Anomaly Detection in the Oracle Database Ecosystem Using Density Based Spatial Clustering

A technical paper prepared for presentation at SCTE TechExpo24

Jim Prather
Data Scientist - Platform Data Analytics Team
Cox Communications
Jim.prather@cox.com

Debanjali Battacharya, Genpact

Table of Contents

List of Figures

List of Tables

1. Introduction

In any large modern company, data has become the lifeblood of the organization and databases have become the beating hearts which supply that vital resource to every aspect of the company. Given that the failure of a single database, even for a short amount of time, can potentially lead to hundreds of thousands of dollars in lost revenue, it has become imperative to ensure complete reliability of every database within the ecosystem.

With hundreds to thousands of databases needing to be monitored, it has become increasingly difficult for Database Administrators (DBAs) to maintain adequate vigilance on every single database using standard monitoring techniques. Recently, companies have been turning to Machine Learning algorithms to "study" each database, determine if a database is displaying signs of distress, and then alert a DBA that action may be required on a given database.

One of the newest and most promising algorithms in use at Cox Communications is Density Based Spatial Clustering (DBScan). Fundamentally, the DBScan algorithm looks at groups of points which lie closer together (i.e. have a higher spatial density) and then assigns them to be in the same cluster. The process repeats until every data point has been assigned to a cluster, or else has been labelled an outlier. It is these outliers, or anomalies, which may be harbingers of database problems.

Each night eight of the most important metrics, in five-minute increments, over the past thirty days of data are fed into the ML algorithm for each database. By using Principal Component Analysis, the data is converted from an eight-dimensional manifold to a three-dimensional surface and then used to create one DBScan model per database. Given the trained model, whenever a new datapoint arrives, it is simply compared to the data in the pre-trained model to determine if the datapoint is "normal", or if it is an anomaly which should be investigated further.

By operationalizing DBScan ML techniques on database monitoring data, database alerts have been accelerated by 15 minutes over existing monitors and decreased false positive alerts by a factor of six.

2. Clustering Algorithms

In data science, there are two primary categories of algorithms – supervised learning and unsupervised learning. In supervised learning, you are given the "correct answer" and are attempting to train a model to match the predicted answer to the correct answer. In unsupervised learning, you do not have the "correct answer", so the challenge is to extract patterns and structure from the data itself without any human interaction.

One of the most common categories of unsupervised learning is cluster analysis. This type of analysis attempts to group similar objects into different sets, called clusters. These groupings may be defined by connectivity to nearby points (Hierarchical Clustering), distance to a cluster centroid (K-means Clustering), correlation and dependences between data points (Gaussian Mixture Model) or grouping areas of higher data point density (DBScan).

3. DBSCAN Clustering

While every clustering algorithm has its own strengths and weaknesses, the DBScan algorithm is particularly well suited to anomaly detection. To begin with, it does not require the number of clusters to be pre-defined. Furthermore, it does not need the data to be regularly shaped, and most importantly, it works well when the data contains noise. This algorithm will automatically create the clusters and will treat the noise as outliers. It is this last feature which is of greatest interest when creating an anomaly detection algorithm.

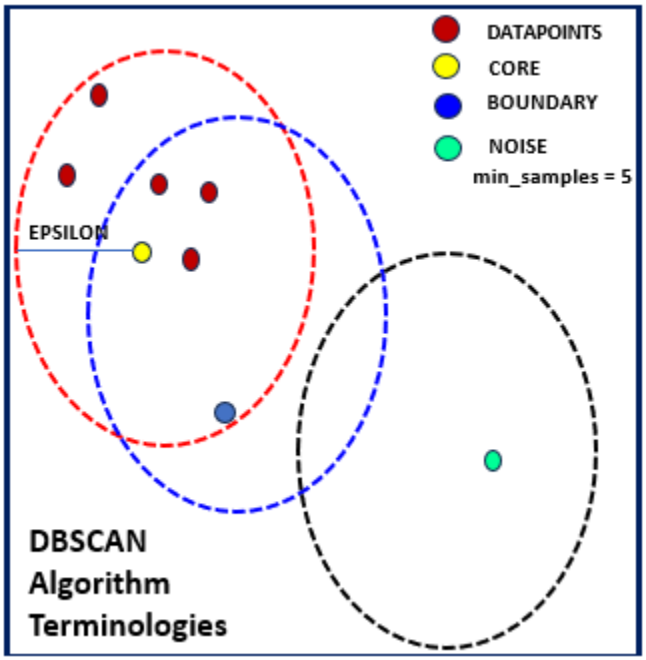

Figure 1: DBScan Cluster Definition Process

The DBScan algorithm requires two parameters to be defined: Epsilon (ε) and the MinPts. Given those parameters, the algorithm selects a datapoint and considers a circle around that datapoint with a radius of ε. It then considers all the additional datapoints which fall within that circle. If the number of datapoints are greater than or equal to the minimum number of datapoints as defined by the MinPts parameter, then all those datapoints are assigned to the same cluster. Once the initial cluster is defined, then all those datapoints become the centers of their own circles, each with radius of ε, and the process is repeated. All datapoints in those new circles which meet the required criteria are added to the initial cluster. That process is repeated until no more data points can be added to the original cluster.

Once no more points can be added, a new, unlabeled point is randomly selected to become the starting point of a new cluster and the process is repeated. Once all the clusters have been defined, then any unlabeled data points are defined as outliers. Finally, all the data points which have been labelled as boundary points are checked to see if they have been assigned to the best possible cluster. Any points which should be assigned differently are placed in the appropriate cluster.

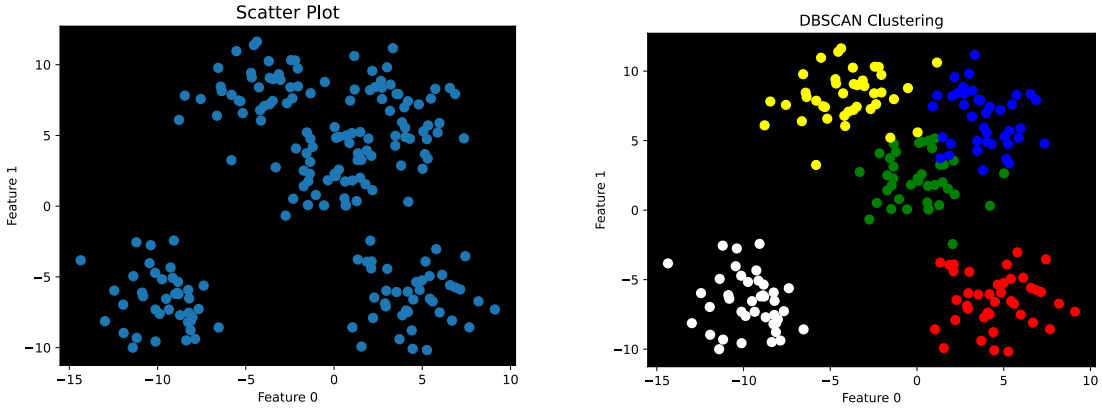

Figure 2: DBScan Clustering Output - five clusters with no outliers

4. Productionalizing Model Training and Creation

Within the Oracle database ecosystem, the collection of dozens of performance metrics from nearly one hundred seventy-five production databases, and more than three hundred database instances, has been automated. That data is loaded to a centralized repository every five minutes. While so much data is a boon for analytics, it is far too much to be analyzed using manual methods or even using classical analytic methods in a real time fashion. To take full advantage of this wealth of data, ML models were needed which could learn what "normal" looked like for each database and could warn the DBAs when a database was beginning to experience abnormal behavior.

4.1. Data Normalization and Resampling

Before the data can be used in any clustering model some minor feature engineering needs to occur. While this data is collected at 5-minute intervals, the date fields in the collected data are timestamps which are measured to the millisecond. Such precision makes the clustering algorithm overly laborious, so the data is resampled to be at the five-minute grain. Furthermore, as is common with all clustering algorithms, it is important that all the metrics are standardized to be on the same scale. Without such normalization, the largest metric will dominate all distance calculations and important variations in smaller metrics could be lost. In our case, all metrics were normalized to fall on a scale of 0 to 1 so that each metric would contribute a similar amount to the distance calculations used by the DBScan algorithm.

4.2. Principal Component Axes

With the data resampled and normalized, it became imperative to remove some of the dimensions of the data. A problem which is common in many machine learning algorithms, and particularly troublesome in clustering algorithms, is the "curse of dimensionality". This moniker refers to the inability of a model to identify patterns due to the high number of predictive features creating a sparse feature space. Every new feature exponentially increases the possible number of buckets in which a datapoint may reside. It does not take many features before there are vastly more "empty buckets" than there are ones containing a datapoint.

To illustrate this concept, consider the images below.

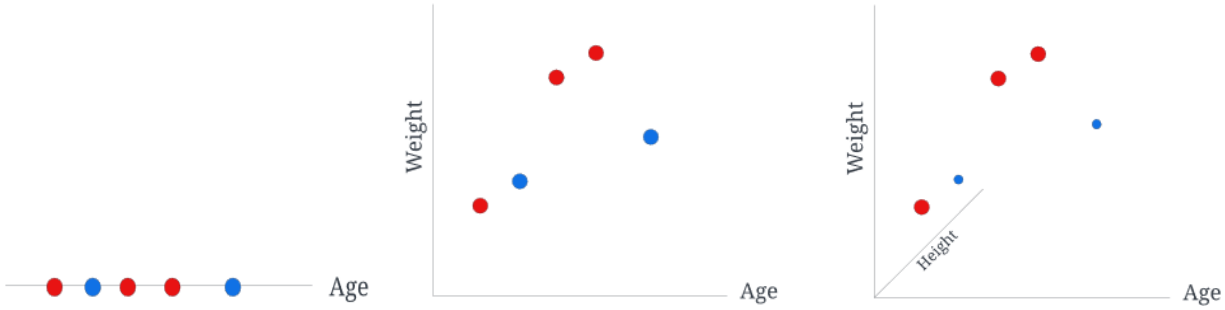

Figure 3: Predicting Heart Disease – red means positive for heart disease

Assume you are trying to predict heart disease. At first, you are only considering age as a predictive variable. The image on the left makes it clear that this is obviously not enough to make an accurate prediction. Now suppose you add a second variable, weight, to the model. It is immediately obvious that the data becomes far more spread out. Adding a third variable causes the data to be spaced even further apart. This simple example should be enough to show that more dimensions being added to the model further spreads the datapoints apart. While for some models this separation is useful, for a model based on clustering, especially one based on cluster density, higher dimensional data is a liability not an asset.

Fortunately, there is a standard practice for reducing dimensionality while still retaining most of the information encoded in the data – Principal Component Analysis (PCA). Just as any vector in a 2-dimensional space can be projected onto a rotated coordinate system and decomposed into new x and y components, any vector in higher dimensional space may be projected onto modified dimensions and decomposed into new principal components. By projecting the datapoints onto the eigenvectors of the covariance matrix, PCA makes it possible to reduce the number of dimensions while choosing the coordinates which retain the greatest amount of information from the original data.

Included in the Cox process requirements was a mandate to avoid unexplainable, or “black box”, code wherever possible. Given such a direction, projecting the original dimensions onto a three-dimensional surface was the logical choice. This choice reduced dimensionality and provided the greatest explanatory power while still allowing for visualization of the resultant data.

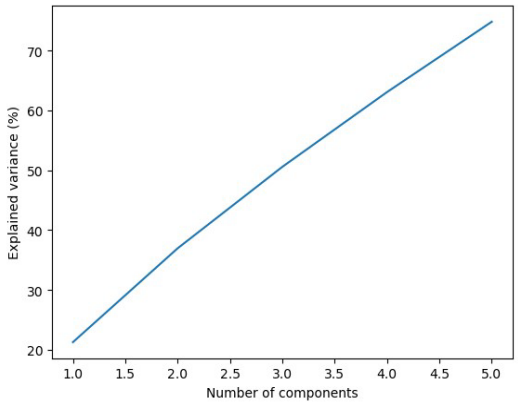

Figure 4: Plot of PCA components vs. explained variance

As is obvious from Figure 4 using 3 principal components still retains better than 50% of the variance in the data while reducing the dimensionality of the original data by nearly 2/3. Furthermore, when you look at the scatterplot in Figure 5 below, the data is easily understandable and potential outliers are visually obvious.

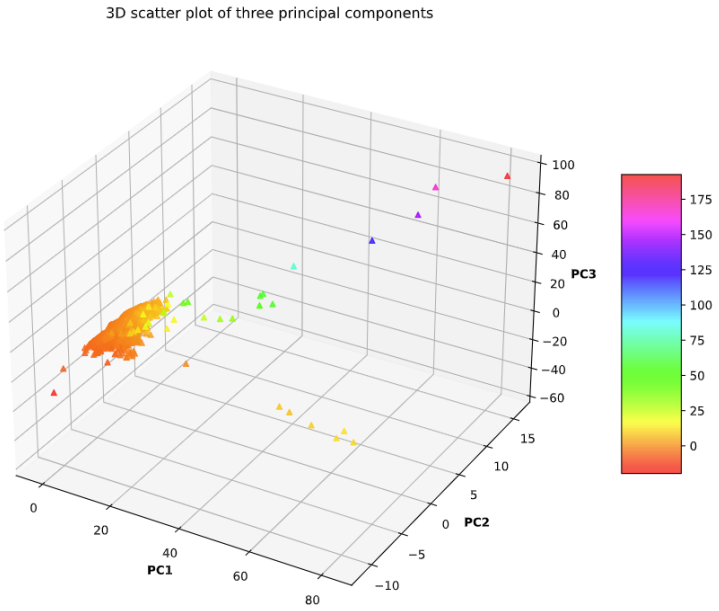

Figure 5: Scatterplot of database metrics along three principal component axes

4.3. Model Training and Creation

With the data engineering completed and the dimensionality reduced, it was time to focus attention on training the clustering model. Since there are nearly one hundred seventy-five databases, and each database has its own unique clustering "fingerprint", it was necessary to train and store a separate model for each database which could then be used to determine whether subsequent datapoints are anomalous.

To train a DBScan model, the two parameters mentioned previously, ε and the MinPts, were required. Since these values may vary from one database to another, they were placed in a parameter file with a separate set of parameters for each database. Since databases are highly overengineered, the expectation was that there would be very few anomalies worth considering. To this end, the default parameters should be set to have as many datapoints as possible be identified as members of a cluster. In an effort to achieve this, the original parameters were set as

$$\varepsilon = .95$$

$$\text{MinPts} = 4.$$

While the MinPts parameter for a few of the databases has been changed to

$$\text{MinPts} = 3,$$

the

$$\varepsilon = .95$$

parameter has proven to be the appropriate distance choice universally.

For each database, ninety days of data were processed through the DBScan algorithm from the python sklearn package to create a unique clustering model. That model was saved as a .pkl file with the name and location of the file stored along with the ε and the MinPts values in the parameter file.

One of the greatest benefits of this process was the speed at which a model may be created. The DBScan algorithm only requires 5 to 10 seconds to train and create a new clustering model. When there are nearly one hundred seventy-five such models needed, this speed has been necessary to create a production grade process.

See Figure 6 below for a sample scatterplot of a database with a normal cluster and multiple outliers.

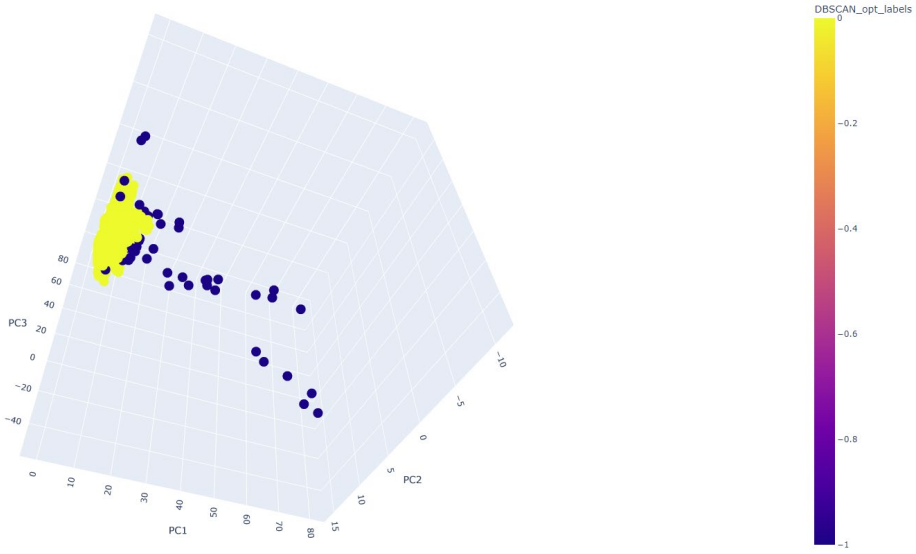

Figure 6: 3-D Scatterplot of database metrics showing cluster (yellow) and outliers (blue)

4.4. Testing for New Anomalies

Testing new datapoints is, in principle, a straightforward process – load the saved model into memory, process the new datapoints through the model, and review the label assigned to each datapoint. When the process must be applied to 175 databases, however, an additional layer of complexity is added. Even so, the process is straightforward to describe. First loop over all the databases. For each database, select the new datapoints which have been created since the previous run, then load the associated

```
*.pkl
```

file containing the stored model, and finally process the new datapoints through the model. Once the labels for the new datapoints have been generated, store any anomalies in a table for future use in alerting.

4.5. Scheduling

As anyone who regularly works with databases can attest, database performance may change abruptly. The application of a patch, introduction of a new batch job, or even presence of a new query may adversely impact the behavior of a database. Many types of anomaly detection models, such as timeseries based models, may take days or even weeks to adjust to profile changes in a system. Such slow response times can lead to excessive false positive anomaly alerts.

To avoid such excessive alerting, the DBScan models for all the databases are retrained nightly starting at midnight. This schedule allows the models to adapt to all new changes for a database profile in less than 24-hours, so even the largest changes to a database behavior will be rapidly accounted for.

5. Anomaly Severity, Database Health and Alerting Algorithm

5.1. Defining an Anomaly's Severity

To define an anomaly's severity, it was necessary to find a baseline value against which it could be compared. Since there are no inherent "zero values" which define anomaly severity, the decision was made to use the distance from the cluster centroid as the baseline from which to define an anomaly's severity. Given this decision, it was first necessary to define an anomaly's distance from the centroid of the cluster of normal datapoints. If the anomaly coordinates are defined as x_i and the centroid coordinates as y_i, then the Euclidean distance between the anomaly and the centroid is:

$$d = \sqrt[2]{\sum_i (x_i - y_i)^2}$$

Of course, a simple distance metric does not give any information regarding the magnitude of severity. For some databases, an anomaly distance of 0.5 might be nearly normal while for others a distance of 0.5 is an extreme anomaly. To overcome this issue, it is necessary to define a relative scale for each database. For this scale, two fixed points are required. The cluster centroid has already been defined as 0 on the scale, so all that remains is to define the upper end of the scale. Given that there is no absolute number which will suffice for all databases, the upper bound of the severity scale has been defined as the distance of the most severe anomaly for a given database for that day.

Therefore, if d_n is the anomaly furthest from the centroid cluster, for an anomaly d_i the severity calculation becomes:

$$anomaly\ severity = \frac{d_i}{d_n}$$

5.2. Database Health Metric

With the anomaly severity metric defined, it becomes a simple matter to calculate a database health score (DHS). Given a sorted list of n anomalies with distances d_1 …., d_n where d_1 is the smallest distance and d_n is the largest distance, calculating a health score is as simple as taking the average anomaly severity for the current measurement period. Since the desire is for a health metric rather than a severity metric, however, the calculation below is used to get the required value (1 – average anomaly severity).

$$Database\ Health\ Score = 1 - \frac{\frac{1}{(n-1)}\sum_{i=1}^{n-1} d_i}{d_n}$$

5.3. Alerting Algorithm

While having the DHS defined is excellent progress, it is not sufficient to allow for an automated alerting system. As mentioned earlier, the parameters used in the DBScan model are optimized to place as many data points as possible in clusters. Even so, with measurements every five minutes on nearly 175 databases, there are always a few anomalies every measurement period. Since it did not make logical, or

logistical, sense to send alerts on dozens of databases every single day, an additional layer of logic needed to be applied to filter out some of the extraneous noise.

Based on work from previous alert analysis, it was observed that the "one-off" anomalies seldom signify overall system problems. When system issues begin to occur, anomalies would begin to appear in clusters. Using this knowledge, the additional requirement was added that a database must have at least three anomalies within the past 15 minutes before an alert is generated. Once that additional threshold has been reached then an email would be generated listing the anomalies, showing a 3-D chart plotting the daily datapoints and anomalies, and providing a link to the diagnostic tools for the specific database generating the alert.

Finally, the DHS is used to create a severity measure. While the actual thresholds may vary from database to database, the default health measures are 0 to 20% health = high severity, 21% to 50% health = moderate severity, 51% to 100% health = low severity. While any cluster of 3 or more anomalies in a 15-minute window will generate an alert, the low and moderate severity alerts are considered informational while the high severity alerts are calls to action for the DBAs.

6. Results

The results shown below have been collected over a period of three months from 3/17/2024 to 6/17/2024. The metrics are a comparison of the alerts generated by the DBScan model and the alerts generated by the current "gold standard" timeseries-based alert model. Since the two models displayed alert on different cycles, any alert for a given database is counted as a single instance. For example, regardless of whether a given database generated twenty alerts in a day or a single alert in a day, it is simply counted as a single positive count. This decision removes the problem of the relative scale of the alerting systems being fundamentally different.

Table 1: Comparison of DBScan Alert Algorithm to Timeseries Algorithm

	Distinct DBs Alerted	Distinct Severe Alerts	Avg Daily DBs Flagged	Relative Response Time
Timeseries Alerting	75	75	0.81	+16:32 minutes
DBScan Alerting	108	12	1.16	-16:32 minutes

The results above show an interesting pattern in the two alerting models. The DBScan model is more sensitive than the timeseries-based model overall. This sensitivity causes roughly 50% more alerts to be generated overall and alerts to be generated on more databases in any given day. In other words, the model is significantly noisier. That same sensitivity, however, also allows for detection of potential issues over 16 minutes earlier than the timeseries-based anomaly model. When the DHS threshold is applied, however, the DBScan alerting model becomes less noisy than the timeseries based model by a factor of 6.

The takeaway is that, with the implementation of the Health Score in the DBScan model, the process becomes simultaneously more sensitive to potential issues arising in the system while becoming significantly less noisy for the DBAs monitoring the overall database ecosystem.

7. Conclusion

No alerting system can completely replace the expertise of human experience, but the size and complexity of modern IT systems are requiring an ever-growing reliance on automated monitoring. The DBScan-based alerting algorithm employed within the Cox ecosystem has proven to be a distinct success. It has improved the support teams' response time when monitoring hundreds of different databases by providing earlier warnings and more reliable information. In fact, as of the publishing of this paper, the process has been so successful that it has been expanded to monitor over 85 SQL Server databases and more than 50 MySQL databases as well.

While undeniably useful in the Cox database ecosystem, the true strength of the DBScan algorithm is its data agnostic nature. So many alerting systems are custom designed to fit one specific type of system and only a few distinct types of metrics. This clustering process can be applied to any system producing numeric metrics on a regular cadence. It is the authors' hope that this system may be extended to other IT systems outside the database ecosystem in the foreseeable future.

Abbreviations

DBA	database administrator
DBScan	density based spectral clustering with anomalies
DHS	database health score
IT	information technology
MinPts	minimum points
ML	machine learning
PCA	principal component analysis
.pkl	pickle file extension
sklearn	Scikit Learn Python package
SQL	structured query language

Bibliography & References

D. Deng, "DBSCAN Clustering Algorithm Based on Density," 2020 7th International Forum on Electrical Engineering and Automation (IFEEA), Hefei, China, 2020, pp. 949-953, doi: 10.1109/IFEEA51475.2020.00199.,

D. Deng, "Research on Anomaly Detection Method Based on DBSCAN Clustering Algorithm," 2020 5th International Conference on Information Science, Computer Technology and Transportation (ISCTT), Shenyang, China, 2020, pp. 439-442, doi: 10.1109/ISCTT51595.2020.00083.,

Wu Ying, Yang Kai and Zhang Jianzhong, "Using DBSCAN clustering algorithm in spam identifying," 2010 2nd International Conference on Education Technology and Computer, Shanghai, 2010, pp. V1-398-V1-402, doi: 10.1109/ICETC.2010.5529221.,

Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), Munich, 1996, pp. 226-231