Alp 301: Stones2Milestones, Recommendation Systems 1

April 2021

Table of Contents


```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse)
pacman::p_load(here)
pacman::p_load(lmtest)
pacman::p_load(glue)
pacman::p_load(broom)
pacman::p_load(ri2)
pacman::p_load(margins)
pacman::p_load(glmnet)
pacman::p_load(kableExtra)
pacman::p_load(stargazer)
pacman::p_load(knitr)
pacman::p_load(doParallel)
pacman::p_load(corrplot)

rm(list = ls())
#setwd(here("/Users/jazon/Public/ALP301/alp301/projects/stones2milestones_recsys/data"))

story_info<- read_csv("data/all_story_obs.csv")
utility_mat<- read_csv("data/utils_mat_filtered.csv",col_types = cols(.default =
col_double()))
num_cols <- 2527

child_ids<-as.integer(utility_mat$child_id_code)
story_ids<-as.integer(colnames(utility_mat[,3:num_cols]))
utility_mat[is.na(utility_mat)]<-0.0
```

```
stories_with_text<-(story_ids %in% story_info$story_id_code)
utility_matrix<-utility_mat[,3:num_cols]
utility_matrix<-utility_matrix[,stories_with_text]
story_ids<-colnames(utility_matrix)
utility_matrix<-as.matrix(utility_matrix)

rm(utility_mat)  # this variable is no longer needed
```

In the last tutorial, we learned about basic recommendation algorithms, and applied them to Music and MovieLens datasets. This tutorial utilizes these techniques for S2M data.

We will use our utility matrix with 1087 stories and 11202 children to build item based and user based collaborative filtering models, and run dimensionality reduction algorithms and use them to make recommendations.

We will use the story_info dataset to analyze the recommendations made by our algorithms. The story_info dataframe includes various observables extracted from the story text. These include the following variables.

`level`: The difficulty level of the story

`totpage`: The number of pages

`wordcount`: The number of words

`has_geoarea, has_color, has_fruits, has_vegetables, has_animals, has_sports`: Binary variables indicating whether the story text includes any words related to geographic areas, colors, fruits, vegetables, animals, or sports.

`geoarea,color,fruits,vegetables,animals,sports`: If the story contains references to any of the above, these are the extracted matches from the story text for words in these categories.

Story similarity based recommendations

We'll build an item-based collaborative filtering model as we did in our previous tutorial on recommendation systems with music data.

First, calculate cosine similarity for stories by building the similarity matrix using user activity.

```
#Ignore NA's for now:
sim1<- t(utility_matrix) %*% (utility_matrix)
sim2<- (colSums(utility_matrix)) %*% t(colSums(utility_matrix))
similarity_matrix <- sim1 / sim2
```

Then, let's build our recommender function.

```
Itemitem_top_X_recommendations<-function(story_index,X){
  sim_vector<-similarity_matrix[-story_index,story_index] #From the similarity matrix, lets
get the vector corresponding to 'index of story'.
  index <- which(sim_vector >= sort(sim_vector, decreasing=T)[X], arr.ind=TRUE)
  return(index)
}
```

Let's make recommendations to someone who read story 100. First, let's have a look at what they read so far by examining the story characteristics.

```
#From index in the matrix
get_story_details<-function(story_index){
  story_id<-story_ids[story_index]
```

```
  story_info%>%filter(story_id_code==story_id)->story_characteristics
  return(story_characteristics)
}

#From 'story text file' id
get_story_details_2<-function(story_name){
  story_info%>%filter(story_id_code==story_name)->story_characteristics
  return(story_characteristics)
}

story_characteristics<-get_story_details(100)

story_characteristics

## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names     cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>     <lgl>
## 1           180       3       350     1     1        1 {'Bunty'} TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>
```

Let's now make a recommendation based on story 100, and look at its characteristics.

```
recommended_id<-Itemitem_top_X_recommendations(100,1)

get_story_details(recommended_id)

## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names            cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>            <lgl>
## 1          1710      16      1145     1     1        9 {'Chhole-poori'… TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>
```

Do the levels and the topics match? What do you think?

Let's transform this function to make recommendations to users with a known list of items. We use the similarity matrix to calculate a 'score' for each unknown item, which is the sum of the similarities with all of the given user's items.

```
Item_based_top_X_recommendations<-function(uservector,X){
  index_known<-which(uservector!=0) #Get the indices of stories known.
  index_unknown<-uservector==0
  if(length(index_known)>1){
    item_scores<-colSums(similarity_matrix[index_known,])
  }else{
    item_scores<-(similarity_matrix[index_known,])
  }
  names_unknown<-story_ids[index_unknown]
  index <- which(item_scores[index_unknown] >= sort(item_scores[index_unknown],
decreasing=T)[X], arr.ind=TRUE)
  return(names_unknown[index])
}

All_Item_Scores_IBCF<-function(uservector){
    index_known<-which(uservector!=0) #Get the indices of stories known.
  index_unknown<-uservector==0
```

```
  if(length(index_known)>1){
    item_scores<-colSums(similarity_matrix[index_known,])
  }else{
    item_scores<-(similarity_matrix[index_known,])
  }
  return(item_scores)
}

#Top five recommendations to user 100
recs<-Item_based_top_X_recommendations(utility_matrix[100,],5)

#What does user 100 read?
user_interacted<-names(utility_matrix[100,utility_matrix[100,]>0])

#Lets look at `user_interacted`
print(user_interacted)
```

```
##  [1] "21"   "40"   "41"   "125"  "163"  "199"  "246"  "272"  "302"  "306"
## [11] "324"  "394"  "424"  "498"  "509"  "550"  "572"  "609"  "618"  "626"
## [21] "649"  "690"  "776"  "855"  "861"  "917"  "918"  "940"  "946"  "974"
## [31] "992"  "995"  "1054" "1077" "1125" "1205" "1242" "1251" "1283" "1302"
## [41] "1367" "1493" "1533" "1540" "1570" "1710" "1721" "1746" "1916" "1921"
## [51] "1955" "1971"
```

```
get_story_details_2(user_interacted[1])
```

```
## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names           cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>           <lgl>
## 1            21      23       522     1     1        2 {'Fati', 'Mmm'} TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>
```

```
#Let's compare with story 1710, our first recommendation.

get_story_details_2(1710)
```

```
## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names            cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>            <lgl>
## 1          1710      16      1145     1     1        9 {'Chhole-poori'… TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>
```

Question

Plot the popularity of the overall items and the results of the (top 1, top 5) item-based recommendations for every user, using code from the basic rec sys tutorial that you completed. Is this method recommending more popular items than average?

User Similarity Based Recommendations:

Now let's make user-based recommendations. We will start by calculating the user similarity matrix.

```
#User similarity matrix:

sim1<- (utility_matrix) %*% t(utility_matrix)
sim2<- (rowSums(utility_matrix)) %*% t(rowSums(utility_matrix))
similarity_matrix_user <- sim1 / sim2
similarity_matrix_user[is.nan(similarity_matrix_user)]<-0

User_top_X_recommendations<-function(userid,X,ratings_matrix,similarity_matrix_user){
  #We need to remove items that they already know to from our recommendations.
  user_row<-ratings_matrix[userid,]
  known_stories<-user_row!=0
  unknown_stories<-user_row==0
  other_users_ratings<-ratings_matrix[-userid,]
  similarity_vector<-as.vector(similarity_matrix_user[userid,-userid])
  item_scores<- (similarity_vector %*% other_users_ratings)/sum(similarity_vector)
  names_unknown<-story_ids[unknown_stories]
  index <- which(item_scores[unknown_stories] >= sort(item_scores[unknown_stories],
decreasing=T)[X], arr.ind=TRUE)
  return(names_unknown[index])
}

All_Item_Scores_UBCF<-function(userid,ratings_matrix,similarity_matrix_user){
    user_row<-ratings_matrix[userid,]
  known_stories<-user_row!=0
  unknown_stories<-user_row==0
  other_users_ratings<-ratings_matrix[-userid,]
  similarity_vector<-as.vector(similarity_matrix_user[userid,-userid])
  item_scores<- (similarity_vector %*% other_users_ratings)/sum(similarity_vector)
  return(item_scores)
}
```

Let's test it on user 100 again.

```
#Top 5 recommendations to user 100
recs<-User_top_X_recommendations(100,5,utility_matrix,similarity_matrix_user)
print(recs)

## [1] "357"  "709"  "759"  "1294" "1389"

#Lets compare story `user_interacted` which is something they've read, vs our top user based
recommendation""
get_story_details_2(user_interacted[1])

## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names           cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>           <lgl>
## 1            21      23       522     1     1        2 {'Fati', 'Mmm'} TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>

get_story_details_2(recs[1])

## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr> <lgl>
## 1           357      10       188     1     1        0 set() FALSE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
```

```
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>
```

Question

Plot the popularity of the overall items and the results of the (top 1, top 5) user-based recommendations for every user. Is this method recommending more popular items than average?

Content Based Filtering

In the two approaches explained above, we use the utility matrix to calculate similarities. When calculating item-based similarities, we use rows of the utility matrix. In a way, we're using the users reading a story as 'features' of the story. When calculating user-based similarities, we used the stories each user read as the 'features' of the users. In both cases, we calculated similarities in these 'feature' spaces.

There is no reason why we should restrict ourselves to these two approaches. We can use user characteristics (age, grade, etc.), or item characteristics (topic of the story, length of the story, etc) as features and calculate similarities using these features. Here, as an example, we will use story characteristics to calculate similarities, instead of the utility matrix. The approach we take is exactly the same, but we use our new feature matrix instead of the utility matrix to calculate the cosine similarity matrix.

Let's create the matrix:

```
story_info%>%filter(story_id_code %in% story_ids)%>%select(one_of("totpage","wordcount",
          "has_geoarea","has_color","has_fruits",
          "has_vegetables","has_animals","has_sports",
          "story_id_code"))->story_numeric
story_numeric<-arrange(story_numeric,story_id_code)
story_chars_matrix<-as.matrix(story_numeric[,-1]) #Remove the ID variable


sim1<- (story_chars_matrix) %*% t(story_chars_matrix)
sim2<- (rowSums(story_chars_matrix)) %*% t(rowSums(story_chars_matrix))
similarity_matrix_story_info <- sim1 / sim2
```

Given the similarity matrix, we can make recommendations exactly as we did before!

```
Story_char_based_top_X_recommendations<-function(uservector,X){
  index_known<-which(uservector!=0) #Get the indices of stories known.
  index_unknown<-uservector==0
  if(length(index_known)>1){

item_scores<-colSums(similarity_matrix_story_info[index_known,])/ncol(similarity_matrix_story_
info[index_known,])
  }else{
    item_scores<-(similarity_matrix_story_info[index_known,])
  }
  names_unknown<-story_ids[index_unknown]
  index <- which(item_scores[index_unknown] >= sort(item_scores[index_unknown],
decreasing=T)[X], arr.ind=TRUE)
  return(names_unknown[index])
}

All_Item_Scores_Story_char_filter<-function(uservector){
    index_known<-which(uservector!=0) #Get the indices of stories known.
  index_unknown<-uservector==0
  if(length(index_known)>1){
```

```
item_scores<-(colSums(similarity_matrix_story_info[index_known,])/ncol(similarity_matrix_story
_info[index_known,]))
  }else{
    item_scores<-(similarity_matrix_story_info[index_known,])
  }
  return(item_scores)
}
```

Evaluating collaborative filtering models

$$Precision\ at\ N\ =\ Mean\left(\frac{Recommended\ and\ used}{N}\right)$$

$$Recall\ at\ N\ =\ Mean\left(\frac{Recommended\ and\ used}{Total\ items\ used\ by\ user}\right)$$

$$Root\ Mean\ Squared\ Error: \sqrt{Average((PredictedRating\ -\ ActualRating)^2)}$$

Let us now evaluate our algorithms. When calculating precision and recall, each user in the test set is taken as the current 'active user', and a random subset of their existing history is hidden. This means we will randomly remove part of the artists they listen to, and see how well our recommendations match with the held out data. There are various ways of doing this, and here we will use 'all but k'. For example, in "all but 1", if the user listened to 10 artists, we will use 9 of those to predict the 10th, or in "all but 5" we will use 5 of those to predict the remaining 5. Note that when evaluating performance using the 'all but k' approach, we can only use data points which have more than 'k' items. When calculating RMSE, we calculate the predicted score for every item, and calculate the squared difference between the predicted score and the actual ratings for every item and every user, and then rake the average.

```
#All but K approach and RMSE, testing function:
test_all_but_k_items<-function(k,ratings_matrix){
  precision_vector<-vector()
  recall_vector<-vector()
  rmse_vector<-vector()
  for (userid in 1:nrow(ratings_matrix)){
    user=ratings_matrix[userid,]
    index_of_known<-which(user!=0 )
    if(length(index_of_known)>k){
      takeout<-sample(1:length(index_of_known),size=k)
      stories_removed<-index_of_known[takeout]
      save_old<-user[stories_removed]
      user[stories_removed]<-0
      ratings_matrix[userid,]<-user
      recommended<-Item_based_top_X_recommendations(user,k)
      matched=length(intersect(story_ids[stories_removed],recommended))
      precision_vector<-append(precision_vector,matched/k)
      recall_vector<-append(recall_vector,(matched/(length(index_of_known))))
      user[stories_removed]<-save_old
      ratings_matrix[userid,]<-user
      itemscores<-All_Item_Scores_IBCF(ratings_matrix[userid,])
      rmse<- sqrt(mean((ratings_matrix[userid,]-itemscores)^2))
      rmse_vector<-append(rmse_vector,rmse)
    }
  }
  return(c("Precision"=mean(precision_vector),"Recall"=mean(recall_vector),
"RMSE"=mean(rmse_vector)))
}
#Performance of item based recommendations
test_all_but_k_items(1,utility_matrix)
```

```
##    Precision       Recall         RMSE
## 1.033252e-03 8.476711e-05 7.646425e-02

test_all_but_k_items(5,utility_matrix)

##   Precision      Recall        RMSE
## 0.006694664 0.002055146 0.090172742

test_all_but_k_items(10,utility_matrix)

##   Precision      Recall        RMSE
## 0.014961538 0.005508811 0.111280307

#This is not run by default,
#because it takes a long time.
#Delete "eval=FALSE" in rmarkdown
#and run it yourself.

#We will evaluate it in the first 50 users, to save time,
#because matrix operations take a long time to compute.
#Note that we need to re-calculate the similarity
#matrix for every user when we hide
#part of their history.
#Try running this for the full dataset on your own.
test_all_but_k_users<-function(k,ratings_matrix){
  precision_vector<-vector()
  recall_vector<-vector()
  rmse_vector<-vector()
  #Change the line below to run it on the entire dataset
  num_rows=min(50,nrow(ratings_matrix))
  for (userid in 1:num_rows){
    user=ratings_matrix[userid,]
    index_of_known<-which(user!=0)
    if(length(index_of_known)>k){
      takeout<-sample(1:length(index_of_known),size=k)
      stories_removed<-index_of_known[takeout]
      save_old<-user[stories_removed]
      ratings_matrix[userid,stories_removed]<-0
      sim1<- (ratings_matrix) %*% t(ratings_matrix)
      sim2<- (rowSums(ratings_matrix)) %*% t(rowSums(ratings_matrix))
      sim_matrix_user <- sim1 / sim2
      sim_matrix_user[is.nan(sim_matrix_user)]<-0
      recommended<-User_top_X_recommendations(userid,k,ratings_matrix,sim_matrix_user)
      matched=length(intersect(story_ids[stories_removed],recommended))
      precision_vector<-append(precision_vector,matched/k)
      recall_vector<-append(recall_vector,(matched/(length(index_of_known))))
      ratings_matrix[userid,stories_removed]<-save_old
      allscores<-All_Item_Scores_UBCF(userid,ratings_matrix,sim_matrix_user)
      rmse<-sqrt(mean((allscores-ratings_matrix[userid,])^2))
      rmse_vector<-append(rmse_vector,rmse)
    }
  }
  return(c("Precision"=mean(precision_vector),"Recall"=mean(recall_vector),
"RMSE"=mean(rmse_vector)))
}
#Performance of user based recommendations
test_all_but_k_users(1,utility_matrix)
test_all_but_k_users(5,utility_matrix)
test_all_but_k_users(10,utility_matrix)
```

```r
#All but K approach and RMSE, testing function:
test_all_but_k_storychars<-function(k,ratings_matrix){
  precision_vector<-vector()
  recall_vector<-vector()
  rmse_vector<-vector()
  for (userid in 1:nrow(ratings_matrix)){
    user=ratings_matrix[userid,]
    index_of_known<-which(user!=0)
    if(length(index_of_known)>k){
      takeout<-sample(1:length(index_of_known),size=k)
      stories_removed<-index_of_known[takeout]
      save_old<-user[stories_removed]
      user[stories_removed]<-0
      ratings_matrix[userid,]<-user
      recommended<-Story_char_based_top_X_recommendations(user,k)
      matched=length(intersect(story_ids[stories_removed],recommended))
      precision_vector<-append(precision_vector,matched/k)
      recall_vector<-append(recall_vector,(matched/(length(index_of_known))))
      user[stories_removed]<-save_old
      ratings_matrix[userid,]<-user
      itemscores<-All_Item_Scores_Story_char_filter(ratings_matrix[userid,])
      rmse<- sqrt(mean((ratings_matrix[userid,]-itemscores)^2))
      rmse_vector<-append(rmse_vector,rmse)
    }
  }
  return(c("Precision"=mean(precision_vector),"Recall"=mean(recall_vector),
"RMSE"=mean(rmse_vector)))
}
#Performance of content filtering
test_all_but_k_storychars(1,utility_matrix)

##   Precision      Recall        RMSE
## 0.052414052 0.006281388 0.077246698

test_all_but_k_storychars(5,utility_matrix)

##  Precision     Recall       RMSE
## 0.05578063 0.01987519 0.09115497

test_all_but_k_storychars(10,utility_matrix)

##  Precision     Recall       RMSE
## 0.05819231 0.02617565 0.11269396
```

Questions

Why do the testing functions give you three separate values for RMSE? Why are they increasing?

Summarize the activity of users: How many stories they interact with,open, and finish, using the examples from the data descriptives tutorial. Compare the performance of item based and user based collaborative filtering on those with +30 stories interacted, versus those with fewer stories interacted. Which method performs better on less active users?

Combining the utility matrix with the user_info dataset, compare the performance of collaborative filtering methods on older children (grade 4 or above) versus younger children. Do you see any patterns?

Matrix Factorization and Dimensionality Reduction

Let's create a lower dimensional representation of our utility matrix.

```
#Let's start with a 5-dimensional representation
d<-5
U<-svd(utility_matrix,nu=d,nv=d)$u
Vprime<-svd(utility_matrix,nu=d,nv=d)$v

#Top recommendations based on low dimensional representation:

SVD_top_X_recommendations<-function(U,Vprime,userid,X,ratings_matrix){
  user_vector<-U[userid,]
  scores<-U[userid,]%*%t(Vprime)
  user_row<-ratings_matrix[userid,]
  unknown_stories<-user_row==0
  names_unknown<-story_ids[unknown_stories]
  index <- which(scores[as.logical(unknown_stories)] >=
sort(scores[as.logical(unknown_stories)], decreasing=T)[X], arr.ind=TRUE)
  return(names_unknown[index])
}

SVD_item_scores<-function(U,Vprime,userid){
  user_vector<-U[userid,]
  scores<-U[userid,]%*%t(Vprime)
  return(scores)
}
```

Let us make recommendations to user 100 again and compare against a story they've read.

```
toprec<-SVD_top_X_recommendations(U,Vprime,100,1,utility_matrix)

#Lets compare the content of story `user_interacted` which is something they've read, against
our top SVD recommendation
get_story_details_2(user_interacted[1])
```

```
## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names           cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>           <lgl>
## 1            21      23       522     1     1        2 {'Fati', 'Mmm'} TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>

get_story_details_2(as.numeric(toprec))
```

```
## # A tibble: 1 x 22
##   story_id_code totpage wordcount   she    he n_people names            cardinal
##           <dbl>   <dbl>     <dbl> <dbl> <dbl>    <dbl> <chr>            <lgl>
## 1           709      13       931     1     1        4 {'honk', 'Honk'… TRUE
## # … with 14 more variables: has_geoarea <lgl>, geoarea <chr>,
## #   has_country <dbl>, country <chr>, has_color <lgl>, colors <chr>,
## #   has_fruits <lgl>, fruits <chr>, has_vegetables <lgl>, vegetables <chr>,
## #   has_animals <lgl>, animals <chr>, has_sports <lgl>, sports <chr>
```

Cross Validation for SVD

How do we choose how many dimensions to use? It's a good idea to do cross-validation, using precision at K as our metric.

```
#This is not run by default,
#because it takes a long time.
#Delete "eval=FALSE" in rmarkdown
#and run it yourself.
```

```r
# Parallelize computing to make things faster
cluster <- makeCluster(min(detectCores(logical = TRUE) - 1,8))
registerDoParallel(cluster)
clusterEvalQ(cluster, {})

folds <- 5
splitfolds <- sample(1:folds, nrow(utility_matrix), replace = TRUE)
candidate_d <- c(2,4,6,8,10,12,14,16,18,20) #Candidate dimensions
k<-5 #We will use precision at 5

# Export objects to the parallel sessions
clusterExport(cluster, c("utility_matrix", "splitfolds", "folds",
"candidate_d","SVD_top_X_recommendations","k","story_ids"))

system.time({
results <- foreach(j = 1:length(candidate_d), .combine = rbind) %dopar%{
   d<-candidate_d[j]
   results_cv <- matrix(0, nrow = 1, ncol = 4)
   colnames(results_cv) <- c("d","precision at 5","recall at 5","rmse")
   mean_precision_vec<-rep(0,folds)
   mean_recall_vec<-rep(0,folds)
   mean_rmse_vec<-rep(0,folds)
   for(i in 1:folds){
      train_set <- utility_matrix[splitfolds != i , ]
      valid_set <- utility_matrix[splitfolds == i, ]
      Vprime<-svd(train_set,nu=d,nv=d)$v
      precision_vector<-rep(0,nrow(valid_set))
      recall_vector<-rep(0,nrow(valid_set))
      rmse_vector<-rep(0,nrow(valid_set))
      for (userid in 1:nrow(valid_set)){
        user=valid_set[userid,]
        index_of_known<-which(user!=0)
        if(length(index_of_known)>k){
          takeout<-sample(1:length(index_of_known),size=k)
          stories_removed<-index_of_known[takeout]
          save_old<-user[stories_removed]
          user[stories_removed]<-0
          valid_set[userid,]<-user
          valid_u<-valid_set%*%Vprime
          recommended<-SVD_top_X_recommendations(valid_u,Vprime,userid,k,valid_set)
          matched=length(intersect(story_ids[stories_removed],recommended))
          precision_vector[userid]<-(matched/k)
          recall_vector[userid]<-(matched/(length(index_of_known)))
          user[stories_removed]<-save_old
          valid_set[userid,]<-user
          scores<-SVD_item_scores(valid_u,Vprime,userid)
          rmse<-sqrt(mean((scores-valid_set[userid,])^2))
          rmse_vector[userid]<-rmse
        }
      }
      mean_precision_vec[i] <- mean(precision_vector)
      mean_recall_vec[i]<-mean(recall_vector)
      mean_rmse_vec[i]<-mean(rmse_vector)
   }
   results_cv[1,]<-c(d,mean(mean_precision_vec),mean(mean_recall_vec),mean(mean_rmse_vec))
   return(results_cv)
}
})
stopCluster(cluster)
```

```
ggplot(data=data.frame(results),aes(x=d,y=rmse))+
  geom_line()+
  xlab("Number of dimensions in SVD")+
  ylab("RMSE")
```

Questions

How many dimensions would you prefer to use in the SVD method?

Compare the performance of SVD method against collaborative filtering, and also compare the popularity of the items recommended by the SVD method versus the average popularity of the stories.

How would you modify these methods to recommend unpopular stories that don't receive much activity?

How would you make recommendations to a user with no history of activity?

Inspect the output of the various recommendation methods, and analyze the diversity of recommendations as you did in `rec_sys_tutorial`. Is the algorithm recommending stories with varying features and popularity?