Statistical Anomaly Detection

version 7/15/13 10:28 AM

Philosophy

eBay has a very large number of servers, running a layered set of software. The hardware and software layers of these servers are monitored, logging cpu load, memory usage, and many other *monitored signals* at frequent intervals. Like every large web site, eBay has occasionally suffered *disruptions*, where some of its services were degraded or even completely unavailable to customers. A post-mortem analysis of these disruptions has always revealed that at least one monitored signal exhibited unusual behavior before the disruption began to significantly affect end-users, but these anomalies failed to be detected.

Thus the problem addressed by this report: how to monitor signals in way that can detect disruptions before they affect users, and do so with few false positives. We need to decide when a signal is exhibiting *anomalous* behavior that is likely to indicate a site disruption. Then we can scan all the monitored signals, and raise an *alert* when we detect an anomaly.

Summarizing the terminology, our goal is to define properties of *monitored signals* that are highly likely to indicate a site *disruption*. We call such properties an *anomaly*, and when we see one we raise an *alert*.

To illustrate our approach, it helps to compare with the methodology developed for machine learning problems. It is well established that learning problems can best be tackled by splitting them into two parts: developing features, and then finding a function that maps features to a target. For example, suppose the problem is to recognize the scanned image of a digit, and produce a target that is a numeral from 0 to 9. It will be very difficult to get a high recognition rate if the raw bitmap is feed directly into a machine learning algorithm. In this case feature development is straightforward: simply normalize the images. Here's a second more typical example. Users list items for sale on eBay, and sometimes the buyer is not satisfied and files a complaint (a bad buyer experience or *BBE*). The problem is to predict in advance items that might lead to a BBE. It's clear that good prediction for this problem requires some non-trivial factor development work. Simply dumping all the information submitted by the seller when listing into a learning algorithm will not work well. At the very least you also need to dump historical information for the seller. And since fraudulent sellers are likely to create new accounts, information on all sellers is needed together with a way to detect linked accounts. It is unlikely that a learning algorithm can work well on directly on all the raw data.

Back to anomaly detection. The theme of this report is that detecting disruptions also splits into two parts: developing the right 'features', and then feeding these features into a statistical system that detects anomalies in the features. If done correctly, the detected anomalies will have a high correlation with site disruptions and can be used to create alerts with a low false positive rate. Most literature on anomaly detection focuses on the second part (for example Chandola 2009). Our goal is to illustrate the importance of the first part.

A Warm-up Problem (Muninn)

As a warm-up problem, consider a signal from the highest software layer: the number of searches (**srp)** done on the US site ebay.com. This is part of a system called *Muninn*. **srp** is recorded every 2 minutes. Figure 1 shows the value of **srp** on Mar 25. The horizontal axis ranges from midnight to midnight, PDT. Circled in blue are two suspicious blips, occurring around 4:00 am and 10:30 pm. Even the most sophisticated statistical method can not reliably determine if these should be considered anomalies, that is, if they indicate a site disruption.

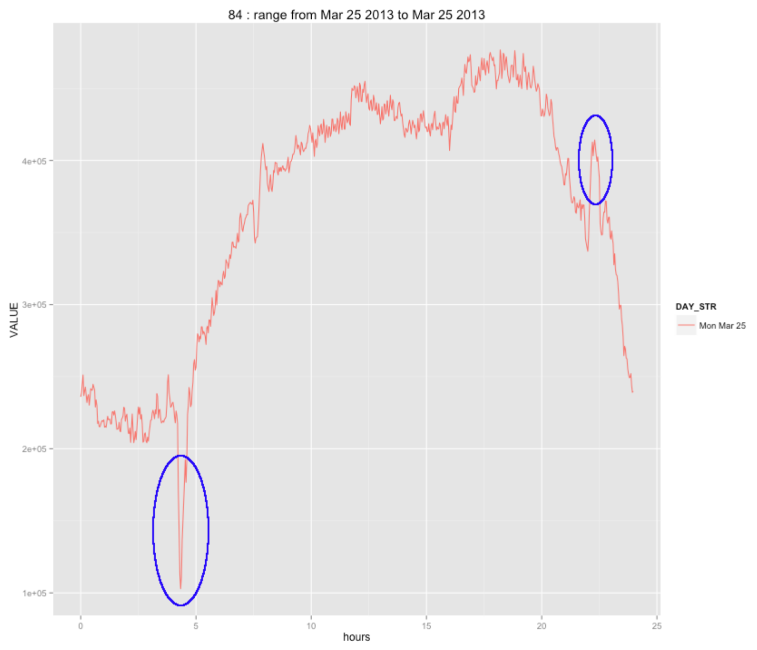

Figure 1 The y-axis is the number of searches/2 minutes. The x-axis covers a 24 hour period on Mar 25. The circled regions look suspicious.

But now consider Figure 2 which shows the values of **srp** on two additional days. Based on this data it seems likely that the blip at 4:00 am on Mar 25 should be categorized as an anomaly, but not the blip at 10:30 pm. This suggests a simple feature, based on comparing the current value of **srp** with its value 24 hours ago, 48 hours ago, 72 hours ago, etc. Using this feature the circled value on the left in Figure 1 will show up as an anomaly, but the circled on the right will not. Notice this very important point: having a history of the signal is key to devising a good feature.

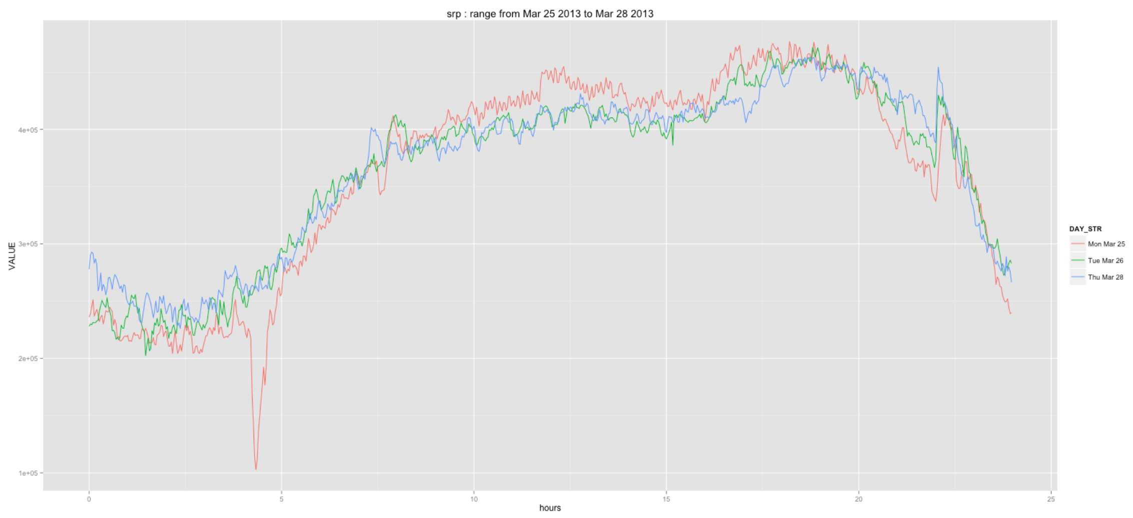

Figure 2 The y-axis is the number of searches/2 minutes. The x-axis covers a 24 hour period. The three curves represent three different days: Mar 25, Mar 26, and Mar 28.

A more Complex Feature (Atlas)

Here is a more interesting example of a feature, again related to searches. There is a monitoring system that periodically issues a query and reports *metrics* on the resulting item set, such as the number of items returned, the average list price, the fraction of items that are auctions, etc. The monitoring system repeatedly cycles through a fixed set of 3000 queries. This is part of the *Atlas* system. As mentioned in the previous section, the goal is to detect a problem with the search software (a disruption) before users do. In this case the signal is identical to what users see, so the system can't literally find the disruption before a user. But it can detect and report the problem much more quickly than waiting for a user to phone customer support, having customer support recognize the call as a site problem, and report it to engineering.

Figure 3 is a plot of a (metric, query) pair.

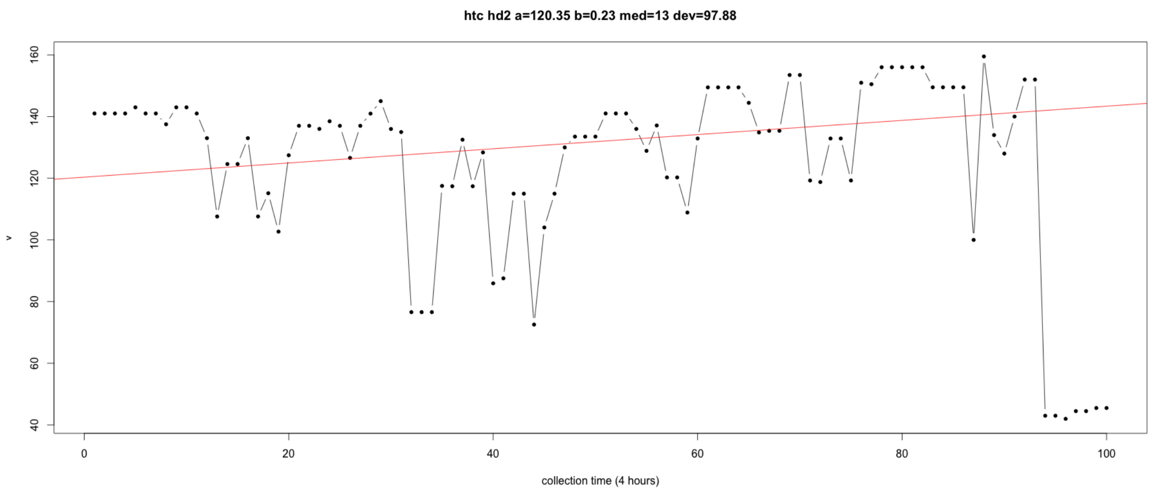

Figure 3 Data for the query *htc hd2*. The y-axis is the value of the metric *median sale price*. The x axis is the time the query was issued. The leftmost point is 12:00 pm Nov 11, the rightmost point is 4:00 am Nov 28. The dots are 4 hours apart.

The horizontal axis is time, with the right-hand side the most recent data point. The vertical axis is the value of the metric *median sale price* on the query *htc hd2*. The signal is very noisy, but it appears to have unusual behavior in the region $95 \leq t \leq 100$. But this should not be considered an anomaly because there's no disruption: this type of behavior is perfectly normal, as we now show. Although the points near t=100 are not an anomaly, they are surprising, and we quantify the *surprise* by fitting all the points to a line (shown in red), and computing the (unsigned) deviation of each point from the line. The surprise of a point is its deviation divided by the median deviation of all the points. For the most recent value (rightmost point at t=100), this is 97.9/13.4 ≈ 7.30.

Should a surprise value of 7.3 be considered an anomaly? Figure 4 shows a histogram of surprise values of *median sale price* for 2300 queries. A surprise of 7 is not unusual. Because the histogram is so bunched up, a histogram of log(surprise) is also shown. Since log(7.3) ≈ 2, it is more clear that this value of surprise is not all that unusual. Quantitatively, the percentile of the surprise for the query *htc hd2* is about 96%.

Incidentally, this example shows the difficulty in getting a low false positive rate. There are about 3000 queries, 40 metrics, and 6 collection periods per day. So there are 3000×40×6 =

720,000 graphs just like Figure 3 each day. To drop as low as 1 false positive per day would require only triggering on graphs with a surprise so high that it happens 0.00014% of the time. Such a stringent cutoff is likely to overlook many real disruptions.

The way around this is aggregation. Since there will always be a few queries with a high surprise, we construct a feature that captures the number of queries with a high surprise. A sudden change might be a good indicator of a disruption. This is done separately for each metric. To make this quantitative, instead of counting the number of queries with a high surprise, we examine a quantile (specifically 0.9) of the surprise values of all the queries of a metric. Here's how it works.

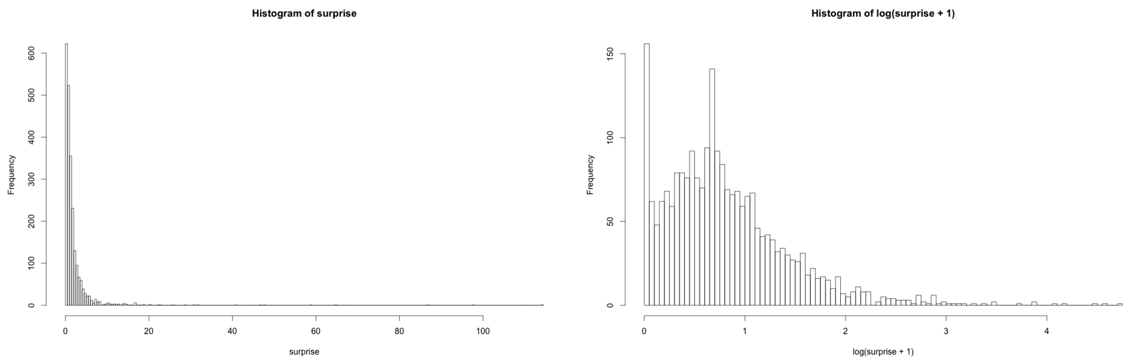

Figure 4 Histograms of surprise values for 2300 queries. Because the histogram is so bunched up near 0, a histogram of log(surprise+1) is also shown.

The surprise of the most recent value, as computed in Figure 3, depends on three things: the metric, the query and the 4-hour collection window of the latest value. The quantile is computed by picking a metric and collection window, gathering up the surprise of all the queries, and then taking the 90% quantile of those numbers. So there is a quantile for each (metric, collection-period) pair. Every four hours the following process is performed for each metric M: recompute the surprise in M for each query and determine the 90% quantile of these numbers. This gives a new value for the 90% quantile for M. The quantiles when M is *median sale price* over a two-week period are shown in Figure 5.

On the left is a histogram of their values, showing they are clustered near 3.6, with a range from 3.0 to 4.6. The right hand displays the same data in a time series, suggesting there's no strong correlation. This shows the feature is fairly smooth, and might be candidate for anomaly detection. But is it too smooth? Will it actually show an anomaly during a disruption? We went back over historical data and discovered that for each known disruption at least one metric had an anomaly using this feature. And there was only one anomaly that did not correspond to a known disruption (although we never determined if this was an unknown disruption or a false positive).

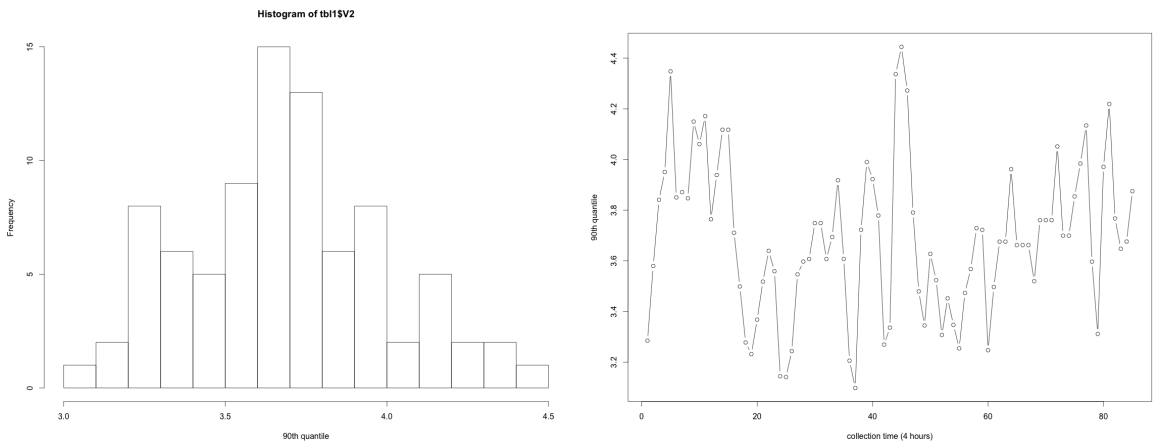

Figure 5 90th quantile of surprise for *median sale price* from Nov 14 to Nov 28. The 100 values of the quantile are shown as a histogram on the left, and plotted individually in time-sorted order as a time-series on the right.

Here is an example of an anomaly that corresponds to a genuine disruption. While Figure 5 shows the data from up to 07:00 on Nov 28, Figure 6 adds four more time points. Now this is clearly an anomaly. Figure 5 is vaguely normal with a mean of 3.7 and a standard deviation of 0.3. The anomaly in Figure 6 is 29.6, which is |29.6 – 3.7/0.3 ≈ 86 standard deviations from the mean. So the historical value of the quantile appears to be a great feature, and once again archiving historical values are required to compute it.

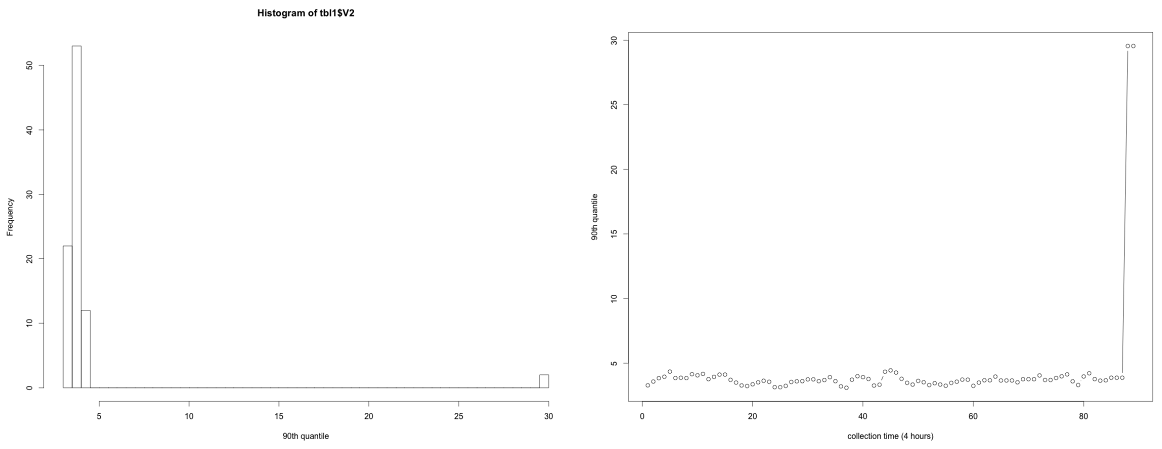

Figure 6 90% quantile of surprise from Nov 14 to Nov 28, but going 16 additional hours (4 collection times) past Figure 5. Shown as a histogram on the left, and a time series on the right

The Statistical Test

As we said in the introduction, we split anomaly detection into two parts. We've just discussed feature construction, what about part two, the statistical test? Combined with good features, even ultra-simple statistical tests already give excellent results.

For the warm-up problem (Muninn), the features were the value of **srp** 24 hours ago, 48 hours ago, 72 hours, etc. The statistical test is a robust version of |*x* – mean|/sd. The mean is replaced by the median, and the standard deviation is replace by the median absolute deviation (MAD). MAD is computed just like it sounds: compute the median, then for each value compute the absolute value of distance to the median, and then take the median of these distances. Summarizing: every two minutes when a new value of **srp** is recorded, set *x* to be the latest **srp**, compute the mean and sd over the past 30 **srp**'s (not including the most recent) and declare an anomaly if |x – median|/MAD is large.

For the second problem (Atlas), the feature was the historical values of the 90th quantile of the surprise values of all the queries. We noted this is a fairly smooth signal, so in this case rather than the robust test using median and MAD, we use the more traditional |*x*-mean|/sd to test for anomalies.

Results

The second example (Atlas) has been implemented and replaces an earlier rule-based system. After running for several weeks, Atlas raised its first alert, which identified a real disruption that was not detected by the rule based system. It was subtle: it appeared that only 102 out of 3K queries were affected. But Atlas easily detected this disruption. 28 metrics had more than a 5-sigma deviation of the 0.9 quantile feature. In fact the median value of |*x* – mean|/sd over all 28 metrics was 85. This shows the power of well-constructed features.

Details on Atlas

To compute the surprise, we use robust regression. Also we use weighted regression, so that the most recent points count most. For robust regression we use the R package **rlm**. For weighting we use a function of the form $\mathrm{wt}(x) = 1/(\alpha\, e^{\beta x} + 1)$. We use 100 time points, and want to weight the 100-th point most heavily. We choose $\alpha = 148$ and $\beta = -0.1$ so that $\mathrm{wt}(50) \approx 0.5$, $\mathrm{wt}(72) \approx 0.9$, and $\mathrm{wt}(100) \approx 1$.

Surprise is defined as the deviation divided the median deviation. For a discrete signal that takes its median value more than 50% of the time, this is very sensitive to the regression algorithm. For algorithms like RANSAC (Fischler 1981) the result will be a horizontal line passing through the median, and so the median deviation will be 0. The **rlm** algorithm on the other hand will give a line with non-zero slope and so the median will be a small non-zero number, making the surprise potentially very unstable. To adjust for that, we actually compute the median of the *non-zero* deviations. For continuous signals this is virtually the same as the median of all deviations. For discrete signals we check for the case where the median is taken on more than half the time (actually 60%) and in that case take the regression line to be horizontal. Fortunately these case are rare.

If there are less than 3 nonzero deviations we arbitrarily set surprise to 100 times the deviance divided by the y-intercept of the robustly fitted line. If the intercept is 0, we use 100 times the deviation.

Discussion

In our view there are two types of alerting systems, both of which are important. The first type looks for disruptions that have occurred in the past and are likely to occur again. Such disruptions are probably best detected by a rule-based system and can potentially be automatically mitigated. The rules are detecting well-understood disruptions with known fixes. Unfortunately not every disruption is of this form. That's why we need the second type of alerting system, described in this report. They look for statistical anomalies. Unlike the first type, these require someone to track down the cause of the anomaly and determine if it is a genuine disruption.

This brings up the systems aspect of statistical anomaly detection. One of the alerts from the Atlas system described in the last section was caused by an update to the category demand table. This was not a disruption. On the other hand, the update did cause a large site change and it is a good sign that Atlas detected it. This suggests that statistical alerting should be part of a larger system that knows about site changes and other external events, and can suppress the alerts they cause. Related is the question of Christmas. Many people's reaction to statistical alerting is to ask how the system will learn about unusual shopping days like Christmas. Ideally signals will be archived for several years and so a feature using history would automatically accommodate a yearly pattern like Christmas. However, this is probably not possible for all signals, and so the best system design for this is not to tweak the features and statistics to handle this very special case, but rather to embed it into the larger system. Even if all signals had archived values going back several years, there would still be 1-off events like the British Royal wedding that would need to handled at a higher level.

One objection to our approach is that the selection of features is *ad hoc* and not generalizable. But we believe that just as there is no magic bullet that solves all machine learning problems, there is no amazing outlier detection algorithm that solves all alerting problems. Getting a low false positive rate is very difficult, and can only be done with domain knowledge. We argue the way to use the domain knowledge is in the development of features.

Another issue that comes up is multivariate anomalies. Specifically large changes in the relationship between signals that are not accompanied by similarly large changes in individual signals. Our view is that these are quite rare. They may be useful additions to later versions of Atlas, but their added benefit appears small.

Next Steps

Based on these examples, here's how alerting with a low false positive rate might be extended to the large set of signals monitored at eBay. First, each signal should have a history of at least 2 months (more is better), so that weekly trends can be observed. Second, there should be a list of known disruptions, so that the false positive rate of potential features can be estimated. Then building features could be done as follows:

1. An expert would identify a class of similar signals and devise a feature (or features) for them, much as the quantile feature was devised for the signals from the top query monitor.
2. The feature would be validated by looking at historical outages. The feature should detect all known outages with a minimum of false positives.
3. The properties of the signals that make the feature useful should be identified.

To be scalable, it should be easy to add new signals to the system. Ideally, step 3 above could be used to automatically decide what type of feature should be used when a new signal is added.

To make this concrete, here's a very simplistic example. Suppose there are only two features, the daily history and the weekly history of the signal. When a new signal is added, its historical values could be automatically examined to see which feature is best. For example the coefficient of variation CV (or perhaps better, the MAD divided by the median) could be computed for the signal sampled at 24-hour periods. And then computed for the signal sampled at 7-day periods. If the CV for weekly sampling is much smaller than the CV for daily sampling, then weekly would be a better feature than daily.

The most important lesson from Atlas and Muninn is this: features should be validated on historical disruptions. If none of the example features presented in this report can detect historical disruptions with a low false positive rate, new custom features need to be devised.

Summary

We argue that processing monitored signals into features is more important than devising ever more sophisticated statistical tests. Our experience with Atlas supports this position. A subtle disruption that was not detected by an earlier rule-based system was easily spotted by Atlas, even though it uses an extremely simple statistical test: $|x - \text{mean}|/\text{sd}$. In both our examples the features used historical values of the monitored signals, and this is likely to be important for most useful features.

Bibliography

Chandola, V., Banerjee, A., & Kumar, V. (2009). "Anomaly detection: A survey". *Computing Surveys* **41**(3).

Fischler, M. A., & Bolles, R. C. (1981). "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". *Communications of the ACM*, **24**(6).