Chapter 5

Recommendation Based on Personal Preference

Pei Wang

Department of Computer and Information Sciences
Temple University, Philadelphia, PA 19122, USA
E-mail: pei.wang@temple.edu

This chapter first defines a recommendation process, which helps the user to select products from a large number of candidates according to personal preference. Though both conventional database and fuzzy database have been used for this task, none of the two provide a proper solution. A new approach is introduced, which is easy to design and maintain, and provides well justified results. The central idea of this approach is to interpret user preference as consisting of multiple criteria, each of which is relative to available data. The procedure introduced here forms a membership function at run-time according to user request and available data, then use it to rank the candidates.

5.1 The problem

In web-based e-commerce and on-line shopping, people often meet the "selection problem", that is, the user is looking for a product of a certain type, with flexibility or uncertainty in the details of the request. Typically, the user has certain "constraints" that must be satisfied, as well as certain "preference" that are desired, but not specified in absolute terms.

For example, if the user is looking for a home computer, then most requests are in the form of preference: almost everyone prefers a fast CPU, a large hard drive, a cheap price, and so on, but few of the requests really has a fixed range. If the user is search for a flight, then the departure and arrival airports may be determined, while the other parts of the request (such as departure date and time, arrival time, number of transfers, and so on) may be flexible to different degrees.

In this situation, if there are M products satisfying the constraints, and all of them are displayed to the user, then often it is too much information, and the user gets little help in using the preferences to get a final decision. For this situation, a "recommendation system" (also called "recommender system") is desired [Jameson et. al., (2002); Kautz, (1998)]. Briefly speaking, the function of a recommendation system is to reduce the size of the displayed list from M to N (which is a much smaller number), under the condition that the user's preference is respected in the process.

Obviously, we do not want N to be too large, otherwise the recommendation system does not help much. On the other side, we do not want N to be too small — if N is one, then the "recommendation system" is actually making decision for the user. This is not desired, under the assumption that not all the relevant factors about the final decision can be included in the constraints and preferences, therefore the user still want to compare the top candidates, and to make the final decision, by taking all relevant factors into consideration. This is exactly what we usually expect when asking friends or experts for "recommendations" — we hope them to reduce the number of candidates to a manageable level, without making the final decisions for us.

Of course, the desired N may be different from situation to situation, and from user to user, but in general, we can take a default value in the range between five and ten. Such a choice is consistent with the psychological research on human memory (the well known "7 ± 2" phenomenon), as well as the common practice of picking the "top ten" in various categories.

If we can assume that the user only need one product from the M candidates (which is usually true), and that given sufficient time for analysis, comparison, and actual evaluation, the user would select product P_i, then a "good" recommendation system should almost always provides a top-N list containing P_i in it. With the help of such a system, the user's decision-making process is simplified, but still get (usually) the same result.

According to the above definition of the problem, there are still several ways of making recommendations, which have been explored to various degrees [Jameson et. al., (2002); Kautz, (1998)]. For example, recommendation can be made according to a top-N list determined by the voting of domain experts or users for each category. Such votes can be collected explicitly or implicitly, such as through a "collaborative filtering". Or, recommendations can be made according to the similarity among users and similarity among products. For example, according to the selections the users have made before, they are classified into several groups, and the preference of each group is determined by statistics. When a new user is classified into a certain group, recommendations are made according to the preference of the members of the same group.

What we study in this chapter is a special situation, where we assume that there is no historical information available (such as votes, comments, reviews, past selections, and so on), and the recommendation is only based on the constraints and preferences provided by the current user. Since these requirements are highly personal and change from case to case, there is no way to anticipate them, and to determine a top-N list for every case in advance. Instead, the recommendation has to be formed at run-time according to user's requests.

Formally, the problem is defined as the following. DB is a database (or an information system of another type) that contains descriptions about products of certain category. Each product d_i is specified as a vector $d_i = < d_{i1}, d_{i2}, \cdots, d_{it} >$, in which each d_{ij} is the value of the product on an attribute A_j.

In the simplest form, the user's request consists of a constraint vector $c = < c_1, c_2, \cdots, c_t >$ and a preference vector $p = < p_1, p_2, \cdots, p_t >$. Each element of these two vectors is a condition for the corresponding attribute value.

A condition in constraint, c_j, is a binary expression of the form "X *relation* v", where X is a variable representing an attribute value, v is a constant value of the corresponding attribute, and *relation* is one of the following relations: $=, \neq, <, \leq, >, \geq$, where the last four are only defined on attributes whose values form a total order (therefore $X \neq Y$ can always be further specified as $X < Y$ or $X > Y$). The value of such an expression is either 1 (for "true") or 0 (for "false").

A condition in preference has the same form as a constraint, except that the relation is "$\approx$", for "similar to", and the expression is not binary, but has a value in a certain range. That is, to what extent a preference is satisfied is a matter of degree. Without losing generality, we take the range to be the interval [0, 1]. As a special case, "$X \approx v$" has value 1 if and only if "$X = v$" is true. Therefore, the similarity relation can be seen as a generalization of the equality relation, from binary to multi-valued. For attributes whose values form a total order, we define two new relations: "$X \ll$" is defined as "$X \approx v_{min}$", and "$X \gg$" is defined as "$X \approx v_{max}$", where "v_{min}" and "v_{max}" are the minimum and maximum value of the corresponding attribute, respectively (they do not need to be a finite value). Intuitively, these preferences mean "has a small value" and "has a large value", respectively.

Therefore, constraints and preferences correspond to different ways to assign a "score" to a value of a product on an attribute. After applying on a value d_{ij}, a constraint c_j provides a score in $\{0, 1\}$, a preference p_j provides a score in [0, 1].

Since the variable X in above requirements is just a place holder, it can be omitted without causing confusion. In the following, therefore, we will use "*relation v*" for a constraint, and use "$\approx v$", "$\ll$", or "$\gg$" for a preference. There is a special condition "*null*" for "no constraint/preference", which should be ignored in the recommendation procedure.

For example, if each flight (from a city to another in a given date) is specified only by departure time, arrival time, and prices, then the preference "leave around 9 AM, arrive early, with a low price" is represented as "≈ 9, $\ll$, $\ll$". Similarly, if each computer is specified only by its CPU speed, hard-drive size, and price, then the preference "fast and cheap" is represented as "$\gg$, *null*, $\ll$".

Given these definitions, we can represent a recommendation procedure for top-N products as the following:

(1) Apply the user constraint $c = < c_1, c_2, \cdots, c_t >$ on the database DB. For each product $d_i = < d_{i1}, d_{i2}, \cdots, d_{it} >$, if its value d_{ij} satisfies condition c_j for all $j = 1 \cdots t$, then it becomes a candidate for recommendation.
(2) The above step produces a set of candidates of size M. If $M \leq N$, then recommendation is unnecessary, and the procedure ends. Otherwise the procedure continues.
(3) Apply the user preference $p = < p_1, p_2, \cdots, p_t >$ to each product d_i in the candidate set, and get a vector of scores $< s_{i1}, s_{i2}, \cdots, s_{it} >$, where s_{ij} is the return value of applying preference p_j to value d_{ij}.
(4) Apply a summarizing function $f(x)$ to the score vector of each product d_i in the candidate set to get a total score $s_i = f(< s_{i1}, s_{i2}, \cdots, s_{it} >)$, which is a numerical value.
(5) Select the top N products according to their total scores, and return them as the recommendation to the user.

5.2 The existing techniques

Described in this way, recommendation can be seen as *selective information retrieval*. Can we just use existing techniques, such as relational database, to carry out this process?

Obviously, the constraints we defined above are very close to conditions in database queries. Now the question is: can the preferences defined above be properly handled as conventional database queries? Obviously, a direct mapping does not exist, because database queries are based on binary logic, that is, whether a data item satisfies a condition is a matter of true or false, not a matter of degree. But how about approximative mapping? Maybe we can translate a preference into a query condition without losing too much information.

Such an approximation is possible — actually people are forced to do so on a daily basis. If a user prefers a flight departing around 9 AM, she often has to specify her preference as a query condition for a time interval [9-Δ, 9+Δ], where Δ is a constant, like 30 minutes or so. If a user prefers a fast and cheap computer, he has to specify the CPU speed as above a certain threshold (such as 2 GHz) and the price as below a certain threshold (such as \$1000).

Since the query language of conventional database only uses binary conditions, what happened above is that preferences are converted into constraints, then represented as query conditions. Of course this is an approximation, but is it good enough?

We say that the very idea of "recommendation" is lost in such an approximation, so it is unacceptable as a way to build a recommendation system. Here the problem is: if preference is treated as constraints, whether a product satisfies the user's request is a matter of true or false, and there is no way to rank the candidates that satisfying all the conditions. Consequently, the user simply gets all products that returned by the query, or a random subset of it as the "top-N" list, which is not really based on user preference.

When the user is forced to specify an interval for each attribute value, two extreme cases often happen: the query either returns no product or returns too many. For example, when the user is looking for a "fast and cheap notebook computer", if the request is "translated" into query for "faster than 3 GHz and cheaper than \$500", there may be nothing available, but if it is treated as a query for "faster than 1 GHz and cheaper than \$3000", then there may be five hundred of them. If we really want a certain number (say 5 to 10) of products returned by the query, the conditions in it should be tuned properly. For example, a query for "faster than 2 GHz and cheaper than \$1250" returns five products, which looks like a good recommendation — until we think about the following issues.

To translate the preferences into a query which returns data items around a certain number, the user must be either very familiar with the distributions of the attribute values, or willing to try many different intervals/thresholds. In the former case, the user is an expert in the domain, so rarely needs any recommendation at all. In the latter case, the process is very time-consuming and often frustrating — many readers of this book may have such personal experience. Recommendation system cannot assume either of the two cases, because it is designed exactly to help the users who neither are experts in the field, nor have the time and patience to evaluate many possibilities.

Furthermore, even when "faster than 2 GHz and cheaper than $1250" returns 5 products, the list may exclude some good candidates. For example, there may be a product which is a little bit slower than 2 GHz, but much cheaper than all the five products in the list, or a product which is a little bit more expensive than $1250, but much faster than the five products in the list. No matter how the query condition is determined, such possibilities always exist.

This is the case, because binary query cannot handle trade-off among multiple preferences. When a user preference contains multiple components, it is normal, rather than exceptional, for them to conflict with each other. For example, in almost all shopping situations, users prefer products which are cheap and with high quality, even when we all know that these two preferences usually cannot be optimized together. Without a quantitative measurement on "degree of satisfying" or something like that, trade-off becomes arbitrary.

What it means is that even though database query is still the most often used technique for the users to request for data items, it does not provide much help for selective retrieval, because of the binary expressions used in the query language.

If we want to directly process the preference defined before, we need conditions that different values satisfy to different degrees. Under this consideration, the most obvious solution is to apply fuzzy logic [Zadeh, (1975)] into database query, which leads to the idea of "fuzzy database" [Yang and Lee-Kwang, (2000); Yang et. al., (2001)].

A recommendation system based on fuzzy logic can be designed according to the "recommendation procedure" defined previously, where the processing of the constraints are just like in conventional database. The preferences are specified using "linguistic variables", such as "$\approx fast$" for speed, and "$\approx$ *cheap*" for price. Each linguistic value corresponds to a fuzzy set, with a membership function to calculate the score for each attribute value. Finally, the total score of a product is the minimum of all the individual scores, because in fuzzy logic the (default) function for "AND" (i.e., conjunction of conditions) is "*min*".

In this way, we can indeed get a recommendation system satisfying our previous description, and it no longer suffers from the problem in conventional database discussed above. Since the idea of fuzzy logic has been well known for many years, and this application is not that difficult, why have not we seen many such systems?

Among all reasons, a major issue is the design and maintenance of the membership functions. According to the common interpretation of membership function, it measures a subjective opinion on the "compatibility" between the linguistic value and numerical values [Zadeh, (1975)]. For example, it can be said that "The membership of $1000 to *cheap* is 0.9", "The membership of $1250 to *cheap* is 0.85", and so on. Therefore, the designer of the recommendation system is responsible for specifying membership functions for every linguistic value.

Furthermore, it is well known that the same linguistic value, such as "cheap", corresponds to very different membership functions when used on different categories. Clearly, what can be labeled as "cheap" can have radically different prices for categories like notebook computer, super computer, house, notebook, pencil, and so on. Consequently, a separate membership function is needed for each linguistic value on each category. Even if that can be provided, there is still trouble — these functions may need to be adjusted from time to time. For example, what is considered as "a cheap notebook computer" two years ago is no longer "cheap" according to the current situation in the market.

In summery, though it is possible to build a recommendation system using fuzzy logic, the design and maintenance process is complicated (because of the *relative* nature of the membership functions), and the recommendations are hard to justify (because of the *subjective* nature of the membership functions).

All these problem comes from a common root, that is, though fuzzy logic *represents* and *processes* fuzziness in various ways, it does not properly *interpret* fuzziness. Since this problem has been analyzed in detail in a previous publication [Wang, (1996)], we will not repeat the discussion here. We just want to say that because of this problem, fuzzy logic does not work well in recommendation systems, except in special situations where only a few membership functions are needed, and they do not change over time.

5.3 A new approach

The main content of this chapter is to introduce a new approach for recommendation system design. This approach is similar to the fuzzy-logic approach discussed above, except that the membership functions are automatically generated from the available data by an algorithm, therefore the results are well justified, and the design and maintenance of the system is relatively easy.

This approach of recommendation is a by-product, a practical application, of the author's research on general-purpose intelligent system [Wang, (1995)]. In the following, we only introduce the aspects of the research that is directly related to the recommendation procedure. For how this procedure is related to the big picture of artificial intelligence, please visit the author's website at `http://www.cis.temple.edu/~pwang/` for related materials.

Again, here we treat the constraints in the same way as in conventional database. After the M candidates are returned by the query, the recommendation task is treated as a task of selecting "good" instances of a given concept among given candidates.

Concretely, a preference vector $p = <p_1, p_2, \cdots, p_t>$ defines a concept C_p, the instance of which is what the user is looking for. For example, "fast and cheap notebook computers" is such a concept, and "flights (from one given city to another) leaving around 9 AM (on a certain day) and arrive as early as possible" is another. Such a concept is defined by the preference vector, as a set of properties. That is, "fast and cheap notebook computers" is a subset of notebook computers that have properties as being "fast" and being "cheap".

According to the model of categorization used in this approach, in the recommendation process the concept C_p is defined collectively by all the given properties, each of which contributes to the meaning of the concept, and none of which is sufficient or necessary for the membership by itself. Furthermore, whether an attribute value satisfy a corresponding property is a matter of degree.

Unlike in fuzzy logic, where degree of membership is a subjective judgment that cannot be further analyzed, in our approach the "score" of each value for a given preference is the *proportion of positive evidence among all evidence*, that is, $s = w^+/(w^+ + w^-)$, where w^+ and w^- are the amount of *positive* and *negative* evidence, respectively, for the preference to be satisfied by the value.

How is evidence defined and measured? Let us start from a concrete example. If the price of a notebook computer is \$1250, then to what extent it belongs to the concept of "cheap notebook computers"? According to our theory [Wang, (1996)], such a question cannot be answered without a "reference class", that is, it depends on the answer of another question: "Compare to what?". Adjectives like "cheap" get their meaning from relations like "cheaper than", though the object of the comparison is often omitted in the expression. In the recommendation process, we assume that the default objects of comparison are the other candidates. Therefore, we interpret "cheap" as "cheaper than the other candidates". Since usually there are multiple candidates, and some of them may be cheaper, and others more expensive, than the product under consideration, whether it is "cheaper than the other candidates" is usually a matter of degree.

If there are M candidates that satisfy the constraints, then they are used to score one another for the preferences. To decide the score for a \$1250 computer to be labeled as "cheap", the other $M - 1$ candidates are compared to it one by one in price, where more expensive ones are counted as positive evidence, and cheaper ones as negative evidence, for the labeling (candidates with the same price provide no evidence). The total amount of evidence is the sum of the amount of positive evidence and the amount of negative evidence. Therefore, among the M candidates, if there are m_1 of them are more expensive than \$1250, and m_2 of them are cheaper than \$1250, then the score for a \$1250 computer to be labeled as "cheap" can be simply taken as $m_1/(m_1 + m_2)$ [Wang, (1996)]. Especially, the cheapest candidate gets a score 1.0, and the most expensive one get 0.0, for the given preference.

The above approach can be applied to a preference as far as the values of the corresponding attribute form a total order, even if the values are not numerical. The following is a general definition of evidence in the recommendation process:

- When a value d_{ij} is evaluated according to a preference p_j of the form "$\approx v$" ("similar to v"), if another value d_{kj} (of another candidate) is farther away from v than d_{ij} is, it is positive evidence; if d_{kj} is closer to v than d_{ij} is, it is negative evidence.
- When a value d_{ij} is evaluated according to a preference p_j of the form "$\ll$" ("has a small value"), if another value d_{kj} (of another candidate) is larger than d_{ij}, it is positive evidence; if d_{kj} is smaller than d_{ij}, it is negative evidence.
- When a value d_{ij} is evaluated according to a preference p_j of the form "$\gg$" ("has a large value"), if another value d_{kj} (of another candidate) is smaller than d_{ij}, it is positive evidence; if d_{kj} is larger than d_{ij}, it is negative evidence.

After separating positive and negative evidence from non-evidence among the other candidates, their number can be used as the above m_1 and m_2, respectively, then from them the score of the given value can be calculated according to the previous formula.

For a given attribute, if its values are not only comparable, but also numerical, sometimes the distance between values should be taken into account when scores are calculated. For example, to evaluate the score for \$1250 to be labeled as "cheap", the existence of a \$750 and a \$1200, as the prices of other candidates, are very different. Though both are negative evidence, the former is clearly a much "stronger" one than the latter. For this situation, a more meaningful way to calculate the amount of evidence is to give each piece of evidence a "weight", which is the difference of that value and the given value. For the above case, the weights of the two pieces of evidence are 1250 - 750 = 500 and 1250 - 1200 = 50, respectively. Now the amount of (positive and negative) evidence m_1 and m_2 are weighted sum of pieces of evidence.

For a product d_i, after its attribute values get their scores as a vector $< s_{i1}, s_{i2}, \cdots, s_{it} >$ according to a given preference $p = < p_1, p_2, \cdots, p_t >$, the next step is to combine them into a total score s_i, which represents the membership for the product to be an instance of the concept C_p. Here we treat each preference p_i as an independent channel to collect (positive and negative) evidence for the membership relation. Therefore, evidence collected in each channel should be pooled together. As a default rule (assume all the preferences are equally weighted and each score is obtained from the same number of comparisons), the total score is simply the average of the individual scores, that is, $s_i = (\sum_{j=1}^{t} s_{ij})/t$.

Now let us give a complete and general algorithm for the recommendation procedure described above.

The database DB is a collection of data items, each of which is a vector $d_i = < d_{i1}, d_{i2}, \cdots, d_{it} >$, in which each d_{ij} is the value of d_i on an attribute A_j. In other words, DB is a matrix, where each row is a data item, and each column corresponds to an attribute.

A recommendation request r consists of two components, a constraint vector $c = < c_1, c_2, \cdots, c_t >$ and a preference vector $p = < p_1, p_2, \cdots, p_t >$. Each c_j has the form "$relation_j\ v_j$", where v_j is a constant value, and $relation_j$ is one of the following relations: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$. The evaluation of c_j against a value d_{ij} should return 1 (*true*) or 0 (*false*). Each p_j has the form '$\approx v_j$", "$\ll$", or "$\gg$". The evaluation of p_j against a value d_{ij} should return a real number in [0, 1].

For a given DB and a given r, the recommendation procedure, for a top-N list of data items satisfying r in DB, is the following:

(1) Translate c into a (conventional) database query on DB, and get the candidate set, which includes all data items in DB satisfying c.
(2) Assume the size of the candidates set M. If $M \leq N$, then recommendation is unnecessary, and the procedure may ends here. Otherwise the procedure continues.
(3) Apply the user preference $p = < p_1, p_2, \cdots, p_t >$ to each data item d_i, and get a vector of scores $< s_{i1}, s_{i2}, \cdots, s_{it} >$, where s_{ij} is the return value of applying preference p_j to value d_{ij}.
(4) Let the total score of a candidate d_i to be $(\sum_{j=1}^{t} s_{ij})/t$.
(5) Select the top N data items according to their total scores, and return them as the recommendation to the user. As options, the total score of each may be displayed, and they may be sorted according to their total scores.

In the above step (3), for each value d_{ij} and preference p_j, the score s_{ij} is the ratio of positive evidence among all evidence, that is, $s_{ij} = w^+/(w^+ +$

w^-). The (positive and negative) evidence is collected by comparing d_{ij} to each d_{kj} in the candidate set, as the following:

- p_j has the form "$\approx v_j$": d_{kj} is positive evidence if $|d_{kj} - v_j| > |d_{ij} - v_j|$. d_{kj} is negative evidence if $|d_{kj} - v_j| < |d_{ij} - v_j|$. The weight of the evidence is $||d_{kj} - v_j| - |d_{ij} - v_j||$.
- p_j has the form "$\ll$": d_{kj} is positive evidence if $d_{kj} > d_{ij}$. d_{kj} is negative evidence if $d_{kj} < d_{ij}$. The weight of the evidence is $|d_{kj} - d_{ij}|$.
- p_j has the form "$\gg$": d_{kj} is positive evidence if $d_{kj} < d_{ij}$. d_{kj} is negative evidence if $d_{kj} > d_{ij}$. The weight of the evidence is $|d_{kj} - d_{ij}|$.

The last two cases are the special cases of the first with v_j to be v_{min} and v_{max}, respectively.

If the values of each attribute form a total order (so that "$>$" and "$<$" are defined between any pair of them), but the distance between them are not defined, then the definition of evidence for the first case is modified as the following: d_{kj} is positive evidence if $d_{kj} > d_{ij} \geq v_j$, or $d_{kj} < d_{ij} \leq v_j$. d_{kj} is negative evidence if $d_{ij} > d_{kj} \geq v_j$, or $d_{ij} < d_{kj} \leq v_j$. Furthermore, in all the three cases, each piece of evidence is equally wighted.

5.4 Discussion

The above recommendation procedure is based on the opinion that selection among candidates are usually carried out according to multiple relatively defined criteria, and the goal is to achieve an overall optimum, which does not always coincide with optimum on each criteria. Recommendation helps selection by reducing the number of candidates that need to be presented to the user according to given selection criteria.

Most shopping websites still use conventional database query to carry out the selection process. As a result, non-expert users have to spend lots of time in fine tuning their query to get a desired number of candidates for the final comparison and selection. Compared to that process, the recommendation procedure introduced in this chapter has the following advantages:

- Both (binary) constraints and (fuzzy) preferences are allowed, where the former is expressed in absolute term, while the latter in relative term. This is a more natural way to set selection criteria for most users, especially for users who are not expert in the field. For them, what really matters is often the relative value, not the absolute value, of a data item on an attribute.
- Trade-off and compromise among multiple preference are allowed and

supported. Actually, all difficult selection problems happen in the cases where different criteria have to be balanced against each other. Using the concept of evidence (for membership relation), the above algorithm maps values of different attributes (with different units) into a common (unit-free) dimension.

- By presenting a top-N list to the user, the selection process is simplified without losing promising candidates. Still, the user can bring new factors (that are not in the recommendation request) into account in the final stage of the selection.

Compared to the similar solution based on fuzzy logic, this recommendation procedure has the following advantages:

- The degree of membership for an instance to belong to a concept is no longer a subjective opinion, but a measurement justified according to a theory on cognition and intelligence [Wang, (1995); Wang, (1996)].
- The scores are determined by the available data according to a domain-independent algorithm. Consequently, there is no need to manually design and maintain the membership functions. Instead, such functions are automatically learned from the data, and so are adaptive to the changes in data.

Because of the adaptive nature of the membership function, the system treats "cheap computer" and "cheap notebook" with different standards, simply because the available instances in these two categories have different price distributions. When a new product with the lowest price is added into the system, all other products in the same category automatically become "more expensive", as the result of comparing to it. This is closer to how the human mind works in these situations. Also, this "maintenance-free" solution is easier to be used in practical situations.

What we have introduced so far is the core procedure of a recommendation system, which is simple and domain-independent, and can be extended and customized for various purposes. When this approach is applied in a concrete system, there are still many additional issues, which we will only briefly mention here:

- For certain applications, the preferences on different attributes should not be treated equally. In such a situation, the final score for a data item can be changed from the average of the individual scores to the weighted sum of the individual scores. The weights of the attributes can either be specified by the user as part of recommendation request, or take default values determined by the designer according to the nature of each attribute.

- Some changes should be made in the algorithm to handle incomplete or uncertain data. For example, such data should make the corresponding score "weaker" in the recommendation decision. No matter what data is involved, its net effect should be reflected in the amount of evidence for the membership function.
- The constraint processing part can be simply replaced by the standard relational database query mechanism. In this way, the processing of preference can be designed as a frond-end of database, or even as an extension of SQL.
- In the previous procedure, we assume that there is a total order among the values of an attribute. If that is not the case, the recommendation system still can work after some domain-specific modification. For example, in a flight reservation system, if the (departure or arrival) airport is not part of the constraint, but part of preference, then nearby airports should be taken into consideration. In this case, the "degree of similarity" between two airports can be determined by distance, easiness of access, and so in.
- The (preference-based) recommendation technique introduced above can be combined with other recommendation techniques mentioned at the beginning of the chapter. For example, expert opinions, peer reviews, and so on, can all be taken as evidence collected from different channels for the same membership relation.
- The recommendation system can be connected to an intelligent reasoning system, such as the one from which this approach comes [Wang, (1995)]. Since both systems are based on the same theoretical foundation, the reasoning system can use background knowledge to evaluate similarity relations between non-numerical values of attributes, then let the recommendation system use them in the evaluation of scores.
- The recommendation system can be further personalized by remembering and generalizing user preferences from past requests, and using these learned parameters in future recommendations.
- For advanced users, the system can allow them to override the default functions used in each stage of the recommendation process (for the calculation of degree of similarity, amount of evidence, individual scores, total scores, final recommendations, and so on) with their specially designed functions.
- Since the final result of recommendation is the top-N list, not the absolute values of concrete scores, various approximation algorithms can be used in each stage of the process, especially for the collection of evidence. These algorithms can improve the efficiency of the system, without losing the quality of the result too much.

In general, we believe that the preference-based recommendation technique defined in this chapter can greatly improve the efficiency of on-line shopping and other information selection processes, and make the huge amount of available data more accessible and manageable for common users, especially the users without expert knowledge.

Bibliography

Jameson, A., Konstan, J., and Riedl, J. (2002). *AI Techniques for Personalized Recommendation:* Tutorial presented at AAAI.
URL: `http://www.dfki.de/~jameson/aaai02-tutorial/`

Kautz, H. (editor) (1998). *Recommender Systems*, AAAI Technical Report.
URL: `http://www.aaai.org/Press/Reports/Workshops/ws-98-08.html`

Wang, P. (1995). *Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence*. PhD thesis, Indiana University.
URL: `http://www.cogsci.indiana.edu/farg/peiwang/papers.html`

Wang, P. (1996). The interpretation of fuzziness. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:321-326.

Yang, J. and Lee-Kwang, H. (2000). Treating Uncertain Knowledge-Based Databases. *Academic Press International Theme Volumes on Knowledge-Based Systems Techniques and Applications*, 1:327-351.

Yang, Q. *et. al.* (2001). Efficient Processing of Nested Fuzzy SQL Queries in a Fuzzy Database. *IEEE Transactions on Knowledge and Data Engineering*, 13:884-901.

Zadeh, L. (1975). The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*, 8:199-249, 8:301-357, 9:43-80.