Analysis of DBSCAN Clustering Algorithm

Name: Colin Cooper

Student ID: R00240295

COMP9071: Fraud and Anomaly Detection

Assignment 2

Executive Summary

In this assignment, I ran a number of experiments by running through a python-based DBSCAN clustering function with different parameters. The experiments were as follows:

- **Experiment 1:** Run a knn algorithm to the 10th nearest neighbour, graph the result, and use a visual estimation to determine a good range for the Eps values in Experiment 2.
- **Experiment 2:** Run the DBSCAN clustering algorithm against the sample data with the following parameters:
    - **Eps:** [0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007]
    - **MinPts:** [10, 20, 30, 40, 50, 60]

  The results are graphed against the silhouette score, an internal measure of cluster "goodness" at each combination of Eps and MinPts. The results are also graphed against the number of clusters found to look at the relationships between the results and input Eps and MinPts values. The "anomaly overlap" score - the degree to which DBSCAN found outliers (Cluster -1) which were also classed as anomalies – was also captured during these experiments for use later in Experiment 5.
- **Experiment 3:** I ran one additional run of the DBSCAN algorithm with values (Eps[], MinPts) = ([0.002, 0.004, 0.006, 0.008, 0.01, 0.02, 0.04, 0.06, 0.08], 50) in order to "zoom-in" on the effect of Eps on number of clusters found across a higher range of Eps values. This experiment confirmed that the number of clusters found fell as Eps increased.
- **Experiment 4:** In Experiment 2, the silhouette score-optimised (0.721) values of Eps and MinPts was found to be 0.007 and 60 respectively. In Experiment 4, I wanted to check how well this point performed against several external measures, including:
    - **Anomaly Overlap:** The degree to which DBSCAN found outliers (Cluster -1) which were also classed as anomalies.
    - **Purity Score:** Each cluster was assigned a purity score against a normal / anomaly classification. The overall clustering was also given an anomaly score.
- **Experiment 5:** Taking the (Eps, MinPts) that have the optimal anomaly overlap score (37.2%) in Experiment 2, I ran the DSCAN experiment again using values (0.007, 20) to see how well this performed against the silhouette-optimised result in Experiment 4.

Introduction

For this assignment, I chose to use the DBSCAN clustering algorithm to analyse the provided dataset. This is because after an initial attempt to analyse the data features, it became clear that the data points were not neat globular sets in any of the data dimensions that can easily be represented by centroid values. The data appeared to be of complex shapes in multi-dimensional space which might be better analysed through the DBSCAN method.

DBSCAN

The DBSCAN method seeks to identify high-density regions regardless of the multidimensional shape complexity. The basic algorithm (Starmer, 2022) underpinning DBSCAN is to:

1. Label every datapoint in the dataset that has at least *MinPts* neighbours within a multi-dimension radius of *Eps* as a core point.
2. Starting with a random core point, assign it to a cluster, and assign all neighbouring datapoints that are core points to the same cluster until there are no more neighbouring core points.
3. Then selected another unclustered core point at random and run the Step 2 for that core point. Repeat step 3 until there are no more core points in the dataset.
4. With the remaining non-core datapoints, those that are within a radius of Eps to a core point are also assigned to the nearest clustered core point.
5. If any datapoints are not assigned to a cluster (i.e. they do not have a core point within radius Eps), label these as outliers.

The two main parameters to consider for a DBSCAN experiment are:

Epsilon: the distance within which to count neighbouring points.

Expectations for epsilon values

Low Epsilon	High Epsilon
When epsilon is low, DBSCAN will find fewer neighbours for each data point because it's looking in a smaller multi-dimensional space. Therefore, it will classify fewer datapoints as being core points, and more as outliers. This will result in a larger number of small clusters and more outliers.	When epsilon is high, more points will be labelled as core points, so there will be larger, but fewer clusters. When epsilon is set too high true anomalies could be missed with all data sets assigned to low-resolution clusters that are less meaningful.

MinPts: the minimum number of neighbouring data points within the epsilon distance for a point to be considered a core point.

Expectations for MinPts values

Low MinPts	High MinPts
When MinPts is low, the threshold of neighbouring data points within a radius of *epsilon* for any data point to be classed as a core point is also low. Therefore, we should expect that more points can be considered core points to a cluster. This creates the opportunity for more clusters to be created from the algorithm (especially if epsilon is low), in less dense regions of the datapoint distribution.	When MinPts high, fewer points will meet the criteria for being a core point of a cluster and so we should expect fewer clusters and more of the datapoints in the less dense regions being classified as noise or anomalies.

The main internal metric for evaluating DBSCAN results is the silhouette number. The silhouette formula seeks to balance the cohesiveness of clusters (how close data points in a cluster are to each other) and how separate different clusters are from each other.

The silhouette value for a clustering is defined as:

$$s = \frac{(b-a)}{max(a,b)}$$

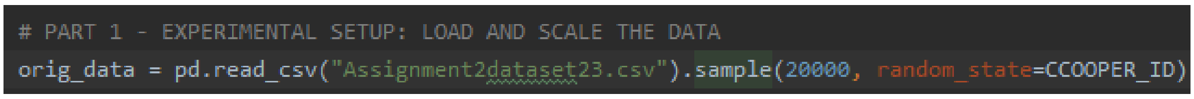
where a is the average distance of each point from other points in the same cluster (cohesion), and b is the minimum average distance from points in another cluster (separation).

If clusters are well separated and contain tight datapoints then a will be very low and b will be very high such that the value will approach $s = \frac{(b)}{b}$, or 1.

Simply speaking, we are looking for a silhouette value that is as close to 1 as possible.

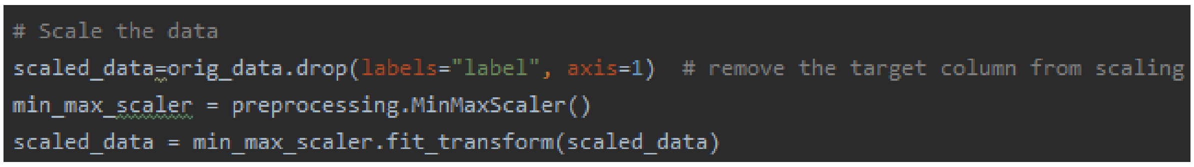
Part 1. Experiment Setup: Load and Scale Data

The data was loaded from the CSV file. However, I only took a sample of the loaded records for the DBSCAN clustering algorithm as I found that trying to run DBSCAN modelling on the whole dataset was extremely slow and often caused out-of-memory issues.



```
# PART 1 - EXPERIMENTAL SETUP: LOAD AND SCALE THE DATA
orig_data = pd.read_csv("Assignment2dataset23.csv").sample(20000, random_state=CCOOPER_ID)
```

The data was then scaled (with the target / label column removed) using the MinMax scaler so that all features used the same 0 to 1 scale to prevent overweighting on features that contain higher values.



```
# Scale the data
scaled_data=orig_data.drop(labels="label", axis=1)  # remove the target column from scaling
min_max_scaler = preprocessing.MinMaxScaler()
scaled_data = min_max_scaler.fit_transform(scaled_data)
```

Part 2. Parameter Analysis - Investigate Sensitivity of DBSCAN to values of Eps AND MinPts (internal measures only)

Internal measures of the quality of clustering refer to those that are not measured against a known baseline or ground-truth. The primary internal measure I looked at when changing Eps and MinPts was the silhouette metric.

Selecting Test Ranges for Eps and MinPts

See Experiment 1 in the related python code

To get an idea of what the best values of Eps might be, I used the “elbow” method to look at the spread of distances each point had from its 10th nearest neighbour. This data was then sorted in descending order and reindexed such that points with a low index would have a high knn distance value, while those values with a high index would have a low knn distance. What I’m looking for is an inflection point on the graph that indicates tighter density to the right, and lower density to the left.

At this point, knn distance value (or a range of values around it) might be a good starting point for using with DBSCAN.

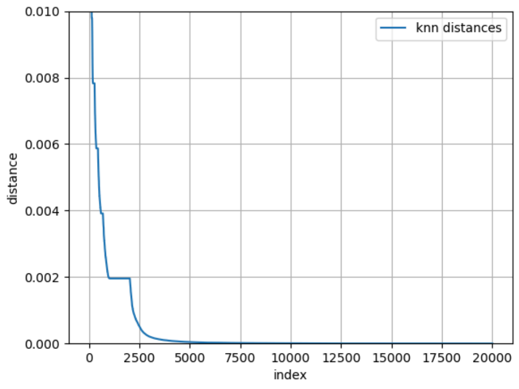


Figure 1 - Elbow graph of distances for the 10th nearest neighbour of each data point

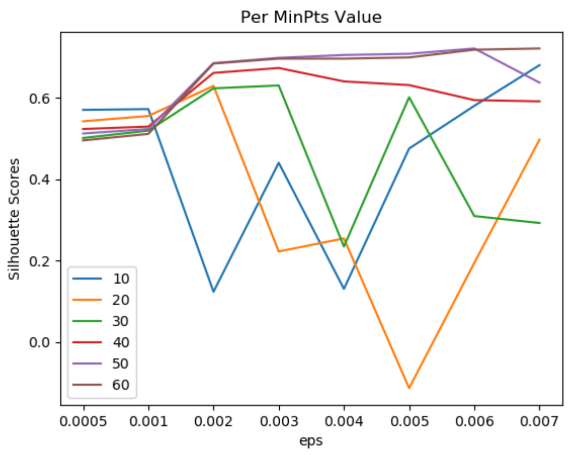
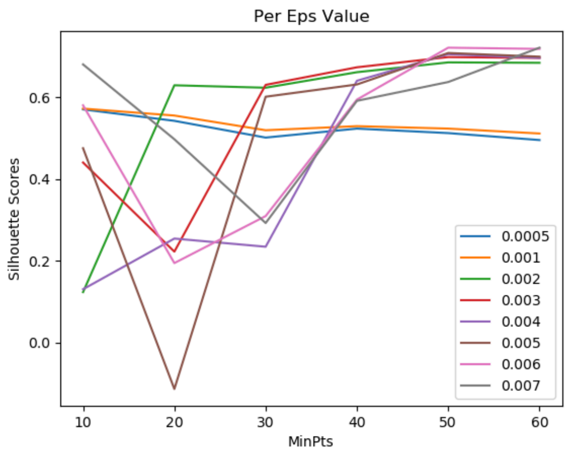
Visually, the inflection point seemed to be somewhere around 0.001, so for this experience I chose a range of test values for Eps of: [0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007].

For MinPts, I used some trial and error to come up with a test range of values. Eventually, I settled on a lower limit of 10, with further test values up to and including 60.

Epsilon and MinPts impacts on Silhouette score

See Experiment 2 in the related python code

I graphed the results of the DBSCAN test runs to produce representations that showed the effect of different Eps and MinPts values on the silhouette score.

 The lower tested values of MinPts (10: blue; 20: yellow; 30: green) produced very inconsistent results where changes in the Eps value created large unpredictable changes in the silhouette score.	 A transposed view of this data shows more clearly that the higher MinPts values produced better results as measured by the silhouette score, but seemed to level out at around 50, indicating that this might be the optimal value for MinPts to derive the best clustering.

The higher test values (40: red; 50: purple; 60: brown) produced higher silhouette scores between Eps values of 0.002 and 0.006, indicating that the best clustering results might be achieved with Eps values in this range.	

Epsilon and MinPts impacts on # of clusters found

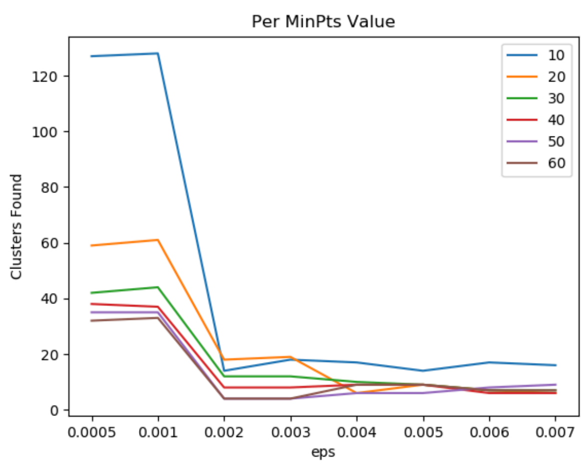
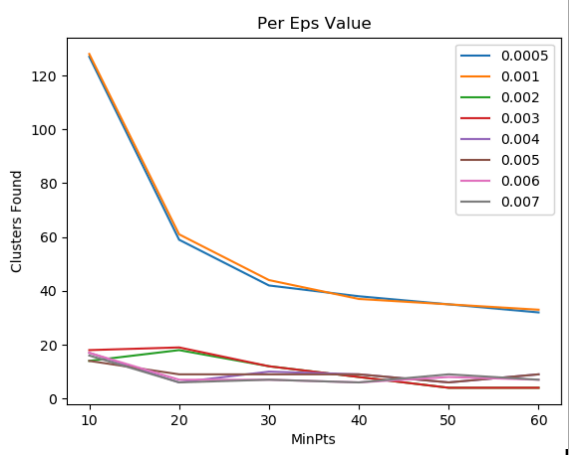
The number of clusters found isn't a good indication of quality without knowing something about the data. We don't yet know the ground truth of how any clustering correlates to known classifications of data (and it should be noted that clustering and classifications are not the same thing, just that a good clustering might be suggestive of a classification that may be known or unknown beforehand).

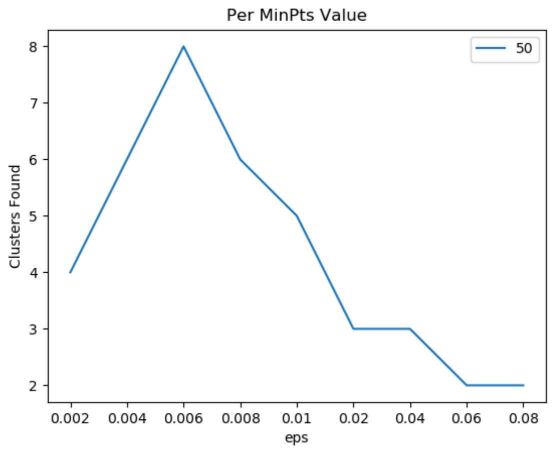
However, it might be interesting to see the impact of the Eps and MinPts values on the number of clusters found by the DBSCAN method.

Expected Behaviour

Eps: As Eps increases, therefore the radius from a core point to include other points within the cluster increases. This means that more points will match the criteria for inclusion within a cluster (Prado, 2017), and there will be fewer outliers found. If the Eps is too small, then this can result in a large number of clusters with very fine-grained differences between those clusters. If the Eps is too large, this can result in more distant data points being grouped together into larger, but fewer clusters (Trajanovski & Zhang, 2021).

MinPts: MinPts is "*The minimum number of data points within the radius of a neighbourhood (i.e. epsilon) for the neighbourhood to be considered a cluster*" (Mysiak, 2020). A lower MinPts is likely to result in more clusters, but with more outliers; while an increasing MinPts will should result in fewer clusters with fewer outliers.

	
As Eps grew from 0.0005 to 0.007, we found fewer clusters, but the number of clusters found for each MinPts value seemed to stabilise at around where Eps = 0.002. To check that this wasn't just a localised effect, I ran an additional experiment (Experiment 3) holding MinPts=50, but running a large range of Eps values from 0.002 to 0.08. The effect of larger	As MinPts grew, we found fewer clusters across all values of Eps. This is the effect that was expected.

Eps resulting in fewer clusters was consistent even within the wider range. 	

Part 3: Using the best parameter value(s) identified in the previous analysis, how did the method perform? Here you should compare using both internal and external measures of cluster validity? What insights can be taken from the results?

See Experiment 4 in the related python code

Using the results from the internal measures, the optimal (Eps, MinPts) value was around (0.007, 60) as highlighted in Figure 2:

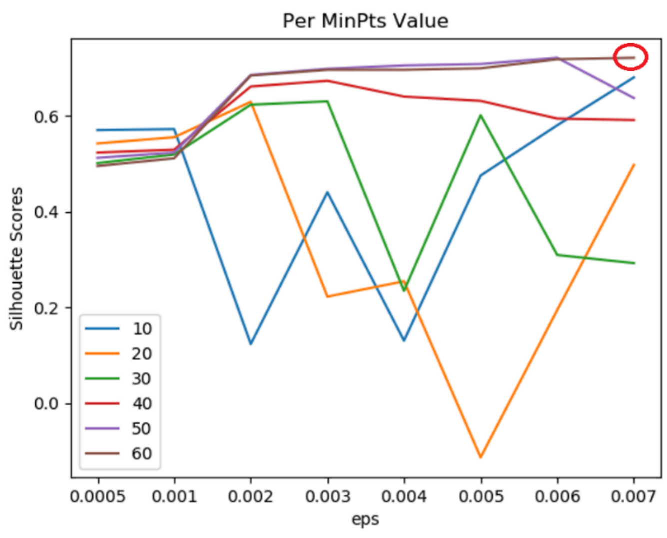


Figure 2 - Optimised silhouette score from the experiment

Internal measures

At this level, the silhouette score was maximised at 0.721 which seems like a reasonably good score. However, this doesn't necessarily mean that the clustering was useful. All we have is a set of data points that have been grouped in a way that isn't very clear from multi-dimensional data values. We don't yet know if this clustering is correlated with the known labels of 0 (normal) or 1 (anomaly) that we have in the date.

External measures

One quick check would be to see if that data DBSCAN has classified as anomalous is similar to that classified in the dataset as anomalous (anomaly overlap). Of the 20,000 sampled records 1,264 of them were classified as anomalous so even a random selection of datapoints would have an average of 6.32% anomalous records. If DBSCAN has done a good job in finding anomalies that match, then we should see a much higher percentage of classified-anomalies within the clustered anomalies. This is because anomalies should be, by their nature rare and so exist in less dense regions of the dataset. DBSCAN will assign datapoints in less dense regions to cluster -1 which is the outlier cluster (and not a true cluster).

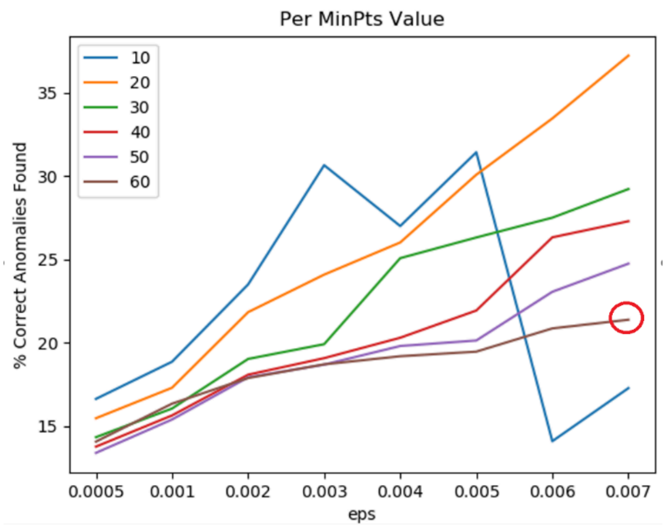


Figure 3 - The optimal DBSCAN parameters don't match well to anomaly overlap

Figure 3 shows that while 21.4% of the clustered anomalies were also classified anomalies, this is clearly far from the optimal result. In fact, where (Eps, MinPts) = (0.007, 20) appears to give the best result (37.2%) even though this only had a silhouette score of 0.497 from earlier.

By cross tabbing the cluster and classification (pd.crosstab) information for the dataset where (Eps, MinPts) = (0.7, 60), we get:

Cluster	**Class = 0**	**Class = 1**	**Purity**	**% Classified Anomalies**
-1	1229	334	78.6%	21%
0	16077	487	97.1%	3%
1	3	265	98.9%	99%
2	581	103	84.9%	15%
3	428	48	89.9%	10%

4	243	0	100%	0%
5	99	19	83.9%	16%
6	76	8	90.5%	10%
			94.99%	

* Purity = max(class 0, class 1)/(class 0 + class 1)

Clusters 0 and 4 are quite “pure” clusters with 97% and 100% normal data respective, while Cluster 1 is also very pure with 99% anomaly data. The purity of a cluster is defined as the percentage of the class with the maximum number of datapoints. So, if all points in the cluster are assigned to just one class, it is 100% pure. So, instead of just considering cluster -1 as being an anomaly cluster, what if we instead considered Clusters 0 and 4 as normal (since they are the purist normal) and all the other clusters as anomalous data points (since they contain more than the known average number of anomalies you’d expect from a random selection of records – i.e. 6.32%):

	Cluster	**Class = 0**	**Class = 1**
Normal	**0**	16077	487
	4	243	0
		16320	**487**
Anomaly	**1**	3	265
	2	581	103
	3	428	48
	5	99	19
	-1	1229	334
	6	76	8
		2,416	**777**

Which can be simplified to a confusion matrix:

	Data predicted Normal	**Data predicted Anomaly**
Class = 0 (actually Normal)	16320 (TN)	2416 (FP)
Class = 1 (actually Anomaly)	487 (FN)	777 (TP)

The “goodness” of this clustering / classification can be measured by purity and entropy. For this experiment the purity score is:

16077 + 243 + 265 + 581 + 428 + 99 + 1229 + 76 / 20000 = 94.99%

Experiment 4 Results:

Metric	**Formula**	**Result**
Accuracy Rate	$\frac{(TP+TN)}{(TP+TN+FP+FN)}$	85.5%
True Positive Rate (Recall)	$\frac{(TP)}{(TP+FN)}$	61.5%
True Negative Rate (Specificity)	$\frac{(TN)}{(TN+FP)}$	87.1%

False Positive Rate	$\frac{(FP)}{(FP+TN)}$	12.9%
False Negative Rate	$\frac{(FN)}{(FN+TP)}$	38.5%
F-Score	$\frac{(TP)}{(TP+\frac{1}{2}(FP+FN))}$	34.9%

I wanted to see if using the anomaly-optimised (Eps, MinPts) of (0.007, 20) would produce a better result, so I created a new experiment that ran DBSCAN with these values.

See Experiment 5 in the related python code

The experiment produced the following results:

Cluster	**Class = 0**	**Class = 1**	**Purity***	**% Classified Anomalies**
-1	312	185	62.8%	37%
0	18008	775	95.9%	4%
1	3	265	98.9%	99%
2	104	39	72.7%	27%
3	243	0	100%	0%
4	44	0	100%	0%
5	22	0	100%	0%
			94.99%	

	Cluster	**Class = 0**	**Class = 1**
Normal	**0**	18008	775
	3	243	0
	4	44	0
	5	22	0
		18317	775
Anomaly	**-1**	312	185
	1	3	265
	2	104	39
		419	489

	Data predicted Normal	**Data predicted Anomaly**
Class = 0 (actually Normal)	18317 (TN)	419 (FP)
Class = 1 (actually Anomaly)	775 (FN)	489 (TP)

Metric	**Formula**	**Result**
Accuracy Rate	$\frac{(TP+TN)}{(TP+TN+FP+FN)}$	94.0%

True Positive Rate (Recall)	$\frac{(TP)}{(TP+FN)}$	53.9%
True Negative Rate (Specificity)	$\frac{(TN)}{(TN+FP)}$	95.9%
False Positive Rate	$\frac{(FP)}{(FP+TN)}$	4.1%
False Negative Rate	$\frac{(FN)}{(FN+TP)}$	46.1%
F-Score	$\frac{(TP)}{(TP+\frac{1}{2}(FP+FN))}$	45.0%

Conclusions

Results Summary

	Silhouette-optimised	**Anomaly Overlap-optimised**
	Eps=0.007, MinPts=0.6	**Eps=0.007, MinPts=0.2**
Silhouette Score	0.721	0.497
Anomaly Overlap	21.4%	37.2%
Accuracy Rate	85.5%	94.0%
True Positive Rate (Recall)	61.5%	53.9%
True Negative Rate (Specificity)	87.1%	95.9%
False Positive Rate	12.9%	4.1%
False Negative Rate	38.5%	46.1%
F-Score	34.9%	45.0%
Clustering Purity	94.99%	94.99%

The DBSCAN clustering method is doing an excellent job of finding anomalies when the Eps and MinPts are tuned to good silhouette values. I initially ran DBSCAN over a range of values and selected two pairs of values that optimised silhouette score and the anomaly overlap score (where class = 0 and cluster = -1).

Even though the overlap-optimised results were generally better due to its ability to more correctly assign normal records, this came at a cost of having a worse ability to assign anomalous records.

Potentially improving the models

Categorical data: X5 contains values of either 0, 1 or 2 which make it possibly a categorical column. If these values represent non-ordinal categories, then it doesn't make sense to include them in distance calculations for DBSCAN, since the difference between 0 and 1 is not less than between 0 and 2 if they represent non-ordinal categories. It would make sense to either exclude the feature completely to avoid computational cost and inclusion of noise into the models, or to transform it into 3 new columns with binary values. A better understanding of what the data represents would help to ensure we are using it in the best way possible.

Benefits of Streaming K-means versus k-means and bisecting k-means for streaming datasets.

Streaming k-means, k-means, and bisecting k-means are popular algorithms for clustering large and high-dimensional datasets. The challenge with streaming data is that the definition of anomalous and normal can change over time and it is necessary to retrain models with new training data as it is available. This adds an additional challenge as to what dataset to train the model on – should it be the whole dataset since the first record was created, or a subset of the data over time with older records being discarded.

Streaming k-means introduces the concept of α (alpha) which allows the data selection for retraining to be customised in a kind of "*exponentially-weighted moving average*" (Anon., n.d.) way, that allows customisation of how older data points are treated and weighted with regards to model retaining. When alpha = 0, older data has no weight and only new data is used for retraining. When alpha = 1, all data since the beginning of the dataset is included. In the range between 0 and 1, less or more weight is given to older data but it is not excluded completely, and it is less influential on the model than newer data.

Bibliography

Anon., n.d. *Clustering - RDD-based API.* [Online]
Available at: https://spark.apache.org/docs/latest/mllib-clustering.html#streaming-k-means
[Accessed 29 April 2023].

Mysiak, K., 2020. *Explaining DBSCAN Clustering.* [Online]
Available at: https://towardsdatascience.com/explaining-dbscan-clustering-18eaf5c83b31
[Accessed 2023 April 29].

Prado, K. S. d., 2017. *How DBSCAN works and why should we use it?.* [Online]
Available at:
https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80
[Accessed 29 April 2023].

Starmer, J., 2022. *Clustering with DBSCAN, Clearly Explained!!!,* s.l.: Youtube.com.

Trajanovski, T. & Zhang, N., 2021. An Automated Behaviour-Based Clustering of IoT Botnets. *Future Internet,* p. 11.