# ALS+TD+KNN

Saier Gong (sg3772)

4/14/2020

### Step 1 Load Data and Train-test Split

```
library(dplyr)
```

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

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

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

```
library(tidyr)
library(ggplot2)
library(lubridate)
```

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

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

```
data <- read.csv("../data/ml-latest-small/ratings.csv") %>%
  mutate(Date=as.Date(as.POSIXct(timestamp,origin="1970-01-01",tz='UTC')))


source(file = "data_split(need modify).R")


#set.seed(0)
#test_idx <- sample(1:nrow(data), round(nrow(data)/5, 0))
#train_idx <- setdiff(1:nrow(data), test_idx)
set.seed(0)
split<-train_test_split(data = data,train_ratio = 0.8)
```

```
## [1] "Movies missing in training after split. Making Changes..."
```

```
data_train <- split$train
data_test <- split$test
```

**Step 2 Matrix Factorization**

```
###some preparation



RMSE<-function(rating,est_rating){
  a<-sqrt(mean((rating-est_rating)^2))
  return(a)
}
```

**Step 2.1 Alternative Least Square Algorithm with Temporal Dynamics Regulation Terms** In this step, we give initial values to parameter $b_u$, $b_i$ and $\alpha_u$, and user feature matrix p and movie feature matrix q.

The objective function is $min \sum(r_{ij} - (q_i^T p_u + \mu + b_i + b_u + \alpha_u dev_u(t))) + \sum \lambda(||q_i||^2 + ||p_u||^2 + b_i^2 + b_u^2 + \alpha_u^2)$. We then update these parameters ad matrix in each iteration and get a convergent result in RMSE.

```
##tu is the data frame of the mean date of rating of each user in 'data' dataset.
als.td<-function(f=10,lambda=0.1,max.iter=5,data,data_train,data_test){
  data<-data %>%
    arrange(userId,movieId)
  tu<-data %>%
  group_by(userId) %>%
  summarise(meanDate=round_date(mean(Date),'day'))

beta<-0.4
#f<-10
#max.iter<-3
#lambda<-0.5



train<-data_train %>%
  arrange(userId,movieId) %>%
  mutate(diff=as.numeric(Date-tu$meanDate[userId])) %>%
  mutate(dev=sign(diff)*(as.numeric( abs(diff)^beta)) )

mu<-mean(train$rating)

#train1<-train %>%
#  mutate(rating=rating-mu)

test<-data_test %>%
  arrange(userId,movieId) %>%
  mutate(diff=as.numeric(Date-tu$meanDate[userId])) %>%
```

```r
  mutate(dev=sign(diff)*(as.numeric(abs(diff)))^beta)

U<-length(unique(data$userId))
I<-length(unique(data$movieId))

#initialize q
guodu<-data %>%
  group_by(movieId) %>%
  summarise(row_1=mean(rating-mu))


#q<- rbind(guodu$row_1,matrix(data=runif((f-1)*I,min = -1,max = 1),nrow = f-1,ncol = I))
set.seed(0)
q<-matrix(data=runif(f*I,-1,1),nrow = f,ncol = I)
colnames(q)<-as.character(sort(unique(data$movieId)))#movie feature matrix

#initialize bu
bu<-rep(0,U)
names(bu)<-as.character(sort(unique(data$userId)))

#initialize alpha.u
alpha.u<-rep(0,U)
names(alpha.u)<-as.character(sort(unique(data$userId)))

#initialize bi
set.seed(0)
bi<-runif(I,-0.5,0.5)
names(bi)<-as.character(sort(unique(data$movieId)))


#set.seed(0)
p<-matrix(nrow = f,ncol = U)
colnames(p)<-as.character(sort(unique(train$userId)))#user feature matrix

train.rmse<-c()
test.rmse<-c()




for (l in 1:max.iter) {


       ####update p
  for (u in 1:U) {
    #the movie user u rated
    I.u<-train %>%
      filter(userId==u)

    q.I.u<-q[,as.character(I.u$movieId)] #the movie feature u has rated
    R.u<-I.u$rating

    A<-q.I.u %*% t(q.I.u)+lambda * nrow(I.u) *diag(f)
```

```r
    V<-q.I.u %*% (R.u-bu[as.character(I.u$userId)]-bi[as.character(I.u$movieId)]-mu-alpha.u[as.characte

    p[,u]<-solve(A,tol = 1e-50) %*% V


  }

    ###update bi
 # for (m in 1:I) {
#    U.m<-train %>%
#      filter(movieId==as.numeric(names(bi)[m]))
#
#
#   pred.R.m<-c()
#   for (j in 1:nrow(U.m)) {
#
#     pred.R.m[j]<-as.numeric(t(q[,m]) %*% p[,U.m$userId[j] ]    )
#
#
#   }
#
#   ###new bi
#   bi[m]<-(sum(U.m$rating)-sum(pred.R.m)-sum(bu)-nrow(U.m)*mu-sum(alpha.u[U.m$userId]*U.m$dev)  )/(nro
#
#
#
#  }




   ###update alpha.u

  for (u in 1:U) {
    I.u<-train %>%
      filter(userId==u)


    pred.R.u<-c()
    for (i in 1:nrow(I.u)) {
      pred.R.u[i]<- as.numeric(t(p[,u]) %*% q[,as.character(I.u$movieId[i])]) #+mu+ bu[as.character(u)]
        #bi[as.character(I.u$movieId[i])]+ alpha.u[as.character(u)] * I.u$dev[i]

    }


    ##new alpha.u
    alpha.u[u]<-(sum(I.u$rating*I.u$dev)-sum(bi[as.character(I.u$movieId[i])]*I.u$dev)-sum(mu*I.u$dev)
                          - sum(pred.R.u*I.u$dev))/(sum(I.u$dev^2)+lambda)

  }
```

```
###update bu
for (u in 1:U) {
  I.u<-train %>%
    filter(userId==u)


  pred.R.u<-c()
  #calculate predict rating of user u
  for (i in 1:nrow(I.u)) {
    pred.R.u[i]<- as.numeric(t(p[,u]) %*% q[,as.character(I.u$movieId[i])])

  }


  ##new bu
  bu[u]<-(sum(I.u$rating) - sum(pred.R.u)-sum(bi)-nrow(I.u)*mu-sum(alpha.u[u]*I.u$dev))/(nrow(I.u)+lam

}
```

```
###update q movie feature

for (m in 1:I) {
  U.m<-train %>%
    filter(movieId==as.numeric(names(bi)[m]))


  p.U.m<-p[,U.m$userId] #the user feature who has rated movie m
  R.m<-U.m$rating

  A<-p.U.m %*% t(p.U.m)+lambda * nrow(U.m) *diag(f)
  V<-p.U.m %*% as.matrix(R.m-bu[U.m$userId]-bi[as.character(U.m$movieId)]-mu-alpha.u[U.m$userId]*U.m$d

  q[,m]<-solve(A,tol = 1e-50) %*% V



}
```

```
  R_hat<-as.data.frame(t(p) %*% q)

  R_hat<-R_hat %>%
    mutate(userId=as.numeric(names(bu))) %>%
    pivot_longer(cols = names(bi)[1]:names(bi)[length(bi)],
                 names_to = "movieId",
                 values_to = "est2") %>%
    mutate(movieId=as.numeric(movieId))



  train.with.est<-train %>%
    mutate(est1=mu+bi[as.character(movieId)]+bu[userId]+alpha.u[userId]*dev) %>%
    left_join(R_hat) %>%
    mutate(est_rating=est1+est2)

  train.result<-RMSE(train$rating,train.with.est$est_rating)
  train.rmse[l]<-train.result



  test.with.est<-test %>%
    mutate(est1=mu+bi[as.character(movieId)]+bu[as.character(userId)]+alpha.u[as.character(userId)]*dev)
    left_join(R_hat) %>%
    mutate(est_rating=est1+est2) %>%
    filter(!is.na(est_rating))

  test.result<-RMSE(test.with.est$rating,test.with.est$est_rating)
  test.rmse[l]<-test.result

}#wai ceng da xun huan de kuo hao

result.tibble<-tibble(train_rmse=train.rmse,
                      test_rmse=test.rmse)

return(r=list(p=p,q=q,mu=mu,bu=bu,bi=bi,alpha.u=alpha.u,rmse=result.tibble))
}


#r0<-als.td(f=10,lambda = 0.1,max.iter = 5,data,data_train,data_test)
#save(r0,file = "example_in_als+td.RData")
```

We show an example of $f = 10, \lambda = 0.1, max.iter = 5$ of RMSE, both training and test.

```
load("example_in_als+td.RData")
print(r0$rmse)
```

```
## # A tibble: 5 x 2
##   train_rmse test_rmse
##        <dbl>     <dbl>
## 1      0.900     0.989
## 2      0.660     0.898
## 3      0.621     0.898
```

```
## 4      0.606     0.906
## 5      0.598     0.914
```

```r
source( 'cv_saier.R')
```

```r
####cross validation


f_list <- seq(10, 20, 10)
l_list <- seq(-2, -1, 1)
f_l <- expand.grid(f_list, l_list)
K=5

#train_summary<-matrix(0,nrow = 5,ncol = 4)
#test_summary<-matrix(0,nrow = 5,ncol = 4)
#for(i in 1:nrow(f_l)){
#    par <- paste("f = ", f_l[i,1], ", lambda = ", 10^f_l[i,2])
#    cat(par, "\n")
#    current_result <- cv.function(data_train=data_train, K = 5, f = f_l[i,1], lambda = 10^f_l[i,2])
#    train_summary[,i]<-current_result$mean_train_rmse
#    test_summary[,i]<-current_result$mean_test_rmse
#
#    #print(train_summary)
#    #print(test_summary)
#
#}

#print(train_summary)
#print(test_summary)
```

```r
#save(train_summary,file = "train_summary.RData")

#save(test_summary,file = "test_summary.RData")
load("train_summary.RData")
load("test_summary.RData")

#colnames(test_summary)<-c("test10_0.01","test20_0.01","test10_0.1","test20_0.1")

print(train_summary)
```

**Step 2.2 Parameter Tuning**

```
##      train10_0.01 train20_0.01 train10_0.1 train20_0.1
## [1,]    0.6938345    0.5790885   0.8786546   0.8238524
## [2,]    0.5590424    0.4345692   0.6277711   0.5762773
## [3,]    0.5203303    0.3829107   0.5915101   0.5297911
## [4,]    0.5053706    0.3583057   0.5763731   0.5090470
## [5,]    0.4999065    0.3446382   0.5688656   0.4984395
```

```r
print(test_summary)
```

```
##      test10_0.01 test20_0.01 test10_0.1 test20_0.1
## [1,]   0.8358058   0.7723412  0.9193069  0.8788701
## [2,]   0.7393705   0.6769988  0.7261995  0.6861949
## [3,]   0.7264820   0.6721865  0.7038050  0.6597834
## [4,]   0.7392118   0.6900889  0.6988539  0.6534687
## [5,]   0.7655292   0.7168115  0.6998979  0.6546065
```

```r
print(colMeans(train_summary))
```

```
## train10_0.01 train20_0.01  train10_0.1  train20_0.1
##    0.5556969    0.4199025    0.6486349    0.5874815
```

```r
print(colMeans(test_summary))
```

```
## test10_0.01 test20_0.01  test10_0.1  test20_0.1
##   0.7612799   0.7056854   0.7496126   0.7065847
```

So we choose $f = 20, \lambda = 0.01$.

#### Step 3 Postprocessing with KNN

```r
#r<-als.td(f=20,lambda=0.01,max.iter = 5,data = data,data_train = data_train,data_test = data_test)

#save(r,file = "optimize_rmse_and_parameters.RData")
load("optimize_rmse_and_parameters.RData")
print(r$rmse)
```

#### Step 3.1 Run the Model with Optimized Parameters

```
## # A tibble: 5 x 2
##   train_rmse test_rmse
##        <dbl>     <dbl>
## 1      0.629     0.768
## 2      0.477     0.652
## 3      0.427     0.638
## 4      0.403     0.643
## 5      0.390     0.656
```

```r
source("knn.R")

load("answer.RData")

print(answer$rmse)
```

#### Step 3.2 Postprocessing with KNN

```
## [1] 0.8734229
```