Hortonworks
5470 Great America Pkwy
Santa Clara, CA 95054
(408) 916-4121

Making a Smarter NiFi

William Song
19th August 2016

OVERVIEW

Hortonworks DataFlow powered by Apache NiFi provides unique capabilities to generate index, and leverage provenance data to dramatically improve its users ability to manage complex information flows. Currently, provenance data enables dataflow managers to troubleshoot the data path, to prove what was received or delivered, and to click through the journey of content in the state machine that is Apache NiFi. However, this rich information can be further leveraged to provide the next level of dataflow capabilities. Using machine learning techniques to analyze provenance data we can alert users to anomalies, reduce latency times, and generate processor recommendations.

With the scale and volatility of the Internet of Things, anomaly detection can alert dataflow managers to incoming breakdowns or hidden failures. This paper will focus on key algorithms and approaches that were studied. Then it will recommend future techniques and added features to study.

GOALS

1. To demonstrate that provenance data can be further leveraged to make a smarter NiFi with easy, out-of-the-box tools
2. To evaluate features, algorithms, and approaches appropriate for anomaly detection
3. Suggest paths for future exploration

UNSUPERVISED ANOMALY DETECTION SPECIFICATION

The detection of abnormal instances of data has evolved from simple outlier cleansing to broad industry applications such as fraud detection, cyber defense, and medical monitoring. Anomalies themselves may signal events of special interest within a dataset. An anomaly, or outlier, is

defined to be “an observation that appears to deviate markedly from other observations in the sample.” In addition, anomalies should be rare in a dataset compared to typical data points.

Unsupervised anomaly detection is used for datasets where events are not labelled as normal or anomalous. In practice, most anomaly detection problems are unsupervised as it is time-consuming and expensive to manually label anomalies in a training set. Unsupervised anomaly detection approaches rely purely on the distances or densities of data points to determine outliers.

Different algorithms may be more sensitive to certain types of anomalies. Figure 1 portrays a sample dataset with several types of anomalies. The green data points represent normal clusters of data. *Global anomalies*, such as x1 and x2, are clear anomalies within the context of all the data. *Local anomalies*, such as x3, are anomalies with the context of its local cluster of data. Lastly, it is ambiguous on whether cluster c3 should be classified as a cluster of regular data points or a cluster of anomalous data points. These points are referred to as a *microcluster* [1].

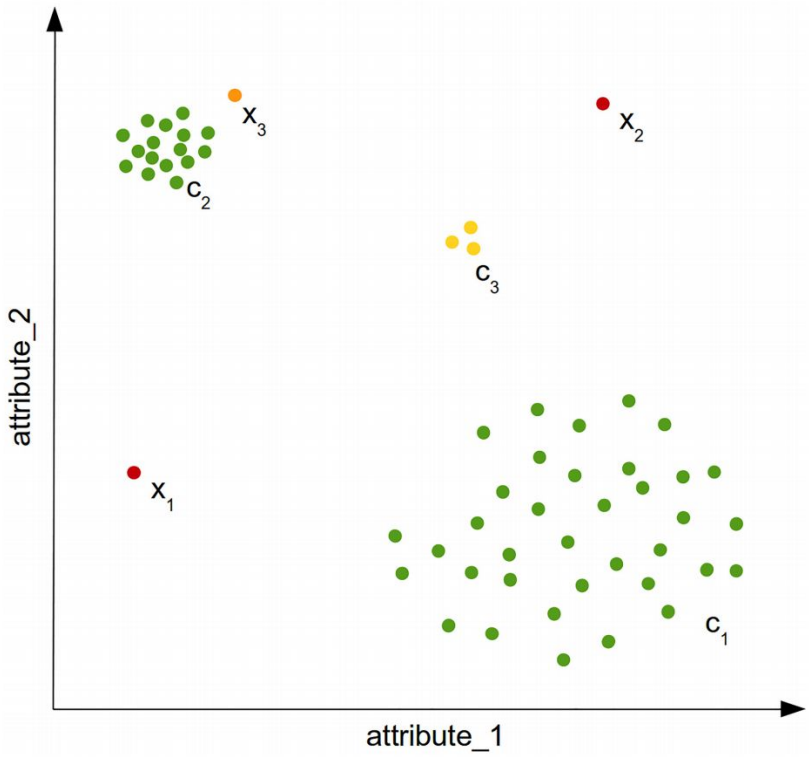


Fig 1. This example dataset illustrates normal data (green), global anomalies (red), local anomalies (orange), and ambiguous microclusters (yellow).
Source: [1]

As a result of relying purely on the data structure, unsupervised anomaly detection algorithms output a “anomaly score,” which measures the likelihood of an instance being an anomaly. This stands in contrast to many machine learning algorithms that will output a label or classification of data points. It is now up to the dataflow manager to define an anomaly detection threshold that will classify a datapoint as an anomaly. In an attempt to make these approaches require minimum user input, techniques to automatically calculate a threshold are explored later in this paper.

FEATURE SELECTION

An initial investigation of the features of the data is essential to an accurate model. The most relevant features should be chosen for the model and usually require domain knowledge of the application.

Provenance Data Reporting

Provenance data was gathered with the Site-to-Site Provenance Reporting Task that reported to the same nifi instance. The data was then saved to the local disk with a PutFile processor. Since the reporting of the provenance data itself would appear as part of the provenance record, data preprocessing was necessary to remove these events.

Feature Analysis

Most features within the provenance data were unusable for machine learning models. Primarily, many features (such as “eventID”, “contentURI”, “lineage identifier”) were simply unique identifiers and would not provide meaningful information for the models. In addition, features that kept track of FlowFile attributes were also too unique to provide consistent, reliable information as well as vary widely from flow to flow..

Some potentially useful features were omitted in an effort to avoid creating too much noise as well as avoid “the curse of dimensionality”[2]. The timestamp features could provide insights on seasonality fluctuations, but this would require domain knowledge of the specific application of the flow (such as whether to bucket times as day/night or by hours of the day). In addition, the “processor type” feature could have been utilized but was found to be redundant when used with the “component ID” feature. Both features provide information on which processor a provenance event occurred in, but the “component ID” feature differentiates between two processors of the same type.

Ultimately, the features settled on are shown in the table below

eventType	durationMillis	lineageDuration	componentId	entitySize
CREATE	-1	0	c8f8b9a0-c8a1-4f9c-9f0d-9b2b809454d8	0
CONTENT_MODIFIED	0	6	e73b710c-59e2-4e78-9586-ade150d14a82	48909
FORK	-1	16	d55f4dec-eab0-4877-aded-0315313bf6cf	48909
ROUTE	-1	19	d55f4dec-eab0-4877-aded-0315313bf6cf	165
DROP	-1	20	d55f4dec-eab0-4877-aded-0315313bf6cf	48909
DROP	-1	20	d55f4dec-eab0-4877-aded-0315313bf6cf	165
RECEIVE	2855	180	7074ebf4-2368-484a-b729-78b8877f496d	48917

This is a truncated version of an example dataset.

The lineage duration is calculated by finding the difference between the “timestampMillis” feature and the “lineageStart” feature. Categorical features were encoded as integer features using a one-of-k scheme.

ALGORITHMS

The two most widely used categories of algorithms are *nearest-neighbor* based algorithms and *clustering* based algorithms. Other algorithms such as statistical algorithms (histogram-based [3]) and classifier algorithms (one-class SVM [4]) were not tested but could provide promise with future testing. All scripts were written in python and was supported heavily by Scikit-Learn library.

The following section will be an analysis of the applied algorithms with a simple simulated dataset. The dataset was generated by a three-processor flow shown in Fig 2. There was then a *1/1000* chance the “executeScript” processor randomly delayed its run time *20-50* times longer than regular execute times (normal execution times took *0-1* milliseconds). These delayed events were then given an attribute that labeled them as an anomaly. There are *371163* provenance events with *9* features in the dataset and *1168* anomalous events (*.3%* of the total dataset).

Nearest-Neighbor-Based Algorithms

As the name implies, nearest neighbor algorithms find the k-nearest-neighbors of each data point. The algorithm then assigns an anomaly score based on the distances to the k-nearest-neighbors. As previously mentioned, it’s not clearly defined on what range of scores

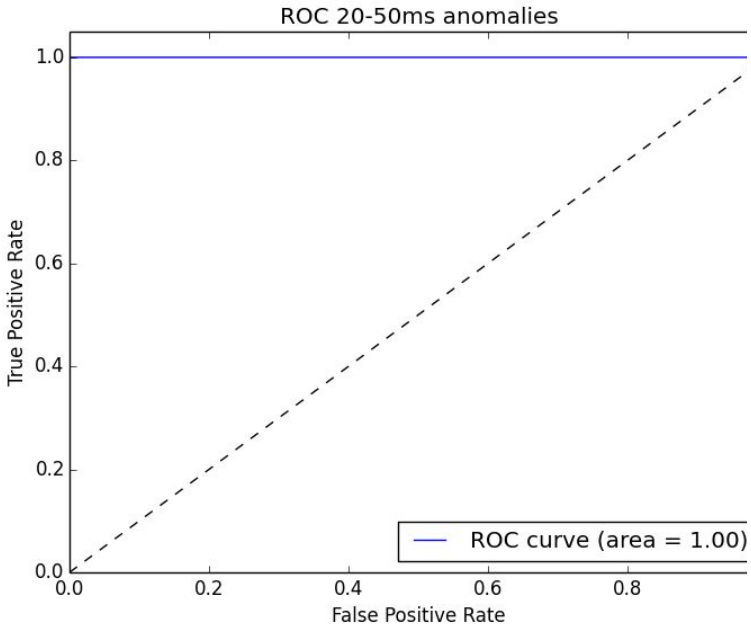
should classify a datapoint as an anomaly and the absolute value of scores can vary widely from application to application.

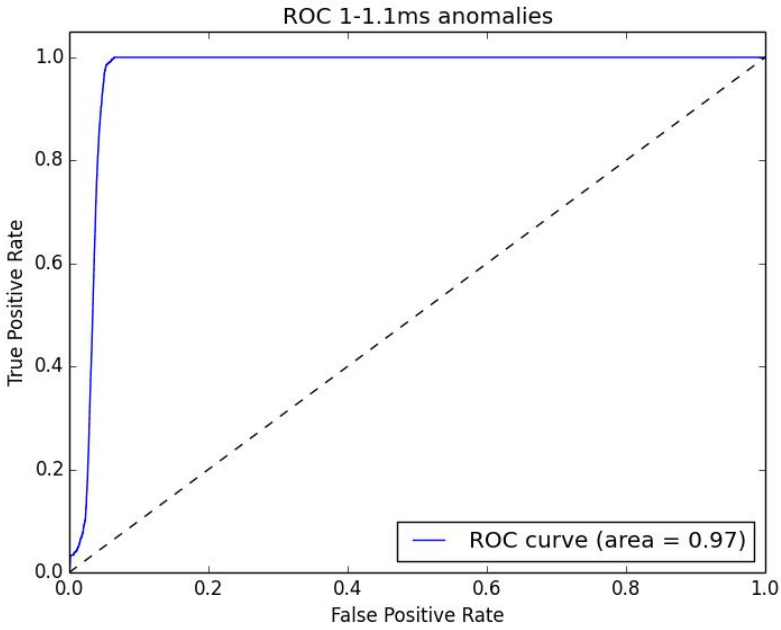
The user-inputted value of *k* can also affect anomaly scores. In general, *k* should be in the range of *10 < k < 50* [1]. In supervised learning, *k* can be determined with cross-validation and test sets, but that is not possible with unsupervised learning due to unlabeled data.

With unsupervised anomaly detection, a simple accuracy value is not sufficient to evaluate a model. Since the data is heavily skewed, a one-line script that simply classified all data as "regular" would still be *99.7%* accurate in this application. In addition, since anomaly scores are assigned rather than a direct classification, varying thresholds must be tested. Therefore, a receiver operator characteristic curve (ROC) was generated for each test. The area under the curve (AUC) was then used to evaluate the model. An AUC score of .9-1 is considered excellent while a score of .5 or less is considered worthless[5].

k-Nearest-Neighbor (k-NN) Global Anomaly Detection

This algorithm is designed to detect global anomalies. However, the algorithm also had success with detecting local anomalies as well. The dataset was modified so that anomalies only took on average *1-1.1ms* (in comparison to *0-1 ms* for normal data*)* to execute in order to simulate local anomalies. Figure 2 shows the ROC curves for k-NN's performance on both sets.





The model performed excellently on both datasets with an AUC of virtually *1.0* and *.97* for the global anomaly set and local anomaly set respectively. The parameter *k = 30.*

K-nn searches runs in roughly O(*n*^2) time complexity where n is the number of events. A binary tree data structure was used to more efficiently search for nearest neighbors and reduce search

complexity to O(log *n*). The model took on average *6s* to index the tree and then on average *38s* to calculate all anomaly scores for a total run time of about *44s* on average.

The hypothesized reason why k-NN global anomaly detection still captures local anomalies well is that provenance data tends to be densely clustered and contains many duplicate events. Therefore, slight deviations from these tight clusters can even be detected as global anomalies. However more investigation should be conducted to confirm.

Local Outlier Factor (LOF)

Local Outlier Factor [6] is a k-NN variant designed to detect local anomalies. The algorithm essentially compares the local density of a datapoint with the local density of its k-nearest-neighbors. This algorithm can detect global anomalies as well as global anomalies will also have low local densities compared to its k-nearest-neighbors. In addition, the absolute value of the anomaly scores outputted by LOF do not vary widely from application to application, providing a clearer threshold on what data points should be considered an anomaly.

Unfortunately, brute force LOF runs in O(n^2) and can not efficiently handle datasets with lots of duplicate data points. Duplicate data points dramatically increase the run time of LOF and can create undefined results [7]. The current python implementation of LOF by Damjan Kužnar originally would have taken 300 days to run on the simulated test set not considering the duplicate data. After converting the code to a *numpy* implementation rather than a pure python implementation, the model still would have taken 3 days. Since these are not practical run times, no results could be reported at this time.

Clustering-based Algorithms

Clustering based anomaly detection algorithms first use a clustering algorithm to cluster together data points and find centroids. Anomaly scores are then assigned based on the distance of a datapoint to its centroid. The most common clustering algorithm used is k-means due to its scalability and linear time complexity. Figure 3 is a visualization of the algorithm.

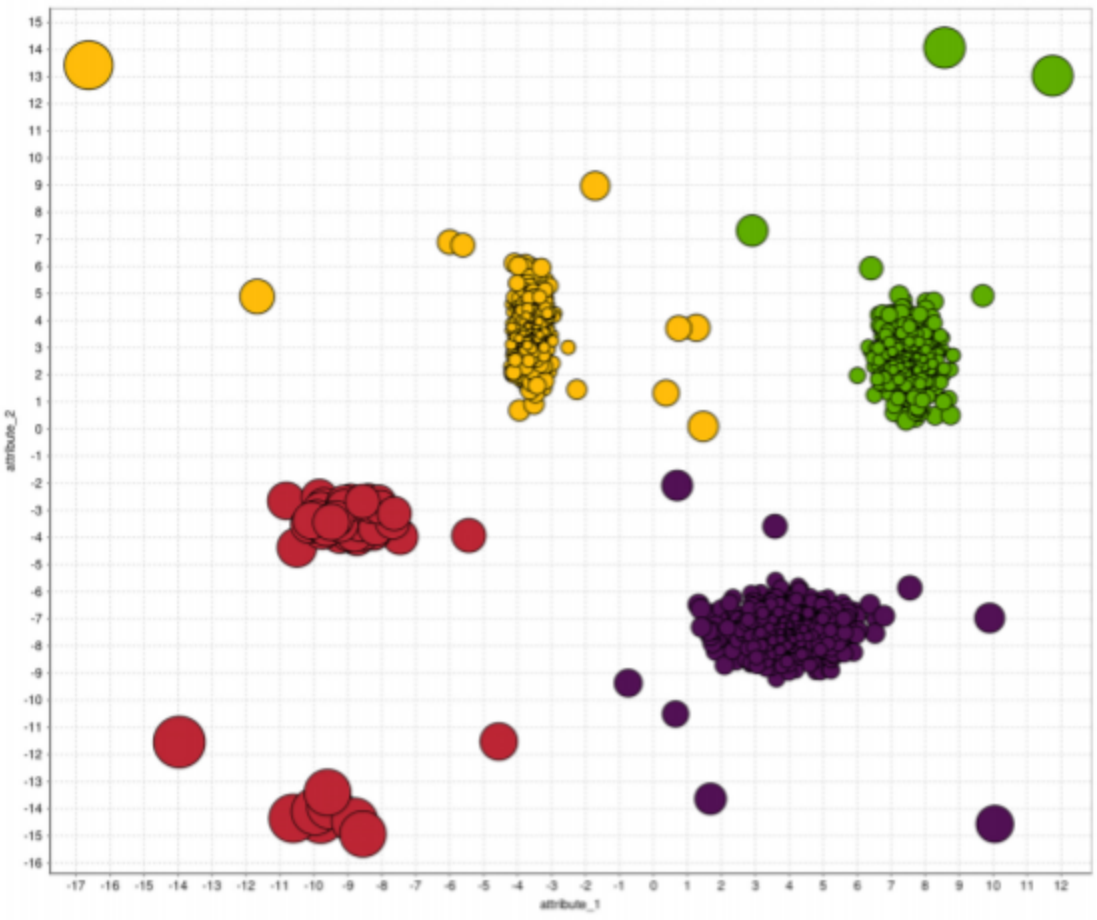


Fig 3. Data points are colored by their assigned clusters and bubble sizes are proportional to anomaly scores. Source: [1]

The biggest disadvantage of clustering-based algorithms is their high sensitivity to the *k* number of centroids parameter. Small changes in *k* can lead to dramatically different results. This is in stark contrast to nearest-neighbor based algorithms which will perform generally the same as long as *10 < k-number-of-neighbors < 50*. However, an investigation of the provenance data showed that data was essentially clustering based on which processor the event occurred in. Figure 4 visualizes data from other sample flows.

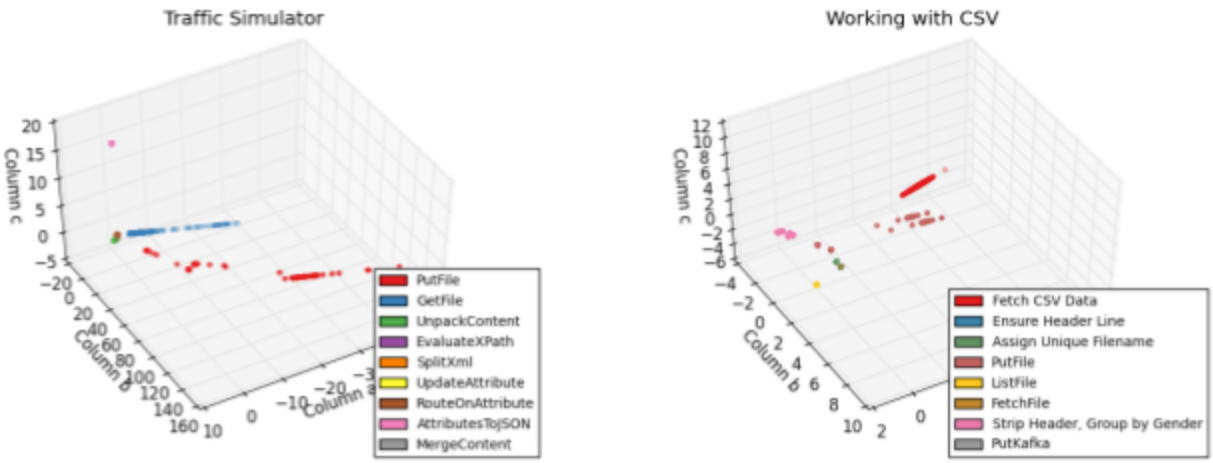


Fig 4. A 3-dimensional visualization of provenance data from sample NiFi flows from the community page. Data points were colored based on processor ID.

This domain knowledge of the provenance data allowed k-number-of-centroids to simply be initialized to the number of unique processors on the flow.

Clustering-based Global Anomaly Detection

For this application an actual anomaly score threshold was defined, so precision and recall was used to evaluate the model rather than ROC plots. Having a defined threshold makes the models even more “out-of-the-box.”

The first step was to cluster the data running k-means clustering with an euclidean distance metric on the data set. The anomaly score threshold is then determined by iteratively calculating distance between each data point and it’s *centroid furthest from the respective data point* (this is not the centroid the datapoint is assigned to). The average of all the distances of these data points is the threshold. Any data point with an anomaly score above the threshold was labeled as an anomaly.

With the simulated dataset, the model got perfect results of *1.0* precision as well as *1.0* recall. The run time was on average 26 seconds.

FUTURE EXPLORATIONS

The techniques studied in this whitepaper only scratch the surface of the possibilities of leveraging NiFi provenance capabilities. The following section will go into promising algorithms and other recommendations for future investigations.

Additional Algorithms

One-class Support Vector Machine

The one-class Support Vector Machine (implementation) is a semi-supervised learning algorithm for anomaly detection[9]. The algorithm simply classifies if a new datapoint is different or similar to the training set. The model assumes that the training set has no anomalies within it. Since one cannot assume that an existing provenance record is clean of anomalous events, this algorithm was not deeply investigated. Nonetheless the algorithm may still be worth pursuing in the future as it is effective with high-dimensional data and can still detect anomalies not within the training set.

K-modes/K-prototypes

The k-modes algorithm (implementation) is used for clustering categorical data[8]. The k-prototypes algorithm is a variation of k-means that can handle a mix of categorical and continuous data. Since NiFi provenance data has many categorical features, this seemed at first a good clustering algorithm for clustering-based anomaly detection. However, once it was found that most of NiFi's categorical features were unique ID's, it was found that k-modes was not as applicable as first thought. If in the future provenance data contains more categorical features, then this algorithm may be revisited.

Optimizing Local Outlier Factor/FADD/G-FADD

The current implementation of Local Outlier Factor could be further optimized with some data structures like K-D tree [10] or Ball tree [11]. Although these indexing structure will improve search speeds, they could only provide marginal benefits as the biggest bottleneck is the large amount of duplicate data within provenance records. FADD and G-FADD are algorithms that directly address the issue of duplicate data and should be further investigated[7].

General Recommendations

Adding additional numerical/continuous provenance features could help improve the robustness of all of these algorithms. Currently there are only 3 numerical features in which the model can detect anomalies. Additional features such as "previous processor ID's" could provide more opportunities for these models to detect anomalies.

All processors should report "eventDuration" as that is an important feature in determining anomalies.

CONCLUSIONS

An investigation of leveraging provenance data with machine learning techniques has been conducted for the first time within the NiFi community. The findings suggest very promising results for implementations of anomaly detection within NiFi. A foundation has been laid for which future engineers can now refer to for further pursuits.

It was found that both k-means-based anomaly detection and nearest-neighbor global anomaly detection were effective at detecting anomalies. The research community generally favors nearest-neighbor based algorithms for anomaly detection due to the number-of-centroids sensitivity of cluster-based algorithms. However, with the discovered domain knowledge of provenance data, the most effective number of centroids can now be automatically determined. Therefore, clustering-based algorithms may be best for NiFi due to its scalability and faster run times.

Finally, it is recommended that additional, useable features are added to provenance data so that models have more criteria to determine events as anomalous.

REFERENCES

1. Goldstein M, Uchida S (2016) A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. PLoS ONE 11(4): e0152173. doi:10.1371/journal.Pone.0152173
2. Radovanović, Miloš; Nanopoulos, Alexandros; Ivanović, Mirjana (2010). "Hubs in space: Popular nearest neighbors in high-dimensional data". Journal of Machine Learning Research
3. Goldstein M, Dengel A. Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm. In: Wölfl S, editor. KI-2012: Poster and Demo Track. Online; 2012. p. 59–63.
4. "2.7. Novelty and Outlier Detection." 2.7. Novelty and Outlier Detection — Scikit-learn 0.17.1 Documentation. Scikit-learn, n.d. Web. 16 Aug. 2016.
5. Tape, Thomas G. "The Area Under an ROC Curve." The Area Under an ROC Curve. University of Nebraska Medical Center, n.d. Web. 17 Aug. 2016. <http://gim.unmc.edu/dxtests/roc3.htm>.
6. Breunig MM, Kriegel HP, Ng RT, Sander J. LOF: Identifying Density-Based Local Outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, USA: ACM Press; 2000. p. 93–104.
7. Lee, Jay-Yoon, U. Kang, Danai Koutra, and Christos Faloutsos. Fast Anomaly Discovery given Duplicates. Thesis. School of Computer Science Carnegie Mellon University, 2012. N.p.: n.p., n.d. Print.

8. Huang, Z.: Clustering large data sets with mixed numeric and categorical values, Proceedings of the First Pacific Asia Knowledge Discovery and Data Mining Conference, Singapore, pp. 21-34, 1997.
9. Schölkopf, Bernhard, et al. Estimating the support of a high-dimensional distribution Neural computation 13.7 (2001): 1443-1471.
10. “Multidimensional binary search trees used for associative searching”, Bentley, J.L., Communications of the ACM (1975)
11. “Five balltree construction algorithms”, Omohundro, S.M., International Computer Science Institute Technical Report (1989)