Selectivity Estimation in Extensible Databases - A Neural Network Approach[1]

Seetha Lakshmi
Informix Software Inc.
Menlo Park, CA 94025
seetha@informix.com

Shaoyu Zhou [2]
Informix Software Inc.
Menlo Park, CA 94025
shaoyu@informix.com

1. Introduction

Extensible database systems allow users to create new data types representing spatial, text, image, and other multimedia objects, and functions which operate on the new types[S96]. The new data types and functions can be used as predicates in SQL queries. Consider an insurance industry application, built using an extensible DBMS that has user defined extensions for spatial, text, date/time, and other business objects. The extensions support new data types such as ***point, circle,*** etc., and functions such as ***contains, interval, text_contains,*** etc. Let the tables Policies (policy_id, name, address, location, vehicle_type, ...) and Claims (policy_id, claim_tag, accident_date, accident_location, accident_report, ...) represent the partial schema containing both SQL'92 and user defined data types (UDTs). Consider a scenario in a targeted marketing application that requires a mailing list of all customers within 5 miles of point L, who have insured a 'sports utility vehicle' and were involved in a 'rear-ended' accident in the past 3 years. The corresponding SQL query would be:

SELECT P.name, P.address
FROM Policies P, Claims C
WHERE ***contains***(P.location, ***circle***(L, 5))
AND P.vehicle_type="Sports Utility"
AND P.policy_id = C.policy_id
AND ***text_contains***(C.accident_report, "rear-ended")
AND ***interval***(C.accident_date,***current_date***)<3 years

[1] Patent pending.

[2] Current affiliation: Microsoft Corp., Redmond, WA. E-mail: shaozhou@microsoft.com

Proceedings of the 24th VLDB Conference
New York, USA, 1998

This query has multiple predicates involving user defined functions (UDFs), namely, ***contains, interval, circle, current_date,*** and ***text_contains.*** The order in which the different predicates are evaluated within the query plan will significantly affect query performance [HS93], particularly when the tables contain a large number of rows. Predicate selectivity, which denotes the fraction of the table that satisfies a given predicate, plays a crucial role in determining the optimal execution order. The optimizer needs accurate selectivity estimates for these predicates in order to come up with an efficient query plan.

The histogram and other statistical techniques, widely used in commercial DBMSs, are well suited for predicates involving SLQ-92 type one dimensional, numeric data and operators such as <, =, and > [CR94, HS92, HNSS95, IP95, PIHS96, SLRD93]. Extending them to arbitrary UDTs and UDFs is, however, non-trivial. A histogram of spatial data types (say circle or box), or images, is rather non-intuitive. Selectivity estimation for predicates involving multi-dimensional and non-numeric data types is still in its infancy [TS96, KVI96]. We believe that using a learning system, which makes crude initial estimates that are progressively improved based on data gathered from operational systems, is a practical approach for solving this problem. In this paper, we present a neural network based approach for estimating the selectivity of predicates involving UDFs. Experimental results showing the accuracy and validity of the proposed scheme are also provided. The proposed scheme has been incorporated in a database administrator's tool for Informix Universal Server, an object relational DBMS. We briefly discuss the design of this tool and its integration with the DBMS.

2. Neural Networks in Selectivity Estimation

Readers are referred to [AIFAQ, HNC96], and the references there, for an introduction to the field of neural networks and an in-depth understanding of the various architectures, learning algorithms, mathematical formalism, similarities and differences between neural networks and classical statistical techniques, etc.

Our neural network approach for selectivity estimation involves two major tasks, 1) construction and training of neural networks and 2) use of trained networks to predict the selectivity. Task 1 can be performed off-line as a DBMS maintenance routine, similar to invoking the UPDATE-STAT or RUNSTAT utilities provided by the DBMS for gathering histogram data. The outcome of task 1 is a set of trained networks that are saved as mathematical equations within a system table in the database. During the query optimization phase, the optimizer retrieves the equation corresponding to a predicate, and evaluates it after substituting the parameters in the equation with the actual values from the query being optimized. Based on our design approach for collecting training data set and our experience with spatial data, we find that the overhead of task 1 is comparable to that of gathering histogram data and the overhead for task 2 is imperceptible.

In the rest of this section we describe the process of constructing and training a neural network for determining the selectivity of an example predicate viz., ***contains*** (P.location, ***circle***(L, 5)).

2.1 Feature Vector as Neural Network Input

The back-propagation class of algorithms used to train neural networks requires the network inputs to be numeric values. Hence, the first step in constructing the neural network is to identify and map the nonnumeric arguments in the predicate. The CIRCLE object ***circle***(L, 5) in the example predicate is a nonnumeric argument. We use the term feature vector to refer to the set of numeric values that represent an object. Feature vector for complex data type objects can be extracted from the meta-data maintained by the database. For example, the CIRCLE object in the spatial extension is a complex data type object made of two other objects, namely, a POINT object representing its center and a floating point object representing its radius. A POINT object, in turn a complex object, is made of two floating point objects representing its x and y coordinates. The meta-data maintained by the database regarding the spatial data types CIRCLE and POINT include the following UDFs:

POINT	***center***(circle-object)
FLOAT	***radius***(circle-object)
FLOAT	*x*(point-object)
FLOAT	*y*(point-object)

By recursively applying the appropriate functions, until a numeric value is returned, the feature vector for the circle object can be obtained as:

{*x(**center***(circle-object)), *y(**center***(circle-object)), ***radius***(circle-object)}

We discuss feature vector extraction of other types of objects in [LZ97].

2.2 Neural Network Configuration

Once the feature vector and input to the neural network are identified, the next step is to construct a neural network. Neural network experts recommend constructing, training, and validating several configurations, and choosing the best one as the final candidate. A possible back propagation based neural network configuration for the example predicate is shown in Figure 1.

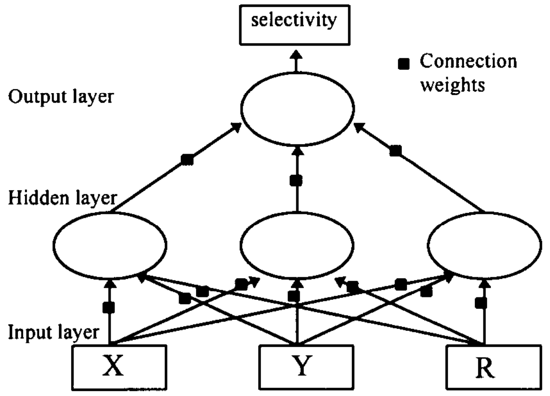

Figure 1. A Back Propagation Network for ***contains*** (…,circle_object)

2.3 Training Data Set

Training a neural network is the process of establishing values for the connection weights. This is carried out by presenting the network with a training data set and using a training algorithms such as the back propagation algorithm to adjust the connection weights. The training data set consists of tuples of input parameters and the corresponding actual selectivity. The data set should be representative of the operational environment in which the trained network will be deployed. It is obtained either by tracing and monitoring the queries in an operational DBMS or by executing synthetically generated queries. The input parameters for the synthetically generated queries can be obtained either by sampling the database or using a random value generator. In our implementation, if the input parameter corresponds to a binary large object such as an image or a character large object such as a text file, we sample a database table containing the appropriate objects; for other input parameter types (e.g. complex objects such as the circle object) we construct the object using the constructor function available as meta-data for that data type. For instance, using the constructor function ***circle***(center, radius) for the circle data type, a random circle object is generated by drawing three random values to represent the x, y coordinates of the circle's center and the radius. Finally, the synthetic query, to be executed against the real database for obtaining the training data set, is formulated as follows:

SELECT COUNT(*) FROM Policies P
WHERE ***contains***(P.location, ***circle***(***point***(***random***(X), ***random***(Y)), ***random***(R))

Note that, in an efficient implementation, the training data set can be gather through a single scan of the table, as opposed to executing mutiple SQL query.

The training data set, then, comprises of tuples of the form (X, Y, R, N/number of rows in table) where X, Y, R are random values, and N is the result of executing the above query. Our empirical studies suggest that even a small training data set (100-200 samples from tables with 32K rows) leads to fairly accurate predictions by the trained network. A portion of the training data set is usually reserved for validation purposes. Once the training data set is available, the network can be trained repeatedly with this data until the desired accuracy is observed on the validation data set. The final configuration of the trained network is represented by the number of input nodes, feature vector, number of hidden nodes, and a vector of connection weights.

3. Empirical Results

We evaluated the predictive capabilities of the proposed technique with a variety of predicates and found the results to be very compelling. Here we present some results obtained with spatial extensions and built-in functions. Our test environment corresponds to the SEQUOIA 2000 benchmark [SFGM93]. It consisted of 3 tables containing 62K, 200K, and 16K of 2-D spatial data. Figure 2 shows the absolute difference between the actual selectivity and the predicted selectivity for 150 queries with predicates involving the following three spatial UDFs:

1. ***contains***(APointTypeColumn, ***circle***(***point***(X,Y),R))
2. ***intersectIn***(APathTypeColumn, ***box***(***point***(X1,Y1), ***point***(X2,Y2)))
3. ***overlap***(APolygonTypeColumn, ***box***(***point***(X1,Y1), ***point***(X2,Y2)))

The actual selectivities ranged from 0 to 1. From Figure 2 we can observe that in each case, for 80% of the test cases

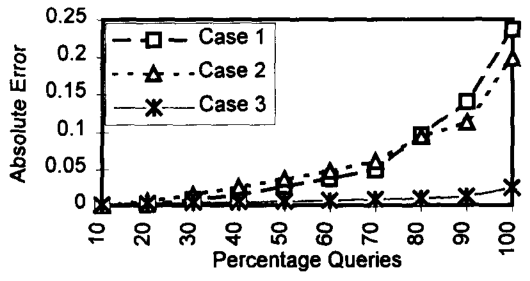

Figure 2. Error Measures

the predicted selectivity is within 10% of the actual selectivity and that the maximum error is within 25%.

We also tested the neural network approach with integer and character data types and operators such as = and >. The data distribution for the columns involved in the predicate had different degrees of skew. The predictions from our neural network models were always within 10% of the actual value. Additional information on our experimental studies regarding sensitivity to the network configuration, training set size, learning algorithms, etc. is available in [LZ97]. Overall, we found the accuracy and the overhead to be within satisfactory levels.

4. Integration with an ORDBMS

The motivation behind our investigation into a neural network approach is to provide a practical solution for a problem faced by the current wave of ORDBMSs. For the initial introduction of this technology, we have developed an interactive tool called BladeWatcher for Informix Universal Server. DBAs will use this tool to construct and train the neural networks for the predicates in poorly performing queries. Note that this tool is for facilitating task 1 described in Section 2. Task 2 is integral to the optimizer and is performed within the DBMS.

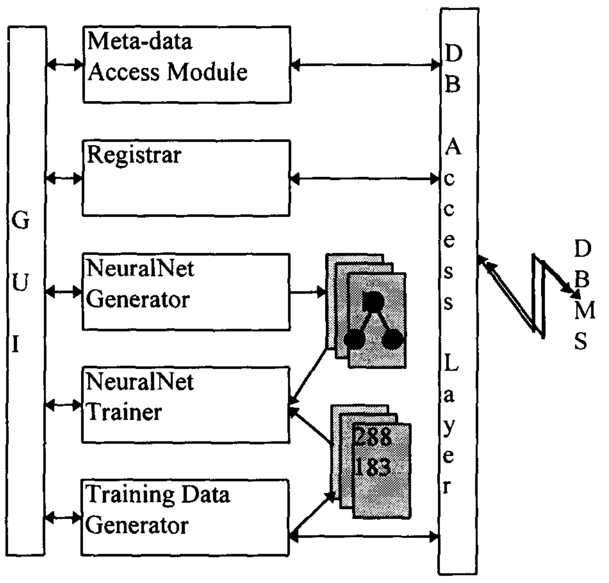

Figure 3. BladeWatcher Architechture

The high level architecture of BladeWatcher is shown in Figure 3. The meta-data access module helps DBAs select specific tables, columns, and UDFs involved in a predicate for which accurate selectivity estimates are needed. It also automatically chooses the feature vector extraction functions and constructors for random query generation. DBAs can approve or override these choices. Bladewatcher can either automatically generate the training data set or accept an external file containing the training data set. The neuralnet generator and trainer modules construct a back propagation or cascade correlation network and train it to the desired degree of accuracy. The trainer module repeatedly uses part of the training data to progressively adjust the connection weights and node biases using the chosen training algorithm. The remaining data in the training set is used as validation data to assess the accuracy of the network and to produce error measures. Based on the error value the DBA can either register the network with the DBMS or perform additional training. The registration module stores the network representation within the database system tables for the optimizer to use. Registered networks can be retrained if the database has undergone a substantial number of updates or if a new training data set reflecting a different usage pattern is available. The registration module also allows the registered networks to be disabled or dropped from the system.

BladeWatcher is a client server tool that communicates with the DBMS through a DB access layer. It accesses the DBMS only for meta data information (which is subsequently cached locally) and for generating the training data set. Actual training of the neural network is done on the client machine.

5. Summary

In this paper we have presented a neural network based approach for estimating predicate selectivities in extensible ORDBMS. While neural network models can be viewed as a class of statistical non-linear regression models, the algorithms used to train the neural networks make them more appealing to practitioners than classical regression technique because the latter can become quite unwieldy when we have to deal with complex data representing spatial, multimedia and other arbitrary user defined data types. The proposed method has been incorporated in a database administrator's tool for Informix Universal Server. With this tool, administrators create and train neural network models to compute the selectivity of UDFs. The trained neural network is stored as a selectivity function which is invoked by the optimizer during the optimization phase. Our experimental studies with spatial extensions to the DBMS show that i) the proposed method provides fairly accurate selectivity estimates; ii) the run-time overhead for evaluating the selectivity function during the optimization phase is imperceptible; and iii) the overhead of training the neural network is comparable to collecting the necessary statistics for building a histogram for traditional SLQ92 data types. Thus, the proposed technique offers a practical solution for an important problem faced by today's ORDBMS and extensible DBMS vendors who allow numerous user defined extensions to their databases in the form of datablades, data cartridges, or extenders.

Acknowledgment

The authors would like to thank Mike Stonebraker , Paul Brown, Bob Hedges, Jeff Sidel, and Piyush Gupta for their support and feedback.

References

[AIFA97] "ai-faq/neural-nets," *archive at ftp://ftp.sas.com/pub/neural/FAQ.html*

[CR94] C.M Chen and N. Roussopoulos "Adaptive Selectivity Estimation Using Query Feedback", *Proc. of the ACM SIGMOD'94*; 161-172.

[HNC96] "Handbook of Neural Computation," *IOP Publishing Ltd.* 1996. Editors: E. Fiesler and R. Beale.

[HNSS95] P.J. Haas, J.F. Naughton, S. Seshadri, and L. Stokes, "Sampling-based estimation of number of distinct value of an attribute," *Proc. of VLDB '95*; 311-322.

[HS92] P.J. Haas and A. Swami "Sequential sampling procedures for query size estimation," *Proc. of the ACM SIGMOD'92*; 341-350.

[HS93] J.M. Hellerstein and M. Stonebraker, "Predicate Migration: Optimizing Predicates with Expensive Predicates," *Proc. of the ACM SIGMOD'93*; 267-276.

[IFMX96] Informix Software, Inc. "DataBlade Developers Kit. User's Guide," 1996.

[IP95] Y. Ioannidis and V. Poosala "Balancing Histogram Optimality and Practicality for Query Result Size Estimation", *Proc. of the ACM SIGMOD'95;* 233-244.

[KVI96] P. Krishnan, J.S. Vitter, and B. Iyer, "Estimating alphanumeric Selectivity in the Presence of Wildcards," *Proc. of the ACM SIGMOD'96*; 282-293.

[LZ97] S. Lakshmi and S. Zhou, "A Neural Network Based Approach for Estimating Selectivity in

Extensible Databases," An Informix CTO Paper (1997).

[PIHS96] V. Poosala, Y. Iaonnidis, P. Haas, and E. Shekita, "Improved Histograms for Selectivity Estimation of range Predicates," *Proc. of the ACM SIGMOD'96;* 294-305.

[SFGM93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, "The SEQUOIA 2000 Storage Benchmark," *Proc. of the ACM SIGMOD'93*; 2-11.

[SLRD93] W. Sun, Y. Ling, N. Rishe, and Y. Deng, "An Instant and Accurate Size Estimation Method for Join and Selection in a Retrieval-Intensive Environment," *Proc. of the ACM SIGMOD'93*; 79-88.

[S96] M. Stonebraker, "Object-Relational DBMSs: The Next Great Wave," Morgan Kaufmann Publishers, Inc. (1996)

[TS96] Y. Theodoridis and T. Sellis, "A Model for the Prediction of R-tree Performance," *Proc. of the ACM PODS'96*; 161-171.