RI 02012 June 2002 Computer Science/Mathematics

IBM Research Report

Conn-Eval : A Connectionist Method for Evaluating Multiple Attribute Items

Jayanta Basak **Manish Gupta**
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.
e-mail : bjayanta,gmanish@in.ibm.com

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

Abstract

The task of evaluating and ranking multiple attribute items is relevant in many different aspects of e-commerce including RFQ, negotiations, personalized catalogues, profiling and customer modeling. Usually a parametric utility function is considered which is a linear weighted sum of the individual attribute utilities, and the weights of the individual attributes are estimated by minimizing the discrepancy between the predicted order and the true order. However, the individual attribute utilities may not be linearly independent and they may not be known a priori. In this paper, we propose a nonlinear model for ranking the multiple attribute items and their subsequent evaluation without assuming any independence between the attributes and any prior knowledge about individual attribute utility functions. A neural network (connectionist model) has been used at the core of the algorithm to learn and rank the items. Since the desired utility value for a bid is unknown, the usual techniques of function approximation cannot be employed in this paradigm, and a new objective function (error measure) is defined in this context. New rules are proposed for automatic selection of learning rate and prescriptions are made for selection of the architecture of the neural network for this task. New query-based sampling technique is also provided to improve the performance of the method. Experimental results illustrate the effectiveness of the method for ranking items with complex utility functions.

1 Introduction

The task of ranking multi-attributed items is relevant in many different aspects of e-commerce like auctions, Request for Quotes (RFQ), negotiation, advanced profiling and product catalogues. A subset of items having multiple attributes with partial or full ordering is given to an agent, and the task of the agent is to evaluate the attribute values of the items and extract (discover / find out) the embedded utility function such that the next set of items with unknown order can be ranked with this utility function. The task is straightforward and simply boils down to function approximation problem if the explicit utility function value is known. However, no explicit utility function value is usually known to the agent, rather a human evaluator can provide a pairwise comparison or ranking of the items.

For example, in an RFQ (request for quote), the received responses or bids need to be evaluated to select the winning bid or set of winning bids. It is equally important in fixing up negotiation strategies where each bidder needs to understand and evaluate the valuation of his counter party for the multi- attributed items. Similar problems persist in the domain of catalogue and on- line shopping assistants, where a set of products with multiple attributes are to be properly evaluated for a buyer and the set of "most interesting" or "most relevant" products are to be shown to the buyer. Similarly, in the case of advanced profiling in a store-front (B2C commerce), a merchant needs to configure optimal grouping of buyers for performing the strategic business decision making, where it is necessary to understand and evaluate the intrinsic customer utilities from their demographics and past behavior.

In all these tasks, which are essentially part of decision-support systems, the problem is to come up with a "good" algorithm to evaluate and rank multi- attributed items, and is in general, related to multi-attribute utility theory (MAUT) [1, 2, 3, 4]. In the development of multi-attribute utility theory (MAUT), parametric utility functions are considered to different levels of complexity including multilinear, multiplicative, and additive models. Solutions to the problem of evaluation of attributes are proposed for the additive models only where the utilities for individual attributes are considered to be independent [5, 6, 7, 8].

For additive utility function, evaluation of the utility and subsequent ranking of the items (bids in RFQ, customers in profiling) has been performed by formulating the problem as a problem of linear programming [5] in the design of a decision analysis system for e-sourcing. In this approach, a technique (WORA) for assessing the weights of the attributes is proposed by formulating the problem as a linear programming task. In order to improve online marketplaces, a multi-attribute resource intermediary (MARI) [8] has been developed which also uses additive model of the utility function.

Subsequently, in [6], a system called Q-Eval has been developed based on the linear program solver where the polytope defining the feasible solution region of an additive utility function model is iteratively reduced and queries are generated successively from the center of the polytope. In marketing research, a commonly used alternative approach is conjoint analysis [9] where attribute values are quantized and cards are designed with all possible quantized attribute values. The rank of a product in the complete ordering is used as its value and then an additive model

is found by regression analysis. Different extensions of conjoint analysis [9, 10] are also available in the literature.

Traditional auction software vendors [11] including Moai, Vertical Net, Ariba, and Clarus have added improvements to their software wherein the bidders (i.e., suppliers) have the ability to specify multiple attributes, such as quality and terms and conditions, etc allowing the bidders to differentiate themselves by other factors besides price. In all the above software weighting factors enable buyers to rate relative importance of attributes to be bid on. Another genre of RFQ software that has emerged is from vendors such as Hologix, Emptoris, Perfect and Menerva Technologies. For example, Hologix's RFQ tool, Attrium, allows complex configuration and bill-of-material relationships. Suppliers can specify offerings with ranges of attributes and price variations for terms and conditions. Perfect's product, Perfect Market, enables a buyer to post an RFQ and the software automatically searches seller's rules (predefined and stored in the database), calculates the best possible offer for each seller, and ranks all the offers according to the criteria that are most important to the buyer. Here again the buyer can specify weighting factors for each attribute using a sliding scale where the user can choose from various options such as "do not care", "important", "more important", "very important", and "most important".

In most of these existing approaches of additive utility models, two key assumptions are made, (i) the attributes are linearly independent, and (ii) the utility functions of individual attributes are known. However, in real life more complex situations can arise where the utility functions not only take multilinear form of dependency but even more complex interactions may exist. For example [4] provide an example (see pg. 232) of a farmer with preferences for various amounts of sunshine and rain because that will impact his crop. The amount of sunshine that the farmer prefers is interlinked with the amount of rain that had happened in the past. In the case of nonlinear utility function having inter-dependent attributes, if the individual attribute utility function is not known then it is difficult to determine that function or even if that is determined, it may not reflect the true utility function of the individual one.

In this paper, we propose an evaluation algorithm for ranking multiple attribute items considering nonlinear interactions between the individual attributes and assuming no a priori knowledge about the individual attribute utility functions. The proposed method also assumes no uncertainty during decision making [4][chapter

5]. A subset of items (bids) is randomly selected from the entire set of items (bids), and the user ranks them and submits the ranked list to the system. Every pair of the ranked list is then fed to a multi-layered feed-forward neural network and the network learns the relative preferences of the item set. Neural network models have been employed widely in the context of function approximation and category learning [12]. However, in the present context, since the desired utility function value is unknown, it is not possible to employ the function approximation algorithms in evaluating and ranking the item set. Therefore, we propose a novel objective function (error measure) and a new technique to adaptively adjust the learning rate has also been proposed in this framework.

Note that, in performing the ranking of multi-attribute items, we do not need to consider the evaluation of individual attributes separately (i.e., which attribute is more important than the other). This is because in the case of nonlinear interactions in the individual attribute utility functions, attribute importance or the evaluations are dependent on the present attribute value and there is no absolute importance of an attribute as used in the case of conjoint analysis or additive utility models. However, attribute selection/evaluation from a ranked list of items can be obtained by the sensitivity analysis of network output with respect to the input attributes. For example, the absolute value of the first derivative of network output with respect to an input attribute will provide an attribute sensitivity index of that attribute with respect to its present value. Various other methods have been proposed in literature for evaluation/selection of features/attributes of the items in the context of classification and clustering [13, 14, 12] which include selective elimination/addition of features and their effect on classification/clustering performance, formulating certain criterion measures and its optimization. However, from a ranked list of items, no such study is available in the literature of neural network study.

2 Problem Definition and Formulation

2.1 Background

In the multi-attribute utility theory, a parametric utility function is assumed for evaluating multiple attribute items (bids) [2, 4]. Various forms of valid utility

functions are available in the literature. The most general form of the utility function, called multilinear utility model is given as

$$u(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{w_i u_i}(\mathbf{x_i}) + \sum_{\mathbf{i=1}}^{\mathbf{m}} \sum_{\mathbf{j>i}} \mathbf{w_{ij} u_i}(\mathbf{x_i}) \mathbf{u_j}(\mathbf{x_j}) + \sum_{\mathbf{i=1}}^{\mathbf{m}} \sum_{\mathbf{j>i}} \sum_{\mathbf{k>j>i}} \mathbf{w_{ijk} u_i}(\mathbf{x_i}) \mathbf{u_j}(\mathbf{x_j}) \mathbf{u_k}(\mathbf{x_k}) + \ldots \tag{1}$$

where $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_m})$ is a vector of m random variables over performance measures, $u_i(.)$ is a single attribute utility function over measure i (scaled in 0 and 1), w_i is the weight for measure i, and the higher order terms represent the interactions between the different attributes. A simplified version of the multilinear utility function model is the multiplicative utility model where the higher order interaction terms are in general expressed as

$$u(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{w_i u_i}(\mathbf{x_i}) + \sum_{\mathbf{i=1}}^{\mathbf{m}} \sum_{\mathbf{j>i}} \mathbf{w w_i w_j u_i}(\mathbf{x_i}) \mathbf{u_j}(\mathbf{x_j}) + \sum_{\mathbf{i=1}}^{\mathbf{m}} \sum_{\mathbf{j>i}} \sum_{\mathbf{k>j>i}} \mathbf{w w_i w_j w_k u_i}(\mathbf{x_i}) \mathbf{u_j}(\mathbf{x_j}) \mathbf{u_k}(\mathbf{x_k}) + \ldots \tag{2}$$

such that the higher order interactions can be modeled as products of the individual attribute weights. A further simplification of the utility function leads to additive model where the higher order interactions are ignored such that

$$u(\mathbf{x}) = \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{w_i u_i}(\mathbf{x_i}) \tag{3}$$

In the formulation of the problem with additive utility function, a linear program solver is employed for an ordered set of bids $B_1 \succ B_2 \succ \cdots \succ B_n$ with the constraints

$$\begin{array}{lcl} \sum_{i=1}^{m} w_i & = & 1 \\ u(\mathbf{x_1}) & \geq & u(\mathbf{x_2}) \\ u(\mathbf{x_2}) & \geq & u(\mathbf{x_3}) \\ \vdots & & \\ u(\mathbf{x_{n-1}}) & \geq & u(\mathbf{x_n}) \end{array} \tag{4}$$

where $u(\mathbf{x_i})$ is the utility for bid . The constraints in equation (4) define a polytope which represents a feasible (valid) region of solution for a ranked set of bids represented by the vectors $(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n})$ such that for each i, $\mathbf{x_i} = (\mathbf{x_{i1}}, \mathbf{x_{i2}}, \cdots, \mathbf{x_{in}})$ as in equation (1). A point in the feasible region is chosen to get the solution to the problem [13]. In [6], the polytope is iteratively refined for each inequality and the successive query is generated in such a way that it reduces the feasible region to a maximum extent pulling the solution towards the center of the polytope.

2.2 Algorithm Conn-Eval

In the proposed algorithm, it is assumed that the utility function not only takes multilinear form with higher order interactions but can also have even more general forms (Section 4). Secondly, the individual attribute utility functions are considered to be unknown and therefore they are to be evaluated separately. Due to the presence of higher order nonlinear interactions, it is not possible to evaluate individual utilities ignoring or keeping the other attributes fixed. We provide a neural network based algorithm for evaluating the multiple attributes and their nonlinear interactions to generate the ranking of the multi-attributed items.

Method Conn-Eval (Input *In* : Set of items for evaluation with multiple attributes; Output *Rl* : Ranked list of items)

1. Select a subset of items and display them to user
2. User ranks the subset of items and submits the ranked sublist r to the system
3. Initialize a multi-layered feed-forward neural network

4. **repeat**
5. **for** every pair of items (a, b) in the ranked sublist r
6. Compute the required change in parameter weights of the neural network **end for** 7. Compute average change in parameter weights for all possible pairs in the ranked sublist r provided by the user
8. **until** there is not any significant change in the parameter weights

9. Evaluate the entire set of items *In* by the network and get the output ranked list *Rl*.

End Conn-Eval.

3 Neural Network based Approach (Conn-Eval)

A multilayered feedforward network [12] is trained on an ordered set of bids in such a way that that the network output provides same ordering of the bids as the desired one. In order to obtain the desired performance, it is crucial to do the training of the network properly with a suitably chosen error measure.

3.1 Error Measure

If $f(\mathbf{x})$ is the function generated by the neural network then $(f(\mathbf{x_i}) - f(\mathbf{x_j}))$ should have the same sign as $(u(\mathbf{x_i}) - u(\mathbf{x_j}))$ for all i and j. In other words, the relative ordering (partial or complete ordering) provided by the trained network must match with that given by the known samples. Any trained network can obtain this with

$$f(\mathbf{x}) = \phi(u(\mathbf{x})) \tag{5}$$

where $\phi(.)$ is a monotonic non-decreasing function (in stricter sense it should be a monotonic increasing function). If the training of the neural network can determine one such function then it serves the purpose. The network is said to have committed an error if for a multidimensional utility function, $(u(\mathbf{x_p}) \leq u(\mathbf{x_q}))$ and $f(\mathbf{x_p}) \geq f(\mathbf{x_q})$ for some p and q. The problem cannot be modeled as a function approximation task (as performed in nonlinear regression or function learning [14]) since the true value of the utility function $u(\mathbf{x_p})$ is unknown and therefore cannot be used to train the network function $f(\mathbf{x})$. Only the relative pairwise ordering of the true utility function i.e., $sign(u(\mathbf{x_p}) - u(\mathbf{x_q}))$ is known for the training bids. Therefore a measure to be optimized (minimized) can be considered as

$$E = \sum_{u(\mathbf{x_p}) \geq u(\mathbf{x_q})} [sign(u(\mathbf{x_p}) - u(\mathbf{x_q})) - sign(f(\mathbf{x_p}) - f(\mathbf{x_q}))](f(\mathbf{x_p}) - f(\mathbf{x_q}))^2 \tag{6}$$

where $f(.)$ is the network output. The first part of the error measure $[sign(u(\mathbf{x_p}) - u(\mathbf{x_q})) - sign(f(\mathbf{x_p}) - f(\mathbf{x_q}))]$ is zero if the bids are in the same order according to the network output as that of the original one, and as a result the error is zero. If the bids are out of order then the first part is nonzero and the network is penalized with an error equal to the squared difference between the network outputs of the bid pairs. The intuition is that even if there is an error in finding correct pairwise order, the network must at least make them equal. The generalization error for the measure (equation (6)) can be expressed as

$$E = \int_{\Omega} (f(\mathbf{x_p}) - f(\mathbf{x_q}))^2 . p(\mathbf{x_p}) . p(\mathbf{x_q}) dx_p dx_q \tag{7}$$

where Ω is the subspace where the network has made an error in ordering the bids, and $p(.)$ is the density function of observing the bids. Considering the independence of the random vectors $\mathbf{X_p}$ and $\mathbf{X_q}$, it can be shown that the generalization error reduces to

$$E = \mathcal{E}_{\Omega}((f(\mathbf{x}) - < f(\mathbf{x}) >_{\Omega})^2) \tag{8}$$

where $\mathcal{E}_{\Omega}$ indicates the expectation over the subspace Ω and $< . >_{\Omega}$ indicates the function mean over the subspace. Therefore the generalization error reveals the fact that minimization of the output error can lead to a function which is constant for all possible values of $\mathbf{x}$. Note that, a constant function also follows the property of non-decreasing monotonicity as mentioned in equation (5).

In order to avoid a constant function, the objective measure can be modified as

$$E = \sum_{u(\mathbf{x_p}) \geq u(\mathbf{x_q})} [sign(u(\mathbf{x_p}) - u(\mathbf{x_q})) - sign(f(\mathbf{x_p}) - f(\mathbf{x_q}))](K.f(\mathbf{x_p}) - f(\mathbf{x_q}))^2 \tag{9}$$

where K is a constant such that $0 < K < 1$. The generalization error can be expressed as

$$E = \int_{\Omega} (L(\mathbf{x_p}, \mathbf{x_q}) - 1)^2 . f^2(\mathbf{x_q}) . p(\mathbf{x_p}) . p(\mathbf{x_q}) . d(\mathbf{x_p}) d(\mathbf{x_q}) \tag{10}$$

where $L(\mathbf{x_p}, \mathbf{x_q}) = K.f(\mathbf{x_p})/f(\mathbf{x_q})$, such that E goes towards zero when $L(\mathbf{x_p}, \mathbf{x_q}) \rightarrow 1$ (necessary but not sufficient) under the assumption that $f(\mathbf{x}) \neq 0$ for all $\mathbf{x}$. Since $L(\mathbf{x_p}, \mathbf{x_q}) = K.f(\mathbf{x_p})/f(\mathbf{x_q})$, it imposes an ordering on $f(.)$ for $K < 1$ and $f(\mathbf{x}) \neq 0$ identically.

Instead of the squared error measure, we have taken the first order difference such that for two bids p and q,

$$E(p,q) = \begin{cases} f(\mathbf{x_q}) - K.f(\mathbf{x_p}) & \text{when } f(\mathbf{x_p}) \leq f(\mathbf{x_q}) \text{ but } B_p \succ B_q \\ 0 & \text{ordering is the same as desired.} \end{cases} \tag{11}$$

The total error measure for all the bid/item pairs is given as

$$E = \sum_{p,q} E(p,q) \tag{12}$$

3.2 Learning Rules

We train a three-layered feedforward network with the objective of minimizing the measure given by equation (11). The input layer accepts the variables or attributes of the bids/items, the first and second hidden layers generate the nonlinear functions and finally in the output layer, the activations are combined to generate the utility function. The steepest gradient descent rule can be derived as

$$\Delta w_{ij}^{l}(p,q) = -\eta . \frac{\partial E}{\partial w_{ij}^{l}} \tag{13}$$

which is simplified as

$$\Delta w_{ij}^{l}(p,q) = \begin{cases} \eta(K\delta_{il}(p)v_{jl}(p) - \delta_{il}(q)v_{jl}(q)) & \text{if } f(\mathbf{x_p}) \leq f(\mathbf{x_q}) \text{ and } B_p > B_q \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where

$\Delta w_{ij}^l(p,q)$	:	the change in w_{ij}^l when the network is presented with the pair of bids (B_p, B_q),
w_{ij}^l	:	weight of the link connecting the neuron i of layer $(l-1)$ to neuron j of layer l,
η	:	learning rate,
$v_{jl}(p), v_{jl}(q)$	:	the output of the jth node in layer l of the network for $\mathbf{x_p}$ and $\mathbf{x_q}$ as input to the network respectively,
$\delta_{jl}(p), \delta_{jl}(q)$	:	error propagated backward from the output layer to node j of layer l.

The output of node j of layer l is

$$v_{jl}(p) = \frac{1}{1 + \exp(-u_{jl}(p))} \tag{15}$$

where $u_{jl}(p)$ is the total input to the j^{th} node of layer l from the previous layer, given as

$$u_{jl}(p) = \sum w_{ij}^l v_{i,l-1}(p) \tag{16}$$

For an input node $v_{j0}(p) = x_{pj}$ where $\mathbf{x_p} = (x_{p1}, x_{p2}, \cdots, x_{pm})$ is an m-dimensional vector. The value of δ can be recursively computed as

$$\delta_{il} = \sum_k w_{ki} \delta_{k,l+1} v_{il}(1 - v_{il}) \tag{17}$$

For the output layer, $\delta = 1$.

The updating of the parameter weights is highly dependent on the selection of learning rate (η). If a large learning rate is chosen then the weights change rapidly, however, near the fixed point, it oscillates and the network does not converge to a particular solution. On the other hand, if a very small learning rate is chosen then network becomes very slow to converge.

3.3 Selection of the Learning Rate

Several studies are made for the optimal selection of the learning rate in the literature of neural networks and machine learning [12]. Certain annealing schedules [15, 12, 16, 17] have also been proposed in the literature in various other contexts. In order to circumvent such problems, conjugate gradient descent error minimization [16] has also been proposed instead of the steepest descent techniques. However, in the context of quotes (response to RFQ), this is being a first attempt (to the knowledge of the authors), no such algorithm has been proposed for automatic scheduling of the learning rate. The optimal learning rate at every iteration can be obtained by the principle of minimum residual error such that at every iteration the weights of the links are updated in such a way that the discrepancy between the ordering obtained from the network and the desired ordering on the training samples gets minimized.

For two different bids $\mathbf{x_p}$ and $\mathbf{x_q}$ (with $u(\mathbf{x_p}) \geq u(\mathbf{x_q})$ and $f(\mathbf{x_p}) < f(\mathbf{x_q})$), the weights need to be updated in such a way that

$$K.(f(\mathbf{x_p}) + \Delta f(\mathbf{x_p})) = (f(\mathbf{x_q}) + \Delta f(\mathbf{x_q})) \tag{18}$$

according to equation (11), where $\Delta f(.)$ is the change in the output of the network due to the change in the parameter weights. In other words,

$$K.\Delta f(\mathbf{x_p}) - \Delta f(\mathbf{x_q})) = f(\mathbf{x_q}) - Kf(\mathbf{x_p}) \tag{19}$$

Consider that

$$\begin{aligned} \Delta f(\mathbf{x_p}) \quad = \quad & \textstyle\sum_i \left(\frac{\partial f(\mathbf{x_p})}{\partial w_i^L}.\Delta w_i^L \right. \\ & \left. + \frac{\partial f(\mathbf{x_p})}{\partial v_{i,L-1}} \textstyle\sum_j \left(\frac{\partial v_{i,L-1}}{\partial w_{ij}^{L-1}}.\Delta w_{ij}^{L-1} + \frac{\partial f(\mathbf{x_p})}{\partial v_{j,L-2}} \textstyle\sum_k \left(\frac{\partial v_{j,L-2}}{\partial w_{jk}^{L-2}}.\Delta w_{jk}^{L-2} + \cdots + \right) \ldots \right) \right) \end{aligned} \tag{20}$$

From equations (14) and (17), the chain rule in equation (20) can be simplified and equation (19) can be expressed as

$$\eta.(\sum_{i,j,l}(\overline{\Delta w_{ij}^{l}})^2) = f(\mathbf{x_q}) - Kf(\mathbf{x_p}) \tag{21}$$

where

$$\overline{\Delta w_{ij}^{l}(p,q)} = K.\delta_{il}(p)v_{jl}(p) - \delta_{il}(q)v_{jl}(q) \tag{22}$$

Thus for steepest gradient descent, the optimal selection for the learning rate is given as

$$\eta_{opt} = \frac{f(\mathbf{x_q}) - K.f(\mathbf{x_p})}{\sum_l \sum_{i,j}(\overline{\Delta w_{ij}^{l}(p,q)})^2} \tag{23}$$

In the on-line updating of the weights of the network, the parameter values depend on the sequence in which the pair of bids/items are presented to the network. We have used the batch mode updating of the parameter values such that

$$\Delta w_{ij} = \frac{1}{N}\sum_{p,q}\Delta w_{ij}(p,q) \tag{24}$$

for all i and j, N being the total number of pairs. In displaying a list of items/bids to a user, often it is more important to display the top items as correctly as possible than the bottom items/bids. In other words, an error occurred in the positioning a top item/bid appears to be costlier than an error occurred down the list. In order to take care of this situation, the errors in the top bids/items should be counted more than that down the list. We, therefore, weigh differently to the mistakes depending on the position of the bid/item pair in the ordered list. Therefore, in the batch-mode updating of the weights, the total error is given as

$$E = \sum_{p,q}\lambda_{p,q}E(p,q) \tag{25}$$

The batch mode updating rule is thus given as

$$\Delta w_{ij} = \frac{\sum_{p,q}\lambda_{pq}\Delta_{ij}(p,q)}{\sum_{p,q}\lambda_{pq}} \tag{26}$$

where λ_{pq} is an weight associated with the pair (p,q) (assuming $p < q$, i.e., $B_p \succ B_q$ in the training set). The parameter λ_{pq} depends on the position of the bid/item p in the ranked list and its distance from the other bid/item q. For example, if the first bid/item is confused with the tenth bid/item then the error is more severe than

the case when it is confused with the second one. Secondly, the parameter should also depend on the absolute position of the bid/item. For example, if there is an error in positioning the first bid/item then the error is counted more than if the error occurs in the tenth bid/item position in the ranked list. Different heuristic measures can be defined for the parameter λ_{pq}. We have defined it as

$$\lambda_{pq} = \left[\frac{(n-p+1)(q-p+1)}{n^2}\right]^{\alpha} \tag{27}$$

where n is the total number of training bids/items in the ranked list. $\alpha > 0$ is a parameter which controls the relative weights between the top and bottom items. We have chosen it as 0.5.

4 Architecture Selection for Conn-Eval

The performance of the network is dependent on the selection of architecture for learning the bids. As discussed in literature [12], a large number of parameters in an architecture can result in poor generalization due to overfitting [16, 18], and a smaller architecture may not be able to capture the details. In the case of overfitting, if the function does not posses the monotonic nature then it will severely affect the generalization performance.

Neural network architecture selection is a widely studied problem in the literature [12, 18, 19], although no specific attempt is made to evaluate their performances in the context of ranking the responses to RFQ. Here we prescribe a guideline for selecting the number of hidden nodes in a neural network for a given kind of utility function.

In general, a multilinear utility function (equation (1))

$$u(\mathbf{x}) = \sum_i \alpha_i u_i(x_i) + \sum_{j>i} \alpha_{ij} u_i(x_i) u_j(x_j) + \cdots + \sum_{k>j>i} \alpha_{ijk} u_i(x_i) u_j(x_j) u_k(x_k) + \ldots \tag{28}$$

can be expressed as

$$u(\mathbf{x}) = \Pi_l(\sum_i \beta_{il} u_i(x_i)) \tag{29}$$

with $u_0(x_0) = 1$. Any such polynomial of order r can be expressed as a product form having r terms. Thus, a utility function can be expressed as

$$\log(u(\mathbf{x})) = \sum_l \log(\sum_i \beta_{il} u_i(x_i)) \tag{30}$$

since the ranking of the bid responses remains unaffected by logarithmic operation (logarithm is a monotonic increasing function). We can express an even more generalized form of utility function as

$$\log(u(\mathbf{x})) = \sum_l \gamma_l \log(\sum_i \beta_{il} g_{il}(\mathbf{x})) \tag{31}$$

where β and γ are constant weights and $g_{il}(\mathbf{x})$ are the subutilities capturing the dependencies among the variables and in general, are increasing or decreasing concave functions. The constants γ specify the power of degree of the subutilities in the interaction (for example quadratic, cubic and higher order terms). Given the individual utility functions, it is evident from equation (30) that only r hidden nodes suffice to find out a correct order of the responses if the multilinear utility function is of order r.

In the context of neural networks, a more generalized form of utility function can be expressed as

$$u(\mathbf{x}) = o(\sum_l \gamma_l h_l(\sum_i \beta_{il} g_{il}(\mathbf{x}))) \tag{32}$$

where $o(.)$ is a function (exponential as in equation (31) characterizing the complex interactions between the weighted subutilities together with the functions $h(.)$. The form of the utility function given by equation (32) is exactly the same form of function as produced by a multilayered feed-forward network with $g(.)$, $h(.)$ and $o(.)$ being the transfer functions of the first, second hidden layers and the output layer respectively. The first hidden layer should have an order of m nodes

Type of Utiility Function (m-dimensional input vector)	Hidden Nodes in the First Hidden Layer	Hidden Nodes in the Second Hidden Layer
Linear	$O(m)$	$O(1)$
Quadratic	$O(m)$	$O(2)$
Polynomial of $O(r)$	$O(m)$	$O(r)$
Log-polynomial of $O(r)$	$O(m)$	$O(r)$
Exp-polynomial of $O(r)$	$O(m)$	$O(r)$

Table 1: Estimates of the number of hidden nodes required in three-layered feed-forward network for different kinds of utility functions.

representing the number of variables in the input bids or objects with each hidden node modeling a concave function (equations (31) and (32)), and the second hidden layer should consist of r hidden nodes, r being the order of the polynomial. The increasing or decreasing nature of the subutilities can be controlled by the positive or negative values of the weights (such as β and γ in equation (32)) which can be embedded as weights of the links from first to second hidden layer of the network.

For different forms of utility functions such as logarithm of polynomials or exponential function of polynomials, the number of hidden nodes needs to be selected in the same way as that of the polynomial utility function, although the output transfer function can be chosen differently. Note that, the selection of transfer function of the output node (i.e., the node in the output layer) in this kind of application is not very crucial as that in the function approximation problem because it deals with the relative order of the input or the relative magnitude of the function and not with the true output value of the function. Table 1 summarizes the order of requisite number of hidden nodes in a three-layer feed-forward neural network for ranking the quotes.

In Table 1, it is assumed that the subutilities are not known. If the individual subutilities are known then only one hidden layer will suffice with the assumption that the subutility functions are fed directly as input and prescribed second hidden layer is used as the first hidden layer of the remodeled network.

5 Selection of Training Samples by Querying

In the algorithm based on neural network described above, the bids or objects are divided into two different sets, namely the training set and the test set. The network is trained on the training set and then tested on the test set. The performance of the algorithm depends on the training samples selected from the bids/items.

Algorithms [20, 21, 22, 23, 24, 25] are designed to selectively obtain the training samples (labeled samples) in order to perform the training (active learning) in various contexts including classification, clustering and function approximation. In [20], a survey is provided on different active learning algorithms in neural networks. Usually in the active learning algorithms, labeled samples are selected in such a way that it maximizes the information content required for the training of the learning machine. In supervised classification algorithms, query is generated (i.e., training samples are selected) from the most confusing decision regions. The most confusing samples are selected by employing a set of very simple learners and samples for which simple learners collectively make maximum number of incoherent decisions, are selected. In some of the algorithms as surveyed in [20], local methods are also employed (like Voronoi diagram) to select the suitable labeled samples. However, in the context of learning utility where only a ranked subset is provided as labeled samples (and not the labels of individual samples), very few query-based sampling techniques [6] to select labeled samples is reported in the literature. The main objective of the active learning is to reduce the cost of labeling the samples and make the learning algorithm faster, i.e., to train the learning machine with fewer examples.

In the present algorithm it is very difficult to employ simple learners to learn nonlinear utility functions for getting the most confusing bid or item pairs. In Iyengar et al. [6], an assumption is made about the linear dependency of known subutility functions. In the problem formulation (of the linear program) the order of items describe a polytope by virtue of the linear inequalities. At every step, a new bid or item is selected in order to minimize the volume of the polytope by selecting a bid which generates a hyperplane dividing the polytope into equal halves.

In the present neural network based training algorithm, no *a priori* knowledge about the individual utility function is considered, and nonlinear interactions are

allowed between the individual attributes. In order to follow the same guideline as in the Iyengar et al. [6], it is necessary to provide a mathematical description of the decision space in terms of complex nonlinear or piecewise linear boundaries generated by the two hidden layers which is hard to analytically describe. Here we provide a much simpler query generation algorithm which is designed particularly for evaluating the multi-attribute utility items.

In order to design the algorithm, a basic assumption is made about the form of individual attribute utilities. Each subutility follow an increasing or decreasing or a single hump-like curve in general even after the complex nonlinear interactions. However, it is assumed that any subutility curve does not possess frequently changing behavior of increasing and decreasing nature. It is assumed that the users do not commit mistakes in evaluating the bids or items (in order to generate the training data). In fact, even if a user commits errors, the present algorithm gets an approximate function based on the erroneous training data. Formally,

Assumption 1 : Let $f(\mathbf{x})$ be a utility function of item represented as attribute vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]'$, then for a set of ranked items (ranked according to utility function $f(\mathbf{x})$), $[\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \cdots, \mathbf{x}^{(N)}]$, for any attribute i, the derivatives $[\frac{\partial f}{\partial x_i}|_{\mathbf{x}=\mathbf{x}^{(1)}}, \frac{\partial f}{\partial x_i}|_{\mathbf{x}=\mathbf{x}^{(2)}}, \frac{\partial f}{\partial x_i}|_{\mathbf{x}=\mathbf{x}^{(3)}}, \cdots, \frac{\partial f}{\partial x_i}|_{\mathbf{x}=\mathbf{x}^{(N)}}]$ should have the same sign or can have at most one sign change if the user feedback is correct.

Based on the assumption, the query generation algorithm is given as :

S : Entire set of items or bids to be ranked.
T : Training set.

begin

1. Initialize the set T = A pair of bids randomly chosen from S.

repeat

2. Train the network with the training set T to obtain the utility function $f(\mathbf{x})$.

3. Sort all items in $S - T$. Without loss of generality, let the sorted set of items or bids be

$$\mathbf{x}^{(1)} \succ \mathbf{x}^{(2)} \succ \mathbf{x}^{(3)} \succ \cdots \succ \mathbf{x}^{(N)}$$

where $N = |S - T|$ is the set of items in the set $S - T$.

4. Compute

$$\mathbf{Z} = \begin{bmatrix} sign(\frac{\partial f}{\partial x_1}|_{\mathbf{x}=\mathbf{x}^{(1)}}) & sign(\frac{\partial f}{\partial x_2}|_{\mathbf{x}=\mathbf{x}^{(1)}}) & \cdots & sign(\frac{\partial f}{\partial x_n}|_{\mathbf{x}=\mathbf{x}^{(1)}}) \\ sign(\frac{\partial f}{\partial x_1}|_{\mathbf{x}=\mathbf{x}^{(2)}}) & sign(\frac{\partial f}{\partial x_2}|_{\mathbf{x}=\mathbf{x}^{(2)}}) & \cdots & sign(\frac{\partial f}{\partial x_n}|_{\mathbf{x}=\mathbf{x}^{(2)}}) \\ \vdots & & & \\ sign(\frac{\partial f}{\partial x_1}|_{\mathbf{x}=\mathbf{x}^{(N)}}) & sign(\frac{\partial f}{\partial x_2}|_{\mathbf{x}=\mathbf{x}^{(N)}}) & \cdots & sign(\frac{\partial f}{\partial x_n}|_{\mathbf{x}=\mathbf{x}^{(N)}}) \end{bmatrix} \tag{33}$$

where n is the number of attributes.

5. For each attribute i and consecutive bid/item pairs, compute sign changes such that

$$\begin{aligned} \mathbf{C_2} &= [change(Z_{11}, Z_{21}), change(Z_{12}, Z_{22}), \cdots, change(Z_{1n}, Z_{2n})] \\ \mathbf{C_3} &= [change(Z_{21}, Z_{31}), change(Z_{22}, Z_{32}), \cdots, change(Z_{2n}, Z_{3n})] \\ &\vdots \\ \mathbf{C_N} &= [change(Z_{N-1,1}, Z_{N1}), change(Z_{N-1,2}, Z_{N2}), \cdots, change(Z_{N-1,n}, Z_{Nn})] \end{aligned} \tag{34}$$

where $change(s_i, s_j)$ is defined as

$$\begin{aligned} change(s_i, s_j) &= 1 \quad \text{if } s_1 \text{ and } s_2 \text{ are opposite signs} \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{35}$$

6. Count the number of 1s in each vector C_i denoted by $|C_2|$, $|C_3|$, $\cdots$, $|C_N|$.

7. Select the bid/item j such that

$$j = argmax_i\{|C_i|\} \tag{36}$$

Add item/bid j to the training set such that

$$T \leftarrow T \bigcup \{\mathbf{x}^{(j)}\} \tag{37}$$

until the error (training error or validation error) is less than certain threshold or the size of training set reaches a maximum limit (governed by the user)

end

Note : In computing the matrix Z in step 4, it has been assumed that the signs of the derivatives are either (+)ve or (-)ve. However, the derivative can also take

a zero value in which case it can lead to a contradictory situation in step 5. For example, let for two different attributes the derivatives of the utility function for sorted set of items take the values as
(+ + 0 0 - - 0 0 + +) and (+ + 0 0 + + 0 0 + +).
In the first case, essentially there are two sign changes through 0 and in the second case there is no sign change of the utility function. However, in order to capture it in the sign change in step 5, the algorithm must have a look-ahead operator to understand which one is a sign change and which one is not. We did this operation by scanning the sign of the derivative of each column in Z sequentially, and converting each 0 with the sign of its predecessor. Therefore, the two different cases exemplified changes to
(+ + + + - - - - + +) and (+ + + + + + + + + +) respectively. Then step 5 is applied to compute the sign change along the columns of Z.

The query generation algorithm (active learning) has been designed based on the *assumption* 1 in order to minimize the confusion in each bid/item of the test set. Although in *assumption* 1, it is stated that the assumption is valid if the user feedback is correct, it is quite possible that a user can make mistakes in real-life scenario and the utility function can change its sign more than once. However, the query generation algorithm does not restrict the utility function to change its sign only once, rather it tries to minimize the number of sign changes. In other words, even if a user commits mistakes, the algorithm finds out the approximate utility function based on the user's feedback without restricting the utility function subject to only one sign change with respect to some attribute.

6 Experimental Results

We generated a batch of test bids where a subset of these test bids are ordered by a known utility function. A neural network is then trained by this subset of ordered bids. The performance of the trained network is then tested on the entire batch of bids ordered by the known utility function. The performance of the neural network based method is also compared with the LP-based technique with known subutility functions. Note that, in the neural network based method no a priori knowledge about the sub-utility functions is assumed. The performance of the network is tested with different types of utility functions having different number

of input variables.

6.1 Performance Measure

In order to quantify the performance, a measure is defined which compares the bid ranking according to the output of the network and that derived from the known utility function. For example, let the true order of 5 bids generated by the given utility function be

$$B_1 \succ B_2 \succ B_3 \succ B_4 \succ B_5$$

and two different predicted orders be

$$B_1 \succ B_3 \succ B_2 \succ B_4 \succ B_5$$

and

$$B_1 \succ B_4 \succ B_3 \succ B_2 \succ B_5.$$

In both the cases, only two bids are interchanged in the true order. However, the effect of the mistake is more severe in the second case because it makes B_4 greater than B_3, B_3 greater than B_2, and also B_4 greater than B_2. In literature, some measures have been used to evaluate the performance in ranking bids [5, 6]. Here we compare every ordered pair of bids with the order according to the network utility function, and the total number of mismatches is counted. The performance is then measured as

$$Performance = \left(1 - \frac{\text{number of mismatches}}{\text{total number of bid pairs}}\right) \times 100\% \tag{38}$$

If the algorithm is able to find out the correct order then the performance will show a 100% accuracy, otherwise it will degrade.

In order to have more importance to the mistakes in the ranking of top bids/items, the performance measure can be modified as

$$Performance1 = \frac{[(N-i+1)(j-i+1)]^{\alpha} \times 100\%}{\sum_{(i,j)\in ErrorSet}[(N-i+1)(j-i+1)]^{\alpha}} \tag{39}$$

where N is the total number of items in the ranked list (including the training set), i and j are the positions of the items in the ranked list with $i < j$ (i.e., $B_i \succ B_j$), and $ErrorSet$ is the set of all pair of items/bids for which the network commits mistakes. The parameter α is the same exponent as used in equation (27) ($\alpha = 0.5$).

Often, it is more important to show the top items correctly to the user instead of displaying the entire ranked set of items. In order to do so, another performance measure is defined which simply counts the number of correct top K items in the top K positions. This is the same measure as used in [6]. Therefore, the second performance measure is

$$Performance2 = \text{number of correct top } K \text{ items in top } K \text{ positions} \tag{40}$$

6.2 Test Utility Functions

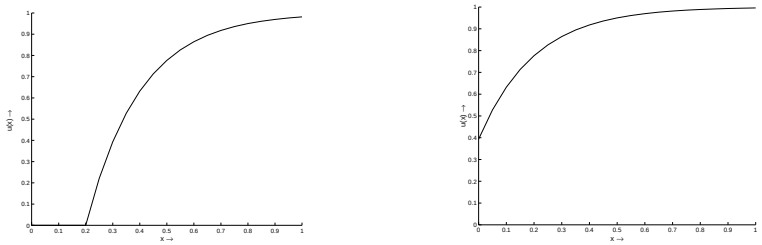
We considered three basic subutility functions and combined them by certain non-linear form (such as product or sum of products). In literature [4], different form of utility functions are mentioned. We have used have the following forms of subutility functions.

$$\begin{aligned} f_1(x) &= 1 - \exp(-\alpha(x-\theta)) && \text{for } x \geq \theta \\ &= 0 && \text{otherwise} \end{aligned} \tag{41}$$

where $\alpha > 0$ is a parameter which controls the steepness of the ascent of $f_1(x)$ (Figure 1) and θ can be positive or negative depending on the type of the variable. For example, in the case of *quality* of a product θ is positive indicating the fact that a product with bad quality has no utility. On the other hand, for *returns policy*, θ can be negative indicating that even if the company does not have any returns policy, the product can have some utility to the user.

The second subutility function (Figure 2) of decreasing nature is chosen as

$$\begin{aligned} f_2(x) &= 1 - \tfrac{1-\exp(\alpha x)}{1-\exp(\alpha\theta)} && \text{for } 0 < x \leq \theta \\ &= 0 && \text{otherwise} \end{aligned} \tag{42}$$



(a) (b)

Figure 1: Nature of utility function given by equation (41), (a) for $\theta > 0$ and (b) for $\theta < 0$.

Here the parameter $\alpha > 0$ controls the steepness of descent of the utility function and $\theta > 0$ controls the extent of the utility function. For example, for the attribute *price*, a product's utility decreases with price and it becomes zero for a user when the price exceeds certain threshold (more than the buying capacity of the user).

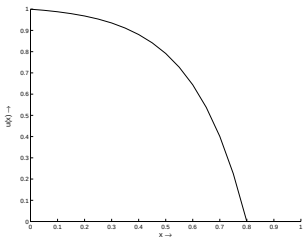


Figure 2: Nature of utility function given by equation (42)

The third form of subutility function (Figure 3) we have chosen is

$$f_3(x) = \exp(-\alpha_1(x-\theta_1)^2 sgn(\theta_1 - x) - \alpha_2(x-\theta_2)^2 sgn(x-\theta_2)) \qquad (43)$$

where

$$sgn(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The parameters α_1 and α_2 are the steepness control parameters for ascent and descent of the utility function on both the sides. The parameters θ_1 and θ_2 control the width of the function.

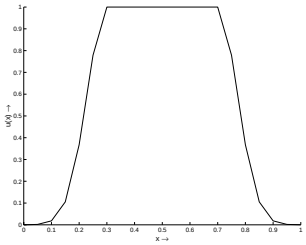


Figure 3: Nature of utility function given by equation (43)

6.3 Synthetic Data

We generated utility function in the context of procuring paper rolls [26] from various suppliers or manufacturers. The attributes that a customer can consider are *Price, Quality of the paper roll, Time to procure the material from the point of order placement, Restriction on the quantity* (e.g., a customer may be interested to procure 100 rolls, but a supplier will not supply less than 200 rolls or a supplier may not have 100 rolls and he can supply only up to 50 rolls at a time), *Shipping cost that the customer must pay, Restriction on width of the paper roll* (customer may like to procure such paper rolls having certain range of width), *Restriction on thickness, Returns Policy, Shipping Insurance, Customer Services Support, Vendor Rating, Manufacturing Capability, Financial Stability, Color of the Paper roll,* and many such other factors. Several of these attributes can be dependent on each other according to the user preferences. For example, if *price* becomes very low then the customer's utility on the *time-to-procure* will not change independently because a change in the lower range of *time-to-procure* will not add much to the utility value as compared to a very high range of *time-to-procure*.

The attributes *quality*, *manufacturing capability* and *shipping cost* take the first kind of utility function (i.e., utility function of ascending nature) given by equation (41) with a positive value of the threshold θ. The attributes *returns policy*, *shipping insurance*, *customer service support* and *vendor rating* also take the first kind of utility function (ascending nature) with a negative value of θ. The attributes *price* and *time to procure* take the second kind of utility function (descending nature as in equation (42)). The attributes like *quantity restriction*, *width* and *thickness* take the third kind of utility function (equation (43)) such that each of these attributes has a high utility if its value lies within certain range and the utility falls off suddenly if it is outside the range.

We defined two different composite utility functions. The first one is given as

$$\begin{aligned} u_1 \;=\; & 0.5u(\textit{price}).u(\textit{quality}).u(\textit{time-to-procure}) \\ & +0.3u(\textit{quantity-restriction}).u(\textit{shipping-cost}) \\ & +0.2u(\textit{width}).u(\textit{thickness}).u(\textit{returns-policy}).u(\textit{shipping-insurance}). \\ & u(\textit{customer-service-support}).u(\textit{vendor-rating}). \\ & u(\textit{manufacturing-capability}) \end{aligned} \tag{44}$$

and the second composite function is given as

$$\begin{aligned} u_1 \;=\; & u(\textit{price}).u(\textit{quality}).u(\textit{time-to-procure}) \\ & u(\textit{quantity-restriction}).u(\textit{shipping-cost}). \\ & u(\textit{width}).u(\textit{thickness}).u(\textit{returns-policy}).u(\textit{shipping-insurance}). \\ & u(\textit{customer-service-support}).u(\textit{vendor-rating}). \\ & u(\textit{manufacturing-capability}) \end{aligned} \tag{45}$$

i.e., u_2 is the product of all individual utilities.

The algorithm has been tested on randomly generated data in $[0, 1]$ for both the composite utility functions. A set of 100 bids/items are generated synthetically in both the cases and a subset has been used to train the network. The utility functions (described in equations (41), (42) and (43)) are imposed on the bid/item attribute vectors. The training data is chosen by random sampling as well as by query sampling algorithm as described in Sections 5 and 6. Figures 4 and 5 illustrate the performance and comparative results of the algorithms both for random sampling and query sampling. The performance is reported here with both the performance measures ($Performance1$ and $Performance2$ given by equations (39) and (40) respectively). Note that, the results shown in Figures 4 and 5 are the average of 10 different experiments. We have also tested the algorithm with different network architectures. Results for only one architecture is reported here as a case study.

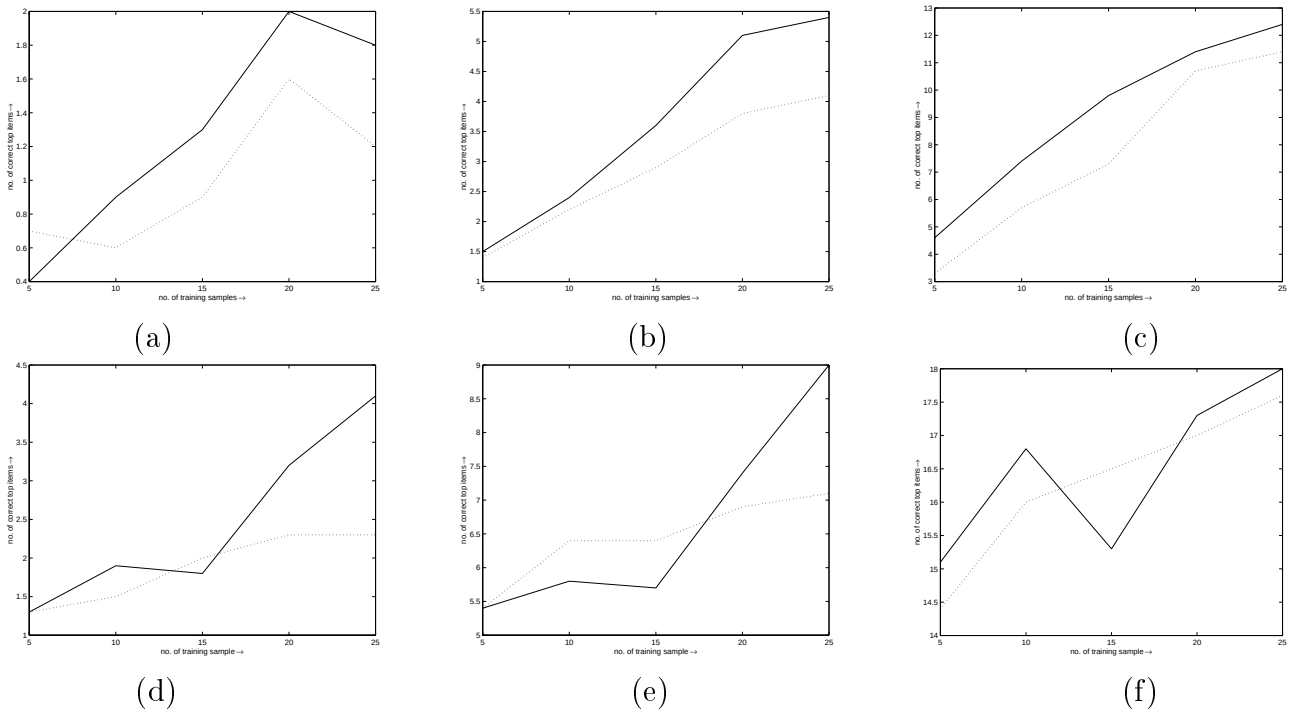
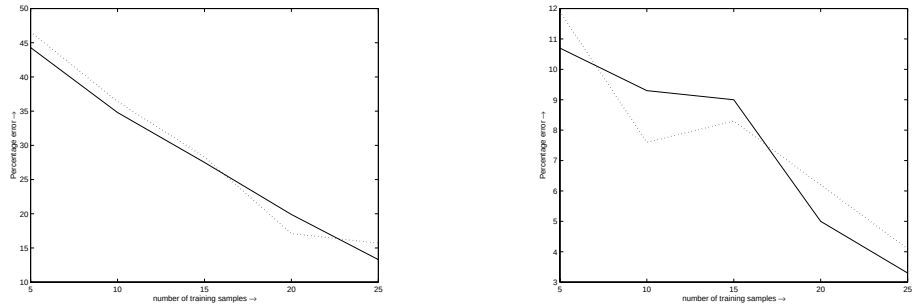


Figure 4: The performance ($Performance2$ as given by equation ()) of conn-eval for random sampling and query-based sampling techniques. The dotted curve show the performance for random sampling, and the solid curve illustrates the performance for query-based sampling technique. The results are average of 10 different experiments with an architecture (16, 10) where 16 is the number of nodes in the first hidden layer and 10 is the number of nodes in the second hidden layer. (a) The number of correct 5 top items in the top 5 positions with imposed utility function u_1 (equation (44)), (b) The number of correct 10 top items in the top 10 positions with imposed utility function u_1 (equation (44)), (c) The number of correct 20 top items in the top 20 positions with imposed utility function u_1 (equation (44)), (d) The number of correct 5 top items in the top 5 positions with imposed utility function u_2 (equation (45)), (e) The number of correct 10 top items in the top 10 positions with imposed utility function u_2 (equation (45)), (f) The number of correct 20 top items in the top 20 positions with imposed utility function u_2 (equation (45))



(a) (b)

Figure 5: The performance or error ($Performance1$ as given by equation (39)) of conn-eval for both random sampling and query-based sampling techniques. The dotted line illustrates the performance for random sampling and the solid line for query-based sampling with an architecture (16,10) for both the cases. (a) The performance for the imposed utility function u_1 (equation (44)). (b) The performance for the imposed utility function u_2 (equation (45)).

6.4 Real-life Data

In the synthetic data set, the bid/item attributes are randomly drawn from uniform distribution in $[0, 1]$. It is observed that the performance of the query-based sampling algorithm is almost comparable with the random sampling technique and does not provide a significant improvement. This is due to the fact that in the case of uniform distribution, the data is uniformly dense everywhere in $[0, 1]$ and as a result, the query generation algorithm does not provide an order of advantage over the random sampling technique. In order to test the efficacy of these algorithms, the method is tested on a real-life data set.

A real-life data set on digital camera is obtained from *amazon.com*. We obtained 197 cameras with 15 attributes. After data cleansing, we retained 93 cameras with 9 numeric attributes. The camera attributes we considered are *price*, *CCD resolution*, *memory card included*, *optical zoom*, *digital zoom*, *width*, *height*, *depth*, and *weight*. Since it is very difficult to obtain the ranking (or partial ranking) of the items in real-life, we imposed utility functions on these attributes separately. (Note that, it is possible to obtain a score for each item in the amazon.com, but the score reflects the overall feedback about the item. However, total number of feedbacks and the sales figures vary very widely across the items. It does not at all reflect any utility value of the items with respect to an individual or any particular

customer segment.)

We imposed the utility function of type 1 (equation (41)) on the attributes *CCD resolution*, *memory card included*, *optical zoom*, *digital zoom* and *depth*. The type 2 utility function (equation ()) is imposed on the attributes *price*, *width*, *height* and *weight*. The composite utility functions for the items are given as

$$\begin{aligned} u_1 &= 0.4u(price) + 0.3u(CCD\text{-}res).u(mem\text{-}card\text{-}incl).u(opt\text{-}zoom).u(digital\text{-}zoom) \\ &\quad +0.3u(width).u(height).u(depth).u(weight) \end{aligned} \tag{46}$$

and

$$\begin{aligned} u_1 &= u(price).u(CCD\text{-}res).u(mem\text{-}card\text{-}incl).u(opt\text{-}zoom).u(digital\text{-}zoom). \\ &\quad u(width).u(height).u(depth).u(weight) \end{aligned} \tag{47}$$

The effectiveness of conn-eval is tested for both the imposed utility functions with random sampling and query based sampling algorithms. Figures 6 and 7 illustrate the performance of the algorithms with two different architectures. Note that, the results reported here are the averages of 10 different experiments. We have tested the algorithms with various architectures like (10,6),(10,7),(10,8) and the ones illustrated in Figures 6 and 7. The results are almost comparable and it shows that the results for query-based sampling does not vary widely depending on the architecture so long as the architecture is selected according to the prescription as given in Section 4. The results for two different architectures are reported here as case studies.

The results as provided so far are obtained for a fully ranked (fully ordered) subset of training samples. However, the strength of the algorithm lies in the fact that it can learn from a partially ordered subset of training samples and obtain a total order on the test set (or the entire set of samples). We have tested the effectiveness of the query-based algorithm on a partially ordered training set on real-life data. The partial order of the training data is generated by comparing a sample with only a fewer percentage of samples in the data. Whenever a sample is selected by the query-based sampling technique, it compared with only 60% of the existing

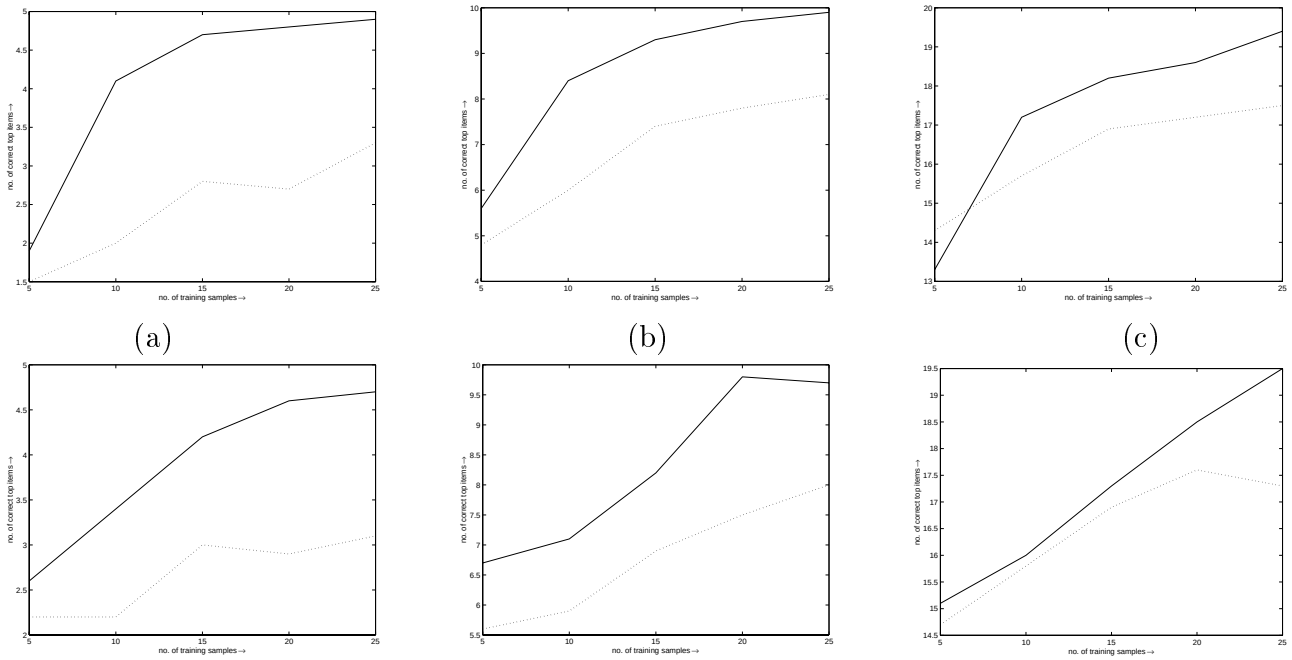


Figure 6: The performance ($Performance2$ as in equation (40)) of conn-eval on real-life data set with an imposed utility function u_1 (equation 46). (a) The number of top 5 correct items in the top 5 positions with an architecture (11,7), (b) the number of top 10 correct items in the top 10 positions with an architecture (11,7), (c) the number of top 20 correct items in the top 20 positions with an architecture (11,7),(d) The number of top 5 correct items in the top 5 positions with an architecture (12,8), (e) the number of top 10 correct items in the top 10 positions with an architecture (12,8), (f) the number of top 20 correct items in the top 20 positions with an architecture (12,8)

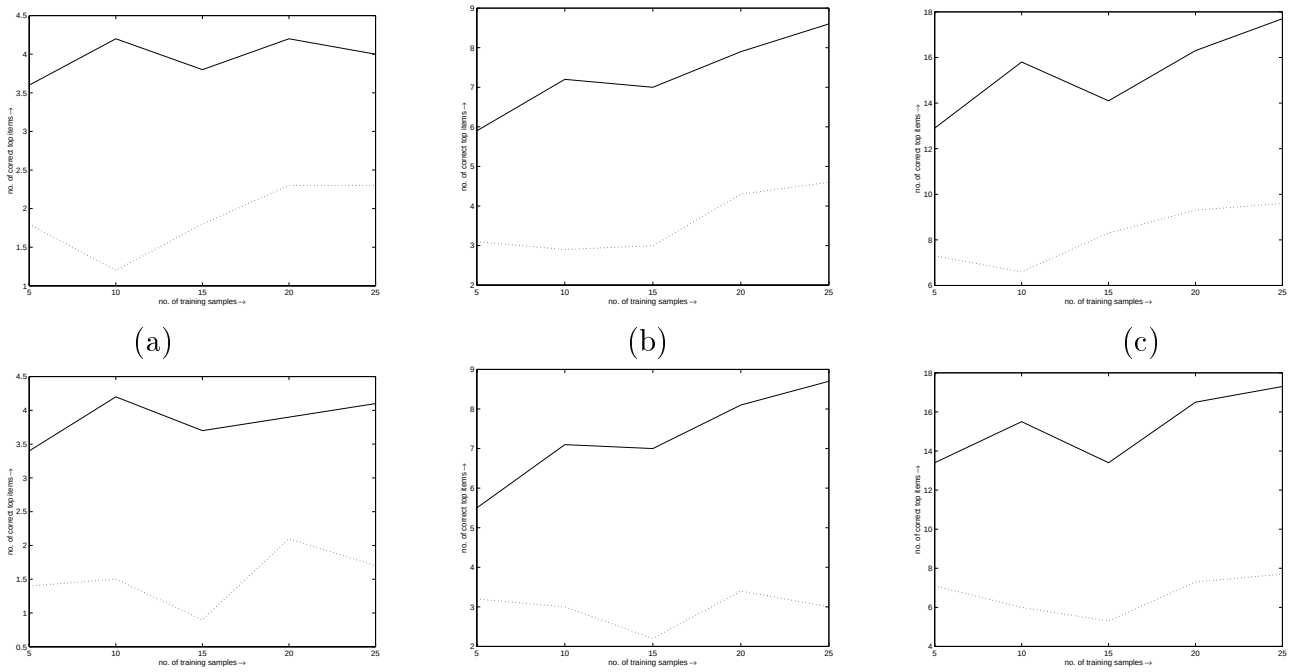


Figure 7: The performance ($Performance2$ as in equation (40)) of conn-eval on real-life data set with an imposed utility function u_2 (equation 47). (a) The number of top 5 correct items in the top 5 positions with an architecture (11,7), (b) the number of top 10 correct items in the top 10 positions with an architecture (11,7), (c) the number of top 20 correct items in the top 20 positions with an architecture (11,7),(d) The number of top 5 correct items in the top 5 positions with an architecture (12,8), (e) the number of top 10 correct items in the top 10 positions with an architecture (12,8), (f) the number of top 20 correct items in the top 20 positions with an architecture (12,8)

samples in the training set (by randomly choosing the samples in training set for comparison with a probability of 0.6). A typical case of partial order obtained with five training samples in the training set is $B_1 \succ B_2, B_3, B_4$ and $B_3, B_4 \succ B_5$. Figures 8 and 9 illustrate the results for query-based algorithm.

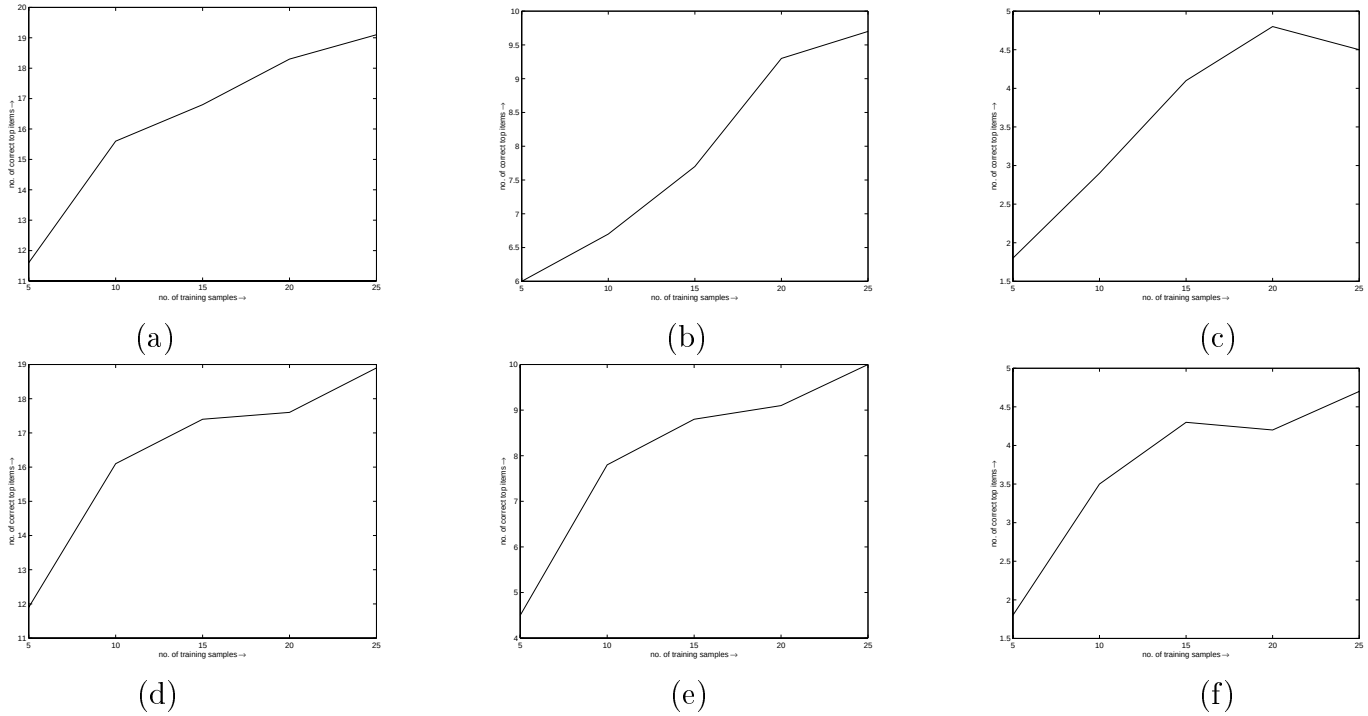


Figure 8: The results (average over 10 different experiments) on real-life data for partially ordered training data set with query-based sampling technique. The utility function imposed on the real-life data set is given by equation (47). The performance is measured according to equation (40) for different architectures; (a),(b), and (c) respectively top 20, top 10 and top 5 correct items for an architecture (12,7), (d), (e) and (f) respectively top 20, top 10 and top 5 correct items for an architecture (11,7).

7 Conclusions and Discussion

A neural network (connectionist model) based algorithm has been presented for ranking multiple attribute items with nonlinear interactions between the attributes assuming no *a priori* knowledge about the individual attributes. The nonlinear interactions between the multiple attributes exist in different domain of e-commerce

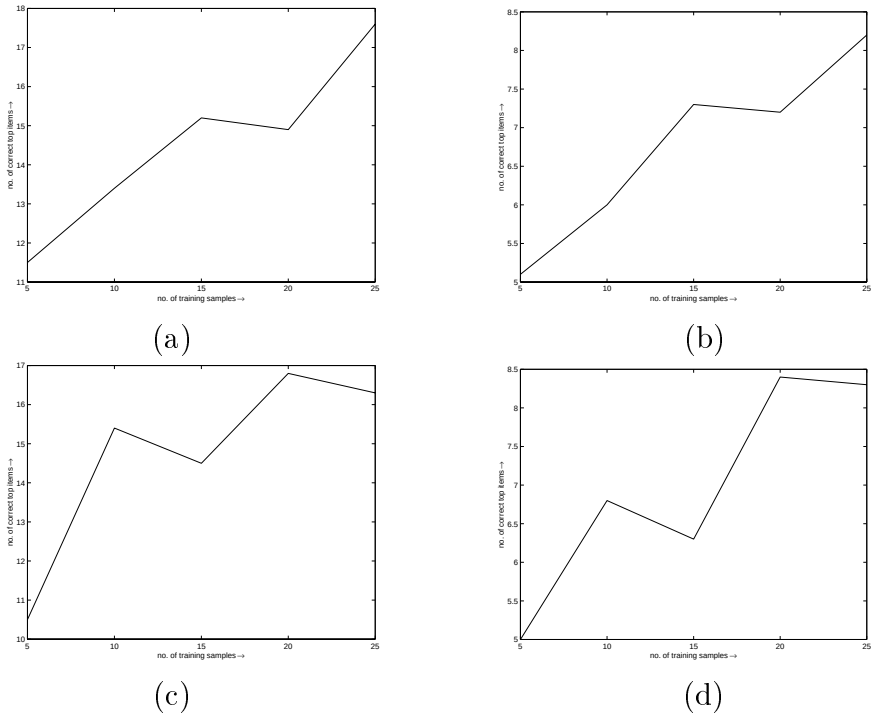


Figure 9: The results (average over 10 different experiments) on real-life data for partially ordered training data set with query-based sampling technique. The utility function imposed on the real-life data set is given by equation (46). The performance is measured according to equation (40) for different architectures; (a) and (b) respectively top 20 and top 10 correct items for an architecture (12,7), (c) and (d) respectively top 20 and top 10 correct items for an architecture (11,7).

such as auctions, negotiations, catalogues and customer modeling. Here we imposed certain complex nonlinear utility functions and the proposed model is found to be effective in ranking the items. The neural network based algorithm has further been modified to select training bids/items (query-based sampling). It has been found that the algorithm performs much better with query-based sampling of the training samples than with random sampling of the training samples in the case of real-life data set. We have also performed experiments where a partially ranked item set is provided as training set (instead of fully ordered training set). The proposed algorithm with query-based sampling is found to perform quite satisfactorily in ranking the item set with partially ordered training data. Since many of the existing techniques [6, 5, 7, 8] are based on the assumption of linear dependency between known utilities of individual attributes, they cannot be duly compared with the proposed algorithm which essentially assumes nonlinear interaction and unknown utility function of the individual attributes.

Attribute evaluation (i.e., which attribute is more important than the other) has not been performed separately in the present work. We can perform the sensitivity analysis of the output with respect to the input attributes after network finds out a ranked list of item. However, relative importance of individual attributes depends on all other attribute values of the bid/item concerned due to the nonlinear interaction of the attributes, and no absolute importance can be associated with individual attributes. Evaluating individual attributes is not within the scope of the present article and it constitutes a scope of our future study.

Conn-eval algorithm incorporates query-based sampling technique to perform faster learning, i.e., to learn the utility function with fewer number of training samples. However, a feed-forward neural network with two hidden layers has a large number of free parameters and with a few training samples, it becomes a highly underconstrained optimization problem in minimizing the error. In order to obtain better generalization performance, error minimization can be performed under certain regularization criteria [12] (for example, incorporating information criteria (AIC) or network cost (MDL) or other regularization measures including network dimension). In the context of ranking item set, the definition of generalization error can be different from that in the context of function approximation and concept learning. These issues can constitute an interesting scope of future study.

We provided a prescription for the suitable neural architectures for different number of attributes with different levels of complexity in the utility function. Studies

can be made for automatic selection of architecture in order to obtain a ranked set of items. In literature, methods are available for architecture selection by growing and/or pruning the networks [12]. Certain objective measures need to be defined in the context of ranking items in order to perform the task of pruning. These measures essentially relate to the generalization measures which (as described before) can constitute a part of further work.

In order to obtain a model capturing the nonlinear interactions between attributes, we have exploited the capacity of connectionist models. However, instead of considering only neural networks, other tools including decision trees, support vector machines, evolutionary algorithms or classical pattern recognition techniques [27] can be explored for this task.

References

[1] J. C. Butler, J. Jia, and J. Dyer, "Simulation techniques for the sensitivity analysis of multi-criteria decision models," *Eur. Journal Operations Research*, vol. 103, pp. 531–545, 1997.

[2] J. C. Butler, D. J. Morrice, and M, "A multiple attribute utility theory approach to ranking and selection," *Management Science*, vol. 47, pp. 800–816, 2001.

[3] R. T. Clemen, *Making Hard Decisions*. Boston MA.: PWS Kent Publishing, 1991.

[4] R. L. Kenney and H. Raiffa, *Decisions with Multiple Objectives*. New York: Wiley, 1976.

[5] M. Bichler, J. Lee, C. H. Kim, and H. S. Lee, "Design and implementation of an intelligent decision analysis system for e-sourcing," Tech. Rep. RC 22048(98946)30, IBM Research Report, April 2001.

[6] V. S. Iyengar, J. Lee, and M. Campbell, "Q-Eval : Evaluating multiple attribute items using queries," in *ACM Electronic Commerce (EC'01), Tampa, Florida, USA*, October 14-17, 2001.

[7] T. L. Satty, *The Analytic Hierarchy Process*. New York, USA: McGrawHill, 1980.

[8] G. Tewari and P. Maes, "Design and implementation of an agent-based intermediary infrastructure for electronic markets," in *ACM Conference on Electronic Commerce (EC'00)*, pp. 86–94, 2000.

[9] P. Green and V. Srinivasan, "Conjoint analysis in marketing research : new developments and directions," *Journal of Marketing*, vol. 54, pp. 3–19, 1990.

[10] R. Johnson, "Comment on adaptive conjoint analysis : some caveats and suggestions," *J. Marketing Research*, vol. 28, pp. 223–225, 1991.

[11] E. Andren, "Negotiation software helps work out e-commerce details," *Gartner Group Research Notes (ECEA)*, vol. 9, 2001.

[12] S. Haykin, *Neural Networks : A Comprehensive Foundation*. New York: Macmillan College Publishing Company, 1995.

[13] J. Basak, R. K. De, and S. K. Pal, "Unsupervised feature selection using neuro-fuzzy approach," *Pattern Recognition Letters*, vol. 19, pp. 997–1006, 1998.

[14] R. A. Devijver and J. Kittler, *Pattern Recognition : A Statistical Approach*. London: Prentice-Hall, 1982.

[15] J. Basak, "Learning Hough transform : A neural network model," *Neural Computation*, vol. 13, pp. 651–676, 2001.

[16] S. Lawrence and C. L. Giles, "Overfitting and neural networks : conjugate gradient and back propagation," in *Proc. IEEE Intl. Joint Conference Neural Networks, Como, Italy*, 2000.

[17] D. Saad, *On-line Learning in Neural Networks (Ed.)*. Cambridge CB2 2RU: Cambridge University Press, 2001.

[18] S. Lawrence, C. L. Giles, and A. C. Tsoi, "What size neural network gives optimal generalization?," Tech. Rep. UMIACS-TR-96-22,CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland, College Park, MD20742, 1996.

[19] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[20] M. Hasenjager and H. Ritter, "Active learning in neural networks," in *http://citeseer.nj.nec.com/404108.html*.

[21] M. Saar-Tsechansky and F. Provost, "Active learning for class probability estimation and ranking," in *Proc. Seventeenth International Conference on Artificial Intelligence (IJCAI-01), Seattle, Washington*, 2001.

[22] M. Hasenjager, H. Ritter, and K. Obermayer, "Active learning in self-organizing maps," in *Kohonen Maps* (E. Oja and S. Kaski, eds.), pp. 57–70, Amsterdam: Elsevier, 1999.

[23] Y. Freund, H. S. Seung, E. Shamir, and N. Tisby, "Selective sampling using the Query by Committee algorithm," *Machine Learning*, vol. 28, pp. 133–168, 1997.

[24] K. K. Sung and P. Niyogi, "Active learning for function approximation," in *Advances in Neural Processing Systems* (G. Teasauro, D. Touretzky, and T. K. Leen, eds.), vol. 7, pp. 593–600, Cambridge, MA: MIT Press, 1995.

[25] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of Artificial Intelligence Research*, vol. 4, pp. 129–145, 1996.

[26] R. Goodwin, P. Keskinocak, S. Murthy, F. Wu, and R. Akkiraju, "Intelligent decision support for the e-supply chain," *Proc. American Association for Artificial Intelligence*, 1999.

[27] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Ed.)*. New York: Wiley, 2000.