A fast and agile item-item recommender : design and implementation

Anton Antonov

Consultant at Synergis

Talk at Wolfram Technology Conference 2011

(Link to a CDF file with the presentation)

Abstract

In this talk are presented the design and implementation of an Item-Item Recommender (IIR) based on linear algebra operations. The presented IIR has great performance and scalability properties. Design and algorithmic approaches are discussed for recommendation proofs, tuning, and diversification. Recommendations of movies, houses, and points-of-interest are going to be demonstrated using a common user interface.

The algorithms discussed are from the fields of sparse matrix linear algebra, collaborative filtering, natural language processing, principal component analysis, association rule learning, and outlier detection.

Structure of the talk

- Demonstration of several recommenders (movies, houses, geo-locations)
- Some philosophy: why Item-Item recommenders should be preferred
- Some theory: using linear algebra algorithms
- Diversification of the recommendations
- Extensions

Who am I

M.Sc. in Mathematics (General algebra) (1997)

M.Sc. in Computer Science (Data bases) (1997)

Ph.D. in Applied Mathematics (Large scale air-pollution simulations) (2001)

Kernel Developer at Wolfram Research, Inc. (2001-2008)

Principal Engineer at Sezmi Corporation (2008-2011)

Consultant (mathematical modeling and algorithms) at Synergis (now)

A movie & TV show recommender

Movie and TV show recommendations [1]

Nearly 6000 movies and TV shows, characterized with actors, directors, writers, genres, and other meta-data.

A house recommender

House recommendations [2]

General description

- Nearly 8000 houses over 60 diffrent neighborhoods.
- Neighborhoods are characterzied with occupations breakdown and location.
- Houses are of different types (house, town-house, apartment).
- Houses are characterzied with construction (number of floors, number of bathrooms)
- and additions (garage, basement, pool, attic).

The circular city

The center of the city is the most interesting place, houses close to it are more expensive.

The neighborhoods (zones) are separated by roads.

The zones on the out-most circle are industrial zones – people drive to them on daily basis.

Interface

A points-of-interest recommender

Points of interest recommendations[3] (for Champaign-Urbana)

Nearly 4000 different points-of-interest in the Champaign-Urbana area.

All points of interest

Screenshot of the recommender

General recommender design

This is the high-level design of the item-item recommender.

General interface design

Using linear algebra algorithms to compute user profile and recommendations from the user history.

Using Model-View-Controllers (MVC's) to tweak the outcomes.

(MVC is also known as the Observer Design Pattern.)

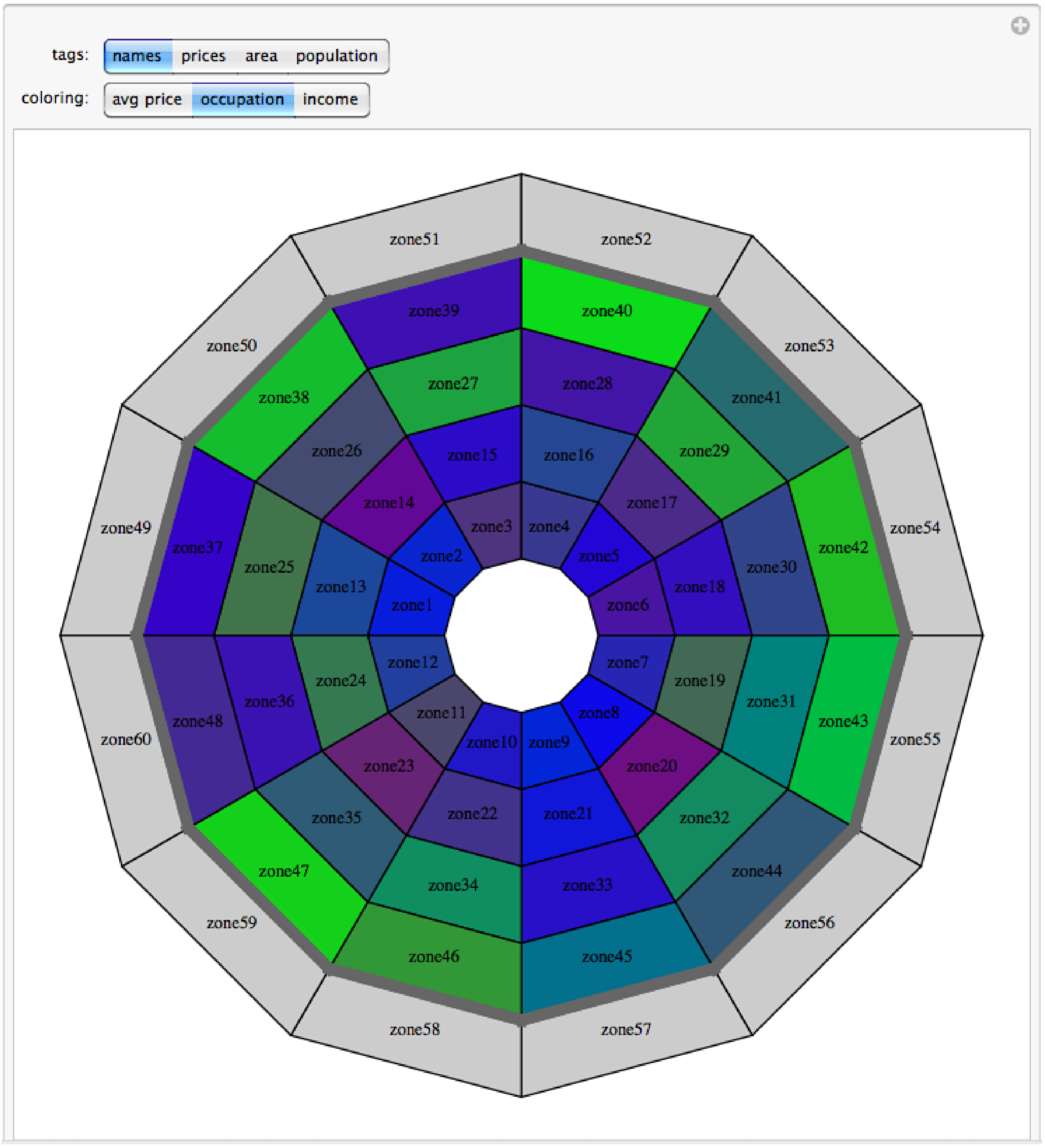
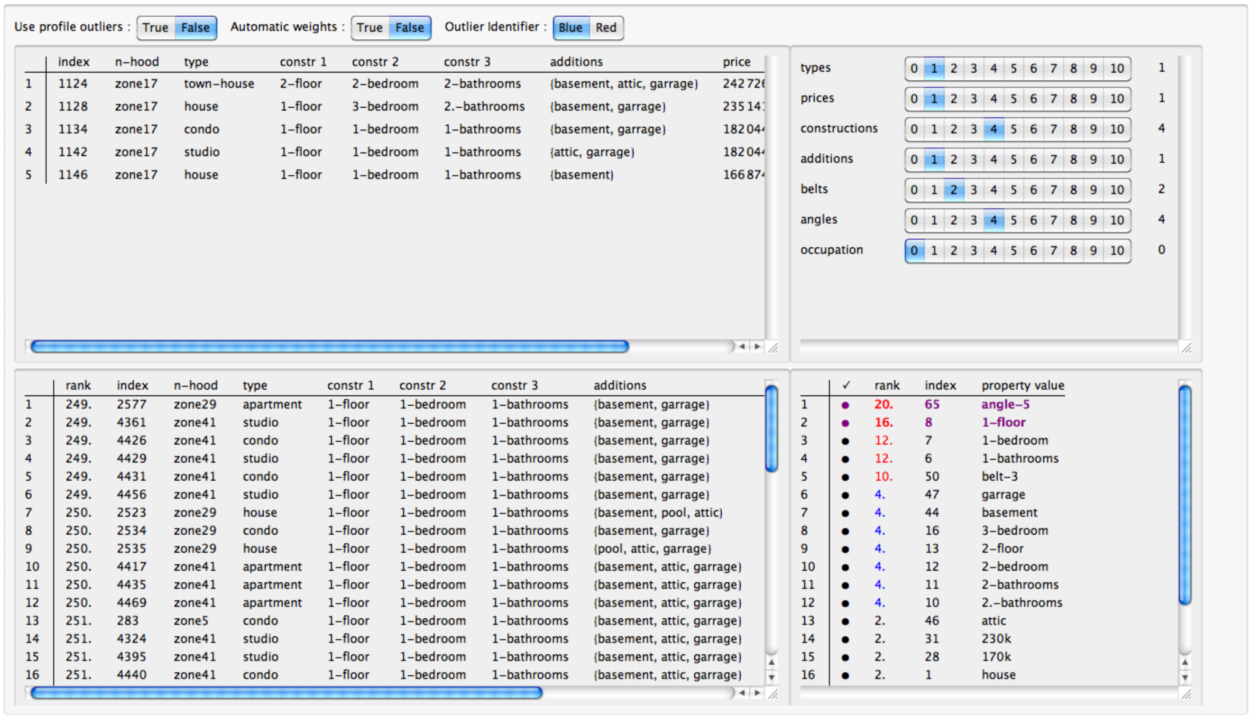

tags:
names
prices
area
population
coloring:
avg price
occupation
income
zone1
zone2
zone3
zone4
zone5
zone6
zone7
zone8
zone9
zone10
zone11
zone12
zone13
zone14
zone15
zone16
zone17
zone18
zone19
zone20
zone21
zone22
zone23
zone24
zone25
zone26
zone27
zone28
zone29
zone30
zone31
zone32
zone33
zone34
zone35
zone36
zone37
zone38
zone39
zone40
zone41
zone42
zone43
zone44
zone45
zone46
zone47
zone48
zone49
zone50
zone51
zone52
zone53
zone54
zone55
zone56
zone57
zone58
zone59
zone60

Figure 1:



Use profile outliers : True [✓ False] Automatic weights : True [✓ False] Outlier Identifier : [✓ Blue] Red

	index	n-hood	type	constr 1	constr 2	constr 3	additions	price
1	1124	zone17	town-house	2-floor	2-bedroom	2-bathrooms	{basement, attic, garrage}	242 72
2	1128	zone17	house	1-floor	3-bedroom	2.-bathrooms	{basement, garrage}	235 14
3	1134	zone17	condo	1-floor	1-bedroom	1-bathrooms	{basement, garrage}	182 04
4	1142	zone17	studio	1-floor	1-bedroom	1-bathrooms	{attic, garrage}	182 04
5	1146	zone17	house	1-floor	1-bedroom	1-bathrooms	{basement}	166 87

types	0 [✓ 1] 2 3 4 5 6 7 8 9 10	1
prices	0 [✓ 1] 2 3 4 5 6 7 8 9 10	1
constructions	0 1 2 3 [✓ 4] 5 6 7 8 9 10	4
additions	0 [✓ 1] 2 3 4 5 6 7 8 9 10	1
belts	0 1 [✓ 2] 3 4 5 6 7 8 9 10	2
angles	0 1 2 3 [✓ 4] 5 6 7 8 9 10	4
occupation	[✓ 0] 1 2 3 4 5 6 7 8 9 10	0

	rank	index	n-hood	type	constr 1	constr 2	constr 3	additions
1	249.	2577	zone29	apartment	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
2	249.	4361	zone41	studio	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
3	249.	4426	zone41	condo	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
4	249.	4429	zone41	studio	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
5	249.	4431	zone41	condo	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
6	249.	4456	zone41	studio	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
7	250.	2523	zone29	house	1-floor	1-bedroom	1-bathrooms	{basement, pool, attic}
8	250.	2534	zone29	condo	1-floor	1-bedroom	1-bathrooms	{basement, garrage}
9	250.	2535	zone29	house	1-floor	1-bedroom	1-bathrooms	{pool, attic, garrage}
10	250.	4417	zone41	apartment	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}
11	250.	4435	zone41	apartment	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}
12	250.	4469	zone41	apartment	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}
13	251.	283	zone5	condo	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}
14	251.	4324	zone41	studio	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}
15	251.	4395	zone41	studio	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}
16	251.	4440	zone41	condo	1-floor	1-bedroom	1-bathrooms	{basement, attic, garrage}

	✓	rank	index	property value
1	●	20.	65	angle-5
2	●	16.	8	1-floor
3	●	12.	7	1-bedroom
4	●	12.	6	1-bathrooms
5	●	10.	50	belt-3
6	●	4.	47	garrage
7	●	4.	44	basement
8	●	4.	16	3-bedroom
9	●	4.	13	2-floor
10	●	4.	12	2-bedroom
11	●	4.	11	2-bathrooms
12	●	4.	10	2.-bathrooms
13	●	2.	46	attic
14	●	2.	31	230k
15	●	2.	28	170k
16	●	2.	1	house

Figure 2:

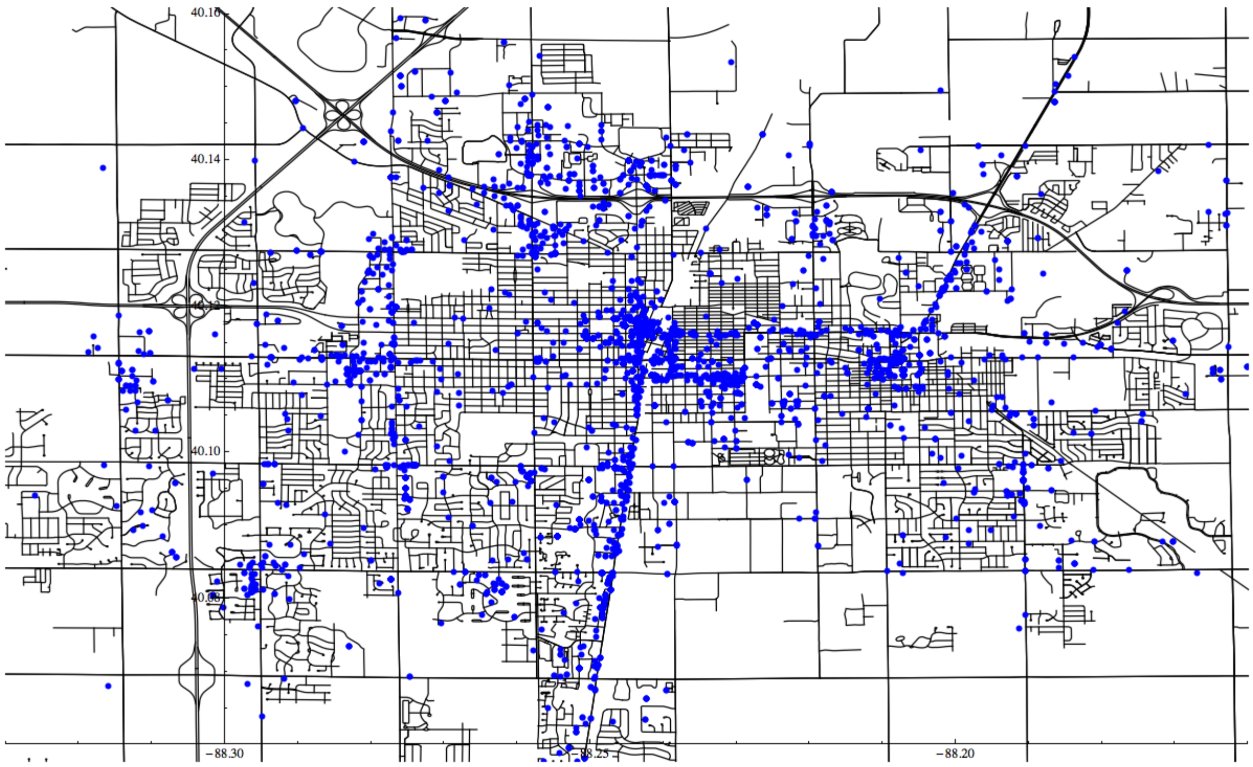
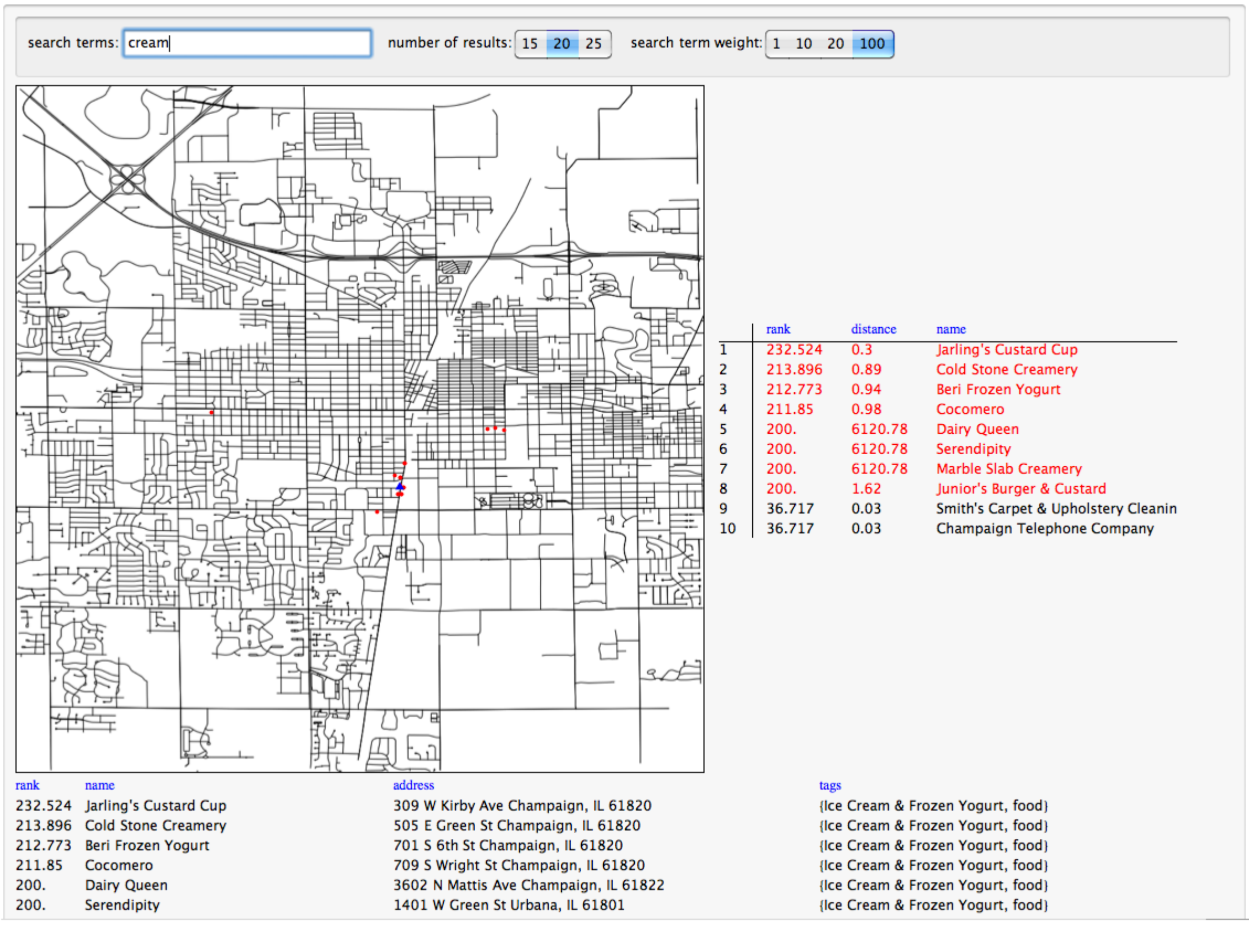


Figure 3:



search terms: cream

number of results: 15 [✓ 20] 25

search term weight: 1 10 20 [✓ 100]

	rank	distance	name
1	232.524	0.3	Jarling's Custard Cup
2	213.896	0.89	Cold Stone Creamery
3	212.773	0.94	Beri Frozen Yogurt
4	211.85	0.98	Cocomero
5	200.	6120.78	Dairy Queen
6	200.	6120.78	Serendipity
7	200.	6120.78	Marble Slab Creamery
8	200.	1.62	Junior's Burger & Custard
9	36.717	0.03	Smith's Carpet & Upholstery Cleanin
10	36.717	0.03	Champaign Telephone Company

rank	name	address	tags
232.524	Jarling's Custard Cup	309 W Kirby Ave Champaign, IL 61820	{Ice Cream & Frozen Yogurt, food}
213.896	Cold Stone Creamery	505 E Green St Champaign, IL 61820	{Ice Cream & Frozen Yogurt, food}
212.773	Beri Frozen Yogurt	701 S 6th St Champaign, IL 61820	{Ice Cream & Frozen Yogurt, food}
211.85	Cocomero	709 S Wright St Champaign, IL 61820	{Ice Cream & Frozen Yogurt, food}
200.	Dairy Queen	3602 N Mattis Ave Champaign, IL 61822	{Ice Cream & Frozen Yogurt, food}
200.	Serendipity	1401 W Green St Urbana, IL 61801	{Ice Cream & Frozen Yogurt, food}

Figure 4:

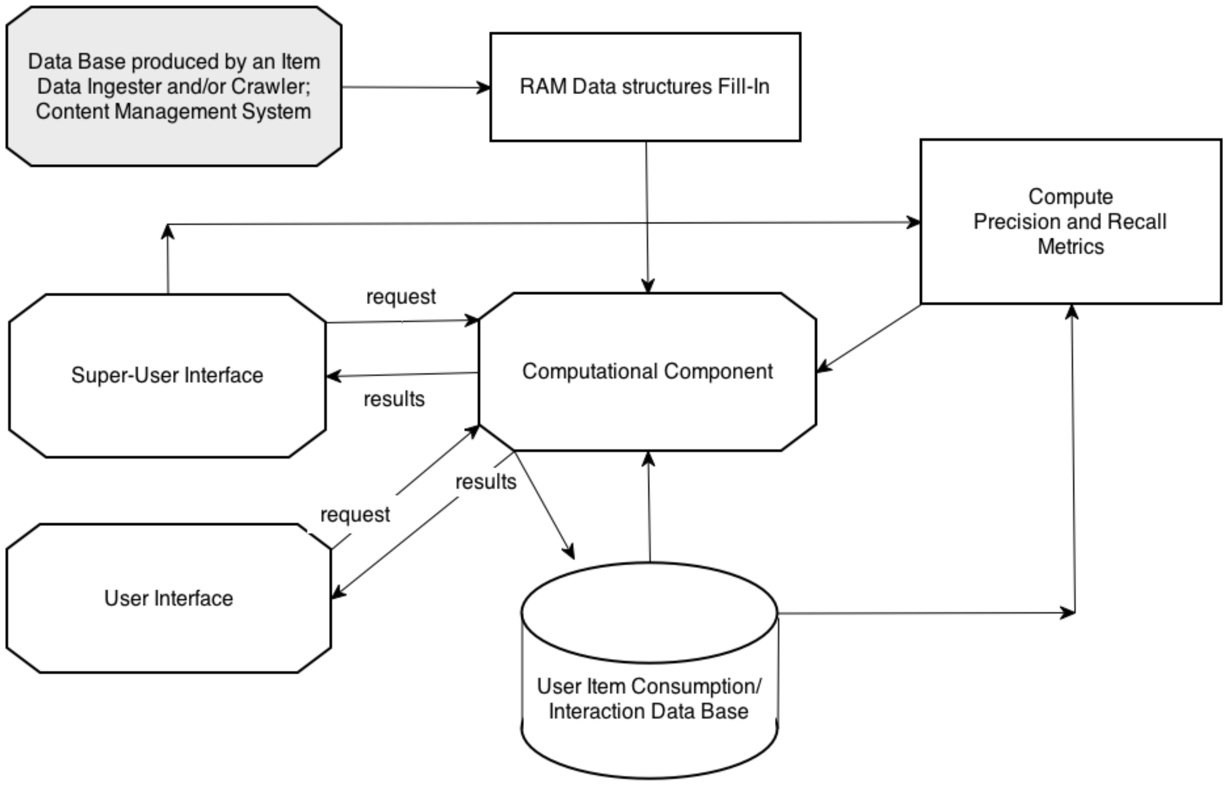

Data Base produced by an Item Data Ingester and/or Crawler; Content Management System
RAM Data structures Fill-In
Compute Precision and Recall Metrics
request
Super-User Interface
Computational Component
results
results
request
User Interface
User Item Consumption/ Interaction Data Base

Figure 5:

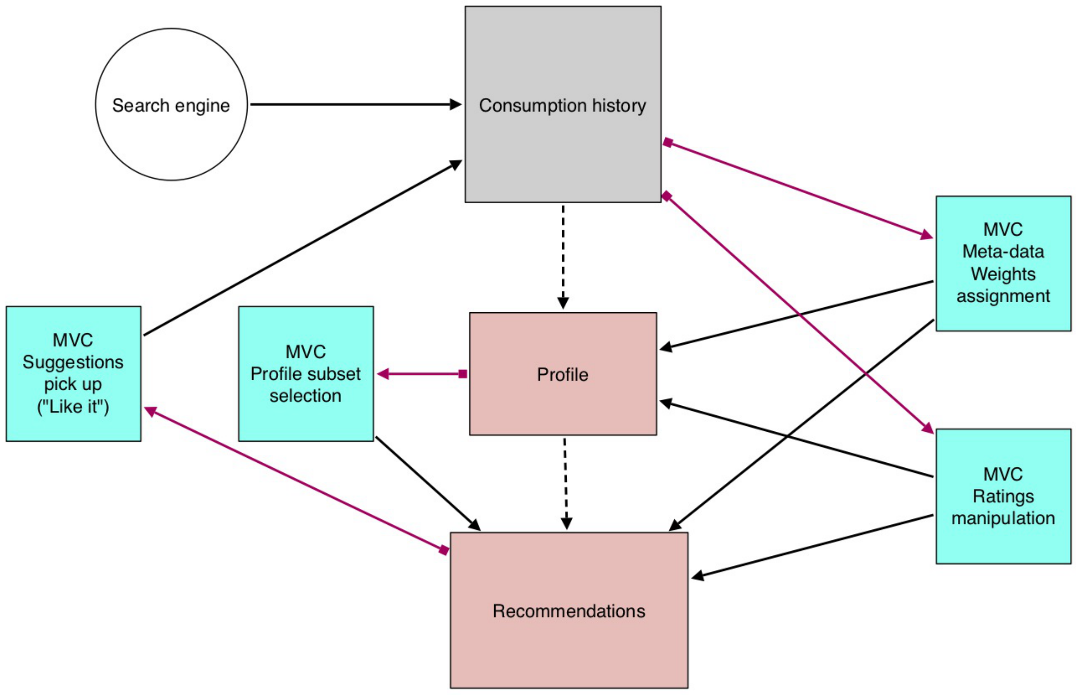

Search engine
Consumption history
MVC Meta-data Weights assignment
MVC Suggestions pick up ("Like it")
MVC Profile subset selection
Profile
MVC Ratings manipulation
Recommendations

Figure 6:

Some philosophy

When talking about recommendation most people think about collaborative filtering.

Collaborative filtering $\approx$ crowd-sourcing.

Item-item recommenders can be used both

1. at cold start (no consumption data)
2. when large consumption data is available.

Item-item recommenders are better; they are more precise, and allow more tuning.

The presented item-item recommender can be used on different kinds of data. (As demonstrated earlier…)

Some theory

The Basic Recommendation Algorithm (BRA) of this item-item recommender takes as arguments : 1. a list of item indexes *itemInds*, 2. star ratings *ratings* for the items in *itemInds*, and 3. an integer k for the desired number of recommendation results.

BRA has the following steps:

1. Represent the data in a item $\times$ meta-data matrix $M := M_{01} \in \{0, 1\}^{n \times m}$. ($n$ items, m meta-datums.)
2. To the columns (or entries) of M_{01} are assigned different weights $\{w_1, ..., w_m\}$.
3. Form the matrix $M := M_{01} diag(w_1, ..., w_m), M \in R^{n \times m}$. This step can be done for each user separately.
4. The item-item similarity matrix $S := MM^T$ can be formed (but it can be larger and dense).
5. Make a vector $v \in R^n$ that has zeros everywhere except at the coordinates corresponding to *itemInds*. The values at those coordinates are determined by *ratings*.
6. Do the two matrix-vector multiplications $s := M(vM)$. The vector s is called the *score vector* for *itemInds* or v. (Also the formula $s := Sv = (MM^T)v$ can be used, but MM^T might be large and dense.)
7. Make the list of pairs $\{s_i, i\}, i \in [1, ..., n_s]$, and name that list *sp*.
8. Sort descendingly *sp* according to the first value of each pair.
9. Take the indexes of the first k pairs, i.e. take $sp(1 : k, 2)$.

Some mathematical relations

1. Transposing of a matrix product:

$$(AB)^T = B^T(A^T).$$

2. Denote with S the show $\times$ show similarity matrix. We have

$$S = MM^T = (M_{01} diag(w))(M_{01} diag(w))^T = (M_{01} diag(w))(diag(w)(M_{01})^T) = M^{01} diag({w_1}^2, ..., {w_{n_m}}^2)(M_{01})^T.$$

3. For the show score vector of the user u we have:

$$s_u = Sv = M(vM) = (M_{01} diag(w))(vM_{01} diag(w)).$$

Some explanation aids

```
MatrixPlot[smat, MaxPlotPoints -> 300]
```

```
MatrixPlot[smat.Transpose[smat], MaxPlotPoints -> 300]
```

Predicted ratings

Assume consumption history of user u consists of the items: $\{i_1, ..., i_{n_u}\}$.

In order to calculate the predicted ratings for the recommended items, the following formula is used

$$r_{uj} = \sum_{i \in \{i_1,...,i_{n_u}\}} S_{ij} r_{ui} / \sum_{i \in \{i_1,...,i_{n_u}\}} S_{ij}, j \in \{j_1, j_2, ..., j_k\},$$

in which S_{ij} denotes the similarity between the shows i and j. (Note that the ratings on the left-hand side are not known – they are the ones that need to be predicted.)

We can do the calculations of formula (4) using linear algebra operations.

1. Compute the matrix $A \in R^{k \times n_u}$

$$A = M(\{j_1, ..., j_k\}) M(\{i_1, ..., i_{n_u}\})^T.$$

2. This formula finds the sums in the denominator of (4):

$$norms = A.[1, 1, ..., 1]^T, where[1, 1, ..., 1] \in R^{n_u}, norms \in R^k.$$

3. This formula finds the sums in the numerator of (4):

$$t = A.[r_{ui_1}, ..., r_{ui_{n_u}}]^T, t \in R^k$$

4. Compute the predicted ratings with the formula $\{r_{uj_1}, r_{uj_2}, ..., r_{uj_k}\} = t/norms$, where the division is element by element, i.e.

$$r_{uj_h} = t_h / norms_h, h \in [1, ..., k].$$

5. Re-order the indexes $\{j_1, j_2, ..., j_k\}$ according to the descending order of the predicted ratings $\{r_{uj_1}, r_{uj_2}, ..., r_{uj_k}\}$.

Meta-data characterization

The items can be characterized with several types of meta-data.

For the recommenders demonstrated earlier I harvested meta-data from

- TIGER
- Yelp
- Wikipedia
- Other sources

Using clustering and Latent Semantic Analysis (LSA) to make characterizations. For example, the themes in the demonstrated movie recommender are derived by using LSA.

Assume, we have data for millions of users and thousands of items. We can find, say, 1000 clusters of items based on users' consumption history. Each cluster is represented as a column in the item × meta-data matrix.

Diversification of the recommendations

Given the consumption history of a user, we can

1. Change weights of characterizing meta-data:

1.1 Manually.

1.2 Automatically (similar to search engine presentation[4]).

2. Use subsets of items:

2.1. Based on recency.

2.1.1. Last five shows.

2.1.2. Random sample of three elements from the last ten consumed items.

2.2. Based on time.

2.2.1. Day of the week. (What shows the user watches on Tuesday?)

2.2.2. Time interval of the day. (What museums are open right now? Or between 7pm and 9pm?)

2.2.3. Time interval from now. (What places and were visited in the last 48 hours?)

2.3. Based on meta-data type.

2.3.1. Pick all items with a particular tag.

2.3.2. Pick recently consumed items with a particular tag.

3. Taking subsets of the profile

3.1. A sampling or an outlier algorithm that would be applied on profile's list of pairs $\{score_i, tag_i\}, i \in N$.

Outlier detection

Optional

The purpose of the outlier detection algorithms is to find those items that have scores significantly higher than the rest.

Taking a certain number of items with scores among top k scores is not the same as an outlier detection, but it can be used as a replacement.

Let us consider the following set of 50 numbers:

```
pnts = RandomVariate[GammaDistribution[5, 1], 50]
```

```
{1.57597, 5.03009, 6.18286, 7.44338, 3.3172, 5.55868, 5.47957, 3.07219, 1.51227, 3.36605, 2.62238, 5.25
```

If we sort those numbers descendingly and plot them we get:

```
nts = pnts // Sort // Reverse;
ListPlot[pnts, PlotStyle -> {PointSize[0.015]}, Filling -> Axis, PlotRange -> All]
```

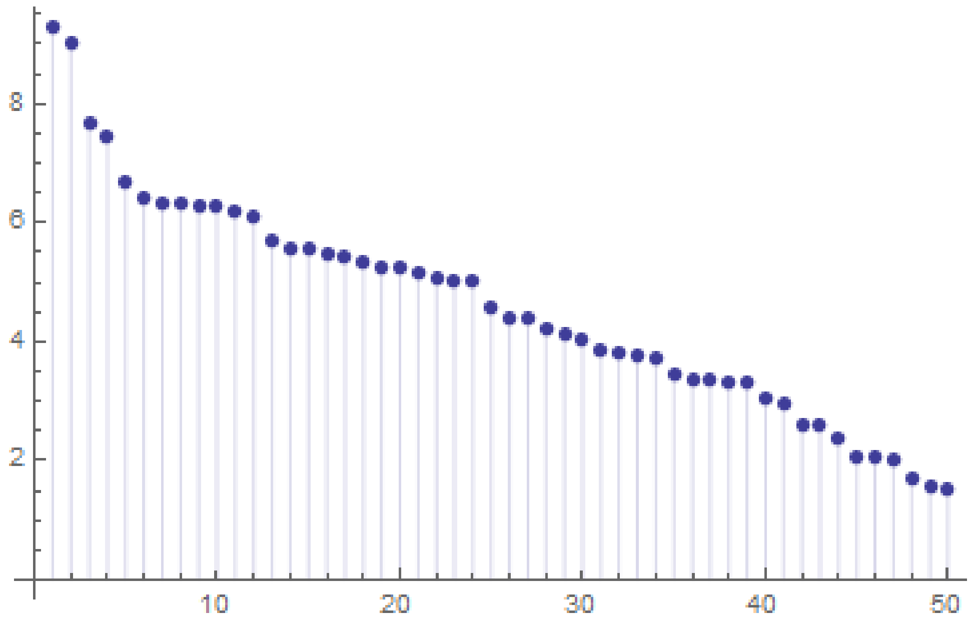


Figure 7:

Let use the following outlier detection algorithm: 1. Find all values in the list that are larger than the mean value multiplied by 1.5; 2. Then find the positions of these values in the list of numbers.

In *Mathematica* this is implemented in the following way.

```
pos = Flatten[Map[Position[pnts, #] &, Select[pnts, # > 1.5 Mean[pnts] &]]]
```

```
{1, 2, 3, 4}
```

Lets plot all points in blue and the so found outliers in red:

```
ListPlot[{pnts, Transpose[{pos, pnts[[pos]]}]}, PlotStyle -> {PointSize[0.015], PointSize[0.009]}, Fill
```

Instead of the mean value we can use another reference point, like the median value. Obviously, we can also use a multiplier different than 1.5.

Computational complexity and scaling

The recommendation algorithm shown above uses sparse linear algebra, hence it is very fast.

Data too large for one processor

Making the computations in parallel is (almost) trivial.

Multi recommender structure

One can hook-up several recommenders for items of different types and provide cross-domain recommendations.

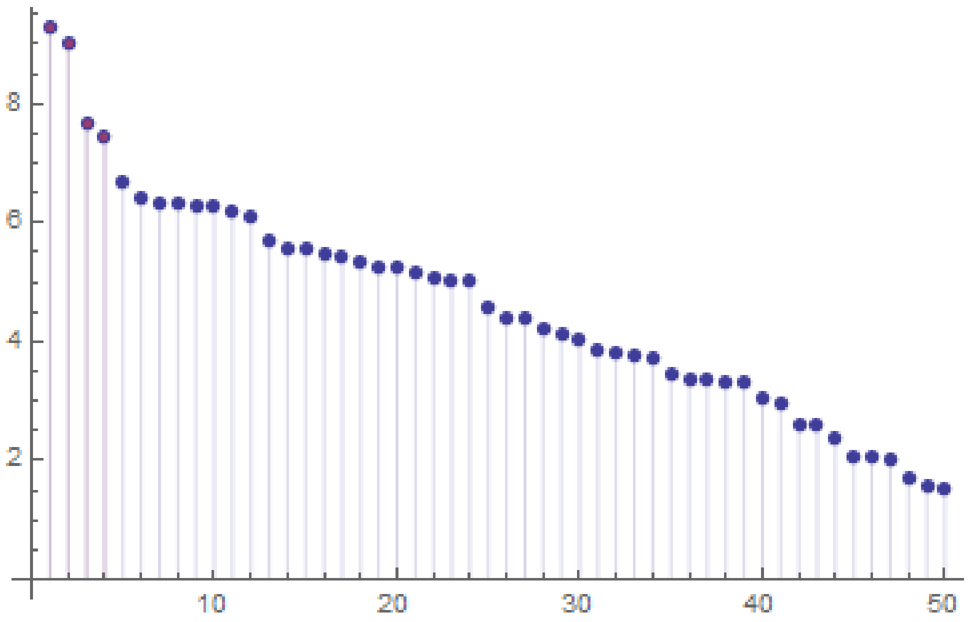
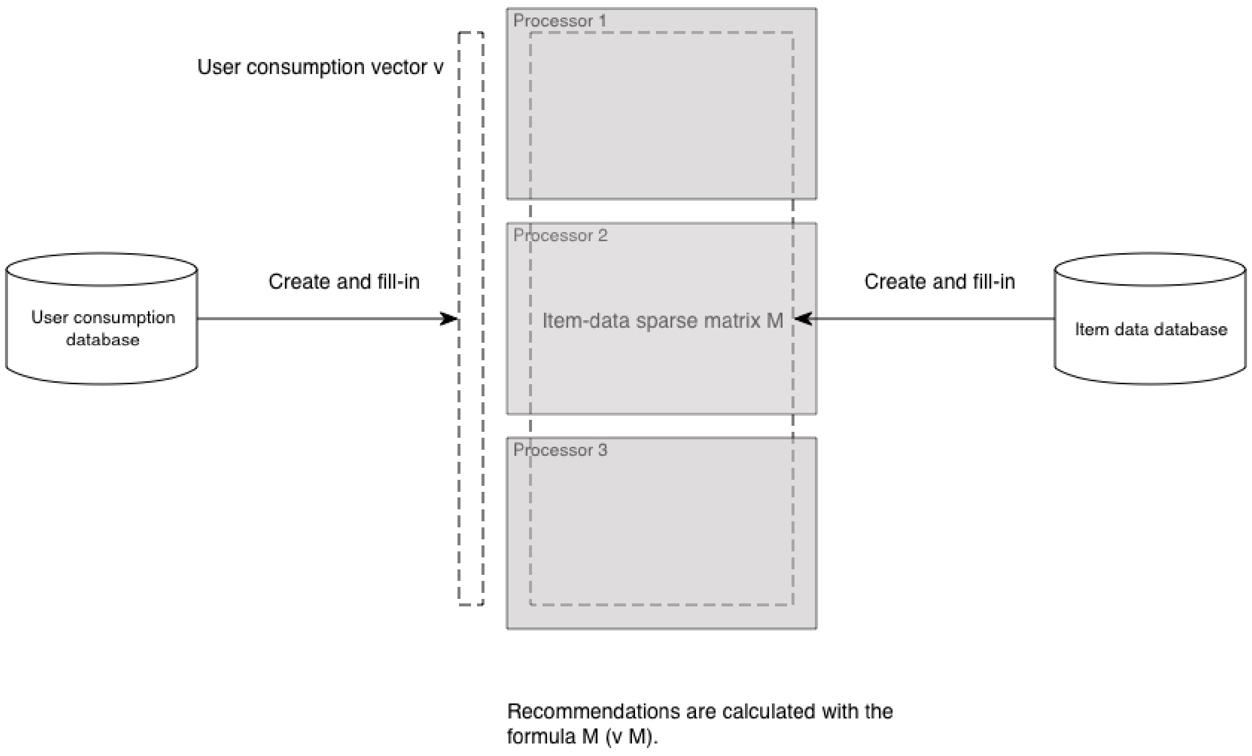


Figure 8:



User consumption vector v
Processor 1
Processor 2
Processor 3
Item-data sparse matrix M
Create and fill-in
Create and fill-in
User consumption database
Item data database
Recommendations are calculated with the formula M (v M).

Figure 9:

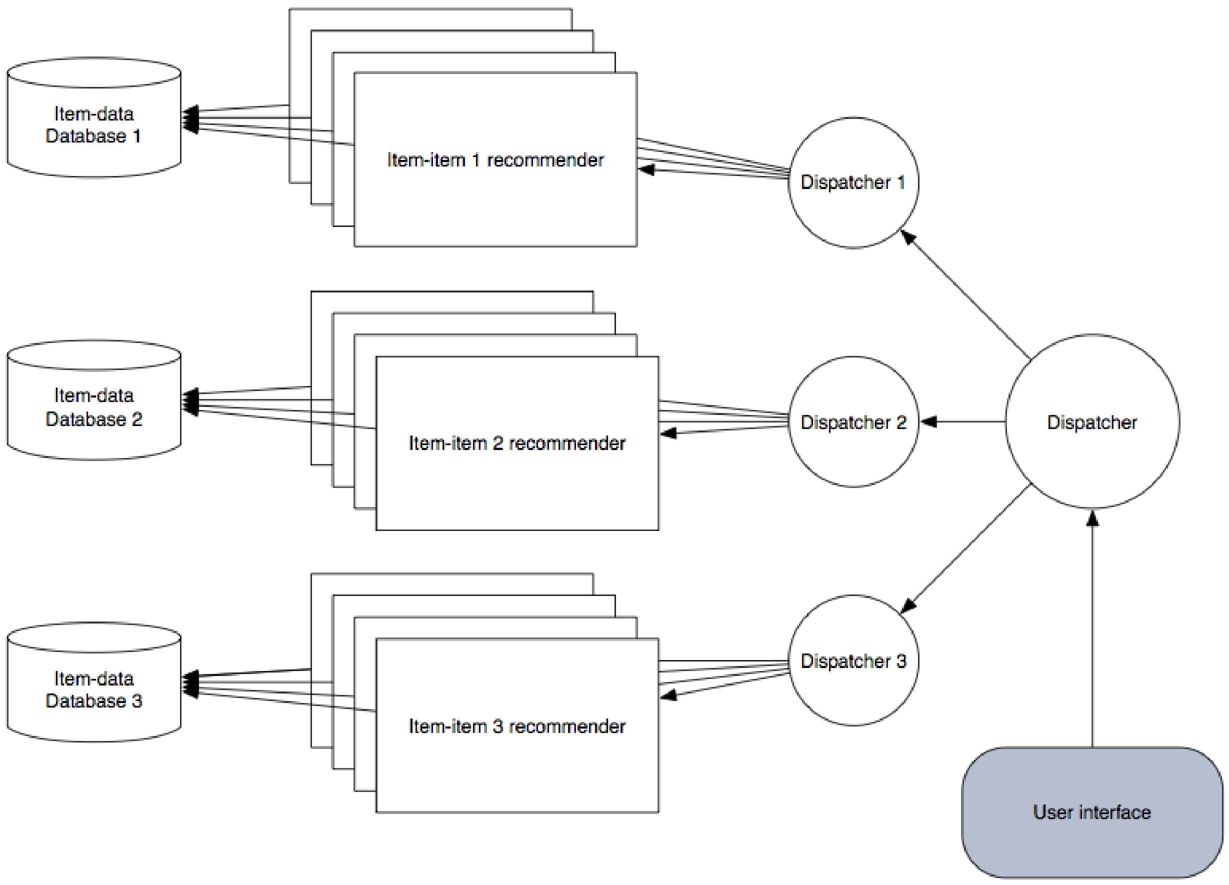

Item-data Database 1
Item-item 1 recommender
Dispatcher 1
Item-data Database 2
Item-item 2 recommender
Dispatcher 2
Dispatcher
Item-data Database 3
Item-item 3 recommender
Dispatcher 3
User interface

Figure 10:

Complexity analysis

Too long to explain here, but analysis and experimentation can and should be done for the computational complexity involved of all parts of the algorithm.

(Extension for large number of users.[5])

Implementations design

Object-oriented design of the item-item recommender was done that encapsulates:

- matrix representation – item $\times$ meta-data matrix
- column interpretation – rules, offsets
- recommendation algorithms

Sparse Matrix Recommender Framework (ItemRecommender object, hub of functionality)[6]

Extensions: carousels

We want to endow our recommendation engine with the feature of finding for each user u sets of items S and L, and sets of items P, for which the following statement would be true: "because the user u has consumed the items in S and likes items that have all tags in P, the user u would like the items in L".

Using carousels is exemplified by Netflix and Bing for iPad. Netflix places carousels at the home page of each user in order to provide more tuned and detailed recommendations.

It is beneficial to give rigorous mathematical deviation of the "carousel problem": carousels for recommendations[7].

Visual aids

Netflix

Bing for iPad

Design

Extensions: Recommendation Query Language

Instead of designing an API to the Item-Item Recommender (IIR) it might be beneficial to design and implement a Recommendation Query Language (RQL) that is an interface to IIR.

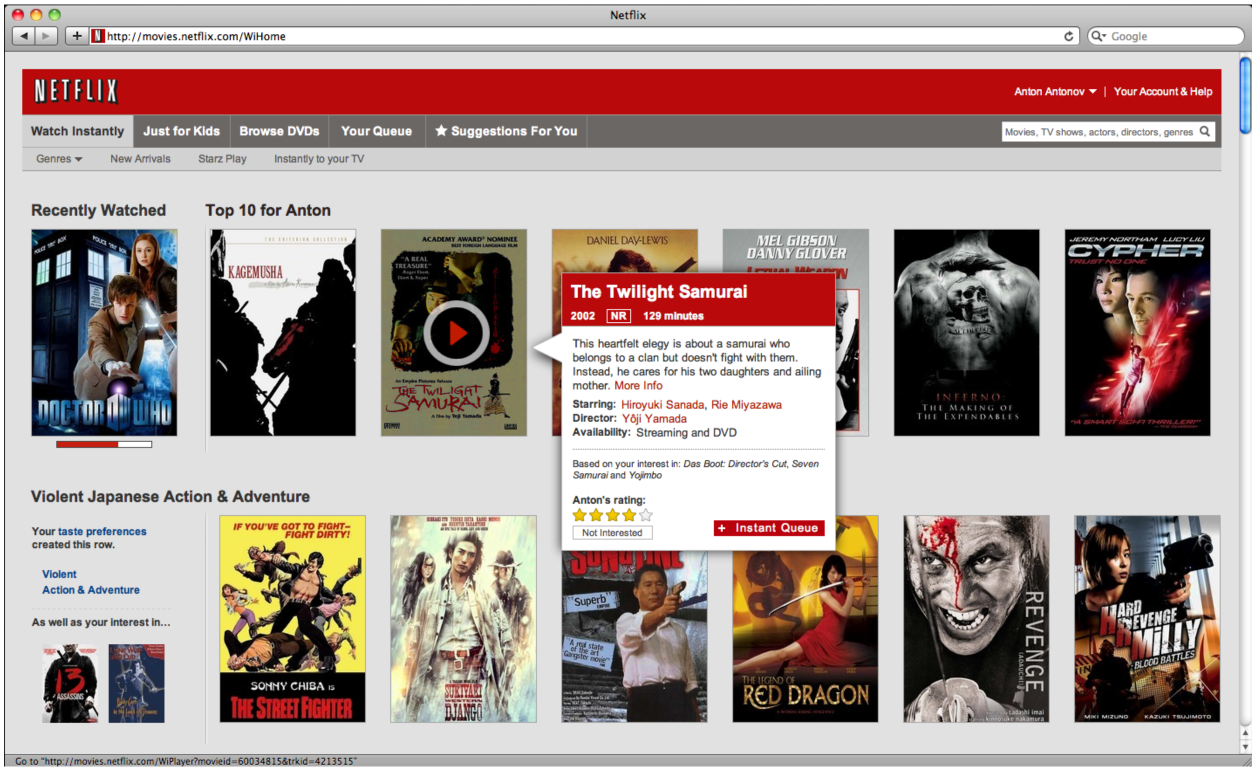
The proposed RQL is very similar SQL.

Here is an example:

{ttfamily recommend for history`$historySet$`order by relevance limit 20}

{ttfamily recommend according to (select from (find profile`$historySubSet$`) if tag in (take tags of`$profileSubSet$`)) limit 20}

Note that RQL (and SQL) are functional programming languages.


Netflix
http://movies.netflix.com/WiHome
Google
NETFLIX
Anton Antonov | Your Account & Help
Watch Instantly
Just for Kids
Browse DVDs
Your Queue
★ Suggestions For You
Movies, TV shows, actors, directors, genres
Genres
New Arrivals
Starz Play
Instantly to your TV
Recently Watched
Top 10 for Anton
Violent Japanese Action & Adventure
Your taste preferences created this row.
Violent
Action & Adventure
As well as your interest in...
Not Interested
+ Instant Queue
Go to "http://movies.netflix.com/WiPlayer?movieid=60034815&trkid=4213515"

The Twilight Samurai

2002 NR 129 minutes

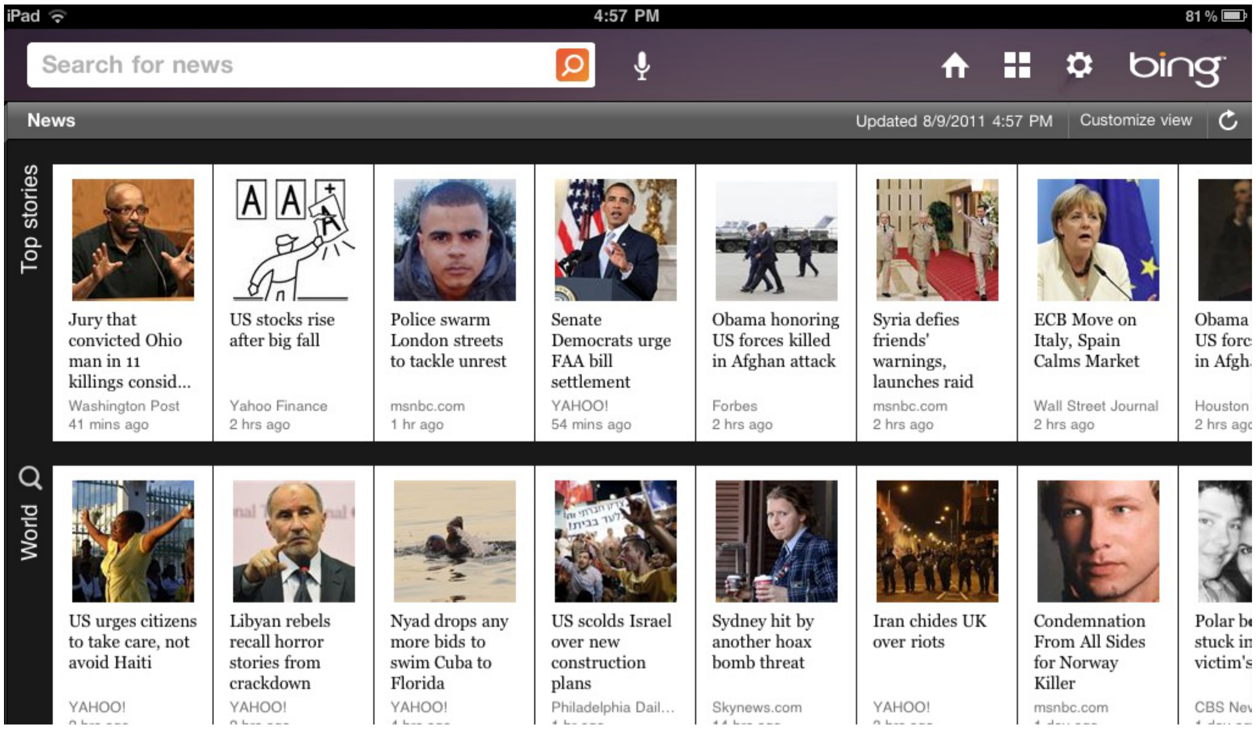
This heartfelt elegy is about a samurai who belongs to a clan but doesn't fight with them. Instead, he cares for his two daughters and ailing mother. More Info

Starring: Hiroyuki Sanada, Rie Miyazawa
Director: Yôji Yamada
Availability: Streaming and DVD

Based on your interest in: *Das Boot: Director's Cut, Seven Samurai* and *Yojimbo*

Anton's rating:

Figure 11:



Jury that convicted Ohio man in 11 killings consid... — Washington Post, 41 mins ago
US stocks rise after big fall — Yahoo Finance, 2 hrs ago
Police swarm London streets to tackle unrest — msnbc.com, 1 hr ago
Senate Democrats urge FAA bill settlement — YAHOO!, 54 mins ago
Obama honoring US forces killed in Afghan attack — Forbes, 2 hrs ago
Syria defies friends' warnings, launches raid — msnbc.com, 2 hrs ago
ECB Move on Italy, Spain Calms Market — Wall Street Journal, 2 hrs ago
Obama US forc in Afgh — Houston, 2 hrs ago

US urges citizens to take care, not avoid Haiti — YAHOO!
Libyan rebels recall horror stories from crackdown — YAHOO!
Nyad drops any more bids to swim Cuba to Florida — YAHOO!
US scolds Israel over new construction plans — Philadelphia Dail...
Sydney hit by another hoax bomb threat — Skynews.com
Iran chides UK over riots — YAHOO!
Condemnation From All Sides for Norway Killer — msnbc.com
Polar be stuck in victim's — CBS New

Figure 12:

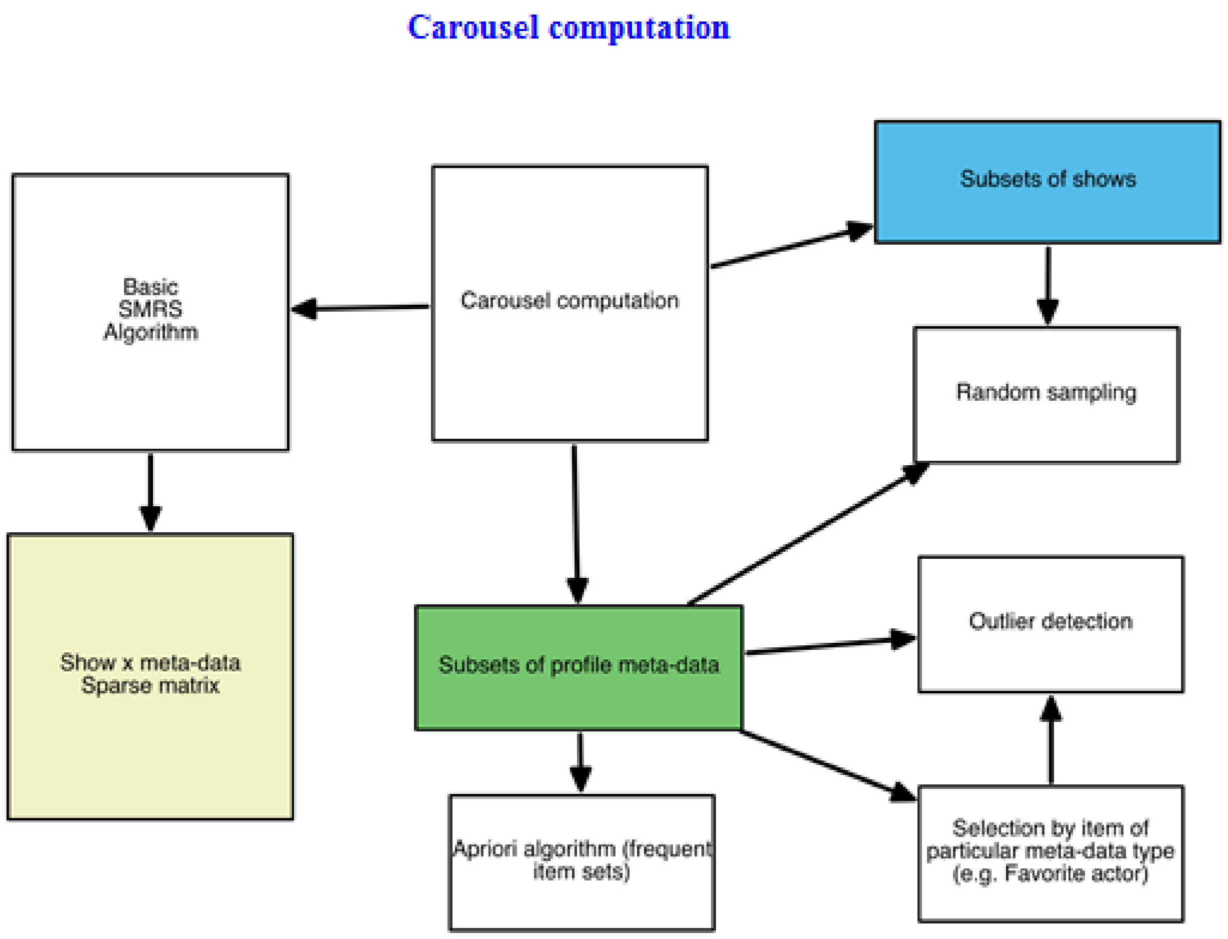

Carousel computation
Basic SMRS Algorithm
Carousel computation
Subsets of shows
Random sampling
Show x meta-data Sparse matrix
Subsets of profile meta-data
Outlier detection
Apriori algorithm (frequent item sets)
Selection by item of particular meta-data type (e.g. Favorite actor)

Figure 13:

BNF for RQL

```
<recommend for profile> ::=
"recommend" <data fields spec>
       "according to" <profile spec>
       "order by" <order spec>
       "with weights" <weight-tag list>
       "limit" <integer>

<recommend for history> ::=
"recommend" <data fields spec>
      "for history" <item history spec>
      "order by" <order spec>
      "with weights" <weight-tag list>
      "limit" integer>
```

```
<order spec> ::= "similarity" | "predicted ratings" | "absolute ratings" | "relevance"

<data fields spec> ::= (<tag type name> | <data field name>) ["," <data fields spec>]

<profile spec> ::= <weight-tag spec> | <profile finder spec> |  <weight-tag list sum> | <profile filter

<profile finder spec> ::= "find profile for" <item history spec> "with weights" <weight-tag list> [<pro

<weight-tag list sum> ::= [<double>] <weight-tag list> | [<double>] <weight-tag list> "+" <weight-tag l

<profile limit spec> ::= "limit" <integer> | "limit with" <outlier detector spec>

<outlier detector spec> ::= <outlier detector names> | <outlier detector function>

<tag spec> ::= <tag value> | <tag id>

<tag seq> ::= <tag spec> ["," <tag seq>]

<tag list> ::= "(" <tag list> ")"

<tag type spec> ::= <tag type name> | <tag type id>

<weight-tag seq> ::= "(" <double> "," <tag spec> ")" | "(" <double> "," <tag type spec> ")" ) ["," <wei

<weight-tag list> ::= "automatic" | "(" <weight-tag seq> ")"

<item history spec> ::= "SQL SELECT statement" | <item history list>

<item history list> ::= "(" <item id> "," <rating> ")" | <item history list>

<item history list> ::= "(" <item id> "," <location id> "," <time spec> "," <rating> ")" | <item histor

<rating> ::= "1"|"2"|"3"|"4"|"5"
```

```
<profile filtering> ::= "select from" <profile spec> ( "if tag in" <tag list> | "if rank >" <double> |
```

```
<profile tags> ::= "take tags of" <profile spec>
```

```
<global weight setting> ::= "use" <weight-tag list>
```

Summary

By representing item-data relationships in a matrix form, we can use sparse linear algebra algorithms to implement an item-item recommender that is fast, very tunable, and scalable both data-wise and hardware-wise.

The presented three recommenders demonstrate the flexibility of the approach taken …