Predicting Ratings in MovieLens 10M

Nathan Harris

2/25/2020

Capstone Project for HarvardX 125, Introduction to Data Science

Overview

MovieLens.org is a movie rating website, created and maintained by the GroupLens research lab at the University of Minnesota. The website was launched in the last quarter of 1997 in response to the the shuttering of movie recommender EachMovie. Use of the site has grown steadily; there are now over five thousand monthly active users.[^1]

GroupLens uses MovieLens to run field experiments in recommendation and provides user, movie, and rating information in a number of datasets. The MovieLens datasets range up to 25 million lines long, and have been used widely in education and research since they were first made available in 1998. The 10 million line MovieLens set was used in the analysis that follows.[^1]

The file contains exactly 10,000,054 lines, with each line representing a discrete rating of a movie by a user. The ratings range from zero to five, by half-unit increments, and were collected between January 1995 and January 2009. This spans the time MovieLens began allowing users to make half-unit ratings in February, 2003. These 10 million ratings represent 69,878 users ranking 95,580 movies.[^1]

This analysis details the creation of an algorithm to predict ratings based on information in the MovieLens data. The success measure is a minimized root mean square error value. This choice of metric, rather than another measure like accuracy, provides some guidance in model development. The aim will be to come as close as possible to the actual rating in each case, rather than to maximize the number of times exactly the correct rating is produced.

The 10 million line dataset was split into three parts for the purposes of analysis: a validation data set with about one million lines, a test set for cross validation and tuning purposes about the same size, and a training set made up of the remaining eight million lines.

Predictions were made beginning with the simplest algorithm, then slowly increasing in complexity. RMSE for the test set was used as the benchmark at each stage.

Methods

Machine learning approaches can be broadly categorized into classification and regression techniques. Classification assigns an observation to one of several discrete groups, while regression predicts a continuous value. Regression approaches were focused on here, as higher-rated films should see commensurate increases in other metrics. Classification approaches would have the benefit of only assigning values that are actual MovieLens ratings, but would lose connections between movies that users considered better or worse. Regression methods can produce highly inaccurate data without rounding or further processing (a prediction of 3.128462 will never exactly match a rating of 3 or 3.5), but RMSE is not hampered.

Recommendation generators can further be classified as memory or model-based. Any full-scale deployed algorithm will be based on a model exploiting similarities between observations. Clustering users or movies

into cohorts to make predictions is the norm here, famously used in the Netflix challenge, where a similar approach was applied to win a cash prize from the streaming service. Memory-based approaches, in this case, using characteristics of a movie rating appended to its row in the table to predict rating, are easy to set up, but don't scale up well.

We will begin with simple memory-based approaches. The basic setup and data cleaning code come from the HarvardX Introduction to Data Science course. The initial analysis approach follows the model set out in the course textbook, with elaboration.[^2]

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The edx dataframe was further partitioned into a train and test set.

```
#create test and train sets, test set is about equal in size to validation set
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.111, list = FALSE)
```

```
edx_train <- edx[-test_index,]
edx_test <- edx[test_index,]
edx_train <- edx_train %>%
  semi_join(edx_test, by = "movieId") %>%
  semi_join(edx_test, by = "userId")
```

Here is a sample of the data contained in the edx_train dataframe:

```
head(edx_train)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     292      5 838983421               Outbreak (1995)
## 3      1     329      5 838983392 Star Trek: Generations (1994)
## 4      1     356      5 838983653           Forrest Gump (1994)
## 5      1     362      5 838984885       Jungle Book, The (1994)
## 6      1     364      5 838983707         Lion King, The (1994)
##                                 genres
## 1                       Comedy|Romance
## 2                 Action|Drama|Sci-Fi|Thriller
## 3                Action|Adventure|Drama|Sci-Fi
## 4                     Comedy|Drama|Romance|War
## 5                    Adventure|Children|Romance
## 6 Adventure|Animation|Children|Drama|Musical
```

The average rating in edx_train is:

```
om<-mean(edx_train$rating)
om
```

```
## [1] 3.512411
```

The predictive power of this single value is:

```
RMSE(mean(edx_train$rating),edx_train$rating)
```

```
## [1] 1.059956
```

User Means and Movie Means

Particular characteristics of the observation can improve the quality of this simple prediction. Identity of the user and specific film are obvious places to start. The predictive power of a user average:

```
la<-0
um<-edx_train%>%group_by(userId)%>%summarize(usermean=om+(sum(rating-om)/(n()+la)))
umpred<-edx_train%>%left_join(um,by="userId")
RMSE(umpred$usermean,edx_train$rating)
```

```
## [1] 0.9692256
```

And the error of a movie's average rating:

```
lc<-0
mm<-edx_train%>%group_by(movieId)%>%summarize(moviemean=om+(sum(rating-om)/(n()+lc)))
mmpred<-umpred%>%left_join(mm,by="movieId")
RMSE(mmpred$moviemean,edx_train$rating)
```

```
## [1] 0.9418797
```

The la and lc values will remain at zero for now, with values to be set and explained later.

Combining both approaches yields better results:

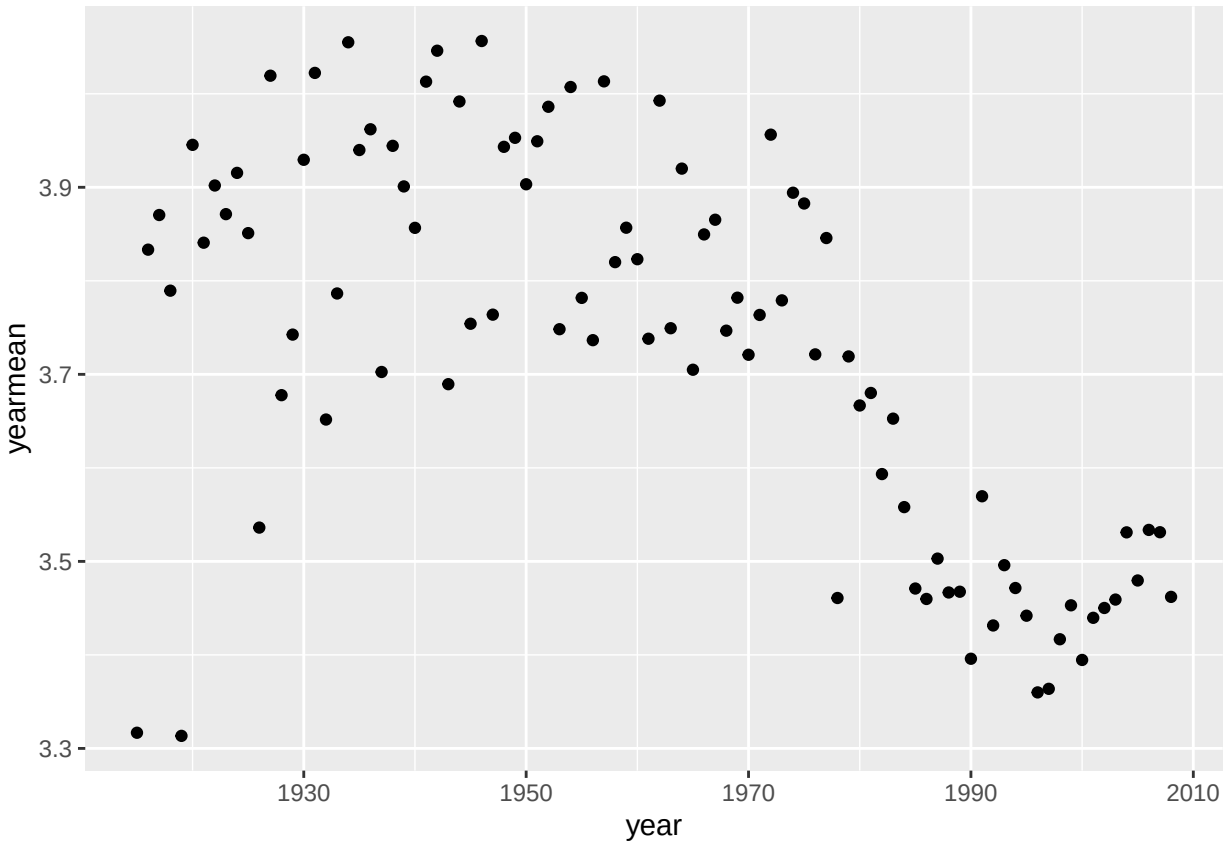
```
umm<-mmpred%>%mutate(usermoviemean=usermean+moviemean-om)

RMSE(umm$usermoviemean,edx_train$rating)
```

```
## [1] 0.8760619
```

Year of Release

There are other characteristics of each observation that might yield more predictive power. The data set gives the year in which each movie was made, though as part of the title. Extracting the year, then finding a yearly average shows some sign of a relationship, but predicting only based on the year is not very powerful.

```
ly=0
yearslist<-as.numeric(gsub("[^0-9]","",str_extract(edx_train$title,"\\([0-9]{4}\\)")))
trainwithyears<-mutate(edx_train, year=yearslist)
ym<-trainwithyears%>%group_by(year)%>%summarize(yearmean=om+(sum(rating-om)/(n()+ly)))
ggplot(data=ym,(aes(year,yearmean)))+geom_point()
```




```
yplot<-trainwithyears%>%left_join(ym,by="year")
RMSE(yplot$yearmean,edx_train$rating)
```

```
## [1] 1.04891
```

Unlike the addition of user and movie averages, adding the year average into the previous prediction actually increases RMSE.

```
yumm<-umm%>%mutate(yearmean=yplot$yearmean,usermovieyearmean=usermoviemean+yearmean-om)
RMSE(yumm$usermovieyearmean,edx_train$rating)
```

```
## [1] 0.8929947
```

Movie Genres

The genres column can also be mined for patterns to exploit. There is very little variation in the presence of individual genre descriptors like "Drama" or "Comedy," but the tags for an individual movie can be very distinctive. Most films in the dataset carry some common genre labels, but a large number of descriptor combinations are used only a few times.

```
lg<-0
gm<-edx_train%>%group_by(genres)%>%summarize(genremean=om+(sum(rating-om)/(n()+lg)))
gyumm<-yumm%>%left_join(gm,by="genres")%>%mutate(usermovieyeargenremean=usermovieyearmean+genremean-om)
#genre alone
RMSE(gyumm$genremean,edx_train$rating)
```

```
## [1] 1.017562
```

```
# user, movie, genre, year combined
RMSE(gyumm$usermovieyeargenremean,edx_train$rating)
```

```
## [1] 0.9598193
```

Regularization via Sum of Least Squares

Now is the time to revisit the la, lc, lg and ly parameters that have, up until now, been set to zero. In the year vs yearmean plot above, notice that some of the oldest year values seem out-of-trend. In fact, these oldest years have only a few ratings each, meaning they are prone to be influenced by extreme values, high or low. Adding a regularization factor to our arithmetic means can pull these extreme values back toward the overall mean rating. Instead of dividing the differences between each rating and the overall average by the number of ratings, we can divide by a larger number and shrink the value.

The best value can be found through cross validation. The validation set can't be touched for this, and while the test set could be used to check, overtraining risks can best be avoided by working within the train set. For each potential correction factor, ten bootstrap samples can be pulled from a random sample consisting of ten percent of the train set. RMSEs can be averaged, and minimum RMSE will help pick the factor.

```
#calculate lambda values for least squares regularization
lambdas<-function(i,y,z){
  #overall mean rating
  omtemp<-mean(y$rating)

  #finding the regularized mean rating
  mtemp<-y%>%group_by_(z)%>%summarize(mean=omtemp+(sum(rating-omtemp)/(n()+i)))

  #adding value and sampling ten percent at random for bootstrapping
  temp_comp<-y%>%left_join(mtemp,by=z)
  length<-nrow(temp_comp)
  temp_comp_small<-temp_comp[sample(c(1:length), length/10, replace=FALSE),]

  #bootstrapping samples
  bstps<-createResample(1:nrow(temp_comp_small),times=10,list=TRUE)
  rsmes<-sapply(bstps,function(x){RMSE(temp_comp_small$mean[x],temp_comp_small$rating[x])})
  final<-c(i,mean(rsmes))
  final
}
```

Applying the function above will tune the correction factor for each of the four predictors. The sequence of potential values to be checked for each of the four parameters is the result of data exploration for each of the variables.

```
#train set user mean rating
cvls<-sapply(seq(0,8,0.5),function(x){lambdas(i=x,y=edx_train,z="userId")})
```

```
## Warning: group_by_() is deprecated.
## Please use group_by() instead
##
## The 'programming' vignette or the tidyeval book can help you
## to program with group_by() : https://tidyeval.tidyverse.org
## This warning is displayed once per session.
```

```
la<-cvls[1,which.min(cvls[2,])]#col 1 val with min of second column in cvls
um<-edx_train%>%group_by(userId)%>%summarize(usermean=om+(sum(rating-om)/(n()+la)))

#train set movie mean rating
cvlm<-sapply(seq(2,7,0.5),function(x){lambdas(i=x,y=edx_train,z="movieId")})
lc<-cvlm[1,which.min(cvlm[2,])]#col 1 val  with min of second column in cvlm
mm<-edx_train%>%group_by(movieId)%>%summarize(moviemean=om+(sum(rating-om)/(n()+lc)))

#train set genres average rating, based on previously regularized movie mean rating
cvlg<-sapply(seq(1,20,1),function(x){lambdas(i=x,y=edx_train,z="genres")})
lg<-cvlg[1,which.min(cvlg[2,])]#col 1 val  with min of second column in cvlsg
gm<-edx_train%>%group_by(genres)%>%summarize(genremean=om+(sum(rating-om)/(n()+lg)))

#train set regularized year mean rating
cvlsy<-sapply(seq(2,20,2),function(x){lambdas(i=x,y=trainwithyears,z="year")})
ly<-cvlsy[1,which.min(cvlsy[2,])]#col 1 val  with min of second column in cvlsy
ym<-trainwithyears%>%group_by(year)%>%summarize(yearmean=om+(sum(rating-om)/(n()+ly)))
```

Rating Timestamp

Using the lubridate package to transform the timestamp into an easily interpretable date allows exploration of the impacts of various components of time on rating. Of all the ways dates and times could be interpreted, the strongest correlation between grouping variables and average rating was with year of rating and month of rating.

```
#slight correlations exist (about 0.5 and -.6) for the month and year of ratings
#though only over a narrow range
dates<-edx_train%>%select(timestamp,rating)%>%mutate(date=as_datetime(timestamp),ratyear=year(date),rat

ydate<-dates%>%group_by(ratyear)%>%summarize(rateyearmean=mean(rating))
ydate[1,2]=om #replace earliest year, only 4 ratings, with overall mean

mdate<-dates%>%group_by(ratmonth)%>%summarize(ratemonthmean=mean(rating))
```

Putting it all Together

With the now regularized averages calculated on the train set, a table can be built for analysis. The averages can be added to the train, test, and validation set so they can be used as predictors.

```
#prepare table
rtprep<-function(x){
  #separate year from name
  yl<-as.numeric(gsub("[^0-9]","",str_extract(x$title,"\\([0-9]{4}\\)")))
  x<-mutate(x, year=yl,date=as_datetime(timestamp),ratyear=year(date),ratmonth=month(date))

  #add regularized movie, user, and genre mean values, calculated on the train set only
  x<-x%>%left_join(um,by="userId")%>%
    left_join(mm,by="movieId")%>%
    left_join(gm,by="genres")%>%
    left_join(ym,by="year")%>%
    left_join(ydate,by="ratyear")%>%
```

```
    left_join(mdate,by="ratmonth")

  #fill overall mean rating in case where user, movie, or genre is unavailable
  replace_na(x$usermean,om)
  replace_na(x$moviemean,om)
  replace_na(x$genremean,om)
  replace_na(x$yearmean,om)
  replace_na(x$ratemonthmean,om)
  replace_na(x$rateyearmean,om)

  #simple prediction based on component columns
  x<-mutate(x,prediction=usermean+moviemean+genremean+yearmean+ratemonthmean+rateyearmean-(5*om))

  #separate genres and create a column for each with a boolean value
  genrelist<-sort(gsub("[^A-z]","",unique(unlist(strsplit(x$genres,"|",fixed=TRUE)))))
  for(i in 1:length(genrelist)) {
    x[[genrelist[i]]] <- with(x,grepl(genrelist[i],x$genres))
  }

  #clear any NAs and remove text columns
  x<-na.omit(select(x,-title,-genres,-timestamp))

}

prep_train<-rtprep(edx_train)
prep_test<-rtprep(edx_test)
prep_val<-rtprep(validation)
```

First, an evaluation on the train set. Each of the single measures as a predictor, followed by a combination of all six:

```
RMSE(prep_train$usermean,prep_train$rating)
```

```
## [1] 0.9692462
```

```
RMSE(prep_train$moviemean,prep_train$rating)
```

```
## [1] 0.9419169
```

```
RMSE(prep_train$genremean,prep_train$rating)
```

```
## [1] 1.017567
```

```
RMSE(prep_train$yearmean,prep_train$rating)
```

```
## [1] 1.048911
```

```
RMSE(prep_train$ratemonthmean,prep_train$rating)
```

```
## [1] 1.059646
```

```
RMSE(prep_train$rateyearmean,prep_train$rating)
```

```
## [1] 1.058238
```

```
RMSE(prep_train$prediction,prep_train$rating)
```

```
## [1] 0.9626404
```

Even with regularization, there is not a large improvement over the two first, best predictors.

```
RMSE(prep_train$usermean+prep_train$moviemean-om,prep_train$rating)
```

```
## [1] 0.8755654
```

Next, the test set.

```
RMSE(prep_test$usermean+prep_test$moviemean-om,prep_test$rating)
```

```
## [1] 0.8856028
```

Other Approaches

Train time and computational resources were the main chokepoints to exploring more elaborate models. A random forest model trained on a random sample of 100,000 lines from prep_train gave an RMSE of 0.89 on the test set, but took over 10 hours to train.

```
#random forest, final model
prep_train_finalmodel<-prep_train[sample(c(1:nrow(prep_train)), 100000, replace=FALSE),]
controlrffinal <- trainControl(method = "none")
mtryfinal<-data.frame(mtry=12)
train_rffinal <- train(rating ~ ., method="rf",
                  data = prep_train_finalmodel,
                  tuneGrid=mtryfinal,
                  trControl = controlrffinal)

#check on test set
predictions_rffinal<-predict(train_rffinal,prep_test)
testsetRMSE<-RMSE(predictions_rffinal,prep_test$rating)
testsetRMSE

#check on validation set
predictions_validation<-predict(train_rffinal,prep_val)
validationRMSE<-RMSE(predictions_validation,prep_val$rating)
validationRMSE
```

Other sampled models produced RMSE values over one. A simple linear model on the six predictors performed slightly better on the test set, with an RMSE of 0.879 on the test set and trained in only a few hours.

```
train_lm <- train(rating ~ usermean+moviemean+genremean+yearmean+rateyearmean+ratemonthmean, method="lm
                  data = prep_train)
predictions_lm<-predict(train_lm,prep_test)
lmRMSE<-RMSE(predictions_lm,prep_test$rating)
lmRMSE

predictions_lmval<-predict(train_lm,prep_val)
lmRMSEval<-RMSE(predictions_lmval,prep_val$rating)
lmRMSEval
```

The resulting linear model has the following coefficients. $prediction = 0.86340 * usermean + 0.89251 * moviemean - 0.00283 * genremean - 0.01099 * yearmonthmean + 0.07021 * rateyearmean - 0.36330 * ratemonthmean - 1.59650$

10,000 line samples used to train decision trees verified the possible importance of age of a film and genre (particularly if "Drama" was listed).

Results

The best available model for the validation set is...

```
RMSE(prep_test$usermean+prep_test$moviemean-om,prep_val$rating)
```

```
## Warning in pred - obs: longer object length is not a multiple of shorter object
## length
```

```
## [1] 1.261624
```

Conclusions

The largest impacts on rating clearly came from the identity of users and movies. It would have been interesting to explore a cohorting approach that more explicitly predicted ratings on the basis of other similar users.

All other characteristics of these observations seem, at best, to add fractional bits of information to these major predictors.

I would be very curious to see how low RMSE could be pushed with bagged or boosted algorithms able to train on the entire dataset. Implementations like SlopeOne could also be useful in predicting ratings for movies a given user has not seen.

As far as additional data, MovieLens now includes social features. A user's friends and their ratings are an obvious source of additional information. Additionally, aggregates of critical consensus like RottenTomatoes or Metacritic might be a helpful addition. There may be an individual correlation, positive or negative depending on the user, between their personal rating and critical consensus.

#Works Cited [1] F. Maxwell Harper and Joseph A. Konstan, 2015. The MovieLens Datasets: History and Context. ACM Trans. Interact. Intell. Syst. V, N, Article XXXX (2015), 20 pages.

[2] Rafael A. Irizarry 2020, Introduction to Data Science Data Analysis and Prediction Algorithms with R