Report-Models

```
## -- Attaching packages ---------------------------------------------------------------------------------

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0

## -- Conflicts -----------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Models

NaiveBayes model

Using formula rating ~ userId + MovieId + month.
Tried adding genre but didn't add to accuracy but took much longer to build model.

```r
library(tidyverse)
library(caret)
library(lubridate)

edx <- readRDS(file = "./rda/small_edx.rds")
validation <- readRDS (file = "./rda/small_Validation.rds")

rating_levels <- c("0.5", "1", "1.5", "2", "2.5", "3", "3.5", "4", "4.5", "5")

train_set <- edx %>%
  mutate(month = as.character(round_date(as_datetime(timestamp), unit = "month")),
         rating = factor(rating, levels = rating_levels))

test_set <- validation %>%
  mutate(month = as.character(round_date(as_datetime(timestamp), unit = "month")),
         rating = factor(rating, levels = rating_levels))

fit <- train(rating ~ userId + movieId + month,
             method = "naive_bayes",
             data = train_set)

naive_bayes_predictions <- predict(fit, test_set, type = "raw")

final_nb_predictions <-
  data.frame(userId = as.character(validation$userId),
             movieId = as.character(validation$movieId),
             nb_preds = naive_bayes_predictions,
             stringsAsFactors = FALSE)

confusionMatrix(final_nb_predictions$nb_preds, test_set$rating)

#
# add predicitions to ensemble
#
```

```
final_nb_predictions <- final_nb_predictions %>%
  mutate(nb_preds = as.double(nb_preds))

all_model_predictions <- all_model_predictions %>%
    left_join(final_nb_predictions, by = c("userId", "movieId"))


rm(train_set, test_set, fit, naive_bayes_predictions, final_nb_predictions)
```

IBCF

```
library(caret)
library(tidyverse)
library(recommenderlab)

edx <- readRDS(file = "./rda/small_edx.rds")
validation <- readRDS (file = "./rda/small_Validation.rds")

rating_levels <- c("0.5", "1", "1.5", "2", "2.5", "3", "3.5", "4", "4.5", "5")

# can't afford to run on entire dataset.  just use the top_n rated movies.
# temp <- edx %>%
#   group_by(movieId) %>%
#   summarise(count = n()) %>%
# #  top_n(1000, wt = count) %>%
#   arrange(-count) %>%
#   mutate(running_total = cumsum(count))

#
# reduce the size of the training set to only those items in the validation set
# no reason to use resources to predict ratings on non-requested items
#
train_set <- edx %>%
  filter(movieId %in% unique(validation$movieId))

test_set <- validation

r <- as(train_set, "realRatingMatrix")
rec.model <- Recommender(data = r,
                        method = "IBCF",
                        parameter = list(method = "Cosine"))

pred <- predict(rec.model, r, type = "ratings")
pred_matrix <- as(pred, "matrix")

# make into a df
ibcf_preds <- pred_matrix %>%
  as.data.frame.matrix() %>%
  rownames_to_column(var = "userId") %>%
  gather(key = movieId, value = IBCF_rating, -userId)

test_set <- test_set %>%
 mutate(userId = as.character(userId),
        movieId = as.character(movieId)) %>%
 select(userId, movieId)

final_ibcf_predictions <- left_join(test_set, ibcf_preds)

#
# add predictions to ensemble
#
all_model_predictions <- all_model_predictions %>%
    left_join(final_ibcf_predictions, by = c("userId", "movieId"))

rm(train_set, test_set, rec.model, pred, pred_matrix, r, ibcf_preds, final_ibcf_predictions)
```

Custom model

R = mu + b_i + b_i_time + b_u, where:

mu = the average of all movie ratings by all users

b_i = the movie bias, b_i = mu - average rating of this movie by all users

b_i_time = bias (popularity) of a movie during a given time period.

b_u = the user bias, b_u = Y - mu - b_i

```
#
# R = mu + b_i + b_i_time + b_u where:
#   mu = the average of all movie ratings by all users
#   b_i = the movie bias, b_i = mu - average rating of this movie by all users
#   b_i_time = bias (popularity) of a movie during a given time period.
#   b_u = the user bias, b_u = Y - mu - b_i
#

library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tidyverse)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
#edx <- readRDS(file = "./rda/small_edx.rds")
#validation <- readRDS (file = "./rda/small_Validation.rds")

train_set <- edx %>%
  mutate(userId = as.character(userId),
         movieId = as.character(movieId),
         rating_date = round_date(as_datetime(timestamp), unit = "day"))

test_set <- validation %>%
  mutate(userId = as.character(userId),
         movieId = as.character(movieId),
         rating_date = round_date(as_datetime(timestamp), unit = "day"))

mu <- mean(train_set$rating)

#
# calculate temporal part of item bias, avg rating of item i at time t
# and add to train_set
#

# build time bins
d1 <- round_date(as_datetime(min(train_set$timestamp)), unit = "day")
d2 <- round_date(as_datetime(max(train_set$timestamp)), unit = "day")
d3 <- interval(d1, d2)
num_bins <- ceiling(d3 / dweeks(10))
b <- d1 + c(0:num_bins) * weeks(10)
t_bins <- data.frame(binId = seq(1, num_bins),
                  start = b[1:num_bins],
                  end = b[2:(num_bins + 1)])

rm(d1, d2, d3, b)
gc()


# Get the time bin this rating date falls into
# Assume t_bins is sorted by ascending end date
# Loop thru t_bins until d < end_date, then break
get_bin <- function(d) {
  bin <- 0
  for (j in seq(1, length(d))) {
    for (i in seq(1, nrow(t_bins))) {
      if (d[j] < t_bins$end[i]) {
        bin[j] <- i
        break
      }
    }
  }
  return(bin)
}

#
# find avg for each movie in an interval and store in matrix
#

# build b_i_time matrix to store the bias
b_i_time_matrix <- matrix(nrow = length(unique(train_set$movieId)), ncol = num_bins)
rownames(b_i_time_matrix) <- unique(train_set$movieId)
colnames(b_i_time_matrix) <- c(1:num_bins)
```

```r
b_i_time_matrix[,] <- 0

# get the time bin for each rating_date
train_set <- train_set %>%
  mutate(time_bin = get_bin(rating_date))

# calc b_i_time = the avg rating for each movie during each time bin
movie_time_bias <- train_set %>%
  group_by(movieId, time_bin) %>%
  summarise(b_i_time = mean(rating - mu))

# store this bias in the matrix [movieId, time_bin]
for(i in 1:nrow(movie_time_bias)) {
    row <- movie_time_bias[i,]
    r <- row$movieId
    c <- row$time_bin
    b_i_time_matrix[r, c] <- row$b_i_time
}

rm(movie_time_bias, row, c, r, i)
gc()

#
# lookup to find a b_i_time
#
get_b_i_time <- function(m, b) {
  bias <- 0
  for (j in 1:length(m)) {
    bias[j] <- b_i_time_matrix[m, b]
  }
  return(bias)
}


# calculate base movie bias = avg rating of the movie
# lambda manually tuned
lambda1 <- 10
movie_bias <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + lambda1))


# calculate user bias
# lambda manually tuned
lambda2 <- 5
user_bias <- train_set %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu) / (n() + lambda2) )

calc_rating <- function(movieId, time_bin, mu, b_i, b_u, rating_date) {

  # base prediction
  pred <- mu + b_i + b_i_time_matrix[movieId, time_bin] + b_u
  # if rated before 2003-05-15 then round-upwards to whole star
  # this produces better accuracy than either round or floor
  pred <- ifelse(rating_date < ymd("2003-05-15"),
                 ceiling(pred),
                 pred)

  # Accruracy is better without considering this
  #
  # Consider fewer 1/2 star ratings than whole star ratings
  # if pred is "near" to whole star then round to whole star
  # otherwise round to 1/2 star
  #
  # rounding thresholds (.4 and .6) optimized by multiple trials
#
#   pred <- ifelse(pred - as.integer(pred) < .5,
#                  floor(pred),
#           ifelse(pred - as.integer(pred) > .5,
#                  ceiling(pred),
#           as.integer(pred) + .5 ) )

  # recover extreme predictions
  pred <- if_else(pred < .5, .5, pred)
  pred <- if_else(pred > 5, 5, pred)

  return(pred)

}


#
# calculate the predictions
#
final_custom_predictions <- test_set %>%
  mutate(time_bin = get_bin(rating_date)) %>%
  left_join(movie_bias, by="movieId") %>%
  left_join(user_bias, by="userId") %>%
  mutate(custom_pred = calc_rating(movieId, time_bin, mu, b_i, b_u, rating_date)) %>%
  select(userId, movieId, custom_pred)

# add predictions to ensemble
all_model_predictions <- all_model_predictions %>%
    left_join(final_custom_predictions, by = c("userId", "movieId"))

#
# Create confusion matrix
#
rating_levels <- c("0.5", "1", "1.5", "2", "2.5", "3", "3.5", "4", "4.5", "5")

final_custom_predictions <- final_custom_predictions %>%
  mutate(custom_pred = factor(custom_pred, levels = rating_levels))

test_set <- test_set %>%
  mutate(rating = factor(rating, levels = rating_levels))
```

```r
q <- confusionMatrix(final_custom_predictions$custom_pred,
                     factor(validation$rating, rating_levels))$overall["Accuracy"]

rm(final_custom_predictions, movie_bias, test_set, train_set, user_bias)
rm(lambda1, lambda2, mu)
rm(b_i_time_matrix, t_bins, num_bins, rating_levels)
rm(calc_rating, get_b_i_time, get_bin)
gc()
```

Ensemble Evaluation

```r
library(tidyverse)
library(caret)

mjb <- all_model_predictions %>%
  select(-userId, -movieId, -rating) %>%
  as.matrix()

rating_levels <- c("0.5", "1", "1.5", "2", "2.5", "3", "3.5", "4", "4.5", "5")
all_model_predictions$ensemble_pred <- round(rowMeans(mjb, na.rm = TRUE) / 0.5) * 0.5

confusionMatrix(factor(all_model_predictions$ensemble_pred, rating_levels),
                factor(validation$rating, rating_levels))
```