edX Capstone Project : Choose Your Own!

Kaggle : Hash Code Archive - Photo Slideshow Optimization

meaner

6/22/2020

1. Introduction

1.1 Executive Summary

This problem is presented at the Kaggle Competition web site by Google ('Playground' category, i.e., no prizes, no medals). It is about how to order the pictures in a slide show in the optimal sequence.

Please note following mistakes we did

We picked up our data set from Kaggle, because it is recommended in the lecture.
The problem is that we picked up a question from the completion category (but not a prize competition again, just to make sure), and we cannot share the code and the data privately (we noticed it today ;). The former is circumvented, because sharing code is not allowed privately, but is okay publicly. When the notebook is published on the Kaggle website, we can share the code with you as well. The notebook is running on the Kaggle at the moment, and as soon as it is ready, we will put it open there.

The second problem is hard to get around. The data set is for the 'competition use' only. Unless you would also join in the competition, and download the data personally, you do not have an access to the data. But I think it is too much to ask for the reviewers. So, my code would not run without the data. The only option I can think of is that I will give up 20 points of the score that comes with the code. My hope is there is still a chance to pass the course, if I am very lucky with other parts. I put all the output of the code in my GitHub repo, including the intermediate products and the submission file, in the effor to give you some proof that the code actually runs.

One more thing: the data science problem described in the following sections is not a predictive type, such as we have training and test data set, and construct some fitting functions using training data. The present question is an optimisation problem, and one of the variants of the traveling salesman problem. Here the distance, or score/penalty, to the next town is defined differently. Some machine learning technique is used, like clustering, but it is of an unsupervised learning, more toward the reinforcement learning where the program tries to maximise the reward. The description of the program looks different from those of supervised machine learning models such as MovieLens, and may appear confusing at the first glance, but this is why.

Otherwise the problem is fun and interesting in the same time. We do not need any domain knowledge, and the problem looks very simple at the beginning, but quickly becomes abstract and complicated. For those people who are about to finish the beginner's stage, this problem is very well recommended as the first grip to join in the Kaggle competition. The completion still runs next one month.

1.2 Problem description

1.2.1 Goal of the problem

We have a table data that contains photo IDs and hash tags in one line (no actual images). A picture can have multiple hash tags. We have 90000 pictures in the data set. We will use the hash tags to reorder the photos in the optimal sequence. The best transition of two sequential pictures that Google defines is with moderate connection. Two sequential pictures should not differ too much that the audience cannot follow, but have to differ enough that the audience would not get bored. This is to say, the tags of two sequential pictures must overlap to some extent, but should not match entirely. Quantitatively, the score of two sequential pictures are minimum of the following 3 numbers.

1. Number of tags that are present only in the first picture
2. Number of tags that are present in both first and second pictures
3. Number of tags that are present only in the second picture

The illustration of the scoring scheme is explained in a pdf file (problem_instructions.pdf) that comes with the data set. We will find the sequence of 90000 pictures that maximizes the total scores of the transitions of the neighboring slides.

1.2.2 Complication of paring vertical pictures

There are two types of pictures in the data set, horizontal pictures (=landscape format) and vertical pictures (portrait format). Vertical pictures cannot be on one slide alone, but has to be coupled with a second vertical picture so that they together fill the horizontal screen of a slide show. This vertical-vertical sequence is not counted in the scoring. Instead, these two pictures after being paired are counted as a single horizontal picture that has tags showing up in either of the two vertical pictures.

1.3 Data set

One table data (d_pet_pictures.txt) is provided for the competition. There is no additional data set other than that. In the very first line, the total number of pictures '90000' is written. From the second line to the end, the table data has 3 columns. One line corresponds to one picture and we have 90000 rows of them. No index or sequential IDs are given.

```r
d0 <- read_csv('../input/hashcode-photo-slideshow/d_pet_pictures.txt')
```

```
## Parsed with column specification:
## cols(
##   `90000` = col_character()
## )
```

```r
knitr::kable(head(d0, 3))
```

90000
H 16 tm2 tl2 td4 t6 t82 t52 t86 th5 t44 t47 ts2 tc5 tl5 tc4 tf1 tw3
V 15 tz2 t56 t42 t47 th5 tx4 tp6 t52 tm2 tw3 tw6 t85 t82 tn6 t87
H 4 t51 tc4 tb5 t61

Taking the first line as an example,

Column	What is means
H(/V)	tells if the picture is in H = landscape or V = portrait format.
16	how many tags this picture has.
tm2 t12...	tags. They are given as a one long string.

An example of submission file (sample_submission.txt) is also included to illustrate the format of the final product.

1.4 EDA

The following consideration refer to the notebook published by Tong Hui Kan, '441k in 11 min', in the Kaggle competition platform. The basic knowledge that we acquired from EDA are:

- In the table data we have
  - 60000 vertical photos and
  - 30000 horizontal photos.
- When we pair all the vertical photos and count the pair as a single horizontal photo, the total number of the photos available is 60000, 30000 horizontal photos and 30000 vertical pairs.
- The number of unique tags are 220.
- The minimum number of tags that a picture has is 1. The maximum number of tags that a picture has is 19.
- The minimum number of tags that a vertical picture has is 3.
- There are some correlations of tags, i.e., there are group of pictures that seem to have similar set of tags.

1.5 Analysis of the problem - How to get maximum score in one transition

The score is **minimum** of the 3 numbers above (1.2.1). However high one of the 3 numbers is, the score is reduced to the one that performs worst. Therefore the highest score is obtained when the 3 numbers above are equal. This means,

A) The tags of both pictures in connection are overlapped exactly by half.

B) The numbers of the tags of the two pictures are equal.

C) As a result, there are same numbers of tags in 3 sectors of the Venn diagram – in the left, in the intersect, and in the right.

D) The numbers of the tags of the two pictures are both even number.

The total number of tags in the table data is 902257. If we do not lose any points, the maximum score one can reach is the half of it, 451128 points. In reality the highest possible score is a less than that, since the pictures which have odd numbers of tags only half of them can contribute to the scores, but after floored to an integer. The maximum score one can attain is 443406 according to the notebook by Tong Hui Kan.

1.6 Plan / Strategy

The plan consists of two parts.

1. Get vertical pictures in pair
2. Arrange whole photons in a sequence to attain the maximum score

1.6.1 How to pair two vertical pictures.

The strategy here is to minimize the loss of the tags, but make all them counted for the score. We will rephrase the points **A)** to **D)** here again, but in a more intuitive way.

1. The best score in one transition is attained when the two pictures with equal number of tags are laid one after the other. For instance, if we have two 12-tags pictures in a sequence, the maximum possible score is 6 (=12/2). If one is 12 and the other is 4, the maximum possible score is 2 (=4/2). If one is 12 and the other is 11, the maximum possible score is 5 (= floor(11/2)). This means that we should sort out the pictures first in terms of the number of tags and start with one of the pictures that have the largest numbers of tags.

2. If we have more pictures in our reserve that have the same number of tags, the chances are high that we find a pair that attains the optimal score. This means that, when we pair two vertical photos, we will try to make the final numbers of the combined unique tags about equal for all pairs. For instance, if we have vertical pictures of 19, 19, 3, and 3-tags, it is best to couple 19+3, 19+3, instead of 19+19, 3+3. The former gives 22-22 sequence, and may hit 11 points, while the latter 38-6 sequence ends up 3 in the best case.

3. At the vertical-picture pairing, we should also aim at making an even number tags in the pair in order not to waste the tags. If we have pictures of 19, 18, 3, and 2-tags, for instance, the pairs must be 19+3, and 18+2. The maximum score of the latter is 10, but the 19+3 pair might contribute 11 points in the later arrangement. If we make 19+2, 18+3 pairs, the maximum scores of both are 10.

4. To increase the number of unique tags at the vertical pairing, one should pick two cards that do not have any overlap of tags. Overlapping tags do not contribute to increase the number of unique tags in the resultant horizontal picture.

We will first sort all the vertical cards according to the number of tags, and split the stack into two in the middle. The cards in the first stack has 19,18,. . . ,10 tags, and the second stack 10,. . . , 3 tags. We will start with one of the vertical cards that have maximum number of tags (=19), and look for a best match starting from the end of the second stack (=3).

In order to reduce the computation time, we will only look for the first k_1=32 cards, and pick the best match among them. If we hit the highest score (=no overlap of tags), we will cut the search there, and stop looking for further.

1.6.2 To Get maximum score in whole sequence

After paring two vertical photos, and count them as one, we have in total 60000 'horizontal' photos to play with. An obvious way to attain the highest score is to try out all the possible orders, and pick the one that gives the maximum score. The number of possible permutations is !60000, or 10ˆ260634. and this is too much a computation to finish. As is suggested in by Tong Hui Kan, we will pick one starting card, and check the score with the cards, but only of limited number of trials, say $k_2 = 128$.

1. The second cards (the possible partners) must be the ones that have the same number of tags. If such cards are not available, we will look for the cards with one size smaller number of tags.

2. If we limit ourselves to compare only first $k_2 = 128$ cards to see the matching. the number of computation is reduced to $< k \cdot \frac{n(n+1)}{2} = 10^{11}$, where $n = 60000$.

3. If the score reaches maximum, namely, the half the number of the tags of the starting card, we stop looking for a match.

4. In order to increase the chances of the early matching, we will first order the cards according to the correlation before starting the search (see section 2.5).

2. Method

2.1 Prepare libraries

We use R language to write a computer program. We start with importing libraries. In case they are not available, we install them.

```r
# install packages if necessary
if(!require(corrr))       install.packages("corrr")
if(!require(usedist))     install.packages("usedist")
# if(!require(Biomanager))  install.packages("BiocManager")
if(!require(ComplexHeatmap)) install("ComplexHeatmap")
if(!require(DataCombine)) install.packages("DataCombine")
if(!require(coop))       install.packages("coop")
if(!require(knitr))      install.packages("knittr")
if(!require(kableExtra)) install.packages("kableExtra")

# import library
library(tidyverse)
library(corrr)
library(stringr)
library(lubridate)
library(dplyr)
library(DataCombine)
library(factoextra)
library(dendextend)
library(cluster)
library(pheatmap)
library(usedist)
library(ComplexHeatmap)
library(DataCombine)
library(data.table)
library(coop)
library(knitr)
library(kableExtra)
```

First we clear all the variables on the memory. The file is on Kaggle notebook, under the directory ../input. The system time is recorded to know execution time of the code.

2.2 Read input file

```
#--------------------------------
# Clear memory / specify input file / record time of start
rm(list=ls())
in_file <- '../input/hashcode-photo-slideshow/d_pet_pictures.txt'
sys1 <- Sys.time()
```

When we test the code, we will do it with a smaller data with the first n lines of records. n=90000 is the full execution of the program. Some execution parameters, such as searching margins are specified here. Larger the margin, the computation time is longer.

```
# n <- 128   # preliminary execution
# n <- 1024
# n <- 2048  # preliminary
n <- 90000 # full version
k1 <- 32  # searching margin for vertial pairing
k2 <- 128 # searching margin for whole sequence
```

Here we read the data file in d1 and manipulate it. The original data are not separated by ','. We will separate the data in column wise, while keeping the tags in one long string. The index column X0 is inserted for the identification of the pictures.

```
d1 <- read_csv(in_file, skip=1, col_names=F)
d1 <- head(d1, n)
d1 <- d1 %>% mutate(H = str_extract(X1, '^[H|V]'),
                    X2=str_extract(X1, '^[H|V]\\s(\\d*)\\s')) %>%
  mutate(X2=str_replace(X2, '^[H|V] ', '')) %>%
  mutate(X2=str_replace(X2, '\\s', ''))  %>%
  mutate(X2=as.integer(X2))  %>%
  mutate(X1=str_extract(X1,'\\st.*$')) %>%
  mutate(X1=str_replace(X1,'^\\s', ''))  %>%
  mutate(X0=1:n) %>%
  subset(select=c(4,2,3,1))
knitr::kable(head(d1, 3))
```

X0	H	X2	X1
1	H	16	tm2 tl2 td4 t6 t82 t52 t86 th5 t44 t47 ts2 tc5 tl5 tc4 tf1 tw3
2	V	15	tz2 t56 t42 t47 th5 tx4 tp6 t52 tm2 tw3 tw6 t85 t82 tn6 t87
3	H	4	t51 tc4 tb5 t61

Some house keeping parameters are calculated below. maxs: number of maximum tags that a picture has. This is actually the target value at making a pair of vertical pictures, therefore +3 (=minimum number of tags that a vertical picture has) is added to
aim for 22-tags pairs. 'tags' are the array of unique tags, ntags is the number of unique tags, which is 220 in total.

```
#--------------------------------------------
# house keeping parameters
#--------------------------------------------
maxs <- max(d1$X2) + 3L
tags <- str_split(d1$X1, '\\s')
tags <- unique(unlist(tags))
tags <- tags[order(tags)]
ntags <- length(tags)
# write_csv(d1, 'b1x.txt')
```

2.3 Make a large table of tags

We thought through how to deal with the tags data. Tags are given as combinations of 2 to 3 characters, either digits or alphabets. They are all in lower cases, and the first character is 't' that probably means 'tag'. The example are shown above. There are similar-looking tags like 't6', 't06', 'tz', 'tz6', and they should be counted all independently.

Since we plan to calculate the correlation of pictures to build a clustering before round robin search, we made a big table with 220 columns of unique tags with integer entry 1 if the picture has that tag. Otherwise the column values are all 0. The total number of tags of an entry i is sum(a[i,]). The correlation of two entries (or sum of the intersect) is sum(a[i,] * b[j,]). The total number of tags in left join is sum(a[i,] - b[j,] > 0).

```
#============================================
# create a big table  => skip usually
#--------------------------------------------
d2 <- sapply(1:(nrow(d1)), function(i){
  y <- str_split(d1$X1[i], '\\s', simplify = T)
  s <- rep(0L, length(tags))
  for(j in 1:(d1$X2[i])){
    yx <- paste('^', y[j], '$', sep='')
    s <- s + ifelse(str_detect(tags, yx), 1L, 0L)
  }
  return(as.integer(s))
})
d2 <- transpose(data.frame(d2))
colnames(d2) <- tags
d2 <- as.tibble(d2)
```

Calculating d2 takes time. It is wise to store it in a text file so that one can restart a program there.

```r
write_csv(d2, 'b2.txt', append = F)
#============================================
# d2 <- read_csv('b2.txt', col_types=cols(.default = col_integer()))
d2 <- head(d2, n)
knitr::kable(head(d2, 3))
```

t01	t02	t03	t04	t05	t06	t07	t1	t11	t12	t13	t14	t15	t16	t17	t2	t21	t22	t23	t24	t25
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The data d2 consists only of the tags. Other record, such as picture ID, format of the pictures are added back as d3.

```r
d3 <- bind_cols(X0=d1$X0, X0V=0, H=d1$H, X2=d1$X2, d2)
knitr::kable(head(d3,3))
```

X0	X0V	H	X2	t01	t02	t03	t04	t05	t06	t07	t1	t11	t12	t13	t14	t15	t16	t17	t2	t21
1	0	H	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	V	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	H	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.4 Utility functions

We will define utility functions below. s_sc() takes data.frame where the sequence of slides are stored, and returns the total score, when total=T is specified (default setting). If total=F, s_sc() returns the score of individual transitions (a long list), cumulative score of it, and also actual and highest possible scores.

```r
#=============================================
# scores : acutal and max /
#          individual and cumulative
#=============================================
s_sc <- function(dx, total=T){
#############################################
  s1 <- 0
  s2 <- numeric()
  dd <- dx[,tags]

  nr1 <- nrow(dx) -1
  sc2 <-0
  sm2 <-0
  for (i in 1:nr1){

    x <- unlist(dd[i, ])
    y <- unlist(dd[i+1, ])

    j1 <- sum((x - y) == 1)
    j2 <- sum(x * y)
    j3 <- sum((y - x) == 0)

    sc1 <-  min(c(j1,j2,j3))
    sm1 <-  floor(sum(x)/2)
    sc2 <-  sc2 + sc1
    sm2 <-  sm2 + sm1

    s1  <- s1 +sc1
    s2   <- rbind(s2, c(i, sc1, sm1, sc2, sm2))
  }
 ifelse(total, return(s1), return(s2))
}
#-------------------------------------------
```

The function s_di() calculates the distance (=score, or point) of two pictures placed in sequence.

```
#=============================================
# distance of two slides
# x: y: one line of photos
s_di <- function(x,y){
#=============================================
  j1 <- sum((x - y) > 0)
  j2 <- sum(x * y)
  j3 <- sum((y - x) > 0)
  #  ntags/(min(c(j1,j2,j3)) +1)
  return(min(c(j1,j2,j3)))
}
#-------------------------------------------
```

2.5 Sort data in the order of correlation

At the pairing of vertical pictures, we thought over what are the 'good' pair of two vertical pictures are. As we discussed above, even number of unique tags and the lack of common tags are rather obvious. In addition to them, we would like the resultant horizontal photos as similar as possible. The assumption (to be confirmed later, though) is that, since the maximum number of tags a picture has is 19, while the number of the total unique tags used in the data set is 220, it would be more difficult to find a pair with similar set of tags, than to find a pair of totally different set of tags. This is still an assumption to be confirmed, since if one pair have exactly same set of tags, it also reduces the total score, but such case can be avoided at the calculation of distances. Therefore, instead of blindly picking a random two pictures, we ordered our reservoir of cards in the context of the matching tags. For instance, if we are trying to match the pictures of 19 tags to those of 3 tags, the pictures are first sorted within the stack of 19-tag cards, and within the stack of 3-tag cards, and the program picks one card from each stack to match. In this way, we can maximize the similarity of the resultant pairs than picking up random pairs from two stacks. The function v_cor() below takes care of this process.

```
#============================================
# Reorder/cluster pictures with teh stacks of
# same number of tags
v_cor <- function(d5a){
#-------------------------------------------
# order by correlation

  uq <- unique(d5a$X2)

  d5o <- d5a[F,]
  for (u in uq){
    # progress report # surpressed for the report
    # print('')
    # print(Sys.time())
    # print(paste('u:', u, sep='')  )
    # flush.console()
    d5i <- d5a %>% filter(X2==u)
    if (nrow(d5i)==1) {
      d5o <- bind_rows(d5o, d5i)
      next
    }
#    print('transpose...') # progress report # surpresed in report
#    flush.console()
    c1 <- transpose(d5i[,tags])
    colnames(c1) <- d5i$X0

#    print('correlate...')
#    flush.console()
    c2 <- tcrossprod(data.matrix(c1)) %>% as_cordf() %>% rearrange()

#    print('mutate...')
#    flush.console()
    c3 <- c2 %>% mutate(XR = rownames(c2)) %>%
      rename(X0=rowname) %>% select(X0, XR) %>%
      mutate(X0=as.integer(as.numeric(X0)), XR=as.integer(as.numeric(XR))) %>%
      arrange(X0)

    c4 <- left_join(d5i, c3, by='X0') %>% arrange(XR) %>% select(-XR)
    d5o <- bind_rows(d5o, c4)
  }
  return(d5o)
}
```

For the speed of the execution, tcrossproduct() from corrr library is used to calculate the correlation instead of original transpose t() and correlation cor() functions of R-base.

2.6 Vertical pairs

v_pa() below takes a data.frame made of sequence of pictures, and return a data.frame with the same format, but the vertical pictures are paired in some optimal way.

```
#============================================
# Pairng vertical pictures.
# In return data.fram, vertical pictures are
# counted as one horizontal picture
v_pa <- function(d3){
#-------------------------------------------
# First we sort out the vertical photos only
# The horizontal photos d5H is added back to the
# stack at the end of the function

  d5H <- d3 %>% filter(H=='H')
  d5V <- d3 %>% filter(H=='V')

  # Number of vertical photos
  n_d5 <- nrow(d5V)

  # All vertical photos are split into two stacks,
  # one (na) with more tags (ntags=19...10),
  # and the other(nb) with less tags (ntags=10...3)

  na <- floor(n_d5 / 2L)
  nb <- na
```

```r
# Sort the cards in the descending order of number of tags (X2).
# Correlate cards inside the stack
# Second ordering key (X0) is necessary so that d5a and d5b
# does not have an overlap

d5a <- d5V %>% arrange(desc(X2),desc(X0)) %>% head(na)
d5a <- v_cor(d5a)

# Sort the cards in the ascending order of number of tags (X2).
# Correlate cards inside the stack

d5b <- d5V %>% arrange(X2, X0) %>% head(nb)
d5b <- map_df(v_cor(map_df(d5b, rev)), rev)

# av is the first stack. Only tag part is extracted

av <- d5a[,tags]

print('')
print('initial preparation finished.')
flush.console()

#--------------------------------------
# Loop for searchign pairs
# b is a buffer to store the pair
# i : index for av
# j : index for bv

d5bi <- d5b
b <- numeric()
av0s <- 0

for (i in 1:nrow(d5a)) {

  # Second stack
  bv <- d5bi[,tags]

  # Staring minimu. We will try to make x1_min = 0

  x1_min <- maxs
  j_min <- 1

  # Look at only first k1 cards.
  # In case that the remaining cards are less than k1
  # stop loop there.

  nbv <- min( c(k1, nrow(d5bi)  ))

  # initial values
  j  <- 1
  j1 <- j

  # unlist makes the calculation between av and bv significantly faster
```

```r
    aa <- unlist(av[i,])
    bb <- unlist(bv[j,])

    # While looking for a match in bv,
    # if aa changes from even to odd, or vice versa
    # report it

    if (av0s != sum(aa)){

        while ((sum(aa) + sum(bb)) %%2 !=0 ){

        j <- j + 1
        bb <- unlist(bv[j,])

        if(j >= nrow(bv) ){
          j <- nrow(bv)
          break
        }
      }
      j1 <- j
    }
    nbv <- ifelse( (j1 + nbv) <= nrow(bv), nbv, nrow(bv) - j1)

    #------------------------------
    # Start looking for good bb
    while(j <= nbv) {

      # Candidate of match.
      bb <- unlist(bv[j,])

      # If the nubmer of unique tags in aa and bb
      # does not make up an even number,
      # just increment j (got to next candidate)
      # wihtout further processing.
      # Incement nbv as well not to hit the limit.

      while (sum(bb) %% 2 != (av0s %% 2) & j < (j1 + nbv)){
        j <- j + 1
        bb <- unlist(bv[j,])
        nbv <- ifelse((nbv +1 + j) <= nrow(bv), nbv+1, nbv)

      }
      # Now the retultant pair should have even number of
      # unique tags

      bb <- unlist(bv[j,])

      # Is there any common tags?
      x1 <- sum(aa * bb)

      if (x1 < x1_min){
        # If we find better match, replace the current choice
        x1_min <- x1
        j_min <- j
      }
      if (x1==0){
        # If we hit the best score, stop looking for match

        x1_min <- x1
        j_min <- j

        break
      }
      j <- j + 1

    } # j loop (bv) finished

    bb_min <- unlist(bv[j_min,])

    # Store the pair in the buffer
    b <- append(b, c(d5a$X0[i], d5bi$X0[j_min], x1_min, sum(aa)+ sum(bb_min)))

    # Show progress report # but surpressed for report
       print(paste('i:', i, '/', na, ' paired:', d5a$X0[i], '/', d5bi$X0[j_min], ' | ',
                    'j1:', j1, ' nbv:', nbv,' ',format(Sys.time(), "%H:%M:%S"),' | ',
                    sum(aa), '/', sum(bb_min), ' x1_min: ', x1_min, sep=''))
         flush.console()

    # prepare for the next round
    av0s <- sum(aa)

    # Remove the mahing pair from the stack
    d5bi <- d5bi[-j_min,]

  }
  # Shape the buffer b so that one can
  # easily see the pair
  b <- matrix(b, na, 4, byrow=T)

  #--------------------------------------------
  # Now we know ID pairs which vertical photo
  #  makes a good pair with other vertical photo.
  # Here actual reordering of pictures.

  for(i in 1:nrow(b)){

    # index in d5a and d5b
    # that mathces the IDs in b
    # b: photo index
    # iax: index in d5a and d5b

    ia <- which(d5a$X0==b[i,1])
    ib <- which(d5b$X0==b[i,2])

    ax <- unlist(d5a[ia,tags])
    bx <- unlist(d5b[ib,tags])

    cx <- ax + bx
    cx[which(cx >= 1)] <- 1

    # Store peir information in a new column XOV
    d5a[ia,]$XOV <- b[i,2]

    # Update the number of tags (X2)
    # as the sum of tags in aa and bb
    d5a[ia,tags] <- t(cx)
    d5a[ia,]$X2  <- sum(cx)

  }

  # Attach the marged data.set d5a
  # back to the list of the horizontal photos
  # that we separted at the beginning
  # of this function

  d3x <- bind_rows(d5H, d5a)
  return(d3x)
}
```

```
# Pair
d3x <- v_pa(d3)

# Store result in a text file
# so that we can restart teh program from here
write_csv(d3x, 'b3x.txt')
```

2.7 Order all pictures

The actual ordering of pictures starts here.

```
#============================================
# Greedy search
#--------------------------------------------
# How long did it take until here?
sys2 <- Sys.time()

#--------------------------------------------
# In case we resume from the point
# that the vertical pairs are created
d3x <- read_csv('b3x.txt',col_types=cols(.default = col_integer(), H = col_character()))

#--------------------------------------------
# Shuffle horizontal photos and vertical pairs

set.seed(1978, sample.kind = 'Rounding')
i_rows <- sample(nrow(d3x))
d3x <- d3x[i_rows,]

#--------------------------------------------
```

```r
# Sort pictures in the descending order of
# number of tags
d3x <- d3x %>% arrange(desc(X2))

#--------------------------------------------
# Cluster the photos so that
# we can find a mathching counterpart
# with in short margin of search.

d3x <- v_cor(d3x)

#--------------------------------------------
# Create placeholders

d3xi <- d3x
d4x <- d3x[F,]

#--------------------------------------------
# This is the starting picture
# We will start the picture
# that come up the top of the stack
# when we sort the cards according to
# the number of tags

ax  <- unlist(d3xi[1,tags])

# Remove it from the stack now
d3xi <- d3xi[-1,]

# Store it in submission buffer
d4x[1,] <- d3xi[1,]

#--------------------------------------------
# Initialize cummulative score
s1 <- 0
#--------------------------------------------
# Initialize cummulative maximum (optimal) score
s2 <- 0

#--------------------------------------------
# Here starts the search
# ii is the index of the second cards of
# transitions

k3 <- k2

for(ii in 2:nrow(d3xi)){

  # As it becomes progressively harder
  # when the number of tags becomes less,
  # we will extend the margin of search
  # for pictures that have less tags.
```

```r
if(sum(ax) <= 4 ){k3 <- k2 *4}
if(sum(ax) <= 10){k3 <- k2* 2}

# The search stops either when we hit
# we the limit k3, or the stadk of
# the card (= pictures) exhausted.

nr <- min(c(k3, nrow(d3xi)))  # look first 1/8

# Initial distance befor the search
# We will make a_dist as large as possible

a_dist <-0

# ix is the index of the second card

for(ix in 1:nr){
  a_distx <- s_di(ax, unlist(d3xi[ix,tags]))

  # If we find a better match, replace it.
  if(a_distx > a_dist){
    a_dist <- a_distx
    ix0 <- ix
  }

  # If we hit the best possible score
  # stop the search
  if(a_distx == floor(sum(ax)/2)) {
    a_dist <- a_distx
    ix0 <- ix
    break
  }
}


# Update the scores
# s1: actutal score
# s2: theoretical maximum

s1 <- s1 + a_dist
s2 <- s2 + floor(sum(ax)/2)

# The best match is stored in the buffer d4x
d4x <- bind_rows(d4x, d3xi[ix0,] )

# Set the second card of the transition found this round
# as the first card of the next round
ax <- d3xi[ix0,]
ax <- unlist(d3xi[ix0,tags])

# Progress report at every 100 trials.
# surpressed for report
  if(ii %% 100 == 0){
     print(paste('ii:', ii, '/', nrow(d3xi), ' score:', a_dist, '/',
                  d3xi[ix0,]$X2, ' | ',  s1, '/', s2, ' || ',
                format(Sys.time(), "%H:%M:%S"), sep=''))
    flush.console()
   }

  # Remove the card already taken
  # from the stack

  d3xi <- d3xi[-ix0,]

  # If there is no card left in the stack,
  # get out of the loop

  if (nrow(d3xi) <= 1 | any(is.na(d3xi)) ) {
    print('break...')
    flush.console()
    break
  }
}
# Store the result here, so that we can resume the program
# from here

write_csv(d4x, 'b4x.txt')
d4x <- read_csv('b4x.txt',col_types=cols(.default = col_integer(), H = col_character()))
#--------------------------
sys3 <- Sys.time()
knitr::kable(sys3 - sys2)
#============================================================
# Prepare submission file
c4 <- character()
c4 <- toString(nrow(d4x))
for (i in 1:nrow(d4x)){
  c <- ifelse(d4x[i,]$X0V==0, as.character(d4x[i,]$X0-1),
              paste(as.character(d4x[i,]$X0-1), as.character(d4x[i,]$X0V-1), sep=' '))
  c4 <- append(c4, c)
}
knitr::kable(head(c4,10))
write_csv(data.frame(c4), 'c4.txt', col_names=F)
#============================================================
# END
############################################################
```

3. Result

The final score is 411493, and thsi is about the middle of the competition rank as of 23 June 2020.

4. Summary / Future plan

The following is what we learned in the project, and the plan what to impletement in the code in the rest of the competition time.

1. In the process of paring vertical photos, the missed tags are less than 1% even with the small number of trial matching (k_1=32 frames) . There are still small number of even/odd-tag pairs, such as 12+9. We

do have a rule implemented to avoid such pairs, but because of the short range of the search margin, the execution of the rule is imperfect. We would like to set the rule more explicitly.

2. We implemented the correlation clustring to increase the uniformity of the tags after pairing the vertical photos. It is hard to see, however, how much such clustering actually helped to improve the final score. We would like to make the effect more visible in a quantitative manner.

3. We would like to increase the matching margin to $k_2 = 1024$ or even higher. We tentatively tried up to $k_2 = 8192$ on a laptop, and the score steadily improved to 430k. The problem is that this is a 'notebook only' competition, and the code has to run on a R-notebook platform at the Kaggle web site. The maximum connection time is 6 hours, and a session times out after 1 hour of idle time. We need to tune the size of the matching margin so that we can use the maximum computation time, but make sure that the calculation finishes in 6 hours.

4. At the very early stage of working on the problem, we tried splitting the data in small number of clusters to ease the computation. When the nubmers of the cards are smaller, it is easier to handle them. If we squeeze the size of the clusters from 90000 to 3000 cards, say, by splitting them into 32 clusters, we would lose maximum about 10×32 points at the connections between the clusters, which is totally bearable. In addition the clustering corresponds to splitting the slides first in large categories of different themes such as travels, pets, and families. That would intuitively make sense for the use case of the present problem. However, it took whole two days to sort 90000 photos in 32 categories by k-means clustering on our laptop, which is not feasible on the Kaggle notebook. We would like to revisit the idea again.

5. We will discuss here a bit of detail about an idea how to fix the bad connections. We refer this rearrangement treatment to the post '442k in 2 hours' by aDg4b on Kaggle. After finishing the round robin stage of k_2=128 cards, we already attain a score about 410k. This means that about 5 to 10% of losses here and there, but in the large part of the connections, maximum scores are already reached. In order to improve the score, we cut the whole sequence in pieces again at the bad connections, and reorder them by chunk so that the connections become better. Inside the chunks, the order of the pictures are already perfect. We have about 5000 chunks of perfect sequences, and will rearrange them to make progressively larger chunks. We will always couple two of them at a time, and if one chunk found a best matching neighbour, the pair is removed from the arena. After one round is over, the whole set of the sequence is cut again in pieces, but now in the smaller number of bigger chunks. The chunks will be processed in the same way again. We will repeat the process until we cannot build any larger chunks any more.

   The idea sounds working on the desk. However, the problem is that the score drops significantly (~10%) at the start of the rearrangement (we have tried preliminary implementation already). The reason is that a 'bad' connection was not perfect, but it does have some kind of optimal score after the round robin matching. Such 'moderately' good connections will be lost if we cut the sequences in chunks and rearrange them to attain the perfect connections in other part. We are still thinking how to best avoid such initial drop of the score.