Tackling the Issues to Create a Recommender: What Are Left Unresolved?

Watanabe Takuya, Edirium K.K.

Combinatorial Approach to Machine Learning Seminar
20 September 2022, Kyushu University

Introduction

- in this presentation,
  - give a broad overview of the problems we face when we want to construct a recommender
    - by showing specific instances of the problems
    - some problems are, at least tentatively, solved, others not
  - do not talk about very details of each algorithm
  - do not talk about "system" related matters too much
    - nonetheless, to create an algorithm implementable within a system is vital

What is Recommender

- you can think of it as a sort of search engine
  - search engine: search/display what a user want to see
  - takes **implicit** queries as its input
    - view, purchase etc.
  - contrasts with "ordinary" search engines, e.g., Google Search
    - take explicit queries (usually a query string) as input

https://www.store-raycassin.jp/

Input and Output

- input: sequence of user action
  - interaction between user and item
    - viewed an **item**
    - clicked a recommended **item**
    - register an **item** to my favourites
    - put an **item** into cart
    - purchased an **item**
    - stored as (user x item x event type)
- output: sequence of the list of recommended items
  - sometimes needs to output without input event
    - e.g., first seen user on the top page

Two Approaches (1/2)

- content-based
  - select recommending items based on items' content (features, attributes)
  - query could be an item itself: item to item nearest neighbour search
  - sticking to the keyword search engine analogy, it roughly corresponds to pre-Google search engines
    - we give search keywords, and the engine returns the best match pages to the given keywords
    - simple word vector based comparison is performed between the query and pages

Two Approaches (2/2)

- behaviour-based
  - collect users' behaviour and try to draw "collective" insights from aggregated data
  - in its purest form, does not care the content at all, e.g., collaborative filtering
  - somewhat similar to PageRank's idea
    - not the page content, but the inter-page link structure is used to rank pages

Content-based Recommender

- item has various attributes
  - item name, description, price, category, colour, texture, size etc.
- highly domain dependent
  - golf course reservation, apparel, motorbike parts etc.
- traditional content-based recommenders rely on textual/tabular data
- item images are also relatively easy to exploit by deep learning

Behaviour-based Recommender

- two types of collaborative filtering
  - item-based
    - "people watched **this item** also watched these items"
    - offers item recommendations for an **item**
    - PageRank equivalent, in its impact, for the recommender industry, introduced by Amazon
  - user-based
    - "people watched similar items as **you** also watched these items"
    - offers item recommendations for an **user**

	item 1	item 2	item 3	item 4	item 5
user 1	x			x	
user 2			x		x
user 3	x		x	x	
user 4		x			x
user 5		x		x	

similar

General Framework

- typical development path
  1. formulate a problem
  2. devise an algorithm
  3. implement the algorithm
- considering various aspects of the **end** performance, i.e. performance realised by the implementation
  - problem should be stated in a form which leads to good performance in the end

Two Types of Performance (1/2)

- system-oriented performance
  - response time (latency)
  - batch processing rate (bandwidth)
  - memory consumption
- goal-oriented performance
  - CTR (click through rate)
  - CVR (conversion rate)
  - sales amount etc.

Two Types of Performance (2/2)

- more subtle factors
  - fit to the underlying computer architecture
  - ease of implementation
  - ease of analysis/interpretation, e.g.,
    - what part of this calculation contributes most to the overall latency?
    - why did this algorithm produce better results compared to that?

Good Algorithms (1/2)

- best/worst case analysis sometimes gives us a wrong impression
  - e.g., Quicksort
    - its worst case time complexity is as bad as bubble sort, i.e. $O(n^2)$
    - no one cares!
  - same applies to upper/lower bound analysis for goal-oriented performance
- what we want are:
  - algorithms **practically** work well in most cases

Good Algorithms (2/2)

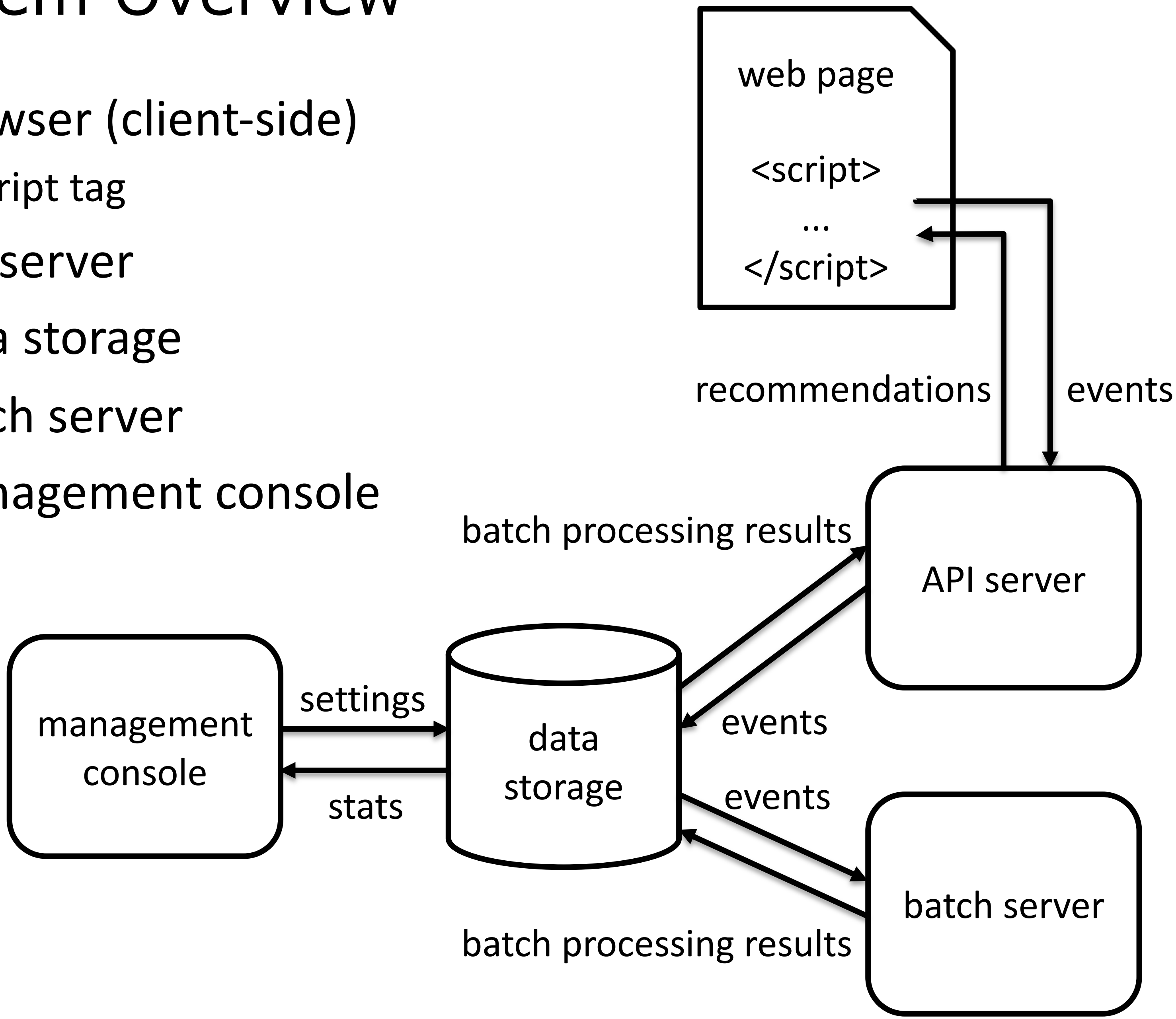
- what kind of algorithms are good?
  - fast
    - not necessarily corresponds to small time complexity
    - constant/lower order portion sometimes greatly affects in practice
  - small memory footprint
    - likewise, not necessarily corresponds to small space complexity
    - programming language's memory management implementation needs to be considered
  - produce good, hopefully superior, CTR/CVR etc.
- above objectives often contradicts each other
  - engineers seek to find a best balance
  - order of the above objectives vaguely represents their priority

Construct a Recommender

- at first, we need something working well for each user
  - create a content-based recommender, sensitive to user's view history
- then, we want to consider overall user behaviour
  - implement collaborative filtering (batch processing)
- how to combine the above two?
  - create an "ensemble" mechanism to mix recommendations
- how to compare performance of ensembles? how to deal with "atypical" cases?
  - create an A/B testing mechanism with fallback paths
- how to adjust each ensemble's share based on goal-oriented performance
  - implement a multi-armed bandit problem based optimiser
- also want image analysis?
  - yes!

System Overview

- browser (client-side)
  - script tag
- API server
- data storage
- batch server
- management console



Create a Content-based Recommender

- a simple content-based recommender
  - query: item of the currently viewing page
  - nearest neighbour search from the above item
  - recommend *k*-nearest neighbours
- problems:
  - results are fixed per item
  - computationally expensive (> 10M items)
    - can be batch processed

History Sensitive Content-based Recommendation

- take item view history into consideration
  - give diminishing weights to each item viewed previously
  - calculate the "centre" of viewed items
    - for simple numerical attribute, take weighted average
    - for string attribute, take weighted mixture of bag-of-words vectors
  - nearest neighbour search from the "centre"
  - recommend *k*-nearest neighbours
- problems:
  - treat each attribute equally
    - user would have different degrees of interest on each attribute
  - computationally expensive (> 10M items)
    - can **NOT** be batch processed

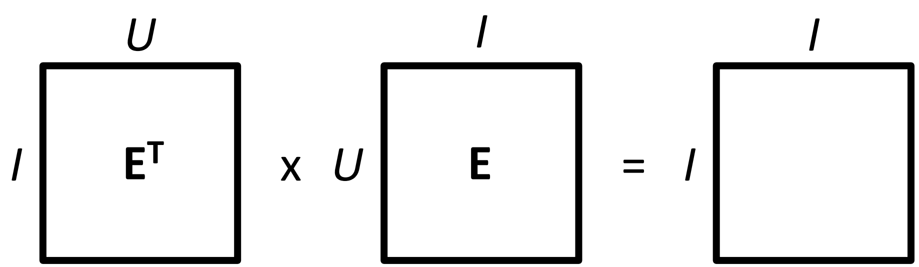
t-4	t-3	t-2	t-1	t
item 1	item 5	item 3	item 2	item 4
0.2	0.4	0.6	0.8	1.0

Attribute Weight Adjustment

- simple assumption:
  - attributes with low deviation: fairly interested
  - attributes with high deviation: not so interested
- in addition to the "centre" calculation,
  - calculate each attribute's deviation from item view history
  - take inverse of the deviation as attribute weight
  - nearest neighbour search from the "centre", with each attribute weighted accordingly
  - recommend *k*-nearest neighbours
- problems:
  - just heuristics, not statistically controlled nor supervised
  - computationally expensive (> 10M items)
    - can **NOT** be batch processed

Implement Collaborative Filtering

- based on user-item event matrix,
  - item-based
    - matrix multiplication to get item-item matrix
  - user-based
    - matrix multiplication to get user-user matrix
    - extract frequently seen items from nearest users
- problems:
  - results are relatively static (change not so much over time)
  - computationally expensive (> 10M items, > 10M users)
    - can be batch processed



Time-aware Collaborative Filtering

- users' (collective) interest may change periodically
  - e.g., what typically I want most in Monday and one in Sunday may be different
- take event occurring time into consideration
  - slice user-item event matrix to make per day event matrix
  - give each slice a temporal weight
  - take weighted sum of per day matrices to obtain global event matrix
  - do matrix calculation
- problems:
  - just heuristics, not statistically controlled nor supervised
  - computationally expensive (> 10M items, > 10M users)
    - can be batch processed

Algorithm Selection

- now we have a content-based recommender and collaborative filtering
  - how do we choose/combine algorithms
- manually select an algorithm for each recommendation area?
  - this is what traditional recommenders offer
- problems:
  - we have two probably good algorithms, but cannot use both at the same time
  - cannot compare algorithms' goal-oriented performance

Rank-based Ensemble (1/2)

- combine multiple recommenders' output by their ranking
  - give each rank a weight (inverse of rank)
  - take sum of each item's weight
  - sort items keyed by the above weight sum

original rankings

rank	1	2	3	4	5
weight	1.0	0.5	0.33	0.25	0.2
output 1	item 2	item 3	item 5	item 1	item 4
output 2	item 3	item 1	item 4	item 5	item 2

combined ranking

rank	1	2	3	4	5
weight	1.5	1.2	0.75	0.58	0.53
output	item 3	item 2	item 1	item 5	item 4

Rank-based Ensemble (2/2)

- why rank based? why not based on scores?
  - each algorithm has different internal scoring mechanism, so it is difficult to properly normalise scores
  - rank is more robust, in my opinion (not observation)
- problems:
  - just heuristics, not statistically controlled nor supervised
  - cannot compare algorithms' goal-oriented performance

Algorithm Hierarchy

cb/cf ensemble

↓

ranking

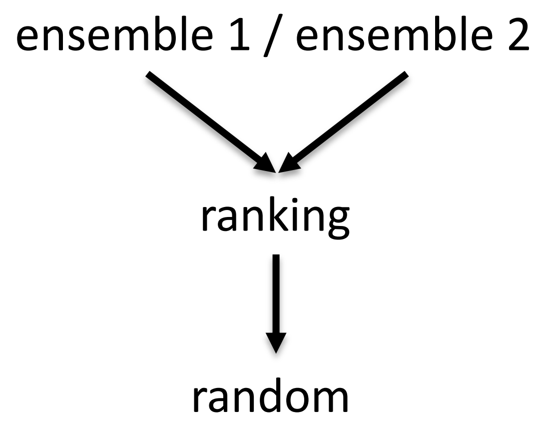
↓

random

- sometimes there is no clue as to who the user is
  - no previous item view history, viewing top page
  - what to recommend?
    - both content-based and collaborative filtering do not work
- need a fallback algorithm
  - view/purchase ranking would be appropriate
- but if there is no ranking under specific circumstances, e.g., for a specific category?
  - randomly show something
    - better than nothing, probably
- problems:
  - algorithm performance may be path dependent
    - e.g., ranking performance padding after other algorithms and pure ranking performance would be different
  - latency prediction much more difficult

A/B testing with Algorithm Hierarchy

- want to compare performance of different algorithms
  - under equal conditions as far as possible
- specify A/B testing groupings within algorithm hierarchy
  - place multiple algorithms per layer
  - assign an algorithm to each session according to the specified ratio
- problems:
  - need manual evaluation/tuning as a next step
  - need to split data (session)
    - may be too sparse
  - latency prediction much more difficult



Optimising Algorithm Share (1/2)

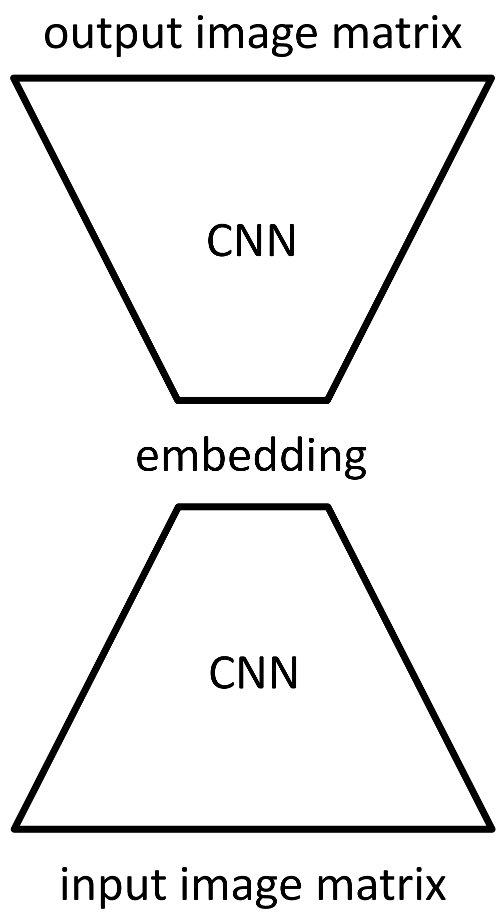
- adjusting algorithm share can be seen as a multi-armed bandit problem
  - want to reliably measure each algorithm's goal-oriented performance (exploration)
  - want to achieve good overall goal-oriented performance (exploitation)
  - have to be done online

Optimising Algorithm Share (2/2)

- Thompson sampling
  - reward is 1/0 (recommendation is clicked or not) ~ Bernoulli distribution
  - want to estimate probability distribution of expected reward ~ beta distribution
  - sample from *beta*(number of 1s, number of 0s) for each algorithm, choose the best one
    - batch sampling is more efficient
- problems:
  - there would be interactions between algorithms/ensembles
  - algorithm hierarchy itself should be subject of optimisation
    - need a meta-heuristics?

Image Analysis

- want to choose items with similar images
- very standard/straightforward pipeline
  - autoencoder based embedding extraction
  - nearest neighbour search on embedding space
  - return *k*-nearest neighbours
- problems:
  - items have multiple images
  - computationally expensive (> 10M items)
    - can be batch processed



Applying Rank-based Ensemble Technique to Images

- combine multiple nearest neighbour search results
  - calculate *k*-nearest neighbour images for each image
  - make a list of items to which those *k*-nearest images belong for each image
  - combine item lists in a rank-based ensemble way
- problems:
  - not particularly good for items which have many images
  - computationally expensive (> 10M items)
    - can be batch processed

Conclusion

- lot of heuristics, although we believe they perform generally well, statistical regulation/modelling may better be incorporated
- extensive use of nearest neighbour search
  - batch nearest neighbour search can be processed efficiently in vectorised (matrix) form
  - online nearest neighbour search under latency constraint (< 100 ms) is a **big** problem
    - huge neural networks often exhibit memory-based characteristics, could be substitutes for nearest neighbour search
    - essentially, transferring online computation cost of the nearest neighbour search to batch computation cost of model training