

Statistical Change Detection for Multi-Dimensional Data

XIUYAO SONG, MINGXI WU, CHRISTOPHER JERMAINE and SANJAY RANKA
University of Florida

This paper deals with detecting change of distribution in multi-dimensional data sets. We use sequential hypothesis testing methods from statistics to define a general, Monte-Carlo framework for solving this problem. We also define a specific statistical test for distributional change within the framework, that we call the *density test*. Our experimental results show that the density test has substantially more power than the two existing methods for multi-dimensional change detection.

Categories and Subject Descriptors: G.3 [Mathematics of Computing]: PROBABILITY AND STATISTICS—*reliability*; H.2.8 [DATABASE MANAGEMENT]: Database Applications—*algorithms*

General Terms: Reliability, Algorithms

Additional Key Words and Phrases: Adaptive sampling, change detection, density test, kernel density estimation, Monte-Carlo simulation

1. INTRODUCTION

This paper considers the problem of building a rigorous statistical test for detecting change of distribution in multi-dimensional data. Such a test would have numerous applications in a variety of disciplines. For example, one may monitor the information about the set of sales observed in the most recent day (in a commercial enterprise) or the prescriptions written recently (in a hospital) or the phone calls or emails recently observed (in the security domain) and ask: is the distribution of recently observed data different from what has been observed before?

In developing a test for distributional change, the primary goal should be to construct a test with the greatest power possible – that is, with the ability to detect the most subtle changes in the data while still maintaining a low false positive rate. Other considerations, such as the ability of the test to scale to very large problem sizes, are secondary. In practice, it is highly unlikely that a test for distributional change would be used to check whether one massive data set collected at one time differs from another massive data set collected at a different time. In “real-life”, few distributions are stationary over time, and so in most domains one would expect that such a large-scale comparison would always result in the declaration that the two data sets are in fact significantly different. Few users would find this sort of result useful or interesting.

A far more useful application would be a comparison where recent data are compared with a number of smaller baseline data sets. For example, in a syndromic surveillance application, one may first partition hospital prescription information based upon the day when the prescriptions were written, and then choose a set of canonical days to serve as baselines. When a new day’s set of prescriptions are

recorded, the set is checked against these baselines to see if it differs from them. If the new day's data differs from all of the baselines, then the hospital epidemiologist can be informed. It makes little sense to compare the new data against *all* of the baseline data at once, because the underlying distribution can be expected to vary naturally over time, so *any* new data would likely differ substantially from one or more of the baselines.

The need for power in such a scenario results from the way that the test is applied. First, the partitioning creates a number of smaller data sets, and with less data, it is more difficult to detect change. Second (and even more important), if a new set is compared against multiple baselines, a multiple hypothesis testing correction such as the application of Bonferroni's inequality would be required [Miller 1966]. This sort of correction requires that the acceptable false positive rate be lowered to account for the fact that multiple tests are run. There is always a direct relationship in any test between the acceptable false positive rate and the power of the test. At a lower false positive rate, a test will have lower power. To be useful in a multiple-test scenario, a change detection method should have naturally high power in order to be able to better withstand the erosion in power that naturally accompanies a low false positive rate.

Problem Definition and Prior Work

The abstract problem we consider is as follows. We assume two sets of i.i.d. samples S and S' , taken from two unknown, multi-dimensional, non-parametric distributions F_S and $F_{S'}$, respectively. In practice, S is a baseline data set, and S' contains the most recently observed data that we wish to test for change. We then define the null hypothesis H_0 , which asserts that F_S and $F_{S'}$ are in fact identical. The goal is to design a proper statistical test that is able to refute H_0 if it is not true. If H_0 is true, then the probability of making an error (where the test says that F_S and $F_{S'}$ are different when in fact they are not) should be at most p , where p is a user-supplied parameter that controls the test's significance level.

For uni-dimensional data, many tests from statistics fit into this framework. The most well-known is the Kolmogorov-Smirnov test; other relevant tests include the Lilliefors test, the Shapiro-Wilk test, the t -Test, and the Anderson-Darling test (for a high-level overview, see Kanji [Kanji 1993]). There has been at least one uni-dimensional test contributed by the data mining/management research community [Kifer et al. 2004].

However, there has been little attention to the problem of extending such tests to multi-dimensional data. Methods such as outlier detection apply to such data [Breunig et al. 2000; Knorr et al. 2000], but they do not detect a mismatch in distribution among data sets; they check for single points that are not in-keeping with the baseline distribution. Some work from data mining is relevant to *describing* changes in multi-dimensional data [Bay and Pazzani 2001; Aggarwal 2003] but it does not address the statistical significance of those changes. Statistical tests for significant spatial irregularities such as Kulldorff's spatial scan statistic [Kulldorff 1997; Neill and Moore 2004] and those for detecting disease outbreaks [Wong et al. 2003] are closely related but are more specific in that they do not check for a generic change of distribution. Rather, such tests work by partitioning the data in some meaningful fashion (temporally or spatially), and then looking for "hot spots" in

the data that may signal some sort of disease outbreak.

In fact, aside from solutions that rely on mapping two or three-dimensional data to a single dimension and applying one of the uni-dimensional tests [Maa et al. 1996], one of the only tests proposed by the statistics community for this problem was published within the last year, called the *cross-match* test [Rosenbaum 2005]. However, as we demonstrate experimentally, the cross-match test may be of questionable power, and it is computationally expensive, requiring a maximal matching computation over a graph having $O((|S| + |S'|)^2)$ edges. To function, all of these edges must be stored in main memory; even then, the running time is cubic in the size of the data set. This renders cross-match generally unsuitable for multi-dimensional data sets larger than a few thousand points. One additional multidimensional change detection test that is closely related to Kulldorff’s spatial scan test was proposed by Dasu et al. [Dasu et al. 2006], but this is a test that relies on a discretization scheme of the data space called the *kdq-tree*. Space partitioning schemes tend to suffer from the curse of dimensionality. As such, one might expect that the power of the test will suffer in more than a few dimensions – an issue that we consider experimentally later in this paper.

The Density Test

This paper’s contribution is the definition and experimental evaluation of an alternative test for multidimensional distributional change, which we call the *density test*. This test is a unique instantiation of a generic detection framework which we subsequently refer to as the *change detection framework* or ChDF for short. The ChDF makes use of an inference method to infer the distribution that produced the sample S . Then, a distance function is used to measure the discrepancy between two sets of samples S and S' . In order to infer the distribution of the distance function under the null hypothesis, a Monte-Carlo simulation is used.

The ChDF itself is not entirely novel – in fact, both Kulldorff’s spatial scan test and Dasu et al.’s test can be seen as being instantiations of the framework. However, previous application of the ChDF has been somewhat haphazard, ignoring key statistical questions. Previously, little attention was paid to the problem of how long to run the Monte-Carlo simulation, and what the effect of the simulation is on the Type I error of the resulting test. In defining the density test, we pay careful attention to these questions, in order to ensure the statistical significance of the results.

Our Contributions

The specific technical contributions of the paper can be broadly broken into two different categories.

- First, we carefully study the Type I error (false positive rate) incurred via the use of the ChDF, and consider the application of sequential statistical test techniques to the problem of determining how long to run a Monte-Carlo simulation and what the error incurred by the simulation is. This removes what has previously been an arbitrary (but fundamentally important) user-defined parameter from the process. Since the cost of a test that makes use of the ChDF is generally linear in the number of Monte-Carlo trials used to run the test, the number of

trials should be kept low in order to ensure efficiency. However, since the error associated with the test increases as the number of trials decreases, the number of trials should be kept high to ensure accuracy. Previous users of the framework have often utilized a magic number of 1000 (or 10,000) trials. We instead replace this magic parameter with a statistically meaningful decision process. Thus, our work can be used to improve the efficiency and correctness of any other method making use of the ChDF, such as Kulldorff’s spatial scan statistic [Kulldorff 1997].

- The key weakness of previous applications of the ChDF to multivariate data has been the use of distance functions that are of questionable utility for multidimensional data. Specifically, previous methods make use of a discretization of the data space. Unfortunately, it is well known that discretizing high-dimensional data is a dangerous approach. Thus, we define and evaluate an alternative test within the ChDF called the *density test*. The density test itself makes use of several novel statistical techniques. For example, in order to obtain a sensitive distance metric even for high-dimensional data, we make use of a unique Expectation Maximization [Dempster et al. 1977] algorithm in conjunction with a kernel density estimator to infer the distribution of the baseline data.

In addition to these key technical contributions, the density test is experimentally compared with two alternative tests via a total of more than 60 different change detection tasks over 13 real data sets, having from 3 to 26 dimensions each. The strengths and weaknesses of the various tests are carefully considered. The density test is found to have substantially more power (especially for detecting changes along multiple dimensions at once) compared to the two existing methods, and has a computational cost that is competitive.

Paper Organization

The remainder of the paper is organized as follows. In Section 2, we address the first technical challenge described above. We first define the ChDF, and carefully consider the statistical issues brought up when Monte-Carlo methods are used to infer the distribution of a distance function under the null hypothesis. Two adaptive sampling procedures are defined in Section 3 that allow for the ChDF to adapt to the specific change detection problem. Section 4 describes the density test itself, and how it is implemented within the ChDF. Section 5 experimentally tests the power and accuracy of the density test compared with the two existing alternatives. Related work is covered in Section 6, and Section 7 concludes the paper.

2. THE CHANGE DETECTION FRAMEWORK

This section first describes the generic ChDF considered in the paper, and then considers the first technical question raised in the Introduction: how much Monte-Carlo computation is enough when performing a test within the ChDF?

2.1 High-Level Overview

As described in the Introduction, the abstract problem considered in this paper is as follows. We assume that we have two data sets S and S' , and our goal is to determine whether or not S and S' were generated by selecting samples from

the same unknown, multi-dimensional distributions. In practice, S is a “baseline” data set, and S' is a set of new observations that we wish to check collectively for deviation from the baseline.

A generic framework that can be used to develop such tests is as follows. First, an inference mechanism \mathcal{F} is used on S to learn F_S , the distribution that produced S . Then, let $\delta = d(S, S')$. δ is the distance between S' and S using some distance function (or statistic) d . In general, d may encode any arbitrary computation over S and S' .

If the null hypothesis that S and S' were generated using the same distribution holds, then the distribution of δ evaluated over S' can be predicted using the distribution $\mathcal{F}(S)$. By sampling many imaginary data sets from $\mathcal{F}(S)$ and computing the distance from each of those data sets to S , we can infer the distribution of $d(S, S')$ under the null hypothesis. Let Δ be a random variable whose distribution is exactly the distribution of δ under the null hypothesis. Then in order to refute the null hypothesis at a significance level p , we simply check whether δ is found to have an extreme value (that is, we check whether $Pr[\Delta \geq \delta]$ is less than p). If it is, then we can reject the null hypothesis and declare that there has probably been a distributional change observed in the data. This framework is the one that we refer to in the paper as the generic ChDF.

An excellent example of a test that makes use of the ChDF is Kulldorff’s spatial scan statistic [Kulldorff 1997]. This particular test has seen considerable attention in the data mining community in the past few years [Neill and Moore 2004; Kulldorff 1999]. The goal of Kulldorff’s test is to find abnormally “dense” regions in a spatial data set that may signal distributional change in spatial data. The inference mechanism \mathcal{F} used by Kulldorff’s method is fairly simple: first, the data space is discretized into a set of cells, and then the baseline distribution parameters are learned by counting the number of baseline data points falling into each cell. The distance function d used by Kulldorff’s method is the maximum value of a likelihood ratio statistic, tested over all possible spatial regions defined by the discretization.

2.2 The Basic Monte-Carlo Approach

In general, once F_S has been learned, it is not possible to analytically derive the distribution of the random variable Δ in order to check whether or not $Pr[\Delta \geq \delta]$ is less than p . In order to address this problem, Monte-Carlo methods must be used. Specifically, a large number of data sets are sampled from the inferred baseline distribution $\mathcal{F}(S)$, and these are used to infer the distribution of Δ under the null hypothesis.

The following algorithm outlines the Monte-Carlo approach. The approach has six parameters: p , d , K , S , \mathcal{F} , and δ . In order, these are: the acceptable false positive rate, the distance function between two samples, the number of Monte-Carlo samples to take, the baseline data set, the inference method, and the observed distance value from S to S' .

The algorithm is fairly simple. K samples from $\mathcal{F}(S)$ are taken, and the distance from S to each sample is computed. If a small number of those distances exceed the distance from S to S' (specifically, if the number is less than pK) then the algorithm declares that a change has been observed.

Algorithm 1 NaiveSample($p, d, K, S, \mathcal{F}, \delta$)

```

1:  $cnt = 0$ 
2: for  $i = 1$  To  $K$  do
3:   Sample  $R$  from  $\mathcal{F}(S)$ 
4:   if  $d(S, R) \geq \delta$  then
5:      $cnt++$ 
6:   end if
7: end for
8: if  $pK > cnt$  then
9:   return change
10: else
11:   return noChange
12: end if

```

Magic Numbers and False Positives. All of this algorithm’s parameters except for p and K are defined by the particular change detection protocol, but p and K must be supplied by the user on a per-test basis. Since p is the test’s desired significance level, choosing an appropriate value for p is not too large a burden to place upon a user, and in fact it can be seen as a feature of the framework, because it allows the user to specify how sure he/she wishes to be in any declared change.

However, choosing K is a different issue altogether. This parameter is a “magic number” in every sense of the term, requiring a careful choice that has a significant effect both on test accuracy and on the computational efficiency of the test, since the number of iterations of the test’s core **for** loop is the major determining factor in the test’s computational cost. Choosing K is a very fundamental problem.

The next subsection as well as Section 3 of the paper consider the effect of the number of iterations of the core loop on the ChDF. In the next subsection, we consider how the effect of a user-supplied K on the accuracy (false positive rate) of a test may be incorporated into the ChDF in a statistically rigorous fashion, without ignoring its effect on test accuracy. The next full section of the paper then considers how the K parameter may be removed entirely from the framework.

2.3 Adding A Binomial Test to the ChDF

Previous instantiations of the ChDF have ignored the effect of K on the accuracy of the test. They regard line 8 of Algorithm 1 as being equivalent to testing $Pr[\Delta \geq \delta] < p$. However, a careful study of line 8 reveals that this line is in fact implicitly running a binomial test for statistical significance that asks: Does the inequality $Pr[\Delta \geq \delta] < p$ hold, given K and cnt ? If this test mistakenly says that this inequality holds, then it will lead to a false positive that cannot be taken into account by Algorithm 1. Thus, Algorithm 1 ignores the effect of the Monte-Carlo process on the quality of the test result. What the algorithm *does* do correctly is to take into account the possibility that even if $Pr[\Delta \geq \delta] < p$, then we still have a chance to experience a false positive if we are unlucky enough to observe a very large δ . This chance for error is guarded by p in Algorithm 1.

To properly measure K ’s effect, we must consider the *true* probability that we incorrectly refute the null hypothesis within the framework, and incorporate this

probability into the computation. In a principled approach, the Type I error should be broken into two separate probabilities, α and β :

- (1) α is the probability that we refute the null hypothesis H_0 because the binomial test in line 8 of Algorithm 1 mistakenly asserts that $Pr[\Delta \geq \delta] < \beta$, which usually caused by insufficient iterations of the sampling algorithm (that is, K was too small), given that the null hypothesis H_0 does in fact hold.
- (2) β is the probability that refute the null hypothesis H_0 while it is true because although the binomial test in line 8 of Algorithm 1 correctly asserts that $Pr[\Delta \geq \delta] < \beta$, but we were unlucky enough to observe a very large distance value δ .

Note that the naive version of the ChDF sampling algorithm (Algorithm 1) takes into account β , but ignores α . The naive algorithm must be slightly modified in order to incorporate α and β :

Algorithm 2 BetterSample($\alpha, \beta, d, K, S, \mathcal{F}, \delta$)

```

1:  $cnt = 0$ 
2: for  $i = 1$  To  $K$  do
3:   Sample  $R$  from  $\mathcal{F}(S)$ 
4:   if  $d(S, R) \geq \delta$  then
5:      $cnt++$ 
6:   end if
7: end for
8: if  $\{\sum_{i=0}^{cnt} Pr[CNT = i | (Pr[\Delta \geq \delta] \geq \beta)]\} < \alpha$  then
9:   return change
10: else
11:   return noChange
12: end if

```

For the most part, this algorithm is no more complex than the previous one. The only changes is that we replace line 8 in Algorithm 1 with a binomial test in Algorithm 2, whose error is guarded by α . In this case, a finite, pre-defined number of sampled distances are computed. The number of distances no less than the test distance δ is represented by a random variable CNT and the value of this variable is stored in cnt ; a larger value of cnt tends to result in a declaration of an unchanged distribution. A smaller value of cnt means that few sampled distances were no less than δ , indicating that δ is unusually large, and so there has been a change. The question that must still be answered, however, is “How large must cnt be in order to declare that we have seen a change?” In other words, how do we compute whether $\{\sum_{i=0}^{cnt} Pr[CNT = i | (Pr[\Delta \geq \delta] \geq \beta)]\} < \alpha$? Equivalently, we ask: Is the probability of observing $\Delta \geq \delta$ greater than or equal to β ? If it is, then there has been no change in the data, since δ is not a relatively large value. Whether or not $Pr[\Delta \geq \delta]$ is larger than β or smaller than β is indicated by cnt :

- (1) If cnt is much smaller than $K\beta$, it is likely that $Pr[\Delta \geq \delta]$ is smaller than β .
- (2) If cnt is much larger than $K\beta$, we would expect that $Pr[\Delta \geq \delta]$ is larger than β .

(3) If cnt is similar to $K\beta$, it might be hard to decide. To be safe, we assume that $Pr[\Delta \geq \delta]$ is larger than β .

In order to describe formally how we can make such a decision based upon cnt , considering a simple analogy may be of some use. Imagine that we throw K balls at a bucket, and we assume the probability of a ball falling in a bucket is at least β . A “ball falling in the bucket” is analogous to a sampled Δ value being at least as large as the observed δ . Assuming that H_0 holds and $F_S = F_{S'}$, then δ should not be too large relative to Δ , and for some not-too-large β , $Pr[\Delta \geq \delta] \geq \beta$ should hold. Mathematically:

$$H_0 : F_S = F_{S'} \Rightarrow \exists \beta, Pr[\Delta \geq \delta] \geq \beta \quad (1)$$

Thus, if we observe that more than $K\beta$ of our sampled Δ values are greater than δ , we consider H_0 to be true. However, if the number of observations of $\Delta \geq \delta$ are very small, then the assumption H_0 is unlikely to be true. Just as the “balls in the bucket” are binomially distributed, given H_0 , the number of observations of $\Delta \geq \delta$ is binomially distributed. Thus, the binomial probability of seeing a value of cnt or less observations ($\Delta \geq \delta$) can be a measurement for the “smallness” of the observed cnt value.

To formalize this, notice that in each Monte-Carlo sampling iteration, we actually obtain one of two results: $\Delta \geq \delta$ or $\Delta < \delta$. However, we actually do not need to calculate the real probability that $Pr[\Delta \geq \delta]$ is larger than β as long as we know that β is its lower bound. This is then a classic application of the one-sided binomial test, with number of experiments equal to K and probability of success in each experiment equal to β . To apply the binomial test, we compare each sampled distance d_i to the observed δ value, and treat these comparisons as a series of Bernoulli trials of the form $X_i, i = 1, 2, \dots, K$. If $d_i \geq \delta$, the corresponding $X_i = 1$ (a success), else $X_i = 0$ (a failure). The hypothesis h_0 that is tested can then be denoted as:

$$h_0 : Pr(X = 1) = \beta$$

By the null hypothesis h_0 , the number of success observations should follow a binomial distribution with parameters (K, β) . cnt can be calculated by $\sum_{i=1}^K X_i$. To apply the test, we need to compute the chance of observing cnt or less one values in $X_1 \dots X_K$ when the probability of a single success is β . This probability is the p -value of the Binomial test and can be computed as:

$$p\text{-value} = \sum_{i=0}^{cnt} \binom{K}{i} \beta^i (1 - \beta)^{K-i} \quad (2)$$

Given a significance level of α , if this p -value is less than α , we can refute the null hypothesis H_0 at the α level in line 8 of Algorithm 2. This method is illustrated in Figure 1. The shaded area of the binomial curve indicates the total probability of seeing cnt (or less) sampled Δ values no less than δ . Smaller values of cnt will imply smaller size of the shaded area. If the mass of the shaded area is less than α , we know cnt is probably too small to support the null hypothesis H_0 , and we can state with confidence $1 - \alpha$ that H_0 is not true.

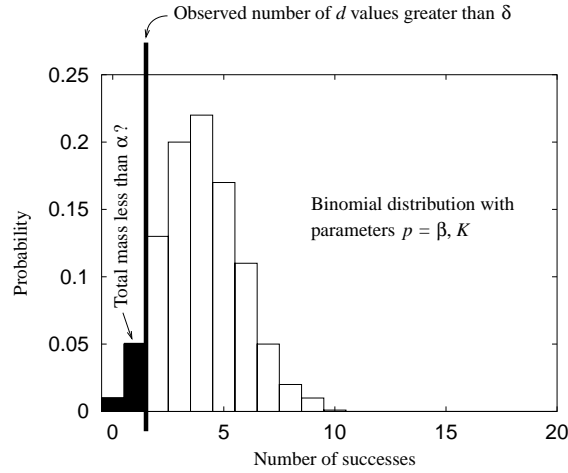


Fig. 1. The binomial testing procedure. Assume that $Pr[\Delta \geq \delta] \geq \beta$ (with $\beta = 0.2$ in this example). Thus, out of 20 samples from Δ , we expect four or five to exceed δ . However, only two exceeded this value. Since the probability of only two samples exceeding δ is less than α , we conclude that $Pr[\Delta \geq \delta]$ probably is less than 0.2 in reality.

Both α and β introduce type I (false positive) error. Since the allowable type I error is given by the user-supplied parameter p , we choose α and β such that $p = 2(\alpha + \beta)$.

3. ADAPTIVE SAMPLING

The framework described in Section 2 uses a Monte-Carlo re-sampling procedure to simulate the distribution of the statistic δ under the null hypothesis H_0 . The problem with Algorithm 2 from Section 2 is that it requires that the user supply a magic number K to govern the number of samples taken. If K is too large, then the algorithm will be very expensive computationally. If K is too small, then the test will lose power, since it will not have enough samples to be sure that the observed δ is in the top β^{th} percentile of the distribution of Δ .

In this section, we refine Algorithm 2 and consider two methods for *adaptively* deciding when to stop the sampling. Rather than forcing the user to choose a K beforehand, we instead keep sampling until a proper statistical decision can be made with respect to the null hypothesis H_0 , or until the sample size reaches a pre-specified threshold *MaxNumSample* (in this case, for safety we declare that there has not been a change in distribution).

The first approach we propose is based upon the Sequential Probability Ratio Test (SPRT) which is from the sequential analysis theory [Wald 1992]. Unfortunately, the classic SPRT is a bit too general to be a perfect fit within the ChDF, because it has the ability to choose between two different hypotheses (when the ChDF only has a single hypothesis to check: Did S and S' come from the same distribution?). As we show experimentally, this extra generality can result in extra Monte-Carlo iterations. Thus, we design a second approach by modifying the standard one-tailed Binomial test described in the Section 2.3. Both approaches can assist the

test framework to stop at an optimal time, and their specific characteristics will be examined in the Experimental Section of the paper.

3.1 Sequential Probability Ratio Test

The characteristic feature of sequential analysis is that the number of observations required by the test procedure is not determined in advance. The decision to terminate the test depends, at each stage, on the results of the observations previously made. The merit of a sequential test is that it usually requires much smaller number of observations than equally reliable test procedures based on a predetermined number of observations, and it does not suffer from the issues of multiple hypothesis testing. In the following we describe the use of the SPRT, one particular test that belongs to the sequential tests family.

At a high level, the SPRT is a generic sequential procedure that repeatedly makes use of a likelihood ratio test to choose from among two alternative hypotheses h_0 and h_a over some model parameter θ . In applying the SPRT to the ChDF, we use the following model. As described previously, we imagine that a Bernoulli random variable X governs whether $\Delta \leq \delta$. This variable has probability θ of taking the value one, and θ is unknown. Recall that our goal is to try to refute (with low probability of error) the hypothesis that θ exceeds β . To do this using the SPRT, we need to set up two competing hypotheses that the test can choose from. This is achieved by specifying two values β_0 and β_1 , where $\beta_1 < \beta < \beta_0$. The SPRT will then be used to choose either $h_0 : Pr[X = 1] = \beta_0$ or $h_a : Pr[X = 1] = \beta_1$. If the test chooses the latter, then a change has been detected. If it chooses the former, then a **noChange** result is returned. Note that if the true value of θ is between β_0 and β_1 , the result of the test is arbitrary. Thus, these values are chosen to be close enough to β so that when $Pr[X = 1]$ is between the two values, the result of the test is not important because the real parameter value is so close to the threshold value β .

In the SPRT, two positive constants A and B are precomputed based on the type I and type II error requirements of the test (type I error is the error that we make when h_0 is true but we reject it; type II error is the error that we make when h_0 is false but we accept it). At each stage of the experiment (at the K^{th} trial for any integral value K), the likelihood ratio statistic λ is computed. In our instantiation of the SPRT for a Bernoulli model, λ is a function of the K Bernoulli observations X_1, \dots, X_K , which are summarized by the variable *cnt*. *cnt* is the number of ones observed among the K trials. Specifically, λ is:

$$\lambda(X_1, \dots, X_K) = \log \frac{\beta_0^{cnt} (1 - \beta_0)^{K - cnt}}{\beta_1^{cnt} (1 - \beta_1)^{K - cnt}}$$

At each iteration of the SPRT, if $B < \lambda(X_1, \dots, X_K) < A$, the experiment continues by taking an additional observation X_{K+1} . If $\lambda(X_1, \dots, X_K) \geq A$, the process is terminated with the acceptance of h_0 . On the other hand, if $\lambda(X_1, \dots, X_K) \leq B$, the process is terminated with the acceptance of h_a . If we denote the type I error by $error_I$ and type II error by $error_{II}$, it can be shown that $A = \log \frac{1 - error_I}{error_{II}}$ and $B = \log \frac{error_I}{1 - error_{II}}$. With some algebraic manipulations, we have that h_0 will be accepted when:

$$cnt \geq \frac{\log \frac{1-error_I}{error_{II}}}{\log \frac{\beta_0}{\beta_1} - \log \frac{1-\beta_0}{1-\beta_1}} + K \frac{\log \frac{1-\beta_1}{1-\beta_0}}{\log \frac{\beta_0}{\beta_1} - \log \frac{1-\beta_0}{1-\beta_1}} \quad (3)$$

and h_a will be accepted when:

$$cnt \leq \frac{\log \frac{error_I}{1-error_{II}}}{\log \frac{\beta_0}{\beta_1} - \log \frac{1-\beta_0}{1-\beta_1}} + K \frac{\log \frac{1-\beta_1}{1-\beta_0}}{\log \frac{\beta_0}{\beta_1} - \log \frac{1-\beta_0}{1-\beta_1}} \quad (4)$$

For each value K , the right-hand side numbers of inequality (3) and (4) are called *acceptance number* and *rejection number* respectively in the SPRT. Obviously, these two numbers are determined by the test parameters and the number of samples we have taken. In our implementation of the SPRT, we set β_0 to be $\beta - 0.005$ and we set β_1 to be $\beta + 0.005$. Also, the type I error of the SPRT is set to α , the level of the one tailed Binomial test. In our implementation, we set the type II error of the SPRT test to the same value as its type I error. Finally, in practice, a number *MaxNumSamples* is specified by the user. When the SPRT has sampled *MaxNumSamples* times and it still cannot make a decision, then the test is stopped (and **noChange** is declared). In our implementation, *MaxNumSamples* was set to 1000. With these parameters, Algorithm 3 presents the pseudo code for using the SPRT in the ChDF.

Algorithm 3 SPRT($\alpha, \beta, d, MaxNumSample, S, \mathcal{F}, \delta$)

```

1: Set  $\beta_0$  and  $\beta_1$  based on  $\beta$ 
2: Specify type II error of SPRT  $error_{II}$ ,
3: Set type I error of SPRT as  $error_I = \alpha$ 
4: cnt=0
5: for i=1 To MaxNumSample do
6:   Do line 3-6 of Algorithm 2
7:   Compute AcceptanceNumber based on inequality (3)
8:   if  $cnt > AcceptanceNumber$  then
9:     return noChange
10:  end if
11:  Compute RejectionNumber based on inequality (4)
12:  if  $cnt < RejectionNumber$  then
13:    return Change
14:  end if
15: end for
16: Return "I do not know"

```

3.2 Step One-tailed Binomial Test

Note that the SPRT is not a perfect fit for our particular problem. The SPRT must be set up in such a way that it chooses among two competing hypotheses. In our particular application, we are really only interested in refuting the single hypothesis that $Pr[X = 1]$ exceeds β . The fact that the SPRT is not a perfect

fit for the ChDF framework may introduce some additional Monte-Carlo iterations above and beyond what is strictly required.

To address this, we develop a second testing procedure called the Step One-tailed Binomial Test (or SOBT for short) that tries to refute the hypothesis $h_0 : Pr[X = 1] \geq \beta$. At the highest level, if the SOBT can refute this hypothesis, then it returns a value of **change**. On the other hand, the test never actually accepts this hypothesis (or any other hypothesis). In a probabilistic fashion, it simply becomes “tired of trying” when it becomes very unlikely that it will ever be able to refute the null hypothesis and eventually stops, in which case (to be safe) it returns **noChange**.

Intuitively, there is a strong reason that such a test may outperform the SPRT for our specific application. Imagine that the SPRT is trying to decide among $Pr[X = 1] = 0.025$ and $Pr[X = 1] = 0.015$ in order to refute $Pr[X = 1] \geq 0.02$, and the first few Monte-Carlo trials all result in one values for X . Since 0.025 and 0.015 are close, it may be impossible to decide from among the two parameter values with high confidence, and so the SPRT must continue. However, it is probably very easy to decide that we are never going to refute $Pr[X = 1] \geq 0.02$ if the first two or three samples are all ones. The SOBT described in this Section will “realize” this very early on and simply stop trying. This will often speed a **noChange** result considerably. A very similar argument describes why the SOBT may also be preferred in the **change** case as well.

At a high level, the SOBT works as follows. Just as in the simple binomial test from Section 2.3, the SOBT tries to refute the null hypothesis $Pr[X = 1] \geq \beta$. The test always maintains two cutoff values lying on the left tail and right tail of the null binomial distribution. If cnt is less than the left cutoff value, we reject h_0 . If cnt is greater than the right cutoff value, we do not actually accept h_0 , but instead decide that it is unlikely that we will ever be able to reject h_0 , and declare **noChange**. Since we are not refuting any null hypothesis in this case, this does not introduce any type I error into the process. The major difference between the SOBT and the simple binomial test is that the test will continue to conduct more trials if cnt falls in between the left cutoff value and the right cutoff value. This process can be repeated until a pre-specified maximum sample size is reached. Algorithm 4 formally presents the test procedure.

Algorithm 4 SOBT ($L, R, batchSize, d, MaxNumSample, S, \mathcal{F}, \delta$)

```

1:  $cnt = 0$ 
2: for  $i=1$  To  $MaxNumSample/batchSize$  do
3:   Repeat line 3-6 of Algorithm 2  $batchSize$  times
4:   if  $cnt < L[i]$  then
5:     Return Change
6:   end if
7:   if  $cnt > R[i]$  then
8:     return noChange
9:   end if
10: end for
11: Return “I do not know”

```

Since our test is one-tailed test, we do not have a significance level for the right tail. However, if we observe large cnt , the chance for us to ever refute h_0 is small; therefore, we can pick a fixed probability γ which we think is “safe” to determine the right cutoff value. The larger the value of γ , the more efficient SOBT algorithm is going to be. However, the less power it will have. In our implementation, we pick $\gamma = 0.1$.

The left tail cutoff value $L[i]$ in each iteration i should be defined in such a way that SOBT is still a level α test, and must take into account the fact that the test is actually a sequence of individual tests. To accomplish this, we use the properties of a geometric series. Let the first iteration have a significance level $\frac{1}{2}\alpha$, the second iteration have a significant level $\frac{1}{4}\alpha$, and the i^{th} iteration have a significant level $\frac{1}{2^i}\alpha$. The sum of all the significance levels over a large number of iterations is upper bounded by α since $\frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^i} + \dots = 1$. Notice that in iteration i , if cnt is greater than $L[i]$, we have $1 - \gamma$ chances to continue to the next iteration. Thus, the significance level in the i^{th} iteration should be set to $\frac{1}{2^i}\alpha \times \frac{1}{(1-\gamma)^{i-1}}$, where $(1 - \gamma)^{i-1}$ is the chances that SOBT will enter the i^{th} iteration. These cutoffs can be determined using Algorithm 5.

Algorithm 5 Calculate L and R (β , $batchSize$, $MaxNumSample$)

```

1:  $L[1 \dots \frac{MaxNumSample}{batchSize}] = R[1 \dots \frac{MaxNumSample}{batchSize}] = \{0, \dots, 0\}$ 
2: Determine  $L[1]$  and  $R[1]$  by  $Bin(\beta, batchSize)$ 
3: for  $i = 2$  To  $\frac{MaxNumSample}{batchSize}$  do
4:   for  $a = L[i - 1]$  To  $R[i - 1] + batchSize$  do
5:     for  $b = L[i - 1]$  To  $R[i - 1]$  do
6:        $Pr[cnt_i = a] += P_{i-1}(cnt_{i-1} = b) \times Bin(y = a - b | \beta, batchSize)$ 
7:     end for
8:   end for
9:   Use the distribution of  $cnt_i$  to determine  $L[i]$  and  $R[i]$ 
10: end for

```

In Algorithm 5, we use arrays L and R to store the left and right cutoff values respectively. In the first iteration of the SOBT, the total number of “successes” (stored in cnt) follows a Binomial($\beta, batchSize$) distribution, just as in the simple binomial test. Therefore, we can easily determine $L[1]$ and $R[1]$ in the second step of Algorithm 5. The probability to the left of $L[1]$ is $\frac{1}{2}\alpha$ and the probability to the right of $R[1]$ divided by the probability to the right of $L[1]$ is γ . Once we determine $L[1]$ and $R[1]$, we can calculate $L[2]$ and $R[2]$. Assume that after we take the second $batchSize$ new Monte-Carlo samples in iteration 2 of the SOBT, we have seen cnt_2 successes. In order to calculate $L[2]$ and $R[2]$, we have to know the distribution of cnt_2 . Since we know the distribution of cnt_1 and we know the distribution of the total number of successes in the second iteration of the SOBT (denoted by y) follows Binomial($\beta, 5$) distribution, we can calculate the distribution of $cnt_2 = cnt_1 + y$ easily by lines 4-8 of Algorithm 5. These lines compute the exact probability associated with every possible value of cnt_2 . Note that we know that if the SOBT reaches the second iteration, cnt_1 must be in the range $L[1]$ to $R[1]$, and so cnt_2 must be in the range $L[1] + batchSize$ to $R[1] + batchSize$. After we

compute the distribution of cnt_2 , we can determine the cutoffs $L[2]$ and $R[2]$. We can then continue the process to determine $L[3]$ and $R[3]$, $L[4]$ and $R[4]$, and so on. This iterative process is presented in lines 3-10 of Algorithm 5, where a is the possible range for cnt_i , b is the possible range of cnt_{i-1} , and P_{i-1} denotes the distribution of c_{i-1} .

4. THE DENSITY TEST

As studied thus far, the ChDF is a generic framework and can be used to realize many specific tests; Kulldorff's scan statistic and Dasu et al.'s test are example of tests that fit within the ChDF. This section describes a new test for distributional change that is also implemented within the ChDF. We call this new test the *density test*. At a high level the density test works as follows:

- (1) First, the baseline S is randomly partitioned into two halves $S = S_1 \cup S_2$. In order to produce samples from $\mathcal{F}(S)$, we simply take a bootstrap sample from S_2 . The reason that we partition before bootstrapping is that our distance computation will build a kernel model using the baseline; by only building the model over S_1 and bootstrapping from S_2 , we make sure that the model is not constructed on the same data that we bootstrap from.
- (2) Next, to compute the distance $d(S_1, S')$, we first compute a kernel density estimator using S_1 ; the resulting density function is denoted by $\mathcal{K}(S_1)$. The distance d is computed as the probability density of $\mathcal{K}(S_1)$ at S' (hence the name "density test"), which is denoted by $\mathcal{K}(S_1)(S')$. Intuitively, if S' does not differ much from S , we expect that the value of $\mathcal{K}(S_1)(S')$ will be large because a sample from $\mathcal{K}(S_1)$ would be likely to produce S' . On the other hand, if S' is very different from S , we expect that $\mathcal{K}(S_1)(S')$ will be small.¹
- (3) Next, just as in algorithms 3, and 4, $\mathcal{F}(S_2)$ is used to generate multiple alternative data sets under the null hypothesis. As mentioned in step (1), this is done via a simple bootstrap re-sample of size $|S'|$ from S_2 .
- (4) For each data set R that is produced via a bootstrap re-sample from S_2 , $\mathcal{K}(S_1)(R)$ is computed. Using either the SPRT or the SOBT procedure from the previous section and depending upon whether or not $\mathcal{K}(S_1)(R)$ exceeds $\mathcal{K}(S_1)(S')$, we either declare **change**, **noChange**, or else we decide that an additional bootstrap sample from S_2 is required. In the latter case, the process is repeated once again from step (3) above until a final verdict is reached.

While the above list describes the basic outline of the density test, a few points regarding some specifics of the test bear further discussion and are covered in the remainder of this section. First, we consider the problem of learning a kernel model for S_1 . Specifically, we discuss why classical fixed-bandwidth methods are likely a poor choice, and propose a data-dependent learning algorithm for computing a more appropriate kernel model. Second, we consider a few issues associated with

¹Since larger values of $\mathcal{K}(S_1)(S')$ indicate a low likelihood of change (in this case we expect a smaller value of distance), it may be more intuitive to use the negative logarithm of $\mathcal{K}(S_1)(S')$ for clarity, which would result in a function whose value increases with increasing dissimilarity. This is necessary, for example, in order to use $d(S_1, S')$ directly in conjunction with algorithms 1, 2, 3, or 4 from the previous sections of the paper.

computing the density of $\mathcal{K}(S_1)$ at R . We also discuss why (for maximum power) the density test should actually be run in two different directions: checking for a change from S to S' and also checking for a change from S' to S . Finally, we give an analysis of the cost of the density test using big- O notation.

4.1 Density Estimation Via Gaussian Kernels

In order to compute the distance from S to S' , the density test makes use of a kernel density estimator (KDE), which is one of the most common non-parametric approaches for learning a data distribution. A KDE approximates the data distribution via superposition of many individual kernel functions at points sampled from the distribution that is being modeled. By allowing each sample to spread its density to nearby areas of the data space, kernel estimators smooth out the contribution of each observed data point over a local neighborhood. The extent of this contribution is dependent upon the shape of the kernel function adopted and the bandwidth accorded to it.

Formally, let $x_1, \dots, x_{|S_1|}$ be the data points in data set S_1 . If S_1 is smoothed using a KDE, then the estimated density at any point s is:

$$\mathcal{K}(S_1)(s) = \sum_{x_i \in S_1} \frac{1}{|S_1|} G(\Sigma_{x_i}, s - x_i) \quad (5)$$

In this equation, G is a “kernel function”. One kernel is centered at each of the data points in S_1 . Σ_{x_i} is the bandwidth or spread of the kernel function centered at x_i , and in the case of a k -dimensional Gaussian kernel (which is used by the density test), Σ_{x_i} is a $k \times k$ positive-definite co-variance matrix.

It is generally acknowledged that in practice, the quality of a kernel estimate depends less on the shape of kernel defined by G than on the bandwidth of each kernel. Unfortunately, in defining the density test we faced a significant technical hurdle with respect to choosing the kernel bandwidth: virtually all applications of the KDE method (and almost all papers in the literature) make the implicit assumption that that all kernels in the model share the same bandwidth; that is, $\Sigma_{x_i} = \Sigma_{x_j}$ for any (x_i, x_j) pair. Since the power of the density test fundamentally relies on the quality of the kernel model and its ability to determine when data are abnormal, such an assumption is very problematic. Different portions of the data space tend to have different characteristics in “real-life” multi-dimensional distributions. Often, there is a clear boundary past which it is clear from the data that no density exists; kernels close to the boundary should have bandwidths that project no density past this boundary. Kernels in an occupied but sparse portion of the data space should have large bandwidths. Kernels in dense regions of the data space should have tighter bandwidths. Visualizations of high dimensional data may show “spokes” radiating out in different directions from a central hub in the data. In such a situation, the bandwidth within each spoke should be elongated along the spoke but not allow the projection of density into the empty areas within the spokes.

Given the dearth of appropriate methods for modeling such real-life distributional characteristics, we choose to use a variation on the standard statistical method of maximum likelihood estimation to choose the kernel bandwidths that were most

likely to have produced the data. Rather than attempting to maximize the likelihood (or log-likelihood) directly as is usually done, we instead attempt to maximize the so-called “pseudo log-likelihood” of the model over all possible choices of each Σ_{x_i} , where this quantity is defined as:

$$L = \sum_{x_j \in S_1} \log \left(\sum_{x_i \in S_1 \wedge i \neq j} \frac{1}{|S_1| - 1} G(\Sigma_{x_i}, x_j - x_i) \right) \quad (6)$$

Note that the pseudo log-likelihood is nothing but the log-likelihood of S_1 over the model, with the constraint that it is known that no data point was generated by the kernel centered at it. Thus, the probability that a data point is generated by itself is zero. We do not maximize the log-likelihood directly because each data point occurs exactly on the center of a kernel. Thus, a model constructed via a direct maximization of the log-likelihood would simply associate a zero-bandwidth kernel with each data point, so that $G(\Sigma_{x_i}, x_i - x_i)$ is always infinity.

In order to perform this maximization, we derive an Expectation-Maximization (EM) algorithm. Since the kernel model with Gaussian kernels can be viewed as an instance of a Gaussian Mixture Model (GMM) with $|S_1|$ components, we can use a simple variation on classical Gaussian EM in order to learn the “optimal” bandwidths. Actually, applying EM to learn the bandwidths is somewhat simpler than applying EM on a general GMM because the centroids of the Gaussian components are fixed at the data points of S_1 , and all Gaussian components have equal weight $\frac{1}{|S_1|}$. Thus, these parameters do not need to be learned in classical GMM.

The remainder of this subsection assumes a basic understanding of the EM framework, and how it is used to learn a GMM. Many excellent tutorials cover this subject (Bilmes’ tutorial [Bilmes 1997] is one of the best known and most complete).

Assume S_1 has dimensionality k . Let $\omega_1, \dots, \omega_{|S_1|}$ be the Gaussian kernels centered at $x_1, \dots, x_{|S_1|}$ respectively. The update rule for the bandwidth of the kernel associated with data point i is:

$$\Sigma_{x_i} = \frac{\sum_{x_j \in S_1} P(\omega_i | x_j) (x_j - x_i)(x_j - x_i)^T}{\sum_{x_j \in S_1} P(\omega_i | x_j)}, \quad (7)$$

In equation 7, $P(\omega_i | x_j)$ is the probability that x_j is generated by kernel associated with x_i . This probability is often referred to as the “soft membership” of point x_j in Gaussian component ω_i . The soft membership is computed with the formula given in Equation (8). The second line of the equation is customized to our particular situation, where we take into account the constraint that the j^{th} data point cannot be generated by the kernel centered at that data point. Note that $\sum_{i=1}^{|S_1|} P(\omega_i | x_j) = 1$ still holds for all j ’s, $j = 1, 2, \dots, |S_1|$.

$$\begin{cases} P(\omega_i | x_j) = \frac{p(x_j | \omega_i)}{\sum_{t=1}^{|S_1|} p(x_j | \omega_t)} & i, j = 1, 2, \dots, |S_1|, i \neq j \\ P(\omega_i | x_i) = 0 & i = 1, 2, \dots, |S_1| \end{cases} \quad (8)$$

In this equation, $p(x_j | \omega_i)$ is the density of the i^{th} Gaussian kernel at the point x_j . and it is calculated by the Equation 9. Again, the second line in the equation

is given because of the special pseudo log-likelihood objective function.

$$\begin{cases} p(x_j|\omega_i) = \frac{1}{(2\pi)^{dim/2}|\Sigma_{x_i}|^{1/2}} \exp(-\frac{1}{2}(x_j - x_i)^T \Sigma_{x_i}^{-1}(x_j - x_i)) & i, j = 1, 2, \dots |S_1|, i \neq j \\ p(x_i|\omega_i) = 0 & i = 1, 2, \dots |S_1| \end{cases} \quad (9)$$

Now we can summarize how we optimize the bandwidth of the Gaussian kernels in Algorithm 6 below.

Algorithm 6 LearnBandwidth(S_1 , $MaxIteration$, ϕ)

```

1:  $t = 0$ 
2: while  $t < MaxIteration$  do
3:   Calculate the density  $p(x_j|\omega_i)$  for  $i, j = 1, 2, \dots |S_1|$  by equation (9)
4:   Calculate the soft membership  $P(\omega_i|x_j)$  for  $i, j = 1, 2, \dots |S_1|$  by equation (8)
5:   Calculate bandwidth  $\Sigma_{x_i}$  for  $i = 1, 2, \dots |S_1|$  by equation (7)
6:   Calculate the value of the objective function  $L_t$  by equation (6)
7:   if  $\frac{L_t - L_{t-1}}{L_{t-1}} < \phi$  then
8:     break
9:   end if
10:   $t++$ 
11: end while

```

According to this algorithm, we stop the computation whenever the iteration number exceeds the maximum iteration number allowed, or when the objective function L converges toward its optimal value. We decide that the objective function has converged if the fractional difference of two consecutive L values is less than ϕ . In our implementation, $\phi = 0.01$ and $MaxIteration = 100$.

Is This Method Effective? Ultimately, the answer to this question is determined by the accuracy of the resulting change detection test, but it is easy to give some anecdotal evidence of how much more effective the KDE resulting from the EM method described above is compared to more traditional kernel bandwidth selection methods.

For example, consider the following simple experiment over the 24-dimensional *Streamflow* data set (see the next section for a description of this data set). We first sample 1000 points from the distribution associated with the data set, and then learn two different kernel models from those 1000 points. The first is learned using Scott's rule [Scott 1992], which is a standard, uniform bandwidth selection method. The second is learned using the EM algorithm described above.

We then sample 1000 additional points from the two KDEs, resulting in three, 1000-point samples in total (one from the original distribution, one from a standard bandwidth estimator, and one from our EM estimator). Projections of these three samples onto two dimensions (the 23rd and 24th dimensions of the *Streamflow* data set) are shown in Figure 2. It is very clear from the three plots that a great deal of information regarding the original distribution has been lost through the application of Scott's estimator. Sampling from the resulting KDE seems to have created a large

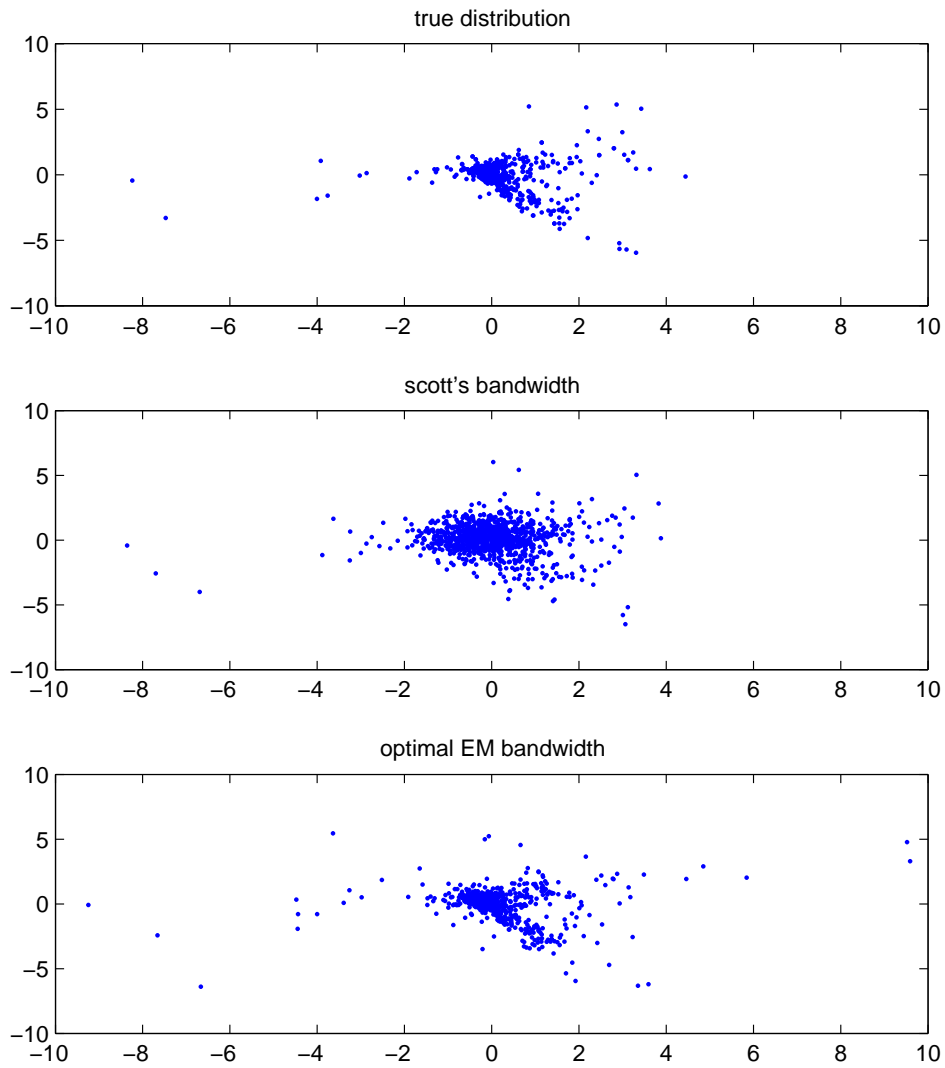


Fig. 2. Samples from the original distribution (top), a KDE using Scott’s bandwidth (center), and a KDE learned using our EM method (bottom).

and shapeless “blob” of points. On the other hand, the EM estimator seems to do an excellent job of preserving the characteristics of the original distribution. The experiments in the next section will show how sensitive the resulting KDE can be when it is used to detect changes in high-dimensional data.

4.2 Distance Via Log-Likelihood

In the density test, we instantiate the distance function d based on the probability density of the $\mathcal{H}(S_1)$ at a set of points. Specifically, the distance from S_1 to a sample set R is simply the negative of log-likelihood (LLH) of R under the inferred

density function $\mathcal{K}(S_1)$. This negative of log-likelihood does not provide a true distance metric in the strict sense. Nevertheless, it is a good candidate “distance” function in the sense that given two sets of data points R_1 and R_2 , if $d(S_1, R_1) < d(S_1, R_2)$, we can make the conclusion that R_1 was produced by sampling from a distribution that is closer to the distribution used to produce S_1 .

Formally, the distance is defined in Equation (10).

$$d(S_1, R) = - \sum_{x_j \in R} \log \sum_{x_i \in S_1} \frac{1}{|S_1|} G(\Sigma_{x_i}, x_j - x_i) \quad (10)$$

In this expression, $G(\Sigma_{x_i}, x_j - x_i)$ is the density of a Gaussian kernel evaluated at $x_j - x_i$. The Gaussian kernel is centered at the origin and having bandwidth (covariance matrix) Σ_{x_i} calculated by Algorithm 6.

There is one issue regarding the distance used by the density test that bears further discussion. Any time that characteristics (parameters) of a distribution are inferred from samples, there is some variability in the inference. This variability can often be taken into account, but the process of “taking it into account” also introduces some variability. At some point, the error has to either be accepted or dealt with directly. For example, consider the classic problem of inferring the mean of a distribution by sampling the distribution. The sampling process will introduce some error, which is typically described by the process’ variance. However, the variance *itself* must be estimated, and this estimate itself has some error.

A similar problem occurs whenever the ChDF is instantiated. In the density test, when a KDE is learned, some error may be incurred because the distribution that produced S was not inferred with total accuracy. This is mostly taken into account via the bootstrap re-sampling process where the distribution of Δ is obtained with the use of $\mathcal{K}(S_1)$ as well, but the bootstrap process itself may incur a small error.

A concern is that this error may inject additional false positives above and beyond those admitted by the user-supplied p value. One characteristic of the LLH-based distance is that this measure is sensitive to outliers, since the density of a Gaussian kernel falls off exponentially from the center of the kernel. This characteristic has two main effects. On one hand, if S' has a slightly different distribution from S , then S' should have some “outlier” points that are far from any kernel in $\mathcal{K}(S_1)$. These “outlier” points should then result in a large test statistic value, and it is high likely that the density test will declare a distributional change in S' . In this situation, being sensitive to outliers facilitates the power of the density test. On the other hand, it is very common that the data sets S and S' are not absolutely “clean”. Both of them may have a few outliers that have not been well-modeled by the inference process, even if their underlying distributions are identical. In such a situation, the test data set S' may have a few data points far from any point in S_1 , in a very low density region of the inferred distribution $\mathcal{K}(S_1)$. In this case, δ would have a large value that could not be taken into account properly via the bootstrap procedure that is used to learn the distribution of Δ , resulting in a false positive that cannot be controlled by the user-specified p .

In order to take this into account, we use a simple “dropping” strategy. As this may imply, we modify the LLH-based distance by dropping the outliers in the test data set. The fraction of outliers dropped is governed by a parameter τ . Specifically,

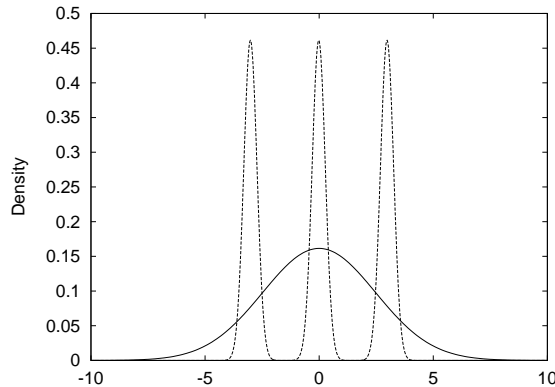


Fig. 3. Two distributions that must be checked for deviation in both directions, because one is covered by the other.

to calculate the distance from S_1 to S' , we first calculate the log-likelihood of each data point of S' in the density function $\mathcal{K}(S_1)$, then drop the lowest $\tau|S'|$ log-likelihoods. The distance is defined to be the negative of the sum of the remaining $(1 - \tau)|S'|$ log-likelihoods. To maintain the properness of the hypothesis test, the “dropping” strategy is applied to all of the Monte-Carlo distance samples $d(S_1, R)$ as well. We use $\tau = 5\%$ in our implementation and experiments, and this seems to provide for excellent results.

We stress that since the Monte-Carlo process takes the dropping into account, it does not affect the correctness of the test. The dropping strategy may have a slight, adverse effect on the test’s power, but it provides for an additional measure of safety with respect to guaranteeing a correct false positive rate.

4.3 Running the Test in Two Directions

One final issue to be discussed with respect to the density test is that any change detection test within the ChDF that makes use of probability densities should be done in two directions for maximum power. First, it is necessary to check whether S' was produced by sampling from F_S , and then it is necessary to check whether S was produced by sampling from $F_{S'}$. The reason for the two-dimensional test is that a difference in distribution may be very easy to detect in one direction using a probability density function, but not in the other. For example, imagine that F_S is a uni-dimensional Gaussian, while $F_{S'}$ is a uni-dimensional mixture of low-variance Gaussians, where $\sigma^2(F_S) = \sigma^2(F_{S'})$ and $\mu(F_S) = \mu(F_{S'})$. This is illustrated in Figure 3, where $F_{S'}$ is a mixture of three Gaussians. A typical sample from $F_{S'}$ will have data points located only in relatively high-density regions of F_S , which means that the δ statistic may be useless for comparing F_S and $F_{S'}$. To put it another way: at every location where the density of $F_{S'}$ is high, the density of F_S is also high. However, a large enough sample from F_S will likely have data points located in the low-density troughs between the peaks in $F_{S'}$. This renders a test for distributional change from S to S' far more sensitive than a test of change from S' to S .

In order to handle this in the context of the density test, the test is run twice. First, the test is run exactly as described in this section. If no change is observed, the test is run a second time in exactly the same way, except that the roles of S and S' are reversed. In order to ensure a false positive rate of p over both tests, the total probability of occurring a false positive should be split in two over both tests by making use of Bonferoni's inequality: in the first test, a false positive rate of $p/2$ should be used, and in the second test, a false positive rate of $p/2$ should be used in order to ensure an overall false positive rate of p .

4.4 Time Complexity Analysis

Assume $|S| = |S'| = N$ and $|S_1| = |S_2| = |R| = \frac{N}{2}$. Let k be the dimensionality of the data points.

In the density test, the time cost consists of two parts: the time to compute the optimal bandwidth for the Gaussian kernel density and the time to calculate the LLH-based distance. The optimal bandwidth is computed by Algorithm 6 with time complexity $O(c_1(\frac{N}{2})^2k^2)$, where c_1 is the number of EM iterations needed before convergence of the objective function. c_1 is bounded by *MaxIteration* in Algorithm 6 and in most cases c_1 is less than 50.

By Equation (10), the time complexity of computing a distance is $O((\frac{N}{2})^2k^2)$, too. Note that we do not have to redo the LLH computation in every iteration of Monte-Carlo sampling process. We can simply compute the log-likelihood of every data point in S_2 once, and then we bootstrap sample from these log-likelihood values and add them to make a distance sample. Thus, the time required for computing all of the distances (including test statistic and the Monte-Carlo distance samples) is $O(2 \times ((\frac{N}{2})^2k^2 + c_2\frac{N}{2}))$, where c_2 is the number of iterations in Monte-Carlo sampling process. c_2 is bounded by *MaxNumSample* and our experience indicates that c_2 stays fairly constant (less than 500 and often less than 10) if either the SPRT or SOBT stopping criteria from the previous Section is used. By adding up the time for optimal bandwidth and the time for distance, the overall time complexity of the density test proposed in this paper is $O(N^2k^2)$.

5. EXPERIMENTS

This section details a set of experiments designed to test the utility of the density test, as well as two other available tests for multidimensional data. We describe four sets of experiments, each of which is designed to address one of the following questions:

- (1) How is the false positive rate of our test correctly governed by the input parameter p ?
- (2) How does the power (false negative rate) of our approach compare with that of the other two alternative methods, the cross-match test and kdq-tree test?
- (3) In Section 3, we gave two sequential procedures to simulate the distribution of the statistic δ under the null hypothesis H_0 : the classic SPRT from sequential analysis theory, and our own SOBT that we developed. How do the options compare with regard to the time efficiency?
- (4) How scalable is our approach presented in the paper compare with the cross-match test and kdq-tree test?

Table I. The 13 experiment data sources.

Data source	Description
<i>CAPeak</i> (3-D)	It contains 1,817 records of mountain peaks in California, and includes each peak's longitude, latitude and height.
<i>El Nino</i> (5-D)	The data is from UCI KDD archive. It contains oceanographic and surface meteorological readings taken from a series of buoys positioned throughout the equatorial Pacific. we remove those data records with missing values and end up to have 93,935 records. We also remove the Spatio-Temporal attributes from the dataset since the changes on these attributes are not meaningful.
<i>Houses</i> (9-D)	The data set is from CMU Statlib datasets Archive. It contains 20,640 observations on housing prices with 9 economic covariates.
<i>Pine</i> (10-D)	This data set and following two data sets, <i>spruce</i> and <i>krummholz</i> , are from UCI KDD Archive. These data sets describe the forest cover type for 30 x 30 meter cells. We remove the binary variables of the datasets. The number of records are 35,754, 211,840 and 20,510 for Ponderosa Pine, Spruce-Fir and Krummholz respectively.
<i>Spruce</i> (10-D)	refer to the <i>pine</i> data set.
<i>Krummholz</i> (10-D)	refer to the <i>pine</i> data set.
<i>Bodyfat</i> (15-D)	The data set is from CMU Statlib datasets archive. It gives the estimates of the percentage of body fat determined by underwater weighing and various body circumference measurements for 252 men.
<i>Boston</i> (16-D)	the data set is from CMU Statlib datasets Archive. It consists of 506 records of Boston house price data, with corrections and augmentation of the data with the latitude and longitude of each observation. We remove 5 attributes that are irrelevant to finding interesting changes.
<i>Batting</i> (17-D)	This data set and the following <i>pitching</i> data set are from The Lahman Baseball Database. They contain pitching and batting statistics for Major League Baseball players from 1871 through 2005. The number of records is 85,979 and 36,245 for <i>batting</i> and <i>pitching</i> respectively.
<i>Pitching</i> (22-D)	refer to <i>batting</i> data set.
<i>Streamflow</i> (24-D)	We create this data set by the source data from U.S. Geological Survey (USGS) and National Climate Data Center (NCDC). It contains 7305 records of the weather and stream flow status on some observation stations in CA state. We applied PCA reduction on the data set to reduce the dimensionality down to 24.
<i>FCATread</i> (25-D)	This data set and the following <i>FCATmath</i> dataset is from National Center for Education Statistics (NCES) and Florida Information Resource Network. we create the data sets by matching the regular FL schools' information and the grade 3's FCAT reading and FCAT math scores of year 2002. The number of records is 1404 for FCATread and 1405 for FCATmath.
<i>FCATmath</i> (26-D)	refer to <i>FCATread</i> data set.

Datasets. Our tests make use of 13 experimental data sets, which are generated respectively from the 13 real data sources described in Table I. The dimensionality is indicated in the parenthesis right after each data source.

Since all of our experiments will be performed using the above data sources, we require that the size of each data source be reasonably large so that we can sample with replacement from each data source in order to obtain a true multivariate probability distribution that does not produce too many duplicate values. Since

the majority of our tests are performed on samples of size 1700², most of the data sets are large enough that sampling with replacement 1700 times will not produce a very large number of duplicates. However, among the 13 data sources, five have relatively small size: *CAPeak*, *Bodyfat*, *Boston*, *FCATread* and *FCATmath*. For these small-sized data sources, we “bump up” the data set size to 20,000 points while maintaining the distributional characteristics of the data as follows. We first randomly draw a sample point A from the small-sized data source. We then sample five points, A_1 to A_5 (with replacement) from A ’s five nearest neighbors based on Euclidean distance. By averaging A and its associated five samples A_1 to A_5 , we produce a brand new data point. We repeat the above process 20,000 times and end up with a new “bumped-up” data source having 20,000 points.

5.1 Experiment One: False Positive Rate

The first set of experiments are designed to determine the false positive rate of our method, in order to see if it is correctly governed by the input parameter p . We also check whether the the cross-match test and the kdq-tree test are correct in this respect.

To run each test, we sample 1700 data points (with replacement) from the data source and call these 1700 data points an *instance*. We repeat this sampling 100 times to get 100 instances. Because an instance is sampled with replacement from a finite population, each of the data points in an instance is identically distributed. An experimental data set contains 100 independent instances and each instance consists of 1700 i.i.d. data points, and hence any test for change over any subset of those points should return that no change has occurred. In this way, we can test the false positive rate.

To estimate the false positive rate of each of the change detection methods over a data set, we split the data set in half and apply a change detection test on all of the 100 data set instances. There are two possible outcomes associated with each instance. The detection test either says there is a distributional change in this instance, or says there is no any distributional change in this instance. The false positive of a test is defined to be the fraction of the time that the test refutes the null hypothesis H_0 when H_0 does in fact hold. Given the 100 instances of an experiment data set, the false positive rate can be approximated by the ratio of the number of instances that has been declared to have a distributional change to the number of total instances that actually have no distributional changes (i.e, 100).

The parameters and running environment of the three change detection methods are set up as follows. Since Experiment three in subsection 5.3 shows that the SOBT is more efficient (that is, it results in a more aggressive stopping criteria), we use SOBT in conjunction with the density test. In our density test, $\alpha = \beta = p/4$ and $p = 0.08$. The same p is used in the cross-match and kdq-tree tests. In the kdq-tree test, we use the same parameter settings as in the original paper [Dasu et al. 2006]. Namely, the minimum side length of a cell is set to 2^{-10} and the maximum number of points in a cell is set to 100. However, since there is no stopping criteria in this test, we arbitrarily use 1000 Monte-Carlo trials (1000 is

²1700 is chosen because it is the number of samples that the cross-match test (the slowest of the three distributional change tests that we considered) can process in a reasonable amount of time.

Table II. The false positive rates (%) on 13 experiment data sets.

<i>test</i>	<i>average</i>	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}
<i>density</i>	8	8	9	6	6	9	7	11	8	11	6	6	8	6
<i>cross-match</i>	7	7	11	7	7	8	10	4	5	2	10	6	4	5
<i>kdq-tree</i>	4	5	6	3	7	1	8	3	5	0	2	0	2	9

the value of *MaxNumSample* used by the density test).

All experiments were run on an Intel Xeon machine with dual CPU of 2.8GHz, and 5GB of RAM. We keep this setup for all the experiments throughout this section.

The false positive results are reported in Table II, where D_i represents the experiment data set created out of the i^{th} data source. The average false positive rate over 13 experiment data sets is given in the “average” column of the table.

Discussion. These results show that all the three change detection methods have a false positive rate at or less than the 8% allowed by the supplied p value, and hence a positive result from any of the tests seems to be a safe indicator of distributional change. Both the density test and the cross-match test have a false positive rate very close to the desired p value, and the kdq-tree test tends to be rather conservative.

5.2 Experiment Two: Power

This set of experiments are designed to test the false negative rate of the density test against the false negative rate for the cross-match test and the kdq-tree test.

Experiment Setup. In order to test the false negatives, it was necessary to have an instance $\langle S, S' \rangle$ such that S and S' are samples from F_S and $F_{S'}$, respectively, where $F_{S'}$ is not equivalent to F_S . Because any distributional change in the real world takes place gradually, if we take a snapshot of the distributional change at some moment, $F_{S'}$ will be a mixture of distribution F_S and a different distribution F_C . Thus, we choose to model distributional change by creating $F_{S'}$ by sampling from some F_C with a probability of L and sampling from F_S with a probability of $1 - L$ for some parameter L . Unless otherwise specified, F_S is the data source described in the beginning of subsection 5.1.

We consider five different types of distributional change, created by five different methods for defining F_C . Three of the changes take place in the full-dimensional space, and two are single-dimensional changes. F_C is defined as follows.

Full-Dimensional Changes

addgauss: In this case, F_C is obtained by sampling from the original data set and adding Gaussian noise to all the dimensions of the data source.

gmm: In this case, F_C is a GMM which has three equal-weight components. The centroids of the Gaussian mixture model are three outliers chosen from the original data source. The covariance matrix of components is a diagonal matrix with the variance of the subset on the diagonal.

mixcluster: In this case, we divide the data source into two clusters by a K-means clustering algorithm. F_S is the cluster of larger size, and F_C is the cluster of smaller size.

Single-Dimensional Changes

Table III. The false negative (%) for changes on multiple dimensions.

<i>changetype</i>	<i>test</i>	<i>avg.</i>	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}
<i>addgauss</i>	<i>density</i>	10	1	0	1	2	1	1	12	0	11	14	32	25	35
	<i>cross-match</i>	47	47	47	45	66	29	44	53	31	40	42	62	45	63
	<i>kdq-tree</i>	58	0	23	64	87	77	79	75	71	1	86	99	27	61
<i>gmm</i>	<i>density</i>	5	0	0	3	1	6	5	5	0	27	0	2	13	3
	<i>cross-match</i>	45	24	46	34	56	53	44	38	44	76	25	63	43	39
	<i>kdq-tree</i>	27	0	2	0	62	51	21	0	12	88	4	90	10	10
<i>mixcluster</i>	<i>density</i>	6	6	7	4	1	0	4	2	0	1	24	1	6	21
	<i>cross-match</i>	49	49	36	67	40	51	41	51	39	75	31	48	44	59
	<i>kdq-tree</i>	10	0	0	78	0	0	0	0	0	36	0	10	4	0

Table IV. The false negative (%) for changes on single dimension.

<i>changetype</i>	<i>test</i>	<i>avg.</i>	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}
<i>add1D</i>	<i>density</i>	34	57	36	31	26	24	22	3	2	46	48	64	57	28
	<i>cross-match</i>	53	77	56	62	60	52	51	45	29	24	48	51	69	67
	<i>kdq-tree</i>	73	76	20	82	85	96	87	83	98	20	56	99	71	70
<i>multiply1D</i>	<i>density</i>	36	19	16	19	18	19	14	1	21	78	74	73	66	48
	<i>cross-match</i>	55	49	44	49	68	55	48	47	55	53	34	83	62	67
	<i>kdq-tree</i>	73	7	82	76	68	66	90	69	98	82	61	96	68	81

add1D: In this case, F_C is created by adding a standard 1-D Gaussian variable to a single randomly-chosen dimension of the original data set. In each instance of the changed data set, a different dimension may be selected.

multiply1D: Here F_C is created just as in the previous case, except that the value of the randomly-selected dimension is multiplied by two.

For each data source, we create five experimental data sets, with each one corresponding to one type of distributional change. Each experimental data set contains 100 individual instances of size 1700. S and S' have are each of size 850. The parameter L (which controls the fraction of data points in S' that are sampled from set F_C) is chosen based upon the characteristics of the data and the type of change. Although the value of L will not affect the fairness of experiment because all the three detection methods are tested on the same sets of data, we still need to carefully choose L in order to make the experiment informative: it should not be too easy to detect the change, nor should it be too difficult. In our experiment, we calibrate the value of L so that the *cross-match* test can detect around 50% of the changes.

The false negative of each change detection test is defined to be the fraction of test instances that have been declared not to have a distributional change. The false negatives of full-dimensional changes are reported in Table III. The false negatives of the single-dimensional changes are reported in Table IV. In order to make our experiments reproducible, we give the values of parameter L for each type of change over all the 13 data sources in Table V.

Discussion. Table III shows that the density test is the most powerful out of the three methods compared. Though it is true that there was some variation from data set to data set, it seems clear that the density test is the most obvious, general-purpose choice. For each of the five types of changes, the density test had the lowest average false negative rate. Depending upon the type of change, the kdq-

Table V. The parameter L of every change on all the 13 data sources.

<i>parameter</i> L	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}
<i>addgauss</i>	0.17	0.2	0.15	0.1	0.2	0.15	0.1	0.13	0.15	0.12	0.12	0.11	0.08
<i>gmm</i>	0.2	0.15	0.15	0.1	0.11	0.12	0.13	0.12	0.1	0.14	0.08	0.1	0.12
<i>mixcluster</i>	0.12	0.14	0.12	0.12	0.1	0.12	0.1	0.12	0.12	0.13	0.15	0.12	0.11
<i>add1D</i>	0.12	0.35	0.25	0.3	0.4	0.3	0.25	0.3	0.35	0.4	0.5	0.3	0.5
<i>multiply1D</i>	0.18	0.4	0.3	0.25	0.35	0.3	0.3	0.2	0.2	0.38	0.5	0.3	0.35

Table VI. The statistic sample size by two different stopping criteria

<i>changetype</i>	<i>criteria</i>	<i>avg.</i>	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}
<i>nochange</i>	<i>SPRT</i>	104	191	119	22	91	23	78	70	101	38	161	183	131	149
	<i>SOBT</i>	18	29	19	9	11	28	34	28	17	13	19	7	10	18
<i>mixcluster</i>	<i>SPRT</i>	383	345	397	363	382	382	382	382	382	382	444	382	387	370
	<i>SOBT</i>	187	176	160	177	195	195	195	195	195	195	177	195	195	182

tree test and the cross-match test were generally comparable, with the former doing better on one-dimensional change, and the latter doing better on multi-dimensional change.

5.3 Experiment Three: Stopping Criteria

This set of experiments are designed to test the relative utility of the two methods described in the paper for choosing when to stop the Monte-Carlo repetitions within the density test. In general, the fewer iterations required to declare whether or not there has been a change, the better.

Experiment Setup. We compare the statistic sample size of two stopping criterion based on two cases: when there is not a distributional change, and when there is a mixcluster-type distributional change. In each case, we create 13 experimental data sets out of the 13 data sources, respectively. Each experimental data set contains 10 instances, and each instance has 1700 data points. The no-change instances are generated the way described in Section 5.1. The mixcluster instances are generated the way described in Section 5.2. For each experimental data set, we run the density test using both the SPRT stopping criteria and the SOBT stopping criteria. We average the number of Monte-Carlo trials required by each invocation of the SPRT and the SOBT over 10 instances and report the results in Table VI.

For both SOBT and SPRT, we use the same parameters that are used to test the false positive and false negative rates. That is, $\alpha = \beta = p/4$ and $p = 0.08$. Additionally, the step size is 5 in SOBT. In the SPRT, $\beta_0 = 0.025$, $\beta_1 = 0.015$ and the type II error is set to 0.02.

Discussion. These results show that SOBT tends to halt the the Monte-Carlo sampling process faster than the SPRT. When there is no distributional change, the statistic sample size by SPRT is at least 5 times that of SOBT. When there is a distributional change, the statistic sample size by SPRT is 2 times of that of SOBT.

5.4 Experiment Four: Scalability Analysis

Our final set of experiments test ability of the three change detection methods to scale to larger data sets.

Table VII. The running time on streamflow data set. Time unit is seconds by default. Otherwise, m stands for minutes, and h stands for hours.

<i>size</i>	100	200	300	400	500	600	700	800	900	1k	2k	3k	4k	5k	6k	7k
<i>density</i>	0.5	1.1	2.1	3.1	4.2	8.1	9.2	17	21	19	92	3m	5m	8m	10m	17m
<i>cross-match</i>	1.2	4.6	10	19	31	46	65	88	114	146	15m	51m	2.4h	4.6h	8h	26h
<i>kdq-tree</i>	0.01	0.02	0.03	0.06	0.08	0.12	0.2	0.2	0.3	0.3	0.8	2	3	7	10	13

Table VIII. Fraction of the time (in %) reported in table VI that is devoted to one-time computation that could be re-used for multiple tests.

<i>average</i>	100	200	300	400	500	600	700	800	900	1k	2k	3k	4k	5k	6k	7k
92.6	83.4	87.3	91.8	91.5	92.2	92.6	91.2	95.3	95.4	94.3	95.4	95.5	94.6	94.7	93.6	93.2

Experiment Setup. We test the scalability of the methods in terms of their running time on data sets of different size. The experiment is run on the 24-dimensional data source *Streamflow*. We sample repeatedly (with replacement) from the *Streamflow* source to create a series of data sets of size 100, 200, 300, ... 1000, 2000, ... 7000.

On each data set, we record the length of time required to run the change detection test to completion. The density test was implemented in Matlab. The cross-match test was implemented in Matlab as well, with the matching algorithm that is required by the test as an external subroutine being implemented in C (the C implementation was the one recommended by the designer of the cross-match test). The kdq-tree test is implemented in C. The results are reported in Table VII.

Discussion. The kdq-tree has significant performance benefits over both the density test and the cross-match test. Given that the kdq-tree relies mostly on a computationally efficient recursive partitioning of the data space, this is not too surprising. However, the density test still scales to reasonable data set sizes (see the discussion in the Conclusion of the paper). The cross-match test is generally unusable past a few thousand data points.

We also point out that the density test has a very high, one-time, fixed cost for the construction of the KDE. After this cost has been paid, the model can be saved and the test can be run repeatedly in a very small fraction of the time reported in Table VI. To illustrate the fraction of the total time associated with the density test that is a one-time cost, consider Table VII. This Table shows the fraction of the running time for the density test that is devoted to one-time computation for each of the data set sizes. The average fraction is around 93%. These results show that if the same algorithm has to be repeated multiple times, the computational cost of the density test is only 6 to 8 times more than the tree-based method. For a dataset with 7000 points, the overall requirements are on the order of a minute. This makes it extremely practical for a number of applications.

6. RELATED WORK

In this section, we briefly review the three tests from the literature that are closest to our own. We also briefly discuss alternative methods for kernel bandwidth selection.

The first test, based upon Kulldorff's spatial scan statistic [Kulldorff 1997], can be used to find significant spatial clusters in data; the particular application that it was developed for is detecting disease outbreaks. This statistic is obtained by first partitioning the geographic space into cells which may be of irregular shapes.

Then a zone (denoted by z) is defined as the combined region of any number of contiguous cells which can be enclosed by a circle. Denote the probability that an individual is a case within a zone by p , and use q to denote the probability associated with the region outside the zone. Kulldorff's test statistic is defined as the likelihood ratio of $\frac{\sup_{z \in Z, p > q} L(z, p, q)}{\sup_{p=q} L(z, p, q)}$, where $L(z, p, q)$ denotes the likelihood of observing the cases inside and outside zone z simultaneously. The null hypothesis for the resulting test is $H_0 : p = q$, and the alternative hypothesis is $H_1 : p > q$. The null distribution of the likelihood ratio test statistic is obtained by the Monte-Carlo method based on the assumption that the points are generated by either a non-homogeneous Poisson process or a Bernoulli process. One significant issue that must be addressed when implementing this test is computational: how does one find the region that maximizes the likelihood ratio statistic in a reasonable time? In general, this task requires a high polynomial number of computations. As a result, several research groups have looked at speeding this test [Agarwal et al. 2006; Agarwal et al. 2006; Neill and Moore 2004].

Though Kulldorff's test does check for a change of distribution, it looks for a very specific distributional change: a significant high-density region with respect to the baseline. If such a region is found, then clearly a change has occurred, but the reverse is not necessarily true. As such, Kulldorff's test is not really suitable for general-purpose change detection. Dasu et al. [Dasu et al. 2006] proposed an information-theoretic approach to detecting changes in multi-dimensional data streams that generalizes Kulldorff's test. Their approach depends on a spatial partitioning scheme (called a *kdq-tree*) to partition the data space into small cells. Based on the data count in each cell, two empirical distributions are built, representing the reference window and the testing window respectively. The Kullback-Leibler (KL) distance is used to measure the distance between the two empirical distributions. In order to measure the significance of the obtained KL-distance, they use a non-parametric bootstrap method to construct the null distribution for the KL-distance and flag an alarm if the test statistic falls in the critical region of the null distribution.

As pointed out in the introduction to the paper, Kulldorff's test and Dasu et al.'s test fit easily within the ChDF studied in this paper, and both could be improved via the use of the dynamic stopping criteria that we have discussed.

A third test for multivariate distributional change from the statistics literature is the cross-match test [Rosenbaum 2005]. In the cross-match test, every observation in $S \cup S'$ is ranked along each dimension. The rank of an observation x_i , denoted by r_i , is a vector with k elements. Then the inter-point distance δ_{ij} between two observations x_i and x_j is defined to be the Mahalanobis distance between their ranks. Formally, the distance is given by:

$$\delta_{ij} = (r_i - r_j)^T M^{-1} (r_i - r_j)$$

where M is the sample variance-covariance matrix of the ranks r_i . After that, all the $\binom{|S| + |S'|}{2}$ pairwise distances are used to construct an optimal non-bipartite matching, i.e. a matching of the observations into disjoint pairs to minimize the total distance within pairs. The cross-match statistic, A_1 , is defined to be the

number of pairs containing one observation from S and one from S' . A_1 is shown to have a restricted occupancy distribution. Furthermore, the conditional distribution of A_1 given $|S'|$ converges to the normal distribution. In a specific application where a p -value and the size $|S'|$ are given, if the cross-match statistic falls into the rejection region of the normal distribution, the test will claim that the distributions of S and S' are significantly different.

Several data driven methods have been proposed for kernel bandwidth selection. The plug-in rule [Sheather and Jones 1991] seems to be the most widely used. This method assumed fixed bandwidth for all the kernels and is not suitable when data exhibits local scale variations. The optimal bandwidth selection methods proposed by Silverman [Silverman 1986] and Wand and Jones [Wand and Jones 1995] are of limited practical use for multidimensional data as the derivation of asymptotics involves multivariate derivatives and higher order Taylor expansions. We did not find any method in the literature that associates different bandwidth for each kernel for more than three dimensional datasets. The method proposed by Comaniciu [Comaniciu 2003] allows for variable bandwidth but assumes that the range of data scales is known and chooses the bandwidth that is the most stable across scales for each data point. Unlike our method where each kernel has a unique bandwidth, this method assigns a same bandwidth for all the data points that converge to the same mode.

7. CONCLUSION

We have described a new test for distributional change called the *density test*, and evaluated the test via an extensive set of experiments. Across all of our experiments, density test uniformly shows the most power compared to the available alternatives. In terms of running time, the density test is not the most efficient of the three methods tested, but it does easily scale to data sets that are thousands of points in size. As discussed in the Introduction, unlike in many data mining problem domains where scalability to millions of data points is desirable, extreme scalability is not necessarily an important characteristic of a change detection method. In “real-life”, few distributions are stationary, and so it is highly unlikely that any sort of massive comparison over million-point databases would be undertaken, since there is almost a total certainty of distributional change. A far more useful application would be a comparison of the recent data with a large number of smaller baseline data sets to see if the recent data are different from all of them, or to see whether it is possible to eliminate one or more of the baselines from consideration. The density test would excel in such a situation, because to compensate for the large number of tests, a multiple hypothesis testing correction would be required [Miller 1966] – lowering the p or false negative parameter and making the power of the test more important, and because most of the computation associated with the density test is a one-time cost that would be amortized over the many tests.

In addition to its power, the density test has several other advantages compared to other, similar tests. Foremost among those is the fact that the number of Monte-Carlo repetitions required during change detection is determined dynamically, at test time, in a statistically rigorous fashion.

There are several possible avenues for future work. Our study of the generic

change detection framework of Sections 2 and 3 of the paper could potentially be used to define a very large number of different detection tests, and it is very possible that other tests exist within the framework whose power would exceed that of the density test.

Another important problem is extending the density test for use with the sort of very low p values that one might expect due to a multiple hypothesis testing correction. The density test (and other methods that rely on Monte-Carlo techniques) will require some modification to work in such a situation, because to accept a change at a given p value, the standard, sequential approach described in this paper will generally require on the order of at least $1/p$ Monte-Carlo repetitions. If p is 10^{-6} , this is unacceptable. To address this, we plan to explore methods for sampling from the tail of the null distribution Δ , that allow for fast acceptance of change even if the p value is very small.

REFERENCES

- AGARWAL, D., MCGREGOR, A., PHILLIPS, J. M., VENKATASUBRAMANIAN, S., AND ZHU, Z. 2006. Spatial scan statistics: Approximations and performance study. In *Proc. 12th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*.
- AGARWAL, D., PHILLIPS, J. M., AND VENKATASUBRAMANIAN, S. 2006. The hunting of the bump: On maximizing statistical discrepancy. In *Proc. 17th Ann. ACM-SIAM Symp. on Disc. Alg.*
- AGGARWAL, C. C. 2003. A framework for change diagnosis of data streams. In *SIGMOD*. 575–586.
- BAY, S. D. AND PAZZANI, M. J. 2001. Detecting group differences: Mining contrast sets. *Data Min. Knowl. Discov.* 5, 3, 213–246.
- BILMES, J. 1997. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models.
- BREUNIG, M. M., KRIEGEL, H.-P., NG, R. T., AND SANDER, J. 2000. Lof: Identifying density-based local outliers. In *SIGMOD*. 93–104.
- COMANICIU, D. 2003. An algorithm for data-driven bandwidth selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 2, 281–288.
- DASU, T., KRISHNAN, S., VENKATASUBRAMANIAN, S., AND YI, K. 2006. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Proceedings of the 38th Symposium on the Interface of Statistics, Computing Science, and Applications (Interface '06)*.
- DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39, 1, 1–38.
- KANJI, G. K. 1993. *100 statistical tests*. SAGE publications Ltd.
- KIFER, D., BEN-DAVID, S., AND GEHRKE, J. 2004. Detecting change in data streams. In *VLDB*. 180–191.
- KNORR, E. M., NG, R. T., AND TUCAKOV, V. 2000. Distance-based outliers: Algorithms and applications. *VLDB J.* 8, 3-4.
- KULLDORF, M. 1999. Spatial scan statistics: models, calculations, and applications. In *In Scan statistics and Applications, edited by Glaz and Balakrishnan*. Birkhauser, 303–322.
- KULLDORFF, M. 1997. A spatial scan statistic. *Communications in Statistics: Theory and Methods* 26, 6, 1481–1496.
- MAA, J.-F., PEARL, D. K., AND BARTOSZYNSKI, R. 1996. Reducing multidimensional two-sample data to one-dimensional interpoint comparisons. *The Annals of Statistics* 24, 3, 1069–1074.
- MILLER, R.G., J. 1966. *Simultaneous Statistical Inference*. McGraw-Hill, New York.
- NEILL, D. AND MOORE, A. 2004. Rapid detection of significant spatial clusters. J. Guerke and W. DuMouchel, Eds.
- ROSENBAUM, P. R. 2005. An exact distribution-free test comparing two multivariate distributions based on adjacency. *Journal of the Royal Statistical Society B* 67, 4, 515–530.
- ACM Journal Name, Vol. V, No. N, Month 20YY.

- SCOTT, D. 1992. *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley-Interscience, New York.
- SHEATHER, S. AND JONES, M. 1991. A reliable databased bandwidth selection method for kernel density estimation. *Journal of Royal Statistical Society Series B*, 53, 683–690.
- SILVERMAN, B. W. 1986. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London.
- WALD, A. 1992. *Sequential Analysis*. John Wiley and Sons, Inc, New York, London, Sydney.
- WAND, M. AND JONES, M. 1995. *Kernel Smoothing*. Chapman and Hall.
- WONG, W.-K., MOORE, A., COOPER, G., AND WAGNER, M. 2003. Bayesian network anomaly pattern detection for disease outbreaks. In *Proceedings of the Twentieth International Conference on Machine Learning*, T. Fawcett and N. Mishra, Eds. AAAI Press, Menlo Park, California, 808–815.