

# Guessing the extreme values in a data set: a Bayesian method and its applications

Mingxi Wu · Chris Jermaine

Received: 19 March 2008 / Revised: 1 December 2008 / Accepted: 10 December 2008  
© Springer-Verlag 2009

**Abstract** For a largenumber of data management problems, it would be very useful to be able to obtain a few samples from a data set, and to use the samples to guess the largest (or smallest) value in the entire data set. Min/max online aggregation, Top-k query processing, outlier detection, and distance join are just a few possible applications. This paper details a statistically rigorous, Bayesian approach to attacking this problem. Just as importantly, we demonstrate the utility of our approach by showing how it can be applied to four specific problems that arise in the context of data management.

**Keywords** Sampling · Online aggregation · Monte Carlo · Extreme values · Bayesian

## 1 Introduction

This paper deals with a ubiquitous problem in data management: guessing the maximum/minimum value (or some other extreme statistics) over a data set. Formally, given an arbitrary function  $f()$  which maps any input data point  $d$  in a data set  $D$  to a real value, our goal is to accurately guess the  $k$ th largest  $f()$  value  $f_{(k)}$  for all  $d \in D$ , such that  $|\{f(d) : f(d) > f_{(k)} \wedge d \in D\}| = k - 1$ .

This particular problem arises in many applications, for example:

- As pointed out by Donjerkovic and Ramakrishnan [9], in top-k query processing knowing the cutoff value beforehand allows the “top-k” portion of the query to be transformed into a relational selection. The resulting query can then be processed without modification to the database engine.
- In outlier detection for data mining, the simple yet effective algorithm [4] prunes points from the outlier candidate set when it has been determined that there are too many points that are close by the candidate. If it were possible to accurately guess the distance to a point’s  $k$ th nearest-neighbor, this pruning could be done without actually finding those close-by points.
- In distance join processing [13,33], the goal is to find the  $k$  closest pairs over two different data sets. The fact that  $k$  is supplied to the join (as opposed to a cutoff distance) makes the query more useful, but it greatly complicates the computation. If the cutoff distance were known beforehand, then the problem can be solved using any one of a large number of efficient algorithms [3,7,22].
- In spatial anomaly detection [35], given a spatial data set placed on an  $n \times n$  grid, the goal is to find the rectangular regions within which subsets of the data set exhibit anomalous behavior. The naive algorithm will enumerate all of the  $O(n^4)$  rectangles and compute an anomalous score for each rectangle. If we can classify the rectangles into groups, and estimate the  $k$ th largest score for each group, we can potentially prune the entire group without further computing the scores associated with each group member.

Many other applications exist, though space precludes listing them all here.

Under certain circumstances, guessing  $f_{(k)}$  is trivial. If  $f(d)$  simply returns an attribute of  $d$ , then the solution can

---

M. Wu (✉) · C. Jermaine  
Computer and Information Science and Engineering Department,  
University of Florida, Gainesville, FL 32611, USA  
e-mail: send2mingxiwu@gmail.com

C. Jermaine  
e-mail: cjermain@cise.ufl.edu

be as easy as pre-computing and storing the largest  $k$  attribute values from the data set. However, the problem can become arbitrarily difficult depending on the nature of  $f()$ . In the general case,  $f()$  may encode an unanticipated, arbitrary multi-attribute relational selection predicate—that is,  $f(d)$  returns  $-\infty$  if some selection predicate does not evaluate to *true*. In other cases,  $f()$  may perform an arbitrary, non-linear numerical computation over the attributes of  $d$  that is impossible to anticipate.

### 1.1 A Bayesian approach

In this paper, we propose a novel approach to solving this problem. Since we are trying to guess extreme values for any arbitrary and unanticipated function  $f()$ , we argue that it is impossible to solve the problem by using a statistical synopsis to model the data. Models for the data are often useful for describing what is “typical”, but the extreme value queries we are interested in specifically refer to the outliers in the data. Thus, standard approaches are of limited utility. For example, consider a 1% sample of a database having 100 million records where we are interested in  $f_{(10)}$ , but we have no prior knowledge of  $f()$  and so it is not possible to bias the sample to larger  $f()$  values. Since we have less than a 10% chance of sampling any of the records resulting in one of the 10 largest  $f()$  values, how can we guess  $f_{(10)}$  with high accuracy?

Thus, rather than trying to guess  $f_{(k)}$  by modeling the data, we instead guess  $f_{(k)}$  by watching and modeling the behavior of the *queries* we have seen.<sup>1</sup>

Some query aggregate functions may result in  $f()$  values that are typically very small, with a few tremendous outliers that greatly boost the value of  $f_{(k)}$ . Some query aggregate functions may result in  $f()$  values that are tightly, normally distributed around the mean  $f()$  value, meaning that  $f_{(k)}$  is not too different from the typical  $f()$  value. As queries are asked, our method watches and learns what “typical” queries tend to look like. Then, when a new query is asked, we look at the first few  $f()$  values obtained and decide (in a statistically meaningful way) which type of the “typical” queries we are experiencing.

Of course, we may be wrong when we guess what type of query is being asked, and so there is a degree of uncertainty with our belief. This uncertainty is incorporated into the probabilistic guarantee on the accuracy of our guessed  $f_{(k)}$ . In this way, our method is an example of a so-called *Bayesian* statistical technique, in that we make use of a “prior” or guessed query distribution in order to associate a belief with the type of query that has been issued.

### 1.2 Our contributions

This paper has the following technical contributions:

- We propose a new method for guessing the answer to an extreme value query over an arbitrary function. The method is statistically rigorous and makes use of unique Bayesian statistical techniques. Such techniques have been mostly ignored in the data management literature to date.
- Along with the approximate answer, our method returns the distribution associated with the guess’ error, and so it can be used to associate confidence bounds on the guess’ accuracy.
- We devise a method to learn the prior query distribution by watching query results as they are produced. The learning algorithm requires that for each query result, we compute only three aggregate values. In this way, the learning method is inexpensive, and can easily be incorporated into a DBMS or a specific application with little or no cost.
- We argue for the utility of our approach by detailing four separate applications of our method to specific problems that occur when dealing with large databases.

## 2 Problem definition

In this section, we formalize our problem and describe the solution that we study in the paper.

### 2.1 Estimating the extreme value

Our goal is to provide estimation algorithms that will support processing for extreme value queries of the form:

```
SELECT g1(d1)
FROM D AS d1
WHERE g2(d1) AND k - 1 = (
    SELECT COUNT(*)
    FROM D AS d2
    WHERE g2(d2) AND g1(d1) < g1(d2));
```

In the above query,  $g_1()$  is an arbitrary function that maps a tuple  $d$  to a real value<sup>2</sup> and  $g_2()$  is a relational selection predicate. This query asks for the  $k$ th largest  $g_1(d)$  value over all the database tuples that satisfy the selection predicate  $g_2()$ . If we change “ $g_1(d_1) < g_1(d_2)$ ” to “ $g_1(d_1) > g_1(d_2)$ ”, then the query is easily modified to ask for the  $k$ th smallest  $g_1(d)$

<sup>1</sup> We will use the term *query* very loosely in the paper, and its exact meaning will depend upon the application.

<sup>2</sup> For ease of exposition, we assume that each tuple maps to a distinct real value.

value when the selection predicate defined by  $g_2()$  is satisfied. For ease of exposition, in the remainder of the paper we assume that the search is for a large value.

The fact that we have two separate functions  $g_1()$  and  $g_2()$  complicates things a bit, so we encode both individual functions within a single function  $f()$ :

$$f(d) = \begin{cases} g_1(d) & \text{if } g_2(d) \text{ is true} \\ -\infty & \text{otherwise} \end{cases}$$

We use the notation  $f_{(k)}$  to denote the answer to the query.

### 2.2 A natural estimator

Assuming that there are no index structures to help us locate tuples with specific  $f()$  values, an obvious solution to the problem is to sequentially scan the database once and evaluate  $f()$  over each tuple. Then we can return the  $k$ th largest  $f()$  value encountered. However, this algorithm may be too slow if the database is large, or if  $f_{(k)}$  must be evaluated repeatedly for different functions (as in the application to outlier detection that we will consider).

The fundamental idea in this paper is that, in order to obtain a sub-linear-speed algorithm to compute  $f_{(k)}$ , one can use a simple random sample (without replacement) from the database. By examining the  $f()$  values in this sample, it may be possible to estimate the  $k$ th largest/smallest value in the query result set.<sup>3</sup> The estimator that we study works as follows. Suppose that we have access to  $n$  samples from a database of size  $N$ . Then a natural estimator for the  $k$ th largest value in the query is the  $k$ th largest value in the sample, where  $\frac{k'}{n} = \frac{k}{N}$ . Since to make use of this estimator,  $k'$  must be an integer, we use  $k' = \lceil \frac{n}{N} \times k \rceil$ .

In the remainder of the paper, we use  $\widehat{f}_{(k)}$  to denote the ( $k'$ )th largest value in the sample. Also note that, since our estimator and its target work within the context of the real set obtained by mapping each database tuple to a real value, in the remainder of the paper, “database sample” will refer to the real values corresponding to the sampled database tuples.

For example, suppose that we wish to guess the largest value in a database of size 15. Applying  $f()$  to each tuple, we obtain the real set:

$$\{1, 2, -\infty, 4, 5, -\infty, 7, 8, 9, -\infty, 11, -\infty, -\infty, 14, 15\}$$

Thus,  $f_{(1)} = 15$ . Now, we take a five-item sample from this set, which happens to be:

$$\{11, -\infty, 4, 1, -\infty\}$$

<sup>3</sup> Although we consider the case where a single sample is used for a single estimate, the technique developed in this paper can easily be extended to deal with the online case, where the entire randomized database will be scanned. At each instant during the scan, the set of tuples retrieved thus far is a sample of the database.

Then our estimator  $\widehat{f}_{(1)}$  is 11, where  $k' = \lceil \frac{5}{15} \times 1 \rceil$ .

Characterizing the accuracy of this estimator is far from trivial. The fundamental question we ask in this paper is:

*How does the estimator  $\widehat{f}_{(k)}$  relate to the true  $f_{(k)}$ ?*

Given a rigorous characterization of this relationship, it is possible to both correct  $\widehat{f}_{(k)}$  to obtain an even better estimate, and to characterize the accuracy of the estimator. Since the difference between  $\widehat{f}_{(k)}$  and  $f_{(k)}$  is affected by the scale of the values under consideration, this paper studies the behavior of the ratio  $\frac{f_{(k)}}{\widehat{f}_{(k)}}$  as a way to characterize the accuracy of the estimator. In particular, we are interested in characterizing the *distribution* of this ratio, because this distribution facilitates the use of  $\widehat{f}_{(k)}$  to produce confidence bounds on  $f_{(k)}$ . For example, if we know that there is a 90% chance that the ratio is between  $l$  and  $h$ , then there is a 90% chance that  $f_{(k)}$  is between  $l \times \widehat{f}_{(k)}$  and  $h \times \widehat{f}_{(k)}$ .

### 3 Overview of our approach

This section gives an overview of the approach that we use to characterize the ratio  $\frac{f_{(k)}}{\widehat{f}_{(k)}}$ . First, we discuss the statistical property of the database most relevant to the characterization: the shape of the right tail of the distribution of  $f()$  values. Then we give an overview of a unique Bayesian approach to dealing with the importance of the tail’s shape.

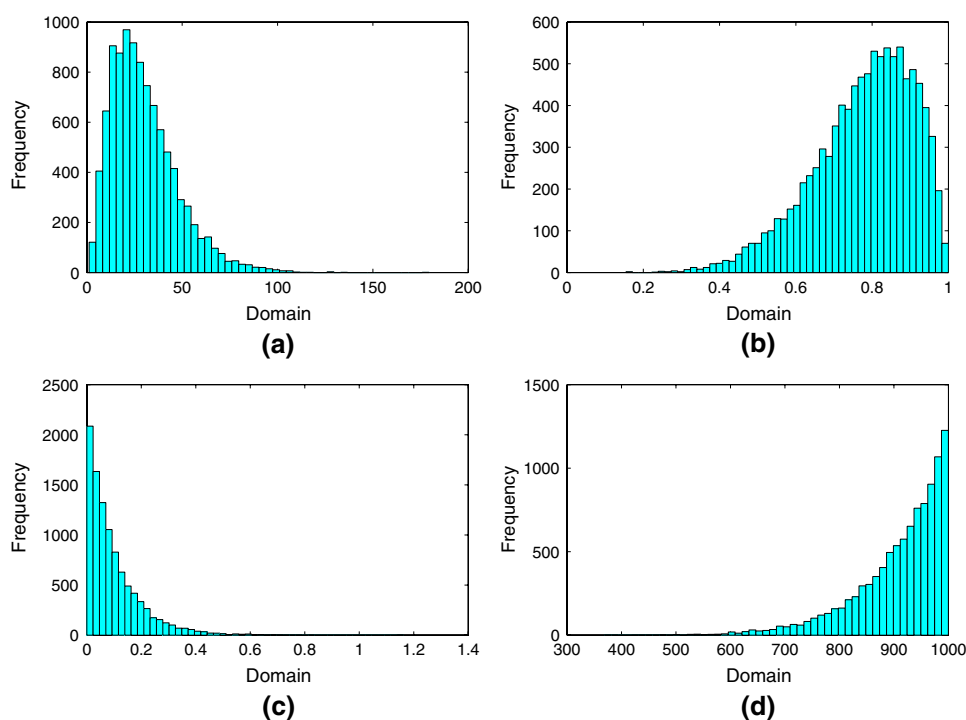
#### 3.1 Importance of the query shape

Unlike many other estimation problems, characterizing the estimator  $\widehat{f}_{(k)}$  is extremely challenging, because unlike classical estimation problems where simple statistical properties such as the distribution’s variance are important, the actual shape of the query result set’s distribution is most closely related to the accuracy of the estimator. This is best illustrated by an example.

Figure 1 depicts a set of histograms showing the distributions of four synthetic query result sets. Each query result set contains 10,000 values. Now, imagine that we wish to use a sample of size 100 to compute  $\widehat{f}_{(1)}$ , and we ask: How accurate is  $\widehat{f}_{(1)}$  as an estimate for  $f_{(1)}$ ? To answer this question, we re-sample (without replacement) 100 times to produce 100 different  $\widehat{f}_{(1)}$  values, and compute the median for  $\frac{f_{(1)}}{\widehat{f}_{(1)}}$  over each data set. The median values recorded are 3.07, 1.06, 4.45 and 1.01 for (a), (b), (c), and (d), respectively.

The relative magnitudes of these four values can be explained by examining Fig. 1. The distributions corresponding to queries (a) and (c) have long tails to the right, so one has only a small chance of sampling any values close to the tip of

**Fig. 1** The histograms of four different query result sets with different domains (scales). **a** and **c** have long tails to the *right*. **b** and **d** have no tails to the *right*



the right tail where  $f_{(1)}$  is located. Therefore, we observe a large ratio between the true answer and the estimator. In contrast, the shapes of query (b) and (d) have no tail to the right, so one would expect that a sample has an excellent chance of including values close to  $f_{(1)}$ . As a result, we observe a ratio close to one for these two queries.

### 3.2 Basics of the Bayesian approach

Since the query shape is so important when evaluating the accuracy of  $\widehat{f}_{(k)}$ , it must be incorporated into the process of characterizing the ratio  $\frac{f_{(k)}}{\widehat{f}_{(k)}}$ . Our basic approach is to assume that there exists a large number of possible query shapes: from “easy” shapes with no skew to very “hard” shapes with a heavy right skew. Each shape has a weight or probability associated with it that specifies the extent to which we think this is the current shape we are experiencing—representing such a belief with a probability is the hallmark of the so-called “Bayesian” statistical approach [21].

The initial set of weights that we start out with before any data have been encountered are known using Bayesian terminology as the *prior distribution*. While there are many ways to develop a reasonable prior distribution, we choose to learn the prior from the historical query workload. Then, as data from the current query are encountered, the weights are updated in a statistically rigorous fashion to take into account the new data. In Bayesian terminology, the updated weights represent the *posterior distribution*. These updated weights are then used to produce confidence bounds. For example,

if a database sample of a reasonable size is obtained that is consistent with a query having a heavy rightward skew, the updated weights will tend to favor query shapes with corresponding rightward skew, and the confidence bounds for the accuracy of  $\widehat{f}_{(k)}$  that we report will be suitably wide.

#### 3.2.1 The “Dangers” of the Bayesian approach

At first glance, assuming the existence of a prior distribution may seem dangerous. Since we will learn our prior from the previously observed queries, we are assuming that a new query will never be totally different from all of the queries in the training workload. In the case where we see a “new” query, the shape corresponding to the new query will necessarily have a zero prior weight, since the query was totally unanticipated. If the new query has a tail that is far nastier than anything else we have ever seen, then we may be too aggressive with our confidence bounds—this is the danger inherent in the Bayesian approach.

In fact, related dangers are inherent in *all* estimation techniques that do not have access to all of the data, including classic methods that are widely used in the data management literature. For example, consider the classical, sampling-based estimator for a SUM SQL query [11, 12]: first a  $1/\alpha$  sample of the database is taken, then the query is applied to the sample and the result is scaled up by a factor of  $\alpha$ . It is an often-ignored fact that in order to bound the accuracy of such an estimate in the classical fashion, *the variance of the estimator is also estimated from the same sample*. If the variance estimate is too low (which may be the case if there is

one particularly high-value record that did not appear in the sample) then any resulting confidence bounds are worthless. The implicit assumption underlying the classic method is that the database characteristic in question—the variance—can be estimated accurately from the sample. In comparison, the Bayesian approach makes an explicit assumption regarding the availability of a prior distribution. In either case, the possibility of an error exists. In fact, a statistician from the so-called “Bayesian” school would argue that it is better to make such assumptions explicit using a prior than to “hide behind” arguments such as the unbiasedness of a variance estimate. As we will show experimentally, our application of the Bayesian approach is very robust to errors in the prior, and turns out to be quite successful in practice.

### 3.3 Proposed Bayesian inference framework

Given this background, we now describe the three steps of our Bayesian inference framework:

1. The **learning phase** uses statistical methods to build a prior shape model composed of a number of candidate shape patterns. Each shape pattern represents a class of queries. A weight is assigned to each shape pattern, indicating how likely a future query’s shape matches that shape pattern. Both the weights and the shape patterns are learned offline, from the historical query workload using an EM algorithm [6].
2. The **characterization phase** derives an error distribution for each learned shape pattern. Before we can use the learned prior shape model to predict the behavior of  $\widehat{f^{(k)}}$  on a real-life query, as a preparation for the next phase we need to derive the distribution of  $\frac{f^{(k)}}{\widehat{f^{(k)}}$  for each learned shape pattern. This is done using Monte Carlo methods [29], after the shape model has been learned but before it is time to actually answer an extreme-value query.
3. The **inference phase** uses the results of the characterization phase and a sample from the new query’s result set to produce the error distribution for the actual query under consideration. The characterization phase applies only to the learned shape patterns, and not to any real-life query. When it is time to actually answer a query online, the prior weights of the shape model are updated based upon the observed samples to produce the posterior weights. Using the posterior weights and the error distribution for each shape pattern, an error distribution for the ratio  $\frac{f^{(k)}}{\widehat{f^{(k)}}$  is obtained. Since  $\widehat{f^{(k)}}$  can be computed from the sample, confidence bounds on  $f^{(k)}$  can easily be derived from the resulting distribution.

The next three sections of the paper describe each of the three phases in more detail. In these sections, we simplify the

exposition by assuming  $f^{(k)}$  never returns  $-\infty$ ; that is, the size of the query result set is the database size. In the Experiments Section where the framework is actually applied to two specific problems, we will discuss how to remove this assumption when necessary.

## 4 The learning phase

Any Bayesian method requires a generative, probabilistic model for the data. The process should be both general (in the sense that it allows the production of any data set that might be observed) and specific (in the sense that it produces all important properties of the underlying data and is tailored for the specific problem at hand).

In our case, each individual “data point” that is produced by the generative process is a single query result set. Since (as described in Sect. 3.1) the shape of the query result set is so important in determining the quality of the estimator  $\widehat{f^{(k)}}$ , the generative model will pay special attention as to how the shape is handled.

Informally, we assume the following generative process for producing each query result set:

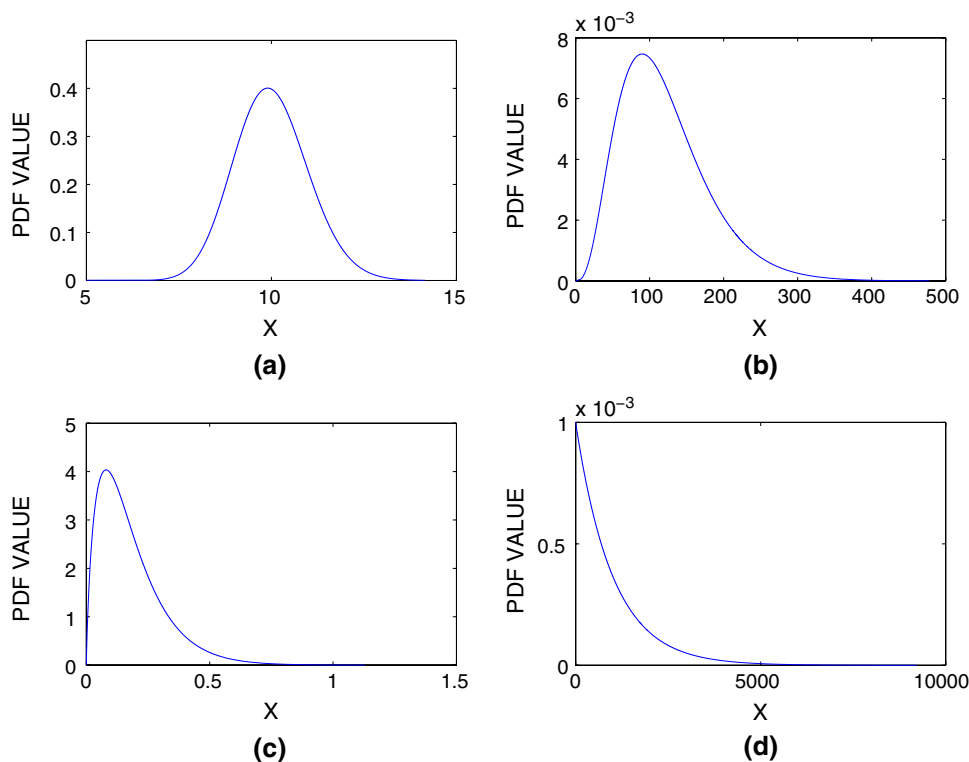
- First, a biased die is rolled to determine which shape pattern the query result set will be generated by. We assume the existence of some set of  $c$  different shape patterns, and the die roll selects one of them.
- Next, an arbitrary scale for the query is randomly selected. This scale defines the magnitude of the items in the query result set. In Fig. 1, the scale determines how large the labels are on the  $X$ -axis.
- Finally, the shape and scale are used as inputs to instantiate a parametric model for the data. For reasons described subsequently, we will make use of the Gamma distribution from statistics as our parametric model. This distribution is repeatedly sampled from to produce the query result set.

Given the intuitive process described above, the next step is to formalize it. Mathematically, this is done by defining a *probability density function* (PDF) for the process. This is a function that takes as an input a query result set, and returns how probable it is that this query result set would be produced by the process. After defining the PDF, we will then consider how to “learn” the model; that is, we consider how to tailor the model to a specific query workload.

### 4.1 Choosing an appropriate parametric model

It is first necessary to choose some parametric distribution to model the query shape. Given the discussion of Sect. 3.1, there is one overarching concern: our distribution must be

**Fig. 2** The PDFs of four Gamma distributions with increasing skew to the *right* from **a** to **d**



able to model arbitrarily long tails to the right. Modeling all of the dips and bumps of a real-life distribution is not necessary, because we are concerned only with the relationship between the distribution's tail and the rest of its mass.

Given this consideration, the Gamma distribution family becomes a natural choice, since it can produce a shape with arbitrary right-leaning skew. Figure 2 shows the PDFs of four instances of the Gamma family, each with increasing skew and longer tails to the right. Figure 2a depicts a bell (normal) shape, which does not have any skew to the right and only a very short tail relative to the distribution's variance, whereas Fig. 2d is highly skewed to the right with an exceedingly long tail.

We stress that though the Gamma distribution underlies our model, we do *not* assume that the  $f()$  values in the database look anything like a Gamma distribution. The Gamma distribution is used only to model the relationship between the values in the far right tail of the data distribution and the values that are more likely to be sampled—those that are closer to the main body of the distribution. The Gamma distribution does this well because of its ability to take on shapes having arbitrary skew. As we will show experimentally, using the Gamma distribution, our method can handle data sets that could not possibly have been sampled from a Gamma distribution, including those with a left skew and those with multiple modes, including very small modes or “bumps” far out in the right tail.

#### 4.2 Deriving the PDF

We now turn our attention to deriving the PDF associated with the resulting, three-step, generative process. Formally, the PDF of the Gamma distribution can be expressed in terms of the Gamma function  $\Gamma$ :<sup>4</sup>

$$p(x|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \quad x > 0 \quad (1)$$

In Eq. 1, the parameter  $\alpha > 0$  is known as the *shape parameter*, since it influences the shape or skew of the distribution, while the parameter  $\beta > 0$  is called the *inverse scale parameter*, since  $\frac{1}{\beta}$  influences the domain (scale) of the distribution.

Let  $\vec{y} = \langle d_1, \dots, d_N \rangle$  denote a query result set which has  $N$  tuples. Assuming that the tuples are independently drawn from a Gamma distribution, using Eq. 1 the likelihood of observing a query  $\vec{y}$  given  $\alpha$  and  $\beta$  is:

$$\begin{aligned} p(\vec{y}|\alpha, \beta) &= \prod_{i=1}^N \left\{ \frac{\beta^\alpha}{\Gamma(\alpha)} d_i^{\alpha-1} e^{-\beta d_i} \right\} \\ &= \frac{\beta^{N\alpha}}{\Gamma^N(\alpha)} M^{\alpha-1} e^{-\beta S} \end{aligned} \quad (2)$$

In Eq. 2,  $M = \prod_{i=1}^N d_i$  and  $S = \sum_{i=1}^N d_i$ .

<sup>4</sup> The Gamma function is defined as  $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt$ , where  $\alpha > 0$ .

Since we are interested in characterizing a ratio, we are uninterested in the scale parameter and do not want to bias our model to any particular scale. Thus, we treat the inverse scale parameter  $\beta$  as a random variable that is uniformly chosen from  $(0, \max)$ , where  $\max$  is a huge number chosen to be large enough that it permits a scale that is arbitrarily small. Then, the likelihood of observing a query  $\vec{y}$  given the shape  $\alpha$  and unknown  $\beta$  is given by:

$$p(\vec{y}|\alpha) = \int_0^{\max} p(\beta) \times \frac{\beta^{N\alpha}}{\Gamma^N(\alpha)} M^{\alpha-1} e^{-\beta S} d\beta$$

$$\approx \frac{1}{\max} \times \frac{M^{\alpha-1}}{\Gamma^N(\alpha)} S^{-N\alpha-1} \Gamma(N\alpha + 1) \tag{3}$$

Equation 3 is the result of taking expectation of Eq. 2 with respect to  $\beta$ . It shows that evaluating the likelihood of a given query result set requires exactly three aggregate values:  $M$ ,  $S$  and  $N$ , denoting the product and the sum of all the values in the query’s result set and the size of the query, respectively. Consequently, these three numbers are all that we need to collect with respect to each query. Subsequently,  $\vec{y}$  will refer to this triplet  $\langle M, S, N \rangle$ .

Note that Eq. 3 is valid for a given shape parameter. Since our model assumes that a shape parameter is chosen at random from a weighted set where the probability of choosing shape  $\alpha_j$  is  $w_j$ , the likelihood of observing an entire query set  $\vec{y}$  is:

$$f^*(\vec{y}|\Theta) = \sum_{j=1}^c w_j p(\vec{y}|\alpha_j) \tag{4}$$

In Eq. 4, the  $w_j$ s are each non-negative weights satisfying the constraint that  $\sum_{j=1}^c w_j = 1$ . The complete set of model parameters is  $\Theta = \{\theta_1, \dots, \theta_c\}$ , where  $\theta_j = \{w_j, \alpha_j\}$ .

### 4.3 Learning the parameters

$\Theta$  is unknown and must be learned from the historical workload. To learn  $\Theta$ , we follow the basic principle of *Maximum Likelihood Estimation* (MLE), whose goal is to find the parameter set most likely to have produced the observed data.

Given a set of independent, historical queries  $Y = \{\vec{y}_1, \dots, \vec{y}_r\}$ , applying Eq. 4 the likelihood of observing  $Y$  is:

$$L(\Theta|Y) = \prod_{i=1}^r f^*(\vec{y}_i|\Theta)$$

Often, it is preferable to work with  $\log(L(\Theta|Y))$  because the product given above becomes a summation. That is, we wish to find  $\Theta^*$  so that it maximizes:

$$\Lambda = \max_{\Theta} \log(L(\Theta|Y))$$

In order to optimize the objective function  $\Lambda$ , we employ the *Expectation-Maximization (EM)* framework [6] from statistics and machine learning to iteratively maximize the log-likelihood.

#### 4.3.1 The EM algorithm

EM is used to solve MLE problems made difficult by the fact that there are one or more “hidden” variables that cannot be observed in the data. EM is an iterative method, whose basic outline is described in Algorithm 1.

---

#### Algorithm 1 Basic EM algorithm

---

- 1: **while** The model continues to improve **do**
  - 2: Let  $\Theta$  be the current “best guess” as to the optimal configuration of the model
  - 3: Let  $\bar{\Theta}$  be the next “best guess” as to the optimal configuration of the model
  - 4: **E-Step:** Compute  $Q$ , the expected value of  $\Lambda$  with respect to all possible values of the hidden variables. The probability of observing each possible set of hidden values is computed using  $\Theta$ .
  - 5: **M-Step:** Choose  $\bar{\Theta}$  so as to maximize the value for  $Q$ .  $\bar{\Theta}$  then becomes the new “best guess”.
  - 6: **end while**
- 

In our problem, the hidden variables are the identities of each particular shape that was used to produce each training query. Since the details of some derivations below are similar to the example in [6], most of the resembling derivations are omitted. As a result of the **E-Step**, we have:

$$Q(\Theta, \bar{\Theta}) = \sum_{j=1}^c \sum_{i=1}^r \log(\bar{w}_j) \times p(j|\vec{y}_i, \Theta)$$

$$+ \sum_{j=1}^c \sum_{i=1}^r \log(p(\vec{y}_i|\bar{\alpha}_j)) \times p(j|\vec{y}_i, \Theta) \tag{5}$$

In Eq. 5,  $p(j|\vec{y}_i, \Theta)$  is the posterior probability of query  $\vec{y}_i$  coming from the  $j$ th shape pattern’s distribution, and is given by:

$$p(j|\vec{y}_i, \Theta) = \frac{w_j p(\vec{y}_i|\alpha_j)}{\sum_{l=1}^c w_l p(\vec{y}_i|\alpha_l)}$$

In the **M-Step**, we need to obtain the update equations for the weights  $\bar{w}_j$  and the shapes  $\bar{\alpha}_j$ . To update the weights, we use a Lagrange multiplier to maximize  $Q$  with respect to  $\bar{w}_j$ . This gives us the following update equation for  $w_j$ :

$$\bar{w}_j = \frac{1}{r} \sum_{i=1}^r p(j|\vec{y}_i, \Theta)$$

Next we maximize  $Q$  with respect to each  $\bar{\alpha}_j$  by taking derivative of  $Q$  and setting the result to zero. The part of  $Q$  relevant to  $\bar{\alpha}_j$  is:

$$Q_2 = \sum_{j=1}^c \sum_{i=1}^r \log(p(\vec{y}_i | \vec{\alpha}_j)) \times p(j | \vec{y}_i, \Theta)$$

Unfolding the log operation and taking the derivative with respect to  $\vec{\alpha}_j$ , we have:

$$\frac{\partial Q_2}{\partial \vec{\alpha}_j} = \sum_{i=1}^r \{ \log M_i - N_i \psi(\vec{\alpha}_j) - N_i \log S_i + N_i \psi(N_i \vec{\alpha}_j + 1) \} \times p(j | \vec{y}_i, \Theta) \quad (6)$$

In Eq. 6,  $\psi()$  is the Digamma function.<sup>5</sup> To update  $\vec{\alpha}_j$  we set Eq. 6 to zero and solve it by the bisection method [5].

The EM algorithm repeatedly applies these update equations in an iterative fashion until the parameters begin to stabilize (this is typically measured by an iteration-to-iteration fractional change in  $\Theta$  that is less than 1%).

## 5 The characterization phase

The learning phase provides us with a set of weighted shape patterns that describe the historical workload. As a preparation for the inference phase to make use of these shapes, we need to derive the error distribution associated with each shape. In this section, we show how to determine the distribution of the ratio  $\frac{f(k)}{\widehat{f(k)}}$  for a query result set that we know has been generated by a shape parameter  $\alpha$  and a scale parameter  $\frac{1}{\beta}$ . The extension to unknown parameters is considered in the next section.

Since a query is treated as a sample from a parametric distribution, our estimator  $\widehat{f(k)}$  and the final answer  $f(k)$  can be viewed as a result of the following two-stage sampling process:

- Stage One. A query is produced by drawing a sample of size  $N$  from the distribution  $\text{Gamma}(x|\alpha, \beta)$ . The  $k$ th largest value in this sample is  $f(k)$ .
- Stage Two. In order to estimate  $f(k)$ , a subsample of size  $n$  is drawn without replacement from the sample obtained in stage one. The  $(k')$ th largest value in this subsample is our estimator  $\widehat{f(k)}$ .

Given this process, it is clear that  $f(k)$  and  $\widehat{f(k)}$  are correlated. As a result, it is very hard to analytically obtain the distribution of the ratio between them. Therefore, we resort to Monte Carlo methods to obtain their ratio's approximate distribution, expressed in the form of a cumulative distribution function (CDF).

<sup>5</sup> The Digamma function is defined as  $\psi(z) = \frac{\Gamma'(z)}{\Gamma(z)}$ .

---

### Algorithm 2 Naive Monte Carlo Sampling

---

**Input:**  $N, k, n, \alpha, \beta, num$

**Output:** the Monte Carlo sample array  $MC$

```

1: Let  $MC = \emptyset$ 
2: for  $i = 1$  To  $num$  do
3:   Stage 1: draw an i.i.d. sample of size  $N$  from the
     Gamma( $x|\alpha, \beta$ ) distribution, then find  $f(k)$  from it
4:   Stage 2: draw a subsample of size  $n$  from the
     sample obtained in stage 1, then find  $\widehat{f(k)}$  from it
5:    $MC = MC \cup \left\{ \frac{f(k)}{\widehat{f(k)}} \right\}$ 
6: end for
7: Return  $MC$ 

```

---

#### 5.1 Naive Monte Carlo sampling

In order to obtain the distribution of a statistic, the Monte Carlo approach obtains a large number of independent samples of the statistic directly from the underlying generative model. The samples can then be organized into a sorted list so that the approximate CDF for the target distribution can be calculated by simply counting the fraction of samples less than the CDF's input variable.

It is not hard to imagine how a naive Monte Carlo algorithm for obtaining our particular error distribution would work. The algorithm would simply repeat the above two-stage process  $num$  times (where  $num$  is the number of Monte Carlo samples to produce), and return the array of  $num$  ratios produced. The basic Monte Carlo approach is given as Algorithm 2. The input parameters are the database size  $N$ , the rank of the item to be estimated  $k$ , the sample size  $n$ , the two Gamma distribution's parameters  $\alpha$  and  $\beta$ , and the number of Monte Carlo samples to obtain  $num$ , respectively.

Though simple, the naive algorithm is slow. The cost is  $O(N)$  per Monte Carlo iteration, where  $N$  is the database size. Since we are sampling for an extreme value in order to avoid scanning the entire data set (which is itself an  $O(N)$  operation), making use of an  $O(num \times N)$  Monte Carlo algorithm is unacceptable.

#### 5.2 Practical Monte Carlo sampling

Fortunately, we can do much better by simulating the two stages of the sampling process to produce a result that, statistically speaking, is indistinguishable from an actual execution of the naive method.

To reduce the cost of obtaining one Monte Carlo sample, we need to efficiently sample both  $f(k)$  and  $\widehat{f(k)}$ . It turns out to be easily possible to sample the order statistic  $f(k)$  directly from its CDF (the details of how to do this are deferred to Sect. 5.5). Thus, the problem of sampling for the ratio  $\frac{f(k)}{\widehat{f(k)}}$  reduces to the problem of sampling for  $\widehat{f(k)}$  given a value of  $f(k)$ .

To solve this reduced problem, we devise a Monte Carlo method called the *top-k dependent* (TKD) sampling technique that can efficiently produce  $\widehat{f}_{(k)}$  given  $f_{(k)}$ . The TKD method first determines whether or not the subsample includes  $f_{(k)}$  by means of a Bernoulli trial. Depending upon the result, the TKD method then figures out, in a randomized fashion, the composition of the  $k'$  largest items in the subsample; the  $(k')$ th largest is then returned. Algorithm 3 formally describes the TKD method. The input parameters are identical to Algorithm 2, with the addition of the sampled  $f_{(k)}$ .

---

**Algorithm 3** TKD Sampling

---

**Input:**  $N, k, n, \alpha, \beta$ , a single sample from  $f_{(k)}$   
**Output:** a sample of  $\widehat{f}_{(k)}$ , corresponding to the input  $f_{(k)}$

- 1:  $k' = \lceil \frac{n}{N} \times k \rceil$
- 2: Let  $b \sim \text{Bernoulli}(x|\frac{n}{N})$  \*/  $b$  is the result of a Bernoulli trial \*/
- 3: **if**  $b == 1$  \*/  $f_{(k)}$  is included in the subsample \*/ **then**
- 4: Let  $m \sim \text{Hypergeometric}(x|N - 1, k - 1, n - 1)$
- 5: **if**  $m + 1 < k'$  **then**
- 6: Let  $S$  be  $n - m - 1$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated above at  $f_{(k)}$
- 7: Return the  $(k' - m - 1)$ th largest value in  $S$
- 8: **else**
- 9: Let  $S$  be  $m$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated below at  $f_{(k)}$
- 10: Return the  $(k')$ th largest value in  $\{f_{(k)}\} \cup S$
- 11: **end if**
- 12: **else**
- 13: Let  $m \sim \text{Hypergeometric}(x|N - 1, k - 1, n)$
- 14: **if**  $m < k'$  **then**
- 15: Let  $S$  be  $n - m$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated above at  $f_{(k)}$
- 16: Return the  $(k' - m)$ th largest value in  $S$
- 17: **else**
- 18: Let  $S$  be  $m$  samples from a  $\text{Gamma}(x|\alpha, \beta)$  distribution, truncated below at  $f_{(k)}$
- 19: Return the  $(k')$ th largest value in  $S$
- 20: **end if**
- 21: **end if**

---

### 5.3 Explanation of the TKD algorithm

We now briefly explain why and how the TKD algorithm works.

Recall that we treat a database as a size  $N$  sample from a Gamma distribution whose scale parameter is unknown. Suppose the  $k$ th largest value in the database is known, and a size  $n$  subsample is taken from the database, the goal of the TKD algorithm is that without knowing the exact value of each element in the subsample, how can we directly produce the  $(k')$ th largest value in it by sampling an order statistic from a truncated Gamma distribution?

At a high level, the TKD algorithm does the following. We divide the Gamma distribution into three parts: the lower tail that is less than the  $k$ th largest value, the  $k$ th largest value, and

the upper tail that is greater than the  $k$ th largest value. Both the upper and lower tails correspond to a truncated Gamma distribution. In order to directly produce the  $(k')$ th largest value, we first determine the composition of the subsample from the database, i.e. how many elements in the subsample belong to each of the three parts. Next, based on the composition, we can infer which part of the Gamma distribution the  $(k')$ th largest value in the subsample comes from, and which order statistic is to be sampled from the corresponding truncated Gamma distribution to produce the  $(k')$ th largest value.

Determining the subsample’s composition can be simulated by conducting one trial of a special instance of the Multi-variate Hypergeometric distribution, which is a generalization of the Hypergeometric distribution. Imagine a database tuple as a color ball. The entire database becomes a bucket of color balls. There are three types of tuples in the database: the  $k$ th largest tuple that is the goal of the whole sampling process, the  $(k - 1)$  tuples that have values larger than the  $k$ th largest tuple, and then the remaining tuples which all have values smaller than the  $k$ th largest tuple. We use a single shaded ball to represent the  $k$ th largest tuple, black balls to represent the top  $k - 1$  tuples, and white balls to represent the non-top- $k$  tuples. If one were to reach in and take  $n$  balls out of the bucket, we want to know what is the number for each of the three colors appearing in the  $n$  balls. It is equivalent to do one trial of the Multi-variate Hypergeometric( $X|k - 1, 1, N - k, n$ ) distribution.

To simulate a Multi-variate Hypergeometric trial, one can first focus on simulating the result of one color ball. Conditioned on the result, all balls of the tried-color can be excluded from the bucket, and the remaining balls in the bucket becomes a dimension-reduced Multi-variate Hypergeometric trial, where the sample size and population size are also adjusted to reflect the fact that we have unveiled the result for a particular color. This process continues until all of the colors have been processed.

In our case, we first focus on simulating the result of the shaded ball (this is line 2 of the algorithm). Since there are  $N$  tuples in the database, there is an  $n/N$  chance of including the  $k$ th largest tuple in the subsample, and this is determined via a single Bernoulli (yes/no) trial.

Depending on the Bernoulli trial result, together with the facts that there are  $N - 1$  black and white balls, and  $k - 1$  of them is black, we can do a Hypergeometric( $X|N - 1, k - 1, n - 1$ ) trial (line 4 of the algorithm) when the shaded ball is sampled (thus the sample size becomes  $n - 1$ , since we have known the shaded ball appears in the subsample), or a Hypergeometric( $X|N - 1, k - 1, n$ ) trial (line 13 of the algorithm) when the shaded ball is not sampled (thus the sample size remains to be  $n$ ).

Once we know the composition of the subsample, it is an easy matter to infer the color of the  $(k')$ th largest value in the

subsample. In line 5, when the total number of the sampled black balls and the shaded ball is less than  $k'$ , the  $(k')$ th largest value in the subsample must be the  $(k' - m - 1)$ th largest value of the sampled white balls. Since all white balls are less than the  $k$ th largest value of the database, we can sample an order statistic from the upper-tail-truncated Gamma distribution to produce a sample of the  $(k')$ th largest value (line 6 and 7). On the other hand, when the total number of the sampled black balls and the shaded ball is not less than  $k'$ , the  $(k')$ th largest value in the subsample must be the  $(k')$ th largest value of the non-white balls (line 9 and 10).

Two similar situations occur when the shaded ball is not sampled. When the number of black balls included in the subsample is less than  $k'$ , the  $(k')$ th largest value must be the  $(k' - m)$ th largest value of the sampled white balls (line 15 and 16). On the other hand, when the number of black balls in the subsample is greater than  $k'$ , the  $k'$  largest value of the sampled black balls is returned (line 18 and 19).

### 5.4 Example use of the TKD method

Since the TKD method has several different branches depending upon the results of the random samples obtained on lines (2), (4), and (13) of Algorithm 3, it is useful to illustrate the possible scenarios by means of an example.

Using the previous section's notation, suppose that  $N = 10$ ,  $k = 4$ ,  $n = 4$ ,  $k' = 2$  and  $f_{(4)} = 7$ . Figure 3 illustrates the four possible scenarios that TKD may encounter in processing these inputs. In Fig. 3, each three-bucket group represents the result of either a Bernoulli or Hypergeometric trial. Balls represent samples, and the top bucket in each group contains the entire sampled query result set. The bottom-left bucket in each group contains the subsample, and the bottom-right bucket contains the remaining  $N - n$  samples. The black balls are the top- $(k - 1)$  largest values in the query result set, and the shaded ball with a value label represents the  $k$ th largest value over all.

The TKD method begins by first determining whether or not the  $k$ th largest value appears in the subsample; this creates cases (1) and (2) in Fig. 3, and corresponds to line (2) of Algorithm 3. In either case, it becomes necessary to determine the value of the  $(k')$ th largest value in the subsample. In our example, in the case that the 7 appears in the subsample, we need to determine how many of the black (large-valued) balls also appear in the subsample. This is done via a call to  $\text{Hypergeometric}(x|9, 3, 3)$  on line (4) of Algorithm 3, which generates a Hypergeometric random variable with the specified distribution.

In Fig. 3a, this random trial determines that none of the black balls are retrieved by the subsample. Thus, the TKD method concludes that only one of the top- $k$  largest values are included in the subsample, which is less than  $k' = 2$ . Since we know that all the other items (the three white balls)

on this side must be smaller than  $f_{(k)} = 7$ , in order to sample the second largest value from the subsample, the TKD method returns the largest of three samples from the truncated  $\text{Gamma}(x|\alpha, \beta)$  distribution, where  $x \in (0, f_{(k)} = 7)$ . A "truncated" distribution is simply a probability distribution with one of its tails chopped off.

However, this random trial could have determined that enough black balls were contained in the subsample that the  $(k')$ th largest value in the subsample is from the top  $k$  overall (line 8 of Algorithm 3). In Fig. 3b the  $\text{Hypergeometric}(x|9, 3, 3)$  trial determines that one of the black balls is retrieved by the subsample. Thus, the subsample contains two of the top- $k$  values. The TKD method determines the black ball's value by drawing one additional sample from the truncated  $\text{Gamma}(x|\alpha, \beta)$  distribution, where  $x \in (f_{(k)} = 7, \infty)$ . From this set of two top- $k$  items, the second largest value is returned to the caller.

Two analogous situations occur if the Bernoulli trial has determined that the  $k$ th largest overall does *not* appear in the subsample (lines 12–21 of Algorithm 3). In Fig. 3c, few enough black balls are included in the subsample that the  $(k')$ th largest will come from the Gamma distribution truncated *above* at  $f_{(k)} = 7$ ; in Fig. 3d, enough black balls are included in the subsample that the  $(k')$ th largest will come from one of the black balls, which are sampled from a Gamma distribution truncated *below* at  $f_{(k)} = 7$ .

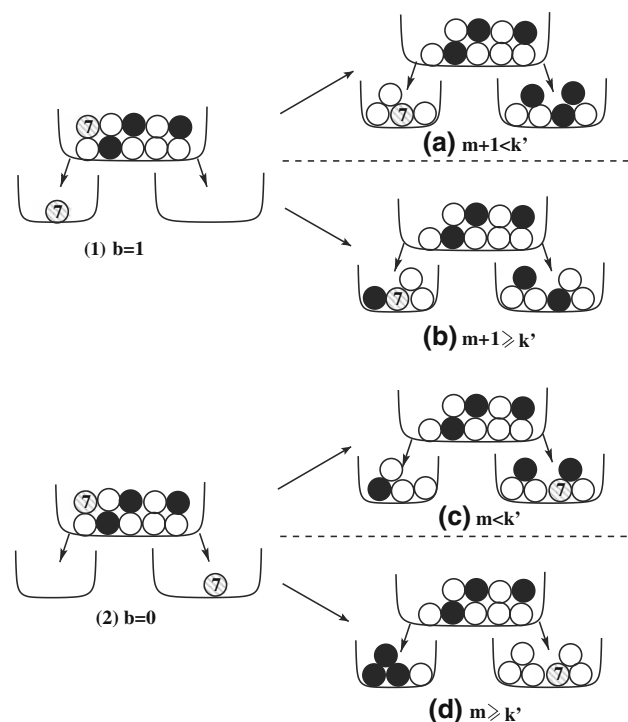


Fig. 3 Illustration of TKD sampling

### 5.5 Additional technical details

In this section, we address a few remaining technical issues regarding the Monte Carlo sampling process.

#### 5.5.1 Sampling $f_{(k)}$

The process described above assumes that we have an efficient method to sample a value for  $f_{(k)}$  without having to generate the entire data set. To sample  $f_{(k)}$  efficiently, we can first obtain its CDF, which is defined by the following lemma:

**Lemma 1** *Given a query size  $N$ , a rank  $k$ , a shape parameter  $\alpha$ , and a scale parameter  $\frac{1}{\beta}$ , the CDF  $F_{f_{(k)}}$  for  $f_{(k)}$  is:*

$$F_{f_{(k)}}(x) = \sum_{i=0}^{k-1} \binom{N}{i} [1 - F_{\text{Gamma}}(x)]^i [F_{\text{Gamma}}(x)]^{N-i} \quad (7)$$

where  $F_{\text{Gamma}}(x)$  denotes the CDF for the  $\text{Gamma}(x|\alpha, \beta)$  distribution.

*Proof* Let  $Y$  be a random variable counting the number of values in the query result set that are greater than or equal to  $x$ . Thus,  $Y$  counts the size of the set  $\{f(d_i) \geq x\}$  for  $i \leq N$ . Since whether or not each  $f(d_i) \geq x$  is an independent Bernoulli trial, we see that  $Y$  follows the  $\text{Binomial}(N, 1 - F_{\text{Gamma}}(x))$  distribution. Then:

$$\begin{aligned} F_{f_{(k)}}(x) &= \Pr[Y < k] \\ &= \sum_{i=0}^{k-1} \binom{N}{i} [1 - F_{\text{Gamma}}(x)]^i [F_{\text{Gamma}}(x)]^{N-i} \end{aligned} \quad \square$$

Given the CDF of  $f_{(k)}$ , it is easy to sample  $f_{(k)}$  using the following two-step process:

1. Sample  $u$  from the  $\text{Uniform}(0, 1)$  distribution
2.  $f_{(k)} = F_{f_{(k)}}^{-1}(u)$

Step two can be implemented by solving for  $x$  in the equation  $u = F_{f_{(k)}}(x)$ . Since a CDF must be monotonically increasing, we can use binary search to obtain the solution. Note that the computation of  $F_{f_{(k)}}(x)$  requires  $O(k)$  time, which is fast since  $k \ll N$ .

#### 5.5.2 Sampling from a truncated Gamma

The TKD procedure needs to be able to sample an order statistic from a truncated Gamma, which ensures line (7), (10), (16) and (19) of TKD algorithm use only  $O(k')$  time. If a CDF for the truncated Gamma can be obtained, sampling an order statistic from the truncated Gamma can be

implemented just as described by Lemma 1. Thus, it suffices to provide the CDFs for the required truncated Gamma distributions:

$$F(x) = \frac{\int_0^x \text{Gamma}(z|\alpha, \beta) dz}{\int_0^{f_{(k)}} \text{Gamma}(y|\alpha, \beta) dy}, \quad 0 < x < f_{(k)} \quad (8)$$

$$F(x) = \frac{\int_{f_{(k)}}^x \text{Gamma}(z|\alpha, \beta) dz}{\int_{f_{(k)}}^{\infty} \text{Gamma}(y|\alpha, \beta) dy}, \quad x > f_{(k)} \quad (9)$$

#### 5.5.3 A note regarding the scale parameter

The reader may notice that we have omitted any mention as to how the inverse scale parameter  $\beta$  is obtained or dealt with by the Monte Carlo process. This may seem like a significant oversight, since  $\beta$  is not supplied. However, the “scale” is simply a multiplicative factor. Thus, since we are interested in the *ratio* of two values sampled from the same Gamma distribution, the scale is irrelevant. As a result, any of the Monte Carlo methods from this section can be implemented by simply choosing an arbitrary scale parameter larger than zero and using it consistently.

---

#### Algorithm 4 Approximating the Error Distribution

---

**Input:**  $p_j$  for  $j \in \{1, \dots, c\}$ ,  $MC_j$  for  $j \in \{1, \dots, c\}$ ,  $num, x$

**Output:**  $F_{\text{ratio}}(x)$

- 1: Sort  $MC = \bigcup_{j=1}^c MC_j$  in ascending order.
  - 2: For each entry in  $MC$ , if  $MC[i]$  came from shape pattern  $j$ , set  $MC[i].prob = p_j/num$ .
  - 3:  $i = 0; tot = 0;$
  - 4: **while** (the current sample  $MC[i]$  is less than  $x$ ) **do**
  - 5:  $tot += MC[i].prob$
  - 6:  $i ++$
  - 7: **end while**
  - 8: Return  $tot$
- 

## 6 The inference phase

At this point, we have most of the tools necessary to complete the framework. Assume that we have completed the learning and characterization phases and have used a set of samples from a database to calculate  $\widehat{f_{(k)}}$ . We wish to characterize the distribution of  $\frac{f_{(k)}}{\widehat{f_{(k)}}$ .

Recall that the prior shape model consists of  $c$  weighted shape patterns. Given the same samples used to compute  $\widehat{f_{(k)}}$ , we can “update” those weights of shape patterns to incorporate the observed data using Bayes’ rule [21]. This is done by computing the posterior probability that we are sampling from the  $j$ th shape pattern. In classic Bayesian fashion, the updated weight  $p_j$  is:

$$p_j = \frac{w_j p(\vec{q} | \alpha_j)}{\sum_{l=1}^c w_l p(\vec{q} | \alpha_l)} \quad (10)$$

Recall that  $\vec{q}$  denotes the aggregate triplet  $\langle M, S, n \rangle$  corresponding to our database sample,  $w_j$  is the prior weight for shape pattern  $j$ , and  $p(\vec{q} | \alpha_j)$  is the PDF of shape pattern  $j$ .

Once the posterior weights are known, the final step in computing the distribution of the ratio  $\frac{f^{(k)}}{\hat{f}^{(k)}}$  is to combine the posterior weights with the Monte Carlo error distribution for each individual shape in order to compute a final, posterior distribution for the ratio. This is formalized in Algorithm 4, which details how to use this information to compute the total probability that  $\frac{f^{(k)}}{\hat{f}^{(k)}} < x$ . The arguments to the algorithm are, in order: the set of posterior weights  $p_1, p_2, \dots, p_c$ , all of the Monte Carlo samples  $MC_1, MC_2, \dots, MC_c$  (one set of samples for each shape pattern), the number of Monte Carlo samples for each shape pattern  $num$ , and finally the CDF input value  $x$ .

In this algorithm, all of the Monte-Carlo samples are first arranged in a sorted order from smallest ratio value to largest. The samples are then weighted according to the posterior weights. To calculate the probability that the ratio  $\frac{f^{(k)}}{\hat{f}^{(k)}}$  is less than an input  $x$ , we scan from the low end to high end of the array  $MC$  and stop once we find the current Monte Carlo sample is greater than the input  $x$ . When we complete the scan, the sum of the probabilities processed will closely approximate the probability that  $\frac{f^{(k)}}{\hat{f}^{(k)}} < x$ . Given the ability to compute this probability, it is then trivial to associate bounds with the ratio  $\frac{f^{(k)}}{\hat{f}^{(k)}}$ .

## 7 Experiments

This section details five sets of experiments. We first perform a set of experiments designed to test the accuracy and applicability of the Bayesian approach to extreme value estimation. Then we test application of the proposed Bayesian framework to four different, specific problems in the data management domain: approximate MAX (or top-k) aggregates, distance-based outlier detection, spatial anomaly detection, and multimedia distance join.

### 7.1 Applicability of the Bayesian approach

**Goals** As discussed previously in the paper (particularly in Sects. 3.2 and 4.1), there are some natural concerns regarding the application of the Bayesian approach to the problem of extreme value estimation. The experiments in this subsection are designed to directly test whether or not these concerns are legitimate. Specifically, we wish to answer two questions:

1. Since the shape model is derived from the Gamma distribution family, does our method actually work for query shapes that look nothing like a Gamma distribution?

2. Second, since the prior shape model is learned from the historical query workload, how will the Bayesian framework function when the future query distribution has changed from the historical query distribution?

**Experimental setup** When evaluating these questions, the relevant metric is confidence bound coverage accuracy. That is, we wish to be able to ensure that no matter what the data look like and what the prior is, if the user specifies  $p\%$  confidence bounds, that the bounds we return are in fact  $p\%$  confidence bounds. To test the method's robustness, we use six strangely shaped, non-Gamma synthetic distributions to generate various query result sets. The generative distributions are illustrated in Fig. 4. They are constructed to be multimodal, exhibit left and right skew with different degrees, and are discontinuous. Clearly, they bear little resemblance to the Gamma distribution.

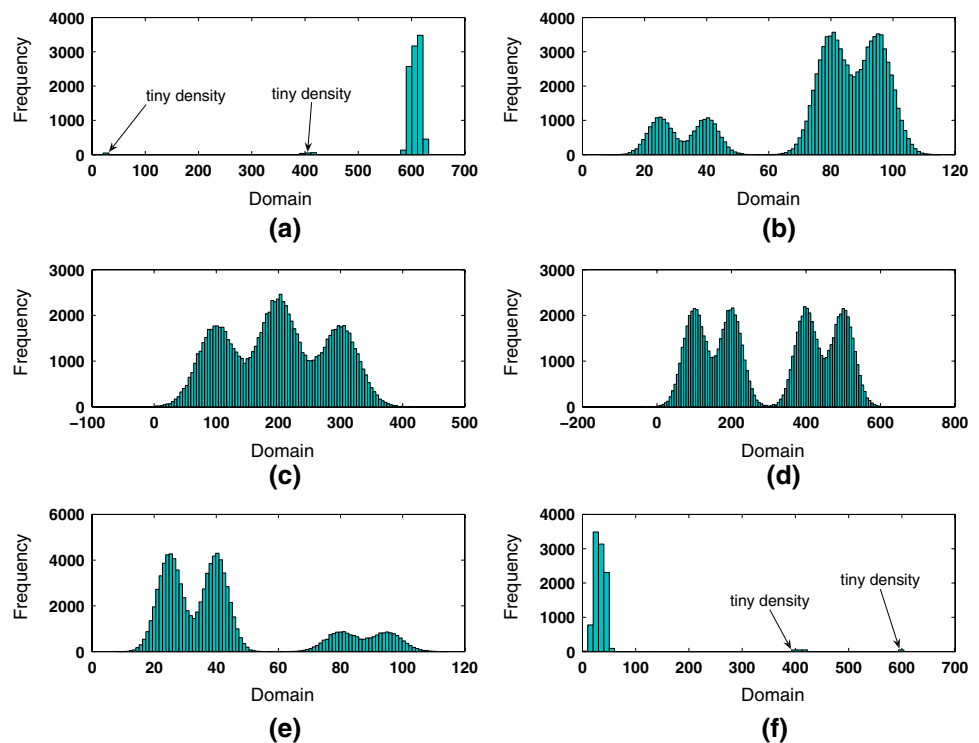
The six synthetic data sources are representative for the following reasons. In general, we loosely regard that any distribution is composed by a main body part and a tail part, both of which can be a multi-modal shape. We choose two modals to indicate the multi-modal situation. The body and tail of a distribution certainly can have one, three or more modals. However, one modal can be seen as a merge of two modals, and three and more modals can be merged to two big modals. So, we believe placing two modals in the main body part and the tail part can capture a broad class of multimodal distributions. Next, we classify any distribution to be either symmetrical or non-symmetrical. For the symmetrical cases, we categorize the distribution to be either modal-split case (case c) or non-modal-split case (case d). For non-symmetrical cases, we further classify the distribution to be either left-skew (case a and b represent left slim tail and fat tails respectively) or right-skew (case e and f represent right slim and fat tails).

Using the discussion in Sect. 3.1, we order the shape patterns using the expected value of  $\frac{f^{(k)}}{\hat{f}^{(k)}}$ . The shape in Fig. 4a has the smallest expected value for this ratio, while the one in Fig. 4f has the largest.

Given these ordered shapes, we run a series of five tests. For each test, we begin by training our model using 500 randomly generated queries each having 1,000 tuples sampled from one of the distributions illustrated in Fig. 4. In the first test, the training queries are sampled uniformly. In the second, they are sampled according to a Geometric distribution with parameter 0.1, so that the first distribution in the ordered shape set is most likely to be sampled and the last one is least likely. In the third, fourth, and fifth, the Geometric parameters are 0.2, 0.4, and 0.8, respectively.

For each learned model, we then run 500 test queries, each of size 50,000. The test queries are always sampled *uniformly* from the generative model. In this way, we test the case

**Fig. 4** The histograms for the six synthetic query distributions considered in the first set of experiments; they are ordered from the easiest case to the hardest case



**Table 1** Coverage rates for 95% confidence bounds using sample size 50, 150, and 300, respectively

| Prior weights   | Sample size = 50 |         |          |          | Sample size = 150 |         |          |          | Sample size = 300 |         |          |          |
|-----------------|------------------|---------|----------|----------|-------------------|---------|----------|----------|-------------------|---------|----------|----------|
|                 | $k = 1$          | $k = 5$ | $k = 10$ | $k = 20$ | $k = 1$           | $k = 5$ | $k = 10$ | $k = 20$ | $k = 1$           | $k = 5$ | $k = 10$ | $k = 20$ |
| Uniform         | 0.84             | 0.86    | 0.81     | 0.85     | 0.91              | 0.92    | 0.91     | 0.93     | 0.95              | 0.95    | 0.95     | 0.96     |
| Geometric (0.1) | 0.84             | 0.86    | 0.82     | 0.85     | 0.90              | 0.92    | 0.91     | 0.92     | 0.95              | 0.95    | 0.94     | 0.96     |
| Geometric (0.2) | 0.78             | 0.80    | 0.77     | 0.80     | 0.87              | 0.88    | 0.88     | 0.90     | 0.93              | 0.93    | 0.93     | 0.95     |
| Geometric (0.4) | 0.86             | 0.89    | 0.85     | 0.88     | 0.92              | 0.93    | 0.93     | 0.94     | 0.95              | 0.96    | 0.96     | 0.97     |
| Geometric (0.8) | 0.88             | 0.89    | 0.86     | 0.89     | 0.93              | 0.94    | 0.94     | 0.95     | 0.96              | 0.96    | 0.96     | 0.97     |

where the query generation is very different from the training distribution. For example, with a Geometric parameter of 0.8, we are very unlikely to see more than a few training queries from the last few distributions in Fig. 4, though we will *test* quite often using those distributions (due to the uniform test generation). For each of the 500 test queries, we obtain 95% confidence bounds for the actual query answer using sample sizes of 50, 150, and 300, and also using 4 different  $k$  values. For each sample size, we compute the fraction of our algorithm produced confidence intervals (coverage rate) that did, in fact, contain the actual query answer. These accuracies are given in Table 1.

**Discussion** In general, the results show the high reliability of the Bayesian framework and clearly illustrate the robustness of the shape model. Using 300 samples, Table 1 shows almost perfect (95%) coverage in every case. There seems to be no dependence on the Gamma distribution (since the test

data were clearly not Gamma-distributed) and very little (if any) dependence on the accuracy of the prior weights, since with sample size 300 Table 1 shows nearly perfect coverage no matter what the training distribution is (recall that the test distribution is always uniform).

One other interesting finding is that there *is* a dependence on sample size, since Tables 1 shows coverage accuracy that is somewhat less than the expected 95% for extremely small samples. This is analogous to problems that occur when we have too few samples to obtain a good variance estimate using a classical estimation regime (see Sect. 3.2). However, we note that once a sample of size 300 has been obtained the coverage is nearly perfect. We stress that 300 is a relatively tiny sample from a real-life database that may have billions of data points.

These findings are not surprising. Robustness to errors in the prior with an adequate sample size is a widely recognized merit of the Bayesian approach. As more and more samples

are taken, the posterior distribution that we use to generate the bounds becomes less dependent on the prior distribution. It is generally acknowledged that in most circumstances, after a few hundred samples the prior carries little (if any) weight, and the sample is used almost exclusively to compute the result. Our results verify this supposition.

#### *When does this break?*

The results above suggest strongly that our framework is quite robust to errors in the prior, as well as to data sets that are not well-modeled by a Gamma distribution. Still, a reasonable question is: This cannot work *all* of the time, right? When will all of this cease to work?

The answer is that our framework is quite tolerant of errors in the prior and non-Gamma data, but it is only a matter of degree. It *is* possible to have such an erroneous prior that our framework breaks. Likewise, it is possible to observe data that are so poorly modeled by a Gamma distribution that our framework does not produce adequate bounds.

For a simple example of a case where our framework would have no ability to correctly bound the error of the ratio estimator, consider Fig. 4. If the historical query workload were generated *entirely* using the model of Fig. 4a, then an actual query generated using the model of Fig. 4f would be handled incorrectly. The reason is that the prior we learn would admit absolutely no possibility of having a long, rightward tail for the distribution of objects in the database, because the training data set did not contain any such shape query—there would not be any shape model that is anything close to Fig. 4f, and so the resulting bounds would be useless. Our methods are tolerant to shifts in the make-up of the query distribution from training to testing (this is what the experiments of this section have shown). But they cannot handle the case where the historical query workload is totally inadequate, either because it is too small to contain examples of the worst sort of query, or because the worst sort of query did not appear until after training had finished or the distribution of queries changed over time to admit new query types. Simply stated: a “bad” query must appear in the set used to train the prior model, or the model will not admit the possibility of experiencing a bad query and the resulting bounds will be useless.

Clearly, one can construct a scenario where the relationship between training and testing data is so distant that our method performs arbitrarily bad. However, this is probably more of an academic worry than a realistic one. A more significant (and usually less several) problem is that it is possible to break our model with query workloads that are so “non-Gamma” that an MLE under a Gamma distribution cannot easily capture what is going on in the right tail of the historical workload. One problem case that we have observed is when there is a single outlier far out in the right tail, and yet

the rest of the data are relatively well-behaved. This seems to be due to the “averaging” effects of the MLE implemented by the EM algorithm. The MLE tries to accurately model the *whole* query result set, and not specifically the outliers. A single outlier would cause the learned shape model to have a more extreme rightward shift than if the outlier were not present, but not extreme enough to cover the outlier adequately. This is especially true if the number of database objects is large enough to pull the shape back towards the main group of points, away from the outlier. As the ratio of outliers to “normal” database objects drops, the problem will become worse. That is not to say that our method is non-robust to skewed distribution. As demonstrated by our multi-modal data: even small sets of outliers in the training queries (such as the bumps in the tail of Fig. 4f) are big enough to be modeled well. But single points far out in the tail will defeat the Gamma model, causing the tail to be pulled back to the main and away from the outlier. This in turn will result in confidence bound coverages that are too low, because they do not admit a high enough possibility of extreme outliers.

## 7.2 Approximate MAX (or top-k) aggregates

The most straightforward application of our Bayesian framework is using it to guess the largest value in a set. For example, one could easily use our Bayesian inference framework to facilitate an online answer to a top- $k$  selection query with an arbitrary selection predicate and aggregate function, along with accuracy guarantees. The records in the data set would be scanned in a randomized fashion, and at all times, the top  $k$  sampled records would be presented to the user. In order to give the user some idea of the quality of the answer set, the samples could be used to obtain confidence bounds on  $f_{(k)}$ . By comparing the  $k$ th best record returned to the user with the bounds for  $f_{(k)}$ , the user may be given an idea of the quality of the answer set that has been obtained thus far.

#### *Application details*

The Bayesian inference framework developed in this paper could easily be used to provide the bounds on  $f_{(k)}$ . First, the previously observed query result sets, each represented by an aggregate triplet, would be used to train the prior shape model. During the training, all data points not accepted by a given training query are ignored when computing the query’s aggregate triplet. (that is, we ignore all  $f() = -\infty$  values). Also, all values not equal to  $-\infty$  are shifted so that their minimum value (the origin) is zero.

When it is time to evaluate a query, the samples from the new query result set are used. Again, we ignore all  $-\infty$  values. Given actual sample and database sizes  $n'$  and  $N'$ , for the purpose of our Bayesian framework we use  $n = |\{f(d) : f(d) \neq -\infty \wedge d \in \text{DB sample}\}|$  and  $N = \frac{N'}{n'} \times n$ . That is,

not only do we ignore  $-\infty$  values, but we “scale down” the size of the database to account for the expected number of  $-\infty$  values that cannot count towards  $f_{(k)}$ . To be consistent with the training, we also shift all of the sampled values so that the smallest value is also at the origin.

Next,  $\widehat{f_{(k)}}$  is computed from the sample. As described in the paper, the samples are also used to compute the posterior shape model, which in turn is used to compute the CDF  $F_{\text{ratio}}$  for the ratio  $\frac{f_{(k)}}{\widehat{f_{(k)}}$ . Given a confidence level  $p$ , bounds *low* and *high* are then chosen so as to minimize *high*–*low* subject to the constraint that  $F_{\text{ratio}}(\text{high}) - F_{\text{ratio}}(\text{low}) = p$ . Finally,  $\text{low} \times \widehat{f_{(k)}}$  and  $\text{high} \times \widehat{f_{(k)}}$  are given as level- $p$  confidence bounds for the answer.

*Experimental evaluation*

**Goals** When evaluating the utility of applying our Bayesian framework to this problem, there are two important questions to answer:

1. First, are the confidence bounds produced reliable for real-life data distributions, arbitrary selection predicates and aggregate functions, and various values of  $k$ ?
2. Second, how narrow are the bounds when a reasonably large sample has been obtained? That is, are they narrow enough to actually be useful?

**Experimental setup** To test these questions, we selected seven real data sets, summarized in Table 2.<sup>6</sup> They are publicly available and span a range of problem domains with different characteristics. For each data set, a “query” is generated as follows. First, a tuple  $t$  is randomly selected. Then a selectivity  $s$  is randomly chosen from the range 5 to 20% and the  $s \times$  (data set size) nearest (Euclidean) neighbors of  $t$  are chosen as the actual query result set. The query’s aggregate function is defined as the weighted sum of three arbitrarily chosen continuous attributes per query, where the weights are uniformly chosen from zero to one. The query answer is defined to be the  $k$ th largest value of the aggregate function, as applied to tuples in the query result set.

For each data set, after training on 500 randomly selected queries using 10 shapes, 500 test queries, are generated, and a 10% sample of the data set is used to provide 95% confidence bounds for the final answer to each query (we also experimented with using only 50 training queries; the results were nearly identical and so are omitted for brevity). Table 3 reports the observed coverage rates of the reported confidence bounds for various values of  $k$ .

In the second set of tests, we use  $k$  values of 1 and 10, respectively, and vary the sample size from 5 to 20% of each

**Table 2** Data description

| Data set    | Continuous/feature | Size      |
|-------------|--------------------|-----------|
| Letter      | 16/17              | 20,000    |
| CAHouse     | 7/9                | 20,640    |
| El Nino     | 7/7                | 93,935    |
| Cover type  | 10/55              | 581,012   |
| KDDCup99    | 34/41              | 4,898,430 |
| Person90    | 12/13              | 5,000,000 |
| Household90 | 7/11               | 5,523,522 |

These data sets consist of both categorical and continuous features. Continuous/Feature denotes the number of continuous features over the number of total features

**Table 3** Coverage rates for 95% confidence bounds with various  $k$  values using a 10% sample

| Data set    | $k = 1$ | $k = 5$ | $k = 10$ | $k = 20$ |
|-------------|---------|---------|----------|----------|
| Letter      | 1.00    | 0.98    | 0.97     | 0.94     |
| CAHouse     | 0.97    | 0.99    | 0.97     | 0.96     |
| El Nino     | 1.00    | 1.00    | 0.99     | 0.99     |
| Cover type  | 1.00    | 1.00    | 0.99     | 0.99     |
| KDDCup99    | 0.92    | 0.91    | 0.92     | 0.93     |
| Person90    | 0.92    | 0.94    | 0.90     | 0.91     |
| Household90 | 0.98    | 0.97    | 0.97     | 0.97     |

data set. We then compute the best, the 25 percentile, the median, the 75 percentile, and the worst relative confidence bound width at 95% (the relative confidence bound width is half the width of the bounds divided by the query answer). Figures 5, 6, 7, 8, and 9 give the results.

**Discussion** In general, the confidence bounds provided show high reliability, which would seem to confirm the correctness of our framework and the appropriateness of a Gamma prior distribution for this problem, even for arbitrary, real-life data sets. There is *some* variation in coverage accuracy; for three of the seven data sets, the bounds were too conservative (showing coverage that was significantly higher than 95%), and for the KDDCup99 and Person90 data sets, the coverage accuracy was a bit lower than 95%. However, the confidence bounds overall were remarkably accurate given the difficulty of the problem. We feel that this is very strong evidence that the bounds generated will be safe and accurate given an arbitrary, real-life data set and query distribution.

The results shown in Figs. 5, 6, 7, 8, and 9 also demonstrate that depending on the data set in question, the bound width at 95% accuracy can be quite narrow, even for a 10% sampling fraction. For example, in Fig. 7, for five of the seven data sets, a 10% sample provides for 95% bounds on the maximum value in the data set whose range is  $\pm 15\%$  of the actual query answer. Not surprisingly, this range generally shrinks

<sup>6</sup> The first five data sets are from the UCI Machine Learning Repository. The last two data sets are from <http://usa.ipums.org>.

as  $k$  grows, since a larger value of  $k$  means that we are trying to guess values that lie further from the extreme right tail of the distribution.

### 7.3 Distance-based outlier detection

More generally, our framework is applicable to any problem where the goal is to find a few records in a set that are “close to” or “far away from” all of the other records in the set. In particular, this applies to distance-based outlier detection [4, 17, 28]. Given an arbitrary distance function  $dist$  (which may or may not be a metric distance), the goal is to pick the  $t$  ( $t \ll N$ ) database points whose distance to their  $k$ th nearest neighbor ( $k$ th-NN) is largest. The smallest distance in the result set is the *cutoff* distance.

#### Application details

A very efficient algorithm for this task (due to Bay and Schwabacher [4]) is a nested loop algorithm (Algorithm 5). At all times, Bay and Schwabacher’s (Bay’s) algorithm maintains a result set. For each point in the data set, the algorithm checks to see if it can find more than  $k$  close-by points with respect to the current cutoff distance value. As soon as the algorithm can find enough close points, the candidate outlier is pruned. Bay and Schwabacher have shown that if the points examined during the pruning step are considered in a randomized order, then excellent performance can be achieved.

---

#### Algorithm 5 Bay and Schwabacher’s Nested Loop Algorithm

---

```

1: Initialize cutoff
2: Let Outliers = {}
3: for each point  $p \in D$  do
4:   Let countClose = 0
5:   for each point  $q \in D$ , in a random order do
6:     if  $dist(p, q) < cutoff$  then
7:       countClose ++
8:       if countClose  $\geq k$  then
9:         break the inner for loop and continue to process next  $p$ 
10:      end if
11:    end if
12:  end for
13:  Add  $p$  to Outliers
14:  if  $|Outliers| > t$  then
15:    Remove the point from Outliers having the smallest  $k$ th-NN distance
16:    Set cutoff to be the smallest  $k$ th-NN distance for any point in Outliers
17:  end if
18: end for

```

---

Our Bayesian framework can easily be used to reduce the number of distance computations required by this algorithm. To apply our framework, we view each database point as a query, and the set of distances from the point to each of the

other points in the database is viewed as the point’s query result set. We randomly select a few points from the database as a training set and compute the distances from each training point to all of its neighbors. These queries are used to train a shape model. This model can then be used to speed Bay’s algorithm as follows:

1. First, we can carefully choose the order in which line (3) of Algorithm 5 considers the database points. If we can guess which points are outliers and consider them first, we will be sure that the cutoff value will be very large early on. This will increase the effectiveness of the algorithm’s pruning. To choose such an advantageous ordering, we observe that the set of distances computed for training from each database point  $p$  to each of the training points is actually a sample of all of  $p$ ’s neighbor distances. This sample set can then be reused to compute  $\widehat{f^{(k)}}$  for  $p$ , as well as the distribution of the ratio  $\frac{f^{(k)}}{\widehat{f^{(k)}}}$  for  $p$ .<sup>7</sup> By computing the median value  $x$  for  $\frac{f^{(k)}}{\widehat{f^{(k)}}}$  (that is, the point  $x$  where  $F_{ratio}(x) = .5$ ), we can use  $x \times \widehat{f^{(k)}}$  as a guess for  $p$ ’s  $k$ th-NN distance, and order the data set accordingly.
2. Second, we speed the algorithm by altering the loop of lines (5)–(12) by adding a second, probabilistic pruning condition after line (11). This additional pruning step is performed periodically (for example, after iterations 10, 20, 40, 80, 160, and so on). Given all of the sampled neighbor distances that have been computed in line (6) for the particular point  $p$ , we use the Bayesian framework and the trained model to *guess* the distance from the point to its  $k$ th-NN. If this guess is less than *cutoff*, then we can prune the point and still be reasonably sure that we have not pruned an outlier. In order to be “reasonably sure”, we should choose an upper bound on the  $k$ th-NN distance that holds with high probability. In our implementation, we choose  $x$  so that  $\frac{f^{(k)}}{\widehat{f^{(k)}}}$  is less than  $x$  with 99% probability (that is, we choose  $x$  so that  $F_{ratio}(x) = 0.99$ ), and our guess as to the upper bound of  $k$ th-NN distance is  $x \times \widehat{f^{(k)}}$ .

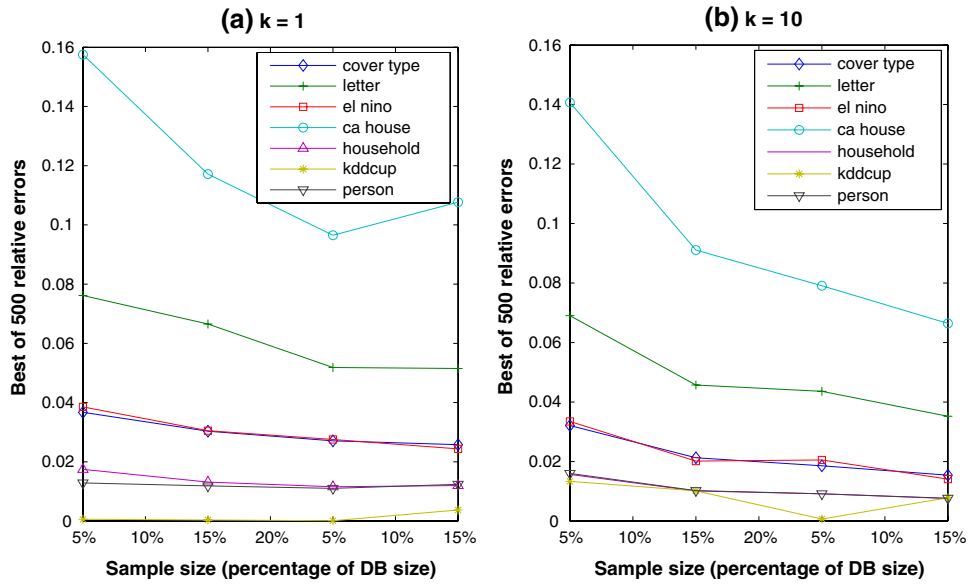
#### Experimental evaluation

**Goals** We wish to experimentally test whether our Bayesian framework can be used to effectively speed Bay’s algorithm. There are two primary questions that we wish to answer:

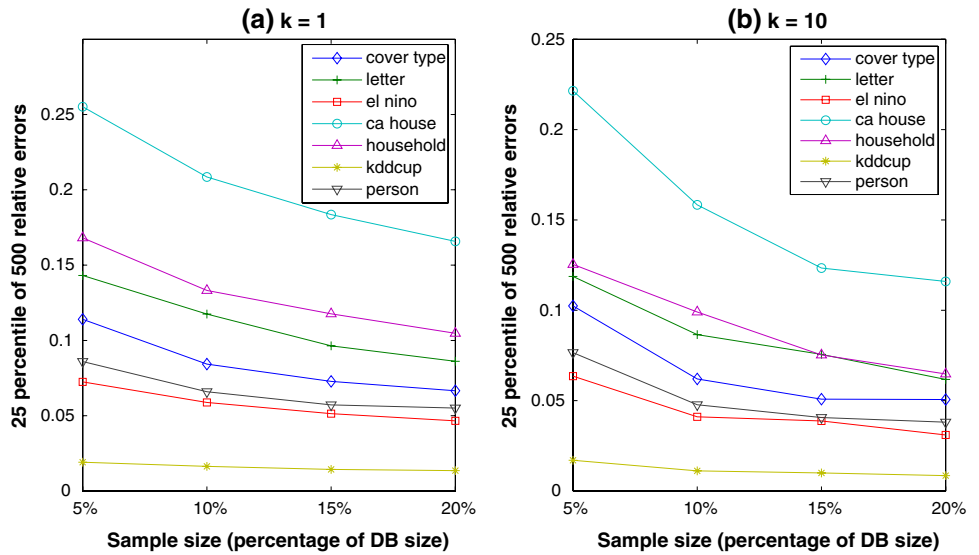
1. First, what sort of speedup compared to Bay’s original algorithm can our framework help to provide?

<sup>7</sup> Note that in this application, because we are looking for nearest neighbors,  $f^{(k)}$  is used to denote the  $k$ th smallest value in the set.

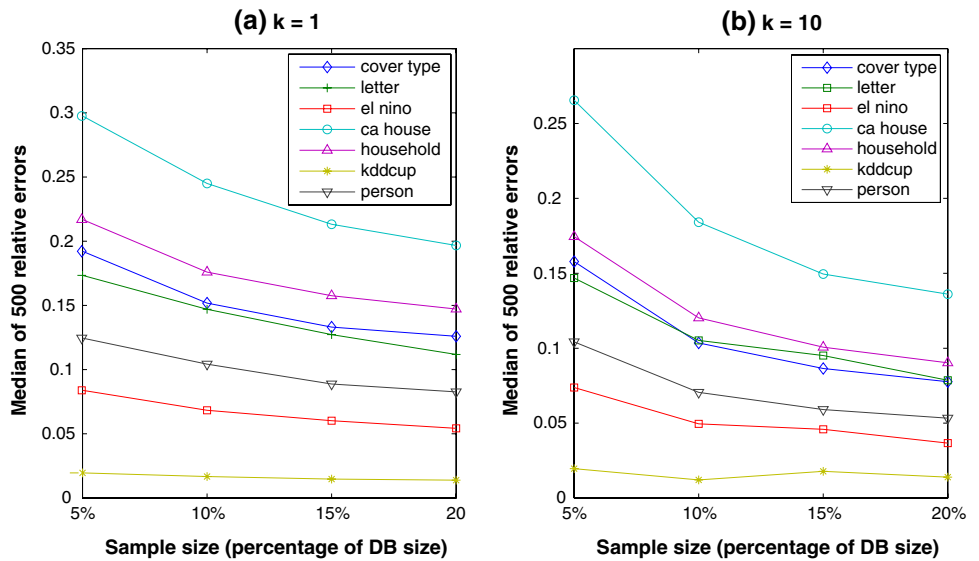
**Fig. 5** Best of 500 test queries' relative errors



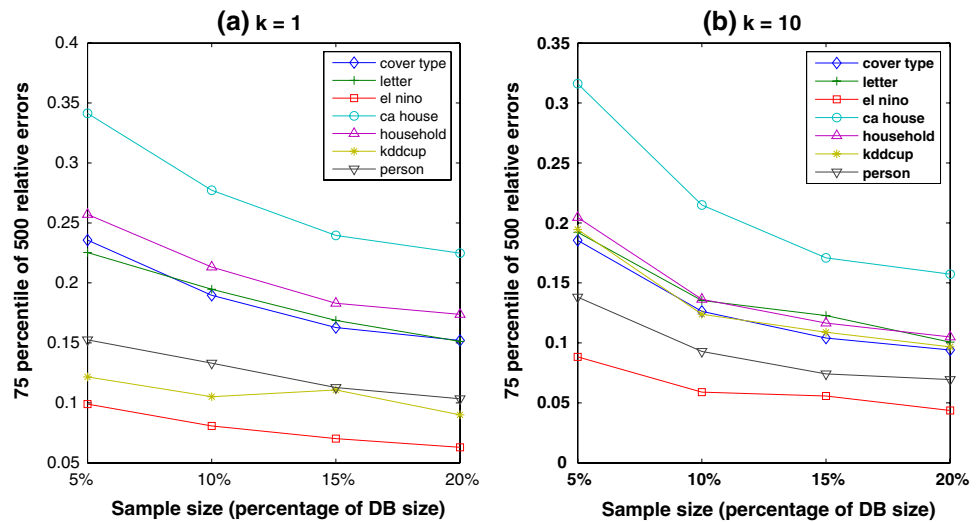
**Fig. 6** The 25 percentile of 500 test queries' relative errors



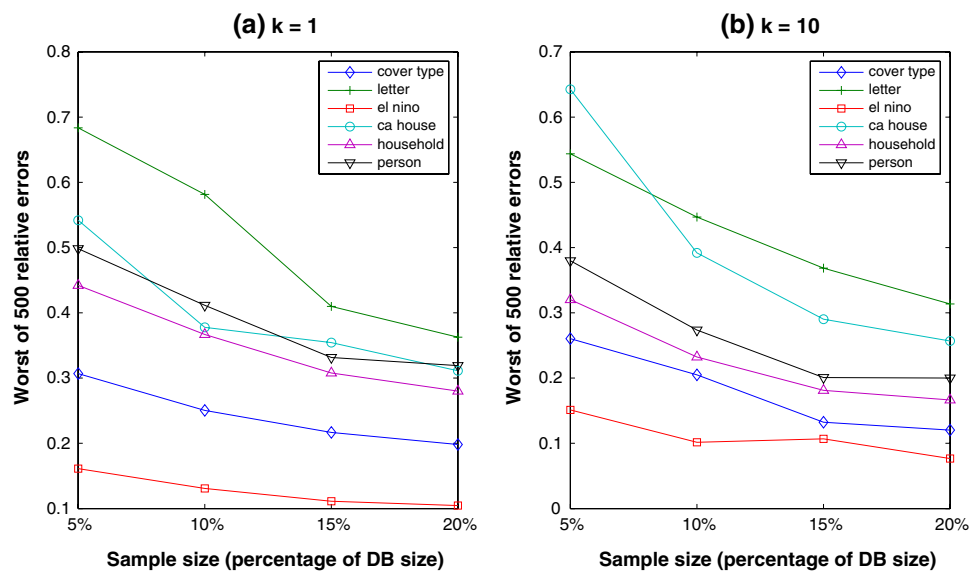
**Fig. 7** Median of 500 test queries' relative errors



**Fig. 8** The 75 percentile of 500 test queries' relative errors



**Fig. 9** Worst of 500 test queries' relative errors. KDDcup data set is not plotted here, since the error is beyond 1



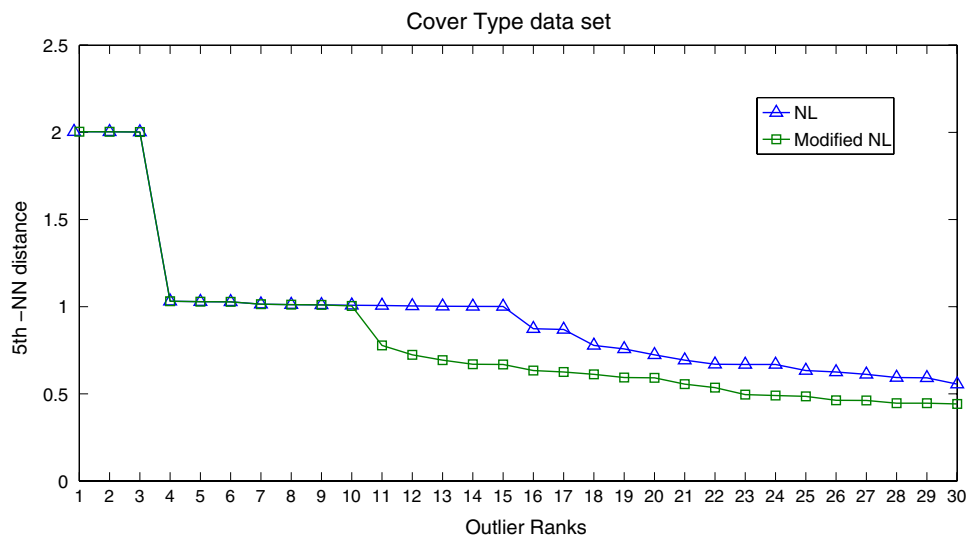
- Second, what exactly is accuracy lost due to the probabilistic pruning that our modified algorithm provides?

**Experimental setup** We began our experiments by running Bay's nested loop algorithm over the seven data sets from the previous subsection to obtain the top 30 5th-NN outliers. We record the total number of distance computations required as well as the actual answer set returned by Bay's algorithm. Next, we run Bay's algorithm augmented with our Bayesian framework as described above (making use of 80 training points and 10 shapes) and compute the speedup of the augmented algorithm with respect to the number of distance computations required. We also compute the overlap of the result set with Bay's result set, as well as the average relative "error" of the augmented algorithm. For example, consider Fig. 10, which shows the 5th-NN distances for the top 30 outliers discovered by both versions of Bay's algorithm for the Cover Type data set (this is the data set where the augmented

version of the algorithm returned the fewest "true" outliers). Let  $b_i$  be the 5th-NN distance for the  $i$ th outlier returned by Bay's algorithm, and let  $ab_i$  be the corresponding value for the augmented version of Bay's algorithm. Then the average relative error is computed as  $\sum_{i=1}^{30} |ab_i - b_i| / \sum_{i=1}^{30} b_i$ . For each data set, the speedup, result set overlap size, and the average relative error are given in Table 4.

**Discussion** These experimental results show that by simply plugging our Bayesian framework into Bay's outlier detection algorithm, one can generally obtain a factor of four improvement in running time. Furthermore, this improvement is obtained virtually for "free" and with almost no loss in result quality. In every experiment, the modified version of Bay's algorithm returned more than 20 out of the 30 outliers that were returned by Bay's original algorithm, but even that statistic tends to under-state the quality of the result. The actual difference in quality between the two result sets

**Fig. 10** Comparison of 5th-NN outlier distances for Bay and Schwabacher’s original nested loops algorithm, and the modified version of Bay and Schwabacher’s algorithm



**Table 4** Result of making use of the Bayesian framework within Bay and Schwabacher’s algorithm

| Data set    | Speedup | Overlap | Error |
|-------------|---------|---------|-------|
| Letter      | 2.61    | 25      | 0.02  |
| CAHouse     | 3.22    | 28      | 0.00  |
| El Nino     | 5.33    | 27      | 0.02  |
| Cover type  | 4.29    | 21      | 0.14  |
| KDDCup99    | 3.92    | 24      | 0.02  |
| Person90    | 5.28    | 24      | 0.07  |
| Household90 | 4.29    | 28      | 0.00  |

For each data set, the table shows the speedup resulting from the application of the framework to the algorithm (Speedup), the size of the result set overlap between the “exact” version of the algorithm and the approximate one (Overlap), and the average relative error of the approximate version (Error)

is always quite small; in five of the seven cases the average relative error is less than 2%. In the worst case (the Cover Type data set), the error is 14%, but close examination of Fig. 10 shows that nearly all of this error is due to the loss of outliers 10 through 15, when it is unclear how much of a problem the loss of those few outliers might actually be. This is due to the fact that outliers one, two and three are clearly dominant—they are much further away from the other database points than any other reported points, and one might reasonably question whether any other reported points are actually outliers.

#### 7.4 Spatial anomaly detection

Spatial anomaly detection has been studied with much intensity recently in statistics [18, 19], machine learning [24], and data mining [1, 2, 25, 26].

Work in data mining has focused on studying the performance issue of spatial scan statistics [18, 19]. These works require a score function  $\Lambda$  to measure the extent to which the data in a region  $R$  differs from the rest of the data in the database. In general,  $\Lambda$  is at least loosely based upon some sort of likelihood ratio test (LRT) [8], which is a statistical hypothesis test that is commonly used to compare differences among data distributions. Most data mining researchers have assumed the score function to be a convex function of an event measurement and a baseline measure over a spatial region. For example, the event measurement can be the count of disease incidents in a given region. The baseline measure can be the population count at risk for that given region. In [25, 26], data is arranged on a spatial grid and an *overlap- $kd$  tree* data structure is designed to partition the spatial area into overlapping regions. The upper bounds of the scores of subregions contained in each overlapping region is computed. Unpromising regions are pruned based on these bounds. The method has  $O((N \log N)^2)$  performance for sufficient dense clusters, where  $N$  is square root of the grid size. Agarwal et al. [1, 2] studied a more general setup where data is not necessarily placed on a spatial grid. They proposed an approximate algorithm that requires  $O(tN^2 \log N)$  time complexity, where  $N$  is the number of data points. The idea is to use a set of linear discrepancy functions to approximate the spatial score function. For each linear function, the optimal region that maximizes the linear function is found. By plugging the set of optimal regions into the original score function, the algorithm returns the one that yields the largest value as an approximate answer.

Our work is different in the sense that we do not require the spatial score function to be convex of the event measurement and the baseline measure. Instead, we only require that the score function map any spatial region to a real value.

We follow the grid setup approach where data are arranged into a rectangular spatial grid, and the goal is to find the particular sub-rectangle  $R$  in the grid (where  $R$  is characterized by a lower left and an upper right coordinates) over which some statistic  $\Lambda$  is maximized. Note that the idea can be similarly applied when we do not place the data on a grid, the only requirements are that we have a method to group the candidates and we can take random samples from each group.

The difficulty of solving this problem exactly is that for an  $n \times n$  grid, there are  $O(n^4)$  different rectangles to consider in order to discover the top  $k$  most anomalous ones—there are  $O(n^2)$  possible lower left points in the grid, and  $O(n^2)$  possible upper right points in the grid, leading to  $O(n^4)$  possible rectangles in all. If  $n$  is very large, or if computing  $\Lambda$  is very expensive, then it can take a long time to search through all of those candidates to find the top  $k$ .

#### 7.4.1 LRT basics

Since spatial anomaly detection is frequently based upon a LRT statistic—indeed, both of the experiments we devise to test the application of our Bayesian framework to spatial anomaly detection are concerned with LRTs—it is informative to begin this particular application with an overview of the LRT and its associated statistic.

The most common statistical test for anomalous data is the so called *likelihood ratio test*. The LRT is a hypothesis test that facilitates the comparison of two models: one parametric statistical model associated with the hypothesis that there is an anomaly, and another parametric statistical model associated with the hypothesis that there is no anomaly—the so-called “null model”. For the LRT to be valid, both parametric models should take the form of identical likelihood functions  $L(\theta|X)$ , where  $X$  is the database data and  $\theta$  is a set of parameters coming from the parameter space  $\Theta$ . The (restricted) parameter space allowed under the null model (this set is denoted as  $\Theta_0$ ) is the complement of the parameter space allowed in the case of anomalous data (the “anomalous” subspace is denoted as  $\Theta - \Theta_0$ ). To check for an anomaly using the LRT, two hypotheses  $H_0 : \theta \in \Theta_0$  and  $H_a : \theta \in \Theta - \Theta_0$  are compared against each other by computing the statistic:

$$\lambda(X) = \frac{\sup_{\Theta_0} L(\theta|X)}{\sup_{\Theta} L(\theta|X)} \quad (11)$$

This statistic is computed by first computing a maximum likelihood estimate (MLE) under both parameter spaces  $\Theta_0$  and  $\Theta$ , and then computing the ratio of the likelihoods obtained via the two MLEs. In 1938, Wilks [34] showed that the asymptotic distribution of  $\Lambda = -2 \log \lambda$  (which we

subsequently refer to as the *LRT statistic*) is chi-squared with  $(p - q)$  degrees of freedom under the null hypothesis that  $\theta \in \Theta_0$ .  $p$  is the number of dimensions (or free parameters) in  $\Theta$ , and  $q$  is the number of dimensions of  $\Theta_0$ . Thus, to check for an anomaly at confidence level  $\alpha$ , one checks whether  $\Lambda(X) \geq c$ , where  $c$  is a non-negative number computed by finding how far out in the tail of a chi-square distribution one has to go to find  $(1 - \alpha)\%$  of the mass.

For a very simple example of the sort of case where the LRT is applicable, imagine that we wish to test whether the disease rate within a spatial area is different than the disease rate outside of the area.  $\Theta$  would contain two probabilities or rates of infection:  $p_{\text{in}}$  and  $p_{\text{out}}$ . On the other hand,  $\Theta_0$  would contain only a single rate  $p$ , where  $p = p_{\text{in}} = p_{\text{out}}$ . The data describing a spatial region would contain four numbers: (1)  $n_{\text{in}}$ —the number of people inside the area; (2)  $n_{\text{out}}$ —the number of people outside of the area; (3)  $k_{\text{in}}$ —the number of diseased people inside the area; and (4)  $k_{\text{out}}$ —the number of diseased people outside of the area. The likelihood function  $L()$  would then be a binomial function:

$$L(\theta|X) \propto p_{\text{in}}^{k_{\text{in}}} (1 - p_{\text{in}})^{n_{\text{in}} - k_{\text{in}}} p_{\text{out}}^{k_{\text{out}}} (1 - p_{\text{out}})^{n_{\text{out}} - k_{\text{out}}}$$

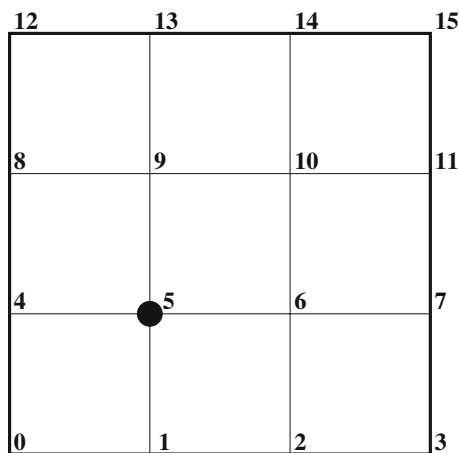
The degrees of freedom of the null distribution is one in this example, since there is one more parameter in  $\Theta$  than in  $\Theta_0$ .

#### 7.4.2 Applying the Bayesian framework

Given some LRT statistic  $\Lambda$ , the naive method to answer a top  $k$  query over a spatial data set placed on a grid would be to apply  $\Lambda$  to each of the  $O(n^4)$  rectangles in the grid and then return the set having the  $k$  largest  $\Lambda$  values. Luckily, it turns out that the Bayesian framework described in this paper can be used to greatly speed this search, as we discuss in this subsection.

The basic methodology for applying our Bayesian framework is quite simple. Given an  $n \times n$  spatial grid, we classify the  $O(n^4)$  rectangles into  $O(n^2)$  classes according to their lower left corners. For example, consider Fig. 11, which illustrates a  $3 \times 3$  grid. Then the rectangles (5,10), (5,11), (5,14) and (5,15) belong to the same class indexed by the lower-left corner 5. If we treat each class as a “query”, where the query result set is the LRT statistics corresponding to all the rectangles in the class indexed by the lower-left corner, then we can apply our Bayesian framework to find the anomaly as follows:

1. First, we randomly pick a set of lower-left points from the grid as our set of training queries. For each of the training queries, we obtain the entire set of LRT statistics, by computing  $\Lambda$  over each of the  $O(n^2)$  possible upper-right points associated with each lower-left point in the training set.



**Fig. 11** Illustration of classifying the rectangles according to their lower-left corner. Rectangles (5, 10), (5, 11), (5, 14) and (5, 15) belong to the same class indexed by the lower-left corner 5

2. Second, we learn a shape model from the queries obtained in step 1.
3. Third, for every possible lower-left point, we randomly sample a small number of upper-right points.  $\Lambda$  is then computed for each of the sampled rectangles associated with each lower-left point. These samples are then used along with our Bayesian framework to obtain a posterior distribution on the best  $\Lambda$  value associated with each lower-left point, one-at-a-time. Then the various lower-left points are sorted based upon the medians of their posterior distributions, from largest to smallest.
4. Finally, the lower-left points are considered in order by a nested-loops algorithm. For each lower-left, we scan all of the valid upper-rights in a random order. For each (lower-left, upper-right) combination, a  $\Lambda$  value is calculated. The sampled  $\Lambda$  values associated with a given lower-left are used by our Bayesian framework to guess whether or not the current lower-left can ever produce a  $\Lambda$  that will exceed the current best  $\Lambda$ . If the probability of this happening is very low, then we move onto the next lower-left point immediately, without considering the remaining associated upper-rights. Thus, in the best case (that is, when the Bayesian framework allows for perfect pruning), it would be possible to reduce the overall complexity from  $O(n^4)$  to  $O(n^2)$ .

Pseudo-code for the actual nested-loops pruning algorithm seeking top-1 rectangle is given as Algorithm 6.

### 7.4.3 Experimental evaluation: synthetic data

**Goals** We wish to experimentally test the scalability and accuracy of the proposed Bayesian solution in this spatial anomaly detection task. There are two primary questions we wish to answer:

#### Algorithm 6 Nested Loops Spatial Anomaly Detection

```

1: Let cutoff =  $-\infty$ 
2: Let bestSoFar = (-1, -1)
3: for each lower-left point ll, in a descending order sorted on their
   estimated medians do
4:   Let counter = 0
5:   Let allLambdas = {}
6:   for each potential upper-right point ur (in random order) do
7:     Compute cur =  $\Lambda(ll, ur)$ 
8:     counter++
9:     allLambdas = allLambdas  $\cup$  {cur}
10:    if cur > cutoff then
11:      Let cutoff = cur
12:      Let bestSoFar = (ll, ur)
13:    end if
14:    if counter is a power of 2 then
15:      Use allLambdas to estimate an upper bound (UB) for the
        current ll's  $\Lambda_{\max}$ , such that  $Pr[\Lambda_{\max} < UB] = 0.99$ 
16:      if  $UB < cutoff$  then
17:        Break/*prune the current ll (query)*/
18:      end if
19:    end if
20:  end for
21: end for
22: Return bestSoFar

```

1. First, does the proposed method scale well with the size of the grid?
2. Second, what is the accuracy lost due to the probability pruning of the candidate rectangles?

**Experimental setup** In this case,  $\Lambda$  encodes a simple binomial likelihood model. This model is chosen because computing the LRT statistic is fast, so we can repeatedly run and re-run tests over reasonably large grids.

Our setup is as follows. For each cell  $c$  in the grid, we randomly generate a population size  $n_c$ . Then, the number of “successes”  $k_c$  in the cell is generated by sampling from a  $\text{Bin}(n_c, p)$  random variable for success rate  $p$ .

Given a test rectangle  $R$  on an  $n \times n$  grid, denote the “success” rate within  $R$  by  $p$  and within  $\bar{R}$  by  $q$ . Then we set up a LRT that makes a decision on the following two competing hypothesis:

- $H_0: p = q$
- $H_a: p \neq q$

From the prior material in this subsection, recall that in general for a LRT,  $\Lambda = -2 \log \lambda = -2 \log \text{MLE}_0 + 2 \log \text{MLE}_c$ , where  $\text{MLE}_0$  is the resulting likelihood of running an MLE over the parameter space associated with  $H_0$ , and  $\text{MLE}_c$  is the resulting likelihood of running an MLE over the parameter space associated with the full parameter space. In this particular case,  $\text{MLE}_0$  requires a single division to estimate  $p$ :

$$p = \frac{\sum_{c \in R \cup \bar{R}} k_c}{\sum_{c \in R \cup \bar{R}} n_c}$$

Then,  $\text{MLE}_0$  itself is computed by plugging the resulting  $p$  back into the original binomial likelihood model. The  $\text{MLE}_c$  routine is just two invocations of  $\text{MLE}_0$ , one over the data within the test region  $R$ , the other over the data outside of  $R$ ; the two  $p$ 's are plugged into two binomial models, and the resulting likelihoods are multiplied. Since the complete parameter space only has one more dimension than the null parameter space, the null distribution is chi-squared with one degree of freedom.

For our tests, we ran 50 trials on a grid of size  $n \times n$  where  $n = 16, 32, 64, 128$ . Our initial cutoff value for pruning was chosen to correspond to an overall false positive rate of 5% (computed using the chi-squared distribution along with a Bonferroni correction [10]).

We ran two sets of experiments. The first is designed to test when the null hypothesis is true, what is the false alarm rate and speedup. For each cell, the population  $n_c$  is always sampled from a Normal ( $\mu = 1 \times 10^4$ ,  $\sigma = 1 \times 10^3$ ) distribution. The “success” rate for each cell is always set to 0.001.

We also simulated the case where the alternative hypothesis holds. The setup was the same as the standard null case, except in the alternative case, we generated a random “hot spot” of size  $3 \times 4$ , within which the “success” rate is 0.003. We consider the hit rate, the containment rate and the average relative error as our accuracy measures. A trial produces a hit if the discovered rectangle is the hot spot. If a trial finally produces a result region that contains the hot spot, we increment the counter of containment cases. In a containment case, the relative error is computed by dividing the areas difference between the returned rectangle and the hot spot by the hot spot area. Results are given in Tables 5 and 6.

**Discussion** In general, the results show excellent scalability compared to the naive nested loops algorithm. In both the null case and the alternative case, the speedup increases faster than linearly with respect to the grid size  $n$ .

The algorithm clearly had the ability to detect hot spots in the data. In every case where there was a hot spot, the algorithm was able to at least partially detect it. Even in few

**Table 5** Null case: total false alarms and average speedup over 50 random trials

| $n \times n$    | $16 \times 16$ | $32 \times 32$ | $64 \times 64$ | $128 \times 128$ |
|-----------------|----------------|----------------|----------------|------------------|
| False alarm no. | 0              | 0              | 0              | 0                |
| Speedup         | 2.8            | 9.5            | 25.3           | 92.1             |

The speedup is computed by dividing the total number of rectangles by the actually examined rectangles in both the training and the pruning stage

**Table 6** Alternative case: average hit rate, containment rate, relative error, and computing rate for binomial data

| $n \times n$     | $16 \times 16$ | $32 \times 32$ | $64 \times 64$ | $128 \times 128$ |
|------------------|----------------|----------------|----------------|------------------|
| Hit rate         | 1              | 1              | 0.88           | 0.91             |
| Containment rate | 1              | 1              | 0.98           | 1                |
| Relative error   | 0              | 0              | 0.026          | 0.03             |
| Speedup          | 1.99           | 6.09           | 19.38          | 50               |

The “hit rate” describes the fraction of experiments for which exactly the true hot spot was returned by the algorithm. The “containment rate” is the fraction of the time that the hot spot was fully contained in the rectangle returned by the algorithm. The relative error is computed by dividing the difference of the areas between the reported rectangle and the generated hot spot by the hot spot’s area. Speedup is computed by dividing the total number of rectangles by the actually examined rectangles in both the training and the pruning stage

cases where the reported region is not exactly the hot spot, the relative error is small considering the hot-spot is only size  $3 \times 4$ .

#### 7.4.4 Experimental evaluation: real data

**Goals** The goal of the experiments detailed in this subsection is to see if the promising experimental results from the prior subsection carry over to a real-life database and real-life application of spatial anomaly detection.

**Experimental setup** The problem we consider is spatial anomaly detection on a real data set extracted from the ARM database [15]. The data set and statistical model for anomaly detection that we use are only briefly described here; for more detail, we refer the reader to our paper which is specifically concerned with anomaly detection over the ARM database [35].

The ARM database describes antibiotic resistance patterns in hospitals as a function of time. It consists of antimicrobial resistance data for nearly 400 hospitals collected over a 15-year period. Over those 15 years, the trend in resistance rates is generally upward, due to selective pressures on the bacteria which causes them to evolve resistance to common antimicrobials. However, the key question that we would like to answer is: Is the trend *uniformly* upward over time, or are there significant spatial irregularities in the trend? If resistance trends show no geographic affinity, then it might indicate that resistance patterns unfold in isolation, and so local programs at individual hospitals aimed at careful antimicrobial stewardship will be useful. If resistance trends have strong spatial correlations, then it might indicate that successful antimicrobial stewardship must be a wider effort, since wide-area trends exhibiting strong influence.

The ARM data are collected as follows. For each year, for each (bacteria/antimicrobial drug) combination, each

participating hospital submits the number of isolates (bacteria instances) tested, as well as the number of times that the bacteria was found to be susceptible to the given drug. In order to apply LRT to our ARM mining problem, we place the ARM data on an  $n \times n$  grid. Our goal is to find the rectangular regions within which subsets of the data set exhibit anomalous antimicrobial resistant trends over time. To do this, we define a LRT statistic as follows:

1. For each cell on the grid, there may be one or more hospitals. Let  $x$  denote the number of susceptible cases and  $n$  denote the number of isolates. For a fixed five year span, let  $i$  index the hospitals and  $j$  index the years, then the data is reported as  $(x_{ij}, n_{ij})$  for every  $i$  and  $j$ .
2. For each hospital, we treat each year's data as a single binomial trial. If we use  $p_{ij}$  to denote the susceptibility rate of hospital  $i$ 's data in year  $j$ , then we have the likelihood function  $L$ :

$$L(\theta|X) = \prod_i \prod_j \binom{n_{ij}}{x_{ij}} p_{ij}^{x_{ij}} (1 - p_{ij})^{n_{ij} - x_{ij}}$$

We assume there is a linear relationship for each hospital's susceptibility rates. That is,  $p_{ij} = kj + c_i$ , where  $c_i$  is an intercept parameter tailored for each hospital  $i$ ,  $k$  is the slope of this linear model, and  $j$  is the year.

We are interested in testing if any subset of hospitals exhibits a different trend than the rest of the hospitals. Therefore, we assume all the hospitals within a test region  $R$  have the same trend  $k_R$ , and all the hospital without region  $R$  have the same trend  $k_{\bar{R}}$ . Then, we have the following competing hypotheses:

- $H_0: k_R = k_{\bar{R}}$
- $H_a: k_R \neq k_{\bar{R}}$

If we replace  $p_{ij}$  with  $kj + c_i$  and compute the likelihood in log space, we have the null log-likelihood:

$$\log L(\theta_0|X) = \sum_i \sum_j \left[ \log \binom{n_{ij}}{x_{ij}} + x_{ij} \log(kj + c_i) + (n_{ij} - x_{ij}) \log(1 - kj - c_i) \right] \quad (12)$$

In Eq. 12,  $k = k_R = k_{\bar{R}}$ .  $\Theta_0 = \{k, c_i's\}$ . Similarly, under the unrestricted parameter space we allow different  $k_R$  and  $k_{\bar{R}}$ .

Then we have:

$$\log L(\theta|X) = \sum_{i \in R} \sum_j \left[ \log \binom{n_{ij}}{x_{ij}} + x_{ij} \log(k_R j + c_i) + (n_{ij} - x_{ij}) \log(1 - k_R j - c_i) \right] + \sum_{i \in \bar{R}} \sum_j \left[ \log \binom{n_{ij}}{x_{ij}} + x_{ij} \log(k_{\bar{R}} j + c_i) + (n_{ij} - x_{ij}) \log(1 - k_{\bar{R}} j - c_i) \right] \quad (13)$$

In Eq. 13,  $k_R$  and  $k_{\bar{R}}$  do not constrain each other.  $\Theta = \{k_R, k_{\bar{R}}, c_i's\}$ .

Just as in the synthetic case, we compute  $\Lambda$  as  $\Lambda = -2 \log \lambda = -2 \log \text{MLE}_0 + 2 \log \text{MLE}_c$ . In this particular case:

3.  $\text{MLE}_0$  implements a numerical optimization routine for maximizing the value of  $L$  in Eq. 12 with respect to the null parameter space. Numerical methods are required due to the lack of an analytic solution.
4.  $\text{MLE}_c$  maximizes the value of  $L$  in Eq. 13 with respect to the complete parameter space by simply invoking  $\text{MLE}_0$  on data within  $R$  and data within  $\bar{R}$  independently.

This is a particularly interesting application of the Bayesian framework, because  $\text{MLE}_0$  requires on the order of a second to compute. This makes the naive, brute-force search on even a small grid prohibitively expensive.

To actually run the experiments, all of the hospitals were organized into an  $16 \times 16$  spatial grid. We first sort the hospitals using the longitude of their physical locations. They are then grouped into  $n$  equi-depth buckets. The placement of hospitals into the  $n$  buckets determines which column of the grid each hospital belongs to. Similarly, we sort all hospitals on latitude, partition them  $n$  ways, and use the partitioning to determine the rows. We run both the naive, nested loops algorithm and an algorithm that makes use of our Bayesian approach.

**Results and discussion** The naive method took 42.69h to complete on this particular data set, and the Bayesian framework was able to reduce this time to 10.83h, for a speedup of 4.0. This is quite remarkable given the very small size of the grid that we tested. Furthermore, there was no loss of accuracy in this particular case: the Bayesian framework discovered exactly the most anomalous region in the database.

### 7.5 Multimedia distance join

The final application that we consider is a multimedia distance join. Given a set of "example" multimedia objects

$P$  and a separate database  $Q$ , the goal is to find the  $t$  closest  $(p, q)$  pairs ( $p \in P$  and  $q \in Q$ ). In a real-life application, the example set may be a number of photographs of a crime suspect, and the database may contain a very large number of photographs extracted from a set of video recordings. By performing the join, we can obtain the  $t$  most likely sightings of the suspect for subsequent expert examination.

#### Application details

If the function  $dist$  is simply the Euclidean distance between two feature vectors, then many algorithms are applicable for this problem [13,33]. However, for more complicated distance functions (especially non-metric functions), the nested-loops algorithm given as Algorithm 7 is really the only alternative.

---

#### Algorithm 7 Nested Loops Distance Join Algorithm

---

```

1: Let  $cutoff = \infty$ 
2: Let  $Matches = \{\}$ 
3: for each point  $p \in P$  do
4:   for each point  $q \in Q$  do
5:     if  $dist(p, q) < cutoff$  then
6:       Add  $(p, q)$  to  $Matches$ 
7:     if  $|Matches| > t$  then
8:       Remove the pair from  $Matches$  having the largest
          $dist(p, q)$  value
9:     Let  $cutoff = \max\{dist(p, q) | (p, q) \in Matches\}$ 
10:    end if
11:  end for
12: end for
13: end for

```

---

We note that even if  $dist$  is a proper metric distance (so that various pruning methods such as a reference-based index could be used to speed the inner loop), in the end a significant fraction of the database records must still be checked for each example point. This is because even advanced methods are restricted in their pruning power. Thus, it is natural to ask: can we make use of our Bayesian framework to determine when an example point  $p \in P$  is unlikely to appear in the result set?

We can accomplish this by selecting a small random subset of the example points. For these points, we scan the database  $Q$  in its entirety, recording the observed  $dist$  values from the database points to our sampled example points. The set of distances associated with each sampled example point is treated as a separate training query within our Bayesian framework, and a shape model is learned. We then make sure to scan the database  $Q$  in a random order, and add a second pruning test after line (11) of Algorithm 7 that attempts to prune the example object  $p$  in a probabilistic fashion (similar to the pruning used in Sect. 7.2). Given the subset of  $Q$  seen so

far, we use our Bayesian framework to make a suitably optimistic guess as to the distance from  $p$  to its closest match in  $Q$ . To ensure that the guess is “suitably optimistic”, we use all of the observed distances to compute the CDF  $F_{ratio}$  as described in Sect. 6, and choose  $x$  so that  $F_{ratio}(x) = 0.01$ . If  $x \times \widehat{f}_{(k)} > cutoff$ , then there is only a very small chance that  $p$  has a match in  $Q$ , and we can prune it.

#### Experimental evaluation

**Experimental setup** The goal in evaluating the resulting algorithm is similar to the goal of the previous subsection. We wish to know: how much of a speedup does this conservative pruning provide for, and what is the effect on result accuracy?

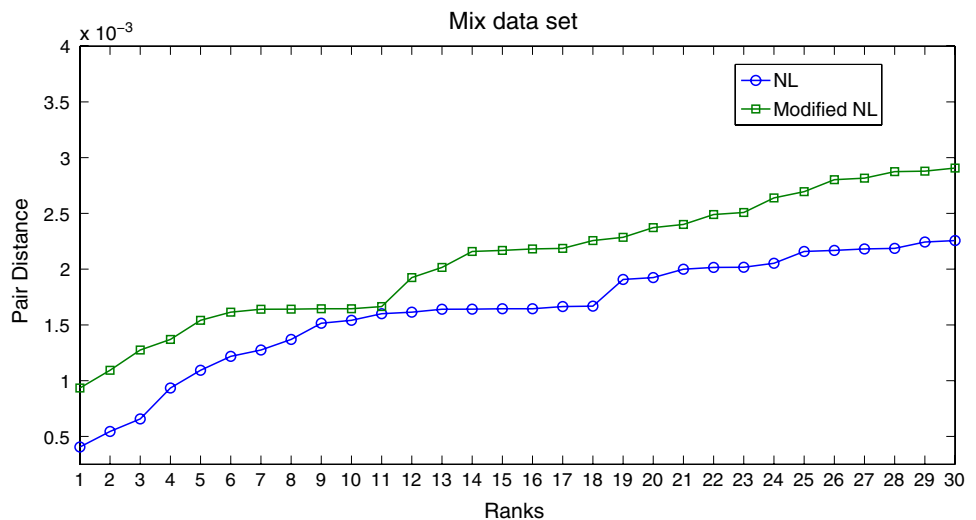
To answer these questions, we use two real image data sets. One image data set consists of 15,117 color images selected randomly from the Internet (we call this the Mix data set). The second image data set is the UCID benchmark color image data set [31], which has 1,338 color images. We transform each image into a 128 dimensional color histogram. In order to create two meaningful distance join queries, we run the k-means clustering algorithm on each data set to segment each data set into two classes. For each data set, our experiment involves computing a top-30 distance join across the two clusters. The distance metric employed is the quadratic distance [32] between the two color histograms. Just as in the last subsection, we compute the speedup, the overlap in result sets, and the relative error, which are reported in Table 7.

**Discussion** In general, we observed a factor of 3–4 improvement on both data sets in terms of the required running time. For the UCID data set, the result quality was no different than for the simple nested loops algorithm. For the Mix data set, 19 out of the 30 “true” matches are returned, with a relative error of 28%. As Fig. 12 shows (this Figure is analogous to Fig. 10), there is some separation between the two result curves, due mainly to the fact that the modified algorithm missed the top three matches. This seems to be due mainly to the extreme heterogeneity of the images from the Mix database, given that they were sampled from the Internet. In other words, this is a case where the Bayesian assumption—that we can build a reasonable prior distribution by looking at a few samples—may be somewhat problematic. Since the images are arbitrary, the next image in  $P$  has little resemblance to any other image in  $P$ . Even so, our

**Table 7** Results for the multimedia distance join experiment

| Data set | Speedup | Overlap | Error |
|----------|---------|---------|-------|
| Mix      | 3.24    | 19      | 0.28  |
| UCID     | 3.52    | 30      | 0.00  |

**Fig. 12** Comparison of the 30 smallest returned pair distances for the basic nested loop join and the modified version that makes use of the Bayesian framework



method still returns 19 of the 30 true answers (out of more than 17 million candidate pairs).

### 7.6 A few final comments

We close by stressing that the section’s goal was not to propose the best possible outlier detection algorithm, or the fastest spatial anomaly detection algorithm, or the fastest possible join algorithm, but was instead to show the type of application for which our Bayesian framework can be useful. In four different cases, it was almost trivial to view the Bayesian framework as a “black box” that can guess the largest or smallest value in a set, and to use this to speed a competitive algorithm by a factor of four or five with little hit in accuracy. Clearly, additional work or more complex application of our methods could result in applications that perform even better, but this is a problem for future work. Our goal was to show the feasibility and utility of this sort of application, and the results of this section seem to be conclusive.

## 8 Related work

Sampling and the use of other statistical methods have long been a popular method in databases and data management [12, 14, 27]. However, to date sampling for extreme values has mostly been ignored as a data management research topic. The reason seems to be the difficulty of the problem. Unlike other aggregate functions such as COUNT, AVERAGE, MEDIAN, and so on, no universal limiting theorems such as the central limit theorem apply to extreme values. For simple aggregate functions such as AVERAGE, the accuracy of an estimate depends primarily on the first two moments of the underlying data distribution (the mean and variance), and these are properties that can easily be estimated. For maximums and minimums, this is not the case.

Not surprisingly, statisticians have studied related problems in detail, and many results are widely known. There are theorems that allow mathematical derivation of the Extreme Value Distribution (EVD) given a known parametric distribution for the original data and the sample size. Here the extreme value refers to the maxima or minima for a given sample. At a high level, all continuous distributions can be classified as one of the following three groups: the Exponential family, the Cauchy family, and the Weibull family. For each distribution family, there exists an associated asymptotic EVD. Knowing the underlying parametric distribution, one can derive the asymptotic EVDs for the *k*th maxima and minima in a similar manner (see Section 7.4 in [16]). In some real life applications, the above EVDs are not applicable. Maritz and Munro [23] presented a Generalized EVD that can be applied to small sample problems as well as large sample problems.

To make use of EVD theory in practice, one typically collects samples of extreme values, and uses the collected sample to estimate the parameters of an asymptotic EVD. For example, imagine that one wants to predict the maximum wind speed of the 25 years’ span since 1997. He/she has collected the maximum wind speed for years from 1997 to 2007 (10 years). Using the 10 extreme value samples, the parameters of an EVD are first estimated, and then the first order statistic of 25 samples is obtained from the resulting EVD as the final estimate. Obviously, the requirement of getting extreme value samples is necessary in this example to make use of EVD theory.

In contrast, our work considers a different scenario where we ask “with a subsample of a finite population, can we bound the extreme value in this *particular* finite population?” Note that here we do not have access to i.i.d. extreme value samples as in the above paragraph. As a result, the traditional EVD theory cannot be applied directly. One could try to fit the sampled data to a parametric distribution that has a

corresponding EVD, and then use the EVD to infer a distribution for the largest value, but since we are interested in arbitrary, real-life data sets that in general do not follow any known parametric distribution, it is doubtful that classic EVD theory (which relies on well-known parametric distributions) will help. Finally, since our approach models the distribution of a ratio estimate, which is scale-agnostic, it can make use of historical queries to predict future queries even though they have quite different scales.

The most relevant work to this topic is the classic survey sampling technique [30], where users design sampling plan to take samples from a finite population and use the obtained sample to infer the properties of the finite population. Our work has been inspired by the superpopulation technique in survey sampling design, where the finite population is treated as a random sample from an infinite superpopulation, and the observed sample is deemed as a subsample of the sample taken from the superpopulation. In order to perform an inference, the subsample is used to estimate the superpopulation first. Once a superpopulation is obtained, it can be used to optimize an estimator that performs well on average over any finite population generated from the superpopulation.

## 9 Conclusion

We have considered the problem of estimating the extreme values in a data set by looking at a small number of samples from it. Because the relationship between the samples and the maximum (or minimum) value in a data set is so dependent upon the distributional properties of the data set in question, we have devised a unique, Bayesian framework for this problem that uses previously observed queries to make a statistically rigorous guess as to the type of query that is currently under consideration. Significantly, we have given four examples of how this framework can be applied to various data management problems. The applications detailed are by no means exhaustive and it is far from clear that we have applied our framework in the best way possible for each problem. However, we believe our results conclusively show that this technique can be applied successfully to many other problem domains.

## References

1. Agarwal, D., McGregor, A., Phillips, J.M., Venkatasubramanian, S., Zhu, Z.: Spatial scan statistics: approximations and performance study, *KDD*, pp. 24–33 (2006)
2. Agarwal, D., Phillips, J.M., Venkatasubramanian, S.: The Hunting of the Bump: On Maximizing Statistical Discrepancy, *SODA*, pp. 1137–1146 (2006)
3. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Scalable Sweeping-Based Spatial Join, *VLDB*, pp. 570–581 (1998)
4. Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule, *KDD*, pp. 29–38 (2003)
5. Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: *Nonlinear Programming: Theory and Algorithms*. Wiley, New York (1993)
6. Bilmes, J.: A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models, University of Berkeley, ICSI-TR-97-021 (1997)
7. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient Processing of Spatial Joins Using R-Trees, *SIGMOD*, pp. 237–246 (1993)
8. Casella, G., Berger, R.L.: *Statistical Inference*, 2nd edn. Duxbury Press, North Scituate (2001)
9. Donjerkovic, D., Ramakrishnan, R.: Probabilistic Optimization of Top N Queries, *VLDB*, pp. 411–422 (1999)
10. Dudoit, S., Shaffer, J.P., Boldrick, J.C.: Multiple hypothesis testing in microarray experiments. *Stat. Sci.* **18**, 71–103 (2003)
11. Haas, P.J., Hellerstein, J.M.: Ripple Joins for Online Aggregation, *SIGMOD*, pp. 287–298 (1999)
12. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online Aggregation, *SIGMOD*, pp. 171–182 (1997)
13. Hjaltason, G.R., Samet, H.: Incremental Distance Join Algorithms for Spatial Databases, *SIGMOD*, pp. 237–248 (1998)
14. Hou, W.-C., Özsoyoglu, G.: Statistical estimators for aggregate relational algebra queries. *ACM Trans. Database Syst.* **16**, 600–654 (1991)
15. <http://www.armprogram.com/>
16. Kinnison, R.R.: *Applied Extreme Value Statistics*. Macmillan, New York (1985)
17. Knorr, E.M., Ng, R.T., Tucakov, V.: Distance-based outliers: algorithms and applications. *VLDB J.* **8**, 237–253 (2000)
18. Kulldorff, M.: A spatial scan statistic. *Comm. Stat. Theory Methods* **26**, 1481–1496 (1997)
19. Kulldorff, M.: Spatial scan statistics: model, calculations, and applications, *Scan Statistics and Applications*, pp. 303–322 (1999)
20. Leadbetter, M.R., Lindgren, G., Rootzen, H.: *Extremes and Related Properties of Random Sequences and Processes: Springer Series in Statistics*. Springer, Berlin (1983)
21. Lee, P.M.: *Bayesian Statistics: An Introduction*. Hodder Arnold, London (1997)
22. Lo, M.-L., Ravishankar, C.V.: Spatial Hash-Joins, *SIGMOD*, pp. 247–258 (1996)
23. Maritz, J.S., Munro, A.H.: On the use of the generalized extreme value distribution in estimating extreme percentiles. *Biometrics* **23**, 79–103 (1976)
24. Neill, D.B., Moore, A.W.: A Fast Multi-Resolution Method for Detection of Significant Spatial Disease Clusters, *NIPS*, pp. 256–265 (2003)
25. Neill, D.B., Moore, A.W.: Rapid detection of significant spatial clusters, *KDD*, pp. 256–265 (2004)
26. Neill, D.B., Moore, A.W., Sabhnani, M., Daniel, K.: Detection of emerging space-time clusters, *KDD*, pp. 218–227 (2005)
27. Olken, F.: *Random Sampling from Databases*, LBL Technical Report, LBL-32883 (1993)
28. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets, *SIGMOD*, pp. 427–438 (2000)
29. Robert, C.P., Casella, G.: *Monte Carlo Statistic Methods*. Springer, Berlin (2004)
30. Sarndal, C.-E., Swensson, B., Wretman, J.: *Model Assisted Survey Sampling*. Springer, Berlin (1992)
31. Schaefer, G., Stich, M.: UCID—An Uncompressed Colour Image Database, *SPIE, Storage and Retrieval Methods and Applications for Multimedia*, pp. 472–480 (2004)

32. Seidl, T., Kriegel, H.-P.: Efficient User-Adaptable Similarity Search in Large Multimedia Databases, VLDB, pp. 506–515 (1997)
33. Shin, H., Moon, B., Lee, S.: Adaptive Multi-Stage Distance Join Processing, SIGMOD, pp. 343–354 (2000)
34. Wilks, S.S.: The large sample distribution of the likelihood ratio for testing composite hypotheses. *Ann. Math. Stat.* **9**, 60–62 (1938)
35. Wu, M., Song, X., Jermaine, C., Ranka, S., Gums, J.: A LRT Framework for Fast Spatial Anomaly Detection, CISE Technical Report, University of Florida (2008)