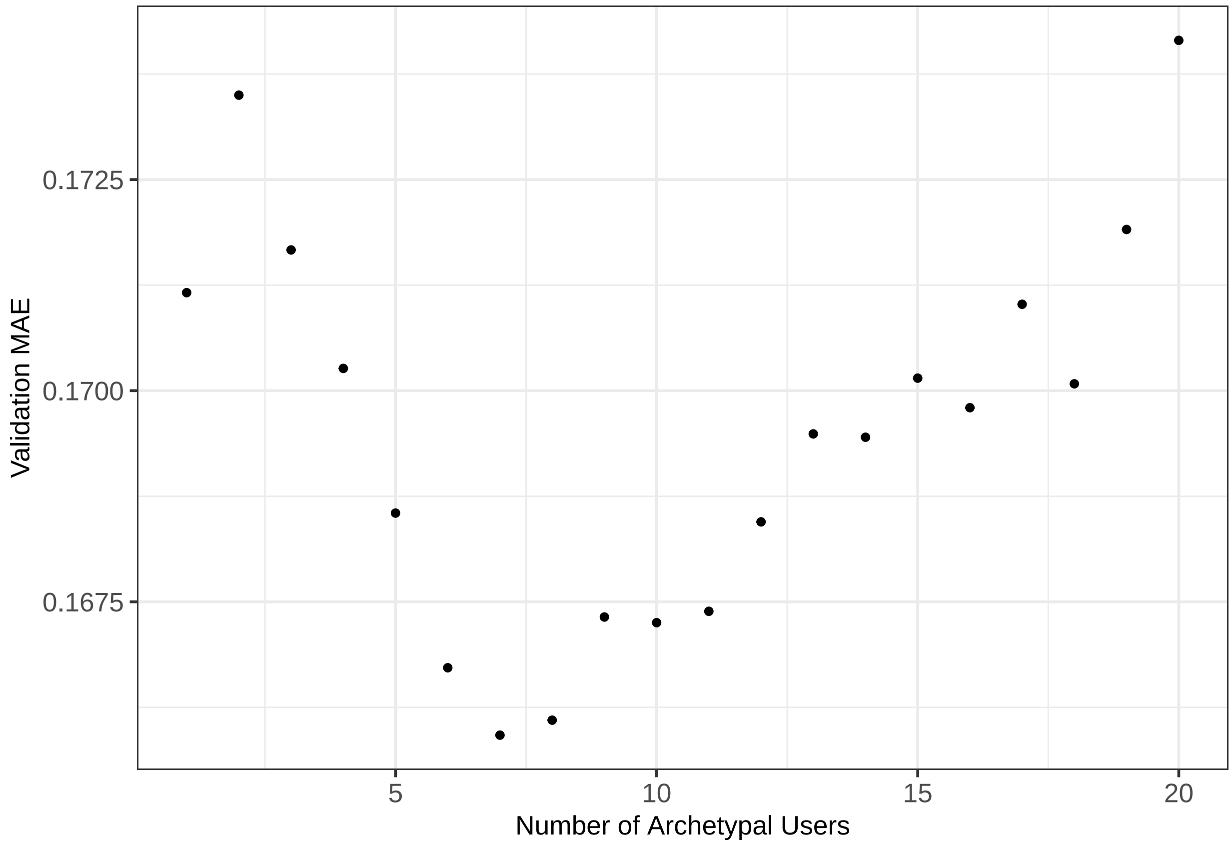
```r
### See Lab8-biscale.R for standardizing movie rating matrix using biScale function. Essentially X_ij -
mat.train.centered <- biScale(mat.train, maxit = 1000, row.scale = FALSE, col.scale = FALSE)

# TODO: GET THREE HIGHEST ALPHAS AND BETAS & Recover their id's in the tables
alpha <- attr(mat.train.centered, "biScale:row")$center
beta <- attr(mat.train.centered, "biScale:column")$center

# compute validation set MAE for rank = 1,2,...,20
# softImpute: fit a low-rank matrix approximation to a matrix with missing values
# impute(object, i, j): produce predictions from the low-rank solution of softImpute
mae.vals = rep(NA, 20)
for (rnk in seq_len(20)) {
  print(str_c("Trying rank.max = ", rnk))
  mod <- softImpute(mat.train, rank.max = rnk, lambda = 0, maxit = 1000)
  preds <- impute(mod, val1$userID, val1$songID) %>% pmin(3.43) %>% pmax(1) # clip rating from 1 to 5
  mae.vals[rnk] <- mean(abs(preds - val1$rating))
}
```

```
## [1] "Trying rank.max = 1"
## [1] "Trying rank.max = 2"
## [1] "Trying rank.max = 3"
## [1] "Trying rank.max = 4"
## [1] "Trying rank.max = 5"
## [1] "Trying rank.max = 6"
## [1] "Trying rank.max = 7"
## [1] "Trying rank.max = 8"
## [1] "Trying rank.max = 9"
## [1] "Trying rank.max = 10"
## [1] "Trying rank.max = 11"
## [1] "Trying rank.max = 12"
## [1] "Trying rank.max = 13"
## [1] "Trying rank.max = 14"
## [1] "Trying rank.max = 15"
## [1] "Trying rank.max = 16"
## [1] "Trying rank.max = 17"
## [1] "Trying rank.max = 18"
## [1] "Trying rank.max = 19"
## [1] "Trying rank.max = 20"
```

```r
mae.val.df <- data.frame(rnk = seq_len(20), mae = mae.vals)
ggplot(mae.val.df, aes(x = rnk, y = mae)) + geom_point(size = 1) +
  ylab("Validation MAE") + xlab("Number of Archetypal Users") +
  theme_bw() + theme(axis.title=element_text(size=10), axis.text=element_text(size=10))
```




```
minval <- min(mae.vals)

# choose k = 9
set.seed(345)
mod.final <- softImpute(mat.train, rank.max = 9, lambda = 0, maxit = 1000)
preds <- impute(mod.final, test$userID, test$songID) %>% pmin(3.43) %>% pmax(1)

mean(abs(preds - test$rating))
```

```
## [1] 0.1661714
```

```
sqrt(mean((preds - test$rating)^2))
```

```
## [1] 0.2302475
```

```
OSR2(preds, train$rating, test$rating)
```

```
## [1] 0.3075651
```

```
# MERGE DATA SETS FOR BLENDING INSIGHTS
# Now try a linear regression without CF as a varible
lin.mod <- lm(rating ~ . -userID -songID -songName -artist -songName , data = train)
summary(lin.mod)
```

```
##
## Call:
## lm(formula = rating ~ . - userID - songID - songName - artist -
##     songName, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4377 -0.1999 -0.1675  0.1297  2.1416
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      9.0886960  0.2431902   37.37   <2e-16 ***
## year            -0.0038447  0.0001216  -31.62   <2e-16 ***
## genreElectronic -0.1455729  0.0061391  -23.71   <2e-16 ***
## genreFolk       -0.1542713  0.0066935  -23.05   <2e-16 ***
## genrePop        -0.1609018  0.0060142  -26.75   <2e-16 ***
## genreRap        -0.1508022  0.0066263  -22.76   <2e-16 ***
## genreRnB        -0.2544473  0.0063480  -40.08   <2e-16 ***
## genreRock       -0.2009932  0.0058993  -34.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2784 on 243599 degrees of freedom
## Multiple R-squared:  0.01612,    Adjusted R-squared:  0.01609
## F-statistic:   570 on 7 and 243599 DF,  p-value: < 2.2e-16
```

```r
preds.lm <- predict(lin.mod, newdata = test) %>% pmin(5) %>% pmax(1)
mean(abs(preds.lm - test$rating))
```

```
## [1] 0.2264584
```

```r
sqrt(mean((preds.lm - test$rating)^2))
```

```
## [1] 0.2744678
```

```r
OSR2(preds.lm, train$rating, test$rating)
```

```
## [1] 0.01605315
```

```r
# Now try random forests (Warning: this took 2 hours to run)
set.seed(345)
rf.mod <- ranger(rating ~ . -userID -songID -songName -artist -songName,
                 data = train,
                 mtry = floor((ncol(train) - 3)/3),
                 num.trees = 100,
                 verbose = TRUE)

preds.rf <- predict(rf.mod, data = test)
preds.rf <- preds.rf$predictions
mean(abs(preds.rf - test$rating))
```

```
## [1] 0.2243339
```

```r
sqrt(mean((preds.rf - test$rating)^2))
```

```
## [1] 0.2715359
```

```r
OSR2(preds.rf, train$rating, test$rating)
```

```
## [1] 0.03696217
```

```r
# --- Blending
val.preds.cf <- impute(mod.final, val2$userID, val2$songID)
val.preds.lm <- predict(lin.mod, newdata = val2)
val.preds.rf <- predict(rf.mod, data = val2)$predictions

# Build validation set data frame
val.blending_df = data.frame(rating = val2$rating, cf_preds = val.preds.cf, lm_preds = val.preds.lm, rf_
#val.blending_df = data.frame(rating = val2$rating, cf_preds = val.preds.cf, lm_preds = val.preds.lm, )

# Train blended model
blend.mod = lm(rating ~ . -1, data = val.blending_df) # -1: no intercept
summary(blend.mod)
```

```
##
## Call:
## lm(formula = rating ~ . - 1, data = val.blending_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.09453 -0.15608 -0.06672  0.14617  1.71772
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## cf_preds 0.699856   0.010592  66.073  < 2e-16 ***
## lm_preds 0.002441   0.072765   0.034    0.973
## rf_preds 0.305409   0.073827   4.137 3.55e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.235 on 11070 degrees of freedom
## Multiple R-squared:  0.9635, Adjusted R-squared:  0.9635
## F-statistic: 9.747e+04 on 3 and 11070 DF,  p-value: < 2.2e-16
```

```r
# Get predictions on test set
test.preds.cf <- impute(mod.final, test$userID, test$songID)
test.preds.lm <- predict(lin.mod, newdata = test)
test.preds.rf <- predict(rf.mod, data = test)$predictions

test.blending_df = data.frame(rating = test$rating, cf_preds = test.preds.cf, lm_preds = test.preds.lm,
#test.blending_df = data.frame(rating = test$rating, cf_preds = test.preds.cf, lm_preds = test.preds.lm

test.preds.blend <- predict(blend.mod, newdata = test.blending_df)

mean(abs(test.preds.blend - test$rating))
```

```
## [1] 0.182587
```

```r
sqrt(mean((test.preds.blend - test$rating)^2))
```

```
## [1] 0.2348386
```

```r
OSR2(test.preds.blend, train$rating, test$rating)
```

```
## [1] 0.2796758
```

```r
OSR2(test.preds.cf, train$rating, test$rating)
```

```
## [1] 0.2140107
```

```r
OSR2(test.preds.lm, train$rating, test$rating)
```

```
## [1] 0.01605315
```

```r
OSR2(test.preds.rf, train$rating, test$rating)
```

```
## [1] 0.03696217
```