Lecture 6: Recommender Systems for E-Commerce

Derek Bridge

Motivation

- **The recommender systems we have looked at so far**
  - **maintain a profile of the user's *long-term* interests, either**
    - content-based or
    - collaborative
  - **can be used either**
    - reactively: user requests a recommendation ("user pull")
    - proactively: system makes recommendations unbidden ("system push")
  - **but cannot (easily) respond to the user's *short-term* goals and interests**

Motivation

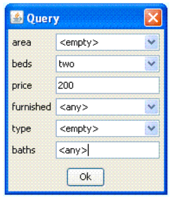
- **We'll look at short-term goals and interests in the context of a user accessing an online product catalog**
- **The user will reveal his/her constraints and preferences in the form of a *query* (or sequence of queries)**
- **Products in the catalog will have descriptions**
  - **these will tend to be *structured descriptions***
- **The recommender system will be primarily**
  - **content-based: matches the query against the product descriptions**
  - **reactive: makes recommendations on request**

Running example

Id	Address	Type	Bdrms	Bthrms	Rent	Furnished	Location
1	16 Oxford Road	Flat	1	1	265	Yes	Acton
2	2 Heathfield Road	House	3	2	370	Yes	Acton
3	101 Nassau Road	Flat	2	1	271	No	Barnes

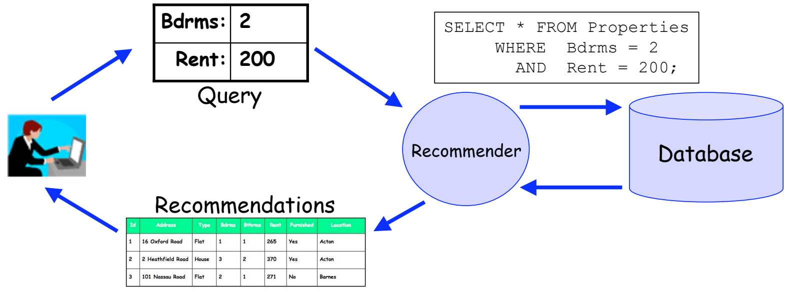
Eliciting the query: forms

- User can express constraints by submitting a form
  - fields can be left blank to indicate 'wild card'



Retrieval

- The query values are used to build an SQL query
  - executed against the database
  - its results are shown to the user



Filter-based retrieval

- Typically, SQL will be used to perform *filter-based retrieval*
  - exact-matching
- This brings two problems
  - result set may be empty: query is *over-specified*
  - result set may be too long: query is *under-specified*
- Some systems will attempt to lessen the first of these problems, e.g.
  - User's query: Rent = 200
  - SQL: `SELECT * FROM Properties`
    `WHERE Rent > 190 AND Rent < 210`
  - But this has at least two weaknesses! What are they?

Similarity-based retrieval

- An alternative is *similarity-based retrieval*:
  - score each item (based on similarity to the query)
  - rank them on their scores
  - recommend those at the top of the ranking
- In this case,
  - result set is never empty (no matter how under-specified the query is)
  - result set can be a manageable length, and in any case is ordered

Similarity, *sim(q, i)*

- A *global* similarity function, *sim(q, i)*, is defined as a combination of *local similarity* functions, $sim_A(q, i)$, one per attribute (column) in the query

	Id	Address	Type	Bdrms	Bthrms	Rent	Furnished	Location
i:	1	16 Oxford Road	Flat	1	1	265	Yes	Acton
		q:	Flat	3		200	No	Hayes
		$sim(q, i) = \sum$:	sim_{type}	sim_{bdrms}	sim_{bthrm}	sim_{price}	$sim_{furnished}$	$sim_{location}$

Global and local similarities

- E.g. sum the local similarities

$$sim(q,i) = \sum_{A \in q} sim_A(q,i)$$

- E.g. take a weighted sum

$$sim(q,i) = \sum_{A \in q} w_A \times sim_A(q,i)$$

- E.g. can take averages or weighted averages

Local similarity functions

- E.g. one local similarity function is called the *overlap function*
  - very good for non-numeric attributes, especially ones with just two values, e.g. Type, Furnished

$$sim_A(q,i) = 1 \text{ if } q = i$$
$$0 \text{ otherwise}$$

Local similarity functions

- For numeric attributes, the *absolute difference* can form the basis:

$$abs(q-i)$$

- But, attributes with large ranges can overpower other attributes. So *normalize:*

$$\frac{abs(q-i)}{A_{\max} - A_{\min}}$$

- And,this is a distance function but we need a similarity function so *subtract from 1*:

$$sim_A(q,i) = 1 - \frac{abs(q-i)}{A_{\max} - A_{\min}}$$

Local similarity functions

- Human experts might define domain-specific similarity functions, esp. for non-numeric attributes
  - E.g. $sim_{location}(q, i)$

	Acton	Barnes	Chelsea	Ealing	Hayes
Acton	1	0.6	0.3	0.9	0.8
Barnes		1	0.2	0.8	0.7
Chelsea			1	0.6	0.5
Ealing				1	0.8
Hayes					1

Exercise

- Assuming
  - global similarity is the sum of local similarities
  - Bdrms has range 0-8
  - Rent has range 100-750

what is the global similarity of property number 1 for the query shown?

	Id	Address	Type	Bdrms	Bthrms	Rent	Furnished	Location
i:	1	16 Oxford Road	Flat	1	1	265	Yes	Acton
q:			Flat	3		200	No	Hayes
$sim(q, i) = \sum$:								

Utility-based retrieval

- Some people say that what we actually want is *utility-based retrieval*
- Items will be scored by their *utility*
- The scoring functions typically compute a *global utility* as the sum, weighted sum, average or weighted average of *local utilities*
- E.g. a weighted sum

$$util(q,i) = \sum_{A \in q} w_A \times util_A(q,i)$$

Utility-based retrieval

- How do we then define local utility?
- Often, we use similarity as a 'proxy' for utility

$$util_A(q,i) = sim_A(q,i)$$

  - the more similar the query and the item, the more useful
- In this case, utility-based retrieval and similarity-based retrieval are just two names for the same idea!

Utility-based retrieval

- But utility-based retrieval and similarity-based retrieval are not always just two names for the same idea
  - for some attributes, especially some numeric attributes, local utility may be defined differently

Local utility function: less-is-better

- For an attribute such as Rent, where the user's value *q* specifies a preferred maximum,

$$util_A(q,i) = 1 \text{ if } i \le q$$
$$= 0.5 \times \frac{A_{\max} - i}{A_{\max} - q} \text{ otherwise}$$

- We'll graph this
- Rent has range 100-750. The rent for property 1 is 265. What's the local utility if the user's preferred maximum rent is (a) 300, (b) 200?

Local utility function: more-is-better

- For an attribute such as Bdrms, where the user's value *q* specifies a preferred minimum,

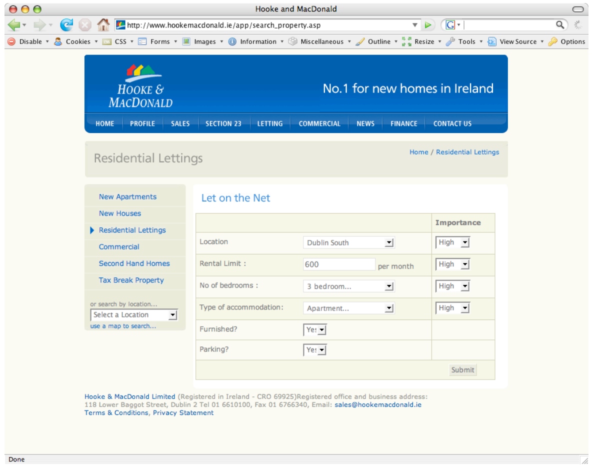
$$util_A(q,i) = 1 \text{ if } i \ge q$$
$$= 0.5 \times \frac{i - A_{\min}}{q - A_{\min}} \text{ otherwise}$$

- We'll graph this
- Bdrms has range 0-8. Property 1 has 1 bdrm. What's the local utility if the user's preferred mimimum is (a) 1, (b) 2?

Recap

- In contrast to filter-based retrieval, similarity-based retrieval and utility-based retrieval
  - compute a score for each item
    - typically, a sum of local similarities/utilities, one per attribute in the query
    - typically, for local utility, similarity is used as a proxy - but not always
  - ranks the items in order of descending score
  - recommends the top-ranking items
- A surprisingly under-used idea
  - but, where used, highly successful

Hooke & Macdonald



Problem

- **When using similarity-based retrieval, the recommendations may be similar to the user's query (and when using utility-based retrieval, the recommendations may be of high utility)**
- **…but they tend to be similar to each other as well**
- **This reduces the chance that one of the recommendations will satisfy the user**

Diversity

- **We must endeavour to recommend items that are similar to the query/have high utility but that are *different from each other***
  - **the result set must be *diverse***
  - **this increases the chances that at least one of the recommended items will satisfy the user**
- **Diversity is different in nature from utility/similarity**
  - **utility/similarity is a property of an individual item (with respect to the query)**
  - **diversity is a property of the result set as a whole**

Diversity-enhanced retrieval

- **Suppose we want to recommend *k* items to the user**
- **The idea in the *Bounded Greedy Selection* algorithm is**
  - **To start with a larger set of candidates ($b \times k$ candidates) that are of high utility**
  - **Then to select *k* of these, one by one, ensuring that each one we select is of**
    - high utility
    - but also <u>low total similarity</u> to those already selected

Bounded Greedy Selection algorithm

- *Candidates* = $b \times k$ items of highest utility (found using utility-based retrieval)
- *Result* = [ ]
- Do the following *k* times
  - *Best* = the member *i* of *Candidates* for which *Quality(i, q, Result)* is highest
  - insert *Best* into *Result*
  - remove *Best* from *Candidates*
- recommend *Result*

Bounded Greedy Selection algorithm

$$Quality(i,q,Result) = util(q,i) + RelDiversity(i,Result)$$

$$RelDiversity(i,Result) = 1 \text{ if } Result \text{ is empty}$$
$$= \frac{\sum_{j \in Result}(1 - sim(i,j))}{size(Result)}$$

Problem

- None of this is personalized!
- Maybe we want the system to use our long-term profile to 'fill in the gaps'
  - complement explicitly specified preferences with user-specific default ones from the user's long-term profile when otherwise unspecified

Example

- The CASPER online recruitment system
  - When a user enters a query, the server returns a set of high utility job adverts
  - Client-side, actions such as saving or replying to job adverts are interpreted as positive feedback
  - These adverts are stored in a long-term profile
  - When the user next enters a query,
    - the server returns a set of high utility job adverts again
    - But also, client-side, they are re-ordered by similarity to the adverts in the long-term profile
- B.Smyth, K.Bradley & R.Rafter (2002): *Personalization techniques for online recruitment services,* Communications of the ACM, vol.45(5), pp.39-40

Problems

- The following problems remain, irrespective of whether we use filter-based, similarity-based, utility-based or diversity-enhanced retrieval:
  - seldom are we able to specify all our requirements up-front
  - seldom are we satisfied with the initial set of results
- We've been assuming a single-shot system
  - submit query, view results, end of story
- If not satisfied, our only option is to revise the query and submit again
  - typically with no guidance!
  - can lead to 'stonewalling'

Solution

- A conversational recommender system
  - an iterative approach
  - users can elaborate their requirements as part of an extended recommendation dialog
  - → next lecture