



## **University of Huddersfield Repository**

Baadel, Said

A Machine Learning Clustering Technique for Autism Screening and Other Applications

### **Original Citation**

Baadel, Said (2019) A Machine Learning Clustering Technique for Autism Screening and Other Applications. Doctoral thesis, University of Huddersfield.

This version is available at <http://eprints.hud.ac.uk/id/eprint/35113/>

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: [E.mailbox@hud.ac.uk](mailto:E.mailbox@hud.ac.uk).

<http://eprints.hud.ac.uk/>

**A MACHINE LEARNING CLUSTERING TECHNIQUE  
FOR AUTISM SCREENING AND OTHER  
APPLICATIONS**

**SAID AWADH BAADEL**

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy

The University of Huddersfield

Submission date: September 2019

## Copyright statement

- i. The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the "Copyright") and s/he has given The University of Huddersfield the right to use such copyright for any administrative, promotional, educational and/or teaching purposes.
- ii. Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the University Library. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.
- iii. The ownership of any patents, designs, trademarks and any and all other intellectual property rights except for the Copyright (the "Intellectual Property Rights") and any reproductions of copyright works, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions

# **ACKNOWLEDGMENTS**

I would like to first and foremost thank the Almighty Lord for enabling me and giving me the strength to be patient and pursue this dream. Being the first Baadel member to have reached this goal, I hope to set precedence to the future Baadel generations in their pursuit of knowledge.

Secondly, I would like to thank my supervisors Prof. Joan Lu, Dr. Fadi Thabtah, and Dr. Qiang Xu for their continuous support and encouragement throughout this long journey. A special thanks to Dr. Thabtah, my friend and mentor, for his direction and believing in me in periods when I had self-doubt, and for providing the ASD datasets used in the case study of this thesis.

# **DEDICATION**

I would like to dedicate this thesis to my family who endured the last few years with patience while I undertook the journey to fulfil my dream. To my beautiful wife Hasnaa, who is my true friend and soulmate, for her full support, motivation, and encouragement throughout this journey. To my children Malaadh, Muaadh and Manaayer for their cheer on, love, and inspiration. Last but not least, to my mum and dad. I hope you are proud and smiling down from the heavens. We finally did it!

This is for you all!

## **Publications**

S. Baadel, F. Thabtah, J. Lu. An Overlapping Clustering Approach for Multi-Label Data Analysis. Statistical Analysis and Data Mining. Wiley. Under review

S. Baadel, F. Thabtah, J. Lu. A New Clustering Approach for Autism based Autistic Trait Classification. Informatics for Health and Social Care. Taylor & Francis. Under review

S. Baadel, F. Thabtah, J. Lu. Cybersecurity Awareness: A Critical Analysis of Education and Law Enforcement Methods. ACM Computing Review. Under review

S. Baadel, J. Lu. (2019). Data Analytics: Intelligent Anti-Phishing Classification Techniques Based on Machine Learning. Journal of Information and Knowledge Management. Vol. 18 (1). Pp 1-17.

S. Baadel, F. Thabtah, J. Lu. (2016) Overlapping Clustering: A Review. IEEE Sponsored SAI Computing Conference, London, UK. Pp 233-237.

S. Baadel, F. Thabtah, J. Lu. (2015) MCOKE: Multi-Clustering Overlapping K-means Extension Algorithm. International Journal of Computer, Electrical, Automation, Control and Information Engineering. 9 (2). Pp 427-430.

S. Baadel, F. Thabtah, J. Lu. (2015). Data Clustering Approaches: A Review of hard and soft partitioning techniques. Canadian University Dubai Speaker Series, Dubai, U.A.E

# **Abstract**

Clustering is one of the challenging machine learning techniques due to its unsupervised learning nature. While many clustering algorithms constrain objects to single clusters, K-means overlapping partitioning clustering based methods assign objects to multiple clusters by relaxing the constraints and allowing objects to belong to more than one cluster to better fit hidden structures in the data. However, when datasets contain outliers, they can significantly influence the mean distance of the data objects to their respective clusters, which is a drawback. Therefore, most researchers address this problem by simply removing the outliers. This can be problematic especially in applications such as autism screening, fraud detection, and cybersecurity attacks among others.

In this thesis, an alternative solution to this problem is proposed that captures outliers and stores them on the fly within a new cluster, instead of discarding. The new algorithm is named Outlier-based Multi-Cluster Overlapping K-Means Extension (OMCOKE). The algorithm addresses an issue previously ignored by other work in overlapping clustering and therefore benefits various stakeholders as these outliers could have real-life applications. The proposed solution has been evaluated on a crucial behavioural science problem called screening of autistic traits to improve the performance of detecting autism spectrum disorder (ASD) traits and reduce features redundancy. OMCOKE was integrated as a learning algorithm with a semi-supervised ML framework approach called Clustering based Autistic Trait Classification (CATC) in Chapter 5. Based on the experimental results obtained on real datasets related to autism screening OMCOKE was able to identify potential autism cases based on their similarity traits as opposed to conventional scoring functions used by ASD screening tools. Moreover, the empirical results obtained by OMCOKE on different datasets involving children, adolescents, and adults were compared to other results produced by common ML techniques. The results showed that our semi-supervised framework offers models with higher predictive accuracy, sensitivity, and specificity rates than those of other intelligent classification approaches such as Artificial Neural Network (ANN), Random Forest, and Random Trees, and Rule Induction. These models are useful since they are exploited by diagnosticians and other stakeholders involved in ASD screening besides highlighting the most influential features. The chapters in this thesis have been disseminated or are under review in various reputable journals and in refereed conference proceedings.

# TABLE OF CONTENTS

<b>CHAPTER 1 .....</b>	<b>1</b>
<b>THESIS INTRODUCTION AND STRUCTURE .....</b>	<b>1</b>
1.1 INTRODUCTION .....	1
1.2 PROBLEMS UNDER CONSIDERATION AND RESEARCH QUESTIONS .....	3
1.3 THESIS RESEARCH ISSUES AND CONTRIBUTIONS .....	5
1.3.1 <i>Generic Issues</i> .....	5
1.3.1.1 Issue 1: Overlapping Threshold .....	5
1.3.1.2 Issue 2: Outlier Detection and Retention .....	6
1.3.1.3 Issue 3: Outlier Detection Clustering in WEKA ML Tool .....	7
1.3.1.4 Issue 4: Literature Review on Clustering and ASD using ML .....	8
1.3.2 <i>Domain Specific Issue</i> .....	9
1.3.2.1 Autism Traits Detection .....	9
1.4 THESIS STRUCTURE .....	10
<b>CHAPTER 2 .....</b>	<b>11</b>
<b>LITERATURE REVIEW: CLUSTERING APPROACHES .....</b>	<b>11</b>
2.1 INTRODUCTION .....	11
2.2 THE CLUSTERING PROBLEM AND MEMBERSHIP TYPES .....	13
2.2.1 <i>Single Membership</i> .....	13
2.2.2 <i>Multiple Membership</i> .....	14
2.2.3 <i>Similarities and Dissimilarities Measures</i> .....	15
2.2.4 <i>Clustering Taxonomy</i> .....	17
2.3 HIERARCHICAL CLUSTERING .....	17
2.3.1 <i>Divisive Hierarchical Algorithms</i> .....	18
2.3.2 <i>Agglomerative Hierarchical Algorithms</i> .....	19



2.3.3	<i>Common Classical Hierarchical Algorithms</i>	19
2.3.3.1	CHAMELEON	20
2.3.3.2	CURE	20
2.3.3.3	ROCK	20
2.3.3.4	BIRCH	20
2.3.4	<i>Challenges and Limitations of Hierarchical Clustering</i>	21
2.4	<b>PARTITIONING CLUSTERING</b>	21
2.4.1	<i>Density-based Partitioning Clustering</i>	22
2.4.1.1	DBSCAN	22
2.4.1.2	OPTICS	23
2.4.1.3	Challenges and Limitations of Density-based Clustering	23
2.4.2	<i>Model-based Partitioning Clustering</i>	23
2.4.2.1	COBWEB	24
2.4.2.2	AutoClass	24
2.4.2.3	Challenges and Limitations of Model-based Clustering	24
2.4.3	<i>Distance-based Partitioning Clustering</i>	25
2.4.3.1	K-Means	25
2.4.3.2	Challenges and Limitations of K-Means Algorithm	27
2.4.4	<i>Common Overlapping Distance-based Partitioning Clustering</i>	30
2.4.4.1	Fuzzy K-Means	31
2.4.4.2	Overlapping K-Means (OKM)	32
2.4.4.3	Weighted Overlapping K-Means (WOKM)	33
2.4.4.4	Kernel Overlapping K-Means (KOKM)	33
2.4.4.5	Multi-Cluster Overlapping K-Means Extension (MCOKE)	34
2.4.4.6	Challenges and Limitations of Overlapping Distance-based Partitioning Clustering Algorithms	34
2.4.5	<i>Outlier Detection in Partitioning Clustering</i>	35
2.4.6	<i>Classification Techniques used in the Thesis Case Study</i>	37
2.5	<b>CHAPTER SUMMARY</b>	39
<b>CHAPTER 3</b>		<b>40</b>

<b>OUTLIER BASED MULTI-CLUSTER OVERLAPPING K-MEANS EXTENSION (OMCOKE)</b>	<b>40</b>
3.1 INTRODUCTION	40
3.2 MCOKE	42
3.3 OMCOKE	45
3.4 EXAMPLE ON OMCOKE	48
3.4.1 OMCOKE Algorithm Step 1	49
3.4.2 OMCOKE Algorithm Step 2	49
3.4.3 OMCOKE Algorithm Step 3	50
3.5 OMCOKE VERSUS OTHER OVERLAPPING ALGORITHMS	51
3.6 CHAPTER SUMMARY	52
<b>CHAPTER 4</b>	<b>54</b>
<b>IMPLEMENTATION AND EVALUATION OF OMCOKE</b>	<b>54</b>
4.1 INTRODUCTION	54
4.2 OMCOKE IMPLEMENTATION	54
4.3 EVALUATION MEASURES	58
4.3.1 Confusion Matrix in Clustering	59
4.3.2 Computing Resources Evaluation Measures	60
4.4 DATA EXPERIMENTAL SETTINGS	61
4.4.1 Evaluation Methodology	62
4.4.2 Description of Overlapping Datasets	63
4.4.2.1 Emotion dataset (Troihidis, et al., 2008)	63
4.4.2.2 Yeast dataset (Elisseeff and Weston, 2001)	63
4.4.2.3 Scene dataset (Boutell, et al., 2004)	63
4.5 EMPIRICAL RESULTS ON THE MULTI-LABEL DATASETS	64
4.6 CHAPTER SUMMARY	68
<b>CHAPTER 5</b>	<b>70</b>

<b>CASE STUDY ON AUTISM SPECTRUM DISORDER (ASD) SCREENING .....</b>	<b>70</b>
5.1 INTRODUCTION .....	70
5.2 BACKGROUND INFORMATION ON ASD DETECTION .....	73
5.3 CLUSTERING BASED AUTISTIC TRAIT CLASSIFICATION (CATC) .....	76
5.3.1 <i>Data Collection</i> .....	77
5.3.2 <i>The initial Dataset and Data Transformation</i> .....	79
5.3.3 <i>Unsupervised Clustering Phase</i> .....	82
5.3.4 <i>Clustering based Autistic Traits Dataset: Initial Model</i> .....	83
5.3.5 <i>Classification</i> .....	84
5.4 EXPERIMENTAL SETTINGS .....	85
5.5 EMPIRICAL RESULTS AND ANALYSIS .....	86
5.6 CHAPTER SUMMARY .....	92
<b>CHAPTER 6 .....</b>	<b>95</b>
<b>CONCLUSIONS AND FUTURE WORK .....</b>	<b>95</b>
6.1 RESEARCH SUMMARY .....	95
6.2 RESEARCH CONTRIBUTIONS.....	95
6.2.1 <i>Overlapping Clustering with self-calculating threshold</i> .....	95
6.2.2 <i>Noise and Outlier detection</i> .....	96
6.2.3 <i>Clustering based Autistic Trait Classification technique to improve ASD Screening: A Case Study</i> .....	97
6.2.4 <i>Outlier Detection Clustering Technique in WEKA ML Tool</i> .....	97
6.3 RESEARCH IMPLICATIONS AND LIMITATIONS .....	98
6.4 FUTURE WORK .....	99
6.4.1 <i>Fusing of Multi-Labelled Clusters</i> .....	99
6.4.2 <i>Distributed Overlapping Clustering with MapReduce</i> .....	102
<b>BIBLIOGRAPHY.....</b>	<b>104</b>
<b>APPENDIX A.....</b>	<b>116</b>

EXTENDING OMCOKE IN WEKA .....	116
1. LOADING AND PRE-PROCESSING DATA IN WEKA .....	116
2. CLUSTERING METHODS .....	117
3. STANDARD OUTPUTS OF THE CLUSTERING METHODS .....	117
4. OMCOKE PARAMETER SETTING.....	119
5. EVALUATION OF THE RESULTING CLUSTER IN OMCOKE .....	120
6. VISUALIZATION OF DATA.....	123
<b>APPENDIX B .....</b>	<b>124</b>
OMCOKE SOURCE CODE .....	124

## List of Figures

Figure 1.1 Data mining steps.....	2
Figure 2.1 Clustering Taxonomy.....	17
Figure 2.2 Clustering Dendrogram.....	18
Figure 2.3 K-means algorithm .....	26
Figure 2.4 K-means flowchart.....	27
Figure 3.1 MCOKE Pseudocode.....	44
Figure 3.2 Outlier detection Pseudocode .....	47
Figure 3.3 OMCOKE Sample dataset .....	48
Figure 3.6 Multi-clustered groups.....	50
Figure 3.7 Overlapping clusters.....	50
Figure 4.1 OMCOKE algorithm integrated inside the clusterers in WEKA .....	56
Figure 4.2 OMCOKE algorithm GUI parameter setting and options.....	57
Figure 4.3 Precision Accuracy of the benchmark datasets.....	66

Figure 4.4 F-Measure of the Benchmark Datasets.....	67
Figure 5.1 CATC-based methodology .....	77
Figure 5.2 Statistics of used ASD Datasets .....	79
Figure 5.3 Pseudocode of Clustering phase in CATC method .....	82
Figure 5.6 Error rate comparisons .....	88
Figure 5.7 Sensitivity & Specificity Rates of the Classifiers.....	89
Figure 5.8 Harmonic mean on the classifiers.....	91
Figure 5.9 ROC Area of the classifiers .....	91
Figure 5.10 # of Rules Generated in PART and RIPPER classifiers .....	92

## List of Tables

Table 2.1 A single membership table example .....	14
Table 2.2 Multiple membership table example.....	15
Table 3.1 OMCOKE Sample dataset .....	48
Table 3.2 K-means output .....	49
Table 3.3 OMCOKE Output.....	50
Table 3.4 Outlier Cluster .....	51
Table 4.1 Confusion Matrix .....	60
Table 4.2 Statistics of used Benchmarks .....	61
Table 4.3 Descriptive Statistics of used Benchmarks .....	61
Table 4.4 Emotion dataset.....	63
Table 4.5 Scene Dataset .....	64
Table 4.6 Overlapping Algorithms Performance Comparisons .....	65
Table 4.7 Outliers Detected in the Three Datasets .....	68

Table 5.1 Statistics of used ASD Datasets .....	78
Table 5.2 ASD Data Feature Attributes.....	80
Table 5.3 AQ-10 Adult Questionnaire (Allison et al., 2012) .....	81
Table 5.4 Accuracy Rates of the Classifiers .....	87
Table 6.1 Search Results of Movie Titles.....	100

## **List of Acronyms**

ABIDE	Autism Brain Imaging Data Exchange
ADI-R	Autistic Diagnostic Interview - Revised
ADOS	Autism Diagnostic Observation Schedule
AI	Artificial Intelligence
ANN	Artificial Neural Network
AQ	Autism Spectrum Quotient
ASD	Autism Spectrum Disorder
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
CARS	Childhood Autism Rating Scale
CATC	Clustering based Autistic Traits Classification
CBCL	Child Behaviour Checklist
CJ	Clinical Judgement
CF	Clustering Feature
CLARA	Clustering Large Applications
COR	Clustering with Outlier Removal
CURE	Clustering Using Representatives
DM	Data mining
FCM	Fuzzy C-means
FN	False Negative

FP	False Positive
KMOR	K-means with Outlier Removal
KOKM	Kernel Overlapping K-means
LVQ	Learning Vector Quantization
MCHAT	Modified Checklist for Autism in Toddlers
MCOKE	Multi-Cluster Overlapping K-Means Extension
ML	Machine Learning
MT	Membership Table
OED	Outlier Eliminated Dataset
OKM	Overlapping K-means
OMCOKE	Outlier based Multi-Cluster Overlapping K-means Extension
PAM	Partitioning Around Medoids
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
RT	Random Tree
RF	Random Forest
RFID	Radio Frequency Identification
SOM	Self-Organization Feature Map
SSE	Sum of Squared Error
TN	True Negative
TP	True Positive
VA	Variable Analysis
WEKA	Waikato Environment for Knowledge Analysis
WOKM	Weighted Overlapping K-means

# Chapter 1

## Thesis Introduction and Structure

### ***1.1 Introduction***

There has been an exponential data growth recently and massive amounts of data being collected and stored from social media sources such as Twitter, Snapchat, Facebook, etc. and other data sources collected from GPS sensors, Radio Frequency (RFID) systems, IoT (Internet of Things) among others. This has led to what is commonly known as Big Data (Coronel and Morris, 2018).

Meaningful information can be extracted from Big Data through data mining (DM) and machine learning (ML) techniques in order to discover and predict hidden patterns and trends. Machine learning (ML) is a branch of artificial intelligence (AI) where systems are able to learn from any given data and adapt to the new information provided in the data (Abdelhamid et al., 2014; Witten and Frank, 2005).

Two fundamental concepts define DM techniques; supervised and unsupervised learning (Town and Thabtah, 2019). In supervised learning, given an input object, we use a set of labelled training data to determine an output of continuous value (regression) or predict a class label associated with it (classification). By using the training dataset, supervised learning techniques would learn from the dataset after seeing some examples and use that knowledge to predict the output value. On the other hand, unsupervised learning techniques have no prior knowledge of the data labels. These techniques extract hidden patterns by looking at similarities between the data objects and grouping them into clusters.



Figure 1.1 below highlights the steps involved in DM activities.

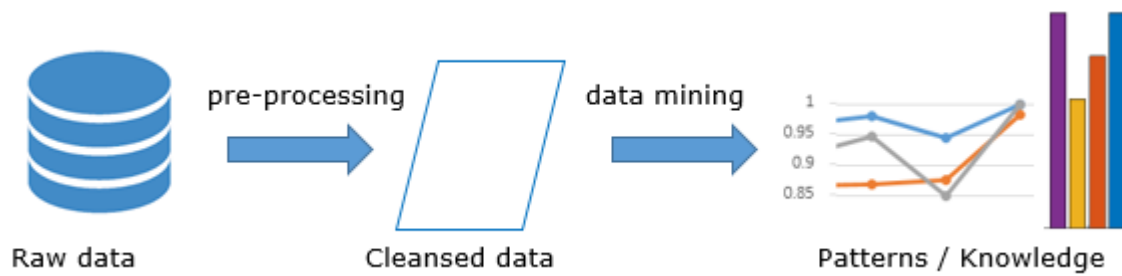


Figure 1.1 Data mining steps

Data clustering, also known as the unsupervised classification, is a research field widely studied in DM and ML domains due to its applications to segmentation, summarization, learning, and target marketing (Aggarwal & Reddy, 2014; Hadi et al., 2008).

Clustering involves the partitioning of a set of objects or data into clusters or subsets such that the objects or data in each subset or cluster contains similar traits based on measured similarities and data from different clusters are dissimilar (Saxena et al., 2017). Many techniques have been explored for clustering processes such as distance-based, probabilistic, and density/grid-based, etc. with the distance based being very popular in research fields (Jain, 2010).

Many clustering algorithms in unsupervised learning constrain objects to single clusters (i.e., objects belong to exactly one cluster) while ignoring the fact that some objects may have attributes that can belong to more than one cluster. Undoubtedly, K-means is the most widely used partitional clustering algorithm (Jain, 2010). There are many reasons attributed to this such as; a) it is straightforward to implement, b) very versatile in that any part of the algorithm can be easily modified, c) it is guaranteed to converge (Selim and Ismail, 1984) at a quadratic rate (Bottou and Bengio, 1995). Thus, the algorithm has been used extensively to solve non-overlapping clustering problems. Overlapping clustering methods remove the constraints and assign objects to one or more clusters building a non-disjoint partition of the data.

Data anomalies commonly referred to as outliers affect the mean distance used in the calculation of objects belonging to a cluster. This is a significant drawback in the K-means algorithm. While many algorithms opt to prune the outliers in the dataset (Liu, et al., 2018; Barai & Dey, 2017), little research has been done to store these outlier objects for further investigation. Therefore, this research addresses this gap by developing a new overlapping clustering algorithm that extends the K-means which stores outliers in a separate cluster.

Many researchers have adopted ML techniques such as supervised learning in classification to predict or detect features in Autism Spectrum Disorder (ASD) screening and diagnosis (Thabtah, 2018a; Abbas et al., 2017; Maenner et al., 2016). In the machine learning phase of ASD screening, data is pre-processed and cleansed before running it to a classifier. Little research has been done to apply clustering of the data at this stage where robust features can be identified and grouped to facilitate the cleansing process. This research addresses this gap by developing a new semi-supervised screening architecture that incorporates unsupervised clustering with supervised classification methods.

This chapter discusses the research problems investigated and the significant contributions of this thesis.

## ***1.2 Problems Under Consideration and Research Questions***

Clustering is an unsupervised learning process that involves grouping a set of data objects into subsets, each of which has its label based on a predefined similarity metric (Arabie, Hubert & DeSoete, 1999). In clustering, some structural characteristics are not known a priori unless some domain knowledge is presented in advance (i.e., there are no labels attached to the data patterns as in the case of supervised classification), thus deeming clustering a challenging problem due to this unsupervised nature (Hrushka, Campello, Freitas, & Carvalho, 2009); (Saxena et. al., 2017). Undoubtedly, the K-means (MacQueen, 1967), and its generic extensions and adaptations are one of the most widely used distance-based partition-clustering algorithms (Hrushka, Campello, Freitas & Carvalho, 2009); (Jain, Murty, & Flynn, 1999); (Lam & Wunsch,

2014). Thus, the K-means algorithm has been primarily utilized to deal with non-overlapping clustering problems that limit each data object to a single cluster.

There are a few known issues that affect the K-means algorithm. These include; defining the number of cluster  $k$  in advance where the data is not known, defining the cut-off threshold for overlapping objects to belong to different clusters, and assigning weight to objects that belong at the intersection of two or more clusters. Another major challenge of K-means and its successors is sensitivity to exceptional data (outliers). K-means often derives clusters by optimizing the mean Sum of Squared Error (SSE) by calculating the Euclidean distance between the data objects and the cluster computed centroids. Therefore, any outliers in the dataset will significantly affect the means and the variance. In return, this will affect how the algorithm calculates and assigns objects to their cluster centroid.

The lack of overlapping clustering algorithms that can store the outlier objects makes it difficult to scrutinize these noise data. This shortcoming has practical implications in domains such as in medical informatics where such data could indicate certain abnormal traits that can be picked up in the screening process or in cyber security domain where outlier data could indicate system intrusion.

This study investigates some of the shortcomings associated with overlapping K-means algorithms listed above. The thesis will answer the following research questions:

- 1) Can we determine the cut-off threshold for overlapping objects using heuristics calculated in the clustering process as opposed to defined ones?
- 2) Can we store outlier objects that could potentially be investigated by experts in the field and derive some value in them as opposed to pruning and discarding them?
- 3) Can a semi-supervised model based on clustering and classification techniques be used in the autism spectrum disorder (ASD) screening in order to improve the performance of the classifier such as specificity, accuracy, and sensitivity?

### ***1.3 Thesis Research Issues and Contributions***

There are different issues related to unsupervised learning, specifically overlapping clustering approaches, that are discussed in this research. We categorize them into generic and domain specific issues.

#### **1.3.1 Generic Issues**

##### ***1.3.1.1 Issue 1: Overlapping Threshold***

The K-means algorithm has been primarily utilized to deal with non-overlapping clustering problems that limit each data object to a single cluster. Overlapping partitioning clustering methods tend to relax or remove the constraints allowing overlaps between clusters. Data objects that are at the intersection of two or more clusters are allowed to belong to multiple clusters with full membership to those clusters. A threshold has to be set to determine an object belonging to multiple clusters. Many algorithms define this threshold as a distance calculated based on the Euclidian average distance of all objects or as a percentage of the average distance defined a priori by the algorithm. These two techniques have inherent shortcomings. Objects that are very close together will have smaller distances to their centroid compared to other objects in other centroids that could be a bit sparse and using the mean on the Euclidean distance of these two clusters as the threshold to belonging may not be optimal and may eliminate some objects from belonging to other clusters. Determining a certain percentage of the average or overall distance a priori as the threshold is also not ideal; since in unsupervised learning, we do not have prior knowledge to the data.

#### **Contribution**

The K-means clustering being a greedy-descent nature algorithm is guaranteed to converge to a local minimum. Data objects are assigned to their nearest cluster center by calculating the distance using Euclidean distance measurement. The sum of squares errors (objective function) is calculated by squaring the Euclidean distances to each cluster centroid as the object is assigned

to the cluster with the smallest value. Our algorithm keeps track of the value of each by updating a constant *maxdist* with the higher value of each object assignment distance. As opposed to the Euclidian average distance, we use the highest Euclidian distance an object had to its assigned cluster. Upon completion of the K-means clustering, all objects would have been assigned to a cluster. We then iterate all objects again and using the constant *maxdist* as the threshold; we relax the object belonging to a single cluster if that object distance to the next cluster is less or equal than the threshold.

Our method, Multi-Cluster Overlapping K-means Extension (MCOKE) was published in Baadel et al. (2016). An improved version of this method is also discussed in Chapters 3 and 4.

### **1.3.1.2 Issue 2: Outlier Detection and Retention**

One of the significant challenges of K-means and its successors such as K-Medoids (Mirkin, 2005; Sheng and Liu, 2006), K-Modes (Huang, 1998), Bisecting K-means (Steinbach, et al., 2000) Kernel K-means (Scholkopf, et al., 1998) and Weighted K-means (Huang, et al., 2005) among others are sensitivity to exceptional data (outliers). Outliers are data objects or points that do not conform to the normal behaviour or model of the dataset, hence are deemed inconsistent or grossly different (Berkhin, 2006). In cases when the input dataset contains few outliers, this may significantly influence the mean distance (the outlier will skew the mean and variance) of the data objects to their respective clusters, and thus K-means tends to discard outliers (Barai & Dey, 2017; Zhang & Leung, 2003; Chandola, et. Al., 2009).

This data can be erroneous, but could also provide value in the specialised field of cybersecurity such that they can be classified as suspicious data in fraudulent activity; that could be useful for fraud detection, intrusion detection marketing, and website phishing sites. Detecting these outliers is advantageous for decision makers (as opposed to discarding them). Therefore, it will be more useful to store these outliers in a separate cluster for potential use as they represent unusual patterns.

## Contribution

It is assumed that most objects being clustered will fall close to the inner radius threshold (i.e., close to their cluster centroid) that are based on the average distance of all objects belonging to the cluster centroids. Anomalies or outliers, therefore, tend to be further away from their closest cluster centroid. Objects that have a distance more significant than the inner radius but less or equal to the outer radius (*maxdist*) are subject to further scrutiny and are flagged to ensure they are not outliers on the border of the clusters. We introduce another variable that calculates the average distance (*averdist*) between the object and the centroid for all clusters. *Averdist* acts as a new threshold for the inner radius between the object and the centroid. A third constant, *maxdistThreshold*, defines the area of the radius to be considered from the outer boundary, for example, 0.99 will mean the area covered inside the outer boundary for objects not to be considered an anomaly. By doing this, we cut off the very extreme objects that are at the borders of the clusters and are assigned to the Outlier cluster for further investigation. In cases where some knowledge of the data is known beforehand, this value (*maxdistThreshold*) can also be adjusted by the user before running the algorithm to cater for the dataset.

Our method, Outlier based Multi-Cluster Overlapping K-means Extension (OMCOKE) has been submitted for review and publication to the reputable Statistical Analysis and Data Mining journal.

### ***1.3.1.3 Issue 3: Outlier Detection Clustering in WEKA ML Tool***

WEKA is an open source ML tool based on the Java platform that contains implementations for different DM methods including filtering, classification, clustering, evaluation, and visualisation among others. This tool is widespread in research study fields in data analytics, bioinformatics, data mining, and computer science. The acronym stands for Waikato Environment for Knowledge Analysis, designed and implemented at New Zealand's Waikato University. However, there is not an overlapping clustering algorithm implemented in WEKA with the ability to detect outliers. Therefore, it is imperious to have an overlapping clustering algorithm in WEKA where such a

contribution can be applied to various data applications and can be of benefit to the different users in the community.

### **Contribution**

OMCOKE has been implemented in Java and incorporated into WEKA 3.8.2 developer version. OMCOKE inherited the different evaluation methods in WEKA to run our experiments. A comprehensive study is detailed in Chapter 4 where we compared our algorithm to those of overlapping clustering domain and implemented the same to an application in behavioural science in Chapter 5. Chapter 4 also describes the implementation of OMCOKE in WEKA and a further graphical exposition of the GUI implementation is provided in **Appendix A** with a sample of the source code in **Appendix B**.

#### ***1.3.1.4 Issue 4: Literature Review on Clustering and ASD using ML***

The main idea of review papers is to serve community members, researchers, lecturers, and students who are interested in understanding core concepts and cutting-edge up to date technology in the related field without being over-burdened with jargon and formulae. Therefore, it is essential to have a thorough investigation of the literature where recent advancements in the domain field are discussed and presented systematically for the different stakeholders.

### **Contribution**

Chapters 2 and 3 of this theses benefitted from literature review papers on clustering techniques in general (Baadel et al., 2015a; Baadel et al., 2015b) and specifically on overlapping clustering (Baadel et al. 2016) that have been disseminated through conference proceedings and paper presentations. Further, the literature review on ASD screening using ML techniques is discussed in chapter 5 and our paper currently being reviewed in a reputable journal. More of this is discussed in the next subsection on Autism Traits Detection.

## **1.3.2 Domain Specific Issue**

### ***1.3.2.1 Autism Traits Detection***

Autism screening is a fundamental step that addresses whether individuals exhibit potential autistic traits related to communication, social or repeated behaviour (Abbas et al. 2018). This step is crucial as the individual and the concerned family become aware of the possibility of ASD traits early and hence can search for the needed formal assessments. There are many ASD screening tools developed by researchers such as Autism Spectrum Quotient (AQ) and Childhood Autism Rating Scale (CARS) (Baron-Cohen, 2001; Baron-Cohen et al., 2006; Krug et al., 2008; Shopler et al., 2010).

Most of the existing autism screening methods utilize scoring functions that compute a final score based on the answers given by users undergoing the screening (caregivers, parents, medical staff, teachers or even the adult patients). To be specific, the screening methods take the answers given in the questionnaire as an input for the scoring function, which in turn processes the input and computes a final score to reflect whether the individual is associated with ASD traits.

On the other hand, ML techniques have been implemented that use artificial intelligence and statistics to create intelligent models by discovering hidden patterns in data so that users can improve decisions (Thabtah et al., 2018). There have been recent attempts to adopt ML techniques in autism screening and diagnosis, i.e. (Abbas, et al., 2018; Thabtah, et al., 2018b; Levy, et al., 2017; Bekerom, 2017; Thabtah, 2017a; Bone, et al., 2016; Chen, et al., 2016). These studies focused primarily on improving time, accuracy, and reducing the dimensionality of the dataset by pinpointing influential autistic symptoms.

### **Contribution**

In Chapter 5, a case study has been discussed in this thesis where we propose a new semi-supervised learning method called Clustering based Autistic Trait Classification (CATC), to



improve the accuracy of the autism screening problem. The utilization of clustering and classification together as a semi-supervised learning technique is rare in autism screening research, and we believe ours to be the first. Unlike existing methods that primarily focused on the classification phase of cases and controls, we intend to utilize clustering with classification to validate instances in the training dataset before constructing the classification systems. CATC integrates unsupervised learning in the pre-processing phase with supervised learning in the classifier's construction phase. By integrating clustering with classification, there is a potential for improving the resulting classification systems by detecting ASD traits more accurately.

Our method, CATC, has been submitted for review and publication to a reputable journal; Informatics for Health and Social Care.

## ***1.4 Thesis Structure***

The thesis is divided into six chapters. Chapter 2 provides a critical analysis of unsupervised clustering techniques in machine learning and data mining. In chapter 3, we discuss the proposed OMCOKE algorithm following its predecessor MCOKE. In this chapter, we outline the MCOKE algorithm first and list some of its shortcomings and how we overcame those issues in the improved OMCOKE algorithm. The implementation of OMCOKE in WEKA ML tool is highlighted in Chapter 4. We detailed the evaluation measures used and compared our algorithm with some of the common overlapping clustering algorithms. We used multi-label datasets in the experimentation phase, and the results and analysis are presented in this chapter. We apply our algorithm to behavioural science in a case study in ASD screening which is detailed in Chapter 5. Experimental analysis is conducted to show how the integration of unsupervised clustering in the classification techniques dramatically improves the prediction accuracy of ASD traits. Finally, we conclude the thesis in Chapter 6 by highlighting how the research issues raised in this chapter were addressed including the contribution of this thesis to the research domain. We also mention briefly possible research direction that can be undertaken to further enhance our algorithm.

## **Chapter 2**

### **Literature Review: Clustering Approaches**

#### ***2.1 Introduction***

Enormous amount of data is generated on a daily basis that pose significant challenges to extract information from them. This is what is commonly referred to as Big Data. The sheer volume, velocity, and variety of this data mean that much meaningful information is lost in this pile of data due to the fact that not enough resources are available to store and process them. One of the common machine learning techniques that can be used to mine and decipher this Big Data into meaningful information is clustering (Feyyad, 1996). For example, much data is collected on social media. Such techniques can be used to send advertisement based on how those individuals cluster together in certain groups. The main area of this thesis is in the field of Machine Learning and Data Mining. This chapter will present the background concepts on traditional clustering approaches.

In order to extract meaningful information from the data and discover hidden patterns that can be used to predict certain trends, machine learning (ML) techniques are used. Some of these techniques include Association Analysis (rules that can predict relations between object variables in a large dataset), Classification (classifying an object to belong to one or more predefined classes), Clustering (grouping objects in clusters that share similar characteristics), and Regression (determining correlations between object attributes).

Two fundamental concepts define ML techniques; supervised and unsupervised learning. In supervised learning, given an input object, we use a set of labelled training data to determine an output of continuous value (regression) or predict a class label associated with it (classification). By using the training dataset, supervised learning techniques would learn from the dataset after seeing some examples and use that knowledge to predict the output value (Thabtah et al., 2019; Thabtah et al., 2010). On the other hand, unsupervised learning

techniques such as clustering have no prior knowledge of the data labels. These techniques extract hidden patterns by looking at similarities between the data objects and grouping them into clusters.

Many data mining algorithms have been developed to address the clustering problem of big data. These include partition-based clustering algorithms (such as K-Means and Fuzzy K-Means), hierarchical clustering algorithms (such as ROCK and BIRCH), density-based algorithms (such as DBSCAN and OPTICS), and model-based clustering algorithms (such as COBWEB and AutoClass) (Nagpal et al., 2013; Aggarwal & Reddy, 2014; Xu and Tian, 2015; Rodriguez et al., 2019).

There are two forms of clustering techniques; single membership and multiple membership techniques. In single membership techniques, objects belong to one cluster only and their calculated membership will either be 0 (does not belong to the cluster) or 1 (belongs to the cluster). Multiple membership techniques allow objects to belong to two or more clusters, allowing them to overlap (Ben N'Cir et al., 2015).

In this thesis, we focus on distance-based overlapping clustering techniques. Distance-based methods can be used with almost any data type with an appropriate distance function thus making them very popular in the data mining literature (Aggarwal & Reddy, 2014). However, in this chapter, we briefly discuss other techniques that are also suitable to solve clustering problems.

The chapter is organized as follows: Section 2.2 highlights the memberships in clustering including the taxonomy of clustering techniques. Section 2.3 looks into hierarchical clustering including some of its challenges and limitations. Section 2.4 discusses the single membership partitioning clustering and some of the related algorithms in this field. More specifically, subsection 2.4.1 discusses in length the popular K-means algorithm including its limitations. Section 2.5 discusses the multiple membership (overlapping) clustering algorithms. Finally, in section 2.6 we will conclude and provide a summary of this chapter.

## 2.2 The Clustering Problem and Membership Types

Clustering involves the partitioning of a set of objects or data into clusters or subsets such that the objects or data in each subset or cluster contains similar traits based on measured similarities and data from different clusters are dissimilar (Saxena et al., 2017). In clustering, some structural characteristics are not known a priori unless some domain knowledge is presented in advance (i.e., there are no labels attached to the data patterns as in the case of supervised classification), thus deeming clustering a challenging problem due to this unsupervised nature (Hrushka, Campello, Freitas, & Carvalho, 2009); (Saxena et. al., 2017).

In many instances, objects are assumed to belong to one cluster. These are called single membership. However, in reality, objects seem to have attributes that tend to overlap between one or more clusters and thus belong to multiple clusters. Objects with such characteristics are said to have multiple memberships.

### 2.2.1 Single Membership

Objects with single membership belong to one cluster only and their calculated membership will either be 0 (does not belong to the cluster) or 1 (belongs to the cluster). Dissimilarity measures are used to compare the object with the centroid, and a one is assigned to its nearest cluster while a 0 is assigned to the other clusters. This form of single membership grouping or is called crisp or hard-clustering.

Single membership clustering can be defined as given a set  $S$  of subsets  $S_1, S_2, S_3, S_4 \dots S_k$ , such that:

$$S_1 \cap S_2 \cap S_3 \cap S_4 \dots \cap S_k = \emptyset \quad (2.1)$$

Where any instance in  $S$  can only belong to one subset and not belong to another subset.

Table 2.1 below shows an example of a single membership where data sets of 3 objects are grouped into 3 clusters. Let's assume a membership table MT (of dimension  $N \times C$ ) such that  $MT(i,j)$  denotes a member of object  $i$  to cluster  $j$  where  $i = 1, \dots, N$  and  $j = 1, \dots, C$  and  $N=3$  and

$C=3$ . Each object in  $MT(i,j)$  is assigned a 1 (one) to denote membership to that cluster and a 0 (zero) for non-membership of a cluster.

Table 2.1 A single membership table example

$MT(i,j)$	1	2	3
1	0.0	1.0	0.0
2	1.0	0.0	0.0
3	0.0	0.0	1.0

In this simple example, object 1 has membership degree of 0, 1, and 0 to clusters 1, 2, and 3 respectively. Therefore, object 1 should belong only to cluster 2. The same applies to objects 2 and 3 belonging to clusters 1 and 3 respectively for a single membership.

### 2.2.2 Multiple Membership

Overlapping clustering allows an object to have multiple memberships, i.e., to belong to two or more clusters. Commonly used technique is referred to Fuzzy clustering techniques that allow objects to belong to multiple clusters with different degrees (Höppner et al., 1999) attained through some dissimilarity measure and assigning membership degrees to the objects. The objects are then assigned to the cluster that has the highest degree. Memberships to all clusters must however always equal to unity as defined in the Equation (2.2) below

$$\sum_{i=1}^c U_{ij} = 1, \quad \forall j = 1, \dots, n \quad (2.2)$$

Where  $U_{ij}$  is between 0 and 1,  $ci$  is the cluster centre and  $j$  is the data point from 1 to  $n$ .

Table 2.2 below shows an example of soft-clustering where data sets of 3 objects is grouped into 3 clusters. Let us assume that the same data in Table 1 above is run through some fuzzy algorithm. Each object in  $MT(i,j)$  is assigned a membership degree that must sum up to 1.

Table 2.2 Multiple membership table example

MT( $i,j$ )	1	2	3
1	0.3	0.4	0.3
2	0.5	0.4	0.1
3	0.0	0.2	0.8

In this simple example, object 1 has membership degree of  $MT(1,1) = 0.3$ ,  $MT(1,2) = 0.4$  and  $MT(1,3) = 0.3$ . The sum of membership equals to 1. While we are able to identify the different degrees of belonging, object 1 has the highest degree of membership towards cluster 2, therefore it should belong to cluster 2. Likewise objects 2 and 3 should belong to clusters 1 and 3 respectively.

### 2.2.3 Similarities and Dissimilarities Measures

In any given clustering process, the similarity and dissimilarity measures play a crucial role in assigning objects to their respective clusters by determining the distance of that object to that particular cluster centroid (Saxena and Wang, 2010). An object is classified as similar to another one if their distance measure between them is small, i.e., they are close to each other. Therefore, clustering algorithms use the similarity measures to cluster similar objects together, and those that are distant are assigned to other clusters (Rand, 1971; Milligan, 1981). A distance measure between an object  $X_i$  and  $X_j$  is denoted as  $d(x_i, x_j)$  where  $d(x_i, x_j) = d(x_j, x_i)$  i.e. the distance measure is symmetric.

The most popular measures used for continuous data is the Euclidean distance (2.3) and Manhattan distance (2.4).

$$\text{Euc. Distance } d(x_i, x_j) = ((x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{in} - x_{jn})^2)^{1/2} \quad (2.3)$$

$$\text{Man. Distance } d(x_i, x_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{in} - x_{jn}| \quad (2.4)$$

Other commonly used distance measures in clustering are summarized in Table 2.2 below.

Table 2.2 Similarity Measures in Clustering

Distance Measure	Application in Clustering	Reference	Equation
<p>Euclidean</p> $d(x_i, x_j) = \sqrt{\sum_{l=1}^n  x_{il} - x_{jl} ^2}$	Applied both for hierarchical and partitioning clustering, this is the most popular measure used for numerical datasets. Commonly used in K-means algorithm and its extensions.	Jain et. al (1999) Aggarwal and Reddy (2014) Saxena et al. (2017)	(2.5)
<p>Normalized Cosine</p> $d(x_i, x_j) = \frac{x_i^T \cdot x_j}{\ x_i\  \cdot \ x_j\ }$	Used mostly in document similarity clustering, and text mining.	Han et al. (2006). Nguyen et al. (2012) Saxena et al. (2017)	(2.6)
<p>Extended Jaccard</p> $d(x_i, x_j) = \frac{x_i^T \cdot x_j}{\ x_i\ ^2 + \ x_j\ ^2 - x_i^T \cdot x_j}$	Used in word and document similarity clustering.	Strehl et al. (2000)	(2.7)
<p>Normalized Pearson Correlation</p> $d(x_i, x_j) = \frac{(x_i - \bar{x}_i)^T (x_j - \bar{x}_j)}{\ x_i - \bar{x}_i\  \ x_j - \bar{x}_j\ }$	Commonly used in similarity clustering of two variables and in gene expression data clustering.	Jiang et al. (2004) Xu and Wunsch (2005)	(2.8)

## 2.2.4 Clustering Taxonomy

Clustering can be broadly viewed in two major categories; hierarchical clustering algorithms, and partitioning clustering algorithms (Fraley & Raftery, 1998; Rokach, 2005; Celebi et al., 2013; Han et al., 2011; Aggarwal and Reddy; 2014). These two have been widely researched and studied due to its various applications and their simplicity in their implementation (Aggarwal and Reddy; 2014). Hierarchical clustering develops a binary tree-based data structure that can be split at different levels to obtain clustering solutions. Partitioning clustering selects a set of initial clusters (seeds) and iteratively improve the partitioning of the objects to their clusters. Figure 2.1 below shows the taxonomy of clustering techniques.

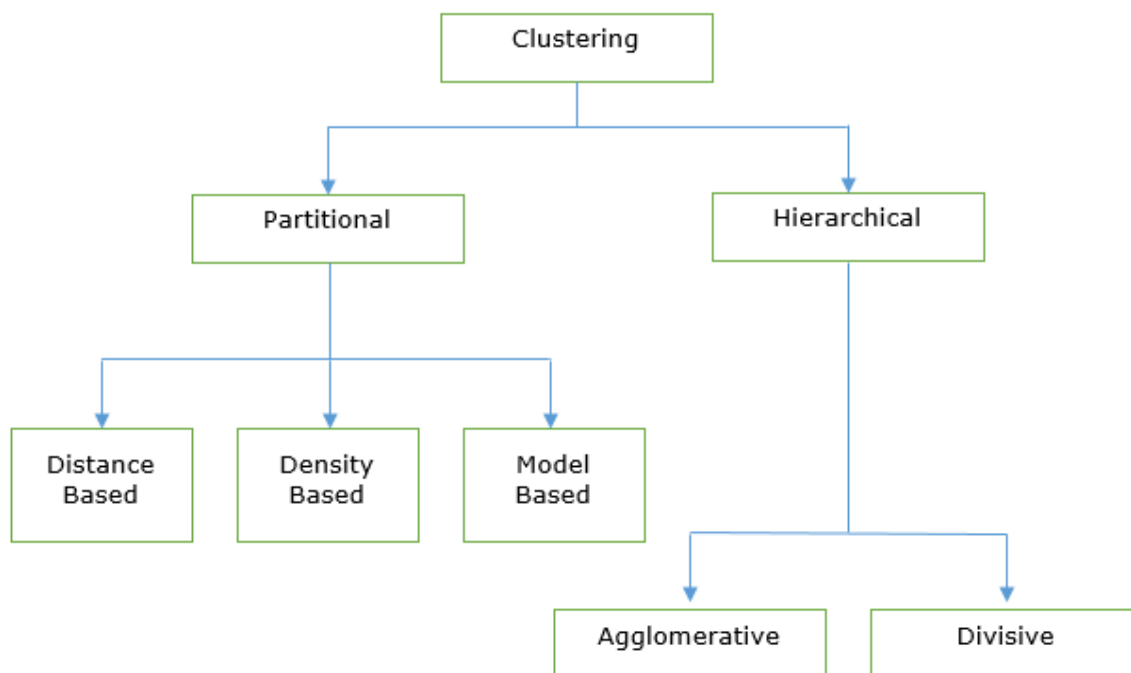


Figure 2.1 Clustering Taxonomy

Brief discussions on each of the clustering categories follows in the subsections below.

## 2.3 Hierarchical Clustering

This category clusters hierarchically following a dendrogram structure and using a similarity criterion either splits or merges the partitions to create a tree-like structure (Berkhin, 2006;



Carlsson and Memoli, 2010; Aggarwal and Reddy, 2014; Vijaya and Bateja, 2017). Hierarchical clustering is either agglomerative that start with a singleton (one-point) and recursively merges two or more clusters from bottom-up, or divisive where it starts with one cluster that contain all data objects and recursively splits them from top-down, until a stopping or termination criterion is attained (Murtagh, 1984; Jain, et al., 1999; Berkhin 2006). Figure 2.2 below shows the two hierarchical clustering dendrogram.

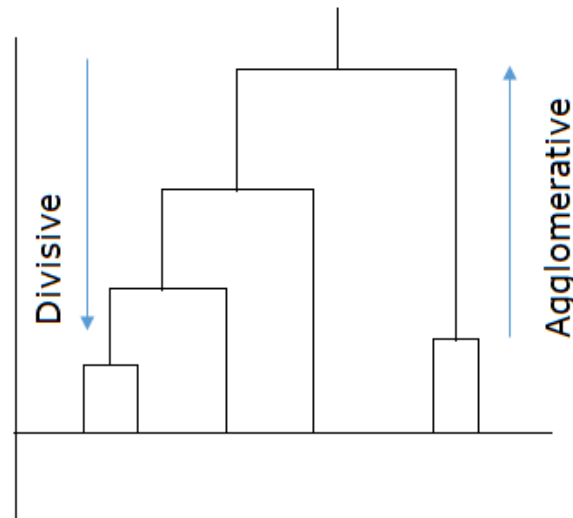


Figure 2.2 Clustering Dendrogram

### 2.3.1 Divisive Hierarchical Algorithms

Divisive hierarchical is a top-down approach where it begins with one root that contains all the data points or the maximal cluster. This root is then recursively considered if it can be split further based on some dissimilarity distance producing a hierarchy similar to a binary tree or a dendrogram. This process is repeated until a singleton is obtained (Roux, 2015).

There are two major considerations needed in using the divisive algorithm that could affect its performance.

Firstly, the splitting method and criteria. As seen above, the technique for splitting the node into two parts is known as *bisecting*. The most widely used algorithm is the Bisecting K-means

(Steinbach et al., 2000) which uses K-means (McQueen, 1967) on the parent cluster  $C$  to determine the best split which maximizes Ward's distance (Ward, 1963) between two possible child clusters  $C_1$  and  $C_2$ . The larger of the split cluster is selected as the new parent for further splitting, and the method is iterated until  $K$  clusters have been obtained.

Secondly, deciding the appropriate cluster to split, i.e., whether the algorithm should choose the cluster with the most significant number of objects or select all clusters at each level. A more compromise alternative is selecting the cluster with the most substantial square error variance.

### **2.3.2 Agglomerative Hierarchical Algorithms**

Agglomerative hierarchical is a bottom-up approach where all data points are represented at the bottom of the dendrogram or binary tree. These points are recorded in a dissimilarity matrix, and the closest sets of clusters are then merged. The dissimilarity matrix is then updated, and the process is repeated where the closest pairs that are less dissimilar are merged bottom-up until one maximal cluster remains that contains all the data points.

The commonly used formula is the Lance-Williams dissimilarity update formula (Lance & Williams, 1967) to compute distance between the clusters by either considering single linkage (nearest neighbour – similarity is that between most similar member), average linkage (group average – considers average pair-wise similarity), and complete linkage (maximal – choosing a cluster pair whose merge has the smallest diameter) (Sneath & Sokal, 1962).

### **2.3.3 Common Classical Hierarchical Algorithms**

One of the primary deficiencies of hierarchical clustering is that they have high computational time complexities making them not suitable for big data. However, hierarchical clustering algorithms have one major advantage in their generation of visual binary trees that make it easy for the users to see the clustering process visually as they are built. The following are some of the classical, popular algorithms that extend the hierarchical clustering techniques:

### **2.3.3.1 CHAMELEON**

Chameleon, by Karypis et al. (1999) is one the most effective hierarchical clustering algorithm. The agglomerative algorithm utilizes graph partitioning methods that use the relative closeness (proximity) and relative inter-connectivity as a determinant to merge two clusters. This technique makes the algorithm very effective in arbitrarily shaped clusters. However, the algorithms major limitation is that the algorithm performs well in low dimensional spaces and not in high dimensional ones (Sexena et al., 2017).

### **2.3.3.2 CURE**

The Clustering Using REpresentative (CURE) algorithm by Guha et al. (1998) represents a cluster by using well-scattered points and then calculates the minimum distance between two representative points in any two clusters selected to merge the clusters. The agglomerative technique of using arbitrary scattered points make CURE effective in arbitrary shaped clusters and robust with outliers.

### **2.3.3.3 ROCK**

The same authors of CURE proposed an algorithm called ROCK (Guha et al., 1999). The algorithm works similar to CURE that handles categorical data. However, a major limitation is that both algorithms have very high time computation complexities.

### **2.3.3.4 BIRCH**

The Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) was proposed by Zhang et al. (1996; 1997). The algorithm produced a data structure called a clustering feature (CF) and CF-tree. The CF contained the number of data points, a linear sum of the data points, and the square sum of the data points in the clusters. The CF of any merged clusters is the sum of the CFs of the two original clusters whereas the CF-tree stores the structure of the entire dataset.

### **2.3.4 Challenges and Limitations of Hierarchical Clustering**

There are a few advantages of hierarchical clustering such as easiness to understand the hierarchical tree (Berkhin, 2006) and the ease of use of different similarity/dissimilarity distance functions (Xu et al., 2005). Major challenges and limitations include:

- i) If misclassification is done, it is very difficult to reassign an object again (Berkhin, 2006).
- ii) Extremely sensitive to outliers (Xu et al., 2005).
- iii) Merge/fusion or split decisions once done are extremely difficult to be undone (Fisher, 1995; Berkhin, 2006).
- iv) Most hierarchical algorithms tend to have quadratic or higher complexity in the number of data points (Chakraborty & Nagwani, 2011; Aggarawal and Reddy, 2014) and thus not very suitable for large data sets.

### **2.4 Partitioning Clustering**

This category divides data into several initial clusters' partitions, and iteratively data is assigned to their closest cluster partition or centroid using a dissimilarity criterion. These produce partitions that can be deemed as hard (where partial memberships are not allowed, and all objects belong to one or more clusters) or fuzzy (where objects may belong to one or more clusters to a certain weight or degree) (Velmurugan and Santhanam, 2011; BenN'Cir et al., 2015; Baadel et al., 2016).

Hard partition algorithms allow all objects to belong to one or more clusters by dividing the data into smaller subsets and unlike hierarchical algorithms, where they do not revisit the points once linked together, gradually improve those clusters by using iterative relocation algorithms. K-mean and its variants split the data into k clusters and represent them by the weighted average, i.e., mean of the centroids. However, the algorithm constraints object to single clusters.

This thesis primarily focuses on distance-based partitioning clustering algorithms and methods that extend K-means based on the squared Euclidean distance that will measure the linear separation between the clusters (for simplicity). This is provided in detail in sections 2.4.3 and the subsequent subsections.

### **2.4.1 Density-based Partitioning Clustering**

This category cluster objects based on a local density criterion where objects are considered densely populated together and are separated by subspaces of low density (Kriegel and Pfeifle, 2005; Kriegel et al., 2011). Sparse areas are often treated as noise data and are pruned from the dataset without assigning them to any clusters (Aggarwal & Reddy, 2014). Some of the major advantages of density-based algorithms are their effectiveness in finding arbitrary shaped clusters. Density-based algorithms identify 3 data points in a dataset namely:

- 1) points that belong to a dense neighbourhood – the core points
- 2) points that do not have a dense neighbourhood but still belong to a cluster – border points
- 3) points that do not belong to any neighbourhood or cluster – noise or outliers

Popular used density-based algorithms include DBSCAN (Ester et al., 1996) and OPTICS (Ankerst et al., 1999) which are briefly discussed below.

#### **2.4.1.1 DBSCAN**

DBSCAN (Ester et al., 1996) is a density-based algorithm that given a fixed-radius threshold, considers two data points to be connected if they both exceed the threshold and are within that neighbourhood radius. All density-connected points form a cluster(s) whereas any data points that do not form clusters are considered noise data and are disregarded. DBSCAN is considered to have a good time complexity of  $O(n \cdot \log n)$  (Xu and Tian, 2015). Some of the variations and implementation of DBSCAN include the X-tree (Berchtold et al., 1996) and the incremental version of DBSCAN by Ester et al. (1998).

### **2.4.1.2 OPTICS**

The OPTICS algorithm (Ankerst et al., 1999) works similar to DBSCAN except that the algorithm stores the clustering order (i.e., the order of the processed data points) as opposed to assigning cluster membership as done in DBSCAN. OPTICS uses a reachability-distance of a point to determine if a point is density-reachable to a core-point (i.e., points in a dense neighbourhood). Those that are reachable then belong to a cluster and those that are not are considered noise data and are disregarded.

### **2.4.1.3 Challenges and Limitations of Density-based Clustering**

While density-based algorithms are considered to have a good time complexity and can find clusters with arbitrary shapes, some of the major challenges include:

- 1) Due to the fixed-radius threshold parameter, it becomes difficult to identify and handle datasets of different densities (Aggarwal and Reddy, 2014)
- 2) When the dataset has uneven density, the clustering result is of low quality (Xu and Tian, 2015).
- 3) By basing on higher density only, some subspaces of clusters cannot be easily distinguishable from the rest of the cluster (Grambeier and Rudolf, 2002)

### **2.4.2 Model-based Partitioning Clustering**

The model-based partitioning clustering algorithms assume that the data objects match a statistical distribution model and those objects are clustered toward that statistical model. The idea is to build a statistical model for each cluster and find one that best fits (Meila and Heckerman, 2001; Fraley and Raftery, 2002). The user specifies the model in the form of parameters allowing the model to change during the learning phase. These models could be either hierarchical or partitional depending on the parameter assumptions entered by the user when running the algorithm. Popular classical model-based clustering includes COBWEB (Fisher, 1987) and AutoClass (Stutz and Cheeseman, 1995).

#### **2.4.2.1 COBWEB**

The algorithm COBWEB (Fisher, 1987) considers each node as a cluster or class that forms into a classification tree and integrates each object incrementally (instead of agglomerative or divisive manner) into the tree based on the best matching node. In this approach, every cluster is described intrinsically instead of it being described as a collection of data points. COBWEB has the ability to adjust the number of clusters without requiring the user to enter this parameter a priori. Due to its incremental nature, COBWEB has a good time complexity (Berkhin, 2006).

#### **2.4.2.2 AutoClass**

The Autoclass algorithm (Stutz and Cheeseman, 1995) that uses the Bayesian learning technique and based on prior distributions (models) determine the optimal number of classes. The user enters the probabilistic distribution (model space) for all the attributes in the dataset, and AutoClass iteratively assigns the objects to a cluster with such probability distribution until there are no more changes in the assignment of objects to the clusters. Thus, the output is usually a number of likely clustering outcomes based on the probability distribution of the attributes. Therefore, any wrong assumptions made by the user will significantly affect the output results.

#### **2.4.2.3 Challenges and Limitations of Model-based Clustering**

While model-based algorithms are considered to have a good time complexity and are able to adjust the number of clusters in its learning phase, some of the major challenges include:

- 1) If user assumptions are false, the model output will be significantly erroneous (Aggarwal and Reddy, 2014)
- 2) Model-based algorithms have a tendency to generate highly unbalanced trees (Berkhin, 2006).

### **2.4.3 Distance-based Partitioning Clustering**

Distance-based methods are generally easy to implement due to their simplicity and can be applied in numerous scenarios. Popular distance-based algorithms include the K-means (MacQueen, 1967), K-Modes (Huang, 1998), CLARANS (Raymond & Han, 2002) and PAM (Partitioning Around Medoids) (Kaufman & Rousseeuw, 1990).

This research focuses on distance-based partitioning clustering algorithms and methods that extend K-means based on the squared Euclidean distance which is fast and have low computational complexity (Borah and Ghose, 2009; Velmurugan and Santhanam, 2010) compared to hierarchical making them suitable for large data sets. In the following subsections, we will discuss the K-means and other overlapping algorithms that extend the K-means.

#### **2.4.3.1 K-Means**

Undoubtedly, K-means is the most widely used and most straightforward partitioning clustering algorithm (Jain, 2010). There are many variations of K-means algorithms that are designed to solve different clustering problems in the literature. Some of the variations of K-means include K-Medoids (Mirkin, 2005; Sheng and Liu, 2006), Fuzzy K-Means (Dunn, 1973; Bezdek, 1981), Genetic K-means (Krishna and Murty, 1999), Bisecting k-means (Steinbach, et al., 2000), K-means++ (Arthur and Vassilvitskii, 2007) and X-means (Pelleg and Moore, 2000) among others. There are many reasons attributed to why the K-means algorithm is so popular such as;

- a) it is very easy to implement,
- b) very versatile in that any part of the algorithm can be easily modified,
- c) it is guaranteed to converge (Selim and Ismail, 1984) at a quadratic rate (Bottou and Bengio, 1995).

The algorithm begins with first assigning K points as the initial cluster centres. This can be done randomly or by based on heuristics. Data objects are then assigned to their nearest cluster centre by calculating the distance using Euclidean distance measurement. The sum of squares



errors (objective function) is then calculated by squaring the Euclidean distances to each cluster centroid, and the object is assigned to the cluster with the smallest value. The recalculation of the centroids is taken as the average of the values of the objects that are part of that cluster.

Input: A vector  $x_1, x_2, \dots, x_n$ ,  $k$  number of clusters

Output:  $k$  clusters

procedure K-means

```
{
    1. Randomly select initial  $k$  number of centroids,  $C_1, C_2, \dots, C_k$ 
    2. Repeat
    3.     Assign each point to the closest centroid to form a cluster
    4.     For  $i = 1, i++, i=k$ 
    5.         Recalculate the mean for each cluster centroid
    6.         Replace  $C_i$  with the mean of all the samples in cluster  $i$ 
    7.     End for
    8. Until convergence criteria is met
}
```

Figure 2.3 K-means algorithm

These steps are then iterated in a loop until the objects in each cluster do not change or until a maximum number of iterations are reached. Figure 2.3 above and figure 2.4 below provide a sample of the K-means algorithm and its corresponding flowchart.

The K-means algorithm aims at minimizing the objective function defined in the Equation (2.9) below for the given set of centroids.

$$C = \sum_{k=1}^K \sum_{x_i \in C_k} ||x_i - c_k||^2 \quad (2.9)$$

Where  $C_k$  is the  $k$ th cluster,  $x_i$  is a point in  $C_k$ , and  $c_k$  is the mean of the  $k$ th cluster.

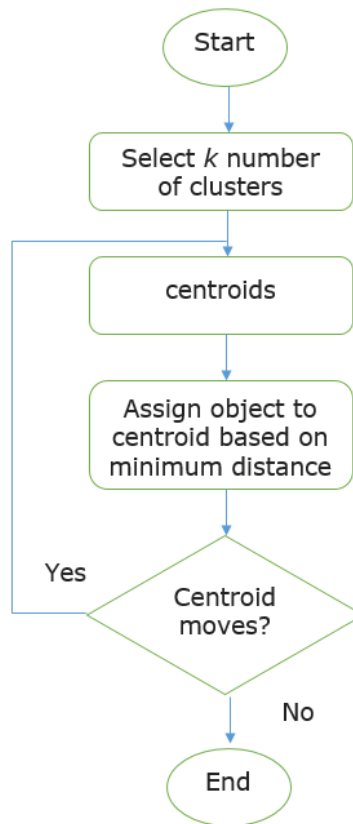


Figure 2.4 K-means flowchart

K-means clustering being a greedy-descent nature algorithm will converge to a local minimum (Selim & Ismail, 1984). For up to two-dimensional Euclidean space with an arbitrary number of  $k$  clusters, the K-means complexity is NP-Hard (Velmurugan and Santhanam, 2010; Manning et al., 2008; Everitt et al., 2001).

### **2.4.3.2 Challenges and Limitations of K-Means Algorithm**

There are a few challenges and limitations to the K-means algorithms namely:

1. Choosing the initial centroids
2. Estimating the number of  $k$  clusters
3. Works on numerical datasets only
4. Sensitivity to outliers

## **A. Choosing the initial centroids**

Choosing the initial clusters largely affects the outcome of the algorithm (Jain et al., 1999; Bradley and Fayyad, 1998; Khan and Ahmad, 2004; Aggarwal & Reddy, 2014). This can be done randomly by picking  $K$  points as  $K$  centres as suggested in (MacQueen, 1967). Forgy (Forgy, 1965) suggested spreading the randomness of the initial location of the clusters. The idea being that this random selection is likely to pick points from the dense regions which may be good centres. However, by randomly selecting objects does not eliminate the possibility of picking an outlier for a centre. This can be minimized by having multiple runs of this method. Other heuristic approaches that have been proposed for cluster initialization includes Ward's distance method (Ward, 1963) that uses the sum of squared errors to evaluate between two cluster distances as suggested in (Milligan, 1981). Kaufman's method (Kaufman and Rousseeuw, 1990) selects the  $K$  centres sequentially by first choosing the most centrally located data object in the data set and subsequently choosing the other centres that have many data objects around it by a heuristic function. K-means++ (Arthur & Vassilvitskii, 2007) selects the first centroid randomly and subsequently chooses the next centroid which is farthest from the currently selected centroid based on a weighted probability score. Bradley & Fayyad (1998) method randomly partitions the set into  $J$  subsets which are then combined into a superset clustered in by k-means  $J$  times initialized each time with a different centre. The centre set that gives the least SSE are considered the final centres.

## **B. Estimating the number of $k$ clusters**

As for estimating the number of  $K$  clusters in advance is not a simple feat considering that no prior knowledge of data is provided (Wagner et al., 2005). The ISODATA algorithm (Ball, 1965) was one attempt in determining the optimal  $K$  where K-means s first ran on the dataset to obtain the clusters which these clusters are then merged if the distance between them is less than a particularly given threshold or split if the standard deviation within the cluster exceeds the same threshold. The Silhouette Coefficient algorithm (Kaufman, 1990) takes into account the inter and

intra-cluster distances for any given data object. The average  $a_1$  of the distances is calculated for all points intra cluster for a given point  $x$ . The average  $a_2$  is then calculated for all other inter clusters that do not contain point  $x$ . These two values  $a_1$  and  $a_2$  are then used to estimate the Silhouette Coefficient of point  $x$ . The average of all the silhouettes becomes the width for all the dataset points. Other methods include the Gap statistic method (Tibshirani et al., 2001) that estimate the number of clusters using gap statistic, Calinski-Harabasz index (Calinski & Harabasz, 1974) among others.

This goes to show that estimating the number of clusters is non-trivial and is a major challenge for the K-means algorithm.

### **C. Works on numerical datasets only**

K-means also has a shortcoming in that it only works for numerical data and not categorical data. The K-mode clustering algorithm (Xu & Wunsch, 2005), a variation of the K-means, works for categorical data sets. The K-mode algorithm differs from K-means in that it uses modes instead of means to calculate the centroids and measures dissimilarity between the categorical data instead of the Euclidean distances between the objects.

### **D. Sensitivity to outliers**

Another drawback for the K-means algorithm is that it is very sensitive to outliers. Since all data objects must be assigned to a cluster, an outlier can easily affect the mean of the data objects in that given centroid. K-Medoids algorithm (Mirkin, 2005; Sheng and Liu, 2006) addresses this problem by choosing the actual data points as the cluster prototypes. The K-Medoid algorithm is further improved by the Partitioning Around Medoids (PAM) algorithm (Kaufman & Rousseeuw, 1990) and the Clustering LARge Applications (CLARA) algorithm (Raymond & Han, 2002).

The algorithm uses the actual data objects as prototypes and randomly assigns an object  $x$  to replace an object  $y$  which is represented in the cluster prototypes. Once this is done, the membership of all data points that belonged to the representative  $y$  are checked and if they are closer to  $x$  then  $y$  is swapped with  $x$ . The cost of swapping is computed as the absolute error

criterion for K-Medoids and is recalculated for every assignment of  $x$  and  $y$  as it obtains the final representative points for each cluster. This fact makes K-medoid computational complexity higher than that of K-means and thus not very suitable for big data sets.

Further discussion on this topic is provided in Section 2.4.5

#### **2.4.4 Common Overlapping Distance-based Partitioning Clustering**

Many clustering algorithms are hard clustering techniques where an object is assigned to a single cluster. Overlapping partitioning clustering methods tend to relax or remove the constraints allowing overlaps between clusters to better fit any hidden structures in the data and assign data objects to one or more clusters building a non-disjoint partition of the data (Ben N'Cir et al., 2015). This has several applications in real-life such as dynamic system identification, document categorization (a document belonging to different clusters), data compression, bioinformatics, image recognition, and classification, model construction, etc. among others (Höppner et al., 1999; Bandyopadhyay and Maulik, 2002; Aggarwal and Reddy, 2014).

There are several overlapping clustering algorithms which are graph-based clustering algorithms (Jonker et al., 2001). Overlapping graph-based methods use greedy heuristics and may be applicable to community detection in complex networks (Fellows et al., 2011). However, it is worth mentioning that these algorithms have major limitations that do not make them practical for real-life problems as outlined by Pérez-Suarez et al. (2013) and are thus out of the scope of this research. Some of the mentioned limitations indicated are:

- i) They produce a large number of clusters in that analysing these clusters could be as difficult as analysing the whole collection.
- ii) There is a very high overlapping in the clusters which would essentially hinder getting useful information about the structure of the data.
- iii) They have a very high computational complexity thus making them unrealistic to apply them to real-life problems.

Extensions of the K-means that allow partitioning overlaps include Fuzzy K-means (Bezdek, 1981), Kernel Overlapping K-means (KOKM) by (Ben N'Cir et al., 2010; Ben N'Cir and Essoussi, 2012), Overlapping K-means (OKM) by Cleuziou (2008), Parametrized R-OKM (Ben N'Cir et al., 2013) and Multi-Cluster Overlapping K-Means Extension (MCOKE) by Baadel et al. (2016).

#### **2.4.4.1 Fuzzy K-Means**

One of the commonly used soft-clustering techniques is the Fuzzy K-means (Bezdek, 1981) commonly referred to as Fuzzy C-means (FCM) (Bezdek et al., 1984). The algorithm works similar to the K-means where the algorithm minimizes the objective function (sum of squares error) until the centroid converges (Pedrycz, 2002). Other algorithms that are a variation of FCM to deal with non-numerical data sets include Fuzzy K-mode and Fuzzy K-medoid. Some extensions of FCM include Possibilistic C-means (Krishnapuram & Keller, 1996; Pal et al., 2005).

The algorithm works similar to that of K-means and the solution will correspond to the local minimum of the objective function. The sum of squared errors (SSE) objective function is defined in the Equation (2.10) below:

$$C = \sum_{k=1}^K \sum_{x_i \in C_k} w_{xik}^{\beta} \|x_i - c_k\|^2 \quad (2.10)$$

Where  $C_k$  is the  $k$ th cluster,  $x_i$  is a point in  $C_k$ ,  $c_k$  is the mean of the  $k$ th cluster and  $w$  is the membership weight of point  $x_i$  belonging to cluster  $C_k$ .  $\beta$  controls the fuzziness of the memberships such that when it approaches one it acts like k-means algorithm assigning crisp memberships.

The algorithm minimizes this SSE iteratively and updates the membership weightage and clusters until convergence criteria are met or improvement over the previous iteration does not meet a certain threshold. By assigning the memberships a weightage degree between 0 and 1, the objects are able to belong to more than one cluster with a certain weight hence generating soft partitions or clusters. The overall weight, however, must add to unity, i.e., 1. Objects are

eventually assigned to clusters that have the highest degree of membership. If the highest degree of membership is not unique, then an object is assigned to an arbitrary cluster that achieves the maximum. By adding a constraint where the data object must belong to a cluster with the highest membership degree, a "1" is imposed on every object in the matrix thus degenerating it to hard-partitioning.

#### **2.4.4.2 Overlapping K-Means (OKM)**

The algorithm proposed by (Cleuzious et. al, 2008) initializes a random cluster prototype with random centroids as an image of the data. Optional threshold value can be entered by the user during the initialization step. The aim is to minimize the objective function given in the Equation (2.11) below.

$$J(\{\pi_c\}_{c=1}^k) = \sum_{i=1}^N ||x_i - \phi(x_i)||^2 \quad (2.11)$$

Where  $\pi_c$  represents the  $c^{th}$  cluster with  $x_i \in \mathbb{R}$ .

After calculating the SSE of the data objects to their centres using the Euclidean square distance, it assigns these objects to their nearest centroids. The algorithm then computes the SSE of the prototype and compares these objects with the prototype centre assignments to determine the mean of the two vectors to become the threshold to assign the objects to multiple clusters. Once the initial assignment of objects to their centroids is done, the mean between each cluster (threshold) is used to determine if the object should belong to the next nearest cluster as well. OKM uses a heuristic to determine the set of possible assignments by sorting the clusters from nearest to furthest and assigning the object to the nearest cluster. If the mean  $m_x$  of the clusters already associated with the object plus the mean  $m_y$  of the next nearest cluster is lower than the threshold (mean of all the clusters associated with the object), then these two clusters are associated, and the object will belong to that cluster as well. This assignment procedure is iterated until the stopping criteria, or the maximum number of iterations is met resulting in new coverage of the data objects in multiple clusters.

#### 2.4.4.3 Weighted Overlapping K-Means (WOKM)

The WOKM is an extension of the OKM and Weighted K-means (Huang et al., 2005) that introduces a weighting vector  $\lambda_c$  of a subset of attributes relative to a given cluster  $c$  that may be assigned to that cluster and a vector  $\gamma_i$  of weights relative to the representative  $\phi(x_i)$  with the aim of minimizing the objective function given by Equation (2.12) below.

$$J(\{\pi_c\}_{c=1}^k) = \sum_{x_i \in X} \sum_{v=1}^P \gamma_{i,v}^\beta |x_{i,v} - \phi_v(x_i)|^2 \quad (2.12)$$

The objective function is optimized by first assigning each data object to the nearest cluster while minimizing the error, and secondly by updating both the cluster representatives and the set of cluster weights. In this algorithm, the distance feature is also weighted by the feature weights contrary to the standard K-means which ignores the weights of any particular feature and considers all of the features to be equally important.

#### 2.4.4.4 Kernel Overlapping K-Means (KOKM)

The Kernel Overlapping K-means (KOKM) algorithm by BenN'Cir & Essoussi, (2012) and BenN'Cir, Essoussi, & Bertrand, (2010) is a variant of OKM that utilizes the use of kernel methods for overlapping clustering. The authors use two variants in their method; one is a kernelization of the Euclidean metric, similar to the one used in OKM, that calculates the distances between the objects and the clusters in a high dimensional mapping space; the second variant performs all the clustering steps where data is implicitly mapped.

The Parameterized R-OKM by BenN'Cir et al., (2013) algorithm is another variant of OKM that lets users regulate the overlaps via a parameter. The algorithm is based on the minimization of the objective function given by Equation (2.13) below.

$$J_{R-OKM}(\Pi, C) = \sum_{i=1}^N |\Pi_i|^\alpha \cdot \|x_i - (\bar{x}_i)\|^2 \quad (2.13)$$



where  $\alpha \geq 0$  and is set by the user.

As the size of the parameter  $\alpha$  increases, the algorithm builds cluster with reduced overlaps, and vice-versa when the size of the parameter approaches zero. The PR-OKM algorithm is reduced to OKM when this parameter is set to exactly zero.

#### **2.4.4.5 Multi-Cluster Overlapping K-Means Extension (MCOKE)**

The MCOKE algorithm introduced by Baadel, et al. (2016) consists of two procedures. The first part is the standard K-means clustering that iterates through the data objects in order to attain a distinct partitioning of the data points given a priori number of  $k$  clusters by minimizing the distance between the objects and the cluster centroids. The algorithm is based on the minimization of the objective function given by Equation (2.14) below.

$$J = \sum_{i=1}^C \sum_{x_i \in \mu_i} d(x_i, v_i) \quad (2.14)$$

Where  $v_i$  is the centre of cluster  $\mu_i$ , and  $d(x_i, v_i)$  is the Euclidean distance between a point  $x_i$  and  $v_i$ . The second part creates a membership table that compares the matrix generated after the initial K-means run to *maxdist* (the maximum distance an object allowed to belong to a cluster). Further details of this algorithm and its improved Outlier detection version are discussed in Chapters 3 and 4.

#### **2.4.4.6 Challenges and Limitations of Overlapping Distance-based Partitioning Clustering Algorithms**

All overlapping distance-based partitioning clustering that extend the K-means algorithm inherit the same limitations and challenges facing the parent K-Means algorithm. These include choosing the initial centroids, estimating the number of  $k$  clusters, sensitivity to outliers, and their ability to work only on numerical datasets. These are all described in section 2.4.3.2 above.

One other major limitation of overlapping clustering algorithms is their abilities to identify objects at the borderline that are a bit sparse from other cluster centroids. These objects tend to be cut from the dataset since the algorithms discard them as outliers. The algorithms do not have the capability of storing the outliers.

With the exception of OMCOKE, the overlapping clustering algorithms require users to set the number of labels in the dataset to be used by the algorithm prior to running them. This is by no means an easy feat especially when details of the dataset is not known in advance.

#### **2.4.5 Outlier Detection in Partitioning Clustering**

Outliers are data objects or points that do not conform to the normal behaviour or model of the dataset, hence are deemed inconsistent or grossly different (Berkhin, 2006). This data can be erroneous, but could also be classified as suspicious data in fraudulent activity; that could be useful for fraud detection, intrusion detection marketing, website phishing sites, etc.

Outlier detection is considered a task in itself; research in the data mining domain has focused on an efficient and optimal way to detect distance-based outliers. Outlier detection surveys such as by Chandola et al., (2009), Bay & Schwabacher, (2003), and Kadam, & Pund, (2013), discussed several approaches used to tackle anomalies and noise data. In Ramaswamy, Rastogi, & Shim, (2000), the authors provide methods that would handle mine outliers efficiently in large datasets. Other recent studies have devised methods in clustering analysis that will prune or screen out outliers from the dataset such as Liu, Wu & Fu, (2018), Barai & Dey, (2017), and Gan & Ng, (2017). For example, Yu, Luo, Chen, & Ding, (2016) proposed an outlier detection method to identify and eliminate outliers in the dataset forming an outlier-eliminated dataset (OED). The authors then applied the K-means algorithm on the OED, thereby improving the accuracy of the clustering.

Similarly, the Barai & Dey, (2017) approach is to divide their algorithm into two steps. The first step calculates the threshold value used in detecting outliers by taking the average of the maximum and minimum values of the pairwise distance of all data. Each data point is then reiterated and compared to the threshold. Those that have a distance value greater than the threshold are deemed as outliers and are subsequently tossed out of the dataset. The second step then runs the K-means algorithm without outliers, thus improving the clustering process.

Liu et al., (2018) also propose a two-phase approach for their clustering with the outlier removal (COR) algorithm. In the first phase, their method runs the K-means algorithm to generate primary partitions and discover outliers. The outliers here are identified as objects with large distances to their nearest centroid. The second phase removes the identified outlier objects, and the remainder is partitioned into  $k$  clusters.

While many studies focus on pruning and discarding the outliers to improve the classification process, rarely do we find algorithms that detect outliers simultaneously while performing clustering (Gan & Ng, 2017). The K-means with outlier removal (KMOR) algorithm is similar to the standard K-means algorithm but introduces an outlier cluster ( $k+1$ ) that takes into account objects that do not fit in the  $k$  defined clusters. The algorithm identifies outliers as objects that are above a calculated threshold which is defined by the average distance multiplied by a specific parameter greater or equal to 0. The average distance is calculated during the clustering phase. The KMOR algorithm requires three parameters such as the  $k$  number of clusters, the maximum number of outliers  $n_0$  (to control the number of objects being assigned as outliers), and finally, the third parameter to classify outliers and those that are not. Two additional parameters are used to help terminate the algorithm. As noted above, the KMOR algorithm requires users to define the maximum number of outliers, including a parameter to classify the outliers and those that are not. This is impractical in real-life scenarios in unsupervised datasets where no prior knowledge of the data is given. Also, their method requires additional parameters to help terminate the algorithm. This is not an easy feat to be determined by novice users.

Unlike other algorithms that prune the outliers and discard them, our algorithm OMCOKE (discussed in detail in Chapter 3 and 4) saves them on a newly created outlier cluster during the iteration process. Our study considers the same idea as the KMOR algorithm and introduces an outlier cluster  $k+1$  that stores the anomalies or outlier objects separately from the regular instances. Our algorithm does not require users to enter parameters to terminate the algorithm or to identify the maximum number of outliers in the dataset; this makes it more practical in machine learning. None of the overlapping K-means algorithms above have the capability to detect outliers and store them for additional scrutiny. Thus, we provide additional value to the literature by introducing this new overlapping clustering method.

#### **2.4.6 Classification Techniques used in the Thesis Case Study**

Since this thesis is about overlapping clustering techniques, we briefly review hereunder some of the classification techniques that are used in the case study chapter (Chapter 5). The classification techniques were used to measure the proposed framework in the case study. The five classification techniques used were RIPPER (Cohen, 1995), PART (Frank & Witten, 1998), Random Forest (Breiman, 2001), Random Trees (Cutler & Zhao, 2001), and Artificial Neural Network [ANN] (Witten & Frank, 2005)

Repeated Incremental Pruning to Produce Error Reduction (RIPPER) is a classification technique that divides the dataset by looking at the least frequent class set and building it by adding attribute values to its body. While RIPPER adds the attributes to the body and generates a rule, it simultaneously prunes some of the rules to reduce redundancy.

PART is a classification technique that uses a mathematical formula called the information gain (IG) to build partial decision trees from the input dataset which are then converted into rule sets using a rule induction strategy. The partial tree (sub-tree) is discarded from the dataset once a path

leading to its leaves is converted into a rule. The process is repeated until all instances in the dataset are removed and the dataset becomes empty.

Artificial Neural Network (ANN) is a technique that uses inter-connected processing components commonly referred to as neurons to convert desired output from a given input dataset. The neural network (output) heavily relies on hidden nonlinearity of the neurons features and weights in the training phase of the classifier to form feature classes that are based on the network's connectivity which is then used in the classifier predictive model.

The random tree (RT) algorithm applies a bootstrap aggregating mechanism that combines classifications of randomly generated set of data for constructing a decision tree. Each node in a RT is split based on the best subset features selected at that node. In a RT, every leaf defines a linear model optimized for that leaf. Instead of computing the best split for any given node in a tree, the algorithm considers a random subset of all data attributes for that particular node to determine the best possible split allowing a reasonably balanced tree.

The random forest (RF) algorithm uses an ensemble of random trees where the value of each tree is calculated independently by using random features to split each tree node to generate classes in the training phase of the classifier. Each tree in the training data is considered as a base classifier used to determine class labels from the unlabelled data, and the mode of all the classes is used as the output to predict.

## **2.5 Chapter Summary**

In data mining and machine learning, clustering of big data is achieved through various techniques. In this chapter, we critically reviewed the two common clustering techniques in ML namely hierarchical and partitioning clustering algorithms. In particular, we analysed the distance-based partitioning algorithm K-means and looked at some of the overlapping algorithms that extend the K-means algorithm. Furthermore, we outlined some of the challenges affecting the K-means algorithm and a bit of focus on the outlier detection issue.

In the next chapter, we propose our overlapping algorithm MCOKE (Multi-Cluster Overlapping K-Means Extension) and discuss some of the strengths and weaknesses of the algorithm. We further propose an improved version of the algorithm that handles outliers.

## Chapter 3

# Outlier based Multi-Cluster Overlapping K-means Extension (OMCOKE)

### ***3.1 Introduction***

Many clustering algorithms in unsupervised learning constrain objects to single clusters (i.e., objects belong to exactly one cluster) while ignoring the fact that some objects may have attributes that can belong to more than one cluster. Undoubtedly, K-means is the most widely used partitional clustering algorithm (Jain, 2010). There are many reasons attributed to this such as; a) it is straightforward to implement, b) very versatile in that any part of the algorithm can be easily modified, c) it is guaranteed to converge (Selim and Ismail, 1984) at a quadratic rate (Bottou and Bengio, 1995). Thus, the algorithm has been used extensively to solve non-overlapping clustering problems.

Overlapping clustering methods remove the constraints and assign objects to one or more clusters building a non-disjoint partition of the data. Overlapping clustering algorithms such as the popular Fuzzy C-Means (FCM) require a particular threshold value be set a priori to determine the cut-off for an observation to belong to multiple clusters. For example, in fuzzy algorithms, if an object exceeds the predefined threshold, then it can be assigned to multiple clusters. However, if the nature of the dataset is not known in advance and the threshold is set to a large number, then observations that have membership values lower than the threshold may not be assigned to any cluster at all since the clusters are sensitive to the threshold values. K-means extended algorithms have one major drawback in that they are very sensitive to noise data. Noise data can be considered as outliers in the dataset and do affect the minimization of the objective function of the sum of squared errors (SSE).

This chapter proposes a new clustering method that extends the K-means algorithm that will assign observations to multiple clusters without the need for pre-defining the minimum belonging threshold. Whenever the algorithm is run, it calculates the maximum distance of any observation that belongs to a given cluster and uses that as the default threshold of belonging. This calculated distance is assigned to local variable *maxdist* and is henceforth used to assign the observations that have a distance lower than it to multiple clusters. The major benefit of this is that users do not have to pre-define the minimum threshold belonging a priori as this will be calculated in the algorithm and will change depending on the nature of the dataset.

The chapter also proposes an enhancement of the MCOKE algorithm that is able to detect the outliers (noise data). While the calculation of *maxdist* can be affected by the inclusion of outliers in the first round of the iteration, we introduce another variable that calculates the average distance (*averdist*) between the object and the centroid for all clusters. *Averdist* acts as a new threshold for the inner radius between the object and the centroid. Another variable *maxdistThreshold* defines the outer radius distance to be considered as the outer boundary, which becomes the cut-off point of objects that are deemed outliers. Once one or more objects have been identified as outliers, the algorithm assigns them together in one cluster, known as the Outlier cluster ( $k+1$ ). The major advantage of this is that instead of pruning and discarding these objects, we save them in order to study the noise data. For example, in real-life scenarios such as in a data security environment, noise data could mean an intrusion attempt.

The proposed clustering method MCOKE was first implemented in a web-based application using JavaScript in order to evaluate proof of concept. OMCOKE has been implemented in Java within the Waikato Environment for Knowledge Analysis (WEKA) environment. WEKA is an open-source application that can be downloaded for free and may benefit different stakeholders such as students, researchers, or managers that are interested in data analysis.

We have conducted experiments using real-life multi-label datasets from Mulan: A Java Library for Multi-Label Learning repository (more details in Chapter 4) indicate that the proposed



OMCOKE algorithm is better in its Precision accuracy compared to some of the commonly used overlapping clustering algorithms.

The rest of this chapter is structured as follows: the proposed algorithm is discussed in section 3.2. Section 3.3 details the enhancements made to the algorithm. Section 3.4 provides a comprehensive example of the different steps taken by OMCOKE algorithm in the clustering process. OMCOKE features compared to other overlapping clustering algorithms is given in sections 3.5 followed by the chapter summary in section 3.6 respectively.

### **3.2 MCOKE**

The proposed MCOKE algorithm consists of two phases. The first phase is the standard K-means clustering that iterates through the data objects in order to attain a distinct partitioning of the data points. The algorithm begins with first assigning  $k$  points as the initial cluster centres. This can be done randomly or based on heuristics. Data objects are then assigned to their nearest cluster centre by calculating the distance using Euclidean distance measurement. The sum of squares errors (objective function) is then calculated by squaring the Euclidean distances to each cluster centroid, and the object is assigned to the cluster with the smallest value. The recalculation of the centroids is taken as the average of the values of the objects that are part of that cluster. These steps are then iterated in a loop until the objects in each cluster do not change or until a maximum number of iterations are reached.

K-means clustering being a greedy-descent nature algorithm, it will converge to a local minimum (Selim & Ismail, 1984) with an arbitrary number of  $k$  clusters. The K-means algorithm does require two things a priori from the users. One is first to choose the initial centroids, and two, to estimate the number of  $k$  clusters in advance. Choosing the initial clusters largely affects the outcome of the algorithm. This can be done randomly by picking  $k$  points as  $k$  centres as suggested in (MacQueen, 1967). Forgy (Forgy, 1965) suggests spreading the randomness of the initial location of the clusters. The idea being that this random selection is likely to pick points

from the dense regions which may be good centres. However, this does not eliminate the possibility of picking an outlier for a centre. This can be minimized by having multiple runs of this method. Other heuristic approaches that have been proposed for cluster initialization includes Ward's distance method (Ward, 1963) that uses the sum of squared errors to evaluate between two cluster distances as suggested in (Milligan, 1981).

Let  $X = \{x_1, x_2, x_3 \dots x_n\}$  be the data,  $c_l = \sum_{x \in C_k} \frac{x}{n}$  be the cluster  $C_l$  centroid,  $n$  be the total number of data objects in the cluster  $C_l$ , and  $k$  be the number of clusters. The objective function of K-means will be defined in the Equation (3.1) below.

$$SSE = \sum_{l=1}^K \sum_{x \in C_k} ||x - c_l||^2 \quad (3.1)$$

As we minimize the K-means objective function during the iteration process, it maximizes the distance function (Xiong et al., 2009). The MCOKE algorithm records the maximum distance of any of the data object belonging to a cluster centroid in a local variable called *maxdist*. In each iteration, the algorithm re-computes the cluster centroids to a more sensible location until the centroids do not change. Steps 1 through 9 of figure 3.1 below illustrates this process.

The variable *maxdist* is used in our algorithm as the global membership threshold to calculate the belonging of observation to a different cluster other than the one initially assigned by the K-means algorithm. For example, if an object is assigned to cluster 1 has a membership distance to cluster 2 shorter than the saved *maxdist* value, then that object may also belong to cluster 2. This comparison is made by iterating through all the objects and re-assigning them new belongings and achieving the multi-cluster assignments which are done in phase 2 of MCOKE which is illustrated in steps 11 through 16 of the pseudocode in figure 3.1 below.

After an initial run of the first step (that includes steps 1 to 9 below), the algorithm will return three things. Firstly, a vector of all the data objects with their assignment to each cluster.

Secondly, a vector containing the final list of the cluster centroids. This vector of all centroids will be used in the second part of the algorithm to determine if the objects should belong to them. Thirdly, the *maxdist* as determined by the Euclidean distance of the objects to the centroids is made the global threshold to compare the similarity of the objects to other clusters.

Input: A vector  $x_1, x_2, \dots, x_n$ ,  $k$  number of clusters

Output: *membership matrix*

procedure MCOKE

```
{
    1. Randomly select initial  $k$  number of centroids,  $C_1, C_2, \dots, C_k$ 
    2. Repeat
    3.     Assign each point to the closest centroid to form a cluster
    4.     For  $i = 1, i++, i=k$ 
    5.         Recalculate the mean for each cluster centroid
    6.         Replace  $C_i$  with the mean of all the samples in cluster  $i$ 
    7.         Update maxdist with highest distance of object assigned to centroid
    8.     End for
    9. Until convergence criteria is met
    10. Draw initial membership matrix
    11. Repeat
    12.     Compare the distance of each object assigned to centroid with maxdist
    13.     If distance to another centroid  $<$  maxdist
    14.         Assign object to that cluster as well
    15.     End if
    16. Until all objects in dataset are traversed
}
```

Figure 3.1 MCOKE Pseudocode

The second part of the algorithm draws an initial membership matrix table with hard clustering result of the data objects. The algorithm softens these partitioning by iterating through the membership matrix and comparing the objects to the final centroids vector using the threshold *maxdist* and reassigning them to the clusters if the distance of the object to those centroids is less than *maxdist*. Let the vector results produced from the K-means algorithm generate a

membership table MT (of dimension  $N \times C$ ) such that  $MT(i,j)$  denotes a member of object  $i$  to cluster  $j$  where  $i = 1, \dots, N$  and  $j = 1, \dots, C$ . Each object in  $MT(i,j)$  is assigned a 1 (one) to denote membership to that cluster and a 0 (zero) for non-membership of a cluster. The algorithm then iterates through the table MT and compares the distance of the objects assigned to their respective clusters with the other final centroids in the table. If the object distance is less than the *maxdist*, then that object is also assigned to that cluster centroid, and the membership table is updated with a 1 (one).

Since the MCOKE algorithm utilizes the K-means in the first phase, it will suffer a major drawback in that the objective function of K-means is designed to optimize while under the constraint of assigning the data objects to non-overlapping partitions. This means that all objects will be assigned to at least one cluster including noise or outliers which may then affect *maxdist* as a good predictor to overlap other objects in multiple clusters.

### **3.3 OMCOKE**

Outliers are data objects or points that do not conform to the normal behaviour or model of the dataset, hence are deemed inconsistent or grossly different (Berkhin, 2006).

The proposed method is an enhancement of the MCOKE algorithm that allows objects to overlap and belong to more than one cluster based on their distance comparison to the *maxdist* variable. *Maxdist* calculates the largest distance of any object assigned to any centroid during the partitioning phase for it to belong to a particular cluster. That distance is used as an outer radius of similarity threshold and as the benchmark to allow objects to belong to other clusters that were not initially assigned to them, allowing them to overlap. However, K-means, being a greedy algorithm, guarantees all objects to be assigned to a cluster including any outliers; hence the *maxdist* radius benchmark could easily be influenced by outliers.

We introduce another variable that calculates the average distance (averdist) between the object and the centroid for all clusters. Averdist acts as a new threshold for the inner radius between the object and the centroid.

$$averdist = \frac{1}{n_i} \sum_{x_i \in C_k} ||x_i - C_k||^2 \quad i = 1, 2, \dots, K \quad (3.2)$$

Where  $C_k$  is the  $k$ th cluster,  $x_i$  is a point in  $C_k$ .

It is assumed that most objects being clustered will fall close to the inner radius threshold (i.e., close to their cluster centroid) that is based on the average distance of all objects belonging to the cluster centroids. Anomalies or outliers, therefore, tend to be further away from their closest cluster centroid. Objects that have a distance greater than the inner radius but less or equal to the outer radius (maxdist) are subject to further scrutiny and are flagged to ensure they are not outliers on the border of the clusters. Therefore, the maxdistThreshold defines the outer radius distance to be considered from the outer boundary, for example, 0.98 will mean the area covered inside the outer boundary for objects is not to be considered an anomaly.

This logic is based on the assumptions that:

- a) Anomalies tend to be in sparse clusters, whereas normal instances usually belong to dense clusters
- b) Anomalies tend to be far from the closest cluster centroid, whereas normal instances tend to be near their closest cluster centroid.

In cases where some knowledge of the data is known beforehand, this value can also be adjusted by the user before running the algorithm.

This modification logic is summarized in the pseudocode provided below.

```
1. For each  $x_i \in C_k$ 
2. Do
3. If ( $\text{dist}(x_i, \text{centroid } C_k) \leq \text{averdist}$ )
4.   Cluster  $\leftarrow x_i$ 
5. Else
6.   If ( $\text{dist}(x_i, \text{centroid } C_k) \geq \text{maxdist} * \text{maxdistThreshold}$ )
7.     Outlier_Cluster  $\leftarrow x_i$ 
8.   Else
9.     Cluster  $\leftarrow x_i$ 
10.  End if
11. End if
```

Figure 3.2 Outlier detection Pseudocode

In Step 6 of the code above, the area covered by the `maxdistThreshold` is multiplied by the `maxdist`, calculated as a percentage of the overall maximum distance for any object belonging. This acts as the cut-off point, and any object that has a distance value greater than the upper percentile of this value is deemed an outlier. Upon identification of at least one outlier, the `k` number of clusters entered by the user prior to running the method is incremented by 1 on the fly; the outlier object is assigned to this newly created cluster. All other identified outliers, a subset `S` from the initial population, are assigned to belong to this newly created cluster. Once an outlier is detected, the algorithm adds `k+1` clusters as the new output vector with the outlier cluster indexes listed as part of the output. This allows for further investigation of those data points as opposed to discarding them as is usual. When no outliers are detected, the algorithm will cluster with overlaps without incrementing the number of `k` clusters.

Our approach adds immense value to the learning process as we save these data objects to investigate and understand their characteristics. These data objects could potentially be a result of an imbalanced data set with high cardinality (i.e., natural overlaps) and perhaps the `k`

number of clusters, which is defined a priori, can be revised to accommodate the data and allow the algorithm to fit the clusters better.

Outliers could also indicate suspicious data objects with malicious intent. Therefore, an outlier cluster that can be investigated has profound real-life implications such as in e-banking, website phishing, cybersecurity, or medical screening. For example, in cybersecurity, historical data can reveal acceptable statistical trends through the data patterns and how they are clustered together. Any outlier objects outside the regular clustered trends will automatically raise red flags. Such red flags can be used in data analytics to alert the user of a potential security threat or an intrusion attempt.

### 3.4 Example on OMCOKE

In this section, a comprehensive example for the reader is provided to demonstrate the OMCOKE algorithm steps. A Sample dataset is provided with two attributes (A & B) in table 3.1 and figure 3.3. For example, consider the following sample data with the input vector [40,4; 37,4; 45,4; 29,2; 38,5; 38,6; 38,3] and a user-defined  $k$  as 4.

Table 3.1 OMCOKE Sample dataset

	A, B
STRATEGY	40,4
COMPUTERS	37,4
ECON	45,4
ENGLISH	29,2
MATH	38,5
STATISTICS	38,6
E-BUSINESS	38,3



Figure 3.3 OMCOKE Sample dataset

### 3.4.1 OMCOKE Algorithm Step 1

In this example, data is run with  $k$  clusters initialized to 4. The user provides the value of  $k$ . The algorithm splits the data into partitions based on  $k$  disjoint initial groupings or clusters and using the objective function iteratively improving the quality of the partitions. Each point is assigned to its closest centroid while recalculation of the centroid of each cluster is done while checking the convergence of the objective function. Once the convergence criteria are met and there are no changes in the clusters, all objects in the dataset would be assigned to their closest centroid. The following table 3.2 and figures 3.4 and 3.5 below demonstrate the cluster assignment for each data points after the K-means algorithm run.

Table 3.2 K-means output

VECTOR	C0	C1	C2	C3
40,4	0	1	0	0
37,4	1	0	0	0
45,4	0	1	0	0
29,2	0	0	0	1
38,5	0	0	1	0
38,6	0	0	1	0
38,3	1	0	0	0

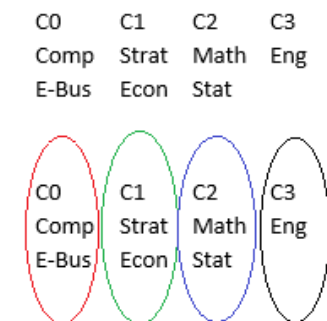


Figure 3.4 Initial groups

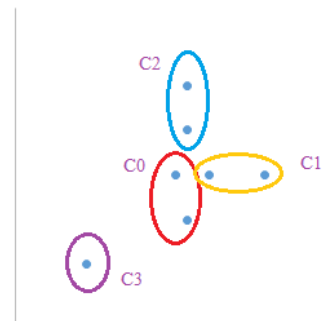


Figure 3.5 Initial clusters

The unlabelled data is now clustered according to their distance to their closest centroid. The K-means algorithm assigns each data object to a cluster. The membership of those objects will have a value of 0 (zero) if it does not belong to the cluster and a value of 1 (one) if it belongs to that cluster.

### 3.4.2 OMCOKE Algorithm Step 2

The MCOKE algorithm calculates the most significant distance an object is assigned to any cluster and records this to the local variable *maxdist* which then iterates and compares it to every object pair in the clusters. If the object belonging to a particular cluster has a shorter distance to



another cluster than the value of *maxdist*, that object is also assigned to the new cluster allowing it to belong to multi-clusters. Table 3.3 and figures 3.6 and 3.7 below demonstrate the cluster assignment after the MCOKE algorithm completes.

Table 3.3 OMCOKE Output

VECTOR	C0	C1	C2	C3
40,4	1	0	1	0
37,4	0	0	1	1
45,4	1	0	0	0
29,2	0	1	0	0
38,5	0	0	1	1
38,6	0	0	0	1
38,3	0	0	1	1

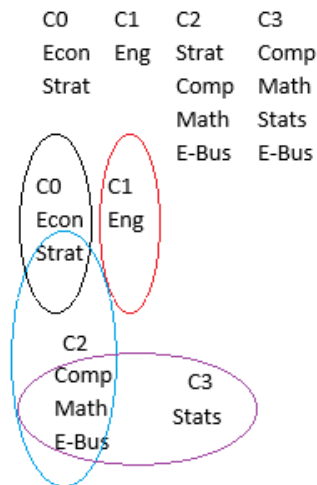


Figure 3.4 Multi-clustered groups

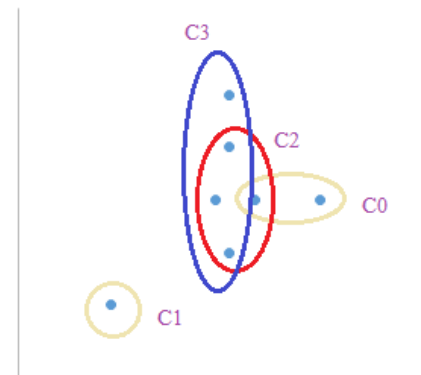


Figure 3.5 Overlapping clusters

Objects that have a membership to a cluster are still assigned a 1 (one) and those that do not are assigned a 0 (zero), i.e., full memberships to the multiple clusters. There is no partial membership based on some weightage criteria for the objects as in fuzzy algorithms. Similar to K-means, the MCOKE algorithm also gets affected by noise or outlier objects. In the example above, it is evident that the object in cluster C1 does not belong with the rest. However, due to the constraints in the algorithm, this object is also assigned a cluster from the  $k$  initialized by the user. This shortcoming is evident in other algorithms such as the overlapping K-means (OKM).

### 3.4.3 OMCOKE Algorithm Step 3

In order to overcome this, the enhancement in OMCOKE looks at the average distance of all other objects as a guideline to determine the inner circle belonging for each object. Before running the algorithm, the user is asked to enter a value for *maxdist* threshold, the radius

distance to be considered from the outer boundary, for example, 0.98 will mean the area covered inside the outer boundary for objects is not to be considered an anomaly.

This is then used as a cut-off to determine an object as an outlier. The object in C1 is thus easily identified as an outlier, and instead of confining the assignment of that object to one of the user-defined  $k$  clusters, a new outlier cluster is initialized on the fly ( $k+1$ ) in the algorithm, and the user  $k$  number of clusters are only used to group the remaining objects. Table 3.4 and figures 3.8 and 3.9 below demonstrates this.

Table 3.4 Outlier Cluster

VECTOR	C0	C1	C2	C3	O
40,4	1	0	1	0	0
37,4	0	0	1	1	0
45,4	1	1	0	0	0
29,2	0	0	0	0	1
38,5	0	0	1	1	0
38,6	0	0	0	1	0
38,3	0	0	1	1	0

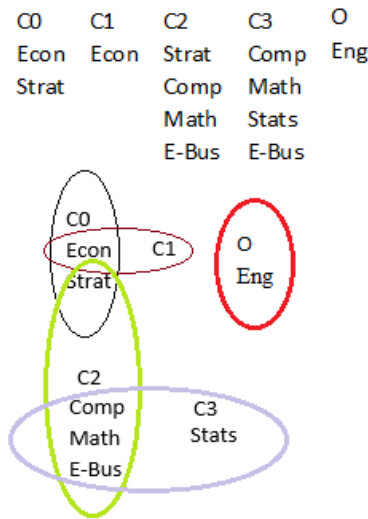


Figure 3.8 OMCOKE K+1 Clusters

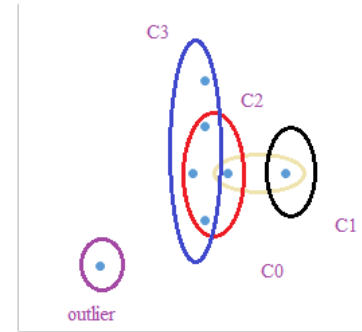


Figure 3.9 Grouping with Outlier

### 3.5 OMCOKE versus other Overlapping Algorithms

In the research literature of clustering, a few overlapping algorithms developed which extend the K-means algorithm. Overlapping partitioning clustering methods tend to relax or remove the constraints allowing overlaps between clusters; this better fits any hidden structures in the data and assign data objects to one or more clusters building a non-disjoint partition of the data (Barai & Dey, 2017).

Extensions of the K-means that allow overlaps include Kernel Overlapping K-means (KOKM) by BenN'Cir & Essoussi, (2012) and BenN'Cir, Essoussi, & Bertrand, (2010)., Overlapping K-means

(OKM) by Cleuziou, G. (2008) & (2009), Parametrized R-OKM by BenN'Cir et al., (2013) and Multi-Cluster Overlapping K-Means Extension (MCOKE) by Baadel, Thabtah & Lu (2016).

A few distinctions can be observed between these algorithms and the one we are proposing. These are listed as the following:

- Unlike the other algorithms, the proposed one does not require the user to determine the threshold to be used for overlapping assignments. This is calculated in the algorithm and will differ depending on the nature of the data. The *maxdist* threshold (inner radius) is the desired cut-off to be used based on the maximum distance value calculated in the algorithm.
- The proposed algorithm does not require users to set a priori the number of labels in the dataset to be used in the calculation of the overlaps of the output data clustering.
- While other overlapping algorithms identify outliers in the dataset, none creates a new outlier cluster on the fly to store them. The proposed algorithm increments user entered *k* number of clusters by 1 and assigned the outliers to this newly created outlier cluster. This is vital information for investigation in the fields of cybersecurity and online banking as such outliers could indicate fraud activities.

### **3.6 Chapter Summary**

In this chapter, a novel algorithm OMCOKE has been proposed. The algorithm differs from other overlapping algorithms in that it does not require a similarity threshold to be defined a priori which may be difficult to set depending on the data samples. It instead uses the maximum distance (*maxdist*) allowed in K-means based on the SSE on Euclidean distance to assign objects to a given cluster as the global threshold. However, the objective function of K-means is designed to optimize while under the constraint of assigning the data objects to hard-partition. This means that all objects will be assigned to at least one cluster including noise or outliers which may then affect *maxdist* as a good predictor to overlap other objects in multiple clusters. The algorithm

was then enhanced to identify and exclude the outlier objects from being assigned to the user entered  $k$ -clusters. This is achieved by incrementing  $k+1$  clusters once an outlier is detected in the dataset. The outlier(s) are then assigned to this newly created cluster while non-noise data are assigned to the  $k$ -clusters in an overlapping manner giving each object full membership of belonging to multiple clusters.

Our algorithm detects and stores outliers during the clustering process making it different from the other overlapping clustering algorithms, thus adding value in this domain. As opposed to discarding anomalies and outliers, our method can provide tremendous benefit to cybersecurity experts, medical practitioners, IT administrators, data mining researchers, and other stakeholders as these outliers could have real-life applications such as fraudulent activities as in the case of cybersecurity, fraud insurance claims in the banking domain, or to help raise flags in the medical field especially in the screening process.

In the next chapter, we show the implementation and evaluation of OMCOKE on different multi-label data sets in real-life application of the algorithms.

## Chapter 4

# Implementation and Evaluation of OMCOKE

### ***4.1 Introduction***

In this chapter, we discuss the implementation of the new proposed Multi-Cluster MCOKE and OMCOKE algorithm and explain the different thresholds it utilizes for the dynamic new cluster creation process. Since the proposed algorithm has been implemented in the WEKA environment (see Section 4.2), its primary Graphical User Interface (GUI) is also briefly highlighted in Section 4.2. We also highlight the integration of OMCOKE within the "Clusterer" package of WEKA and a brief discussion of the input parameters used is provided. Furthermore, a few GUIs of OMCOKE within WEKA are shown, in particular, the different processes to go through clustering. Section 4.3 explains the evaluation measures (pair-based precision-recall measure, confusion matrix, number, etc.) used to produce the results and test the performance of the proposed OMCOKE algorithm and other known predictive models in classification for their comparison.

This Chapter also includes a number of experiments on datasets published at the Mulan: A Java Library for Multi-Label Learning repository (see Section 4.4). The chosen datasets have a different number of variables, variable types, and data examples. Section 4.4 highlights the main characteristics of the datasets used in the experiments. The experimental evaluation validates OMCOKE advantages, particularly in the detection rate of clusters and assigning observations with a better classification precision.

### ***4.2 OMCOKE Implementation***

WEKA stands for Waikato Environment for Knowledge Analysis and is an open source tool based on the Java platform that contains implementations for different DM methods including filtering, classification, clustering, evaluation, and visualisation among others. This tool was designed and implemented at New Zealand's Waikato University to assist students, researchers, and academic staff in conducting quantitative research. Experiments on the datasets using the considered

classification algorithms were conducted using WEKA. The proposed OMCOKE algorithm was implemented in Java and integrated into the WEKA environment.

Since WEKA is built with Java programming language, it organises the different filtering, learning, and visualisation methods in packages. A package can be seen as a container that holds and manages related Java classes and has its subdirectories. In WEKA versions 3.7.2 and upward, information other than classes can be held inside a package such as the functionalities of the JAR file, metadata, source codes, and other related documentation of the classes. These enable users to have better management of which methods can be stored in separate packages, and therefore can use what they need. By default, WEKA keeps packages and their related information in `$WEKA_HOME`, which is located in the user's home directory (i.e., `user.home/WEKAfiles`).

The main class of the proposed algorithm, OMCOKE has been integrated into the WEKA environment inside the "clusterers" package. The "clusterers" package contains the implementation of a few clustering algorithms such as the Simple K-means, EM (Expectation-Maximization) algorithm, Hierarchical Clustering, Cobweb, Density-Based Clustering among others. The proposed algorithm can be accessed from WEKA Explorer or the Command Line platforms for data processing.

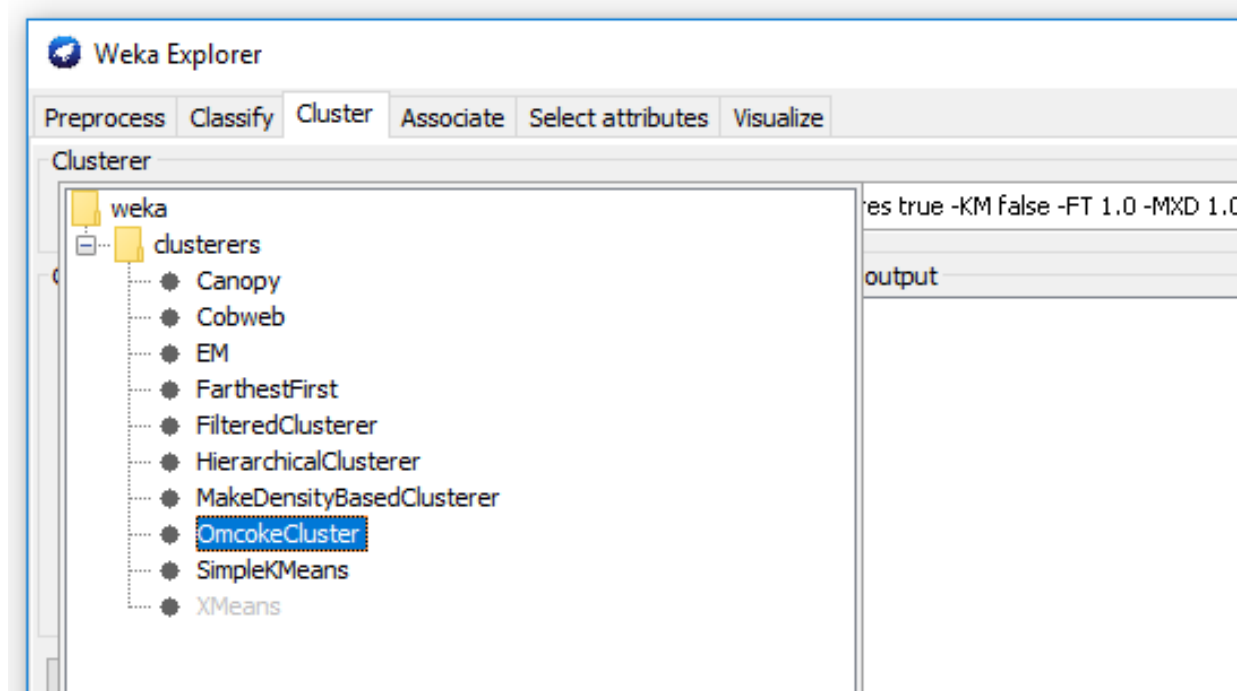


Figure 4.1 OMCOKE algorithm integrated inside the clusterers in WEKA

There are many advantages of integrating OMCOKE to WEKA and the usage of WEKA in general.

The following are some of the inherent pros of WEKA:

- a) An open-source application tool that can be downloaded freely online
- b) Its graphical user interface (GUI) is relatively easy to be used by novice users who are not very adept to programming or software development
- c) Results are displayed on the output pan with clear evaluation metrics used
- d) Graphical results make for clear visualization of the output

However, there are also a few cons of using the WEKA tool. These include:

- a) Coding and utilizing inbuilt WEKA classes is quite challenging
- b) Step-by-step manual on how to use the tool missing on the GUI. This can be very intimidating for first time users
- c) Any updates or bug fixes must go through the WEKA community and may be time consuming

OMCOKE utilizes the *maxdist* variable to calculate the average distance of objects to the cluster centroids. This setting on the WEKA is set to 1 (by default) for the algorithm to use the distance obtained as the threshold to for determining the objects belonging in the overlapping clustering (see figure 4.2 below). We use this setting in our experiments as it sustains the predictive method of our algorithm to assign all objects pairs that may belong to multiple clusters to be assigned to only the selected number of clusters in the initialization phase. This will be in line with other methods used to measure the performance of OMCOKE.

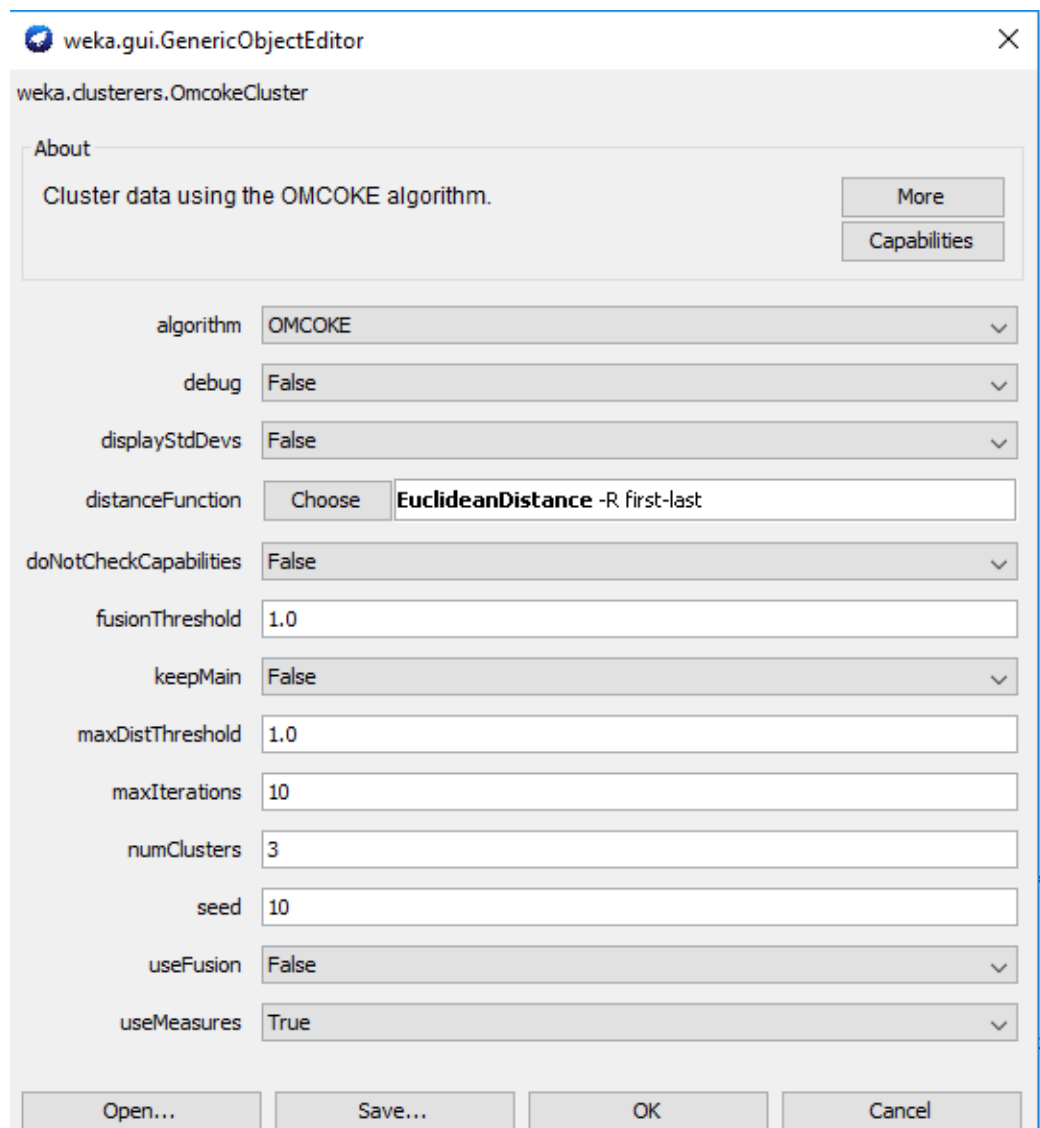


Figure 4.2 OMCOKE algorithm GUI parameter setting and options



Users may change this setting to less than 1 such as 0.95 which would mean limiting the similarity threshold to the fifth percentile of the maxdist average calculated by the algorithm. By doing so, the algorithm would then filter any data objects that are above this threshold which would then be deemed as outliers. However, these outlier observations are not pruned out by the algorithm but instead are assigned to a new cluster on the fly that will list all of these object indexes.

The option useMeasures is also set to "True" by default in our algorithm as that will set our algorithm to calculate the effectiveness of the cluster assignments using the pair-based Precision-Recall measure.

The option useFusion is set to "False" and fusionThreshold set to 1.0 by default as these features have not been implemented yet in our algorithm.

All data formats that are supported by WEKA can be used to be received files in OMCOKE. Data that can be loaded under in WEKA include the following formats:

- a) Files that such as WEKA's ARFF file format, C4.5 Files, text files, comma separated formats in a spreadsheet (CSV), and JSON files.
- b) Using the Uniform Resource Locator (URL) functionality to load data that is saved in a specific server.
- c) Databases connected externally using ODBC to any commercial or open source databases such as Oracle, SQL Server, or MySQL.

For further graphical exposition, we provide a detailed example of data processing and other OMCOKE GUI implementation in WEKA in **Appendix A**.

### **4.3 Evaluation Measures**

A number of measures related to predictive models in clustering that we used are discussed in this Section. In particular, we describe the pair-based approach, common predictive model evaluation criteria such as accuracy, f-measure, precision, and confusion matrix related measures such as false positive, false negative, true positive, and true negative. Moreover,

methods related to computing resources utilised through the clustering processing such as processing time and numbers of data examples are also covered.

We describe some common predictive model evaluation criteria such as accuracy, sensitivity, specificity, one-error, harmonic mean a.k.a. F1, and other related measures such as false positive (FP), false negative (FN), true positive (TP), and true negative (TN).

### **4.3.1 Confusion Matrix in Clustering**

Evaluation of clustering is a very critical process to assess the performance of the algorithm. For DM and ML predictive models, a matrix called the error table, or the confusion matrix, has been developed. The confusion matrix is typically used to evaluate the performance of predictive models with respect to the different metrics that are primarily related to the predictive power of the models. In measuring the performance of the models in clustering, a data node is assigned a predicted centroid by the model based on the similarity attributes to that cluster. If the data node is similar to the predicted centroid, this counts for a correct cluster assignment; otherwise, it is considered a false assignment or a misclassification.

- i) A true positive (TP) decision will assign two similar observations to the same cluster (a correct decision)
- ii) A true negative (TN) decision will assign two dissimilar observations to different clusters (a correct decision).

By doing this, there is a potential of two types of errors that we are bound to commit.

- i) A false positive (FP) decision that will have assigned two dissimilar observations to the same cluster or
- ii) A false negative (FN) decision that will have assigned two similar observations to different clusters.

This is summarized in the confusion matrix in table 4.1 below.

Table 4.1 Confusion Matrix

	Same Cluster	Different Cluster
Same class	TP	FN
Different class	FP	TN

We used the pair-based Precision-Recall measure that is calculated over pairs of observations.

The precision-recall is computed as follows:

$$Sensitivity(\%) = \frac{|TP|}{|TP + FN|} \quad (4.1)$$

$$Specificity(\%) = \frac{|TN|}{|FP + TN|} \quad (4.2)$$

$$Accuracy(\%) = \frac{|TP + TN|}{|TP + TN + FP + FN|} \quad (4.3)$$

$$One\_error(\%) = 1 - Accuracy \quad (4.4)$$

$$F1 = \frac{2 * |Precision * Recall|}{|Precision + Recall|} \quad (4.5)$$

#### 4.3.2 Computing Resources Evaluation Measures

Measuring the processing time taken for the predictive model to be constructed is a useful indicator of the efficiency of the data processing phase. Moreover, the time taken to predict data cases can be measured as part of the model's efficiency. In the WEKA environment, we recorded the time taken to build the predictive model in milliseconds (ms), so we can compare OMCOKE

processing time performance with other clustering algorithms. Moreover, we have also computed how many instances are necessary to build the OMCOKE algorithms, and implemented it as a Java class in the WEKA version. This class contains codes that compute repetitive data scans and can be seen as a performance indicator of the proposed algorithm when compared with other algorithms.

#### 4.4 Data Experimental Settings

In this section, different datasets from Mulan: A Java Library for Multi-Label Learning repository is used to evaluate the proposed OMCOKE algorithm performance. The data repository hosts more than 25 different datasets in the domains of text, audio, video, music, images, and biology to mention only a few. Items of Multi-label datasets can be members of multi-groups which are true for real-world problems and as a result ideal for the study of overlapping clustering. In our empirical experiment, three different domain datasets that have been used along with their specifications are displayed in Table 4.2.

Table 4.2 Statistics of used Benchmarks

<b>Data set</b>	<b>Domain</b>	<b>Instances</b>	<b>Distinct</b>	<b>Labels</b>	<b>Attributes</b>	<b>Cardinality</b>	<b>Density</b>
<b>Emotions</b>	Music	593	592	6	72	1.869	0.311
<b>Yeast</b>	Biology	2417	2412	14	103	4.237	0.303
<b>Scene</b>	Images	2407	2349	6	294	1.074	0.179

Table 4.3 Descriptive Statistics of used Benchmarks

<b>Data set</b>	<b>Min</b>	<b>Max</b>	<b>Mean</b>	<b>StdDev</b>
<b>Emotions</b>	0.01	0.195	0.069	0.031
<b>Yeast</b>	-0.371	0.52	0.001	0.097
<b>Scene</b>	0.0	1.0	0.659	0.214

In order to exhibit OMCOKE performance with respect to different measures when contrasted with a wide range of ML, we selected clustering algorithms using the following criteria:

- a) Utilize a partitioning method that extends the K-means algorithm
- b) The methods use the Euclidian distance to calculate the similarities between the sets of observations;
- c) Work on numeric attributes only
- d) All are known algorithms that have been evaluated by previous researchers in the DM and ML communities;

#### **4.4.1 Evaluation Methodology**

We conduct our experiments on real-life overlapping datasets in order to measure the effectiveness of the methods used to identify such overlapping groups. The three datasets have a wide diversity in their dataset hence make it a suitable combination for use as benchmarks. For example, their sizes vary from 593 (Emotions) to 2417 (Yeast), their dimension (attributes) from 72 (Emotion) to 294 (Scene), cardinality (i.e., overlap rates) from 1.074 (Scene) to 4.237 (Yeast). Their application domain also varies considerably as well, i.e. music, biology, and images.

All experiments have been run on an Intel Core i7 computer with a 3.4 GHz processor and 8.0 GB RAM running on a 64-bit Windows 10 Operating System. We utilised a number of evaluation measures to show the benefits and negatives of the proposed algorithm when compared with other classification algorithms in DM. Precisely, the below measures have been used to evaluate OMCOKE:

- Precision
- F-measure

Evaluation measures use binary functions to compute the relationships between pairs of objects in a cluster assuming that those objects belong to one cluster. In overlapping clustering, these objects could also feature in multiple clusters. Therefore, we chose the above two measures

precisely because of this reason since other evaluation measures such as the Recall will result in a biased number due to the overlaps in the dataset.

## 4.4.2 Description of Overlapping Datasets

### 4.4.2.1 *Emotion dataset (Troihidis, et al., 2008)*

Analysing music signals are used in the detection of emotion in music. In this case, music can be classified into several categories at the same time since they are not usually disjoint, i.e. they can make one feel both "sad" and "angry." The dataset contains sound clips that can be described by 72 attributes which were annotated by three male music experts into 6 emotional clusters. Only the songs that had all three experts unanimously agree on its label were kept resulting in 593 total songs selected for the dataset. The clusters are shown in Table 4.4 below:

Table 4.4 Emotion dataset

Description	# of songs
amazed-surprised	173
happy-pleased	166
relaxing-calm	264
quiet-still	148
sad-lonely	168
angry-fearful	189

### 4.4.2.2 *Yeast dataset (Elisseeff and Weston, 2001)*

The Yeast dataset is classified into 14 gene groups or classes. A gene can belong to several different classes at the same time thus making this a multilabel dataset. For example, a gene YAL014W may belong into the following four groups: { Cell Growth, Cell Division}, {Cellular Organization}, {Cellular Communication, Signal Transduction} and { Transposable elements, Viral and Plasmid Proteins}.

### 4.4.2.3 *Scene dataset (Boutell, et al., 2004)*

The dataset contains 2407 natural scene images. The images were classed into 6 categories. In this case, the images can be classified into different categories at the same time since they are

not usually disjoint, i.e. they become multi labelled and can belong to more than one category such as field + mountain and fall foliage + mountain.

Table 4.5 Scene Dataset

Description	# of Images
Beach	369
Sunset	364
Fall foliage	360
Field	327
Mountain	405
Urban	405
Beach+Field	1
Fall foliage+Field	23
Beach+Mountain	38
Fall foliage+Mountain	13
Field+Mountain	75
Field+Fall foliage+Mountain	1
Beach+Urban	19
Field+Urban	6
Mountain+Urban	1
Total	2407

#### **4.5 Empirical Results on the Multi-Label Datasets**

Our experiments are conducted on real datasets from three different domain namely Emotion, Yeast and Scene that have a strong overlap. For fair comparisons, datasets with different sizes and from different domains have been chosen and are compared to well-known algorithms that have been evaluated by previous researchers in the DM and ML clustering communities. Through experimental study, we evaluate and compare the performance of OMCOKE with 4 existing methods namely: Kernel Overlapping K-means (KOKM), Overlapping K-means (OKM), and Parametrized R-OKM as shown in table 4.6 below.

For each experiment, we set the parameters for KOKM, OKM, and P-ROKM as follows:

- Maximum iterations = 10
- Number of clusters = 3
- Number of labels = Emotions (6), Yeast (14), and Scene (6).
- Minimal improvement = 0.01
- Alpha = 1 and 0.1 for P-ROKM algorithms.

In addition to the number of iterations and clusters set as above, the following parameters were also set in OMCOKE:

- maxdistThreshold = 0.99
- useMeasures = True

Table 4.6 Overlapping Algorithms Performance Comparisons

Method	Emotion		Yeast		Scene	
	P.	F.	P.	F.	P.	F.
<b>KOKMII</b>	0.471	<b>0.641</b>	0.785	<b>0.878</b>	0.193	0.324
<b>OKM</b>	0.467	0.586	0.234	0.376	0.234	0.376
<b>P-ROKM (α=1)</b>	0.474	0.524	0.919	0.565	0.379	<b>0.506</b>
<b>P-ROKM (α=0.1)</b>	0.468	0.578	0.802	0.654	0.288	0.439
<b>OMCOKE</b>	<b>0.565</b>	0.419	<b>0.972</b>	0.496	<b>0.706</b>	0.453

Non-overlapping methods such as the K-means have an overlap equal to 1 simply due to the fact that these algorithms build non-disjoint clusters without considering that an object may belong to more than one cluster. Overlapping methods will have an overlap that is greater than 1 since the objects belong to more than one cluster. The size of the overlaps affect the value of Precision, i.e. there will be a low value of Precision because the observations are assigned to more than one cluster.



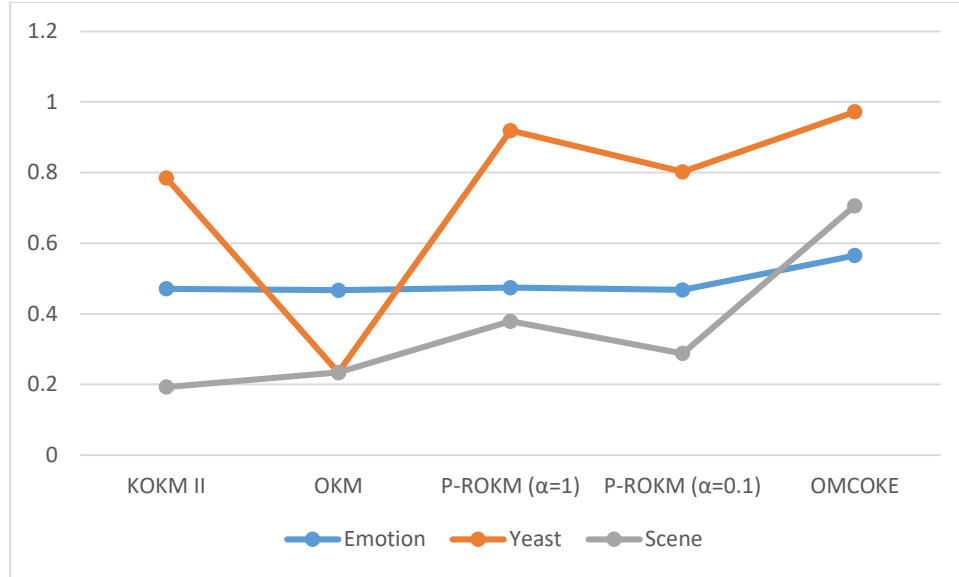


Figure 4.3 Precision Accuracy of the benchmark datasets

The pair-based Precision-Recall method used in the empirical results is calculated over pairs of observations. This allows for the evaluations of clusters independently and compares their partitions with different numbers of clusters in the dataset. It measures whether the predicted pair is correctly assigned in the same cluster as indicated in the true class datasets. However, the Recall measure uses a binary function to compute the relationship between pairs of observations, and not considering that those pairs of observations could also feature in multiple clusters in the overlap. This results in a biased Recall measure, especially when the cardinality in the dataset is large. Thus we chose not to use the Recall in our experiment as a measure of OMCOKE.

It is evident from the above empirical results that the OMCOKE algorithm has a high precision rate and outperforms all the other overlapping algorithms in the study as shown in Figure 4.3 above. This can be attributed to the algorithm's ability to separate outliers from the rest of the data objects when assigning them to clusters. For the Emotion, Yeast, and Scene datasets, OMCOKE precision was 0.565, 0.972, and 0.706 followed by P-ROKM ( $\alpha=1$ ) at 0.474, 0.919, and 0.379 respectively.

The high values of Recalls generally induce high values of F-Measures as opposed to non-overlapping algorithms whose high values of F-measures are generally as a result of the Precision. When compared to the other algorithms, OMCOKE performs relatively well in the F-Measure as shown in Figure 4.4 below, scoring second behind P-ROKM (with  $\alpha=1$ ) in the Scene dataset; the P-ROKM method with the alpha value of 1 yielded an overlap of exactly 1 and dataset had a cardinality of 1.07.

The F-Measure values are higher for clustering methods whose overlap rates are closer to the actual cardinality of the dataset. The cardinality shown in Table 3 is the natural overlaps in the dataset, i.e. the average number of categories each observation can belong to.

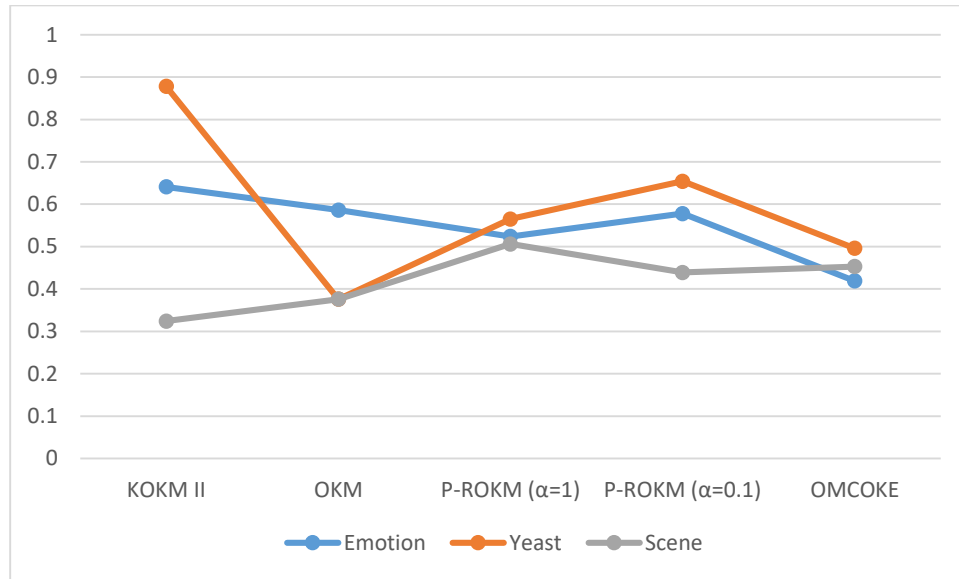


Figure 4.4 F-Measure of the Benchmark Datasets

The analysis shows that F-Measures and Precision are significantly affected by the overlap rate in the actual dataset. Algorithms that have partitions with smaller overlaps fared well in their F-Measure meaning that they produced non-disjointed partitions that fit the data better compared to others. OMCOKE performed reasonably well in the Scene, and Emotion datasets since the cardinalities of the datasets are low (1.074 and 1.869 respectively) nearing 1 but did poorly in the Yeast dataset that had an overlap of over 4.

Our algorithm detected several outliers in the dataset. These are listed in Table 4.7 below.

Table 4.7 Outliers Detected in the Three Datasets

Dataset	Number of Outliers	Position of Outliers in the Dataset
Emotion	1	27
Scene	2	304 1502
Yeast	1	1819

As indicated, an input dataset containing a few outliers significantly influences the mean distance (the outlier will skew the mean and variance) of the data objects to their respective clusters. This explains why OMCOKE outperformed the other methods in all datasets in terms of Precision rate. This also shows that by separating the outliers from the rest of the data, the OMCOKE was able to build its model relatively closer and more acceptable to the actual overlaps in each of the datasets; this is as compared to the other methods for the precision to be higher than the rest.

#### **4.6 Chapter Summary**

In this chapter, we provided the details of OMCOKE implementation and how it is integrated into WEKA environment outlining the required parameters that need to be set. We described the different multi-label data types used in the experiment as our benchmark. In addition, a detail discussion is provided on the evaluation measures used, the experimental settings, and the comparison that was done between OMCOKE and other well-known cluster methods in DM.

We conducted a number of experiments in the WEKA environment that test the overlapping algorithms given real-life overlapping datasets. The results of the experiments are summarized below:

Our algorithm, with the capability of detecting outliers and treating them as a separate cluster, was evaluated and compared with three existing overlapping clustering methods namely: Kernel

Overlapping K-means (KOKM), Overlapping K-means (OKM), and Parametrized R-OKM. We used real-life multi-label datasets for our experiments. The empirical results showed that the F-Measures and Precision were significantly affected by the overlap rate in the actual dataset. OMCOKE did well in the Scene dataset since the cardinality of the dataset is very low, and did poorly in the Yeast dataset that had a significant high overlap rate of over 4. However, when it came to Precision, OMCOKE outperformed the other overlapping algorithms in all datasets indicating that our method had a better detection rate of clusters and for assigning observations with a better precision after it segregated the outliers in the dataset.

a) On the Precision measure, OMCOKE outperforms all other algorithms with a large margin.

It is evident in this empirical result that our method has a better detection rate of clusters with a better classification precision.

b) OMCOKE performed reasonably with the F-Measure especially when the cardinality of the dataset is relatively small.

In the next chapter, we apply clustering with classification (semi-supervised technique) to discover Autism Spectrum Disorder (ASD) symptoms based on historical cases to enhance autism screening efficiency and accuracy. We apply OMCOKE as a clustering technique to identify potential autism cases based on their similarity traits as opposed to a scoring function used by many ASD screening tools. We test this on real datasets related to screening of autism involving children, adolescents, and adults and compared the performance to other common machine learning classification techniques.

## **Chapter 5**

# **Case Study on Autism Spectrum Disorder (ASD) Screening**

### ***5.1 Introduction***

Autism Spectrum Disorder (ASD) is a neurodevelopmental condition that contributes to the delay of social and communication behaviours of individuals (Bolton et al., 1994; Belmonte et al., 2004). Typically, ASD diagnosis is made by clinicians in a clinical set up using visible behavioural indicators in a process referred to as clinical judgment (CJ) (Wiggins et al., 2014; Thabtah 2017a). The official diagnosis process of ASD involves multiple examinations, which in turn cause the waiting time for patients to be lengthy (Thabtah, 2018a). For instance, the waiting time for an ASD diagnosis in the UK averages over 3 years (Crane et al., 2016). Therefore, it is vital that the administration time needed for both screening and diagnosis be reduced to cater for the growing number of ASD patients (Lord and Jones, 2012; Levy et al., 2017; Haber & Wall, 2017).

Autism screening is a fundamental step that addresses whether individuals exhibit potential autistic traits related to communication, social or repeated behaviour (Abbas et al. 2018). This step is crucial as the individual and the concerned family become aware of the possibility of ASD traits early and hence can search for the needed formal assessments. There are many ASD screening tools developed by researchers such as Autism Spectrum Quotient (AQ) and Childhood Autism Rating Scale (CARS) (Baron-Cohen, 2001; Baron-Cohen et al., 2006; Krug et al., 2008; Shopler et al., 2010). Most of these screening methods have been developed using existing clinical autism diagnosis methods and are represented as questionnaires in which each question is associated with a few possible answers in a multiple-choice fashion. The questionnaires used contain measurable indicators (variables/questions) that address communication, behaviour and social skills, of individuals. For example, the Child Behavior Checklist (CBCL) screening method contains more than 100 questions (Achenbach and Rescorla, 2001), and the AQ method contains

50 questions (Baron-Cohen et al., 2006). These make the process of screening lengthy besides inaccessible as most existing screening methods usually do not exist indirectly accessible platforms such as mobile (Thabtah, Kamalov & Rajab, 2018; Thabtah 2018b).

Most of the existing autism screening methods utilize scoring functions that compute a final score based on the answers given by users undergoing the screening (caregivers, parents, medical staff, teachers or even the adult patients). To be specific, the screening methods take the answers given in the questionnaire as an input for the scoring function, which in turn processes the input and computes a final score to reflect whether the individual is associated with ASD traits. For instance, in AQ method, a cut-off score of larger than 32 is an indication of autistic traits (Baron-Cohen et al., 2006; Auyeung et al., 2008). Therefore, the final decision of having ASD traits lay solely on the score calculated by the function. This function in most cases sums up the behavioural indicators' answers and does not attempt to seek for correlations among these indicators and the target class (ASD traits).

To address these shortcomings, there is a need for intelligent methods that can replace the scoring function and improve the efficiency of the screening. Since ASD screening involves forecasting whether individuals have the possibility of ASD traits based on a predefined characterized variable, then this issue is a predictive analysis problem in ML. The screening of ASD traits can be considered a classification problem in which historical data that have been already classified with and without ASD traits is utilized as an input to construct a classification system. This system is then used to guess whether a new individual exhibits any autistic traits. ML can be utilized for ASD screening to improve the classification of the screening and to reduce the process of the screening time. More importantly, ML may provide models that can contain useful information about ASD traits to the diagnosticians especially the correlation among behavioural indicators and how they relate to ASD screening. ML techniques use artificial intelligence and statistics to create intelligent models by discovering hidden patterns in data so that users can improve decisions (Thabtah and Peebles, 2019; Thabtah et al., 2018).

There have been recent attempts to adopt ML techniques in autism screening and diagnosis, i.e. (Abbas, et al., 2018; Thabtah, et al., 2018b; Levy, et al., 2017; Stewart & Lee, 2017; Bekerom, 2017; Thabtah, 2017a; Bone, et al., 2016; Chen, et al., 2016; Ventola, et al, 2016). These studies focused primarily on improving time, accuracy, and reducing the dimensionality of the dataset by pinpointing influential autistic symptoms. Thabtah et al., (2018) proposed a new feature selection method called Variable Analysis (Va) to determine the most influential features related to ASD based on datasets related to adults, adolescents, and children. The authors were able to minimize the number of features to 5-7 based on predictive analysis and filter methods. Abbas et al., 2018 used Random Forest to improve the diagnosis process of autism and Levy et al., (2017) compared 17 different classification-based ML algorithms to seek improvements on the diagnosis performance of autism for children.

In this chapter, we propose a new semi-supervised learning method called Clustering based Autistic Trait Classification (CATC), to improve the accuracy of the autism screening problem. The utilization of clustering and classification together as semi-supervised learning is rare in autism screening research. Unlike existing methods that primarily focused on the classification phase of cases and controls, we intend to utilize clustering with classification to validate instances in the training dataset prior to constructing the classification systems. CATC integrates unsupervised learning in the pre-processing phase with supervised learning in the classifiers construction phase. By integrating clustering with classification, there is a potential for improving the resulting classification systems by detecting ASD traits more accurately. With the CATC technique, the predictive model performance is enhanced in twofold;

- 1) Pre-processing the dataset and clustering them in the training phase of the classification algorithm. By clustering the data first, we will identify relevant traits/features that can be used in the ASD learning phase.

2) Reducing data dimensionality by eliminating features redundancy. Clustering can wrap those traits that may appear in multiple clusters and identify them as stronger or more significant features for the classification algorithms.

The proposed intelligent method considers the hard cases to be classified (Cases that exhibit few autistic symptoms). These cases may exhibit some autistic traits but may not be qualified to be on the spectrum. These cases often cause large false positives and false negatives, which deteriorate the performance of the classification algorithm. Thus, we show that having clustering at the pre-processing phase will enhance the predictability of the classification algorithm and improve the classifier accuracy, sensitivity, specificity, and error rates.

The rest of the chapter is structured as follows: section 5.2 reviews the background information around machine learning in ASD research. Section 5.3 discusses the methodology used for CATC and including a description of the datasets used and the pre-processing of the data. Section 5.4 outlines the experimental preparations and settings. Section 5.5 provides the results and analysis and a comprehensive comparison of different ML techniques including CATC. Lastly, we provide a conclusion in section 5.6.

## ***5.2 Background Information on ASD Detection***

Many of the ASD screening techniques rely solely on a scoring function (which sums up the answers based on some of the behavioural indicators) to determine ASD traits. ML algorithms tend to focus on improving time and accuracy by reducing the number of items in the self-assessment phase that can be used to predict the ASD symptoms while still relying on the scoring function. However, these ML techniques do not focus on eliminating redundancy in the dataset or clustering the data based on strong features.

Some of the researchers that use ML classification algorithms to predict ASD screening and diagnosis include (Wall et al., 2012; Mythili & Shanavas, 2014; Pratap et al., 2014; Bone et al., 2014; Pancer and Derkacz, 2015; Wolfers, et al., 2015; Duda et al., 2016; Bone et al., 2016; Chen et al., 2016; Towle & Patrick, 2016; Bekerom, 2017; Thabtah, 2018; Thabtah et al. 2018).



Thabtah et al., (2018) improved the efficiency of the screening process by reducing the number of items in the self-assessment screening tool called AQ-10 (Allison et al., 2012). The authors proposed a new feature selection ranking method called variable analysis (VA) that would derive small yet effective autistic traits. The authors used different datasets of adult, adolescent, and child in their study and compared their algorithm performance measures with other classification tools RIPPER and C4.5 (Cohen 1995; Quinlan 1994). The results analysis showed that VA selected prominent features for the three datasets without compromising on the specificity, sensitivity, and prediction accuracies measurements.

Abbas et al. (2017) conducted a clinical study of 162 at-risk children that had received a clinical diagnosis. They collected their dataset by splitting their screening process into two parts. The first part is answered by the parent about the child based on the Autism Diagnostic Interview-Revised [ADI-R] (Lord et al., 1994) that have 93 multi-part questions. The second part is a video screener used by parents based on the Autism Diagnostic Observation Schedule [ADOS] (Lord et al., 2000). The authors applied their datasets to Random Forests classifiers. They later combined the questionnaire and video screeners using regularized logistic regression. They then compared their results with some of the non-machine learning screening tools such as the modified checklist for autism in toddlers (MCHAT) and CCBL. Their results suggest that combining the video and questionnaire into a single assessment boosted the sensitivity and specificity rates and overall performance of the study sample.

A study by Levi et al. (2017) utilized 2 ADOS modules; one for children with phrased speech (Module 2) and the other for children with verbal fluency (Module 3) to build sparse models that were used to train about 17 classifiers from 5 different classifier families (linear regressions, nearest neighbour models, general linear models, support vector machines, and tree-based classifiers) for autism screening and diagnosis. The module 2 dataset consisted of 1389 cases where 1319 were considered as ASD and only 70 as No-ASD. Module 3 dataset had 3143 cases with 2870 considered as ASD and 273 No-ASD. The study was applied. The authors aimed at showing reduced subsets of features with their best parameters that can be used in the classifiers

to predict ASD and No-ASD cases. They concluded that SVM and logistic regression performed best with ROC of 93% and 92% respectively and logistic regression and Lasso performed best on module 3 with a ROC of 93%.

In their study of how some frequency-specific brain indices can be used in the early detection of ASD, Chen et al. (2016) used a limited data set from the Autism Brain Imaging Data Exchange database (ABIDE) of 240 with 112 with ASD and 128 with No-ASD. They experimented by looking at the functional brain connectivity as the frequency bands which were considered as the feature attributes of the dataset. The researchers used the support vector machine algorithm and could predict the ASD diagnosis with a classification accuracy of 79%.

Duda et al. (2016) made an experimental comparison of six classification algorithms on a real dataset consisting of 2900 cases with 65 features. The authors first pre-processed the data by removing any instances with more than four missing values. They applied logistic regression models, Random forests, support vector machine, C4.5 among other classification algorithms. They concluded that function based algorithms such as regression models performed better with high classification accuracy compared to the decision tree based algorithms such as Random Forest.

Others such as Pratap et al. (2014) and Pratap & Kanimozhiselvi (2014) use multiple supervised and unsupervised machine learning algorithms such as Naïve Bayes, self-organization feature map (SOM), learning vector quantization (LVQ), artificial neural network (ANN), K-means and fuzzy c-means to test how machine learning methods can be used in the assessment of autism diagnosis. These two studies used a limited dataset of only 100 cases of children and were able to show that using unsupervised learning such as clustering improved the accuracy of the ASD based on the childhood autism rating scale (CARS) diagnostic tool. This study is limited in size does not measure the improvement of crucial other classification metrics and have yet to be verified in other works.

A more recent review by Thabtah, (2018) analysed some of the cons associated with ASD classification studies conducted earlier. The authors instigated that earlier studies had pitfalls in their datasets that were limited in size and had several missing values and imbalances. The author also pointed out that while the studies showed promising results, none were embedded in a screening tool.

Allison et al. (2012) study was aimed at reducing the AQ and Q-CHAT method screening tests by determining the highest ranked items based on DI measure scores. The authors were able to prove that only ten items can be used for screening the first level of ASD traits. These ten items were adopted in a later study by Thabtah, et al., (2018) to build Adult, Adolescent, and Child datasets based on the AQ screening tool. These new datasets are used in the experiments in our paper.

Our study considers key classification evaluation measures. We evaluate and compare the results to highlight the significance of integrating clustering algorithms and specifically Multi-Cluster Overlapping K-Means Extension (OMCOKE) (Baadel et al., 2016) in the pre-processing phase of the screening data. Clustering of the dataset adds the following value to the classification process;

- a) Reduces data dimensionality by eliminating redundancy.
- b) Identifies relevant and strong features that were only used in the supervised learning models. These features may have been cases that exhibited some autistic traits but not qualified to be on the spectrum hence causing large false positives and false negatives.

The following section discusses the proposed clustering based autistic trait classification technique.

### ***5.3 Clustering based Autistic Trait Classification (CATC)***

In this section, we discuss the proposed CATC method based on the architecture shown in Figure 5.1 below. Three data sets (adult, adolescent, and child) are collected via a mobile screening

app called ASDTest (Thabtah 2017b; Thabtah 2018b). The data is then cleaned for our experimentations and is ran through an unsupervised machine learning clustering algorithm. The result of this process is used as our initial model that is loaded to a classifier for the predictive phase. The performance of the classifier is then tested and evaluated for better accuracy, sensitivity, and specificity rates. Further details for each of the steps are outlined in the subsections that follow.

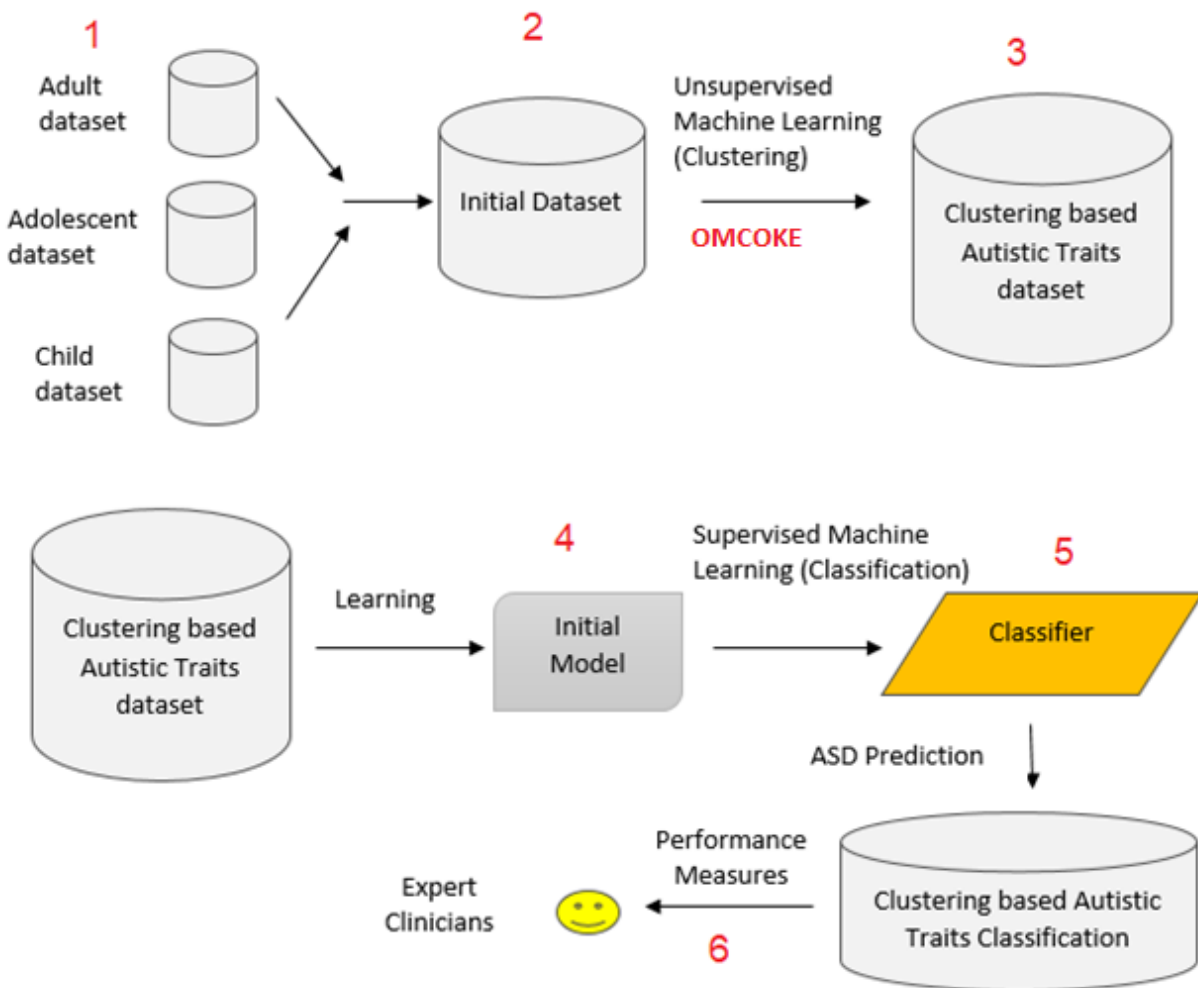


Figure 5.1 CATC-based methodology

### 5.3.1 Data Collection

Initially, data is collected using a mobile screening tool called ASDTests (Thabtah, 2017b; Thabtah 2018b). This tool contains questionnaires based on the Q-CHAT 10, AQ-10 Child, AQ-

10 Adolescent, and AQ-10 Adult screening methods (Allison et al., 2012). The child, adolescent and adult datasets that have been collected contain instances for individuals between 4-11 years old, 12-16 years old and above 16 years respectively. These datasets have been disseminated recently at the University of California Irvine data repository (Lichman, 2013) by (Thabtah et al., 2018).

During the screening process using the ASDTests mobile application, a user answers the screening questions, and a value is calculated based on the answers they enter with a score between 0 and 10. The attribute Class (attribute number 23 in table 2 below) is assigned a YES or a NO based on the score of the answers entered. A score of 6 and above based on (Allison et al., 2012) indicates that the individual has some ASD traits and the class label is labelled as YES. Otherwise, the class is given a value of NO.

The size of the datasets varies between the three groups. The adult dataset has the highest number of instances followed by the child and adolescent. Table 5.1 and Figure 5.2 below summarizes the dataset based on the number of instances and the history of the users with regards to having a family member previously diagnosed with ASD.

Table 5.1 Statistics of used ASD Datasets

Dataset	Instances	Family History of ASD	
		Yes	No
<b>Adolescent</b>	248	44	204
<b>Adult</b>	1118	183	935
<b>Child</b>	509	86	423

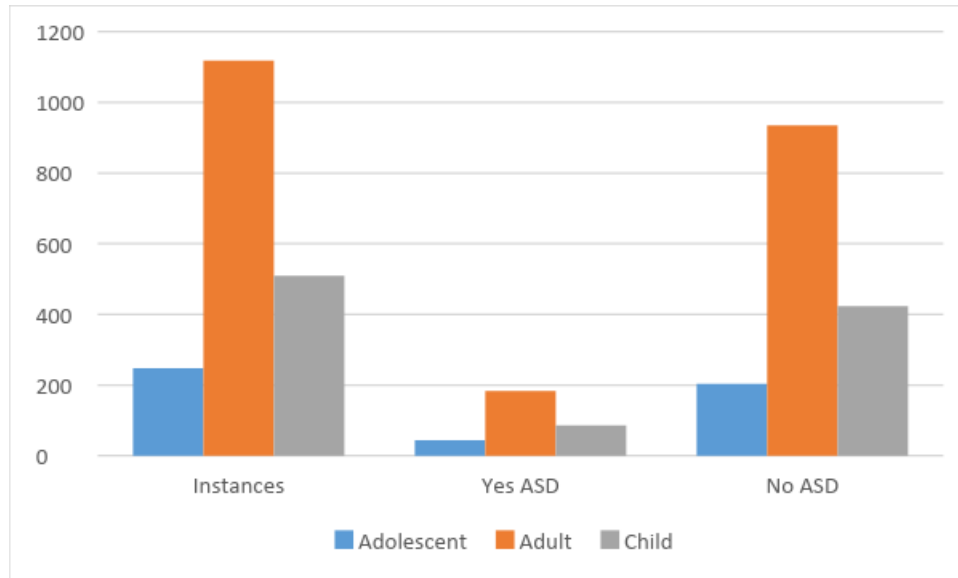


Figure 5.2 Statistics of used ASD Datasets

### 5.3.2 The initial Dataset and Data Transformation

The initial datasets are of multivariable nature with categorical, continuous and binary attributes that contain a total of 23 features (see Table 5.2).

The ASDTest mobile application assigns a “1” if the respondent to any of the questions is “slightly agree” or “definitely agree”, otherwise a zero “0” is allocated for questions 1, 5, 8, and 10 in the AQ-10 Adolescent, questions 1, 5, 7, and 10 in the AQ-10 Child, and questions 1, 7, 8, and 10 in the AQ-10 Adult. A “slightly disagree” or “definitely disagree” had a score of “1” on all remaining questions.

We modified the dataset to include only 16 attributes (1 through 15, and 23) by removing features marked 16-22 in Table 5.2 below in the three datasets. We deem these features to have no direct significance and hence have been discarded a priori to the learning phase. The “Screening Score” (Feature #19 in Table 5.2) has been removed to avoid any possibility of model overfitting since this feature indicates whether individuals have autistic traits based on the scoring function in the AQ-Child 10, AQ-Adult 10 and AQ-Adolescent 10 methods. The screening features (A1 to A10 in Table 5.2) have been transformed by mapping its original values in the screening method to Boolean values 1/0 for the sake of simplicity.

Table 5.2 ASD Data Feature Attributes

#	Feature	Type
1	A1	Binary (0, 1)
2	A2	Binary (0, 1)
3	A3	Binary (0, 1)
4	A4	Binary (0, 1)
5	A5	Binary (0, 1)
6	A6	Binary (0, 1)
7	A7	Binary (0, 1)
8	A8	Binary (0, 1)
9	A9	Binary (0, 1)
10	A10	Binary (0, 1)
11	Age	Integer
12	Gender	String
13	Ethnicity	String
14	Born with jaundice	Boolean (yes or no)
15	Family member with PDD	Boolean (yes or no)
16	Country of residence	String
17	Used the screening app before	Boolean (yes or no)
18	Why_are_you_taken_the_screening	String
19	Screening Score	Integer
20	Screening Method Type	Integer (0,1,2,3)
21	Language	String
22	Who is completing the test	String
23	Class	String

The AQ-10 screening questionnaire is used by the University of Cambridge autism research centre as a referral guide. A sample of the adult questionnaire is provided in Table 5.3 below.

Table 5.3 AQ-10 Adult Questionnaire (Allison et al., 2012)

#	Question
1	"I often notice small sounds when others do not"
2	"I usually concentrate more on the whole picture, rather than the small details"
3	"I find it easy to do more than one thing at once"
4	"If there is an interruption, I can switch back to what I was doing very quickly"
5	"I find it easy to 'read between the lines' when someone is talking to me"
6	"I know how to tell if someone listening to me is getting bored"
7	"When I'm reading a story I find it difficult to work out the characters' intentions"
8	"I like to collect information about categories of things (e.g. types of car, types of bird, types of train, types of plant etc)"
9	"I find it easy to work out what someone is thinking or feeling just by looking at their face"
10	"I find it difficult to work out people's intentions"



### 5.3.3 Unsupervised Clustering Phase

The datasets are pre-processed by applying an unsupervised machine learning clustering method. We employ the OMCOKE algorithm which groups all items into two clusters. The process is summarized in Figure 5.3.

Input: Dataset with N number of attributes

Output: Reduced dataset with N+1 number of attributes

Given a test dataset, the pre-processing works as follow:

2.       Apply Unsupervised ML Clustering (OMCOKE) filter to the dataset
3.       A new "Cluster" attribute is appended on the dataset with each instance on the dataset assigned to either cluster1 or cluster2
3.       Repeat
4.               rename cluster1 as YES and cluster2 as NO
5.               compare attribute "Cluster" with attribute "Class"
6.               If Cluster Matches Class
7.                       Keep instance
8.               Else
9.                       Prune instance
10.              End if
11.       **Until** all instances in the dataset are exhausted

Figure 5.3 Pseudocode of Clustering phase in CATC method

The OMCOKE clustering technique assigns instances to either cluster1 or cluster2 based on their attribute similarities. The OMCOKE algorithm is based on K-means where initial  $k$  clusters are selected at random, and data points are assigned to each cluster using distance to the centroids. The centroids are recomputed, and the process is repeated until there is no movement or change in the assignment of data points to their closest centroid. The OMCOKE algorithm takes into consideration outlier or noise data in the dataset and separates these points to an outlier cluster built on the fly. Algorithm 5.3 above summarizes the OMCOKE clustering.

#### **5.3.4 Clustering based Autistic Traits Dataset: Initial Model**

The datasets contain a Boolean attribute named "Class" that has a value of YES/NO based on a Score. This attribute Class is used to assess whether the user has been screened to have ASD or not and is used in the supervised learning algorithm for their predictions. At the end of step 2 in the CATC pre-processing phase above, we create a new attributed "Cluster" that is appended at the end of the dataset file. Each item is assigned to either cluster1 or cluster2 based on their attribute similarities. These assignments are then compared to the attribute Class to see if they match. Where there is a match, we keep that instance; otherwise we discard it and remove it from the dataset.

The new reduced clustering based autistic dataset only has instances that the clustering algorithm deems to have been accurately labelled during the unsupervised screening process. Key features of applying CATC process includes:

1. Grouping the data items into two clusters based on their strong attributes. The clustering algorithm has assisted in identifying relevant and strong features that were only used in the supervised learning models.
2. Reduce data dimensionality by eliminating redundancy. By clustering the significant features and comparing them to the class score, we toss out any insignificant or redundant items.

We adopt the clustering based autistic traits dataset which has been efficiently streamlined and enhanced to be used in the learning phase in the machine learning process. For example, assume the following simple dataset represented in figure 5.4 below as our original data.



Figure 5.4 Sample Dataset

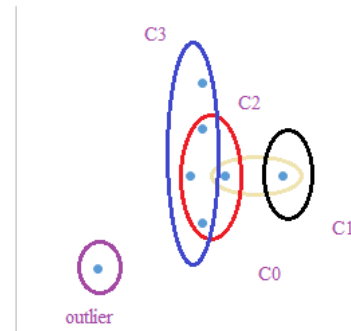


Figure 5.5 Clustered dataset

The clustering algorithm groups the data based on its strong attributes which are identified in C2 (red) and C3 (blue) clusters and the data anomaly is labelled as an outlier and discarded for use in the supervised learning model as indicated in the example on figure 5.5 above.

### 5.3.5 Classification

Finally, we adopt any classification algorithm for our predictive phase. Classification algorithms are generally divided into a two-step process where the dataset is divided into training data and testing data. A model is developed in the training phase by analysing the attributes of the training data. Class labels are built based on the rule techniques that are applied in the training dataset. This training data is further employed in the testing phase where the classifier is used to examine the accuracy of the model (Thabtah et al., 2011). We validate and evaluate the test dataset for better accuracy, sensitivity and specificity rates. In this paper, we tested large numbers of classification algorithms that utilize various district learning mechanisms in order to test the performance of the clustering phase (See section 5.4 for further details).

## 5.4 Experimental Settings

Our experiments are conducted on real-life ASD screening datasets to measure the effectiveness of the enhanced screening data used to identify and predict diagnosis. The three datasets of adult, adolescence, and child have a wide diversity in their ethnicity, language, and age group and are all in the application domain of the study, hence making it suitable for use as benchmarks.

We describe some common predictive model evaluation criteria such as accuracy, sensitivity, specificity, one-error, harmonic mean a.k.a. F1, and other related measures such as false positive (FP), false negative (FN), true positive (TP), and true negative (TN).

All experiments have been run on an Intel Core i7 computer with a 3.4 GHz processor and 8.0 GB RAM running on a 64-bit Windows 10 Operating System. We utilized a number of evaluation measures to show the benefits and negatives of the proposed algorithm when compared with other classification algorithms in DM.

The *Sensitivity* ratio (equation 4.1) is a measure of all cases that have been identified correctly to have ASD in the overall test cases i.e. the true positive rate, whereas the *Specificity* ratio (equation 4.2) is a measure of all cases that have been identified correctly as a No ASD in the overall test cases, i.e. the true negative rate. The *Accuracy* ratio measures the overall classification prediction that has been correctly identified as ASD and No ASD in all test cases (i.e., the confidence level of the classification), whereas the One error is the opposite of *Accuracy* and denotes the number of misclassified instances on the test dataset.

We conduct the experimentation twice for each dataset.

Different classification algorithms have been utilized to measure the true performance of the proposed framework (CATC). Particularly, we adopted RIPPER (Cohen, 1995), PART (Frank & Witten, 1998), Random Forest (Breiman, 2001), Random Trees (Cutler & Zhao, 2001), and Artificial Neural Network [ANN] (Witten & Frank, 2005) algorithms to process the considered

autism datasets with and without clustering. Thus two types of experiments have been conducted per dataset as follows:

Experiment (1): We first load the original datasets (adult, adolescent, child) without any clustering. Then we run the classification algorithms (RIPPER, PART, Random Forest, Random Trees, and Artificial Neural Network [ANN]) using their default settings and record their output results.

Experiment (2): CATC processed dataset in which the clustering is applied and all default settings of OMCOKE are maintained except the number of  $k$  clusters is changed from the default 3 to  $k = 2$ . Once this data has been pre-processed, then it is run using the classification algorithms above.

A tenfold cross-validation testing method on all the classifiers has been deployed in all experiments. This means that the dataset is partitioned into ten subsets where nine data subsets are used for the training phase and one subset for the prediction phase. The process is then repeated 10 times. This will reduce overfitting and ensure a fair evaluation of the derived classifiers.

## ***5.5 Empirical Results and Analysis***

The experiments were conducted for the three datasets, i.e. adult, adolescent, and child. The tables and the figures below show side by side comparison of the machine learning classifiers performance with/out CATC integration. The column CATC is marked as "No" when CATC was not applied to the dataset; otherwise a "Yes" is indicated. Table 5.3 compares the overall classification prediction, i.e. accuracy rate for ML classifiers, noting a significant improvement when CATC is applied before the classification procedure.

Table 5.4 Accuracy Rates of the Classifiers

<b>Dataset</b>	<b>Classifier</b>	<b>CATC Clustering</b>	<b>Adult</b>	<b>Adolescent</b>	<b>Child</b>
<b>Accuracy</b>	RIPPER	No	0.942	0.807	0.878
		Yes	0.969	0.944	0.936
	PART	No	0.962	0.879	0.916
		Yes	0.970	0.917	0.971
	Random Forest	No	0.972	0.911	0.951
		Yes	0.990	0.978	0.990
	Random Tree	No	0.924	0.863	0.874
		Yes	0.998	0.961	0.997
	ANN	No	0.980	0.992	0.980
		Yes	0.999	0.978	0.999

Table 5.4 shows the accuracy rate of the models derived by the ML methods on the adult, adolescent, and child datasets. In all cases, the accuracy of the classifier has been improved by the ML method when CATC was applied prior training phase. In particular, RIPPER had seen an increase in the accuracy rate by 2.7%, 13.7% and 5.8% for the adult, adolescent, and child datasets respectively when CATC was applied on these datasets. In addition, PART predictive accuracy had improved by 0.8%, 3.8% and 5.5% on the three datasets respectively when CATC was applied. Similarly, Random Forest and Random Tree classifiers when integrated with CATC have improved (2.8%, 6.7%, and 4.9%) and (7.4%, 9.8%, and 12.3%) respectively. No significant change is noted in the ANN method. The significant improvement in the accuracy can be attributed to the fact that having clustering in the pre-processing phase of the dataset was able to reduce data dimensionality by eliminating redundancy in the dataset.

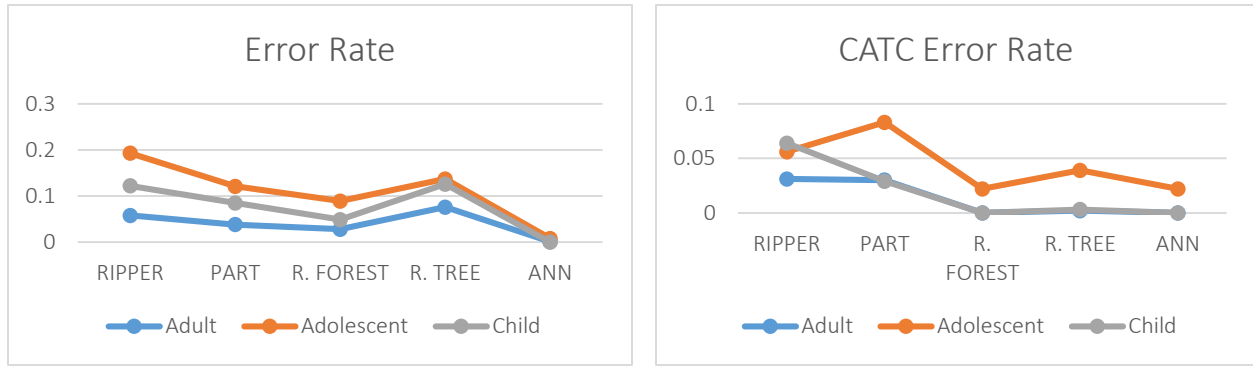


Figure 5.4 Error rate comparisons

Figure 5.6 shows the error rates comparisons of the models derived by the ML methods with and without CATC. The figure reveals that when CATC is applied before classification, the derived classifiers have shown a reduction in the error rates by 2.7%, 0.8%, 2.8%, and 7.4% for RIPPER, PART, Random Forest, and Random Tree respectively in the adult dataset. For the adolescent dataset, classifiers derived when CATC was applied have a 13.7%, 3.8%, 6.7%, and 9.8% lower rates for RIPPER, PART, Random Forest, and Random Trees respectively. Finally, for the child dataset, the error rates are lower when CATC was used by 5.8%, 5.6%, 4.9% and 12.3% for RIPPER, PART, Random Forest, and Random Tree respectively. There is no significant change in the ANN classifier. This shows overall better accuracy and lower error rates for all datasets including those that have large numbers of instances, i.e., adult dataset, and those with a lower number of instances, i.e., the adolescent dataset.

The accuracy rate alone may not be the best measure of performance because even with a 95% accuracy rate we might simply be predicting majority class correctly. Our focus, however, should be the other 5% minority class who might have been screened and diagnosed with autism. Thus, a good predictor of the model performance would be the true positive rate (sensitivity) and the true negative rate (specificity).

Figure 5.7 shows the specificity and sensitivity results of the three datasets by the classifiers with and without CATC. The figure reveals that when CATC was utilized prior to learning the sensitivity and specificity rates of the ML have improved on all datasets. For example, in RIPPER

algorithm case, there is a modest sensitivity rate improvement on the adult and child datasets (2.9% and 2.8% respectively) and a 6.8% on the adolescent dataset, when CATC was applied. In addition, PART classifier's sensitivity rate went up by 0.9%, 6.9% and 7.5% on the adult, adolescent, and child respectively, when CATC was integrated. Similarly, when CATC was applied, Random Forest and Random Trees classifiers observed (1.8%, 11.1%, and 5.2%) and (5.3%, 8.1%, and 14.5%) increase in the sensitivity rates respectively. There is only a minuscule change in the ANN classifier.

Here we note that clustering the datasets identified and grouped similar cases that would have otherwise been difficult to be classified correctly. Cases that may have exhibited some autistic traits but not qualified to be on the spectrum due to overlapping features of No ASD with ASD cases. These cases tend to confuse the learning algorithm in the classification process hence causing large false positives and false negatives.

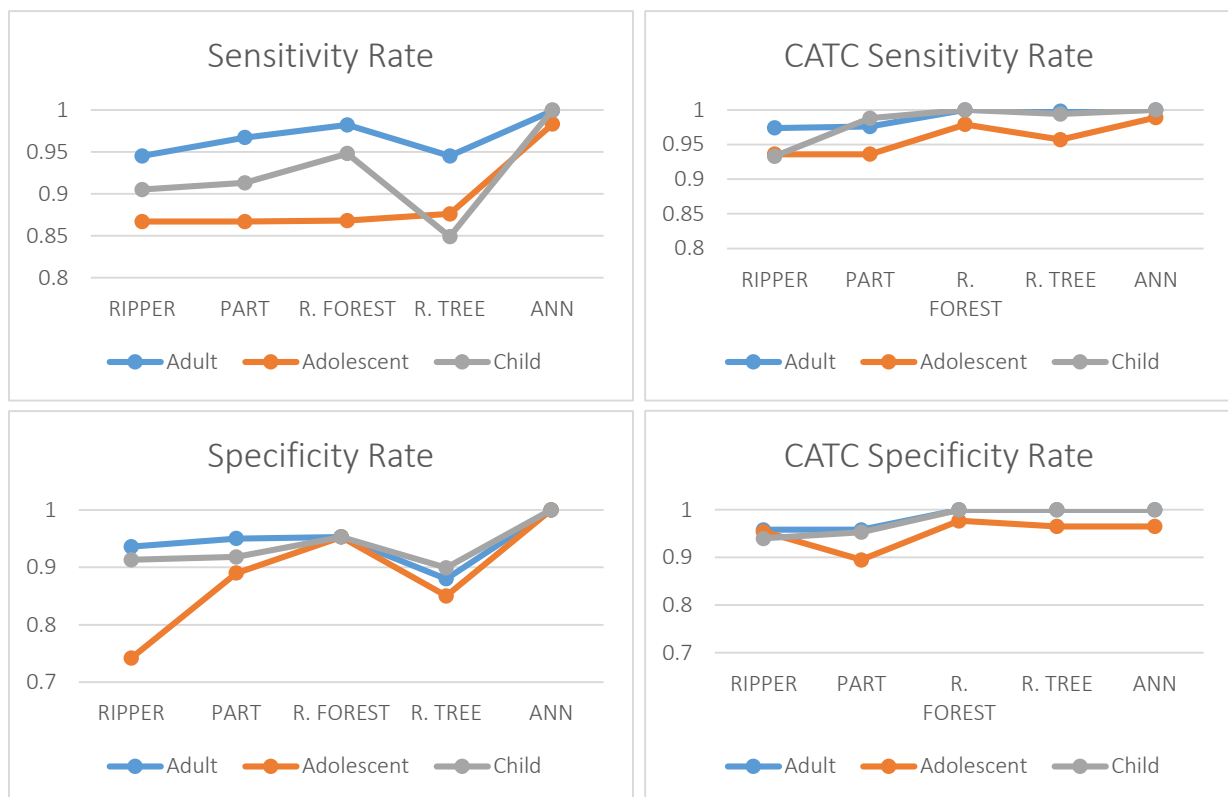


Figure 5.5 Sensitivity & Specificity Rates of the Classifiers



By considering the overlaps and clustering them based on the similarity of their attributes, this issue is resolved, and the performance of the classification algorithms is dramatically improved as shown in Figure 5.7 and Table 5.4.

The specificity rates as shown in Figure 5.3 had seen an improvement of 2.2%, 0.8%, 4.7% and 12% for the adult dataset on the classifiers RIPPER, PART, Random Forest, and Random Tree respectively when CATC was applied. On the adolescent dataset, when CATC was applied, the percentage increment of the RIPPER, PART, Random Forest, and Random Tree classifiers are 21.2%, 0.5%, 2.4%, and 11.5% respectively. Similarly, the performance of the classifiers went up by 2.7%, 3.5%, 4.7%, and 10.1% respectively on the child dataset.

To further understand the sensitivity and specificity rates, we investigated the confusion matrix results produced by the classifiers. Of all the three datasets, the adult dataset had the overall highest number of incorrectly classified instances by the RIPPER, PART, Random Forest, and Random Tree classifiers, whereas, the adolescent dataset had the least. Random Tree had the highest number of incorrectly classified instances (85) followed by RIPPER (65), PART (43), and Random Forest (31) in the adult dataset. Specifically, Random Tree predicted 43 instances with ASD traits that should not have been classified resulting in the lowest specificity rate among the classifiers. On the other hand, Random Forest had the lowest number of false negatives with only 17 instances. CATC improved the classifiers by reducing the number of incorrectly classified instances to 21, 20, 0 and 1 for RIPPER, PART, Random Forest, and Random Tree respectively. In the adolescent dataset, CATC significantly reduced the incorrectly classified instances by the classifiers RIPPER, PART, Random Forest, and Random Tree from 48 to 10, 30 to 15, 22 to 4, and 34 to 7 respectively. Thus, CATC classifiers showed improvement in both sensitivity and specificity rates across the board compared with all classifiers.

With respect to the imbalance in the adult dataset due to the class variable, we included the F1 metric also known as the harmonic mean that not only takes into consideration the precision but also the sensitivity (equation 4.1 above). The F1 measure for the classifiers is shown to have

increased by 12.3%, 0.9%, 2.8%, and 7.4% for RIPPER, PART, Random Forest, and Random Tree respectively when CATC was utilized (See Figure 5.8).

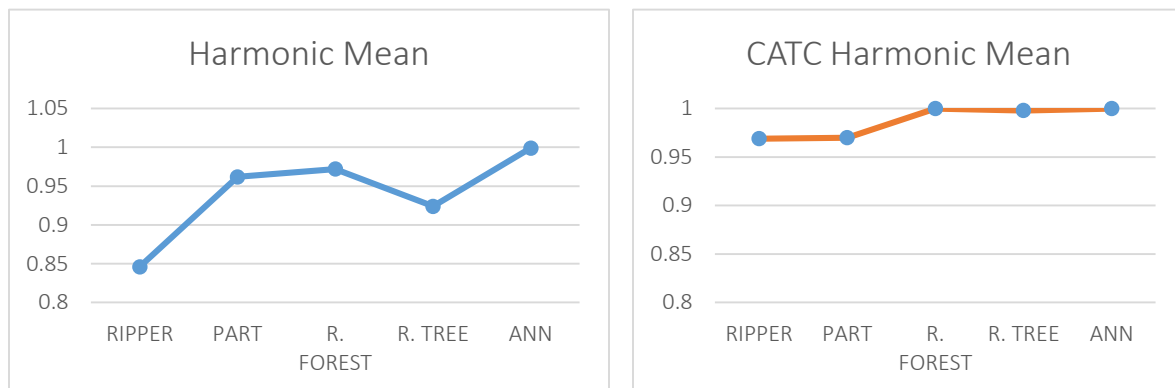


Figure 5.6 Harmonic mean on the classifiers

The Receiver Operating Characteristic (ROC) is an evaluation measure that contrasts the true positives and false positives of the machine learning model. The measure contrasts how the number of correctly classified true positives with the number of incorrectly classified negative values. Figure 5.9 summarizes the ROC values of the classifiers and improvement across the board in the ROC Area rates when CATC is applied. While the improvement rates were decent in the adult dataset, it was slightly significant in the smaller dataset such as the Adolescent dataset. The ROC Area rate was up by 12.3%, 1.0%, 1.6%, and 9.8% on RIPPER, PART, Random Forest, Random Tree classifiers respectively, when CATC was utilized prior to learning.

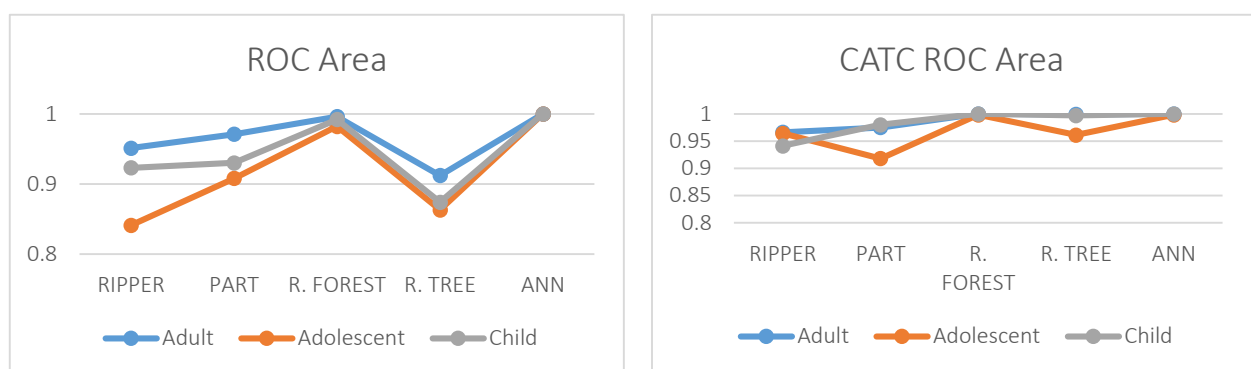


Figure 5.7 ROC Area of the classifiers

We also note that the number of rules generated while running the three datasets on RIPPER and PART decrease when CATC is applied as shown in figure 5.10 (below).

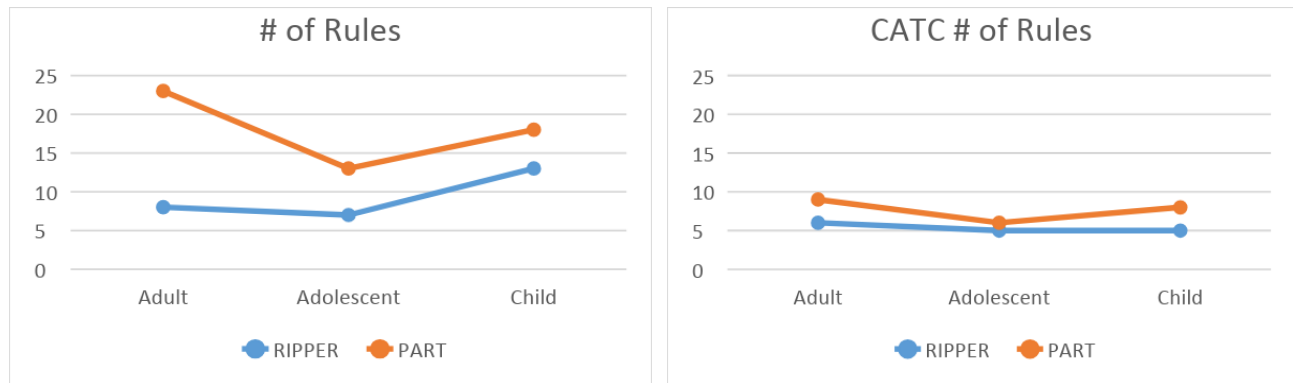


Figure 5.8 # of Rules Generated in PART and RIPPER classifiers

This can be attributed to the fact that redundant rules have been removed in the building of the classifier due to the pre-processing of the dataset and clustering them based on their strong attributes. Thus, the pre-processing with clustering algorithm have assisted in identifying relevant and strong features that were only used in the supervised learning models. This is useful for diagnosticians as fewer rules could mean a reduced amount of time needed in the screening of autism patients.

## 5.6 CHAPTER SUMMARY

In this chapter, Autism Spectrum Disorder has been described along with some of its challenges and the use of ML in the screening process. The screening and diagnosis process is a very lengthy process and inaccessible to the vast majority of people. Screening methods have been developed using existing clinical autism diagnosis methods that take the answers given in a questionnaire format as an input for a scoring function, which in turn processes the input and computes a final score to reflect whether the individual is associated with ASD traits. Recent research has aimed at minimizing the number of questions and focused primarily on improving time, accuracy, and reducing the dimensionality of the dataset by pinpointing influential autistic symptoms. The focus of this chapter was the utilization of clustering and classification together as semi-supervised learning since this form of semi-supervised technique is rare in autism screening research.

In this chapter, we proposed a method that utilizes both clustering and classification in autism screening, a first that we are aware of. The study utilized data obtained via a screening app available on both Android and iOS mobile users and accessible easily online by the users. By introducing clustering a priori to classification, this study was able to add value to existing research in four folds;

- 1) This method was able to reduce data dimensionality by eliminating redundancy in the dataset.
- 2) Cases that may have exhibited some autistic traits but not qualified to be on the spectrum due to overlapping features which caused large false positives and false negatives were resolved.
- 3) The technique used did not rely on the scoring function feature popularly used in other research to determine autistic traits in the screening phase but instead used an unsupervised ML clustering algorithm to identify features based on their similarity measures.
- 4) Clustering the data before application in the learning phase streamlined the data based on only strong features resulting in a reduced number of rules generated by the classifiers.

The proposed model was measured by looking at the accuracy, sensitivity, specificity, and F1 rates and compared to popular classifiers (ANN, RIPER, PART, Random Forest, and Random Trees). CATC improved the classifiers by reducing the number of incorrectly classified instances and improved on the sensitivity and specificity rates on the classifiers. The results also showed a significant reduction in the rules generated by PART.

There were a couple of limitations in this study. The data used was limited to what was collected using the mobile app. The study could have benefited with larger balanced datasets. Also, instances related to toddlers are rare and hard to obtain and were not included in this study.

In conclusion, the case study showed employing CATC in the screening phase significantly improved the performance of the classifiers in all measures and especially the accuracy and sensitivity rates, thus making a substantial positive difference in the prediction of the ASD

diagnosis class. The method used in this study can easily be adopted and applied to other clinical science application domain such as screening for dementia.

The thesis conclusion and future research directions are discussed in the next chapter.

## Chapter 6

### Conclusions and Future Work

#### **6.1 Research Summary**

This work has investigated an issue related to the unsupervised learning technique, especially overlapping partitioning clustering. We studied a number of enhancements in the application of the K-means algorithm in clustering for multi-label datasets. The outcome of this study is a new multi-cluster algorithm called OMCOKE, which contains a novel characteristic of detection and retention of outliers and the ability to determine overlap threshold through heuristics as opposed to user-defined entry. These features, when combined with classification algorithms, have tremendously improved the prediction accuracy and efficiency of both clustering and classification. The improvements and the K-means modifications in this work have been presented in IEEE conferences and have been published in reputable statistics and data mining and analysis journals.

#### **6.2 Research Contributions**

The following sections summarize the different contributions of this research that were outlined in chapter one.

##### **6.2.1 Overlapping Clustering with self-calculating threshold**

One of the significant challenges of K-means clustering algorithms is to determine a suitable threshold that would be used to allow objects to belong to overlap and belong to multiple clusters. Many algorithms require a similarity threshold be determined in advance which is used to determine whether an object will belong to a certain cluster or not whether it can overlap between multiple clusters. This may not be an easy feat to determine a priori for different datasets having different cardinalities.

The OMCOKE algorithm assigned the global threshold to determine the belonging of a data object to a cluster once the K-means algorithm finishes its iterations and picks the maximum distance (*maxdist*) of all objects that were assigned to the clusters. The *maxdist* which is based on the maximum Euclidean distance assigned to all the objects becomes the global threshold to reassign data to overlapping clusters. In this case, the threshold *maxdist* is not determined a priori and is based on heuristics that can change dynamically depending on the data.

The implementation and evaluation of MCOKE have been disseminated and published in Proceedings of the XIII International Conference on Machine Learning and Computing, ICMLC'2015, IEEE SAI Computing Conference, London, UK and in the Canadian University Dubai Speaker Series 2015.

### **6.2.2 Noise and Outlier detection**

Outliers are data objects or points that do not conform to the normal behaviour or model of the dataset, hence are deemed inconsistent or grossly different. Outliers could also indicate suspicious data objects with malicious intent. We modified our algorithm to detect and retain the outliers. Upon identification of at least one outlier, the  $k$  number of clusters entered by the user prior to running the method is incremented by 1 on the fly; the outlier object is assigned to this newly created cluster. All other identified outliers, a subset  $S$  from the initial population, are assigned to belong to this newly created cluster. Once an outlier is detected, the algorithm adds  $k+1$  clusters as the new output vector with the outlier cluster indexes listed as part of the output. This allows for further investigation of those data points as opposed to discarding them as is usual. When no outliers are detected, the algorithm will simply cluster with overlaps without incrementing the number of  $k$  clusters.

The implementation and evaluation of OMCOKE have been submitted to the Statistical Analysis and Data Mining journal.

### **6.2.3 Clustering based Autistic Trait Classification technique to improve ASD Screening: A Case Study**

One of the critical applications that machine learning (ML) can be adapted to is to improve the detection of autistic symptoms in Autism Spectrum Disorder (ASD) screening. In this context, ML techniques can be used to discover ASD symptoms based on historical cases and controls to enhance autism screening efficiency and accuracy. ML offers advanced techniques for discovering concealed information that can be utilized by physicians, clinicians as well as parents to improve medical diagnosis and screening. This case study aimed to improve the performance of detecting ASD traits by reducing data dimensionality and eliminating redundancy in the autism dataset. To achieve this aim, a new ML framework using a semi-supervised learning approach called Clustering based Autistic Trait Classification (CATC) is proposed in chapter 5 in which detecting autistic traits is accomplished using a clustering technique and validation of the classifiers is done by a classification technique.

In chapter 5, empirical results on different datasets related to screening of autism involving children, adolescents, and adults collected using an online mobile application were verified and compared to other common machine learning classification techniques. We measured our model by looking at the accuracy, sensitivity, specificity, and F1 rates and compared them to popular classifiers (ANN, RIPER, PART, Random Forest, and Random Trees).

This experimental study on ASD was submitted to a top-ranked journal, Informatics for Health and Social Care by Taylor & Francis.

### **6.2.4 Outlier Detection Clustering Technique in WEKA ML Tool**

One major contribution of this thesis is the application and integration of OMCOKE algorithm in the WEKA environment inside the "clusterers" package. The "clusterers" package contains the implementation of a few clustering algorithms such as the Simple K-means, EM (Expectation-Maximization) algorithm, Hierarchical Clustering, Cobweb, Density-Based Clustering among



others. The proposed algorithm can be accessed from WEKA Explorer or the Command Line platforms for data processing.

### **6.3 Research Implications and Limitations**

The following is a summary of some of the implications of this research.

- a) Our outlier clustering approach, which is discussed in detail in Chapter 3 and Chapter 4 added immense value to the learning process as we save these data objects in a “k+1” cluster to investigate and understand their characteristics. An outlier cluster that can be investigated has profound real-life implications such as in e-banking (fraud detection), website phishing, cyber security (intrusion detection), or medical screening (ASD or dementia).
- b) Our case study method, which is discussed in detail in Chapter 5 identified potential autism cases based on their similarity traits as opposed to a scoring function used by many ASD screening tools. Not only did the proposal improve the performance of the classifiers and reduced the number of rules generated by the algorithms, but also resolved cases that may have exhibited some autistic traits but not qualified to be on the spectrum due to overlapping features which caused large false positives and false negatives. This has great potential to improve the screening process and directly benefiting diagnosticians and society at large.
- c) There are many advantages of integrating OMCOKE to WEKA. WEKA being an open-source application tool can be downloaded freely online and puts the OMCOKE algorithm directly in the hands of many machine learning researchers and students. Therefore, a positive impact on the enhancement of literature in this domain can be achieved easily.

There are a few limitations encountered in this research primarily in the application of the autistic traits detection case study. We list these limitations as:

- a) The limited dataset used – The study relied on data collected via the mobile app ASDTest. This app was available for download worldwide in both Android and Apple iOS devices. However, only a few

participated in this study. There were 248 adolescents, 1118 adults, and 509 children with 44, 183, and 86 that identified to have a family history of ASD respectively.

- b) Non-inclusion of toddlers – The dataset did not include instances of toddlers. While instances related to toddlers are rare to find, it would have nonetheless enriched the study as ASD screening of young toddlers may provide a better understanding of the symptoms at a very young age and increase the chances of early treatment.
- c) Lack of clinicians’ verification of the results - Classification algorithms play a crucial role in providing a predictive model. However, these models need to be verified by licenced clinicians and specialists who can attest to the classification accuracy.

## **6.4 Future Work**

### **6.4.1 Fusing of Multi-Labelled Clusters**

The OMCOKE algorithm can be extended to increment the  $k$  number of clusters based on the matches and mismatches of the object attributes by assigning weightage to the objects. These multi clusters can then be fused as new clusters based on their similarities and providing the mapping of objects to the clusters with their similarity weights. The following outline the steps:

- a) get user preferred number of  $k$  clusters
- b) Adjust the  $k$  number of clusters to better fit the data by allowing a maximum number of clusters based on the user entry
- c) Upon successful clustering of the data objects, calculate the weightage of each object based on their similarity to the cluster centroid
- d) Fuse and merge multi-clusters based on the object weightage for better multi-labelled clusters and provide a mapping of the objects to the fused clusters

For example, let us assume a Netflix customer is presented with 3 movie genre options to select (i.e., horror, drama, or thriller). The customer selects all three and searches the database which

yields the following scenario of overlapping results of 7 different movie titles (A, B, C, D, E, F, G).

Table 6.1 Search Results of Movie Titles

Movie Title	Horror	Drama	Thriller
A	x	x	
B	x		
C		x	x
D	x	x	x
E	x		x
F		x	
G			x

The 7 movies will have been clustered in 3 categories with some movie titles overlapping between the different genres. These repetitive results can be summarized as:

- 4 movies of horror (A, B, D, E)
- 4 movies drama (A, C, D, F)
- 4 movies of thriller (C, D, E, G)

Based on the number of categories  $k$  selected, the maximum number of clusters (Y) can be computed using the formula  $Y = 2^k - 1$ . In this scenario, the clustering algorithm can be expanded to create 7 clusters (instead of the  $k=3$ ) with the clear options of the overlapping genres.

- A movie – horror/drama
- C movie – drama/thriller
- E movie – horror/thriller
- D movie – horror/drama/thriller
- B movie – horror
- F movie – drama
- G movie – thriller

These assignments can now be enhanced by attaching weightage of the genre to each movie.

For instance,

- A movie – horror (0.6) / drama (0.4)
- C movie – drama (0.7) / thriller (0.3)
- E movie – horror (0.55) / thriller (0.45)
- D movie – horror (0.2) / drama (0.4) / thriller (0.4)
- B movie – horror (1)
- F movie – drama (1)
- G movie – thriller (1)

Given the formula, the number of clusters can grow exponentially when  $k$  is large. Hence this has to be minimized to only small values of  $k$ . The resulting clusters can then be fused to reduce the number of clusters based on their similarity weightage and provide better classification labels.

In the example above, the clusters can now be merged and reduced to form a maximum of 4 clusters based on the object weightage yielding the following results;

- Cluster A (generic genres)
  - B movie – horror
  - F movie – drama
  - G movie – thriller
- Cluster B (mostly horror)
  - A movie – horror (0.6) / drama (0.4)
  - E movie – horror (0.55) / thriller (0.45)
- Cluster C (mostly drama)
  - C movie – drama (0.7) / thriller (0.3)
- Cluster D (drama-thriller)
  - D movie – drama (0.4) / thriller (0.4)

This clustering technique provided a better categorization of the movie and will optimize the customer experience due to the improved classification of the clusters labels based on their weightage of each genre.

The future algorithm will first determine the maximum number of clusters based on the user-defined  $k$ . Data objects will be assigned to each cluster based on their similarity measure allowing objects to overlap to multiple clusters. This is followed by calculating the weightage of each object to their assigned cluster(s). Finally, the clusters are fused and merged based on their

object similarity weightage to create better multi-labelled clusters, and a mapping of each object to the cluster is provided.

### **6.4.2 Distributed Overlapping Clustering with MapReduce**

The literature review and critical analysis in chapter two, we find that multi-label datasets with large dimensionality and cardinality of the overlaps affect the value of Precision and Recall i.e., there will be a low value of Precision because the observations are assigned to more than one cluster. Recall measure uses a binary function to compute the relationship between pairs of observations, and not considering that those pairs of observations could also feature in multiple clusters in the overlap. This results in a biased recall measure, especially when the cardinality in the dataset is large. We also note the effect of outliers in the dataset and how easily they can skew and affect the mean distance calculation used in assigning objects to their clusters. These shortcomings effect even when such moderate sized multi-label datasets are processed locally. The problem becomes stark when we are to consider big data distributed over different locations. It is imperative we device new approaches in overlapping clustering of distributed big data.

Hadoop and MapReduce are popular platforms used today for distributed data processing. These two emerging technologies are increasingly utilized in the real-life implementation of mining big data where the semi-structured data is distributed over large areas and to multiple locations. The Hadoop platform uses distributed file systems and parallel processing to create clusters of computer nodes to store and process data across multiple locations. On the other hand, MapReduce is an open-source API (application programming interface) that distributes the processing of data among thousands of distributed nodes in parallel. The Map function divides a task into smaller units among thousands of nodes, and the Reduce function integrates the outputs derived from the nodes into a single set. Many large social media and technology giants such as Facebook, Twitter, Google, and Amazon have adopted these two techniques for their big data processing and analytics. In a world where massive amounts of unstructured and semi-structured data are generated every minute through different devices and technologies, there is

a need for overlapping clustering techniques to handle such data in a distributed and parallel way by integrating overlapping clustering algorithms in the MapReduce API.

# Bibliography

1. Abbas, H., Garberson, F., Glover, E., & Wall, D. P. (2018). Machine learning approach for early detection of autism by combining questionnaire and home video screening. *Journal of the American Medical Informatics Association*. Vol. 25, No. 8, pp. 1000-1007.
2. Abbas, O. A. (2008). Comparisons between Data Clustering Algorithms. *The International Arab Journal of Information Technology*, Vol 5. No. 3.
3. Abdelhamid N., Ayesh, A., Hadi W. (2014). MCAC: Multi-label Rules Generation via Parallel Associative Classification. *Parallel Processing Letters journal*. Vol 24, No.1, pp. 1-24.
4. Achenbach, T. M., Rescorla, L. A. (2001). *Manual for the ASEBA school-age forms & profiles*. Burlington, VT: University of Vermont, Research Centre for Children, Youth, & Families.
5. Aggarwal, C., & Reddy, C. K. (2014). *Data clustering: Algorithms and applications*. CRC Press.
6. Allison, C., Auyeung, B., & Baron-Cohen, S. (2012). Toward brief “Red Flags” for autism screening: The short autism spectrum quotient and the short quantitative checklist for autism in toddlers in 1,000 cases and 3,000 controls. *Journal of the American Academy of Child Adolescent Psychiatry*, Vol. 51, No. 2, pp. 202–212.
7. Ankerst, M., Breunig, M., Kriegel, H., and Sander, J. (1999). OPTICS: Ordering points to identify the clustering structure. In *SIGMOD conference*, pp. 49-60.
8. Arabie, L. J., Hubert, G., & DeSoete, P. (1999). *Clustering and classification*. World Scientific.
9. Arthur, D., and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027-1035.
10. Auyeung, B., Baron-Cohen, S., Wheelwright, S., & Allison, C. (2008). The autism spectrum quotient: Children's version (AQ-Child). *Journal of Autism Development Disorder*, 38 (7), Pp. 1230–1240.
11. Baadel, S., Thabtah, F., Lu, J. (2015a). Multi-Cluster Overlapping K-means Extension (MCOKE). In *proceedings of the XIII International Conference on Machine Learning and Computing, ICMMLC'2015*.
12. Baadel, S., Thabtah, F, Lu, J. (2015b). *Data Clustering Approaches: A Review of hard and soft partitioning techniques*. Canadian University Dubai Speaker Series, Dubai, U.A.E.
13. Baadel, S., Thabtah, F., Lu, J. (2016). *Overlapping clustering: A review*. IEEE SAI Computing Conference, London, UK. Pp. 233-237.
14. Ball, G.H. (1965). *Isodata, a novel method of data analysis and pattern classification*. Technical report, DTIC Document.

15. Barai, A., & Dey, L. (2017). Outlier detection and removal algorithm in K-means and hierarchical clustering. *World Journal of Computer Application and Technology*, Vol. 5, No. 2, pp. 24-29.
16. Baron-Cohen, S., Hoekstra, R. A., Knickmeyer, R., & Wheelwright, S. (2006). The autism-spectrum quotient (AQ)—Adolescent Version. *Journal of Autism and Development Disorder*. Vol. 36, No. 3, Pp. 343 – 350.
17. Baron-Cohen, S., Wheelwright, S., Skinner, R., Martin, J., & Clubley, E. (2001). The autism spectrum quotient (AQ): Evidence from Asperger syndrome/high-functioning autism, males and females, scientists and mathematicians. *Journal of Autism Development Disorder*, Vol. 31, pp. 5-17.
18. Bay, S., & Schwabacher, M. (2003). Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*.
19. Bandyopadhyay, S., Maulik, U. (2002). Genetic Clustering for Automatic Evolution of Clusters and Application to Image Classification, *Pattern Recognition*, Vol. 35, pp. 1197-1208.
20. Bekerom, B. (2017). Using machine learning for detection of autism spectrum disorder. 26th Twente Student Conference on IT. Enschede, The Netherlands.
21. Belmonte, M. K., Allen, G., Beckel-Mitchener, A., Boulanger, L. M., Carper, R. A., & Webb, S. J. (2004). Autism and abnormal development of brain connectivity. *Journal of Neuroscience*, Vol. 24, pp. 9228–9231.
22. BenN’Cir, C., & Essoussi, N. (2012). Overlapping patterns recognition with linear and non-linear separations using positive definite kernels. *International Journal of Computer Applications (IJCA)*, pp. 1–8.
23. BenN’Cir, C., Cleuziou, G., & Essoussi, N. (2013). Identification of non-disjoint clusters with small and parameterizable overlaps. In *IEEE International Conference on Computer Applications Technology (ICCAT)*, pp. 1–6.
24. BenN’Cir, C., Essoussi, N., & Bertrand, P. (2010). Kernel overlapping k-means for clustering in feature space. In *International Conference on Knowledge discovery and Information Retrieval (KDIR)*, pp 250–256.
25. BenN’Cir, C., Essoussi, N., Bertrand, P. (2010). Kernel Overlapping K-Means for Clustering in Feature Space. In *International Conference on Knowledge Discovery and Information Retrieval (KDIR)*, pp. 250-256
26. Berchtold, S., Keim, D., A., and Kriegel, H. (1996). The X-Tree: An index structure for high-dimensional data. In *VLDB*, 28-39.
27. Berkhin P. (2006) A survey of clustering data mining techniques. In: Kogan J., Nicholas C., Teboulle M. (eds) *Grouping Multidimensional Data*. Springer, Berlin, Heidelberg.



28. Bezdek, J. C. (1981). Pattern recognition with fuzzy objective function algorithms. Kluwer Academic Publishers.
29. Bezdek, J., Ehrlich, R., Full, W. (1984). FCM: the fuzzy c-means clustering algorithm. *Computer and Geoscience* Vol. 10, pp.191–203
30. Bolton P, Macdonald H, Pickles A, Rios P, Goode S, Crowson M, Bailey A, Rutter M. (1994). A case-control family history study of autism. *Journal of Psychology & Psychiatry*. Vol. 35, No. 35, pp. 877–900. doi:10.1111/jcpp.1994.35.
31. Bone, D., Bishop, S., Black, M., ... & Goodwin, M., (2016). Use of machine learning to improve autism screening and diagnostic instruments: effectiveness, efficiency, and multi-instrument fusion. *Journal of Child Psychology and Psychiatry*, Vol. 57, No. 8, pp. 927–37
32. Bone, D., Goodwin, M. S., Black, M. P., Lee, C., Audhkhasi, K., & Narayanan, S. (2014). Applying machine learning to facilitate autism diagnostics: Pitfalls and promises. *Journal of Autism and Developmental Disorders* 45(5), 1–16.
33. Borah, S., Ghose, M. K. (2009). Performance analysis of AIM-K-Means and K-Means in quality cluster generation. *Journal of Computers*, Vol. 19, No. 4, pp. 175-178.
34. Bottou, L, Bengio, Y., (1995). Convergence Properties of the K-Means Algorithms. *Advances in Neural Information Processing Systems* 7, MIT Press, pp. 585–592.
35. Boundaillier, E., & Hebrail, G. (1988). Interactive interpretation of hierarchical clustering. *Intelligent Data Analysis*.
36. Bradley, P.S., Fayyad, U. (1998). Refining Initial Points for K-means Clustering. *15th International Conference on Machine Learning*, pp. 91-99
37. Breiman, L. (2001). Random forests. *Machine Learning Journal*, Vol. 45, No. 1, pp. 5-32.
38. Calinski, T. and Harabasz, J., (1974). A dendrite method for cluster analysis. *Communications in Statistics*, Vol. 3, pp. 1–27.
39. Carlsson, G., Memoli, F. (2010). Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning* Vol. 11, pp. 1425–1470
40. Celebi, M., Kingravi, H., Vela, P. (2013). A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm. *Expert Systems with Applications*. Vol. 40, No. 1, pp. 200-210.
41. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, Vol. 41, No. 3, pp. 1-72.
42. Chen, H., Duan, X. Liu, F., Lu, F., ...& Ma, X (2016). Multivariate classification of autism spectrum disorder using frequency-specific resting-state functional connectivity - A multi-centre study. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, Vol. 64, pp. 1-9.

43. Cleuziou, G. (2008). An extended version of the k-means method for overlapping clustering. In International Conference on Pattern Recognition ICPR, pp. 1–4.
44. Cleuziou, G. (2009). Two variants of the okm for overlapping clustering. *Advances in Knowledge Discovery and Management*. pp 149–166.
45. Cleuzious, G. (2009). Two Variants of the OKM for Overlapping Clustering. *Advances in Knowledge Discovery and Management*, pp 149-166.
46. Cohen, W. (1995). Fast effective rule induction. In proceedings of the Twelfth International Conference on Machine Learning. Tahoe City, California. Morgan Kaufmann.
47. Crane, L., Chester, J., Goddard, L., Henry, L., Hill, E. (2016). Experiences of autism diagnosis: A survey of over 1000 parents in the United Kingdom. *Autism: The International Journal of Research and Practice*, Vol. 20, No. 2, pp. 153-162.
48. Cutler, A., Zhao, G. (2001). PERT-perfect random tree ensembles. *Computing Science and Statistics*, Vol. 33, pp 490-497.
49. Duda, M., Ma, R., Haber, N., & Wall, D. P. (2016). Use of machine learning for behavioural distinction of autism and ADHD. *Translation Psychiatry*, Vol. 9, No. 6, pp. 1-5.
50. Dunn, J. C. (1973). A fuzzy relative of ISODATA process and its use in detecting compact separated clusters. *Journal of Cybernetics*. Vol. 3, No. 3, pp. 32-57.
51. Elisseeff, A., & Weston, J. (2001). A kernel method for multi-labelled classification. In T.G. Dietterich, S. Becker, and Z. Ghahramani, (eds), *Advances in Neural Information Processing Systems*.
52. Ester, M., Kriegel, H., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spartial database with noise. In KDD, pp. 226-231.
53. Ester, M., Kriegel, H., Sander, J., Wimmer, M., and Xu, X. (1998). Incremental clustering for mining in a data warehousing environment. In VLDB, pp. 323-333.
54. Everitt, B.S., Landau, S., Leese, M. (2001). *Cluster Analysis*, Arnold Publishers.
55. Fellows, M. R., Guo, J., Komusiewicz, C., Niedermeier, R., and Uhlmann, J. (2011). Graph-based data clustering with overlaps. *Discrete Optimization*, Vol. 8, No. 1, pp. 2–17.
56. Feyyad, U. (1996). Data mining and knowledge discovery: making sense of data. *IEEE Expert*. Vol. 11, No. 5, pp. 20-25.
57. Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, Vol. 2, pp. 139-172.
58. Fisher, D. (1995). Optimization and simplification of hierarchical clustering. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 118-123.

59. Forgy, E. W (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, Vol. 21: pp. 768-769.
60. Fraley, C. and Raftery, A. E. (1998). How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis, Technical Report No. 329, Department of Statistics University of Washington, 1998.
61. Fraley, C., Raftery, A.E. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*. Vol. 97(458) pg. 611–631.
62. Frank, E., & Witten, I. (1998). Generating accurate rule sets without global optimisation. *Proceedings of the Fifteenth International Conference on Machine Learning*, pg. 144–151. Madison, Wisconsin.
63. Gan, G., & Ng, M. K. (2017). K-means clustering with outlier removal. *Pattern Recognition Letters*, Vol. 90, pp. 8-14.
64. Grambeier, J., and Rudolf, A. (2002). Techniques of cluster algorithms in Data mining. *Data Mining and Knowledge Discovery*. Pp. 303-360.
65. Guha, S., Rastogi, R., and Shim, K. (1998). Cure: An efficient clustering algorithm for large datasets. In *ACM SIGMOD Record*, Vol. 27, pp. 73-84. ACM.
66. Hadi, W., Thabtah, F., Abdelhamid, N., Issa, A. (2008). Naive bayesian and k-nearest neighbour to categorize arabic text data. *Proceedings of the European Simulation and Modelling Conference*. Le Havre, France
67. Han J, Kamber M, Pei J. (2006). *Data mining: concepts and techniques*. Morgan Kaufmann.
68. Han, J. Kamber, M. and Pei, J. (2011). *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers.
69. Höppner, F. Klawonn, F., Kruse, R., Runkler, T. (1999). *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*, Wiley.
70. Hruschka, E. R. de Castro, L. N., Campello, B. (2004). Evolutionary Algorithms for Clustering Gene-Expression Data, In *Proc. 4th IEEE Int. Conference on Data Mining*, pp. 403-406
71. Hrushka, E. R., Campello, R., Freitas, A., & Carvalho, A. (2009). A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and cybernetics, Part C., Applications and Reviews*, Vol. 39, No. 2, pp. 133-155.
72. Huang, Z. (1998). Extensions to the k-means algorithm for clustering large datasets with categorical values. *Data mining and knowledge discovery*, Vol. 2, No. 3, pp. 283-304.
73. Huang, J. Z., Ng, M. K., Rong, H., Li, Z. (2005). Automated variable weighting in k-means clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27, No. 5, pp. 657-668.

74. Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*. Vol. 31, No. 8, pp. 651–666.
75. Jain, A. K., Murty, M.N., Flynn, P.J. (1999). Data Clustering: A Review, *ACM Computing Surveys*, Vol. 31, No. 3, pp. 264–323.
76. Jiang D, Tang C, Zhang A. (2004). Cluster analysis for gene expression data: A survey. *IEEE Trans Knowledge Data Engineering*. Vol. 2004, No.16, pp. 1370–1386.
77. Jonyer, I., Cook, D., Holder, L. (2001). Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, Vol. 2, pp. 19-43.
78. Kadam, N. V., & Pund, M. A. (2013). Joint approach for outlier detection. *International Journal of Computer Science Application*, Vol. 6, No. 2, pp. 445–448.
79. Karyptis, G., Han, E. H., and Kumar, V. (1999). CHAMELEON: Hierarchical clustering using dynamic modeling. *Computer*, Vol. 32, No.8, pp. 68-75.
80. Kaufman, L., and Rousseeuw, P. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons.
81. Khan, S.S., Ahmad, A. (2004). Cluster center initialization algorithm for K-Means clustering. *Pattern Recognition Letters*, Vol. 25, pp. 1293-1302.
82. Kriegel, H., Pfeifle, M. (2005) Density-based clustering of uncertain data. *Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining*, pp 672–677
83. Kriegel, H., Kroger, P., Sander, J., Zimek, A. (2011) Density-based clustering. *Wiley Interdisciplinary Review*. Vol. 1 pp. 231–240
84. Krishna, K., Murty, M. N. (1999). Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, part B: Cybernetics*, Vol. 29, No. 3, pp. 433-439.
85. Krishnapuram, R., and Keller, J. M. (1996). The possibilistic C-means algorithm: Insights and recommendations. *IEEE Transactions on Fuzzy Systems*, Vol. 4, No. 3, pp. 385-393.
86. Krug, D., Arick, J., & Almond, P. (2008). *Autism screening instrument for educational planning* (3rd Edition). ProEd. Austin, TX.
87. Lam, D., & Wunsch, D. (2014). *Clustering*. Academic Press Library in Signal Processing, Signal Processing Theory and Machine Learning, (1).
88. Levy, S., Duda, M., Haber, N., & Wall, D. (2017). Sparsifying machine learning models identify stable subsets of predictive features for behavioural detection of autism. *Molecular Autism*, Vol. 8, No. 65, PMC5735531. doi: 10.1186/s13229-017-0180-6

89. Lichman, M. (2013). UCI machine learning repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: University of California, School of Information and Computer Science
90. Liu, H., Li, J., Wu, Y., & Fu, Y. (2018). Clustering with outlier removal. Proceedings of ACM Sig on Knowledge Discovery and Data Mining (KDD). ACM, New York, NY, USA.
91. Lord, C., & Jones, R. M. (2012). Annual research review: Re-thinking the classification of autism spectrum disorders. *Journal of Child Psychology and Psychiatry*, Vol. 53, No. 5, pp. 490–509
92. Lord, C., Risi, S., Lambrecht, L., Cook, E. H. Jr, Lambrecht, B. L, DiLavore, P. C, Pickles, A., ... & Rutter, M. (2000). The autism diagnostic observation schedule-generic: A standard measure of social and communication deficits associated with the spectrum of autism. *Journal of Autism Development Disorder*, Vol. 30, No. 30, pp. 205–223
93. Lord, C., Rutter, M., Le Couteur, A. (1994). Autism diagnostic interview-revised: A revised version of a diagnostic interview for caregivers of individuals with possible pervasive developmental disorders. *Journal of Autism Development Disorder*, Vol. 24, No. 24. Pp. 659–685
94. MacQueen, J. (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, pp. 281-297, Berkeley, USA.
95. Maenner MJ, Yeargin-Allsopp M, Van Naarden Braun K, Christensen DL, Schieve LA (2016) Development of a Machine Learning Algorithm for the Surveillance of Autism Spectrum Disorder. *PLoS ONE* Vol. 11(12).
96. Manning, C.D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*, volume 1. Cambridge University Press, Cambridge, UK.
97. Meila, M., Heckerman, D. (2001). An experimental comparison of model-based clustering methods. *Machine Learning*, Vol. 42 pp. 9–29
98. Milligan, G. W. (1981). A Monte Carlo study of thirty internal criterion measures for cluster analysis. *Psychometrika*, Vol. 46, No. 2, pp. 187-199.
99. Mirkin, B. G. (2005). *Clustering for Data mining: A Data recovery approach*. CRC Press. Boca Raton, FL.
100. Mythili, M., & Shanavas, M. R. (2014) A study on Autism spectrum disorders using classification techniques. *International Journal of Soft Computing and Engineering*, Vol. 5, No. 6, pp. 7288–91.
101. Murtagh, F. (1984). "A survey of recent advances in hierarchical clustering algorithms which use cluster centers," *Computer Journal*, Vol. 26, No. 4, pp. 354-359.
102. Nagpal, A., Jatain, A., Gaur, D. (2013). Review based on Data Clustering Algorithms. *IEEE Conference on Information and Communication Technologies*.

103. Nguyen, D. T., Chen, L. and Chan, C. K. 2012. "Clustering with Multi-Viewpoint-Based Similarity Measure," IEEE Transactions on Knowledge and Data Engineering, Vol. 24, No. 6, pp. 988-1001.
104. Pal, K., Keller, J. M. and Bezdek, J. C. (2005). A possibilistic fuzzy C-means Clustering Algorithm. IEEE transactions of Fuzzy Systems, Vol. 13, No. 4, pp. 517-530.
105. Pancer K., & Derkacz, A. (2015). Consistency-based pre-processing for classification of data coming from evaluation sheets of subjects with ASDS. Federated Conference on Computer Science and Information Systems, pp. 63–67.
106. Pedrycz, W. (2002). Collaborative fuzzy clustering. Pattern Recognition Letters, Vol. 23, No. 14, pp. 1675–1686.
107. Pelleg, D., Moore, A. (2000). X-means: Extending k-means with efficient estimation of the number of clusters. In proceedings of the seventeenth international Conference on Machine Learning, pp. 727-734, San Francisco, CA, U.S.A.
108. Pérez-Suárez et. al. (2013). OClustR: A new graph-based algorithm for overlapping clustering. Journal on Advances in Artificial Neural Networks and Machine Learning, Vol. 121, pp. 234-247.
109. Pratap, A, Kanimozhiselvi, C. S., Vijayakumar, R., & Pramod, K. V. (2014). Predictive assessment of autism using unsupervised machine learning models. International Journal of Advance Intelligence Paradigms, Vol. 6, No. 2, pp. 113–21.
110. Pratap, A., & Kanimozhiselvi, C. S. (2012). Application of naive Bayes dichotomizer supported with expected risk and discriminant functions in clinical decisions - Case study. Fourth International Conference of Advanced Computing (ICoAC). Pp. 1-4. IEEE.
111. Quinlan, J. (1986). Induction of decision trees. Machine Learning, Vol. 1, No. 1, pp. 81-106.
112. Quinlan, J. (1994). C4.5: Programs for machine learning. Morgan Kaufmann Publishers.
113. Ramaswamy, S., Rastogi, R., & Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. In SIGMOD.
114. Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association, Vol. 66, No. 336, pp. 846– 850.
115. Raymond T. N and Han, J. (2002). CLARANS: A method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No.5, pp. 1003-1016.
116. Rodriguez, M., Comin, C., Casanova, D., et al. (2019). Clustering Algorithms: A comparative approach. PLoS One. Vol. 14, No. 1, pp 1-34.
117. Rokach, L. (2005). Clustering Methods. Data Mining and Knowledge Discovery Handbook, pp 331-352, Springer.

118. Roux, M. (2015). A comparative study of divisive hierarchical clustering algorithms. Cornell University, NY, U.S.A.
119. Saxena, A., Prasad, M., ... Gupta, A. (2017). A review of clustering techniques and developments. *International Journal of Neurocomputing*. Pp 664-681.
120. Saxena, A., Wang, J. (2010). Dimensionality Reduction with Unsupervised Feature Selection and Applying Non-Euclidean Norms for Classification Accuracy. *International Journal of Data Warehousing and Mining*, Vol. 6, No. 2, pp 22-40
121. Scholkopf, B., Smola, A., Muller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, Vol. 10, No. 5, pp. 1299-1319.
122. Selim, S. Z. and Ismail, M. A. (1984). K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.6, No. 1, pp. 81-87.
123. Sheng, W. Liu, X. (2006). A genetic k-medoids clustering algorithm. *Journal of Heuristics*, Vol. 12, pp. 447-466.
124. Shopler, E., Reichler R., & DeVellis, R. (2010). *Childhood Autism Rating Scale™*, Second Edition. WPS.
125. Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. In *KDD workshop on text mining*, vol. 400, pp. 525-526. Boston, USA.
126. Stewart, L. A., & Lee, L. C. (2017). Screening for autism spectrum disorder in low- and middle-income countries: A systematic review. *Autism: The International Journal of Research and Practice*, Vol. 21, pp. 527–539.
127. Strehl, A., Ghosh, J., and Mooney, R. (2000). Impact of similarity measures on web-page clustering. *Workshop on Artificial Intelligence for Web Search*, pp 58–64.
128. Stutz, J. and Cheeseman, P. (1995). Bayesian classification (AutoClass): Theory and Results. *Advances in Knowledge Discovery and Data Mining*, Menlo Park, CA, AAAI Press. Pp. 153-180.
129. Stutz, J. and Cheeseman, P. (1995). Bayesian classification (AutoClass): Theory and Results. *Advances in Knowledge Discovery and Data Mining*, Menlo Park, CA, AAAI Press. Pp. 153-180.
130. Thabtah, F. (2017a). Autism spectrum disorder screening: Machine learning adaptation and DSM-5 fulfilment. *International Conference on Medical and Health Informatics*. Taichin City, Taiwan.
131. Thabtah, F. (2017b). ASDTest: A mobile app for ASD Screening. [www.asdtests.com](http://www.asdtests.com)
132. Thabtah, F. (2018a). Machine learning in autistic spectrum disorder behaviour research: A review and ways forward. *Informatics for Health and Social Care*. DOI: 10.1080/17538157.2017.1399132

133. Thabtah, F. (2018b). An accessible and efficient autism screening method for behavioural data and predictive analyses. *Health informatics journal*, pp. 1-23. 1460458218796636.  
<https://doi.org/10.1177/1460458218796636>
134. Thabtah F., Hadi W., Abdelhamid N., Issa A. (2011) Prediction Phase in Associative Classification. *Journal of Knowledge Engineering and Software Engineering*. Vol. 21, No. 6(2011) pp. 855-876. WorldScinet.
135. Thabtah F., Mahmood Q., McCluskey L., Abdeljaber H (2010). A new Classification based on Association Algorithm. *Journal of Information and Knowledge Management*, Vol 9, No. 1, pp. 55-64. World Scientific.
136. Thabtah, F., Kamalov, F., & Rajab, K. (2018). A new computational intelligence approach to detect autistic features for autism screening. *International Journal of Medical Informatics*,  
<https://doi.org/10.1016/j.ijmedinf.2018.06.009>
137. Thabtah, F., Peebles, D. A new machine learning model based on induction of rules for autism detection. *Health Informatics Journal*, DOI:1460458218824711.
138. Thabtah F., Zhang L., Abdelhamid N. (2019) NBA Game Result Prediction using Feature Analysis and Machine Learning. *Annals of Data Science*, pp. 1-11. DOI: 10.1007/s40745-018-00189-x
139. Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of data clusters via the gap statistic. *Journal of the Royal Statistical Society B*, Vol. 63, pp. 411–423
140. Towle, P., and Patrick, P. (2016). *Autism Spectrum Disorder Screening Instruments for Very Young Children: A Systematic Review*. New York: Hindawi Publishing Corporation.
141. Town P., and Thabtah F. (2019). Data Analytics Tools: A User Perspective. *Journal of Information and Knowledge Management*. 1950002, World Scientific.
142. Trohidis, K., Tsoumakas, G., Kalliris, G., & Vlahavas, I. (2008). Multilabel classification of music into emotions. *Proceeding of the 2008 International Conference on Music Information Retrieval (ISMIR 2008)*, pp. 325-330, Philadelphia, PA, USA.
143. Tsoumakas, G., Katakis, I., & Vlahavas, I. (2010). Mining multi-label data, *data mining and knowledge discovery handbook*, O. Maimon, L. Rokach (Ed.), Springer, 2nd ed.
144. Velmurugan, T. and Santhanam, T. (2010). Computational complexity between K-means and K-medoids clustering algorithms for normal and uniform distributions of data points. *Journal of Computer Science*, Vol. 6, No. 3, pp. 363-368.
145. Velmurugan, T., Santhanam, T. (2011). A survey of partition based clustering algorithms in data mining: an experimental approach. *Information Technology Journal*, Vol. 10, No. 3, pp.478–484



146. Ventola, P., Kleinman, J., Pandey, J., Barton, M., Allen, S., Green, J., Fein, D. (2006). Agreement among four diagnostic instruments for autism spectrum disorders in toddlers. *Journal of Autism and Developmental Disorders*, pp. 839-47.
147. Vijaya , A., Bateja, R. (2017). A Review on Hierarchical Clustering Algorithms. *Journal of Engineering and Applied Sciences*, Vol. 12, pp. 7501-7507.
148. Wagner,R., Scholz, S. W., Decker, R. (2005). The number of clusters in market segmentation. *Data Analysis and Decision Support*. Springer-Heidelberg, pp. 157-176. doi:10.1007/3-540-28397-8\_19
149. Wall, D. P., Kosmiski, J., Deluca, T. F., Harstad, L., & Fusaro, V. A. (2012). Use of machine learning to shorten observation-based screening and diagnosis of autism. *Translational Psychiatry*, Vol. 2, No. 4, pp. 1-8.
150. Ward, J. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, Vol. 58, No. 301, pp. 236-244.
151. Wiggins, L., Reynolds, A., Rice, C., Moody, E., Bernal, P., Blaskey, L., Rosenberg, S., Lee, L., Levy, S. (2014). Using standardized diagnostic instruments to classify children with autism in the study to explore early development. *Journal of Autism and Developmental Disorders*. Vol. 45, No. 5, pp. 1271-1280.
152. Witten, I. & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. 2nd Ed, Elsevier.
153. Wolfers, T., Buitelaar, J. K., Beckmann, C. F., Franke, B., Marquand, A. F. (2015). From estimating activation locality to predicting disorder: A review of pattern recognition for neuroimaging-based psychiatric diagnostics. *Neuroscience and Bio Behavioural Review*, Vol. 57, pp. 328–349.
154. Xu R, Wunsch D. (2005). Survey of clustering algorithms [Internet]. *IEEE Transactions on Neural Networks*. 2005. pp. 645–678. pmid:15940994
155. Xu, D., Tian, Y. (2015). *A Comprehensive Survey of Clustering Algorithms*. *Annals of Data Science*, Springer. Vol. 2, No. 2, pp. 165-193.
156. Xiong, H., Wu, J., Chen, J. (2009). K-Means Clustering Versus Validation Measures: A Data-Distribution Perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 39, No. 2, pp. 318–331.
157. Yu, Q., Luo, Y., Chen, & C., Ding, X. (2016). Outlier-eliminated k-means clustering algorithm based on differential privacy preservation. *Applied Intelligence*, Vol. 45, No. 4. Pp. 1179–1191.
158. Zhang, J. S., & Leung, Y. (2003). Robust clustering by pruning outliers. *IEEE Trans. on Systems, Man, and Cybernetics – Part B* Vol. 33, No. 6, pp. 983–999.

159. Zhang, T., Ramakrishnan, R., and Lin, Y. (1996). BIRCH: An efficient method for very large Databases. In ACM SIGMOD Conference, pp. 103-114.
160. Zhang, T., Ramakrishnan, R., and Lin, Y. (1997). BIRCH: A new data clustering algorithm and its applications. Journal of Data Mining and Knowledge Discovery, Vol 1, No. 2, pg. 141-182.

# Appendix A

## Extending OMCOKE in WEKA

### 1. Loading and Pre-processing data in WEKA

Several standard loaders can be used:

- Files that such as WEKA's ARFF file format, C4.5 Files, text files, comma separated formats in spreadsheet (CSV), JSON files etc.
- Using the Uniform Resource Locator (URL) functionality to load data that is saved in a specific server.
- Databases connected externally using ODBC to any commercial or open source databases such as Oracle, SQL Server, or MySQL.

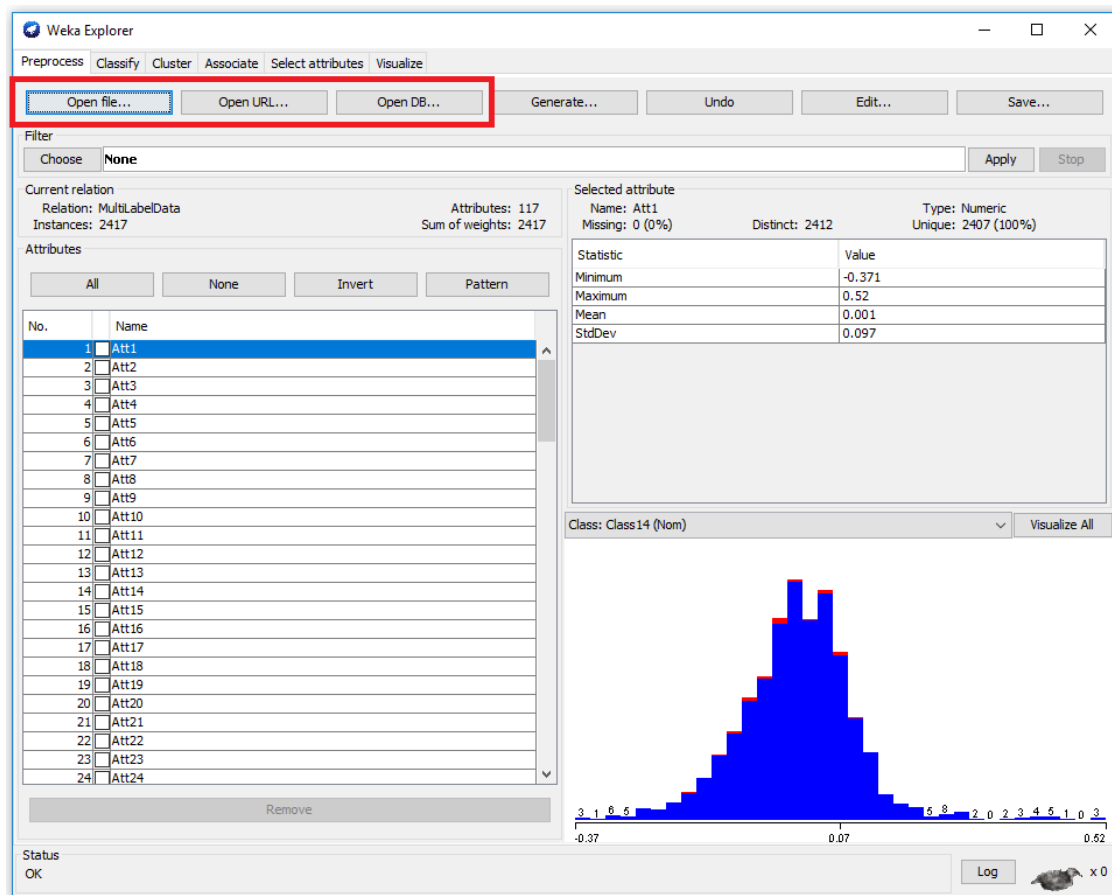


Fig. 1 Loading and Pre-processing data in WEKA

Once loaded, the data is pre-processed in WEKA to show the relationship that exists in the data, the number of instances, attributes present, type of data, and provides basic descriptive statistics on the data including a visual graph.

## 2. Clustering Methods

Several ML clustering methods are available in WEKA to allow users to select as seen on Figure 2 below. OMCOKE highlighted in the option list.

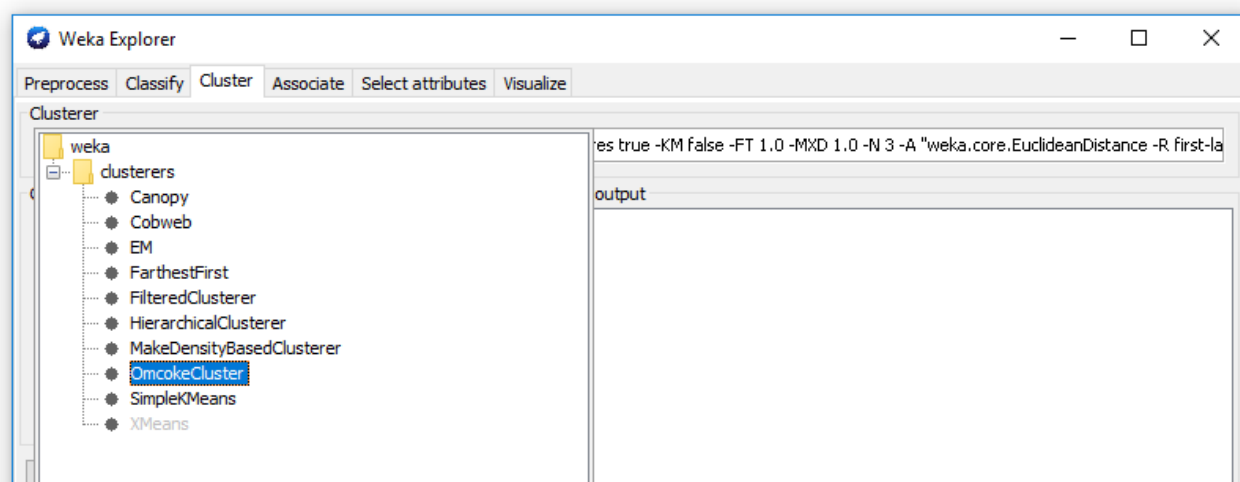


Fig. 2 Clustering Methods in WEKA

## 3. Standard Outputs of the Clustering Methods

Weka provides basic outputs of the learning methods used. Figure 3 shows an example of the output obtained using the K-means method in the Iris dataset. The following are the basic information displayed on the Output panel:

- 1) The learning method used
- 2) The number of instances on the dataset
- 3) The number of attributes considered during the learning process on the dataset
- 4) Number of iterations
- 5) The optimal value achieved in the objective criterion
- 6) Initial cluster assignments
- 7) The final cluster representatives i.e. cluster centroids
- 8) Time taken to build the model

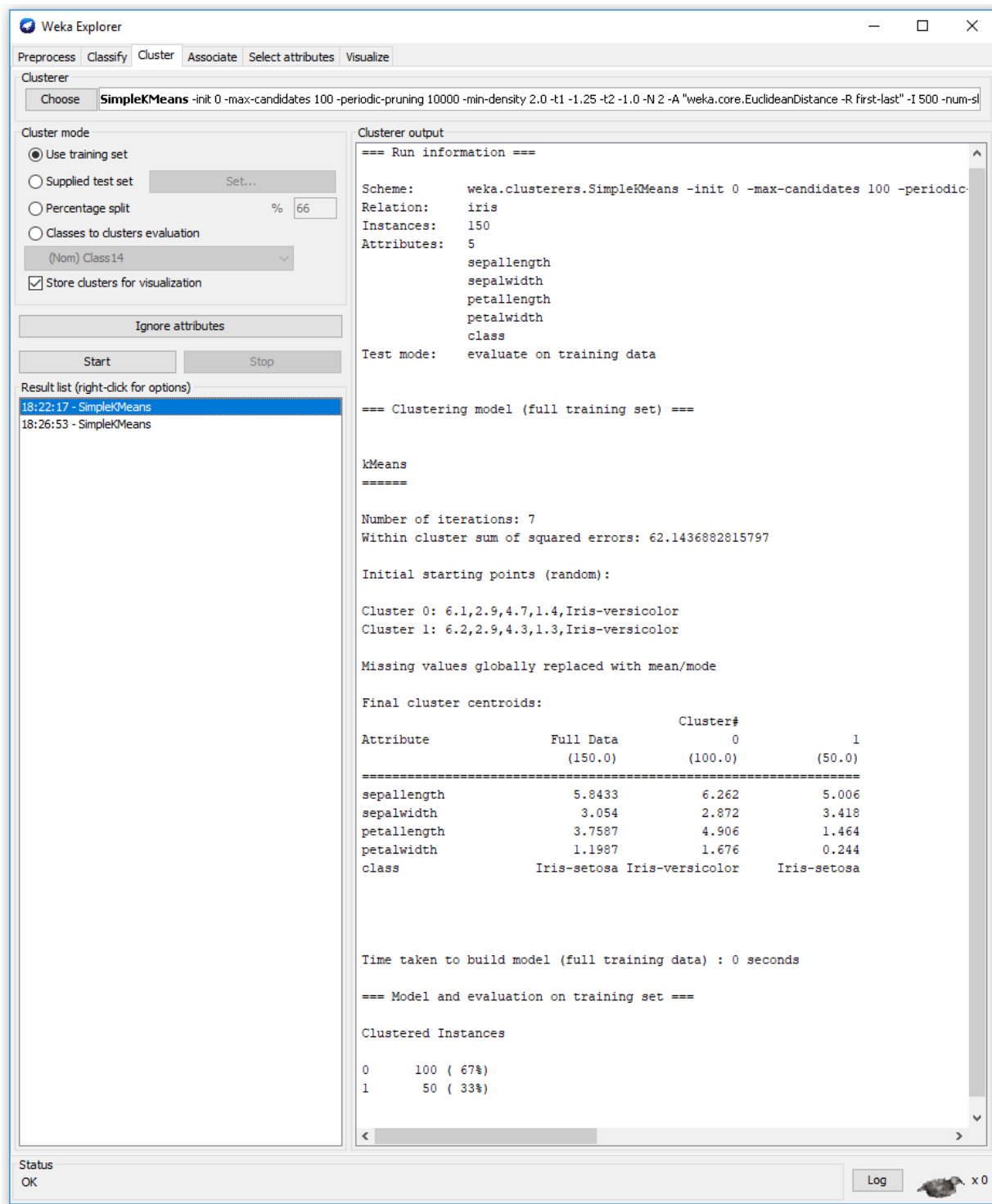
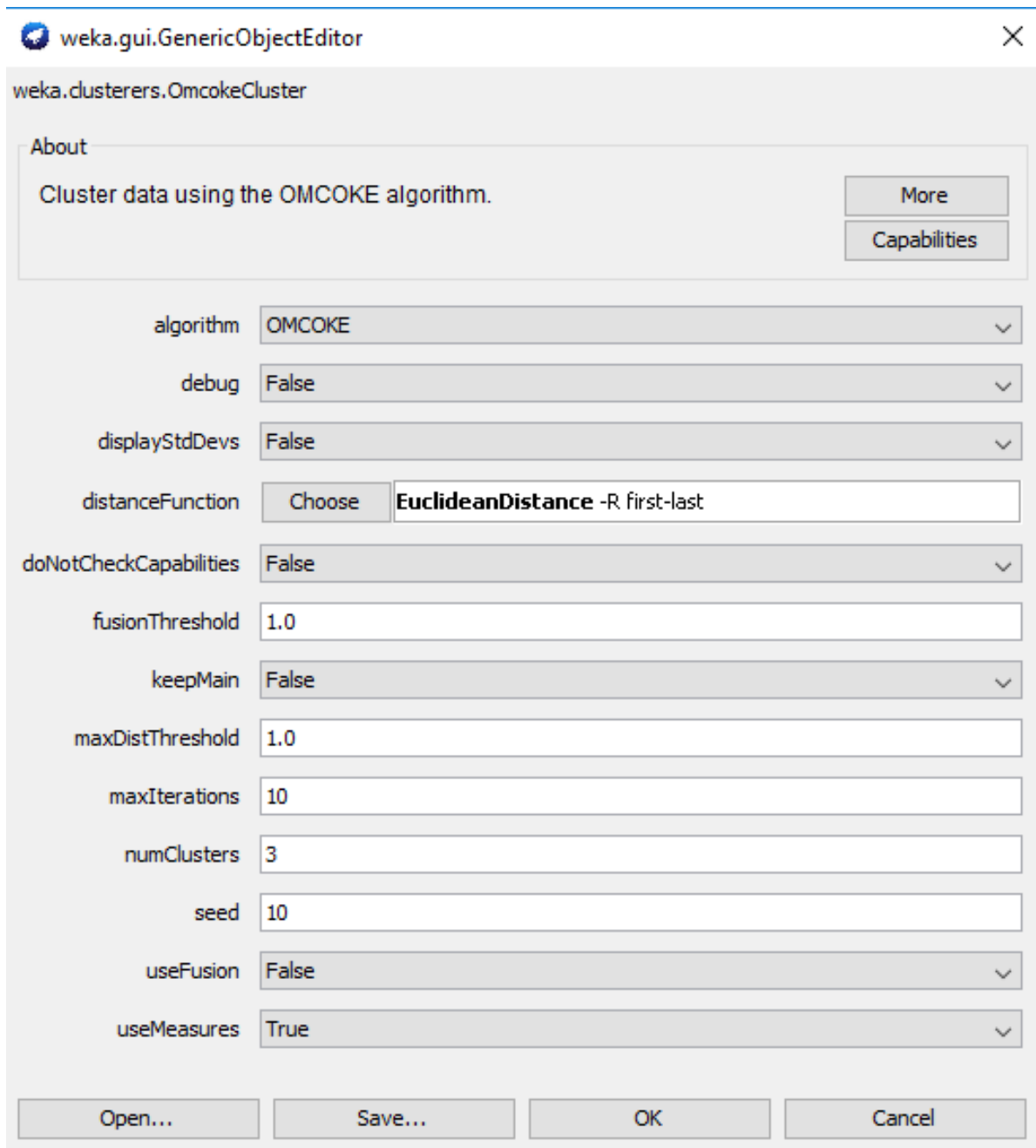


Fig. 3. Example of Cluster output obtained using K-means on Iris dataset

#### 4. OMCOKE Parameter setting

Default values are set on the dialog box and are displayed in Figure 4 below. The useMeasure setting must be set to "True" which is the default setting for the algorithm to display the display measures. OMCOKE has been implemented only to use the Euclidean distance.



The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.clusterers.OmcokeCluster' class. The 'About' tab is selected, displaying the text 'Cluster data using the OMCOKE algorithm.' with 'More' and 'Capabilities' buttons. Below this, various parameters are listed with their current values:

Parameter	Value
algorithm	OMCOKE
debug	False
displayStdDevs	False
distanceFunction	Choose EuclideanDistance -R first-last
doNotCheckCapabilities	False
fusionThreshold	1.0
keepMain	False
maxDistThreshold	1.0
maxIterations	10
numClusters	3
seed	10
useFusion	False
useMeasures	True

At the bottom, there are four buttons: 'Open...', 'Save...', 'OK', and 'Cancel'.

Fig. 4 OMCOKE Parameter Settings in WEKA

The maxDistThreshold is defaulted to 1.0. Altering this will activate the Outlier detection feature in the algorithm which will then create an outlier-cluster on the fly if any outliers are identified in the dataset. The useFusion is defaulted to False as this has not been implemented in the algorithm.

### ***5. Evaluation of the resulting cluster in OMCOKE***

In addition to the standard basic output displayed, when the useMeasure feature of OMCOKE is set to True, the following is also displayed;

- 1) Number of outliers detected in the dataset
- 2) Outlier index of all the data points identified as outliers
- 3) Confusion matrix showing the True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN)
- 4) Pair-based Precision-Recall measures

Figures 5 & 6 below highlights the results of OMCOKE run on the Iris dataset. The first run has the maxdistThreshold set to default, whereas in Figure 6, the value of that is adjusted to 0.95.

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Clusterer

Choose **OmcokerCluster** -algorithm OMCoke -fusion false -measures true -KM false -FT 1.0 -MXD 1.0 -N 3 -A "weka.core.EuclideanDistance -R first-last" -I 10 -S 10

Cluster mode

☒ Use training set

☐ Supplied test set Set...

☐ Percentage split % 66

☐ Classes to clusters evaluation (Nom) class

☒ Store clusters for visualization

Ignore attributes

Start Stop

Result list (right-click for options)

18:57:06 - OmcokerCluster

Clusterer output

```

=== Run information ===

Scheme:      weka.clusterers.OmcokerCluster -algorithm OMCoke -fusion false -measures true
Relation:    iris
Instances:   150
Attributes:  5
              sepalwidth
              sepalwidth
              petalwidth
              petalwidth
              class
Test mode:   evaluate on training data

=== Clustering model (full training set) ===

OMCoke
=====

Number of iterations: 5
Within cluster sum of squared errors: 71.27143164382723

Initial starting points ( random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 4.8,3.1,1.6,0.2,Iris-setosa
Cluster 2: 5,2.3,3.3,1,Iris-versicolor

Final cluster centroids:

Attribute      Full Data      Cluster#
              (150.0)      0          1          2
              (150.0)      (75.0)      (50.0)      (25.0)
=====
sepalwidth     5.8433         6.262       5.5725      5.7157
sepalwidth     3.054         2.872       3.0425      3.0055
petalwidth     3.7587         4.906       3.235       3.5945
petalwidth     1.1987         1.676       0.955       1.1236
class          Iris-setosa Iris-versicolor Iris-setosa Iris-versicolor

===== Measures =====

Number of centroids = 3 centroids

All Points = 150 points
Number of Outliers = 0 points
Outlier indexes = []
TP = 7779
FP = 12312
TN = 26910
FN = 13030
Recall = 0.3738
Precision = 0.3872
F-Measure = 0.3804
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
TP = 7779
FP = 12312
TN = 26910
FN = 13030
Recall = 0.3738
Precision = 0.3872
F-Measure = 0.3804

Time taken to build model (full training data) : 0.08 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      75 ( 50%)
1      50 ( 33%)
2      25 ( 17%)

```

Status OK

Log x 0

Fig. 5 OMCoke evaluation metrics with maxdistThreshold default value of 1



```

OMCOKE
=====

Number of iterations: 9
Within cluster sum of squared errors: 62.20822172715

Initial starting points ( random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 4.8,3.1,1.6,0.2,Iris-setosa
Cluster 2: 5,2.3,3.3,1,Iris-versicolor

Final cluster centroids:

Attribute          Full Data          Cluster#
                   (146.0)          (96.0)
=====
sepallength        5.8433             6.2052
sepalwidth         3.054              2.8479
petallength        3.7587             4.8385
petalwidth         1.1987             1.6521
class              Iris-setosa Iris-versicolor

===== Measures =====

Number of centroids = 3 centroids

All Points = 146 points
Number of Outliers = 4 points
Outlier indexes = [109, 117, 118, 131]
TP = 5745
FP = 4650
TN = 14292
FN = 4716
Recall = 0.5492
Precision = 0.5527
F-Measure = 0.5509

```

Fig. 6 OMCOKE results on Iris dataset with the threshold set to 0.95.

4 outliers are identified in the dataset with this setting and are marked in their corresponding index in the dataset.

## 6. Visualization of Data

WEKA has a visualization tool that can be used to discover the data.

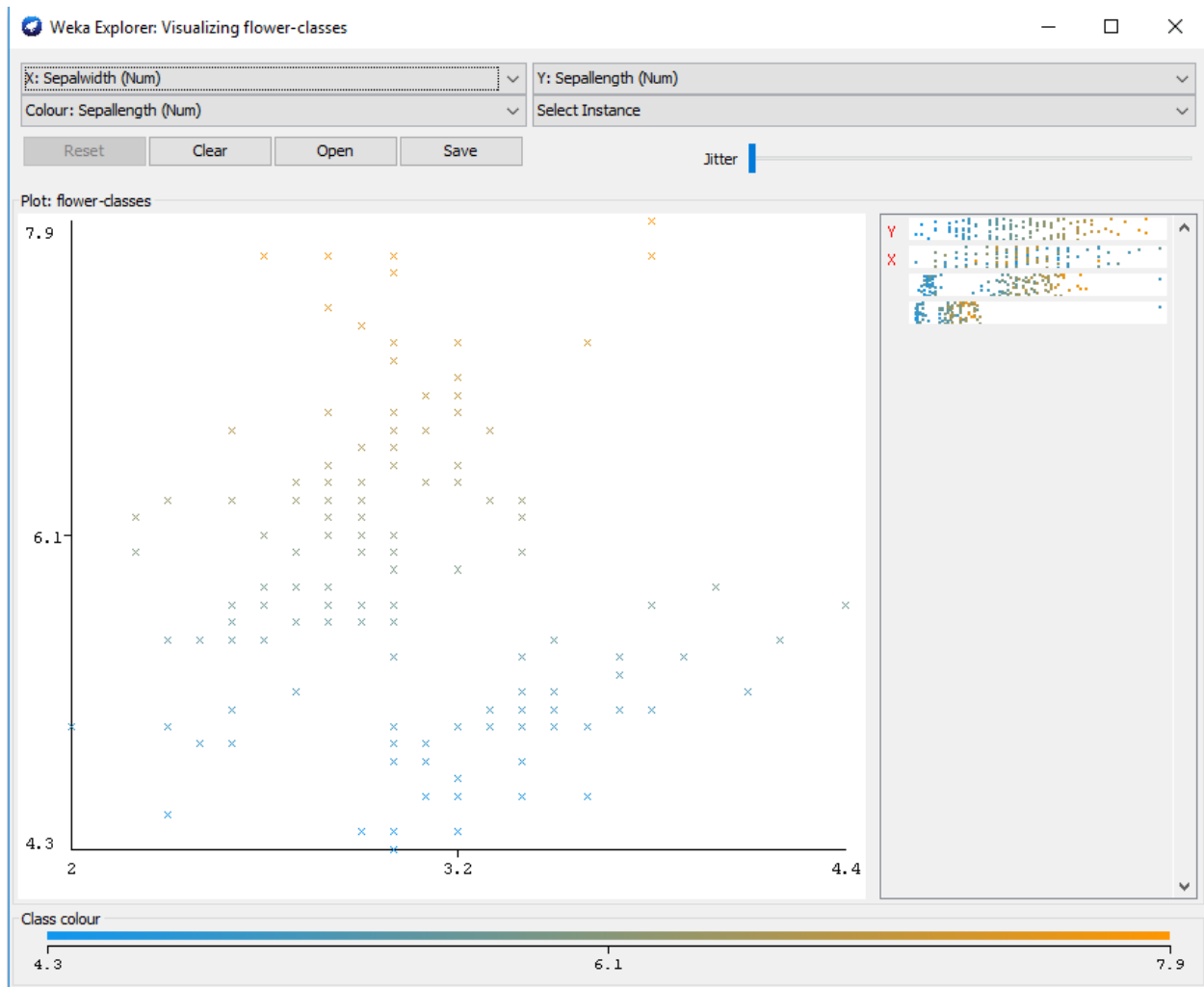


Fig. 7 Visualization of data in WEKA

## Appendix B

### ***OMCOKE SOURCE CODE***

Title: Clusterer

Description: OMCOKE Project by Said Baadel

Copyright: Copyright (c) 2018

```
package weka.clusterers;
import weka.classifiers.rules.DecisionTableHashKey;
import weka.core.*;
import weka.core.Capabilities.Capability;
import weka.core.TechnicalInformation.Field;
import weka.core.TechnicalInformation.Type;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

enum ALGORITHM {
    KMEAN, OMCOKE;
    public static SelectedTag selectedTag(String value) {
        return new SelectedTag(value, toTags());
    }
    public static Tag[] toTags() {
        ALGORITHM[] formulas = values();
        Tag[] result = new Tag[formulas.length];
        for (int i = 0; i < result.length; i++) {
            result[i] = new Tag(i, formulas[i].name(), formulas[i].name());
        }
        return result;
    }
}

public class OmcokeCluster extends RandomizableClusterer implements
    NumberOfClustersRequestable, WeightedInstancesHandler,
    TechnicalInformationHandler {
```

```

private static Logger log = Logger.getLogger(OmcokeCluster.class.getName());

static {
    log.setLevel(Level.ALL);
}

/** new Options
protected String algorithm = ALGORITHM.OMCOKE.name(); //with measures
protected boolean useFusion = false;
protected boolean useMeasures = true;
protected boolean keepMain = false; //exclude the minimum distance from the threshold
/** FT default = 1, range in [0-1] , 0: full fusion, 1: no fusion
protected double fusionThreshold = 1.0;
/** MXD maxdist variable, if 1 same as largest distance of an object that was assigned to a
cluster.
protected double maxDistThreshold = 1.0;
volatile protected String okmStringResults = "";
public SelectedTag getAlgorithm() {
    return ALGORITHM.selectedTag(algorithm);
}
public void setAlgorithm(SelectedTag tv) {
    this.algorithm = tv.getSelectedTag().toString();
}
public String algorithmTipText() {
    return "TODO: Algorithm tip text";
}
public boolean getUseFusion() {
    return useFusion; }
public void setUseFusion(boolean useFusion) {
    this.useFusion = useFusion;
}
public String useFusionTipText() {
    return "TODO: use fusion tip text";
}
public boolean getUseMeasures() {
    return useMeasures;

```

```

}
public void setUseMeasures(boolean useMeasures) {
    this.useMeasures = useMeasures;
}
public String useMeasuresTipText() {
    return "use measures";
}
public double getFusionThreshold() {
    return fusionThreshold;
}
public void setFusionThreshold(double fusionThreshold) {
    this.fusionThreshold = fusionThreshold;
}
public String fusionThresholdTipText() {
    return "TODO fusion threshold";
}
public double getMaxDistThreshold() {
    return maxDistThreshold;
}
public void setMaxDistThreshold(double maxDistThreshold) {
    this.maxDistThreshold = maxDistThreshold;
}
public String maxDistThresholdTipText() {
    return "maxdist threshold";
}
public void setKeepMain(boolean keepMain) {
    this.keepMain = keepMain;
}
public boolean getKeepMain() {
    return keepMain;
}
/**
 * number of clusters to generate.
 */
protected int m_NumClusters = 3;

```

```

/**
 * Holds the initial start points, as supplied by the initialization method
 * used
 */
protected Instances m_initialStartPoints;
/**
 * holds the cluster centroids.
 */
protected Instances m_ClusterCentroids;

/** holds the cluster points
// protected Instances[] m_ClusterPoints;

/** holds the overlapped cluster points
// Instances[] oClusterPoints = null;

/**
 * Holds the standard deviations of the numeric attributes in each cluster.
 */
protected Instances m_ClusterStdDevs;
/**
 * For each cluster, holds the frequency counts for the values of each nominal
 * attribute.
 */
protected double[][][] m_ClusterNominalCounts;
protected double[][] m_ClusterMissingCounts;
/**
 * Stats on the full data set for comparison purposes. In case the attribute
 * is numeric the value is the mean if is being used the Euclidian distance or
 * the median if Manhattan distance and if the attribute is nominal then it's
 * mode is saved.
 */
protected double[] m_FullMeansOrMediansOrModes;
protected double[] m_FullStdDevs;
protected double[][] m_FullNominalCounts;
protected double[] m_FullMissingCounts;

```

```

/**
 * Display standard deviations for numeric attributes.
 */
protected boolean m_displayStdDevs;

/**
 * The number of instances in each cluster.
 */
protected double[] m_ClusterSizes;

/**
 * Maximum number of iterations to be executed.
 */
protected int m_MaxIterations = 100;

/**
 * Keep track of the number of iterations completed before convergence.
 */
protected int m_Iterations = 0;

/**
 * Holds the squared errors for all clusters.
 */
protected double[] m_squaredErrors;

/**
 * the distance function used.
 */
protected DistanceFunction m_DistanceFunction = new EuclideanDistance();

/**
 * Assignments obtained.
 */
protected int[] m_Assignments = null;

/**
 * the default constructor.
 */
public OmcokeCluster() {

```

```

    super();
    m_SeedDefault = 10;
    setSeed(m_SeedDefault);
}

@Override
public TechnicalInformation getTechnicalInformation() {
    TechnicalInformation result;
    result = new TechnicalInformation(Type.INPROCEEDINGS);
    result.setValue(Field.AUTHOR, "Said");
    result.setValue(Field.TITLE, "OMCOKE");
    result.setValue(Field.BOOKTITLE, "");
    result.setValue(Field.YEAR, "2017");
    result.setValue(Field.PAGES, "***-***");
    return result;
}

/**
 * Returns a string describing this clusterer.
 *
 * @return a description of the evaluator suitable for displaying in the
 * explorer/experimenter gui
 */
public String globalInfo() {
    return "Cluster data using the OMCOKE algorithm. Can use either "
        + "the Euclidean distance (default) or the Manhattan distance."
        + " If the Manhattan distance is used, then centroids are computed "
        + "as the component-wise median rather than mean."
        + " For more information see:\n\n" + getTechnicalInformation().toString();
}

/**
 * Returns default capabilities of the clusterer.
 *
 * @return the capabilities of this clusterer
 */
@Override
public Capabilities getCapabilities() {

```



```

Capabilities result = super.getCapabilities();
result.disableAll();
result.enable(Capability.NO_CLASS);

// attributes
result.enable(Capability.NOMINAL_ATTRIBUTES);
result.enable(Capability.NUMERIC_ATTRIBUTES);
result.enable(Capability.MISSING_VALUES);
return result;
}

/**
 * Generates a clusterer. Has to initialize all fields of the clusterer that
 * are not being set via options.
 *
 * @param instances set of instances serving as training data
 * @throws Exception if the clusterer has not been generated successfully
 */
@Override
public void buildClusterer(Instances instances) throws Exception {
    Instances data = new Instances(instances);
    okmStringResults = "";
    InitClusters init = new InitClusters().invoke(data);
    Instances[] m_ClusterPoints = null;

    switch (ALGORITHM.valueOf(algorithm)) {
        case KMEAN:
            m_ClusterPoints = buildClustererKMean(init.instances, init.clusterAssignments);
            break;
        case OMCOKE:
            m_ClusterPoints = buildClustererOMCOKE(data);
            break;
    }
    calcStats(data, m_ClusterPoints);
    log.info("\n" + m_ClusterCentroids.stream()
        .map(i -> i.toString())

```

```

        .collect(Collectors.joining("\n")));
    }

    private Instances[] buildClustererMCOKE(Instances instances) {
        return null;
    }

    /**
     * Map all assignments to its clusters, allow overlapping
     *
     * @param assignments
     * @return Map of entries : e.key = centroidIndex, e.value = List of indexes of instances
     mapped to this cluster:
     */
    public static Map<Integer, List<Integer>> toClusters(List<BitSet> assignments) {
        HashMap<Integer, List<Integer>> result = new HashMap<>(6);
        for (int line = 0; line < assignments.size(); line++) {
            final int ln = line;
            final BitSet bs = assignments.get(line);
            bs.stream()
                .forEach(clusterIndex -> {
                    List<Integer> list = result.get(clusterIndex);
                    if (list == null) {
                        list = new ArrayList<>();
                        result.put(clusterIndex, list);
                    }
                    list.add(ln);
                });
        }
        return result;
    }

    /**
     * @param data
     * @throws Exception
     */
    private Instances[] buildClustererOMCOKE(Instances data) {

```

```

Instances[] m_ClusterPoints = null;
boolean converged = false;

/* overlapped cluster points */
List<Integer> outlier = null;
int[] bestAssignments = null;

List<BitSet> assignments = new ArrayList<>();

// make sure not to pick up similar points
final Instances centroids = BUtils.sample(data, m_NumClusters, m_Seed);

int iteration = 0;

while (!converged) {
    iteration++;
    // distances from each point to centroids
    List<double[]> distances = BUtils.distances(data, centroids, m_DistanceFunction);

    // maximum of "highest" distances among all points
    double maxDist = BUtils.getMaxDist(distances);
    //threshold to cut off the outliers as a ratio of maxDist value
    final double distanceThreshold = maxDist * maxDistThreshold;

    //find overlapping assignments and apply threshold at the same time
    List<BitSet> tempAssignments = distances
        .stream()
        .map(d -> BUtils.bitSetOf(d, distanceThreshold, keepMain))
        .collect(Collectors.toList());

    //converged is false for any changes from last assignments
    converged = assignments.size() == tempAssignments.size()
        && !IntStream.range(0, tempAssignments.size())
            .filter(a -> !assignments.get(a).equals(tempAssignments.get(a)))
            .findAny()
            .isPresent();
}

```

```

log.info(String.format("iteration = %d, converged = %s", iteration, converged));
//outlier consists of all non-assigned points,
// (if keepMain then no outlier was found)
outlier = IntStream.range(0, tempAssignments.size())
    .filter(index -> tempAssignments.get(index).isEmpty())
    .boxed()
    .collect(Collectors.toList());

log.info(String.format("outlier size = %d", outlier.size()));

// update centroids
centroids.clear();
centroids.addAll(
    calculateCentroids(data,
        tempAssignments,
        m_NumClusters,
        m_DistanceFunction));
centroids.setClassIndex(centroids.numAttributes() - 1);
m_ClusterCentroids = centroids;

log.info(String.format("Number of centroids = %d", centroids.numInstances()));
if (iteration == m_MaxIterations) {
    converged = true;
}
// check empty clusters
int emptyClusterCount = m_NumClusters - centroids.size();
log.info(String.format("emptyClusterCount = %d", emptyClusterCount));

// update stats code here

if (!converged) {
    m_ClusterNominalCounts = new double[m_NumClusters][data.numAttributes()][0];
    assignments.clear();
    assignments.addAll(tempAssignments);
} else {

```

```

/** converged */
m_ClusterCentroids = centroids;

// save memory!
m_DistanceFunction.clean();

/** setup global variable
m_Iterations = iteration;

m_ClusterPoints = toClustersInInstances(data, outlier, centroids);
int allPoints = Arrays.stream(m_ClusterPoints)
    .mapToInt(i -> i.numInstances())
    .sum();

if (useMeasures) {
    StringJoiner sj = new StringJoiner("\n");
    sj.add(String.format("\nNumber      of      centroids      =      %d      centroids",
m_ClusterCentroids.numInstances()));
    sj.add(String.format("\nAll Points = %d points", allPoints));
    sj.add(String.format("Number of Outliers = %d points", outlier.size()));
    sj.add(String.format("Outlier indexes = %s", outlier.toString()));

    Map<Integer, List<Integer>> tmpMap = toClusters(tempAssignments);
    PairBasedEvaluation eval = new PairBasedEvaluation(data);
    eval.calc(tmpMap);
    sj.add(eval.getResults());

    sj.add("oooooooooooooooooooooooooooooooooooo");
    OPairBasedEvaluation eval2 = OPairBasedEvaluation.of(data);
    eval2.calc(m_ClusterPoints, allPoints);
    sj.add(eval.getResults());

// save memory!
m_DistanceFunction.clean();

/** setup global variable
m_Iterations = iteration;

```

```

return tempIClusters;
}
// Quickly calculate some relevant statistics
for (int j = 0; j < members.numAttributes(); j++) {
    if (members.attribute(j).isNominal()) {
        nominalDists[j] = new double[members.attribute(j).numValues()];
    }
}
for (Instance inst : members) {
    for (int j = 0; j < members.numAttributes(); j++) {
        if (inst.isMissing(j)) {
            weightMissing[j] += inst.weight();
        } else {
            weightNonMissing[j] += inst.weight();
            if (members.attribute(j).isNumeric()) {
                vals[j] += inst.weight() * inst.value(j); // Will be overwritten in Manhattan case
            } else {
                nominalDists[j][(int) inst.value(j)] += inst.weight();
            }
        }
    }
}
for (int j = 0; j < members.numAttributes(); j++) {
    if (members.attribute(j).isNumeric()) {
        if (weightNonMissing[j] > 0) {
            vals[j] /= weightNonMissing[j];
        } else {
            vals[j] = Utils.missingValue();
        }
    } else {
        double max = -Double.MAX_VALUE;
        double maxIndex = -1;
        for (int i = 0; i < nominalDists[j].length; i++) {
            if (nominalDists[j][i] > max) {
                max = nominalDists[j][i];
                maxIndex = i;
            }
        }
    }
}

```

```

    }
    if (max < weightMissing[j]) {
        vals[j] = Utils.missingValue();
    } else {
        vals[j] = maxIndex;
    }
}
}
}

if (updateClusterInfo) {
    for (int j = 0; j < members.numAttributes(); j++) {
        m_ClusterMissingCounts[centroidIndex][j] = weightMissing[j];
        m_ClusterNominalCounts[centroidIndex][j] = nominalDists[j];
    }
}

if (addToCentroidInstances) {
    m_ClusterCentroids.add(new DenseInstance(1.0, vals));
}

return vals;
}

/**
 * Returns the number of clusters.
 *
 * @return the number of clusters generated for a training dataset.
 * @throws Exception if number of clusters could not be returned successfully
 */
@Override
public int numberOfClusters() throws Exception {
    return m_NumClusters;
}

/**

```

```

* Returns an enumeration describing the available options.
*
* @return an enumeration of all the available options.
*/
@Override
public Enumeration<Option> listOptions() {
    Vector<Option> result = new Vector<Option>();
//
    result.addElement(new Option("\tAlgorithm to be used.\n" + "\t(default KMEAN).",
        "algorithm", 1, "-algorithm <kmean, OMCOKE>"));
    result.addElement(new Option("\tUse fusion in algorithm.\n" + "\t(default false).",
        "fusion", 0, "-fusion"));
    result.addElement(new Option("\tCalc measures for algorithm.\n" + "\t(default false).",
        "measures", 0, "-measures"));
    result.addElement(new Option("\tExclude minimum distance class from thresholding.\n" +
"\t(default false).",
        "KM", 0, "-KM"));
    result.addElement(new Option("\tFusion threshold.\n" + "\t(default 1, no fusion).",
        "FT", 1, "-FT <num>"));
    result.addElement(new Option("\tMaxDist threshold.\n" + "\t(default 1).",
        "MXD", 1, "-MXD <num>"));
    result.addElement(new Option("\tNumber of clusters.\n" + "\t(default 2).",
        "N", 1, "-N <num>"));
    result.addElement(new Option("\tDisplay std. deviations for centroids.\n",
        "V", 0, "-V"));
    result.addElement(new Option(
        "\tDon't replace missing values with mean/mode.\n", "M", 0, "-M"));

    result.add(new Option("\tDistance function to use.\n"
        + "\t(default: weka.core.EuclideanDistance)", "A", 1,
        "-A <classname and options>"));

    result.add(new Option("\tMaximum number of iterations.\n", "I", 1,
        "-I <num>"));

    result.addAll(Collections.list(super.listOptions()));
    return result.elements();
}

```



```

}

/**
 * @param options the list of options as an array of strings
 * @throws Exception if an option is not supported
 */
@Override
public void setOptions(String[] options) throws Exception {

    algorithm = Utils.getOption("algorithm", options);
    useFusion = Boolean.parseBoolean(Utils.getOption("fusion", options));
    useMeasures = Boolean.parseBoolean(Utils.getOption("measures", options));
    keepMain = Boolean.parseBoolean(Utils.getOption("KM", options));
    fusionThreshold = Double.parseDouble(Utils.getOption("FT", options));
    maxDistThreshold = Double.parseDouble(Utils.getOption("MXD", options));
    m_displayStdDevs = Utils.getFlag("V", options);

    String optionString = Utils.getOption('N', options);
    if (optionString.length() != 0) {
        setNumClusters(Integer.parseInt(optionString));
    }
    optionString = Utils.getOption("I", options);
    if (optionString.length() != 0) {
        setMaxIterations(Integer.parseInt(optionString));
    }
    String distFunctionClass = Utils.getOption('A', options);
    if (distFunctionClass.length() != 0) {
        String distFunctionClassSpec[] = Utils.splitOptions(distFunctionClass);
        if (distFunctionClassSpec.length == 0) {
            throw new Exception("Invalid DistanceFunction specification string.");
        }
        String className = distFunctionClassSpec[0];
        distFunctionClassSpec[0] = "";

        setDistanceFunction((DistanceFunction) Utils.forName(
            DistanceFunction.class, className, distFunctionClassSpec));
    }
}

```

```

    } else {
        setDistanceFunction(new EuclideanDistance());
    }
    super.setOptions(options);
    Utils.checkForRemainingOptions(options);
}

```

#### Pair-based Precision-Recall Measures

```

package weka.clusterers;
import weka.core.Instance;
import weka.core.Instances;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

/**
 * https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html
 * https://stats.stackexchange.com/questions/15158/precision-and-recall-for-clustering
 */
public class OPairBasedEvaluation {
    final Map<Integer, Integer> labels;
    final int numLabels;

    private int tp, tn, fp, fn;

    /**
     * label attribute should be set before calling the constructor
     *
     * @param data
     */
    public static OPairBasedEvaluation of(Instances data) {
        data.setClassIndex(data.numAttributes() - 1);
        HashMap<Integer, Integer> m = new HashMap<>(data.numInstances());
        IntStream.range(0, data.numInstances())
            .forEach(i -> {
                Instance inst = data.instance(i);

```

```

        m.put(i, (int) inst.classValue());
    });
    int numLabels = (int) m.values()
        .stream()
        .distinct()
        .count();
    return new OPairBasedEvaluation(m, numLabels);
}

public static OPairBasedEvaluation of(Map<Integer, Integer> labels, int numLabels) {
    return new OPairBasedEvaluation(new HashMap<>(labels), numLabels);
}

private OPairBasedEvaluation(Map<Integer, Integer> labels, int numLabels) {
    this.labels = labels;
    this.numLabels = numLabels;
}

public static int twoCombinations(int k) {
    if (k < 3) return 1;
    return k * (k - 1) / 2;
}

public static int countCalcOneFP(int[] freqs) {
    return Combinations.of(2, freqs)
        .stream()
        .mapToInt(i -> i.get(0) * i.get(1))
        .sum();
}

/**
 * return count streamOf labels in data in the same order streamOf attributes
 *
 * @param clusterPoints
 * @param numLabels
 * @return

```

```

*/
public static int[] countLabels(Instances clusterPoints, final int numLabels) {
    //assert classIndex is set before calling
    Map<Integer, Long> counts = clusterPoints.stream()
        .collect(Collectors.groupingBy(i -> (int) i.classValue(), Collectors.counting()));

    return IntStream.range(0, numLabels)
        .map(i -> (counts.get(i)) == null ? 0 : counts.get(i).intValue())
        .toArray();
}

private int[] countLabels(List<Integer> points) {
    //assert classIndex is set before calling
    Map<Integer, Long> counts = points.stream()
        .collect(Collectors.groupingBy(
            i -> labels.get(i),
            Collectors.counting()));

    return IntStream.range(0, numLabels)
        .map(i -> (counts.get(i)) == null ? 0 : counts.get(i).intValue())
        .toArray();
}

public void calc(List<int[]> cLabels) {
    int numPoints = cLabels.stream()
        .flatMapToInt(i -> Arrays.stream(i))
        .sum();
    calc(cLabels, numPoints);
}

public void calc(List<int[]> cLabels, int numPoints) {
    //All pairs
    int allPairs = twoCombinations(numPoints);
    System.out.println("allPairs = " + allPairs);

    /** All positives (TP + FP) */

```

```

int tpFp = cLabels.stream()
    .mapToInt(i -> twoCombinations(IntStream.of(i).sum()))
    .sum();
System.out.println("tpFp = " + tpFp);

/** calculate TP */

tp = cLabels.stream()
    .flatMapToInt(i -> IntStream.of(i)
        .filter(v -> v >= 2)
        .map(OPairBasedEvaluation::twoCombinations))
    .sum();
System.out.println("tp = " + tp);

/** calculate FP */
/*

//method 1 : FP = ALL_TP_FP - TP
fp = tpFp - tp;
System.out.println("fp1 = " + fp);
*/

//method 2
fp = cLabels.stream()
    .mapToInt(i -> countCalcOneFP(i))
    .sum();
System.out.println("fp2 = " + fp);
assert tpFp == tp + fp;

/** All negatives (TN + FN) */
int tnFn = allPairs - tpFp;
System.out.println("tnFn = " + tnFn);

/** calculate TN */
// tn = tnFn - fn;

```

```

//  System.out.println("tn1 = " + tn);
    tn = countTN(cLabels, numLabels);
    System.out.println("tn2 = " + tn);

    /** calculate FN */

    //method 1
    fn = tnFn - tn;
    System.out.println("fn1 = " + fn);

    //method 2

    /**/labelsClustersCounts
    List<int[]> labelFreqs = BUtils.transposeAndFilterZero(cLabels);
    fn = labelFreqs.stream()
        .mapToInt(i -> BUtils.combinProduct(i))
        .sum();
    System.out.println("fn2 = " + fn);
    */

}

/**
 * @param length ex 3
 * @return { [0,1], [0,2], [1,2]}
 */
private static List<int[]> pairs(int length) {
    assert length >= 2;
    if (length == 2) {
        return new ArrayList<>(Arrays.asList(new int[]{0, 1}));
    }
    List<int[]> result = new ArrayList<>(length * length / 2 - length);
    for (int i = 0; i < length - 1; i++) {
        for (int j = i + 1; j < length; j++) {
            result.add(new int[]{i, j});
        }
    }
}

```

```

    }
    return result;
}

public static int countTN(List<int[]> cLabels, int numLabels) {
    List<int[]> pairs = pairs(numLabels);
    List<int[]> communities = pairs(cLabels.size());
    int result = 0;
    for (int[] cPair : communities) {
        int[] a = cLabels.get(cPair[0]);
        int[] b = cLabels.get(cPair[1]);
        for (int[] pair : pairs) {
            result += a[pair[0]] * b[pair[1]];
            result += a[pair[1]] * b[pair[0]];
        }
    }
    return result;
}

/**
 * @param m_ClusterPoints
 * @param allPoints
 */
public void calc(Instances[] m_ClusterPoints, int allPoints) {
    //assume the label is the last attribute
    int numPoints = allPoints > 0 ? allPoints :
        Arrays.stream(m_ClusterPoints)
            .mapToInt(Instances::numInstances)
            .sum();
    //count all labels in each cluster
    List<int[]> clustersLabelsCounts = Arrays.stream(m_ClusterPoints)
        .map(i -> countLabels(i, numLabels))
        .collect(Collectors.toList());

    calc(clustersLabelsCounts, numPoints);
}

```

```

public int getTp() {
    return tp;
}

public int getTn() {
    return tn;
}

public int getFp() {
    return fp;
}

public int getFn() {
    return fn;
}

public double precision() {
    return (double) tp / (tp + fp);
}

public double recall() {
    return (double) tp / (tp + fn);
}

public double fMeasure() {
    double precision = precision();
    double recall = recall();
    return 2 * recall * precision / (recall + precision);
}

public String getResults() {
    StringJoiner s = new StringJoiner("\n");
    s.add(String.format("Recall = %1.4f", recall()));
    s.add(String.format("Precision = %1.4f", precision()));
    s.add(String.format("F-Measure = %1.4f", fMeasure()));
    return s.toString();
}

// 3- Print out the current results
System.out.println("TP = " + eval.getTp());
System.out.println("FP = " + eval.getFp());
System.out.println("TN = " + eval.getTn());

```



```
System.out.println("FN = " + eval.getFn());

System.out.println("recall = " + eval.recall());
System.out.println("precision = " + eval.precision());
System.out.println("f-measure = " + eval.fMeasure()); }

}
```