

RANKING DATABASE QUERIES USING USER FEEDBACK: A NEURAL NETWORK APPROACH

Ganesh Agarwal, Nevedita Mallick, Silvio Tannert, Srinivasan Turuvekere

Guided by: Prof. ChengXiang Zhai

CS511 Project - Advanced Database Management Systems, Fall 2006

Dept. of Computer Science, University of Illinois at Urbana-Champaign

Abstract

Presently, Internet websites serving structured data allow the user to perform search based on simple equality or range constraints on data attributes. However, to begin with, the user may not know precisely what is "desirable" to him to be able to express it accurately in terms of primitive equality or range constraints. As, in most websites, the results provided to user are sorted with respect to values of one particular attribute, it presents the user with "searching for a needle in a haystack" problem because the user's notion of "interesting" houses is generally a function of multiple attributes.

Our work is among the earliest in the literature to (i) support a family of functions involving multiple attributes to rank the tuples, and (ii) to incorporate user feedback to learn the user's notion of "interestingness" with the help of a neural network and improve the results provided to the user. The user feedback driven approach is effective in modeling a user's intuitive sense of desirability of a tuple, a notion that is otherwise near impossible to quantify mathematically.

1 Introduction

1.1 Motivation

A relational database addresses the problem of querying data from structured data. However, it only supports the retrieval of all data matching a particular filtering criteria set by the user. It also supports the limited ordering of the result sets. In real world use-cases, however, it is very rare that a user will exactly know the proper criteria for his retrieval and will usually have some preferences. Many times these preferences can be conflicting as well. Moreover,

the result of a user query may be huge and may not be ordered according to the user's interest. There is a need for a mechanism by which a user can pose a query and get the result ranked according to his interest. This mechanism should also support user feed back. So after the user gets the top-K results, he should be able to point which tuples were of interest to him and which tuples he wants to give more weight to. Then, depending on this feedback, the result set should be changed (along with the ranking of the result set) reflecting the user's feedback.

1.2 Related Work

There has been recent interest in ranking the results of database query according to some function that models user preference and return the top-K tuples. Research in this area has spanned addition of new relational algebra constructs to support tuple ranking, query optimization for efficient execution of top-K queries, and middleware-based strategies that use traditional SQL constructs and try to minimize the number of database tuples retrieved. In this section, we give a few examples of these. Li et al. [4] introduce a systematic and principled framework, by extending relational algebra and query optimizers, to support ranking as first-class construct in relational database systems. In this work, the ranking function is specified in the query. In contrast, our approach learns the ranking function through user feedback.

Under the top-K queries model used in [2], users specify target values for the attributes of a relation, and expect in return the tuples that best match these values. The paper outlines a

family of strategies to map a top-K query into a traditional selection query that a database engine can process efficiently. The goal of such mapping strategies is to get all needed tuples (but minimize the number of retrieved tuples) and thus avoid "restarts" to get additional tuples.

Agrawal et al. [1] does support fuzzy querying. However, it assumes and tries to optimize the queries by exploiting existing indexes and database heuristics on cardinality & selectivity of relevant attributes. Also, this paper tries to automatically infer the similarity between attribute values using the attribute distributions, which may not be aligned with users notion of similarity.

1.3 Solution Overview

Our work tries to fill the gap between "Automated DB ranking using query and DB heuristics" and "top-K querying and ranking using a fixed user preference function". Our approach is to take a minimum amount of initial input from the user, in the form of constraints on attributes and relative weightings of attributes, relax these to some extent to obtain a working set of tuples from the database, and incorporate a machine learning based framework to learn the user's preferences (iteratively) and present the top-K tuples in ranked order. We use a Neural Network based technique to learn by example through user feedback in the form of desirability ratings on ranked tuples from previous iterations. The once learned preferences can then be used in future search queries and can also be adopted to different needs.

We built a Java based implementation (middleware) of our technique that can interface with any given backend database system. Using real estate data obtained from a realtor database, we have fine tuned the parameters of our learning framework for this particular application and evaluated the effectiveness of our system in learning what the user is looking for.

2 Domain

Our middleware builds on realtor.com search domain by using more advanced search methodologies and taking user feedback into account. To get the initial working set of results, the domain realtor.com is queried by providing the HTML address of the results page which is generated using the input query from the user. The HTML page obtained from the domain is parsed into a tree structure with the nodes in a nested fashion, where each node represents a tag and its contents. A filter is then used to extract information about the tuples from the page. Before parsing the information about tuples, the number of tuples in the search results is parsed which helps in calculating the number of result pages to be parsed. With the help of a node iterator, information about the tuples is parsed and assigned to the appropriate attribute of the tuples. Tuples with missing data for certain attributes are handled as follows. For a numerical attribute, the missing data is assigned the mean value of the range given by the user in the query for that attribute. For a categorical attribute, it is assigned to a new class label "Unknown" which indicates missing value for that attribute.

Moreover, schema matching was performed although the tuples were from the same web database. For example, some tuples had an attribute "Age", whereas other tuples had a derived attribute "Year Built" instead. Hence, the attribute values had to be transformed accordingly to have uniform schema and unit of representation.

3 The Neural Network Approach

We have decided to use a Neural Network based approach to build an initial framework, which will rank the tuples (initially) according to a weighted average of the user's provided input. Later, the network will learn and adjust itself based on the user's feedback. The learning time

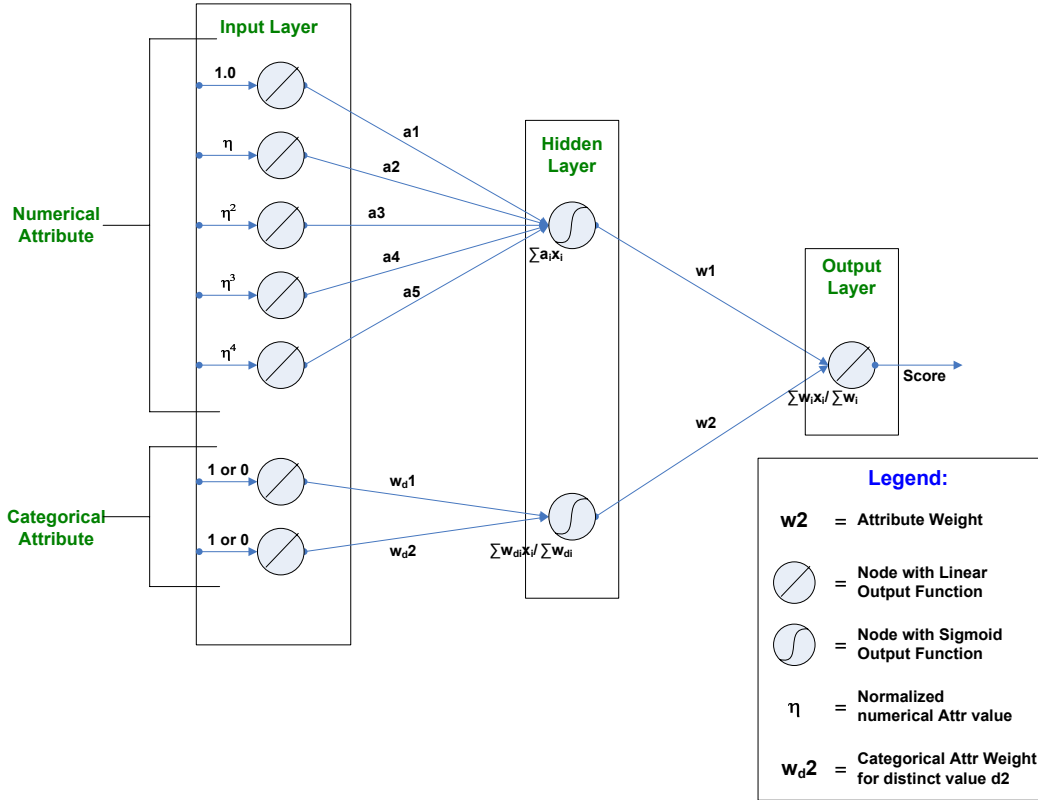


Figure 1: The Neural Network architecture

is linear in the number of examples and its prediction time is independent of the number of examples. The Neural Network has **three layers**: *input layer*, *hidden* and *output layer*. The input nodes correspond to the attribute inputs. For numerical attributes, the input nodes represent the different powers of the input attribute value. The linear combination of these input nodes represents a degree four polynomial *desirability function* for the corresponding attribute value. For categorical attributes, the input nodes represent the desired distinct values corresponding to the attribute. All input nodes for a particular attribute are connected to their respective hidden node. Hence, one hidden node corresponds to the normalized desirability value for an attribute.

We use a Sigmoid function to normalize the desirability value to the range $[0, 1]$. The connections and corresponding weights in the first level (between input and hidden layer) signify the intra-attribute weights/coefficients. The

connections and corresponding weights in the second level (between hidden and output layer) signify inter-attribute weights. The output nodes correspond to the scoring function and hence will give the final score for each tuple. The initial network is built on the basis of the user provided ranges/values and weights for the attributes (cp. Fig. 1).

4 Implementation Details

4.1 Numerical Attributes

If the user has provided a desirable range $[l_i, u_i]$ for attribute A_i . We relax the range and fetch a suitable initial working set of tuples. We will fetch tuples in the range $[l_i^*, u_i^*]$ for attribute A_i . We relax the range depending on the weights between attributes and the length of interval $[l_i, u_i]$ provided by the user.

The increase in range interval:

$$i = (1 - w_i) * (u_i - l_i)$$

Where w_i = weight of attribute A_i relative to other attributes. Thus, attributes with higher weights are relaxed to lesser extents.

$$l_i^* = l_i - \Delta i$$

$$u_i^* = u_i + \Delta i$$

We define a function $P(t(A_i) = x_i|C_i)$ to denote a *desirability function*, which gives the desirability value for a tuple t , where attribute A_i has value x_i and satisfies condition C_i . The desirability distribution graph in Fig. 2 shows the initial distribution.

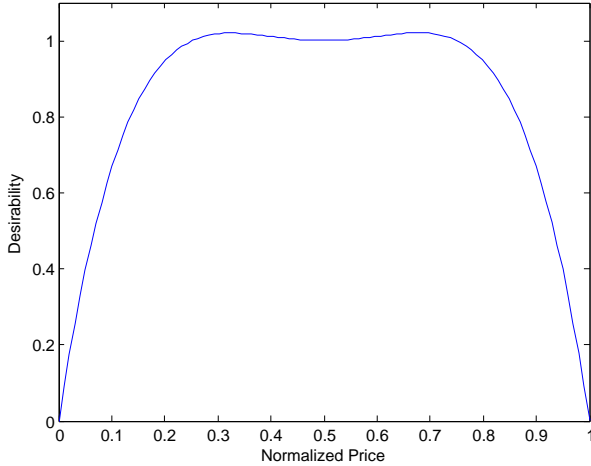


Figure 2: Initial desirability function

The desirability function is a degree four polynomial:

$$D(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 \quad (1)$$

Initially, we plot the graph with values on the x axis as the normalized values on the relaxed range (relaxing done as described above). In order to get the coefficients of the degree 4 curve, we take the following pairs of (x, y) values to solve the above equation:

$$(0, 0) \quad (0.25, 0.99) \quad (0.5, 1) \quad (0.75, 0.99) \quad (1, 0)$$

The reason behind taking these values is that the curve will have high desirability values in

the range provided by the user. This is evident from the steep slope which the curve takes initially and then almost flattens around the middle of the range provided by the user (note that the curve shown has the x axis normalized to relaxed range).

The coefficients of the desirability function correspond to the weights of the connections between the input nodes and the hidden node of the Neural Network. A set of 5 input nodes (whose connections with the hidden node correspond to one of the coefficients) will connect to one hidden node – the weighted sum at the hidden node will thus correspond to the probability distribution function for one of the attributes.

Each node in the Neural Network contains two functions: aggregation function that aggregates the inputs into the node and output function that normalizes the aggregation output in a desired range. We use a *Sigmoid function* to normalize the desirability function values at each hidden node. So the output of the hidden node always lies in the range $[0, 1]$. One hidden node corresponds to one attribute. For a hidden node corresponding to attribute A_j having value x , the function is defined as:

$$H_j(x) = \left(\frac{1}{1 + e^{-a(D_j(x)-h)}} \right) \quad (2)$$

Here, a and h are constants. In our current implementation, based on preliminary testing, we have determined that a value of ($a = 14$) and ($h = 0.5$) provides high variability and distinction within the attribute range.

4.2 Categorical Attributes

For categorical attributes, since there is no ordering of categorical values we cannot fit any desirability function. However, there may still be some similarity/desirability value associated with for each categorical value. We get this value initially from user. These desirability values are stored as the weights of the connections between the respective first level input nodes

and the corresponding hidden node. The number of input nodes for categorical values can be either:

- a) The number of distinct values specified by the user.
- b) The maximum number of distinct values based on the categorical attribute itself. In this case, the weights of the connections connecting distinct values not specified by the user are initialized to 0.

For each tuple, therefore, the input is really a unit vector of n dimensions (n distinct values), e.g. $[1.0, 0, 0, 0]$ or $[0, 1.0, 0, 0]$. This is because a tuple can have one possible distinct value for a particular attribute at any given time.

For categorical attributes, we tried two different approaches: a) use a normalized weighted sum as the aggregation function and linear function as the node output function and b) use a weighted sum as the aggregation function and a Sigmoid function as the node output function. We used the latter approach as it yielded slightly better results.

4.3 Tuple Score

The overall scoring function yielding the output of the Neural Network is a linear function of the hidden nodes' output normalized with respect to the weights w_j of the attributes. For a tuple t , it can be defined as:

$$S(t) = \frac{1}{\sum_{j'=1}^n w_{j'}} \left(\sum_{j=1}^n w_j H_j(t(A_j)) \right) \quad (3)$$

5 Learning using User Feedback

5.1 Gradient Descent

Suppose the user has provided feedback for m tuples t_1, t_2, \dots, t_m . The user will give a score

on a scale of 0 to 1 for these tuples. Suppose there are n attributes A_1, A_2, \dots, A_n .

Let $t_i(A_j)$ denote the value of the attribute A_j of tuple t_i . As before, w_j will be the weight of the attribute A_j provided by the user. It is the weight of the connection between the hidden node corresponding to the attribute A_j and the output node of the Neural Network.

The *mean squared error (MSE) function* on the set of m tuples to which the user has provided feedback can be defined as:

$$E(\vec{a}, \vec{w}) = \frac{1}{m} \sum_{i=1}^m [S(t_i) - US_i]^2 \quad (4)$$

Here, US_i is the user's provided score for tuple i . Since the target function and the number of feedback tuples are fixed, E is only a function of the weights a_{kj} and w_j of the Neural Network. The aim of our learning algorithm will be to adjust the weights using a *gradient descent method* so as to minimize the mean squared error E . The steps will involve the following computations:

Partial differentiation of MSE with respect to the coefficients of the attributes:

$$\frac{\partial E}{\partial a_{kj}} = \frac{1}{m} \sum_{i=1}^m 2[S(t_i) - US_i] \frac{\partial S(t_i)}{\partial a_{kj}} \quad (5)$$

where

$$\frac{\partial S(t_i)}{\partial a_{kj}} = \frac{w_j}{\sum_{j'=1}^n w_{j'}} \frac{\partial H_j(t_i(A_j))}{\partial a_{kj}}$$

$$\frac{\partial H_j}{\partial a_{kj}} = a H_j(t_i(A_j))(1 - H_j(t_i(A_j))) \frac{\partial D_j(t_i(A_j))}{\partial a_{kj}}$$

$$\frac{\partial D_j(t_i(A_j))}{\partial a_{kj}} = (t_i(A_j))^k$$

Partial differentiation of MSE with respect to the weights of the attributes:

$$\frac{\partial E}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m 2[S(t_i) - US_i] \frac{\partial S(t_i)}{\partial w_j} \quad (6)$$

where

$$\frac{\partial S(t_i)}{\partial w_j} = \frac{(\sum_{j'=1}^n w_{j'}) H_j(t_i(A_j))}{\left(\sum_{j'=1}^n w_{j'}\right)^2} - \frac{\sum_{j'=1}^n w_{j'} H_{j'}(t_i(A_{j'}))}{\left(\sum_{j'=1}^n w_{j'}\right)^2}$$

The coefficients of each attribute will then be adjusted as follows:

$$a_{kj}^{new} = a_{kj} - \epsilon \frac{\partial E}{\partial a_{kj}} \quad (7)$$

And the weights of the attributes will be adjusted as follows:

$$w_j^{new} = w_j - \epsilon \frac{\partial E}{\partial w_j} \quad (8)$$

The value of ϵ represents the contribution of learning in a single batch of feedback (rating of m tuples) by the user. We propose to decrease the value of ϵ geometrically over successive learning iterations. This is an effort to model the fact that the impact of user assisted learning will be significant in the first few iterations. Thereafter, small incremental refinements will be made to the already learned knowledge about user preferences. We have tuned the initial value of ϵ by experimentation. We iterate over the above procedure and keep adjusting the weight until the gradient descent converges to a minimum value.

5.2 Back Propagation

We have adopted *back propagation* [3] strategy because it utilizes local computation at each node of the Neural Network and hence simplifies the computation involved in the gradient descent method. In a setting where it is necessary to speed up the training of the Neural Network on huge volumes of data (unlike ours), back propagation has the advantage of being amenable to parallel implementation on multiple processors, where the nodes in the Neural Network are partitioned and assigned to different processors.

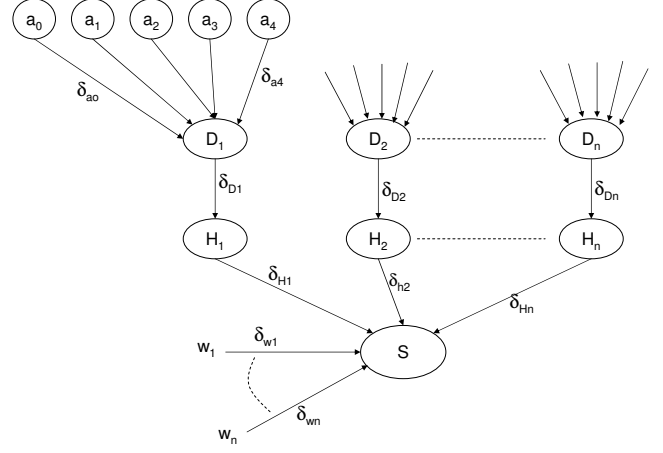


Figure 3: Back Propagation

Consider Fig. 3. For each tuple t_i , $i = 1 \dots m$ the partial derivatives, which will propagate back through the nodes will be as defined below. For simplicity of notation, we will sometimes drop the tuple argument t in the expressions for the partial derivatives.

$\delta_S = \delta_{S(t)}$ = partial derivative of MSE E w.r.t output function $S(t)$ for tuple t .

$\delta_{H_j} = \delta_{H_j(t(A_j))}$ = partial derivative of MSE E w.r.t hidden node function H_j evaluated on value of attribute A_j of tuple t .

$\delta_{D_j} = \delta_{D_j(t(A_j))}$ = partial derivative of MSE E w.r.t desirability function D_j evaluated on value of attribute A_j of tuple t .

δ_{w_j} = partial derivative of MSE E w.r.t weight given to attribute A_j .

$\delta_{a_{kj}}$ = partial derivative of MSE E w.r.t k^{th} coefficient of desirability function D_j for attribute A_j .

$$\delta_S = 2(S(t) - US) \quad (9)$$

$$\delta_{H_j} = \delta_S \frac{w_j}{\sum_{j'=1}^n w_{j'}} \quad (10)$$

$$\delta_{w_j} = \delta_S \frac{\left(\sum_{j'=1}^n w_{j'}\right) H_j - \left(\sum_{j'=1}^n w_{j'} H_{j'}\right)}{\left(\sum_{j'=1}^n w_{j'}\right)^2} \quad (11)$$

$$\delta_{D_j} = \delta_{H_j} H_j(t(A_j)) (1 - H_j(t(A_j))) \quad (12)$$

$$\delta_{a_{jk}} = \delta_{D_j}(t(A_j))^k \quad (13)$$

The overall adjustment of weights will evaluate the above quantities for each of the m feedback tuples t_1, t_2, \dots, t_m using m back propagation computations. The coefficients of the desirability function for each attribute will be updated as follows:

$$a_{kj}^{new} = a_{kj} - \epsilon \frac{1}{m} \sum_{i=1}^m \delta_{a_{kj}}(t_i) \quad (14)$$

And the weights of the attributes will be adjusted as follows:

$$w_j^{new} = w_j - \epsilon \frac{1}{m} \sum_{i=1}^m \delta_{w_j}(t_i) \quad (15)$$

6 Test Results

We tested our system for various cases. Firstly, the general case was taken to see how our system works for simple ranking based on one particular attribute (price). Given a query for a particular range of price, the results initially obtained are not sorted in increasing or decreasing order of price. They are ranked as per the initial desirability curve for the attribute price as shown in Fig. 2. However, when the user gives positive feedback for some tuples with lower price and negative feedback for some tuples with higher price, the network is able to learn the user’s interest in lower values of price and reflects it in the new modified desirability curve for the attribute price shown in Fig. 4. The network now gives higher scores to lower priced houses and lower scores to higher priced houses and the user is provided new users reflecting this. The curve shows that the price has been normalized from 0 to 1 where 0 represents the extended minimum value of price and 1 represents the extended maximum value of the price. It should be noted that the curve is not decreasing for all values of price but it decreases from normalized value of price as 0.2 because the normalized value of 0 to 0.2 represents prices in the extended range but not the

original range in the user’s query. However, if the user provides feedback which indicates that the user likes houses outside the original range, then the desirability curve would be modified accordingly by the network to reflect higher desirability for values outside the original range in query. Importantly, the network uses this knowledge for the user’s future searches.

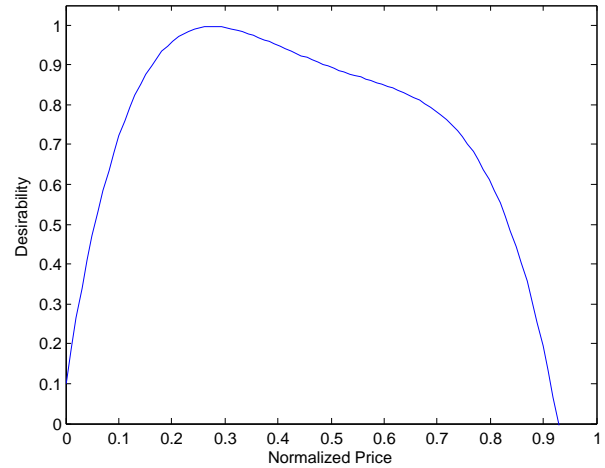


Figure 4: Desirability function after learning process

However, the main strength of our system is when the user’s interest in houses is based on their values for multiple attributes. Let us examine two user cases.

6.1 Case 1

Assume a case where a couple wants to buy a house. The husband first enters the query with higher weight to area ranging from 1,000 to 2,000 sq. ft. and lower weight to age ranging from 1 to 50. However, on seeing the results, the wife puts her demand for newer homes and provides positive feedback for a single tuple with less age and negative feedback for 3 tuples with high. Now, they are presented with the houses with the new ranking. A list of top 20 houses with the above criteria was obtained. On comparison, as can be seen from Table 1, the results returned from realtor.com contained 6 of the houses in its top-20, our middleware had 13 before feedback and 17 after feedback.

It is to be noted that the other 3 tuples returned after feedback were ranked slightly below 20 overall and hence they were not negative results.

6.2 Case 2

Let us now consider a case where a person queries for houses in range of \$125,000 to \$175,000 with area ranging from of 1,000 to 2,500. His intention is to get single family homes with maximum area, for which he is willing to pay around \$150,000. A list of top-10 houses with the above criteria was obtained. On comparison, as seen from Table 2, it can be seen that the results returned from realtor.com contained 3 of the houses in its top-10, our middleware had 6 before feedback and 8 after feedback.

Through these examples, we can see that our system performs highly better than existing systems when the user wants to search for houses with choices on multiple attributes.

7 Future Work

A detailed testing can be done on the developed middleware to learn the best parameters for the neural network and to clearly classify some cases where the learning may be slow. We could also model the network to learn about the different correlation and dependencies between attributes; this information would help rank results even better and faster. We have chosen a simple degree 4 curve to represent the initial curve for all attributes. We could learn, how the system would behave for other different types of curves and find out the curve, which would work the best in most cases. We could develop better learning strategy for categorical attributes in cases when these attributes are related to each other or have some ordering between them. Finally, the architecture of the existing framework can be optimized for parallel calculations on multi processor ma-

chines, which enables faster processing of huge amounts of data.

8 Conclusion

The chief contributions of our work include focusing on the effectiveness of the ranking mechanism rather than the efficiency of the ranking like that of top-K ranking. The novelty in ranking mechanism is that it involves a weighted measure of multiple attributes rather than ranking it based on values of only one particular attribute. Another important contribution is the use of feedback in the search mechanism, which helps drastically to improve the quality of results given back to the user based on the user's perception of best results. The middleware developed is also very scalable as it does not take any initial knowledge of the attributes of the given domain and hence it can be transferred to work in different domains.

References

- [1] S. Agrawal, S. Chaudhri, G. Das, and A. Gionis. Automated Ranking of Database Query Results.
- [2] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems*, 27(2), June 2002.
- [3] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, July 1998.
- [4] C. Li, K. Chen-Chuan Chang, I. Ilyas, and S. Song. RankSQL: Query Algebra and Optimization for Relational Top-k Queries. In *2005 ACM SIGMOD Conference (SIGMOD 2005)*, pages 131–142, Baltimore, Maryland, June 2005.

BASELINE				REALTOR	OUR ALGORITHM	
BASELINE RANK	HOUSE NAME	AREA	AGE	REALTOR RANK	INITIAL RANK	RANK AFTER FEEDBACK
1	525 N HICKORY	1,371	1	35	45	4
2	2506 STRICKER LANE	2,345	2	43	8	1
3	3009 WEEPING CHERRY	2,335	4	36	5	2
4	1401 MITTENDORF	1,215	5	15	70	7
5	2109E PENNSYLVANIA	1,244	7	19	12	6
6	1308 W EADS	1,284	10	1	20	5
7	309 S NEW	1,666	16	18	14	3
8	1317 E HARDING DRIVE	1,715	19	17	46	8
9	1905 #2 CHRISTOPHER	970	22	41	77	31
10	2103 MONONA COURT	1,060	24	6	56	35
11	508 E MECHEURY	2,572	25	64	11	9
12	1738 WESTHAVEN	2,810	26	28	17	10
13	114 E PARK	2,500	26	55	18	11
14	801 BURKWOOD	2,390	26	62	9	17
15	1902 S GEORGE HUFF	2,372	26	58	80	21
16	715 W VERMONT	2,120	26	42	19	15
17	1806 GOLFVIEW DRIVE	2,044	26	48	15	12
18	2115 RANSOM PLACE	2,026	26	60	38	13
19	2507 LYNDHURST	2,014	26	34	13	14
20	2508 S LYNN	1,992	26	40	10	16
COUNT OF TUPLES WITH RANK WITHIN TOP 20 IN BASELINE RESULT				6	13	17

Table 1: Detailed results of test case 1

Note: As Realtor does not query for 2 zip codes simultaneously, the ranks are taken by combining their two ranked lists in a round-robin fashion.

BASELINE				REALTOR	OUR ALGORITHM	
BASELINE RANK	HOUSE NAME	PRICE	AREA	REALTOR RANK	INITIAL RANK	RANK AFTER FEEDBACK
1	1118 W PARK	\$145,000	2,160	31	16	5
2	611 W CLARK	\$164,900	1,920	7	2	1
3	603 W CLARK	\$149,800	1,827	18	9	2
4	2002 KAREN CT	\$154,900	1,725	14	8	16
5	309 S NEW	\$154,900	1,666	15	5	3
6	712 DEVONSHIRE	\$169,900	1,932	6	10	6
7	808 S FOLEY	\$177,000	2,160	32	28	15
8	505 S GARFIELD	\$175,000	1,771	4	25	10
9	1615 GLENN PARK	\$129,900	1,476	20	14	8
10	914 S LYNN	\$149,900	1,431	17	6	7
COUNT OF TUPLES WITH RANK WITHIN TOP 10 IN BASELINE RESULT				3	6	8

Table 2: Detailed results of test case 2

Note: Here only one zip code was taken.