Docebo Recommendation System

git clone https://github.com/lootag/docebo_assignment.git to get the code

Point 1 & 2

As for points 1 and 2, you can find them in the code (main.R). Run the script from the command line with 'Rscript path/to/main.R'.

Point 3

Since no information about the content is provided, the only way to engineer a recommendation system is to base recommendations on the contents that 'similar' users have consumed. The algorithm is very simple, and can be broken down into a few steps. I divide the dataset into a training set (earliest 20% of views for each user) and validation set (remaining views). I've created a function which takes the user Id, the training list and the list of all formal content as inputs, and outputs an unseen formal content. You can look at the code down below.

```
recommend <- function(user, byUser, formal)
{
  require(dplyr)
  require(DescTools)
  myUser <- byUser[[1]]
  sampleSize <- 50
  iterations <- 1:length(byUser)
  for(i in 1:length(byUser))
  {
    if(byUser[[i]]$idUser[1] == user)
    {
      myUser <- byUser[[i]]
      iterations <- iterations[iterations != i]
      break
    }
  }
  userFormal <- myUser$content[!grepl('CS', myUser$content, fixed = TRUE)]
  unseen <- setdiff(formal, userFormal)
  compare <- c()
  if(nrow(myUser) >= sampleSize)
  {
    compare <- sample(myUser$content, sampleSize)
  }
  else
  {
    compare <- myUser$content
  }
  results <- as.list(c())
  for( i in iterations)
```

```
  {
    if(length(compare) <= nrow(byUser[[i]]))
    {
      if(any(compare %in% byUser[[i]]$content))
      {
        results[[i]] <- intersect(byUser[[i]]$content, unseen)
      }
    }
  }
  results <- results[!(results == 'NULL')]
  result <- do.call(c, results)
  result <- DescTools::Mode(result)
  return(result[1])
}
```

As you can see, the routine starts by selecting the dataset which corresponds to the specified user. Then, it determines all the formal content which the user hasn't watched yet. Afterwards, it samples 50 videos that the user has already watched, and if the user has watched fewer than that it will just take all the content that the user has whatched. Then it checks which users have consumed the same content, and for every user that has it stores all the content that this user has consumed but our user of interest hasn't, creating a list of vectors. Finally it creates a single vector from this list, and computes its mode: this will be the recommendation that we make to the user. As for the metrics used to measure this recommender's performance, I've simply checked whether the recommended content was watched by the user in the validation set. This metric yields an accuracy of about 70%. In order to provide a control measure, I've built a dummy recommender, which provides a recommendation taking a random video from those that the user hasn't watched yet. You can find the dummy recommender's code down below.

```
dummyRecommend <- function(user, byUser, formal)
{
  require(dplyr)
  require(DescTools)
  for(i in 1:length(byUser))
  {
    if(byUser[[i]]$idUser[1] == user)
    {
      myUser <- byUser[[i]]
      break
    }
  }
  userFormal <- myUser$content[!grepl('CS', myUser$content, fixed = TRUE)]
  unseen <- setdiff(formal, userFormal)
  result <- sample(unseen, 1)
  return(result)
}
```

This dummy recommendation system yields an accuracy of about 10% with the exact same data, which means that the recommender I've created improves performance quite a lot compared to random recommendations.

Point 4

If some data regarding the content's nature was provided(e.g. topic of the video, length of the video, brief description of the video), we could build a classifier (random forest, naive bayes, neural network) based on that, and this would probably have a substatial impact on accuracy.

Point 5

The solution's computing time increases linearly with the number of views, the number of users and the available content.