Approximate Query Answering by Model Averaging

Dmitry Pavlov
NEC Research Institute
Princeton, NJ, 08540
dpavlov@research.nj.nec.com
Padhraic Smyth
Information and Computer Science
University of California, Irvine, Irvine, CA 92697-3425
smyth@ics.uci.edu

October 1, 2002

Abstract

In earlier work we have introduced and explored a variety of different probabilistic models for the problem of answering selectivity queries posed to large sparse binary data sets. These models can be directly scaled to hundreds or thousands of dimensions, in contrast to other approximate querying techniques (such as histograms or wavelets) that are inherently limited to relatively small numbers of dimensions. In this paper, we extend this work by applying probabilistic model-averaging to the problem of query answering, a scheme that allows the query-answering algorithm to automatically and optimally adapt to both the specific nature of the data and the distribution of queries being issued any specific user. We demonstrate that on real-world and simulated data sets that model-averaging can reduce the prediction error of any single model by factors of up to 50%. Learning the combining weights is a straightforward and scalable optimization problem that can be easily automated, providing a practical framework for approximate query answering with massive data sets.

Keywords: Binary transaction data, query approximation, probabilistic model, itemsets, ADTree, maximum entropy.

1 Introduction and Previous Work

Consider a data set with N rows and d columns (attributes or fields). Assume the attributes are binary or categorical for simplicity. A very common operation in interactive querying and in data mining is determining the number of rows in the data that satisfy a particular conjunctive query, e.g., how many rows satisfy $A = 1, F = 0, Z = 1$ where A, F, and Z are bi-

nary attributes for example. The straightforward way to answer such a query is by directly scanning the data (known as a linear scan) to count how many times the condition $A = 1, F = 0, Z = 1$ matches a row. For very large data sets the time taken to conduct a linear scan may be impractically large. For example, with current technology the rate at which data can be read from disk is about 2 Mbytes/second, and the rate at which data can be scanned in main memory is about 10 Mbytes/second. Thus, with a 1 gigabyte data set (for example) it will take about 10 minutes to load the data from disk and perform a linear scan.

In this context the ability to quickly generate *approximate answers* to queries on large data sets is of importance across a range of data analysis applications, ranging from query selectivity estimation in relational database management to interactive ad hoc exploration of massive data sets.

We focus in this paper on approximate query answering algorithms for sparse high-dimensional binary data sets. Examples of such data sets abound in data mining applications, e.g., where the columns are Web pages and the rows are different visitors to the site, or where the columns are products and the rows are retail customers. Generalizations to sparse categorical data (where each non-zero entry could a count or a rating) can also be developed [17] but are not discussed in this paper.

In terms of the content of the queries we focus on simple conjunctions in this paper, noting that in past work we have demonstrated that it is straightforward to generalize the methods to handle arbitrary Boolean queries [14] — in essence, once one can approximate counts of conjunctions accurately one can then approximate functions of counts, etc. We also assume that the

database consists of one table and our task is to estimate counts only, i.e. we do not need, for example, to perform JOINs or return the actual records satisfying the query.

Prior work in the database community on this problem has focused primarily on *selectivity* or *count* queries. These are queries that are posed to a historical training data set, so that the perfect answer to a query can always be obtained by a full scan of the data at hand (e.g., how many people visited Web pages A, B, and C last week). In this paper we also consider *generalization* or *predictive queries* where the goal is to go beyond the historical data at hand and to forecast the count for a particular query on future data (e.g., how many people will visit Web pages A, B, and C in the next week). In general we have found that the techniques that work well for count queries tend to also work well for generalization queries. From a statistical viewpoint this requires generalization beyond the available historical data whereas count queries (selectivity estimation) in effect can be viewed as a form of data compression.

For count queries, the count of the query, if divided by the total number of records in the data, is the empirical probability of the query. For predictive queries, the goal is to estimate the probability that a randomly selected row in a future data set will satisfy the query, Thus, in this paper we will treat the problem of estimating counts as a problem in estimating probabilities, and will focus on the use of different probabilistic models of the data as the basis for a number of different query answering algorithms. The general idea is that we can build a probabilistic model of the data (or gather some form of cached store of counts) in an offline (cost-free) manner, and then answer queries relatively quickly using a probabilistic model in real-time, without having to access the raw data.

While there is a large body of work on estimating probability distributions from a generalization viewpoint in the statistical and machine learning literatures, there has been relatively little work on learning distributions for the specific purpose of query answering. One exception is [8] which looked at Bayesian network learning for queries, but with somewhat different goals and different techniques to those of this paper.

Ideally we would like our probabilistic models to be (a) accurate, (b) relatively fast compared to a linear scan, (c) scalable in terms of both speed and accuracy to large numbers of records and high dimensional data, and (d) flexible in that the method can trade-off accuracy with time and memory if necessary.

The *independence model* is the simplest probabilistic model we can imagine for this purpose, and it has been widely used in commercial database systems for query selectivity estimation, primarily because of its simplicity and very low time and memory requirements [18]. However, the independence model is often too simple to adequately model the complexities of real-world data sets [5]. Furthermore, it is not flexible in the sense of allowing any tradeoffs between accuracy and time or memory, in the sense that there is no parameter in the model that we can vary to achieve more accurate answers at the cost of waiting longer.

A variety of more sophisticated techniques for approximate query answering, that in general do allow such flexibility, have been proposed in recent years, such as wavelet models [12, 4] and multidimensional histograms [18, 13]. A limitation of these approaches, however, is that they do not scale well to high dimensions due to the "curse of dimensionality" and, thus, tend to only work reliably on relatively low-dimensional problems (e.g., problems with 10 or fewer dimensions).

The relatively simple technique of answering the query based on a random sample [10, 1] does satisfy our flexibility requirement in that we can always increase the size of the random sample (at least offline) to gain accuracy. However, we have shown in earlier work that simple random sampling can be quite inferior in terms of time-accuracy and memory-accuracy tradeoffs when compared to more sophisticated probabilistic models [16, 17, 15].

A variety of flexible probabilistic models such as mixture models [19], Bayesian networks [7, 11], and related dependency models [6] have also been proposed with the goal of finding accurate, fast, and flexible models. However, all of the work reported in these papers is either not scalable to high dimensions, or was only evaluated on relatively low-dimensional problems (say, 10 or fewer dimensions). For example, learning a Bayesian network model offline on 1000-dimensional binary data is quite feasible with current learning algorithms and computational technology. However, if we have a query involving (say) 4 attributes, this requires the marginalization ("summing out") over all of the other 996 variables whose values are not fixed in the model to obtain the probability of interest. The structure of the graph can of course be leveraged to make this sum tractable, but we have found that on real-world binary data sets that the graph underlying the Bayesian network can have relatively large cliques making exact inference intractable [16, 17]. Approximate inference could be carried out using Gibbs sampling but is not particularly well-suited to generating approximate query answers in real-time given its relatively slow convergence (especially for queries involving small probabilities).

In our prior work on approximate query answer-

ing [16, 17, 15] we have proposed and investigated the accuracy, speed, scalability, and flexibility properties of a variety of probabilistic models, including mixture models, maximum entropy models, models based on inclusion-exclusion, and Bayesian networks. We have shown, for example, how to scale up to high dimensions, reporting results on query approximation with on the order of 500 attributes, which is 2 orders of magnitude greater than with previous techniques — here the "trick" is to learn a model in real-time on the variables in the query using cached counts (frequent itemsets) and sidestep completely the issue of building a full joint probability distribution on all d attributes.

We also demonstrated that there are several fundamental trade-offs between accuracy of the answer, and time and memory taken to generate the answer [17]. For example, models with very low time and memory complexity (such as the independence model) generally tend to be relatively inaccurate. A further complicating factor is that the performance of the model can be significantly affected by the nature of the underlying data set (e.g., the sparsity of the data for a binary data set) and the distribution of queries being issued (e.g., the average number of attributes contained in queries).

One clear conclusion from this earlier work that there is no single model or technique that is universally superior across all data sets and query distributions. Equivalently, it is impossible to predict ahead of time which particular technique will provide the best performance on a particular data set and with a particular query distribution.

This paper addresses this question specifically. If different models have different strengths and weaknesses depending on the nature of both the data distribution and the types of queries being asked of the data, is there a way to guarantee that we can automatically achieve the best performance (at least on average)? We use the technique of model-combining (specifically "stacking") and show that it provides a practical "insurance policy" that systematically outperforms any technique that relies on individual models for query approximation. Stacking has been used for regression, classification, and density estimation in past work and here we extend its applicability to query approximation. In particular it automatically adapts to the query distribution to generate predictions from weighted combinations of models. Empirical results demonstrate that the difference in accuracy between the combined model and the best individual model can be substantial.

The two primary novel contributions of this paper are (1) the introduction of the statistical notion of model averaging in the context of query approximation, and (2) the empirical demonstration that this approach provides systematically lower error rates than any single model on the data sets used in this paper.

2 Notation and Statistics for Model Comparison

We use the following notation in the remainder of the paper. We assume that we are given a data set with k binary attributes, n records and a sample S of queries coming from query distribution $\pi(Q)$. By P we denote the true unknown joint probability distribution on the k attributes. We use P_M to denote an estimate of P from the data, based on a probabilistic model M; $P_M(Q)$ is then an estimate of the value of $P(Q)$.

We use the following criteria to compare the performance of different models:

- **Time:** the online time cost is the time taken to answer the query Q online using the model P_M.

- **Memory:** the complexity of the learned model P_M (in practice, the number of bytes required to store the model).

- **Error:** the error is defined as $\sum_{Q \in S} |P^*(Q) - P_M(Q)|$, where $P^*(Q)$ is the observed empirical probability of the query Q in the training or the test data depending on whether Q is a count or a generalization query. In practice, we normalize the error as follows:

$$(2.1) E_{rel} = \sum_{Q \in S} [|P^*(Q) - P_M(Q)|] / \sum_{Q \in S} P^*(Q).$$

In this paper our primary focus is on the *accuracy* of different query approximation schemes, or equivalently, the error in answering queries as defined above. The online time t_P taken to compute the answer and the memory footprint are of interest only to a secondary degree, and the offline time cost of building the probabilistic models is not explicitly considered below (since it is viewed as a "one-time" resource investment, not of primary interest).

3 Probabilistic Models for Query Approximation

We evaluate the performance of several different classes of probabilistic models for query approximation. The algorithms for constructing and querying each of these models are described in detail in [17]. These models can be loosely categorized into three general classes: empirical data models, probabilistic models that are built offline, and probabilistic models that are built online when the query is posed.

Empirical data models considered include (1) a linear scan of the training data and (2) $q\%$ random samples of the training data for values of $q = 1, 5, 10, 15, 20$. For count queries the linear scan method produces exact answers and for generalization queries it effectively assumes that the future data counts will be exactly the same as those seen in the past for the query.

The class of offline probabilistic models consists of relatively simple probabilistic models where a full joint distribution on all k attributes is estimated once and for all "offline" before any queries are issued. The models in this class include the independence model, the Chow-Liu tree model, and mixtures of conditional independence Bernoulli models parameterized by the number of clusters N_C, taking values $5, 10, 25, 50, 80$.

Finally, the probabilistic models in the third class are considerably more complex in form, including Bayesian networks (BN) and maximum entropy (ME) models. For these models a full k-dimensional distribution is often not tractable to work with from either a time or memory viewpoint. Consequently, for this class, summary statistics in the form of itemsets (defined in the Appendix) are computed offline and when a query Q is issued, a joint distribution is constructed in real-time on the variables in Q. These methods never construct a full joint distribution on all k attributes but instead construct joint distributions on the query variables "on the fly" in real-time. Specifically we investigate the inclusion-exclusion model (IIE model), the BN model and the ME model based on itemsets, parameterized by the value of the threshold T in the definition of T-frequent itemsets. This parameter was chosen according to the algorithm described in [17]. For the Microsoft Web data we set T to $10, 30, 50, 70, 90$ and for the Quest data, we set to T to $20, 40, 60, 80, 100$ (both data sets are described below).

4 Query Approximation Performance of Individual Models

Since the primary focus of this paper is on model combining, we limit our discussion of the performance of *individual* models to a subset of the overall experimental results, sufficient to illustrate the main features of the individual models for query answering.

4.1 Description of Experimental Data Sets We conducted experiments on two real-world transaction data sets. The real-world data sets included the Microsoft Anonymous Web data set (publicly available at the UCI KDD archive, `kdd.ics.uci.edu`), and a simulated transaction data set from the IBM Quest data simulator[1]. The general characteristics of these data sets are provided in the Table 1.

4.2 Experimental Results We generated 1000 queries from a query distribution $\pi(Q)$, and evaluated different models with respect to the average memory taken by the model, the average online time taken to answer a query, and the average empirical error as defined in Equation 2.1. All experiments were performed on a Pentium III, 450 MHz machine with 128 Mb of memory.

The distribution $\pi(Q)$ was modeled as follows:

- fix the query lengths to $n_Q = 4$ or $n_Q = 8$;

- randomly select n_Q attributes according to the empirical probability of the attribute taking a value of "1";

- randomly select a value for each selected attribute according to its univariate probability distribution.

This choice of a query distribution $\pi(Q)$ is motivated by the fact that zero values for the attributes are more likely in sparse data sets than positive ones (e.g., see Table 1). Using purely random queries (randomly chosen attributes with randomly chosen values) would result in a preponderance of queries whose count is zero in the data (since any query consisting of more than one positively instantiated attribute will often not have occurred in the data).

Here we show the results for count queries on the Web data to illustrate the basic tradeoffs—qualitatively similar and consistent results have been found for generalization queries and for other data sets (further details can be found in [17]).

The plots in Figure 1 show the the average relative error versus the memory requirements for each model. The left plot corresponds to queries of size 4, the right plot to queries of size 8. Both axes on each plot are on a logarithmic (base 10) scale. The dotted line corresponds to a linear scan of the training data—it is error free for count queries, so that the logarithm of the error equals minus infinity. Thus, we only show how much memory the training data takes. Note that all models to the right of this dotted line are effectively impractical since that they are taking more memory than the full data set.

Different points on each curve were obtained by varying the value of the tradeoff parameter for the respective model corresponding to that curve. Of note is the fact that the number of clusters N_C (in the definition of the mixture models) and the threshold T
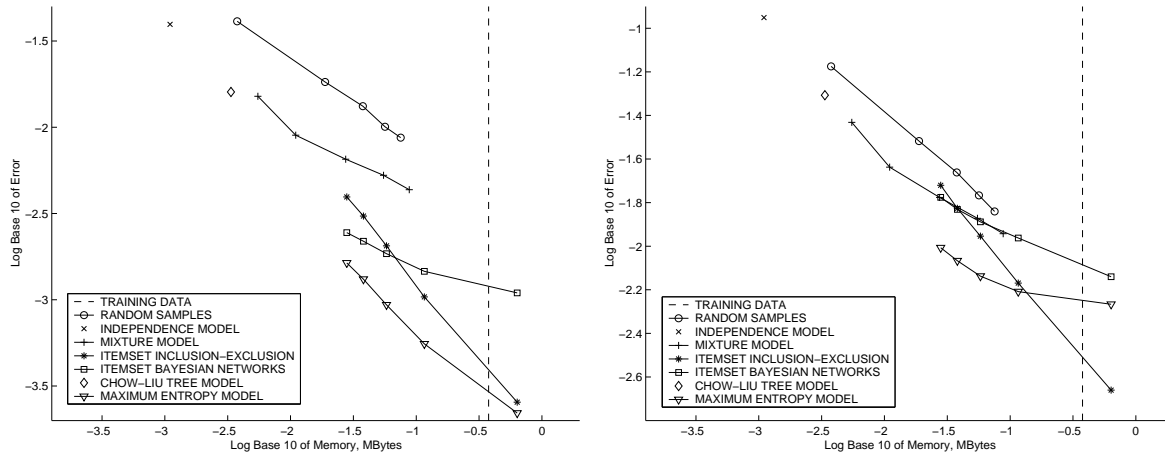
Figure 1: Error versus Memory: average relative error for 1000 queries of length 4 (left) and 8 (right) drawn from $\pi(Q)$ as a function of the average model complexity for the Web data, both on a log scale.
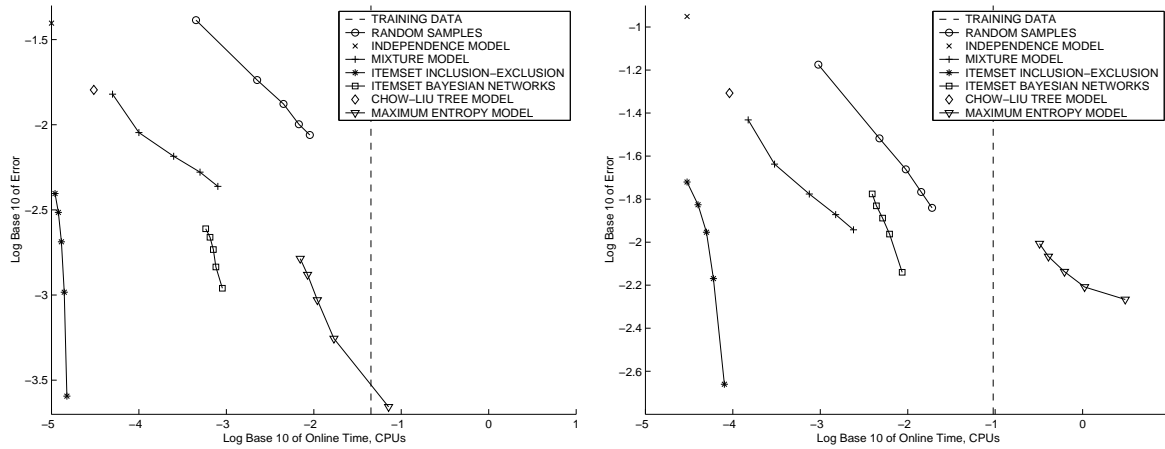

Figure 2: Error versus Online Query Time: average relative error for 1000 queries of length 4 (left) and 8 (right) drawn from $\pi(Q)$ as a function of the average online time for the Web data, both a log scale.

Table 1: General characteristics of the data sets: k is the number of attributes, n is the number of records, $N_{1's}$ is the number of 1's in the data, $E(N_{1's}) = N_{1's}/n$, $E(N_{1's})/k$ is the *density index*, $Std(N_{1's})$ is the standard deviation of the number of 1's in the record, and $Max(N_{1's})$ is the maximum number of 1's in any record.

	k	n	$N_{1's}$	$E(N_{1's}/k)$	$Std(N_{1's})$	$Max(N_{1's})$
MS Web Data Set	294	32711	98654	0.0102	2.5	35
Quest	300	78148	281626	0.0120	1.77	12

(in the definition of itemset-based methods) provide a mechanism for a direct tradeoff of accuracy with time and memory. In particular, the higher the number of clusters N_C, the more parameters are in the mixture model (linearly as a function of N_C) and the more accurately it can answer count queries on the training data. But as N_C increases it also requires more memory. Similarly, as the threshold T is reduced, more itemsets become T-frequent and are used in the models based on itemsets. Thus, for smaller values of T, the itemset-based models use more memory and presumably offer more accurate estimates than for larger values of T.

Figure 2 illustrates how the error varies as a function of the online time for each of the models, answering queries of length 4 (left) and 8 (right) on the Web data.

The primary points of interest from both graphs can be summarized as follows:

- There is significant variability among different models in terms of accuracy, memory footprint, and online query time.

- The independence models are the fastest, have the smallest memory requirements, but are the least accurate of all models. The Chow-Liu tree model provides a modest improvement in accuracy over the independence model but is also substantially less accurate than the more complex models.

- Random samples are also substantially less accurate than the more complex probabilistic models, yet can consume significant time and memory resources.

- The itemset inclusion-exclusion models are much faster than all of other models (with the exception of the independence model). The most complex itemset inclusion-exclusion model is comparable in accuracy with the best of the other methods.

- The maximum entropy models, constructed online from itemsets, are often the most accurate models, but take the most time (apart from the training data). They suffer from an exponential increase in online time as the query length grows, a consequence of the fact that inference in such models scales exponentially with the size of the largest clique in the underlying Markov random field graph.

- Bayesian networks, also constructed online from itemsets, are also among the most accurate models and take time intermediate between inclusion-exclusion and maximum entropy models. Because the models are built from itemsets, they also "inherit" the memory problems of itemset-based methods.

- The mixture models can provide comparable accuracies in the general range of the more accurate of the models, while providing reasonable and time and memory performance points.

Based on the experiments above (and other similar results not shown here due to space limitations) mixture models and the models based on itemsets (inclusion-exclusion models, Bayesian networks, and maximum entropy models) tend to be the most effective in general. However they can vary significantly (relative to each other) in terms of time and memory. An important point is none of these models alone dominates the others over all possible data sets and query distributions. Detailed examples illustrating this statement can be found in [17].

Since the choice of the model is dependent on the data distribution there is motivation to explore techniques that can adapt the model being used to both (a) the nature of the data at hand, and (b) the nature of the queries being issued (in the next section we illustrate how model performance can vary as a function of different queries, for the same data set).

5 Model Combining Using Stacking

In this Section we show that the model averaging technique of *stacking* can be used to combine models, such that the predictions from the combined model tend to perform better on average than any single model over a variety of data sets and query distributions.

5.1 A General Description of Model Averaging

Consider the general situation where we have a set of

models $\mathcal{M} = \{M_1, M_2, \ldots, M_K\}$ that are all candidate models for a true unknown probability distribution P. Instead of relying on any one of these models (i.e. conditioning on a specific model to make a prediction) a more appropriate approach from a statistical viewpoint is to average our predictions over all of the models (a technique known as called Bayesian Model Averaging (BMA) [9]):

$$(5.2) \qquad P(Q|r) \quad = \quad \sum_{k=1}^{K} P_{M_k}(Q|r)P(M_k|r),$$

where the term $P_{M_k}(Q|r)$ is a prediction for the query probability provided by model M_k. The second term, $P(M_k|r)$, is the probability that model M_k is the correct model, given the data r, or in effect, a Bayesian weight for the prediction from model M_k.

The calculation of the model weights is almost always intractable except for trivial problems, since it involves very high-dimensional integrals in parameter space. In current statistical practice Monte Carlo sampling techniques are widely used to generate approximate solutions to such equations. However, these methods are rather computationally intensive and not well-suited to the type of online real-time query answering problem being considered here.

5.2 A Data Driven Alternative to the BMA Equations An alternative to Bayesian estimation of the weighting coefficients $P(M_k|r)$ is to simply treat the weights as arbitrary unknown parameters and to estimate them in a data-driven manner to minimize an empirical loss function (see below). Let α_k represent the weight for model M_k, so that the prediction of our combined model is written as

$$(5.3) \qquad P(Q|\alpha_1, \ldots, \alpha_K, r) = \sum_{k=1}^{K} \alpha_k P_{M_k}(Q|r).$$

We can try to directly find the set of weights α_k that minimize the average difference between the prediction and the true probability for Q, e.g., minimize the expectation of the squared error:

$$E^{\mathrm{emp}}(\alpha_1, \ldots, \alpha_K) = \sum_{Q \in S} \left(\sum_k \alpha_k P_{M_k}(Q|r) - P(Q) \right)^2$$

$$(5.4)$$

as a function of the α's. Note that the averaging here is with respect to the sample from the query distribution $\pi(Q)$ with which queries are issued. Furthermore, the set of α's that minimize this expression will be optimal with respect to both the distribution of the data (as reflected by the true probability $p(Q)$) and also optimal with respect to the distribution of the queries $\pi(Q)$.

The resulting optimization problem of minimizing $E^{\mathrm{emp}}(\alpha_1, \ldots, \alpha_K)$ reduces to a straightforward exercise in solving a set of linear algebraic equations (i.e., linear regression), since the derivative of Equation 5.4 can be written as a linear equation in the $\alpha_1, \ldots, \alpha_K$ weights with all other terms known. The time complexity of solving this equation is $O(K^3 + K^2 N_Q)$. Since K, the number of models, is relatively small this means that the stacking weights can be determined quite quickly. If we constrain the α's to sum to one, or to be non-negative, we get a constrained convex quadratic programming problem, which can still be solved in polynomial time. In our experiments below we use non-negative coefficients constrained to sum to 1. Experimental results on using different types of coefficients (not shown here) indicate that using unconstrained coefficients produces almost identical estimates in practice to the constrained case (agreeing with the results reported in [3] for regression).

This general approach of learning model weights on a validation data set is known as "stacking" in the machine learning and statistics literature [21] and has been demonstrated to provide substantially improved predictive power over individual models for both regression [3] and density estimation [20].

Our adaptation of stacking to the query approximation problem has one main difference relative to prior work on model combining (e.g., [20]), namely, in optimization of the coefficients with respect to the query distribution $\pi(Q)$. In previous work the model coefficients were selected relative to the overall data distribution (e.g., minimizing with respect to the distribution $P(Q)$ for density estimation) whereas here we explicitly minimize with respect to query distribution $\pi(Q)$. This has important practical implications: the models selected for prediction (i.e., those given significant weight in the combined model) will be precisely those models that predict well for queries Q that are "well represented" in the population of queries being generated by $\pi(Q)$. If we have a set of distributions $\pi(Q)$, each for different users or different sets of users, and we estimate weights for each query distribution in this set, we can get very different weight-vector solutions in principle, each tuned to different individual query distributions.

6 Experimental Evaluation of Stacking

6.1 Stacking Compared to Individual Models In this section we compare the performance of the best of the individual models with the performance of stacking, for both the Web and Quest data sets and for both count and generalization queries. We omit both the maximum entropy model and the training data methods from consideration in these experiments since
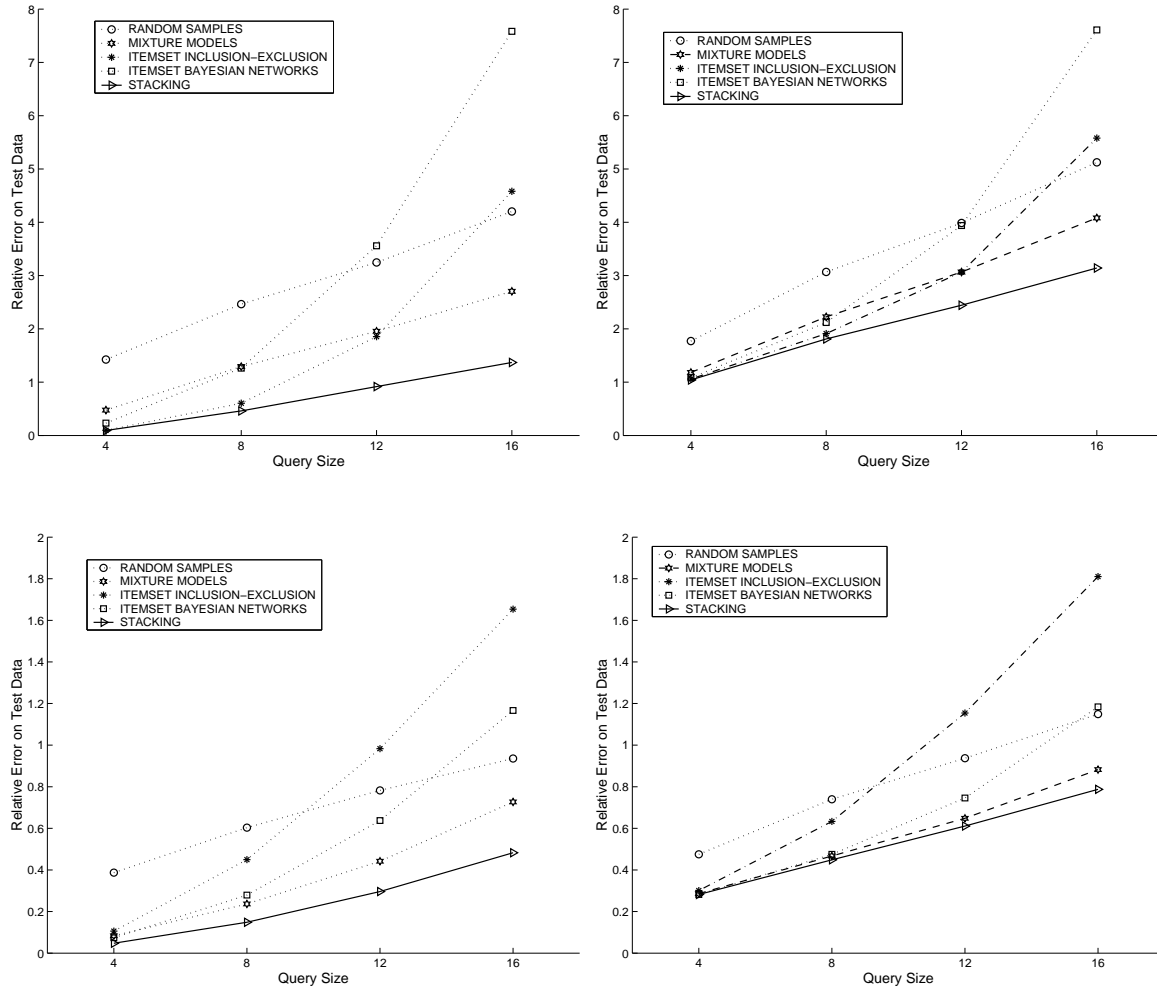
Figure 3: Comparing individual models with stacking on the Web (top plots) and Quest (bottom plots) data sets, plotting mean percentage error (relative to mean count size) as a function of query size for (a) count queries (left plots) and (b) generalization queries (right plots).

they are too slow in practice for large data sets to be of practical consideration. In all experiments below we use all of the other models described earlier. For the mixture models the number of components N_C was fixed at 80 and for random sampling a sample size of 20% of the original data was used. For itemset-based models the threshold T was set to 10 on the Web data and to 20 on the Quest data. This corresponds to selecting the most accurate model (or equivalently, the most complex model) for each of these models, at least for count queries. The query distribution $\pi(Q)$ was modeled as described in Section 4.2. The query sizes were set to 4, 8, 12, and 16.

Each of the data sets was randomly partitioned into 3 roughly equal disjoint subsets of rows: a training set, a validation set, and a test set. The training set was used to estimate parameters for each of the models, the validation set was used for estimation of the stacking parameters, and the test set was used as an independent test data set. Two sets of $N_Q = 500$ queries were randomly generated from $\pi(Q)$ for the purposes of estimation of stacking weights and evaluation of the models on test data. For count queries, the query answers provided by each model were scored against the training data. For generalization queries, the query answers provided by each model were scored against the test data. The process of partitioning the data, training the models, and generating and evaluating query results was repeated 25 times, and the average error recorded.

In Figure 3 we show the resulting average errors for both data sets and for both count and generalization queries, as a function of query size. For clarity only the results from the most accurate models are plotted. For example, the independence model is typically so inaccurate that it skews the graphs considerably when included.

There are a number of important points to be gained from the graphs:

- Accuracy decreases as queries become longer: this is a simple consequence of the increasing difficulty of estimating probabilities as dimensionality increases.

- Errors on generalization queries are higher than for count queries: this is to be expected as prediction is harder than memorization.

- The identity of the single model with the lowest error changes as a function of data set and query size. For example, in Figure 3, for count queries on the Web data set, the itemset inclusion-exclusion model has the lowest error of any individual model for query sizes of 4 and 8, but it has the second-highest error for query sizes of 16.

- The average error for mixtures and random sampling appears to increase roughly linearly as a function of query size, but the error for itemset-based methods increases non-linearly. This suggests that the itemset-based methods are quite sensitive to the nature of the queries being issued.

- Stacking has the lowest average error rate in all experiments: for both count and generalization queries, over all query sizes, and for both data sets.

6.2 Stacking Compared to the "Best" Single Model An obvious competitor to stacking is to select the single individual model that performs best on the validation data set. Note that the best model on the validation data set is not necessarily the model that will perform best on future test data: in fact, this "noise" in model selection is precisely one of the reasons why selecting a single model will on average not perform as well as model averaging. For the same set of experiments described in Section 6, the individual model that had the lowest error rate on the validation data set was identified in each of the 25 runs (for both count and generalization queries and for each of the data sets). The average accuracy of the predictions of this "best single model" on the 500 test queries were then recorded on the test data set (for each of the 25 runs) and compared to the predictions of a stacking model that was trained and tested on the same data.

Figures 4 and 5 show the results obtained where the average percentage decrease in error obtained by using stacking (rather than a single best model) are shown. The percentage improvement in error ranges in size from about 1% (typically on short queries) to up to 50% (typically on longer queries). This phenomenon is well-known in statistics and machine learning, namely, that the selection process itself is noisy, and averaging will systematically outperform virtually any algorithm that uses the data to select a single model for prediction [21, 3, 20]. In fact, stacking also outperforms the single best model chosen on the test data set (results not shown), the so-called "cheating" method, validating similar results found earlier in the context of regression [3] and density estimation [20].

6.3 Online Adaptability of Stacking to a User Query Distribution Stacking also allows us to adapt a model to a user's queries in an online fashion. To illustrate this we ran the following experiment. We generated a sequence of queries, where queries were simulated as coming from a mixture of distributions for length 4 and length 16 queries, where for each length the queries are generated in the manner described earlier. A
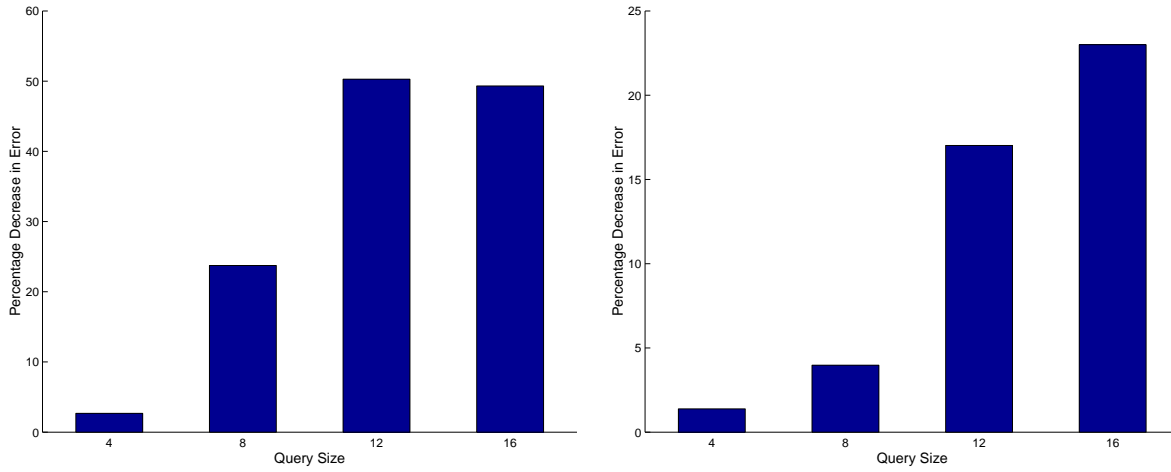
Figure 4: The mean percentage decrease in error obtained by stacking relative to selecting a single model, for the Web data, as a function of query size for (a) count queries (left plot) and (b) generalization queries (right).
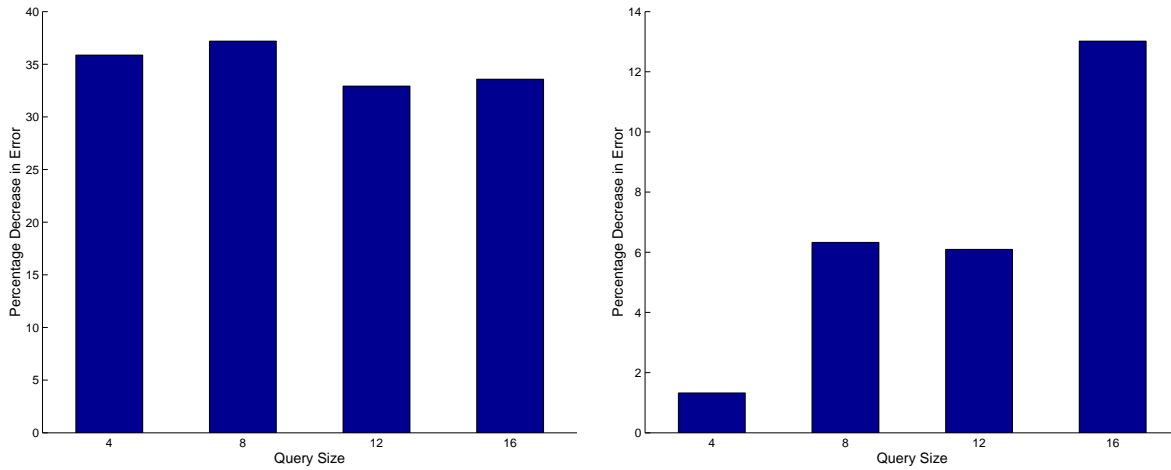

Figure 5: The mean percentage decrease in error obtained by stacking relative to selecting a single model, for the Quest simulated data, as a function of query size for (a) count queries (left plot) and (b) generalization queries (right).
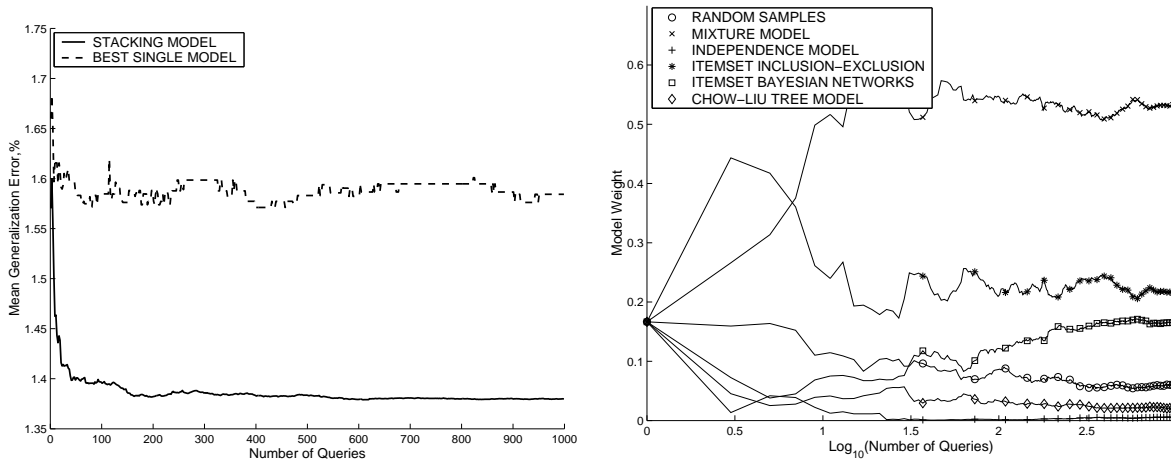

Figure 6: Dependence of the test set error (left plot) and the weights α assigned to individual models (right plot) on the number of queries available for selecting the best individual model and training the stacking model.

fixed size training data set, validation data set, and test data set were generated in roughly equal proportions for the Web data set. Thus, we are simulating the practical situation of having a substantial amount of raw data on which to build and test models, but having a limited amount of queries available from which to estimate $\pi(Q)$. We trained the individual models and the stacking algorithms on the full-sized training data set in the same manner as described in the earlier sections. For each set of queries $Q_1, \ldots, Q_t$, for $t = 1, \ldots, 500$, we simulated the effect of having to select a single "best" model from the validation set, and having to train a stacking model, based on only t queries seen so far. For each value of t, the selected "best" model and the stacked model made predictions and were scored on the test data set (this provides a fair estimate of their overall accuracy: one could also have gotten a somewhat noisier estimate of their accuracy by simply doing "one-step ahead" prediction on query Q_{t+1} for each t).

The left plot on the Figure 6 shows how the generalization error achieved by stacking and the single-best models depends on the number of observed queries. The stacking model quickly adapts to the query distribution even with only 40 or so queries. It also converges to a significantly lower asymptotic error rate than the single-model method. The right plot for Figure 6 illustrates how the individual model weights α_k estimated via the stacking procedure change as the number of observed queries increases. The weights quickly converge to placing most of the weight on the mixture and the itemset inclusion-exclusion model. This is not surprising since according to Figure 3 the mixture model is the best single model on queries of size 16, and the itemset inclusion-exclusion is the best single model on queries of size 4. Once again the weights become relatively stable quite quickly (i.e., only a small number of queries are needed to converge to the optimal weights).

7 Conclusions

We introduced a new model averaging technique for query approximation that is accurate, computationally efficient, and optimal in the sense that the combined model uses weighting coefficients designed to perform optimally in terms of prediction, relative to the specific data set being queried and the distribution of queries being issued. We demonstrated that model averaging can provide more accurate query answers than a variety of individual probabilistic models over a range of data sets and query distributions. Model averaging also significantly outperforms the method of selecting the single best model on a validation data set, being up to 50% more accurate for the data sets and queries in this paper. The methodology for generating the model combining weights is scalable, involving a straightforward optimization problem. The method is quite suitable for online implementation and can be straightforwardly generalized to a constrained optimization problem when memory and time resource constraints are present.

References

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The AQUA approximate query answering system. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 574–576. New York, NY: ACM Press, 1999.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the Twentieth International Conference on Very Large Data Bases (VLDB'94)*, pages 487 – 499. San Francisco, CA: Morgan Kaufmann Publishers, 1994.

[3] L. Breiman. Stacked regressions. *Machine Learning*, 24:49—64, 1996.

[4] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 3:111–122, 2000.

[5] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM Transactions on Database Systems*, 9(2):163–186, 1984.

[6] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'01)*, pages 199–210. New York, NY: ACM Press, 2001.

[7] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'01)*, pages 461–473. New York, NY: ACM Press, 2001.

[8] R. Greiner, R. Grove, and A. Schuurmans. Learning bayesian nets that perform well. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 198–207. San Francisco, CA: Morgan Kaufmann Publishers, 1997.

[9] E. Leamer. *Specification Searches - ad hoc Inference with Nonexperimental Data*. New York: John Wiley and Sons., 1978.

[10] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 19(2):1–11, 1990.

[11] D. Margaritis, C. Faloutsos, and S. Thrun. Netcube: A scalable tool for fast data mining and compression. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[12] Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the 1998 ACM SIGMOD Intl. Conf. on Management*

of Data, pages 448–459. New York, NY: ACM Press, 1998.

[13] M. Muralikrishna and D. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceeding of ACM SIGMOD Conference on Management of Data (SIGMOD'88)*, pages 28–36. New York, NY: ACM Press, 1988.

[14] D. Pavlov, H. Mannila, and P. Smyth. Probabilistic models for query approximation with large sparse binary data sets. In *Proceedings of the UAI-2000*, pages 465–472. San Francisco, CA: Morgan Kaufmann Publishers, 2000.

[15] D. Pavlov, H. Mannila, and P. Smyth. Beyond independence: Probabilistic models for query approximation on binary transaction data. *IEEE Transactions On Knowledge And Data Engineering*, 2002 (to appear).

[16] D. Pavlov and P. Smyth. Probabilistic query models for transaction data. In *Proc. of Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 164–173. ACM Press, 2001.

[17] D. Yu. Pavlov. Probabilistic query models for transaction data. *Unpublished PhD Dissertation, University California, Irvine, http://www.ics.uci.edu/p̃avlovd/research.html*, 2002.

[18] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd Intl. Conf. on Very Large Data Bases (VLDB'97)*, pages 486–495. San Francisco, CA: Morgan Kaufmann Publishers, 1997.

[19] J. Shanmugasundaram, U. Fayyad, and P. Bradley. Compressed data cubes for OLAP aggregate query approximation on continuous dimensions. In *Proceedings of KDD-99*, pages 223–232. New York, NY: ACM Press, 1999.

[20] P. Smyth and D. Wolpert. Linearly combining density estimators via stacking. *Machine Learning*, 36(1/2):59–83, 1999.

[21] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241—259, 1992.

A   Appendix: Definition of Itemsets and Their Properties

For a given data set, an *itemset* is defined to be either a single positively initialized attribute or a conjunction of mutually exclusive positively initialized attributes. An itemset is called **T-frequent** if its count in the data is at least T, where T is some predefined non-negative threshold. The **size** of an itemset is the number of conjuncts it is defined on.

Itemsets have several attractive properties for query answering in binary transaction data sets:

THEOREM A.1. *To provide an error-free answer to any selectivity (count) query expressed as a Boolean expression on the attributes of the binary table r it is sufficient to know all 0-frequent itemsets and their counts in r.*

THEOREM A.2. *If an itemset on a set of variables X is T-frequent then any count query Q (expressed as an arbitrary Boolean expression) involving variables that are all contained in the set X can be answered exactly using only T-frequent itemsets.*

Proofs can be found, for example, in [16].

There exist well-known efficient algorithms to compute all the itemsets from large binary tables, e.g., [2]. Provided that the data are sparse, the running times of these algorithms have been found in experimental analyses to be linear in both the size of the table and the number of frequent itemsets.

We view itemsets as providing summary information about the original data and use them as a basis for model learning. The quality of both (a) the summary information represented by the set of all T-frequent itemsets and (b) the probabilistic model learned from these itemsets, varies as a function of the value of the threshold T. Lower values of T mean more information stored about the data, and higher quality models in principle, but at the cost of extra memory and (potentially) longer online time to query the model.