Using Access Logs To Detect Application-Level Failures
Greg Friedman†, Peter Bodík‡, Lukas Biewald†, HT Levine§, George Candea†
†Stanford University, ‡UC Berkeley, §Ebates.com

Abstract
Most Web applications suffer from application-level faults that decrease their availability. A majority of the time spent recovering from these faults is spent detecting and diagnosing them, and in some cases this can take weeks or even months. We propose failure detection and diagnosis by a combination of an automatic anomaly detection in the HTTP access logs and a visual monitoring tool. Our approach is generic and can be applied to any web application without any special instrumentation and with minimal parameterization. Evaluation performed using HTTP logs from Ebates.com shows that we can detect and localize failures that occurred at this site.

1 Introduction
Web applications are becoming increasingly complex, while at the same time the demands on reliability and availability of these systems are more and more strict. Many Web applications suffer from application-level faults that cause user-visible failures or downtime of the site. As reported in [KF04], as much as 75% of time spent recovering from these failures is spent just detecting them. There are several existing and widely-deployed techniques for detecting failures ([IBM], [HP], [Alt]), but they typically only detect problems that manifest themselves in measurable system metrics (rather than in changes to user behavior). Other techniques for detecting more subtle application-level failures (such as [CKF+]) require instrumentation of the application and/or the middleware.

While the previously mentioned approaches monitor the web applications and the servers, the perfect detector of failures are the users of the web site. They quickly notice that the site is down or that a page is unavailable. But they usually don’t let the operators know about the problem they detected; mostly they just leave the site. However, their behavior is likely to change in response to a failure: for example, if the link from the /shopping_cart page to the /checkout page is broken, users simply can’t reach the /checkout page.

We can’t ask the users about possible problems at the site nor can we directly observe their actions. However, to take advantage of users detecting failures for us, we can simply observe changes or anomalies in their behavior at the web site. Users’ behavior is well captured in the HTTP access logs at the web servers; we can reconstruct access patterns to all pages and sequences of pages in sessions of individual users. By modeling typical user behavior and access patterns, we can notice anomalies that often correspond to problems on the web site. After detecting an anomaly, we also detect the most anomalous pages and significant changes in page transitions. This information helps the operator to quickly detect and diagnose a possible failure.

In addition to serious failures that cause significant anomalies in the HTTP logs, this approach has the potential to also detect much more subtle problems. It took weeks or even months to detect many of the problems that occurred at Ebates.com. An example of such a problem is a particular HTML link that worked on all Internet browsers except a specific version of Netscape Navigator. These problems are almost impossible to detect by analyzing just the daily traffic; we need to consider much longer time periods.

Another reason for detecting failures just by using the HTTP access logs is that these logs are readily available on all web sites. This approach can thus be applied to other web sites without any instrumentation of the web applications.

Contribution
We designed and implemented three on-line algorithms that use HTTP access logs to detect anomalies in user access patterns to web applications. These anomalies are used to detect failures in the application and to localize the most anomalous pages. We
evaluate the effectiveness of the approach, and analyze what characteristics of the algorithms make them well (or poorly) suited to failure detection and localization in web applications.

We also present a GUI tool that allows monitoring of traffic and employs the above mentioned algorithms to present the results to the operator in real-time.

2 Related work

The Pinpoint system ([CKF+], [KF04]) employs anomaly detection methods for problem detection and root-cause analysis in distributed applications. While their approach involves instrumenting the application and/or middleware to discover patterns in application component dependencies, our approach uses only generic HTTP logs and treats the structure of the application as a black box.

Kallepalli and Tian [KT01] employ a form of hidden Markov Models to learn user behavior from web logs, but their focus is building an effective test suite rather than real-time problem detection and localization.

Web Usage Mining (as described in [SCDT00], for example) is a large and active field. These techniques have been applied to a variety of areas, but we believe we are the first to explore their use in application failure detection and diagnosis.

There are many commercial systems for monitoring applications and their software and hardware infrastructure, such as Tivoli from IBM [IBM], OpenView from HP [HP], and Panorama from Altaworks [Alt]. These products focus on system and internal application metrics, rather than changes in user behavior, to discover failures.

3 Approach

3.1 HTTP access logs

Ebayes.com has provided us with 5 sets of (anonymized) HTTP access logs taken from specific periods between July 2001 to October 2004. Each set of logs is a random sample (approximately 40%) of all the HTTP traffic to the site during a certain time period, ranging from 7 to 16 days. During each of the five periods, there is at least one web application failure, as well as significant periods of “normal” application behavior. These HTTP access logs contained the following information about every user request:

• Apache time stamp

• local URL for page accessed

• URL parameters

• session ID

• source application server

• anonymized user ID

3.2 Overview

We experimented with four different approaches to anomaly detection, all using unsupervised statistical learning algorithms:

• Detection of anomalous time periods using a Naive Bayes approach. We break the logs into 10-minute intervals and count the number of hits to each page. The features used are simple metrics, consisting mostly of the relative page hit frequency for the 40 most requested pages.

• A modification to the Naive Bayes approach, where the algorithm performs “probabilistic-weighted learning”, which attempts to minimize the impact of anomalous time periods on its model of normal (non-anomalous) traffic distributions.

• Detection of anomalous time periods using a $\chi^2$-test, to test if the “page profile” (vector of hit counts to the pages) from the most recent time period is from the same distribution as the “page profile” from a longer period (e.g., 6 hours).

• Detection of anomalous user sessions, using a single-class SVM with a Gaussian kernel. The features we used are page monograms, bigrams, and trigrams (i.e., subsequences of pages requests length 1, 2, and 3 from within a given user session).

3.3 Naive Bayes

We trained a Naive Bayes classifier to detect anomalies [Esk00]. We model the status of the web server as a sequence of hidden states $s_i \in \text{Normal, Anomalous}$ and observable feature vector $f_i$. We make the usual Naive Bayes assumption that every state is independent of all other states and every feature is independent of all other features.

$$p(s, f) = \prod_i p(s) \prod_j p(f_j | s_i) \quad (1)$$

We choose the sequence of states $s$ to maximize the joint probability of states and (observed) features as given in the above equation.
The conditional probability of a feature given \( s_i = \text{Normal} \) is modeled by a Gaussian distribution with the mean \( \mu \) and variance \( \sigma^2 \) parameterized by the feature \( j \). The probability of a feature given \( s_i = \text{Anomalous} \) is modeled by a uniform distribution over the range of possible values for the given feature.

Given labeled data, it would be trivial to calculate the mean and variance of \( p(j|s = \text{Normal}) \) using maximum likelihood estimation (MLE). Unfortunately, it is unrealistic to expect labeled data, especially in an online setting. Therefore, we have to do unsupervised learning. The standard approach of using Expectation Maximization (EM) to simultaneously learn the value of \( s \) and \( p(f|s) \) is too slow for real-time use on high-volume data, so we approximate it with two separate methods:

- We perform MLE with the assumption that every state is “normal”. Because a large majority of states are in fact “normal”, this isn’t a terrible assumption, but it does mean that our parameters will quickly adjust to an anomaly and stop finding it anomalous if it persists for too long. We call this Naive Bayes with UnWeighted Learning (NB/UWL), and we say this algorithm is an “eager” learner.
- We perform MLE with each time period weighted by \( p(s_i = \text{Normal}, f_i) \). Thus, the more anomalous a time period appears to the algorithm, the less it incorporates that data point into its model of a “normal” time period. This is similar to the first step of the EM algorithm, with the weights being continuously updated. We refer to this method as Naive Bayes with Probabilistically Weighted Learning (NB/PWL). This method is much slower to characterize a persistent anomalous state as “normal”, and so we say this algorithm is a “careful” learner.

We chose not to learn \( p(s) \), but rather use it as a parameter that trades of between a low false positive rate (for low \( p(\text{Anomaly}) \)) and a high anomaly detection rate (for high \( p(\text{Anomaly}) \)).

An anomaly score is reported for each time period; this score is calculated as \(-\frac{1}{2}\log(p(f_i|s_i = \text{Normal}))\).

### 3.3.1 Features

We used 42 features for both NB/UWL and NB/PWL:

- The percentage of requests, in each time interval, to each of the top 40 pages on the site
- The percentage of requests, in each time interval, to “OTHER”, i.e., all other pages on the site
- The difference in total requests between this interval and the previous interval.

We originally thought that modeling and analyzing the sequence of pages that individual users go through would be crucial to this approach (and in fact the session-anomaly SVM approach did use sub-sequences of length 2 and 3 as features), and even use other information such as URL parameters and inter-request delays. However, somewhat surprisingly, our algorithms have performed fairly well on the data sets we have with just the above 42 features.

### 3.4 \( \chi^2 \)-test for Anomaly Detection

#### 3.4.1 \( \chi^2 \)-test

\( \chi^2 \)-test is used for testing if two samples of data came from the same distribution. The input of the test are two vectors of counts \( A = (a_1, \ldots, a_n) \) and \( B = (b_1, \ldots, b_n) \). The result of the test is significance of \( A \) and \( B \) coming from different distributions. And thus, the higher the significance, the higher likelihood of \( A \) and \( B \) coming from different distributions.

The test is performed in the following way:

1. Let \( S_a = \sum_{i=1}^{n} a_i, S_b = \sum_{i=1}^{n} b_i \), and \( s_i = a_i + b_i \).
2. Compute the expected value for each \( a_i \) and \( b_i \):
   \[
   E_i^A = s_i S_a / (S_a + S_b), \quad E_i^B = s_i S_b / (S_a + S_b).
   \]
3. Compute the total \( \chi^2 \) value of the two vectors:
   \[
   \chi^2 = \sum_{i=1}^{n} (a_i - E_i^A)^2 / E_i^A + (b_i - E_i^B)^2 / E_i^B.
   \]
4. Compute the significance of the test using the \( \chi^2 \) distribution with \( n - 1 \) degrees of freedom.
5. An anomaly score is computed as \(-log(1 - s)\), where \( s \) is the computed significance of the test.

An important requirement of the \( \chi^2 \)-test is that for all \( i \), \( a_i \geq 5 \) and \( b_i \geq 5 \) (otherwise the test is not valid).

#### 3.4.2 Comparing traffic patterns using the \( \chi^2 \)-test

\( \chi^2 \)-test is often used for anomaly detection in network traffic [7]. In our problem we want to detect anomalies in the traffic patterns at the web site. In other words, we want to compare the current traffic pattern (say, the previous 10 minutes) to the historic traffic pattern (say, the previous 6 hours). To compare the traffic during two time intervals we use a \( \chi^2 \)-test.
We represent the traffic during a certain time interval as the number of hits to the top 40 pages at the web site. Thus, traffic is represented as a vector $T = (t_1, ..., t_{40})$, where $t_i$ is the number of hits to the “top $i$” page during the specified time interval. To compare vectors $H$ (historic pattern) and $C$ current pattern, we perform the $\chi^2$-test on vectors $H$ and $C$. If the significance of the test is high (i.e., the two vectors are significantly different), we declare the current traffic pattern as anomalous.

Some of the traffic anomalies are significant after a very short time (e.g., 1 minute), while others might take much longer to become evident. This is the reason why we need to vary the length of the time interval used to estimating the current traffic pattern. In our implementation, we use time intervals with length 1, 2, ..., 20 minutes.

To perform a valid test, we have to exclude all pages that didn’t receive at least 5 hits during the specified time interval. This is especially important for the very short time intervals during which many of the top 40 pages didn’t get enough hits to be considered in the test.

Because the historic traffic pattern should represent the normal traffic pattern at the site, we exclude all time intervals that were marked as anomalous from the historic time interval.

The complete algorithm for detecting anomalies is performed every minute as follows (let $t$ be the current time):

1. compute the historic traffic pattern $H$ (exclude time intervals marked as anomalous)
2. for every $t_0 \in \{1, 2, ..., 20\}$, compute current traffic pattern $C$ from time interval $(t - t_0, t)$ and perform the $\chi^2$-test. If the significance of the test is higher than 0.99, mark the interval $(t - t_0, t)$ as anomalous.

### 3.4.3 Reporting warnings instead of anomalies

The user of such an anomaly detection system probably wouldn’t like to know about every single anomaly we detect every minute. And thus, instead of reporting every anomaly, we group the anomalies into warnings.

When we detect the first anomalous time interval $(t_1, t_2)$ at time $t_2$, we consider time $t_1$ as the start of the anomaly and issue a warning. If we detect another anomaly at time $t_3$ or during the following minutes, we consider them as part of the same warning. If we don’t detect any anomaly for 10 consecutive minutes, we assume the anomaly is over; the following anomaly would thus generate a new warning.

For every warning we’d also like to report pages with the most anomalous traffic pattern and any changes in page transitions that occurred when the anomaly started.

To find the most anomalous pages, we need to compute an anomaly score for every top 40 page. We use the fact that a higher total $\chi^2$ value in the $\chi^2$ test translates into a higher significance. Since each page contributes $c_i = (a_i - E^A)^2 / E^A + (b_i - E^B)^2 / E^B$ to the total $\chi^2$ value, we use $c_i$ as the anomaly score for each page.

### 3.4.4 Detecting changes in page transitions

To detect significant changes in page transitions we need to compare the traffic before the anomaly (time interval $(t_0 - t, t_0)$, where $t_0$ is the start of the anomaly) and during the anomaly $(t_0, t_1)$, where $t_1$ is current time). And thus, for every minute of an anomalous traffic we compare the transitions to and from the top 40 pages before and during the anomaly using the $\chi^2$-test.

### 3.5 Single-class SVM

The last approach, detection of anomalous user sessions using a one-class SVM [SPST+01], was not successful. We attempted to classify entire user sessions as anomalous or normal, based on monograms, bigrams, and trigrams of the pages visited in the session. We would then decide if there was a failure in the application by the number of anomalous sessions in a time period.

Unfortunately, user sessions turned out to be too sparse to classify effectively in isolation. For example, a single user accessing a page that normally sees 1% of site traffic is not anomalous. However, if half of the users suddenly start hitting this page repeatedly, there is clearly an anomaly. Because this approach only looks at a single session at a time, it can’t detect these kinds of anomalies.

### 3.6 GUI tool for visualization and real-time anomaly detection

As part of our work, we also developed a GUI tool for visualizing traffic patterns to the site, and to run the $\chi^2$ detection algorithm and report its results. The tool provides a compact, information-rich visual representation of:

- Traffic to the 40 most requested pages (and “Other”, all other pages)
Figure 1: A screen shot of the GUI visualization tool. The vertical axis has 41 components, representing the top 40 most requested pages and “Other” (all other pages). The horizontal axis represents time. Blue represents high traffic; green, mid-level traffic; yellow, low traffic; and white, no traffic.

- User transitions from each of these pages to their 40 most common following pages in a user session.

- User transition to each of these pages from their 40 most common preceding pages in a user session.

The tool has already proved useful in our analysis of the data, and the CTO of Ebates.com has expressed significant interest in deploying the tool at her site to monitor the site in real time. We believe the tool will become the vehicle for deploying all of our detection and localization algorithms, in addition to being an information visualization tool, as our research progresses.

Figure 1 shows a screen shot of the tool.

3.7 Output of detection algorithms

The algorithms generate an anomaly score for each time period (currently, every 10 minutes for NB and every minute for $\chi^2$). As an example, figure 2 shows the anomaly score generated by $\chi^2$ for one data set. Additionally, when a time period is marked as anomalous, the algorithms generate a warning with information about the most anomalous features. Example output from the NB algorithms is as follows (page names are anonymized):

```
Anomaly detected: 2003-10-08 22:50:00
Anomaly Score: 5.98
Top 5 Anomalous Pages:

<table>
<thead>
<tr>
<th>Page</th>
<th>Anom Score</th>
<th>% of mean page frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>page_a</td>
<td>2.36</td>
<td>1730%</td>
</tr>
<tr>
<td>page_b</td>
<td>1.79</td>
<td>1423%</td>
</tr>
<tr>
<td>page_c</td>
<td>1.14</td>
<td>1048%</td>
</tr>
<tr>
<td>page_d</td>
<td>0.07</td>
<td>37%</td>
</tr>
<tr>
<td>page_d</td>
<td>0.07</td>
<td>42%</td>
</tr>
</tbody>
</table>
```

4 Test Data and Methodology

Ebates.com provided us with 5 sets of HTTP logs to their site, each with 1-2 weeks of data from a subset of their web servers. Each log set was known to contain at least one failure.

Our methodology consisted of the following steps:

1. With little or no knowledge of what events occurred in a data set, run our detection algorithms. For each data set, the models were initialized as untrained at the beginning of the data set and learned in an online fashion.

2. Examine the resulting anomaly detections.
3. Use our GUI tool to visualize traffic patterns in anomalous regions (and in many normal regions as well). Also use Matlab to graph our anomaly scores and traffic patterns over time.

4. Discuss our findings with the CTO, and find out “what really happened” as best as Ebates.com could reconstruct. Repeat steps 3 and 4 as necessary.

5. Classify each site failure event as major or minor. A major fault is one that caused significant disruption to the site; these were the failures the CTO had in mind when she provided the data sets to us. Any other failure which was revealed to us by Ebates, or which we discovered and confirmed with the CTO, was classified as minor.

6. Obtain an estimate from the CTO (on a scale of 1-10) as to how helpful the information provided by the algorithms before the faults would have been in early diagnosis of the problem. The average of the unambiguous responses for an algorithm is its “localization score.”

7. Classify each anomaly detection as either a true or false positive. A true positive was a detection of a confirmed major or minor fault. False positives were any other detections with an anomaly score at least 75% of the anomaly score of the lowest-level true fault in a data set.

8. False positives are further classified into “known” and “possible” FP’s. Known FP’s are clearly attributable to a non-fault event, such as a failure-free update one or more web pages. Possible FP’s are those we could not attribute to an event, or more often, that we could not unambiguously determine whether or not the associated event was a failure.

9. Evaluate our results.

4.1 Notes on results

4.1.1 Subjective aspects

Since this is real-world data, there’s a fair amount of noise in it, and thus a highly precise analysis is difficult. We therefore had to make judgement calls in certain cases, and our results incorporate these subjective judgements. For over 80% of the events in the data set, however, we are quite confident in our analysis.

4.1.2 False positives

There remain some number of time periods in which we still don’t know, for example, if what we detected was a very early manifestation of an impending failure or just an normal update of code to the web site. It is for this reason that we classify false positives into “known” or “possible”, and thus we provide a range of false positives when presenting our summary of results in section 5.

We also note that most of the “false positives” in our results are clearly attributable to known events that did indeed change user behavior. The CTO informed us that getting warnings from our algorithms would generally not be problematic in these cases, and often would actually be useful to provide some level of validation as to the nature of the changes. In the results that follow, however, we maintain a strict definition of false positive; unless there was some kind of known failure associated with the detected anomaly, we count it as a false positive.

4.2 Note on Terminology

In our discussion of results, we describe the behavior of our algorithms as if they were running in real-time on the data sets. For example, if the algorithms detect a problem starting at 3 PM on a particular day in the data sets, and Ebates actually detected the problem at 6PM on that day, we say an algorithm “detected the anomaly 3 hours before Ebates did.”

5 Results Summary

5.1 Performance on Major Faults

There were 6 major faults in our five data sets. For 5 of these faults, the algorithms both detected the problem and provided useful localization information before Ebates was able to diagnose the problem. The other fault was detected concurrently with its occurrence.

The detection performance of the algorithms on these faults was as follows:

<table>
<thead>
<tr>
<th>Measure</th>
<th>NB/PWL</th>
<th>NB/UWL</th>
<th>χ²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults Detected</td>
<td>5/6</td>
<td>4/6</td>
<td>6/6</td>
</tr>
<tr>
<td>Detection rate</td>
<td>83%</td>
<td>67%</td>
<td>100%</td>
</tr>
<tr>
<td>False positives</td>
<td>1-4</td>
<td>1-4</td>
<td>1-3</td>
</tr>
<tr>
<td>Precision</td>
<td>56-83%</td>
<td>50-80%</td>
<td>67-86%</td>
</tr>
<tr>
<td>Local. score</td>
<td>8.5/10</td>
<td>n/a</td>
<td>4/10</td>
</tr>
</tbody>
</table>

χ² had the highest detection rate (100%), followed by NB/PWL and then NB/UWL. χ² also had the
highest precision. NB/PWL performed significantly better than $\chi^2$ in providing useful diagnostic information, as reflected in its localization score. (Note that we did not collect localization score data on NB/UWL).

The following should be noted for the above results table:

- Precision is defined as $\frac{TP}{TP + FP}$; that is, the number of detections of an actual problem divided by the number of total detections.
- A FP was classified as “major” if its peak anomaly score was at least 75% of the peak anomaly score for any of the major faults in the data set. Otherwise a FP is classified as “minor”.
- Our algorithms detected low-to-medium level anomalies on a majority of nights, usually from about midnight to 2AM. These are not included in the false positive scores in our results tables, since they are easily filtered out; we analyze night anomalies separately in sections 5.4 and 7.4.

We now look at time to detection on the major faults. For several of the faults it is impossible to determine exactly when a problem was introduced or even exactly when it started causing problems; we therefore measure the amount of time before Ebates’ initial localization of the fault that our algorithms detected related anomalies. The results are below.

<table>
<thead>
<tr>
<th>Failure</th>
<th>NB/PWL</th>
<th>NB/UWL</th>
<th>$\chi^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set 1</td>
<td>50 hrs?</td>
<td>50 hrs?</td>
<td>50 hrs?</td>
</tr>
<tr>
<td>Data set 2</td>
<td>7 days</td>
<td>n/a</td>
<td>7 days</td>
</tr>
<tr>
<td>Data set 3</td>
<td>3 hrs</td>
<td>3 hrs</td>
<td>-3 mins</td>
</tr>
<tr>
<td>DS 4 Fault 1</td>
<td>n/a</td>
<td>n/a</td>
<td>5.5 hrs</td>
</tr>
<tr>
<td>DS 4 Fault 2</td>
<td>0 min</td>
<td>0 min</td>
<td>0 min</td>
</tr>
<tr>
<td>Data set 5</td>
<td>1 hr</td>
<td>1 hr</td>
<td>1 hr</td>
</tr>
</tbody>
</table>

Our earliest detection occurred in data set 2, where we detected and localized the problem 7 days before Ebates.com did. Our least timely detection occurred on the second fault in data set 4, where an anomaly in traffic was detected at the same time that Ebates.com detected the problem.

The question mark in data set 1 reflects the fact that we are not certain if that detection represents the actual onset of the problem (see section 6.1); if not, the detection in this data set was approximately concurrent with Ebates’ detection.

5.2 Performance on Minor Faults

In addition to the 6 major faults, there turned out to be 7 minor faults in our data set. The detection performance of the algorithms is given below.

<table>
<thead>
<tr>
<th>Measure</th>
<th>NB/PWL</th>
<th>NB/UWL</th>
<th>$\chi^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults detected</td>
<td>4/7</td>
<td>4/7</td>
<td>4/7</td>
</tr>
<tr>
<td>Detection rate</td>
<td>57%</td>
<td>57%</td>
<td>57%</td>
</tr>
<tr>
<td>False positives</td>
<td>3-7</td>
<td>3-7</td>
<td>0-2</td>
</tr>
<tr>
<td>Precision</td>
<td>36-57%</td>
<td>36-57%</td>
<td>67-100%</td>
</tr>
</tbody>
</table>

Our algorithms performed better on the major faults than on the minor faults, which is not unexpected.

Among the minor faults, two of the three false negatives were database alarms, and may not have had any discernible impact on users. The third false negative was a brief introduction and then removal of a problematic page; again, there’s no evidence that this affected users in any significant way.

The three known minor false positives that the NB algorithms detected were all failure-free code updates to the application.

We did not perform localization or TTD analysis on the minor faults.

5.3 Comparison with baselines

Figure 3 summarizes overall detection rate vs. false positives as the anomaly threshold is varied. We compare our algorithms to two simple baselines used in practice. These results are for the three data sets (out of the five) in which we have strong confidence
in our understanding of every event. Note that in this graph, the night anomalies are counted as false positives.

5.4 Night anomalies
Our algorithms detected low-to-medium level anomalies on a majority of nights, usually from about midnight to 2AM. These are accounted for separately from the other false positives because they are easily identified and can be filtered out, and because we wanted to analyze them independently.

To analyze the significance of the night anomalies, we computed a "signal-to-noise" ratio, which measures the anomaly scores given to the night anomalies compared to the anomaly scores given to the major faults. This is done for each algorithm, for each data set.

The S/N ratio is computed as follows: For each data set, measure the peak of the night anomaly score on each night. Average these nightly peaks: this is the "noise" in the data set due to the night anomalies. The "signal" is simply the peak anomaly score during the data set’s major fault (or the average of the two peaks in data set 4). The S/N ratio is the signal divided by the noise.

This is not a perfect measure by any means, but it does give some indication as to of the significance of the night anomalies. The results are as follows:

<table>
<thead>
<tr>
<th>Measure</th>
<th>NB PWL</th>
<th>NB UWL</th>
<th>$\chi^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/N range</td>
<td>2.5-6.5</td>
<td>3.5-8</td>
<td>1.25-40</td>
</tr>
<tr>
<td>S/N avg</td>
<td>4.4</td>
<td>4.8</td>
<td>11.8</td>
</tr>
<tr>
<td>S/N median</td>
<td>3.5</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

We have a few hypotheses about why we see these anomalies at night; see section 7.4.

6 Discussion of Results for Each Data Set
We now discuss our detailed results on each data set separately.

6.1 Data set 1
There was one major crash of the site in this data set.

- All 3 algorithms (NB/PWL, NB/UWL, and $\chi^2$) detected significant anomalies 2 days prior to Ebates’ detection of a major problem, and 2 and a half days before the crash. These anomalies involved the introduction of two pages that were later implicated as being closely related to the crash. (Note that the major bug at fault may have been introduced at this time, or may have been introduced at some time between this point and the crash two days later).

- $\chi^2$ had one (possible) false positive: the early detection 2 and a half days before the crash. NB/UWL and NB/PWL each had two (possible) false positives.

The CTO informed us that even if the initial detection 2 and a half days before the crash did not represent the onset of the major bug, the warning and localization information provided at that time would have been very helpful in diagnosing the problem when the major failure did start to occur.

Figure 4 shows anomaly scores over time for NB/PWL and $\chi^2$. Some interesting differences between the algorithms are apparent, which we discuss in section 7.
6.2 Data set 2

Data set 2 presented an interesting case because a problem was introduced and persisted for over 7 days before it was diagnosed and fixed by Ebates.

- NB/PWL and $\chi^2$ detected the introduction of the problem 7 days before Ebates’ diagnosis. (NB/UWL detected an anomaly at this time too, but it was close to its noise threshold, so we do not count this as a detection for NB/UWL.)

- NB PWL provided localization information at problem introduction time, and during the entire 7 day anomaly, that the CTO said would have been extremely helpful in helping to localize the problem.

- Because of the length of the anomaly, NB/UWL and $\chi^2$ “inverted”; that is, they began to view the anomalous period as normal. NB/PWL, which is a “careful learner”, continued to detect the problem for the entire 7 days. (See section 7.1 for details).

6.3 Data set 3

Data set 3 contained one significant failure which caused a site crash.

NB PWL and NB UWL detected the anomaly 3 hours prior to Ebates’ detection of the problem (4 1/2 hours before the crash). The anomaly was at a low level at initial detection and grew more significant as the problem escalated. $\chi^2$ detected the problem 3 minutes after Ebates.com detected the problem.

The localization information generated by NB PWL was rated 10 out of 10 by the CTO, in terms of how useful it would have been in helping to detect and diagnose the problem earlier than they did. The CTO believes they may have been able to avoid the crash if they had this information.

All three algorithms were affected by significant night anomalies in this data set, lasting from approximately midnight to 2 AM on most nights.

6.4 Data set 4

Data set 4 contained 2 significant failures and one minor problem (a short DB alarm).

The first significant failure was an application-level bug with no obvious performance implications. $\chi^2$ detected this failure 5.5 hours before Ebates could diagnose the problem. Both NB algorithms detected the problem at the same time, but the anomaly scores were within the noise level, so we don’t count this in our results as a detection. However, the localization information provided by the NB algorithms at this time was rated rated by the CTO as 8 out of 10 in diagnostic utility.

The second significant failure was a runaway query causing significant DB performance problems. All three algorithms detected this failure concurrently with Ebates’ detection.

The short DB alarm went undetected by our algorithms.

Figure 5 illustrates our algorithms’ behavior over this data set.
6.5 Data set 5: Case study

For data set 5, we present a more detailed description of the data and results; please refer to Figure 6.

6.5.1 What went wrong

The major problem that occurred during this week was a site slowdown (and then crash) caused by the account page. The bad account page had been up for years, but it caused major problems in this time period because quarterly checks were mailed to a larger-than-ever customer base during this week, causing a previously unseen upsurge in activity to the account page as customer checked their accounts.

The account page problem was first detected by Ebates on Monday (Day 3 on Figure 6) at approximately 6:23am. It was diagnosed by Ebates at approximately 12 noon, and the offending page was taken down about 12:30pm. The problem appeared again on Day 4 at 5:38am - 7:13am (Ebates.com staff tried putting the page back up, but took it down again). On Day 6 starting about 8pm, there was massive quality assurance of this page against 2 servers on the live site to confirm the fix. The fix was completed on Day 6 at 11:21pm.

Three other smaller failures also occurred during this week, all on Day 2 (the day before the account page problem was diagnosed.) These failures manifested as database performance alarms. The first two of these were caused by the payment processing system, which is run once per quarter and was putting a large load on the database. (The CTO knew about these but did not tell us about these until we showed her the details of the anomaly; she joked that she was trying to keep these failures secret, and these results “busted her”.) The third alarm may have been related to the account page problem, but we can’t determine this with certainty.

6.5.2 What the algorithms detected

The anomalies that we discovered using our algorithms and the GUI tool are the following:

1. We detected an mid-size anomaly on Day 2 starting 1:40PM.
2. We detected a significant anomaly on Day 2 from 19:24 until 21:05, centered around two pages not directly related to account activity. The number of hits to these pages increased from less than 5 every minute to about 50 hits a minute.
3. The next anomaly, the largest in the data set, starts at 11:07 on Day 3. The Naive Bayes algorithm reported account-related pages as the most anomalous, followed by login pages.
4. The last significant anomaly starts at 19:51 on Day 6.
5. We detected smaller anomalies every night mostly related to two relatively infrequent pages (different from the other pages mentioned thus far).

6.5.3 Analysis

After comparing the detected anomalies with the actual sequence of events and discussing the results with the CTO of Ebates.com, we can conclude the following:

- **DB alarms:** The detections on Day 2 corresponded to the database alarms. The first database alarm was not detected. The second alarm was detected as it was occurring (and after). The third alarm was detected over an hour before it occurred (anomalous behavior began before the DB hit performance problems). Figure 7 illustrates this behavior.

- **First warning:** The third database alarm on Day 2 was detected as an anomaly from 19:24 to 21:05 on Day 2. This may have been the first manifestations of the account page problem, though we can’t say this for certain because the most anomalous pages are not directly related to the account page. However, Ebates’ CTO said
that knowing about this anomaly would have put the site operators on alert for possible problems, and may have led them to detect the problem earlier than they did.

- **Second warning with localization:** The next anomaly was detected when the Ebates' staff already knew there was a performance problem but didn’t know what was causing it. The warning reported by our algorithms on Day 3 at 11:10 showed the two problematic account pages as the most anomalous ones. The CTO said that this would have been a strong diagnostic hint to the operators, who without this information diagnosed the problem about 50 minutes after this point.

- **anomalies caused by QA testing & fix push:** The fourth anomaly (Day 6 at 19:51) was due to a massive test effort of the fix on the live site.

7 Analysis and Comparison of Algorithms

We now take a closer look at the characteristics of our different detection algorithms.

7.1 Impact of Careful Learning

Figure 8 (a reproduction of Figure 4) illustrates significant differences in the behavior of NB/PWL and \( \chi^2 \).

One difference reflects the impact of the “careful” learning of NB/PWL resulting from its Probability-Weighted Learning. At the introduction of the problematic pages, NB/PWL detects a significant change in the distribution of the traffic to the site. This raises the anomaly score significantly, and so NB/PWL weights this time period extremely low in incorporating it into its model of normal behavior. Thus, it continues to detect the anomaly for all 7 days. NW/UWL, being an eager learner, quickly decides that the problematic behavior is “normal.”

Figure 9: Comparison of the ability to handle long anomalies of NB/PWL vs. NB/UWL. NB/PWL is a careful learner and does not (for the most part) incorporate the anomalous time periods into its model of normal behavior. Thus, it continues to detect the anomaly for all 7 days. NW/UWL, being an eager learner, quickly decides that the problematic behavior is “normal.”

7.2 Sensitivity to Different Anomaly Types

Another difference between NB (both PWL and UWL) and \( \chi^2 \) is what kinds of traffic changes they are most sensitive to. Because NB models each feature as univariate Gaussian, it is most sensitive to increases in frequency to infrequent pages. To understand why, consider a page that normally comprises only 0.1% of hits to the site (with variance 0.01%). This page is modeled as \( N(.001, .0001) \), and thus an increase in that page to, say, 5% of page hits is modeled as extremely improbable. In contrast, the \( \chi^2 \) model does
not give nearly as much weight to infrequent pages; in fact, highly infrequent pages are not generally considered because $\chi^2$ requires at least five hits to each page in an interval for the test to be valid. This results in $\chi^2$ being more sensitive to changes (both increases and decreases) in frequent pages.

This can be seen in Figure 10, which is for another data set. NB is able to detect this anomaly 3 hours before $\chi^2$, because it detects an increase in hits to very infrequent pages. These pages continue to increase in frequency, and 3 hours later they end up having a major deleterious effect on the site. This causes changes in hits to highly frequent pages, which $\chi^2$ then detects.

7.2.1 Localization Ability

Another advantage to NB’s sensitivity to infrequent page increases is that this makes NB an effective diagnostic tool, at least in the data sets we have examined to date. Most of the failures in these data sets were related to bugs in relatively infrequent pages. A corresponding increase in hits to these pages often caused, or was an effect of, the bug’s manifestation. Reporting the anomalies in these infrequent pages thus provided very useful diagnostic information. In contrast, $\chi^2$ often reported some of the most frequently accessed pages as the most anomalous, which is not as useful for localization because these pages were generally affected at some point by most failures.

7.3 Susceptibility to False Positives

The flip side to early detection is sensitivity to false positives, as illustrated in Figure 11. Here, the introduction of a new page to the site (which was normal and did not cause any problems) caused the NB algorithms to detect a false positive. NB/PWL, being a “careful” learner, takes 9 hours to decide that this new page is not problematic. NB/UWL learns this in about 90 minutes. The new page does not cause $\chi^2$ to react at all, and so it avoids the false positive.

7.4 Night Anomalies

We have a few hypotheses about why our algorithms detect anomalies many nights from approximately midnight to 2 AM, as discussed in section 5.4:

- Because there are far less users at night, there is naturally a higher variance in the feature values, causing the anomaly score to fluctuate. If this is the cause, a simple solution would be to sample a set number of past page requests rather than a fixed time interval.
- User behavior may naturally vary significantly with time of day. People may use the site differently late at night, and/or there may be a larger percentage of robots and automated crawlers late at night. This could be addressed by incorporating time of day into our models.
- We know there are automated processes such as database backup that Ebates runs nightly at approximately these times; these may be having a discernible effect on the site.
We have not yet tested these hypotheses, though we intend to do so in the near future.

### 7.5 Algorithm Comparison Summary

A qualitative summary of the high-level characteristics of the 3 algorithms is as follows:

<table>
<thead>
<tr>
<th>Measure</th>
<th>NB/PWL</th>
<th>NB/UWL</th>
<th>(\chi^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTD</td>
<td>very good</td>
<td>very good</td>
<td>good</td>
</tr>
<tr>
<td>FN Rate</td>
<td>good</td>
<td>very good</td>
<td>good</td>
</tr>
<tr>
<td>FP Rate</td>
<td>fair</td>
<td>fair</td>
<td>good</td>
</tr>
<tr>
<td>Localization</td>
<td>very good</td>
<td>good?</td>
<td>fair</td>
</tr>
<tr>
<td>Long Anoms</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

The table above summarizes the performance of algorithms according to 5 measures: TTD (time to detection), FN (false negative) rate, FP (false positive) rate, Localization (utility in helping site operators localize problems), and Long Anoms (ability to handle long-lasting anomalies).

### 8 Conclusion

In this paper we developed and evaluated a set of algorithms and tools for automated detection and semi-automated localization of web site failures. We employed relatively simple machine learning techniques and used simple features derived from readily available HTTP logs. We tested our algorithms on real-world HTTP logs from Ebates.com, and our results were reviewed and critiqued by the CTO of company. Our initial results are highly encouraging. We believe that this approach has the potential to detect and localize subtle application-level errors and other problems that might be difficult to detect by other automated means.

### 9 Future Work

We intend to continue our work on this area, as it seems quite fertile for research. There are many other things we intend to do in the near future, including:

- Refine our quantitative measures for more precise, objective analysis. Do a more thorough comparison with a variety of baselines.
- Test our algorithms on longer data sets (for example, 2 months of continuous logs).
- Deploy the tool at Ebates.com for real-time use on their site.
- Investigate ensemble methods to combine the strengths of the different approaches.
- Refine the algorithms. For example, we would like to address the night anomaly problem and decrease our false positive rate. We believe there are some relatively simple methods we could employ to do this.
- Modify Naive Bayes to use true EM for the unsupervised online learning problem. This has the potential to result in more accurate classification.
- Try to detect more subtle failures by using a richer feature set.
- Perform a more thorough investigation of related academic and commercial work

We look forward to further research in this exciting area.

### Acknowledgements

We would like to thank HT Levine, CTO of Ebates.com, for providing invaluable assistance in this project.

### References

