Final Individual Assigment

Final Assigment

- Deadline: 18th March 23:59
- This individual assignment is the 50% of the final grade and replaces the exam
- There are two main goals:

1. To design a use case idea with a recommendation system.
2. To create a functional MVP in R based on that idea

Final Assignment: Use case

- The purpose of this assignment is to create a real application of a recsys in an industrial setting. Try to emphasise its use and how the technical decisions that you make are based on the use case.
- This application does not need to be financially sustainable or viable.
- Although revenue streams could be proposed and used to guide the recommendation system design.
- The proposal must have a value proposition in the industry you pick.

Final Assignment: MVP

- MVP have to run on teacher's computer. **Send everything required to run it**
- Code quality is not evaluated but the code should be understandable. Comments could be used to clarify the code.
- The code must have the algorithm itself and at least example running it (to get recommendations for one user).
- You can find some R examples at the end of this presentation.

Final Assignment

This assignment must have two deliverables: a document explaining the use case and your rationale for the algorithm and the R code with the algorithm and the running example.

Assignment as a proof of work

- This assignment is designed to help you to create a better CV. It is a good proof of work for your future employers, especially if they are looking for business-oriented data scientist or more management-intensive roles.
- Try to create a useful business case and show you knowledge of recommendation engines.
- Emphasize the connection between the technical side of your assignment and the business model. This is a rare and interesting skill in data science industry.
- Being the best of the intersection is better than being the best of one side.

Assignment as a proof of work (II)

- Although recommenderlab is not used in real industrial settings the MVP is a great way to show your idea is technically feasible.
- **After** you finish this assignment I recommend you to create other two deliverables: a blogpost-like text explaining your decisions guided by the use case and a second piece oriented to explain the technical details of your recsys.
- Show them in your CV, share them on social networks, give talks about them…
- This second technical piece could be different things: another blogpost, an interactive Shiny-app (https://shiny.rstudio.com/) or the source code of your app on an open repository like GitHub.com

Assignment as a proof of work (III)

- I can help you after I finish grading to publish this assignment. You can propose more complex solutions or work the presentation of your deliverables. Also I can give you some orientation to create an interactive version of your recommendation engine or other ideas tah you can have.
- Focus on the assignment and later you can improve it to publish it. But don't procrastinate it. It's useful. My former students used it with excellent results.

What is evaluated

- Applicability: The design must be executable in a real setting (assuming enough budget and data).
- Correctness: It must be based on an accurate knowledge of recommendation systems.
- Working MVP: MVP must run on other computers. So send everything is required to run it (data, R code, external files, …)
- Industry knowledge: understanding of the use case, how to solve it, its challenges and problems and the actual situation of the selected industry.

What is evaluated (II)

- Combination of algorithms: This recsys must implement at least one the techniques to combine recommendation systems that we studied (Hybrid Recommendation, "Over experience" recommendation, …)
- Algorithm design: The algorithm implemented should be oriented to solve the use case. Its design should face the problems of your business case. Try to bridge technical decisions and business decisions in your design.

What is not evaluated

- Financial sustainability of the business model
- Code quality. Although the code must be readable and understandable.
- Recsys accuracy and quality metrics

Tips

- Use the different criteria we studied (stability, explainability, …) and think what criteria are useful for your use case.
- Review again "Deconstructing Recommender Systems" article
- Think what problems your algorithm is going to solve and how can you solve in an algorithmical way.
- Research into the state of the industry and how others have solved this problem before. There are a lot of good ideas out there. Please, cite your external sources if you use them.

Tips

- Create a NEW Rstudio project for this assigment. It's important to avoid name collisions with other sessions. For more info: https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects
- Put all the files you use in this same folder.
- If you can't upload it to the campus due to size limits you can remove the data and replace it with an external link to download it. Please, double check the external resources are available.
- You have to load all the libraries and data files you need in your R code.

Before sending

- Remember I have to be able to run your code and get recommendations from your recsys
- Before sending the assignment make a "clean test" of your code:

1. Remove all elements from your R environment (broom icon on environment tab)
2. Restart your R Session (In Rstudio: Session > Restart R)
3. Run everything in the exact same order that is in your code.

Sometimes you use a variable that is in your environment from previous sessions (or other subject) and therefore it's not going to run in other computer.

This test detects if you rely on something in your environment that is not in your code.

Running examples

```r
# A global state is a set of variables shared across different functions to make your algorith
recc <- NA

trainMyAlgorithm <- function(data) {
  # We use <<- operator instead <- to save this in the global state
  recc <<- Recommender(data, method="POPULAR")
}

applyMyAlgorithm <- function(data) {
  # Note we use recc inside this function.
  pre <- predict(recc, data, n = 5)
  return(as(pre, "list"))
}
```

Running examples

```r
trainMyAlgorithm(MovieLense[1:500,])

applyMyAlgorithm(MovieLense[5,])
```

```
## $`5`
## [1] "Godfather, The (1972)"             "Pulp Fiction (1994)"
## [3] "Shawshank Redemption, The (1994)" "Schindler's List (1993)"
## [5] "Titanic (1997)"
```

Running examples

You can add more than one parameter for filters, hyperparameters…

```r
trainMyAlgorithm <- function(data, randomness=0.1) {
  recc <<- HybridRecommender(
    Recommender(data, method="POPULAR"),
    Recommender(data, method="RANDOM"),
    weights = c(1-randomness, randomness))
}

trainMyAlgorithm(MovieLense[1:500,])
applyMyAlgorithm(MovieLense[5,])
```

```
## $`5`
## [1] "Safe Passage (1994)"              "Great Day in Harlem, A (1994)"
## [3] "Boys, Les (1997)"                 "Santa with Muscles (1996)"
## [5] "Guantanamera (1994)"
```

Running examples

You can add parameters for the apply functions (e.g.: list length, filters, …)

```r
applyMyAlgorithm <- function(data, n=5) {
  pre <- predict(recc, data, n = n)
  return(as(pre, "list"))
}

applyMyAlgorithm(MovieLense[5,], n=20)
```

```
## $`5`
##  [1] "Safe Passage (1994)"
##  [2] "Aiqing wansui (1994)"
##  [3] "Boys, Les (1997)"
##  [4] "Santa with Muscles (1996)"
##  [5] "Tough and Deadly (1995)"
##  [6] "Great Day in Harlem, A (1994)"
##  [7] "Celestial Clockwork (1994)"
##  [8] "Star Kid (1997)"
##  [9] "Guantanamera (1994)"
## [10] "Quiet Room, The (1996)"
```

Running examples

Sometimes you need more than one dataset and more than one variables in global state:

```r
recc <- NA
targeted_movies <- NA

# This algorithm only recommends within one genre
trainMyAlgorithm <- function(data, metadata, genre) {
  targeted_movies <<- metadata[metadata[,genre] == 1, "title"]
  filtered_data <- data[,targeted_movies]
  recc <<- Recommender(filtered_data, method="POPULAR")
}

applyMyAlgorithm <- function(data) {
  filtered_data <- data[,targeted_movies]
  pre <- predict(recc, filtered_data, n = 5)
  return(as(pre, "list"))
}
```

Running examples

```
trainMyAlgorithm(MovieLense, MovieLenseMeta, genre="Children's")

applyMyAlgorithm(MovieLense[5,])
```

```
## $`5`
## [1] "Wizard of Oz, The (1939)"    "Babe (1995)"
## [3] "Beauty and the Beast (1991)" "Lion King, The (1994)"
## [5] "Fly Away Home (1996)"
```

Running examples

Although you can use a global state if you feel more comfortable with that there is other way to implement is that is cleaner and better for real applications.

Instead of using a global state we return an object with the state inside it.

Running examples

```r
# Same last algorithm but using internal state
trainMyAlgorithm <- function(data, metadata, genre) {
  targeted_movies <- metadata[metadata[,genre] == 1, "title"]
  filtered_data <- data[,targeted_movies]
  recc <- Recommender(filtered_data, method="POPULAR")

  # This is the difference in training function:
  # We return a list with recc and targeted_movies
  # Don't use <<- anymore because the variables required
  # by apply function are inside this list that is returned by train function
  return(list(recc=recc, targeted_movies=targeted_movies))
}
```

Running examples:

```
# Now apply function needs other argument: the trained
# algorithm created by trainMyAlgorithm
applyMyAlgorithm <- function(myAlg, data) {
  # To access to recc and targeted_movies we use
  # myAlg$xxxxx syntax
  filtered_data <- data[,myAlg$targeted_movies]
  pre <- predict(myAlg$recc, filtered_data, n = 5)
  return(as(pre, "list"))
}
```

Running examples:

```r
# Save the object (list) to one variable (my_trained_alg)
my_trained_alg <- trainMyAlgorithm(MovieLense, MovieLenseMeta, genre="Horror")

# Use this variable with apply
applyMyAlgorithm(my_trained_alg, MovieLense[5,])
```

```
## $`5`
## [1] "Scream (1996)"
## [2] "Devil's Advocate, The (1997)"
## [3] "Nosferatu (Nosferatu, eine Symphonie des Grauens) (1922)"
## [4] "Evil Dead II (1987)"
## [5] "Army of Darkness (1993)"
```

Resources to get data

- Kaggle datasets (look for recommender): https://www.kaggle.com/datasets?sortBy=relevance&group=public&search=recommender&page=1&pageSize=20&siz
- Classical research datasets: wikipedia, last.fm, book-crossings, movielense, ...: https://www.kdnuggets.com/2016/02/nine-datasets-investigating-recommender-systems.html
- Million Song: https://labrosa.ee.columbia.edu/millionsong/
- Network datasets: https://snap.stanford.edu/data/
- Ecommerce dataset: https://www.kaggle.com/retailrocket/ecommerce-dataset
- More links: https://github.com/caserec/Datasets-for-Recommneder-Systems/blob/master/README.md

Fake dataset

Sometimes is hard to find datasets that fits your business idea so for this assigment it is allowed to use fake datasets.

You can add new attributes to a real dataset or you can create one from nothing

Fake dataset

How to create a new column for metadata (continous data)

```
# Normal distribution:
MovieLenseMeta$newcolumn <- rnorm(nrow(MovieLenseMeta), mean=4, sd=1)

# You can truncate it to fit it into some limits (e.g.: 1-5 rating)
MovieLenseMeta$newcolumn <- ifelse(MovieLenseMeta$newcolumn > 5, 5, MovieLenseMeta$newcolumn)

MovieLenseMeta$newcolumn <- ifelse(MovieLenseMeta$newcolumn < 1, 1, MovieLenseMeta$newcolumn)
```

Fake dataset

How to create a new column for metadata (discrete data)

```
# 1000 numbers from 1 to 5
sample.int(5, 1000, replace = T)
```

```
##    [1] 4 3 1 2 4 1 5 2 4 1 4 1 5 1 5 2 1 2 4 2 3 2 5 5 3 4 2 3 4 3 4 2 3 3
##   [35] 1 2 5 4 1 3 4 4 3 4 1 1 4 3 5 1 1 3 1 2 2 5 5 5 4 5 5 1 1 4 2 2 5 1
##   [69] 4 4 5 3 3 5 4 3 4 2 4 1 1 4 5 3 1 2 5 5 1 2 5 1 5 2 1 4 5 4 2 5 2 1
##  [103] 3 5 2 5 2 2 2 4 4 2 3 2 3 5 1 5 3 3 1 5 3 5 1 2 5 3 5 3 5 5 5 2 1 1
##  [137] 1 2 2 2 5 3 3 3 3 3 1 4 5 2 2 4 2 1 2 3 5 1 4 1 5 4 2 4 1 2 2 3 2 5
##  [171] 5 5 2 2 5 5 3 2 3 4 5 5 1 5 3 4 4 1 1 2 4 5 5 4 3 3 2 5 4 1 2 3 2 3
##  [205] 3 4 5 1 1 3 3 1 5 1 5 1 1 1 3 2 3 2 3 2 1 1 1 1 4 4 4 5 3 3 5 3 4 5
##  [239] 4 3 2 3 4 1 4 4 4 2 5 2 2 5 5 5 4 2 3 2 4 4 4 2 4 4 2 4 1 2 1 4 3 2
##  [273] 3 5 1 5 5 1 1 4 5 5 5 5 1 4 1 5 1 4 4 3 2 3 2 5 4 4 3 5 1 2 3 2 4 1
##  [307] 5 2 1 4 4 2 3 2 5 2 4 3 3 2 1 4 3 3 5 5 1 3 3 4 5 4 5 5 5 3 5 5 1 3
##  [341] 5 2 5 2 3 4 2 5 5 5 2 1 2 3 5 5 4 4 4 5 2 4 2 4 5 3 4 4 1 3 4 5 4 2
##  [375] 5 2 1 2 3 4 4 5 4 1 4 3 4 1 1 1 3 3 5 4 3 4 2 5 4 1 2 1 4 4 3 4 4 5
##  [409] 4 2 1 2 1 5 4 2 4 4 2 3 4 2 3 3 4 5 3 3 4 5 2 4 3 4 3 2 2 4 4 3 5 3
##  [443] 5 1 2 1 5 5 1 2 5 2 1 1 4 4 2 4 5 3 1 5 3 4 3 3 3 2 5 2 1 2 1 5 3 2
##  [477] 1 5 5 3 2 5 3 2 3 2 5 5 4 2 4 3 5 3 2 5 1 3 5 1 5 2 1 4 4 5 5 5 2 2
```

Fake dataset

How to create a new preference matrix (binarised)

```
library("Matrix")
nRatings <- 1000
nItems <- 10000
nUsers <- 800

# Binary feedback
# Create a two-column dataset with user and item IDs
myMatrix <- data.frame(
  user=sample(nUsers, nRatings, replace=T),
  item=sample(nItems, nRatings,  replace=T)
)
# Convert it to binaryRatingMatrix
myMatrix <- as(myMatrix, "binaryRatingMatrix")
```

Fake dataset

How to create a new preference matrix (real)

```
library("Matrix")
nRatings <- 1000
nItems <- 10000
nUsers <- 800

# Binary feedback
# Create a two-column dataset with user and item IDs
myMatrix <- data.frame(
  user=sample(nUsers, nRatings, replace=T),
  item=sample(nItems, nRatings,  replace=T),
  rating=sample(5, nRatings, replace=T)
)
# Convert it to realRatingMatrix
myMatrix <- as(myMatrix, "realRatingMatrix")
```

Typical data structures for recommendation systems

1. Preference matrix (users=rows and columns=items)
2. Item table (like MovieLenseData): each row is an item. It contains properties for each one like genre. You can use to filter, weight them or create a CB system with it.
3. User table: each row is a user. It contains properties for each user like: GPS location, total consumption, name, favourite genre, … You can use it to personalize your recommender system

But if you need other data structure for your idea feel free to mock it or find it in the internet.