Christian-Albrechts-Universität zu Kiel

Technische Fakultät

Institut für Informatik

Master Thesis

„What's between ‚em?" - An approach for detecting in-between class instances

Md Abu Noman Majumdar

Betreuer:

Prof. Dr. Peer Kröger

Zweitbetreuer:

Dr. Daniyal Kazempour

Christian-Albrechts-Universität zu Kiel
March 2023

Abstract

In the unsupervised machine learning setting among their most prominent tasks is either the detection of clusters or of outliers. There exist however instances among outliers that seem to bare a semantic of connecting clusters. They are 'in-between' two or more clusters, acting as potential proxies. In this paper, we propose a formulation for detecting an object in between clusters that can carry important information by which we can reveal potential relationships among clusters that are based on the distance of a point from its $k-$ nearest neighbor. First to find an outlier we have implemented a hierarchical clustering method. After that, we defined a threshold to obtain outliers. Following this, we have mined $k-$ nearest cluster by considering the euclidean distance and cosine distance. To be more accurate we have further considered a set of closest objects from each cluster to the outlier so that we can prune clusters that are behind other clusters. The benefits of this approach are two-fold: First, the proposed method incorporates geometric semantics like clusters being 'behind' other clusters, and second, the pruning substantially contributes to saving computation time. We present the results of an extensive experimental study conducted on real-world and synthetic data sets. The results from the real-world molecular data set highlight and reveal several expected and unexpected observations. Additionally, experiments on another real-world use case, namely consumer complaints, show that the proposed method is capable to discover in-between instances based on our proposed model. The more fine-grained control-permitting experiments on synthetic data-sets augment the understanding of how the proposed in-between instance detection method works, and further illustrate its strengths and limitations. The studies are lastly accompanied by investigating the sensitivity of the parameter settings.

Contents

1 Introduction

Unsupervised machine learning tasks like clustering and outlier detection are well-known and accompanied by a rich body of literature like DBSCAN [Schubert, Sander, Ester, Kriegel, and Xu (2017)], K-means [Likas, Vlassis, and Verbeek (2003)], SLINK [Sibson (1973)], LOF [Breunig, Kriegel, Ng, and Sander (2000)], ABOD [Kriegel, Schubert, and Zimek (2008)]. There are however cases in different domains where one is interested to identify objects that are 'enclosed' / 'in-between' several clusters/classes. This is of interest to identify objects that may share properties from different classes, hence embodying a bridge/connector.

Outliers may have relationships to many classes or clusters, making them valuable observations in some cases because they serve as "bridges" or "connectors" between other groups. In this situation, treating the outlier as a particular case and conducting further analysis to comprehend its distinct features and how it relates to the rest of the data may be helpful. It's worth noting that not all outliers will be interesting or relevant to the analysis. Some outliers may simply be errors in the data that should be corrected or removed, while others may represent a completely different population and may not be relevant to the analysis. It's important to carefully evaluate the nature and significance of any outliers that are identified in the data.

Identifying objects that may share properties from different classes and act as "bridges" or "connectors" between different groups can be a challenging task. One approach to finding such objects is to perform a clustering analysis and look for observations that are outliers or do not fit well into any of the clusters. These observations may be more likely to share properties with multiple classes and act as "bridges" between different groups. Another approach is to use a dimensionality reduction technique such as principal component analysis PCA [Maćkiewicz and Ratajczak (1993)]or t-distributed stochastic neighbor embedding (t-SNE) [Van der Maaten and Hinton (2008)] to visualize the data in a lower-dimensional space. This can help identify observations that are "in-between" different groups and may have characteristics that are shared across multiple classes. This however comes at a risk, namely that through the lower-dimensional projection, the relevant information is lost, obfuscating the detection of potential in-between instances. It's worth noting that finding objects that act as "bridges" between different groups can be a challenging and time-consuming task, and may require additional analysis and interpretation to fully understand their characteristics and how they relate to the rest of the data.

This thesis proposes a formalization for in-between instances that relies on distances as well as on angular information between clusters and outliers. Beyond the formalization, an implementation in form of a prototype is provided where its performance is assessed based on synthetic and real-world data sets. Furthermore, potential weaknesses and strengths are investigated including the hyperparameter sensitivity.

2 Related Work

This section is dedicated to several open questions that may arise for the reader while dealing with the in-between instance problem. On the one hand, an overview of the outlier-detection landscape is sketched which we deem to be a 'relative' to the in-between instance detection problem followed by some elaborations on the fuzzy clustering archetype of algorithms. Alongside the listing of these methods, we also highlight the weaknesses/lacking characteristics/properties that ultimately lead/justify the need for the in-between instance problem.

2.1 Outlier Detection

An outlier is a data point that falls significantly outside the range of the majority of the data points in a data-set [Hawkins (1980)]. Outliers can occur for a variety of reasons, such as errors in data collection or measurement, or they may indicate an unusual or unexpected event or phenomenon.

In statistical analysis, outliers can have a significant impact on the results of the analysis and the conclusions that are drawn. For example, they may cause the mean or median (primarily the mean. the median is supposed to be a "robust" measure) of a data-set to be distorted, or they may affect the results of statistical tests such as t-tests or ANOVA. As a consequence, it is often important to identify and analyze outliers carefully to understand their potential impact on the results of the analysis. The overall picture on outlier methods as elaborated on in [Chandola, Banerjee, and Kumar (2007)].

There are several methods for identifying and analyzing outliers in statistical analysis. In the following, we elaborate on the different archetypes of outlier detection techniques. As a remark to the reader, we want to highlight here that some of these archetypes may overlap, and shall not be regarded as entirely disjunct categories.

Classification-based outlier detection: Classification-based outlier detection methods involve training a classifier to distinguish between inliers (normal data points) and outliers. Once the classifier has been trained, it can be used to predict whether a new data point is an outlier or not. Classification-based outlier detection can be an effective method for identifying outliers in a data-set, particularly when the inlier and outlier data points are well-separated and easy to distinguish. However, it can be challenging to train a classifier to accurately distinguish between inliers and outliers, particularly when the inlier and outlier data points are not well-separated. Representative methods for classification-based outlier detection methods are multi-class and one-class [Singh and Upadhyaya (2012)]. Leaving the supervised realm there are outlier detection methods in the unsupervised setting as is elaborated below.

Clustering-based outlier detection: Clustering-based outlier detection methods involve partitioning the data set into clusters, and then identifying data points that are significantly different from the other data points in their cluster. These data points are considered to be outliers. There are a variety of different clustering algorithms that can be used for outlier detection, including K-means clustering [Likas et al. (2003)], Density-based clustering [Schubert et al. (2017)], and Hierarchical clustering [Sibson (1973)]. Clustering-based outlier detection can be an effective method for identifying outliers in a data-set, particularly when the outliers are significantly different from the rest of the data points in some way (e.g., they are much larger or smaller, or have different characteristics). Among the various clustering algorithms, some have already 'built-in' concepts of noise/outliers like e.g. DBSCAN [Schubert et al. (2017)], where a data point is considered as an outlier if it is neither density-connected, nor density-reachable, nor a core or border point. However, it can be challenging to determine the appropriate number of clusters to use and to tune the parameters of the clustering algorithm to achieve good results, since we are operating in an entirely unsupervised setting. Leaving the unsupervised realm there are outlier detection techniques in the neural network setting as is elaborated below.

Neural network-based outlier detection: Neural networks can be used for outlier detection by training a neural network to distinguish between normal data points (inliers) and anomalous data points (outliers). Neural-based methods can operate in supervised as well as unsupervised settings. There exist different approaches when it comes to using neural networks for outlier detection, including:

Autoencoders: An autoencoder [Vincent et al. (2010)] is a type of neural network that is trained to reconstruct its input data. When an autoencoder is trained on inlier data points, it should be able to reconstruct those data points accurately due to the universal approximation theorem [Hornik (1991)]. Outliers, on the other hand, may be difficult for the autoencoder to reconstruct, indicating that they are anomalously contributing to an increased reconstruction error.

One-class classification: A neural network can be trained to perform one-class classification, where it is only trained on inlier data points and must distinguish between inliers and outliers. Examples of one-class classification algorithms are proposed by the authors in [Ruff et al. (2018)] and [Khan and Madden (2010)]

Anomaly detection: A neural network can be trained to perform anomaly detection, where it is trained on both inlier and outlier data points and must distinguish between the two. This means that in this scenario, contrary to one-class classification, the network has to capture and therefore learn the characteristics of both, inlier and outlier. Examples of neural anomaly detection methods are Anomaly Detection using One-Class Neural Networks [Chalapathy, Menon, and Chawla (2018)] and Anomaly Detection using Recurrent Neural Networks[O'Shea, Clancy, and McGwier (2016)].

Using neural networks for outlier detection can be an effective method, particularly when the inliers and outliers have complex patterns that are difficult to capture with other methods. However, training a neural network can be computationally expensive and may require a large amount of labeled data. Additionally, neural approaches suffer from a black-box character rendering it difficult to follow what has actually been learned or what the characteristics of the derived patterns actually are [Islam, Eberle, Ghafoor, and Ahmed (2021)]. Leaving the neural network realm there are outlier detections in the probabilistic setting as is elaborated below.

Bayesian networks-based outlier detection: Bayesian networks are probabilistic graphical models that represent the dependencies between variables as a directed acyclic graph (DAG). Bayesian networks can be used for outlier detection by representing the data points in a data-set as variables in the network and modeling the dependencies between the variables using a DAG. Outliers can then be identified as data points that have significantly different probabilities or dependencies compared to the other data points in the network. Prominent examples for bayesian network-based outlier detection are the works of [Ogbechie, Díaz-Rozo, Larrañaga, and Bielza (2017)] and [Rashidi, Hashemi, and Hamzeh (2011)].

One potential benefit of using Bayesian networks for outlier detection is that they can handle complex dependencies between variables and can incorporate prior knowledge about the relationships between variables. However, constructing a Bayesian network can be challenging, particularly when the number of variables is large or when the relationships between variables are not well-understood. In addition, Bayesian networks can be computationally expensive to work with, particularly when the network is large or the data is high-dimensional.

It is important to consider the context of the data and the potential causes of the outliers when interpreting the results of the analysis. Outliers may be caused by errors in data collection or measurement, or they may indicate an unusual or unexpected event or phenomenon that is worth further investigation.

2.2 Fuzzy Clustering

Fuzzy clustering, also known as fuzzy c-means clustering [Bezdek, Ehrlich, and Full (1984)], is a machine-learning technique used to group data points into clusters, where each data point can belong to more than one cluster with a degree of membership. This is in contrast to traditional clustering algorithms, which assign each data point to a single cluster. The overall picture of Fuzzy Clustering methods is elaborated on in [Grover (2014)]. Fuzzy clustering is useful in situations where it is not clear to which cluster a data point belongs to, or where the data points may belong to multiple clusters. It is often used in image and text classification, data mining, and other applications where there may be overlap or uncertainty in the data.

To perform fuzzy clustering, the data points are first randomly initialized into clusters, and the degree of membership of each data point in each cluster is calculated using a membership function. The membership function assigns a membership value between 0 and 1 to each data point, with a value of 1 indicating that the data point belongs fully to the cluster and a value of 0 indicating that it does not belong to the cluster at all.

The membership values are then used to update the centroids (representative points) of the clusters, and the process is repeated until the membership values and centroids converge to stable values. The final clusters are then formed by assigning each data point to the cluster with the highest membership value.

There are several variations of the fuzzy c-means algorithm, including the possibilistic c-means (PCM) algorithm and the modified PCM (MPCM) algorithm. These variations can be used to improve the accuracy and stability of the clustering process. Through the probabilistic nature of fuzzy clustering, the first indications can be obtained if an object is ”in-between” multiple clusters, like if the probability of a point p belonging to a cluster C1 and C2 is 0.5 each. However, this approach does not provide any geometric information such as if a cluster is ’behind’ another one, which can especially occur in cases of arbitrarily shaped clusters. Furthermore, c-means, like other k-means and k-medoid variants yield as default, due to their underlying clustering model, convex clusters, rendering it unsuitable for cases where clusters are arbitrarily shaped. This raises the need for a more sophisticated approach to detecting in-between instances.

2.3 In-between Instance Detection

To detect in-between instances in a data-set that has been partitioned into clusters, we can rely on a variety of techniques, including:

Nearest neighbor algorithms: These algorithms identify the data points that are closest to the instance being classified, and use the class or label of those data points to predict the class or label of the instance. Although nearest neighbor algorithms [Muja and Lowe (2014)] operate in an unsupervised manner, they lack a specific underlying model for characterizing if an instance is actually "in-between" two or more classes.

Decision trees: Decision trees are used to classify instances based on their features or attributes. They work by building a tree-like model that splits the data into recursively smaller groups based on the values of the features until a decision can be made about the class or label of the instance. While decision trees provide explainable models [Rokach and Maimon (2005)], they are operating in a supervised setting, meaning that labels are required which we may not have in an unsupervised setting rendering them unfeasible.

Support vector machines (SVMs): SVMs [Hearst, Dumais, Osuna, Platt, and Scholkopf (1998)] are a type of supervised machine learning algorithm that can be used to classify in-between instances. They work by finding the hyperplane that maximally separates the different classes in the data-set, followed by using that hyperplane to classify new instances. Besides their supervised nature, which renders them unsuitable, they furthermore are limited to linear boundaries (unless kernels are used) making them unflexible for detecting in-between instances.

Neural networks: Neural networks are a type of machine learning algorithm that can be used to classify in-between instances. They work by building a network of interconnected nodes, or artificial neurons, that can learn to recognize patterns in the data and make predictions based on those patterns. While neural network techniques are indispensable and are used for the embedding of complex objects (see experimental section), they suffer primarily from the lack of explainability which is owed due to their black-box nature. This raises the need for our proposed approach to be explainable in some sense (i.e. geometric intuition).

To summarize, to the best of our knowledge, none of the so far listed approaches satisfy our needs to have an (a) unsupervised (b) unbound to any linear limitations, and (c) an explainable model for detecting in-between instances.

It is important to carefully evaluate the performance of the algorithm used for in-between instance detection, as the accuracy of the predictions can have significant implications for the application being studied. To get an idea of how we can detect an In-between instance we have looked into this paper [Ramaswamy, Rastogi, and Shim (2000)] that provided the initial impulse to utilize the concept of nearest neighbors to identify outliers.

3 Main Method

This section serves the purpose of formally introducing a first notion for in-between instances, followed by a definition of the in-between instance detection task in general. Furthermore, it is elaborated on the geometric intuition of our in-between instance model, including a proposed algorithm and detailed elaborations. The approach to identifying in-between instances is covered by several parts, as it is elaborated on in the following:

3.1 What is an in-between instance, and how is it defined?

Definition 3.1 (In-between Instances)**.** Given a clustering $\mathcal{C} = \{c_1, .., c_n\} \subseteq \mathcal{D} \in \mathbb{R}^d$. Furthermore, we are given an outlier instance $o \in \mathcal{D}$. o is considered as an in-between instance if:
(1) it is surrounded by at least two clusters c_i, c_j.
(2) As a pruning criterion, we are excluding other clusters from the k nearest neighbor (kNN) clusters of o under the condition that the cosine distance between o and any other cluster $c_m \in c_{kNN}(o)$ with respect to c_i (or c_j) as a reference (origin) point is:

$$d_{cos_{c_i}}(o, c_m) \geq 1$$

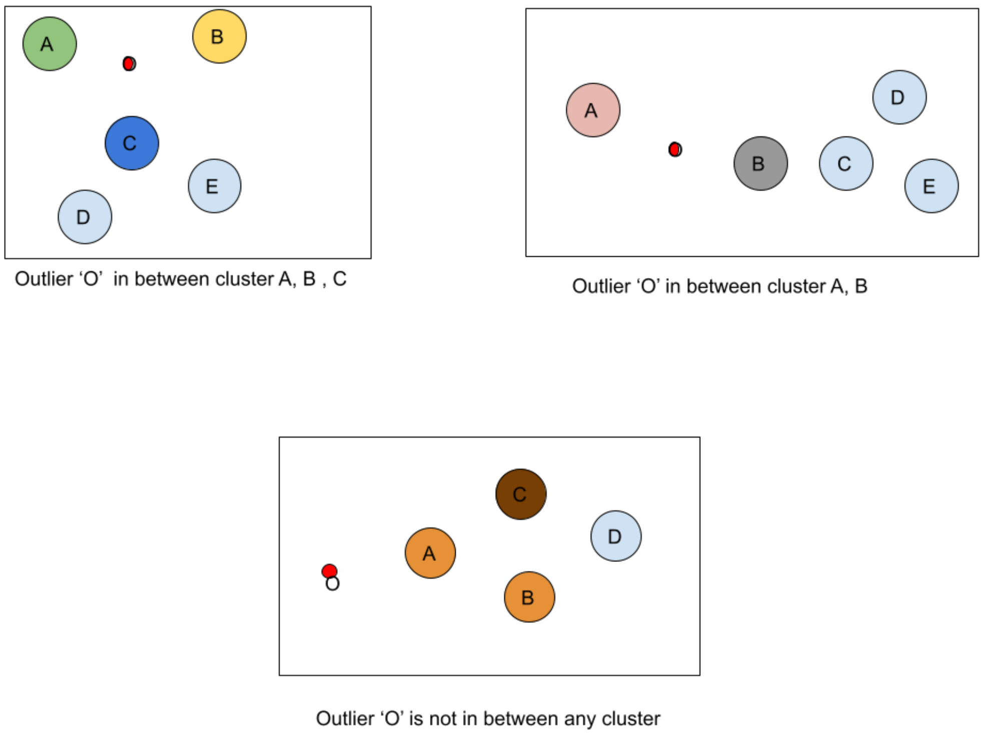

A
B
C
D
E
Outlier ‘O’ in between cluster A, B , C
A
D
B
C
E
Outlier ‘O’ in between cluster A, B
C
D
A
O
B
Outlier ‘O’ is not in between any cluster

Figure 1: Object in between clusters

The first property (1) of Definition 3.2 is illustrated in Figure 1. In the top-left figure, we can see that the outlier (red dot) is an in-between instance to the clusters A, B, and C. In the top-right figure, our in-between instance is surrounded by clusters A and B. In the bottom figure, we got the case where our instance is an outlier since it is at maximum surrounded by a single cluster (A), thus not satisfying (1) of Def. 3.2 of being surrounded by **at least** two clusters. We'd like to highlight at this point that, although the clusters are illustrated through convex shapes, they can as well be arbitrarily shaped in our proposed algorithm.

Definition 3.2 (In-between detection task)**.** Given a data-set $\mathcal{D} \in \mathbb{R}^d$. Furthermore, we are given two subsets of $\mathcal{D}$, namely a clustering $\mathcal{C} = \{c_1, ..., c_n\} \subset \mathcal{D}$ that is a partitioning of $\mathcal{D}$ (a set of sets), and a set of outlier objects $\mathcal{O} = \{o_1, ..., o_m\} \subset \mathcal{D}$. Additionally, it holds that:

$$\forall c_i \in \mathcal{C} : c_i \bigcap \mathcal{O} = \emptyset$$

and

$$\left(\cup_{i=1}^{n} c_i \in \mathcal{C}\right) \bigcup \mathcal{O} = \mathcal{D}$$

The in-between detection task is to find for given sets $\mathcal{C}$ and $\mathcal{O}$ a set of in-between instances $\Lambda = \{\lambda_1, ..., \lambda_r\} \subseteq \mathcal{O}$ as stated in Def. 3.2., that satisfy the following criteria, namely
(1) λ is surrounded by at least two clusters c_i and c_j
(2) based on a predicate Ψ it is determined if a cluster c_i is actually surrounding λ

In the scope of this thesis, this predicate Ψ is based on the cosine distance between λ and another cluster c_i w.r.t. another cluster c_j acting as a reference (origin) point, as stated in Def. 3.2).

3.2 What are the stages of the in-between instance detection task?

In this work, the in-between detection task is comprised of three stages, namely:
First phase:
First of all, we need a collection of clusters and outliers of a data set. For that, we have used **Agglomerative Hierarchical Clustering** because we can explicitly define the threshold distance to create a new cluster which is not possible in the K-means clustering algorithm. First, we applied the agglomerative algorithm with no distance threshold and observe the dendrogram.

A dendrogram is a diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from hierarchical clustering. The main use of a dendrogram is to work out the best way to allocate objects to clusters. After observing the dendrogram we defined the distance threshold. Then we again run the agglomerative algorithm. As a result, we have got a set of clusters where among them some clusters contain only a few instances that are recognized as outliers.

Second phase:
After having a set of clusters and an outlier we have computed the k-nearest neighbor of a particular outlier by applying euclidean distance. These k-nearest clusters will be further filtered based on a predicate Ψ, to get precise neighbor clusters where the outlier will be in between them.

Third phase:
After having the k-nearest cluster for a particular outlier we have applied **In-between Instance** algorithm that we have defined is given below:

Algorithm 1 In-between Instance

```
Require: $clusterLabel, clusterMean[], Outlier$
1:
Ensure: $[(inbetweenobject, [neighclust1, neighclust5, ...]), (), ...]$
2: $K\_NearestCluster \leftarrow []$
3: for $i$ in $clusterLabel$ do
4:     if $i == Outlier$ then
5:         $continue$
6:     end if
7:     $K\_NearestCluster.append((distance.euclidean(clusterMean[outlier, clusterMean[i]), Outlier, i))$
8: end for
9: $K\_NearestCluster.sort(key{=}lambda\ y : y[0])$
10: $K\_NearestCluster \leftarrow K\_NearestCluster[: 5]$
11: $closestCluster \leftarrow K\_NearestCluster[0][2]$
12: $closestDistance \leftarrow K\_NearestCluster[0][0]$
13: $candidateCluster \leftarrow []$
14: $rejectedCluster \leftarrow []$
15: $candidateCluster.append(K\_NearestCluster[0])$
16: for $i$ in $K\_NearestCluster$ do
17:     if $i$ in $rejectedCluster$ then $continue$
18:     end if
19:     for $j$ in $K\_NearestCluster$ do
20:         if $i == j$ then $continue$
21:         end if
22:         $origin \leftarrow clusterMean[i[2]] + (-clusterMean[i[2]])$
23:         $translateOutlier \leftarrow clusterMean[i[1]] + (-clusterMean[i[2]])$
24:         $translateCluster \leftarrow clusterMean[j[2]] + (-clusterMean[i[2]])$
25:         $cosineDistance \leftarrow distance.cosine(translateOutlier, translateCluster)$
26:         if $cosineDistance < 1$ then
27:
28:             if $j$ in $candidateCluster$ then continue
29:             end if
30:             $candidateCluster.append(j)$
31:         else
32:             if $j$ not in $rejectedCluster$ then
33:                 $rejectedCluster.append(j)$
34:             end if
35:             if $j$ in $candidateCluster$ then
36:                 $candidateCluster.remove(j)$
37:             end if
38:         end if
39:     end for
40: end for
41: for $i$ in $rejectedCluster$ do
42:     if $i$ in $candidateCluster$ then
43:         $candidateCluster.remove(i)$
44:     end if
45: end for
46: return $candidateCluster$
```

In this algorithm we have provided three inputs. One is the cluster label. Another input is a list of cluster mean and the last input is an outlier. As output, we are expecting the list of tuples where it will contain only those clusters that satisfy our 'In-between Instance' definition for the given outlier. If our output list contains only a single element, we will not consider our outlier as an in-between instance.
First of all, we have computed k-nearest clusters for the given outlier in the form of a list of tuples. Where each tuple contains the euclidean distance (cluster-mean, outlier), outlier, and cluster label(see Alg. 1, L.2 - L.10). After that, we have iterated over all clusters ($i \in K\_NearestCluster$, Alg. 1, L.16) which we can see in our algorithm from lines 17 to 40. Within each iteration (Alg. 1, L.16) we again iterate over all clusters except the own cluster (see Alg. 1, l.20) from the first iteration loop from lines 19 to 39. In the second iteration loop for a cluster, we have translated the coordinate of the entire dataset to the new origin, of the current reference cluster $i[2]$ defined by a cluster mean. That is why we have translated other related elements according to the new origin which is implemented from lines 22 to 25. A translation is a shift from one location to another, without any change in size or orientation. We can have a look at figure 2 to get a visual understanding of geometric translation.

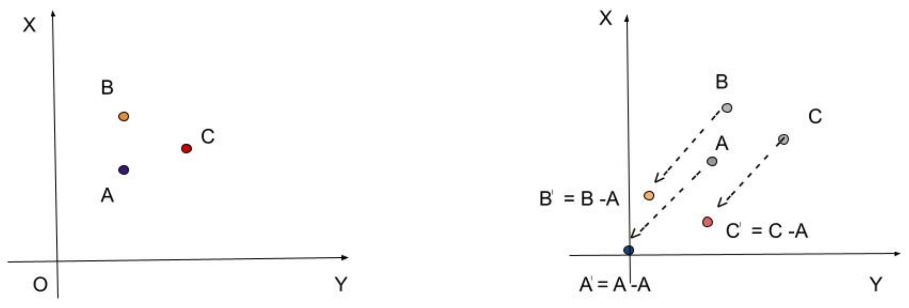


Figure 2: Geometric Translation

After translating the cluster to the origin we have calculated the cosine distance between the outlier and other clusters with respect to the cluster that was translated to the origin. The cosine distance captures the angle α between e.g. an outlier and a cluster with respect to e.g. the first cluster as the origin. If the cosine distance is greater than 1 we have discarded those clusters from the candidate cluster list because they are, from a geometric intuition, behind other clusters.

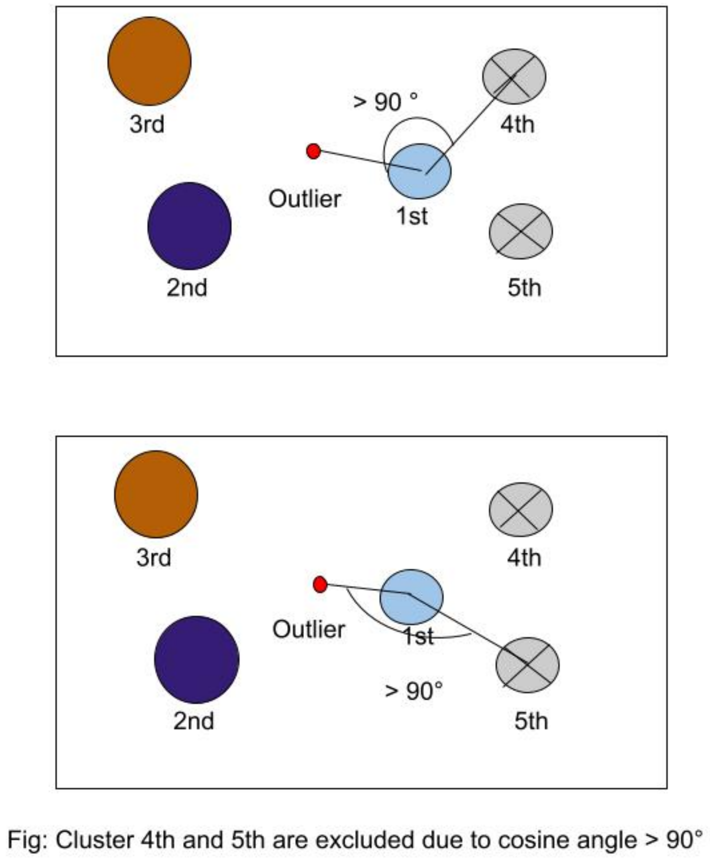

3rd
> 90 °
4th
Outlier
1st
2nd
5th
3rd
4th
Outlier
1st
> 90°
2nd
5th
Fig: Cluster 4th and 5th are excluded due to cosine angle > 90°

Figure 3: Object in between clusters

In figure 3 we have seen that the 4th and 5th clusters are not considered as neighboring clusters for the outlier (our potential in-between instance). This is because both these clusters are behind of 1st cluster which is closer to the outlier. We have excluded them from the candidate neighbor cluster list by calculating the cosine distance with outlier with respect to 1st cluster which is > 1. We have implanted this operation from lines 18 to 31 in our algorithm.
We have used python language and *scipy* package in our algorithm where cosine distance < 1 means the cosine angle is in between $\cos(\alpha) = 90^\circ$ and $-\cos(\alpha) = 90^\circ$. And cosine distance > 1 means the cosine angle is in between $\cos(\alpha) = 90^\circ$ and $\cos(\alpha) = 270^\circ$. We have considered cosine distance < 1 to find out a neighbor cluster of an outlier because it provides a convex surround cluster. Finally, we have got the nearest neighbor cluster of in-between instances by filtering the candidate cluster list that is implemented from lines 36 to 41 in our algorithm. We can reduce the value of cosine distance to get a more precise and opposite cluster. Here precise opposite cluster means if one cluster is located on one side of the outlier and another is located on the other side by almost 180^0 angle. We can get a clear idea by looking at the following figure:

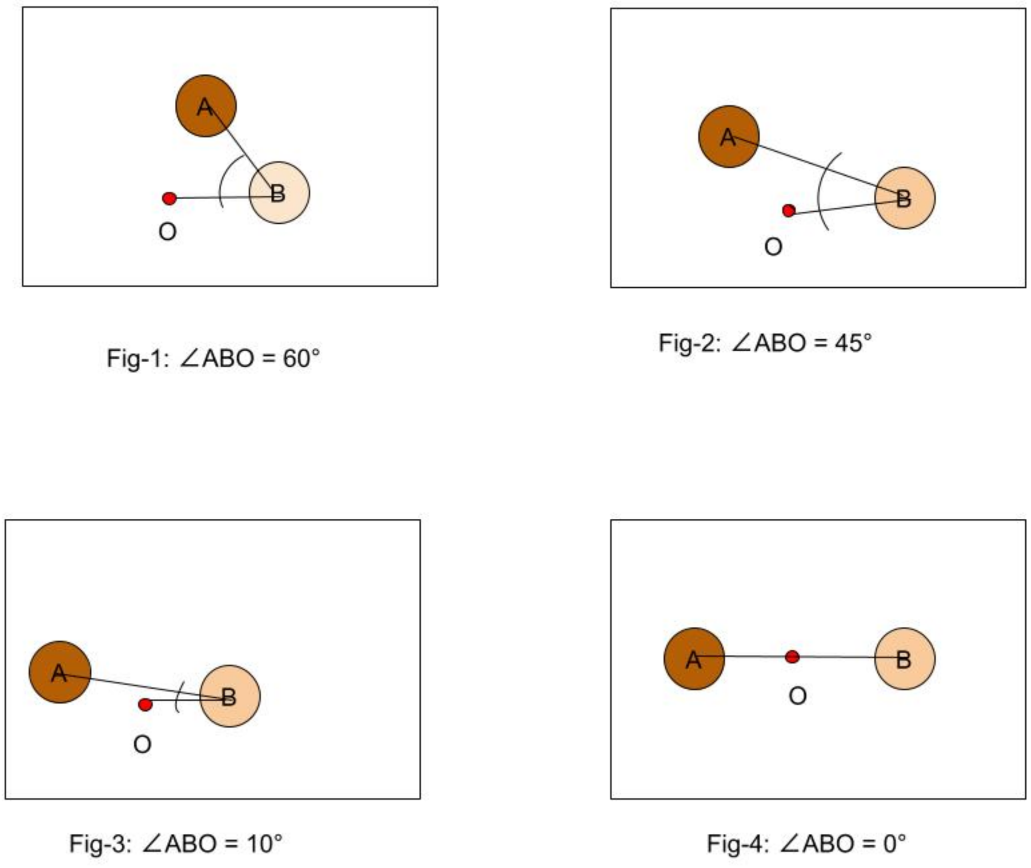


Figure 4: Reducing angle to get more opposite cluster

Here in this figure 4, we can see that in the 1st sub-figure cluster-A and cluster-B are relatively on the same side from the outlier-o and the angle between the outlier-o and cluster-A is 60° with respect to cluster-B. In the 2nd sub-figure, these 2 clusters are located on relatively opposite sides considering the outlier position where the cosine angle between outlier and cluster-A is 45°. Based on the next two subsequent figures we can observe, the more we decrease the cosine angle threshold the more restrictive we become by allowing primarily opposing clusters.

3.3 Challenges/Cases to take care of/insights

To obtain our expected output we have faced several kinds of challenges which we elaborate on in the following:

- First of all, We are looking for a data-set with some clusters and outliers. For that, we have decided to use *K-means clustering algorithm* but we have realized that by applying *K-means algorithm* we do not obtain outliers. As a remedy to this issue, we have decided to use *Agglomerative clustering algorithm.* To define the distance threshold to cut a cluster we have applied this algorithm without a distance threshold and observed the dendrogram to select the distance threshold. After having the threshold, we again implanted this algorithm over the data-set with the computed distance threshold. As a result, we have got a set of clusters and outliers.

- After having a set of clusters we have found out k-nearest cluster for an outlier by euclidean distance. But we observed that all k-nearest clusters are not important to consider as neighbor clusters of the outlier because some clusters are behind other clusters. But we need those clusters that are directly faced by in-between instances. For that, we have considered the cosine distance between two clusters since the cosine similarity takes through the enclosed angle between (two) vectors and also to some extent the geometry. If the cosine distance is less than 1 we have decided that one cluster is followed by other clusters. So which is closer than the other we have considered that and discarded the other one.

- After implementing our new approach to detect in-between instances to our synthetic data-set we did not get our desired output. Our new method was not able to discard those clusters that are behind other clusters. This made it necessary to identify the problem. We have discovered an error in our implementation. The root of the problem lies in the fact that while we are computing the cosine distance between two clusters, we are considering an origin at the null vector (0,0) in a Cartesian coordinate system. But it was supposed to consider our outlier as an origin. In order to be capable to determine what is behind another cluster we need to take the viewpoint from the perspective of an outlier, which requires the outlier to be considered as the center / 'null point'. For that, we have geometrically translated our outlier to the origin and compensated other elements according to our new origin. Here with the term 'compensate' we refer to the necessity to not only translate our outlier but also other elements/objects in order to maintain the relative properties such as distances and angles between the objects.

- Unfortunately we didn't get our expected output yet. Actually, we need to calculate the cosine distance between an outlier and a cluster with respect to another cluster. This is because we are detecting whether a cluster is behind other clusters by considering the outlier. So we have translated a cluster to the origin(0,0) and calculated the cosine distance between the outlier and other clusters to decide whether they are behind or not of this origin cluster. Considering all the previous insights and improvements, the expected outcome/output is obtained. The journey sketched here shows that the geometry-oriented in-between instance detection is far away from being trivial.

4 Evaluation

Building on the previously presented and elaborated algorithm to detect in-between instances we owe some "proof" that convinces the reader of the effectiveness of the proposed method. For that purpose, we provide in this section several experiments. On different data-sets, we apply our approach and inspect the results with the following question in mind: does our algorithm detect in-between instances and the enclosing clusters?

The experiments are conducted on artificial and real-world data. The artificial data-sets bring the benefit of a controlled environment, allowing us to stress-test our approach on data-sets/scenarios with specific pre-defined properties. We have applied our proposed In-between instance detection method to 3 different data-sets. The first one is **Synthetic data-set**, the second one is **Consumer Complaint data-set** and the final one is **Molecular data-set**. In the following we elaborate on each of the three data-set types, with a focus on the following aspects: (1) What are the properties of the data-sets (number of samples, dimensionality, number of clusters, etc.) (2) What is the goal of the conducted experiment/what are we investigating? (3) What could we observe? Followed by a discussion of the results.

4.1 Synthetic data-set

In the following, we have designed a synthetic two-dimensional data-set to conduct our experiments on. This is done with the goal in mind to have a setting where the impact of single changes can be easily followed, without the risk of being obfuscated by the influence of factors that we could not foresee. Also, it permits insights into angular and euclidean distances interplay in the discovery of in-between instance surrounding clusters. Our data-set consists of seven sparse convex and well-separated clusters. As it would be beyond the scope of this thesis, it remains for future work to investigate the effects of arbitrarily shaped and potentially overlapping clusters. Besides the convex and well-separable clusters a single outlier, hence our potential in-between instance, is introduced. In the following, we study the impact of choosing different cosine distance thresholds on the outlier surrounding clusters that are identified by our approach. synthetic data over 2d space to experiment with our algorithm. We can see the data distribution given in figure 4.1.1 below.

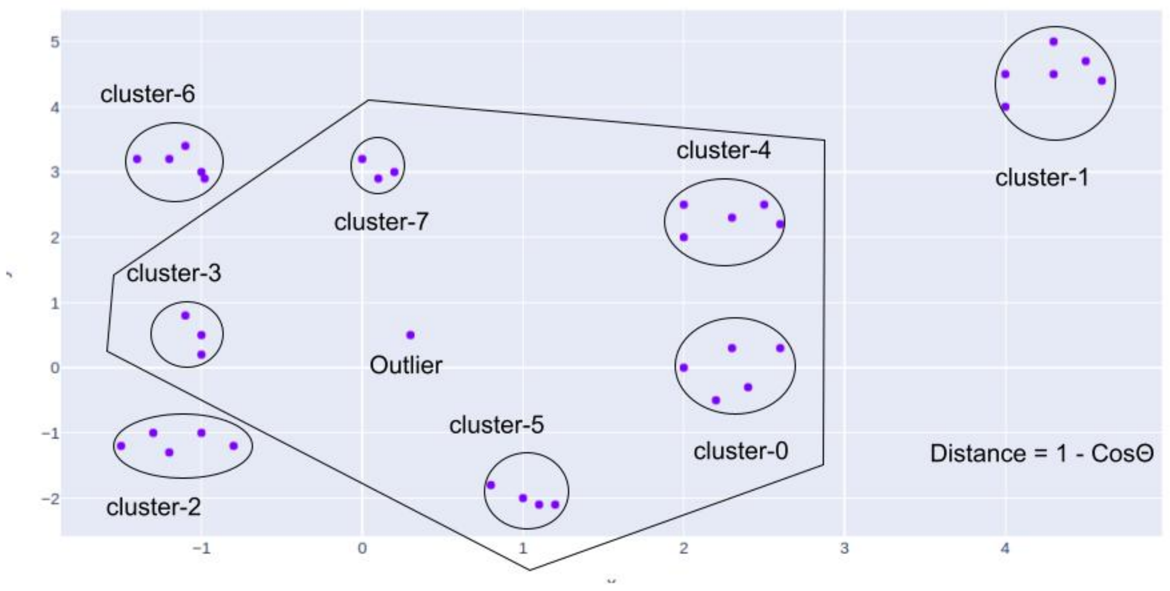


Fig-4.1.1: Outlier is an in-between instance of cluster-0, 3, 4, 5, 7 where cosine distance < 1

Here in the figure 4.1, we can see that there are eight clusters and one outlier where cluster-1 is **behind** cluster-4. Cluster-2 is also **behind** cluster-3 and cluster-6 is **behind** cluster-7. After applying our 'In-between Instance' algorithm, we have obtained cluster-0, 3, 4, 5, and 7 as neighbor clusters where we have considered the cosine distance less than 1 as the pruning criterion. The insights gained from this first experiment is, that when we choose a cosine distance threshold of slightly below 1, we have a significantly large number of neighboring clusters around the potential in-between instance. However, it serves its original purpose of pruning away clusters that are **behind** other clusters, showing the effectiveness of this geometric (angular-based) approach.

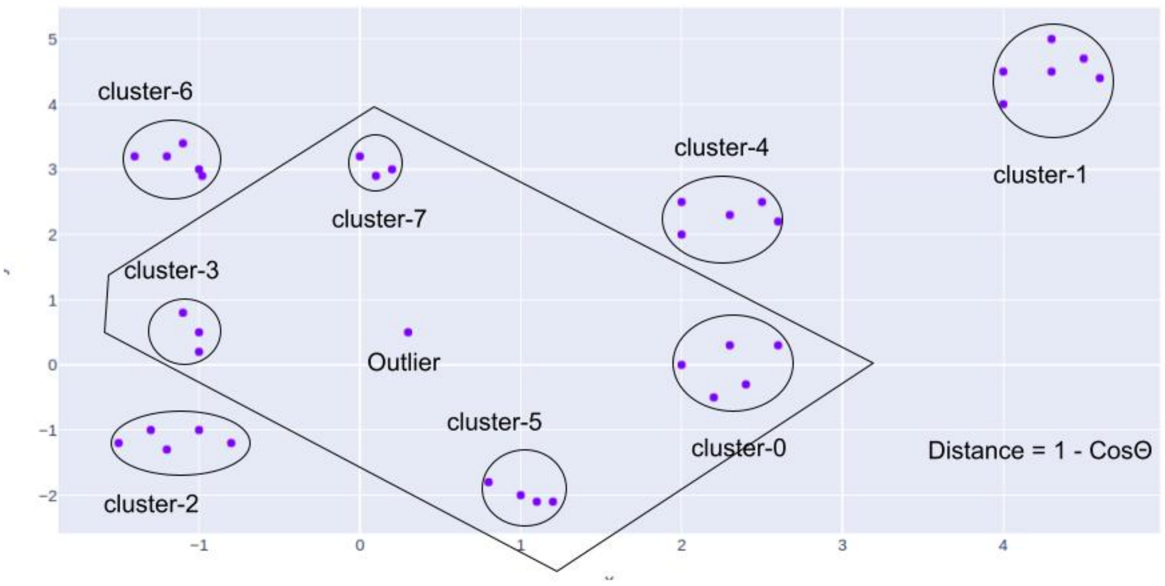


Fig-4.1.2: Outlier is an in-between instance of cluster-0, 3, 4, 5, 7 where cosine distance < 0.7

We remain with our artificial data-set and want to investigate the effects of further lowering the cosine distance threshold. Here we can see that when we reduced the cosine distance < 0.7 cluster-4 has been discarded. This is because when we reduce the cosine angle, the viewpoint with respect to the particular cluster becomes narrower. Here the cosine distance between *outlier* and *cluster-4* with respect to *cluster-0* is $>= .7$. According to **definition 3.2**, cluster-4 has been excluded to detect In-between instances. Now clusters-0, 3, 5, and 7 are neighboring clusters of *In-between instances*'.

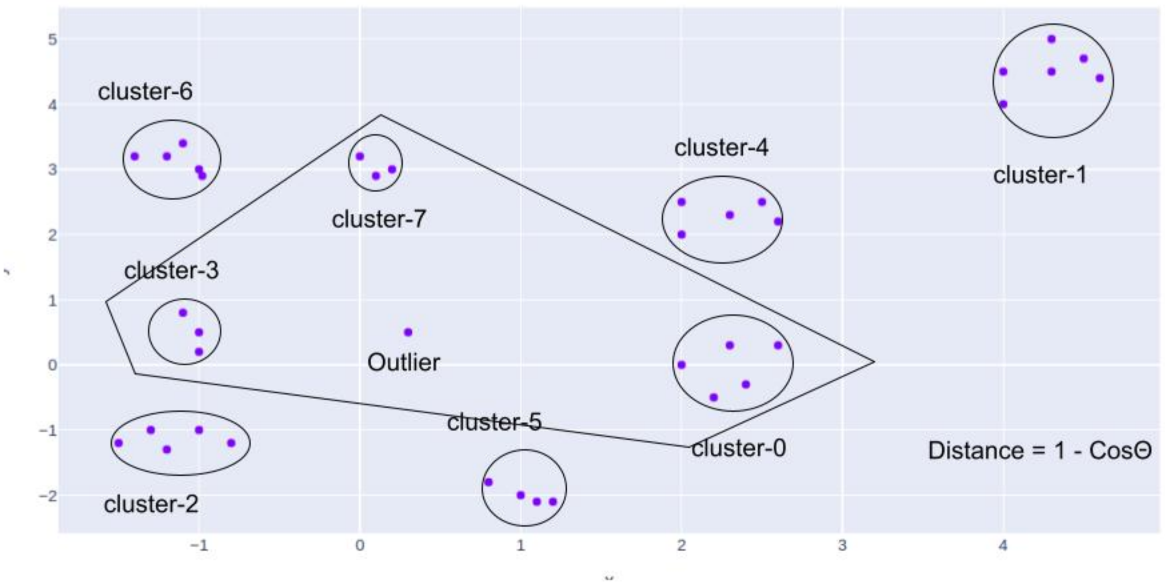


Fig-4.1.3: Outlier is an in-between instance of cluster-0, 3 7 where cosine distance < 0.6

We furthermore stress the cosine distance parameter where we can see that when we reduced the cosine distance < 0.6 cluster-5 has been discarded from the previous neighbor cluster list. Now clusters- 0, 3, and 7 are neighboring clusters of *In-between instances*. Although the reader may be tempted to consider cluster-5 as the closest cluster to the outlier, this is not considered, since we are computing the distance of the outlier to the cluster mean (see Algorithm 1.).we'd like to highlight at this point that the horizontal axis is different to the vertical axis, due to the scaling of the image in this document.

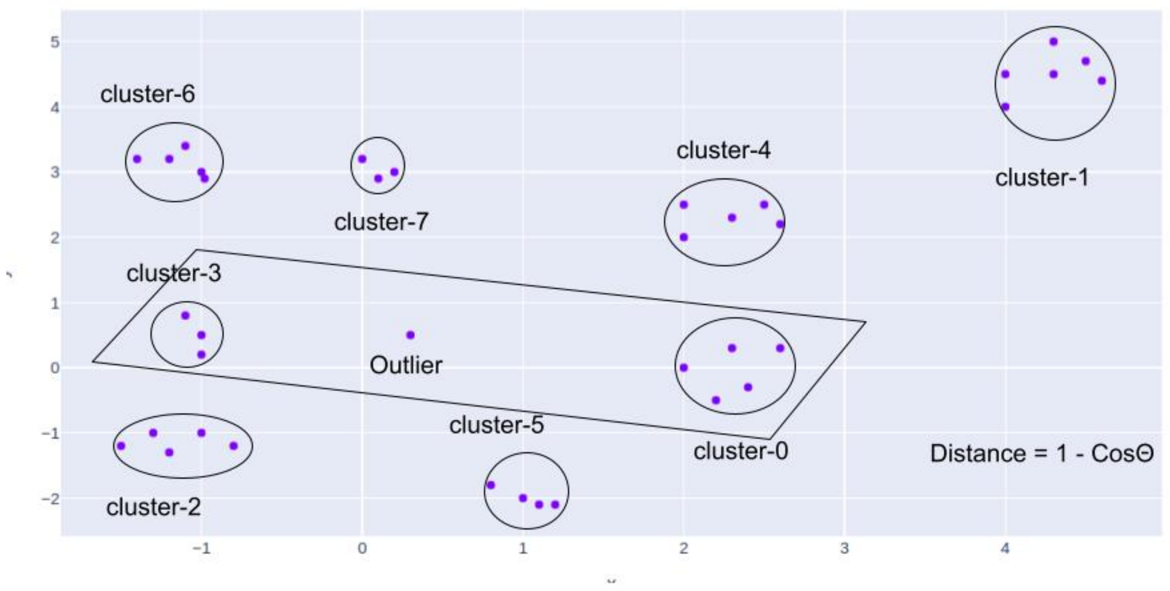


Fig-4.1.4: Outlier is an in-between instance of cluster- 0 , 3 where cosine distance < 0.5

In the next experimental phase when we reduced the cosine distance threshold down to < 0.5 cluster-7 has been discarded from the previous neighbor cluster list. Here the cosine distance between *outlier* and *cluster-7* with respect to *cluster-3* is >= .5. According to **definition 3.2**, cluster-7 has been excluded to detect In-between instances. Now clusters-0 and 3 are neighboring clusters of *In-between instances*'. So far we can observe a steady decrease of potential surrounding clusters w.r.t. our in-between instance. This however raises the question of 'when to stop. One can not in particular make statements that one choice of the cosine distance threshold is actually 'better' or 'worse'. It depends on the expected view, in a way if we want the neighboring clusters to be more distinguishable.

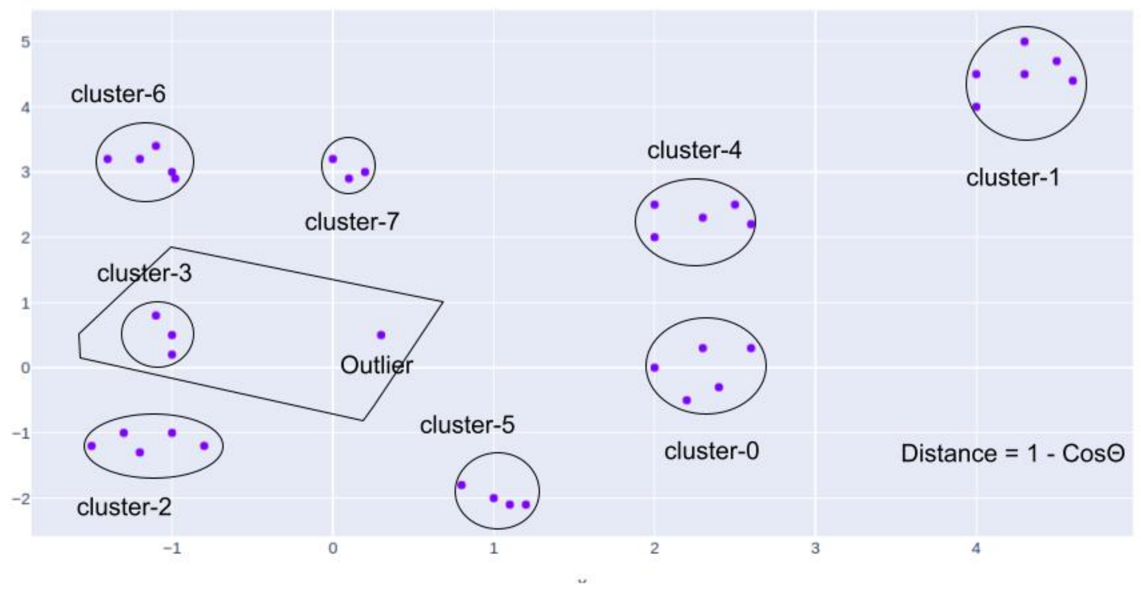


Fig-4.1.5: Outlier is not an in-between instance where cosine distance = 0

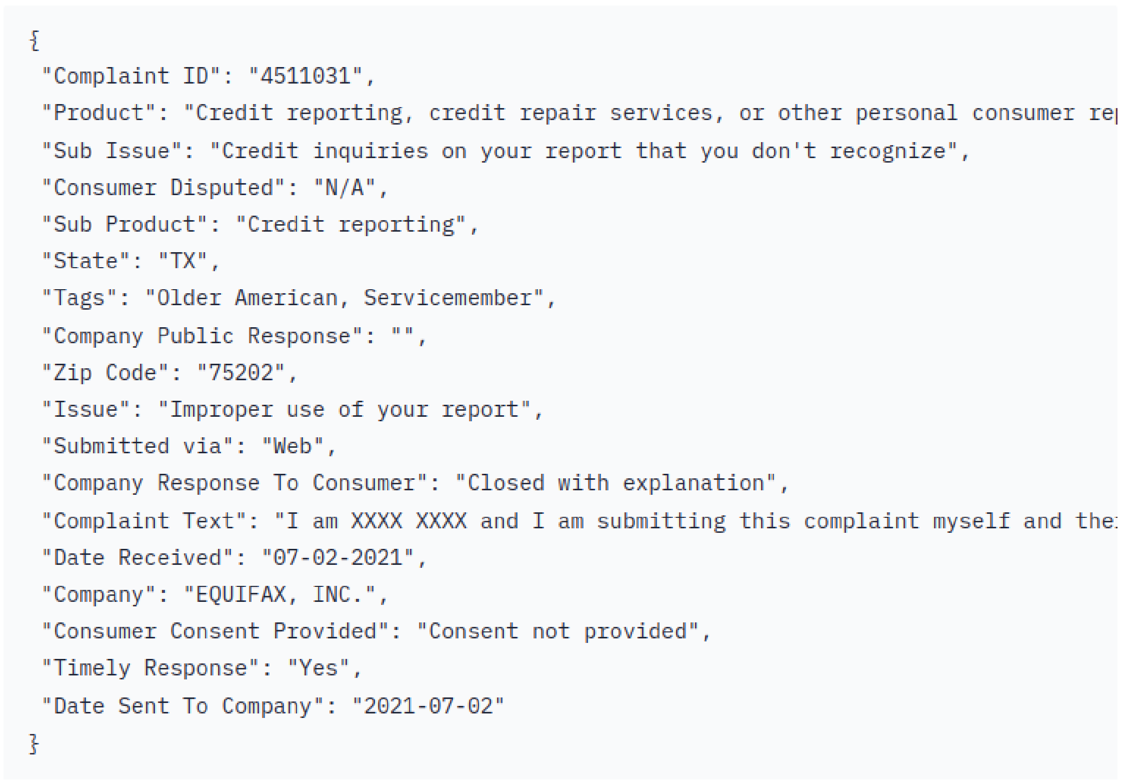
Finally in this last experimental stage we can see that the outlier is no more an *In-between instance* where the cosine distance is = 0. This is because we have obtained only one neighbor cluster which is cluster-3. But according to *definition 3.1* , we know that we need at least 2 neighbor clusters of an outlier to become an *In-between instance*.
After seeing the output of our algorithm implementation over the synthetic data set we have obtained a couple of observations which are:

- the angle parameter is a sensitive parameter to obtain better output. Small changes in this parameter lead to great changes in the in-between instance detection. Because of the smaller angle, it prunes more clusters, by restricting to mostly opposing clusters where pruning is meant in a sense as described in section 3.2 and in figure 4.

- we can see that there is an edge case. When we provide a minimal angle near zero, there is a great possibility that we will not get an in-between instance. The ideal angle would be in the range of 40 - 70 degrees. Then we can have those clusters which are relatively far away from each other. This is based on the assumption that the more restrictive the angle is (i.e. in Fig X, Cluster-3 and Cluster-0 being opposed at 180) the probability is considerably small of having such a particular constellation.

4.2 Consumer Complaint data-set

The Consumer Complaint Database is a collection of 40000 complaints about consumer financial products and services.
The data-set contains different information on complaints that customers have made about multiple products and services in the financial sector, such as Credit Reports, Student Loans, Money Transfers, etc. This work is considered U.S. Government Work. The data-set is a public data-set and it was downloaded from https://catalog.data.gov/data-set/consumer-complaint-database The date of each complaint ranges from November 2011 to May 2019. We have considered four thousand random complaints in our case.
The features/dimensions are:



```
{
 "Complaint ID": "4511031",
 "Product": "Credit reporting, credit repair services, or other personal consumer re
 "Sub Issue": "Credit inquiries on your report that you don't recognize",
 "Consumer Disputed": "N/A",
 "Sub Product": "Credit reporting",
 "State": "TX",
 "Tags": "Older American, Servicemember",
 "Company Public Response": "",
 "Zip Code": "75202",
 "Issue": "Improper use of your report",
 "Submitted via": "Web",
 "Company Response To Consumer": "Closed with explanation",
 "Complaint Text": "I am XXXX XXXX and I am submitting this complaint myself and the
 "Date Received": "07-02-2021",
 "Company": "EQUIFAX, INC.",
 "Consumer Consent Provided": "Consent not provided",
 "Timely Response": "Yes",
 "Date Sent To Company": "2021-07-02"
}
```

https://huggingface.co/data-sets/consumer-finance-complaints-data-set-structure
Figure 4.2.1: Consumer-finance-complaints-data-set-structure

We have considered **Complaint Text** features to cluster the data-set. The text has been embedded into a vector space using the method **doc2vec**[Kim, Kim, and Cho (2017)] embedding with 300 dimensions.

After performing hierarchical clustering we obtain the following clusters (c.f. fig 4.2.2, i.e. a 2D projection of the high-dimensional (already clustered) data using t-SNE; with a remark that t-SNE is just for visualization and does not capture the geometry aka geometry is lost when using t-SNE).

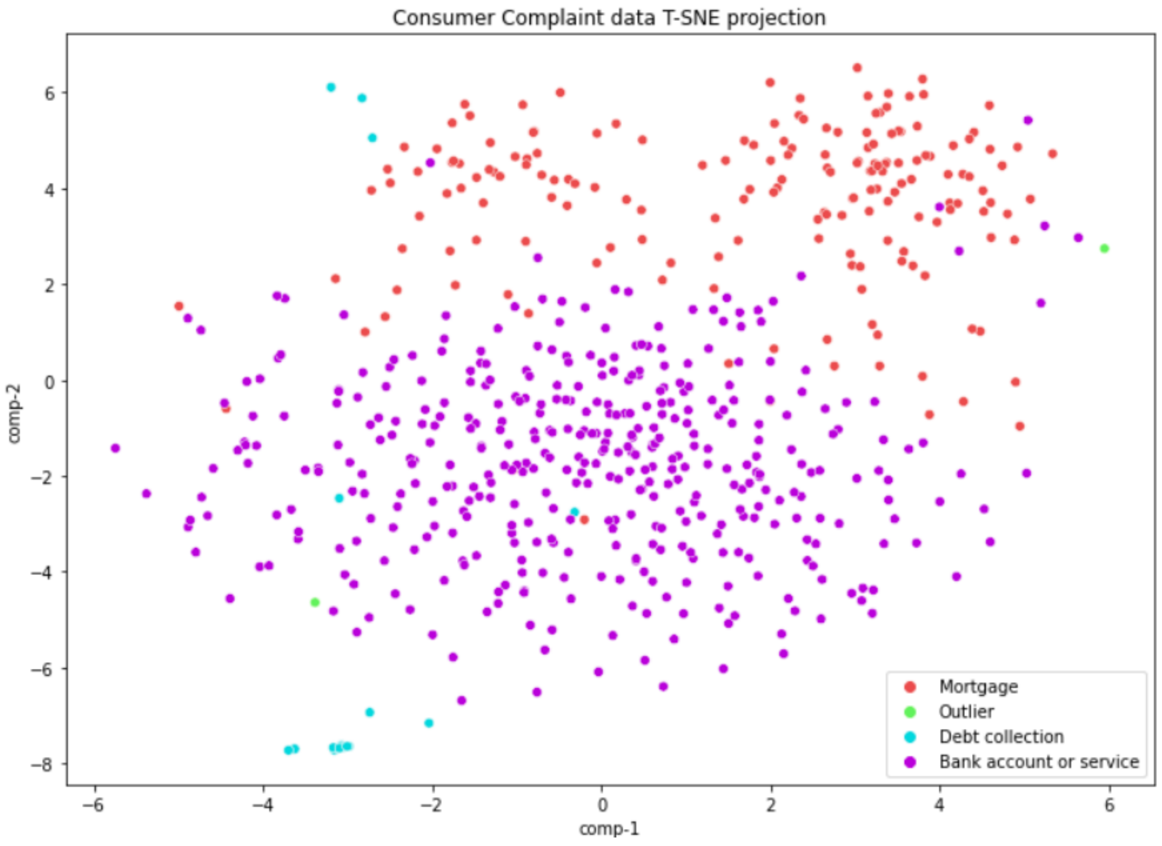


Figure 4.2.2: Consumer Complaint data T-SNE projection

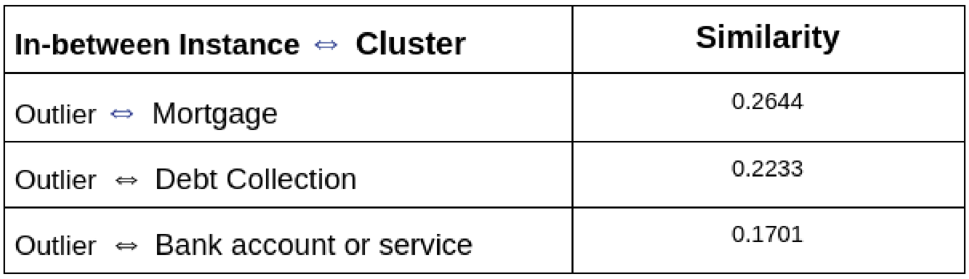
To find a similarity between a cluster and an In-between instance, we have computed a medoid for each neighbor cluster because a medoid is (contrary to a centroid) an existing object in a cluster. Medoids are representative objects of a data set or a cluster within a data set whose sum of the distance to all the objects in the cluster is minimal.

More formally, according to [Struyf, Hubert, and Rousseeuw (1997)] for a given set of n objects $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ with $x_i \in \mathbb{R}^d$ and a distance function $dist$ a medoid is defined as:

$$x_{medoid} = \underset{y \in \mathcal{X}}{\arg\min} \sum_{i=1}^{n} dist(y, x_i)$$

Actually, to find the similarity between two clusters we can not use the mean value of a cluster. This is because we need two existing instances in the computed **doc2vec model** to compute similarity where the mean value is not an existing one value. The solution for this problem is to use medoid instead of mean. After having a medoid of each cluster we have applied **Gensim Similarity Method** between two cluster medoid. This method finds the top-N most similar words/docs. Positive words/docs contribute positively towards the similarity, negative words/docs negatively. This method computes cosine similarity between a simple mean of the projection weight vectors/docs of the given words/docs and the vectors for each word/docs in the model.

We can see the similarity between In-between instances and other clusters in the following similarity table 4.2. At a first glance, the values may not seem that similar, yet they are significant for the following reason: If we take other clusters of the clustering, their corresponding similarity to the potential in-between instance is substantially low (i.e. 0.02 or 0.03). As a consequence, these similarity values underline one of the characteristics of in-between instances, namely that they exhibit a substantially higher similarity to their immediate surrounding clusters.



In-between Instance ⇔ Cluster	Similarity
Outlier ⇔ Mortgage	0.2644
Outlier ⇔ Debt Collection	0.2233
Outlier ⇔ Bank account or service	0.1701

Table 4.2: Similarity between In-between instance and neighbor clusters

In the following, we are having a look at the single clusters (take every single cluster that is involved in an in-between instance) and identify in a qualitative fashion what these complaints are roughly about (e.g. cluster 1 is about debt compensation, cluster 2 is about wrong data filled in the form etc.). However, while performing a more close inspection of the clusters, the embedding renders it difficult to find clear/sharp distinguishable criteria by which the single clusters can be semantically categorized. At this point, we have to emphasize that in our implementation we relied on a cosine distance of slightly less than 1. This in turn means that we consider less than 90° of an angle. As a consequence, if we consider less than 30° or 10°, the angular distance between clusters would be larger and therefore the content would be more distinguishable. Since we are also dealing with high-dimensional data ($d = 300$) distinguishing between different semantics is becoming increasingly difficult. Also, it lies in the *nature* of in-between instances to have shared amounts of semantics from their neighboring clusters, which furthermore renders a distinction in high-dimensional text embeddings significantly challenging.

4.3 Molecular data-set

Besides text-based embeddings there exist further complex structures encoded in a vectorized form. As another real-world example, the upcoming experiments are based on a data-set compiled in [Kazempour, Beer, Oelker, Kröger, and Seidl (2021)]. Here the authors collected from different publications potential molecule candidates that were identified to be effective against SARS and MERS. The data-set encompasses $n = 100$ molecules embedded via the Mol2Vec framework by [Jaeger, Fulle, and Turk (2018)] into a $d = 300$

dimensional feature space.

Since we do not have the required domain knowledge the results from applying the in-between instance detection were forwarded to domain scientists in the field of biochemistry (credits to the Swedish University of Umea). In a more chemical/structural context it was of interest for the domain scientists to see if in-between instances actually share certain structural aspects with members of their surrounding clusters.

From the evaluation by the domain scientists one could obtain the following insights:

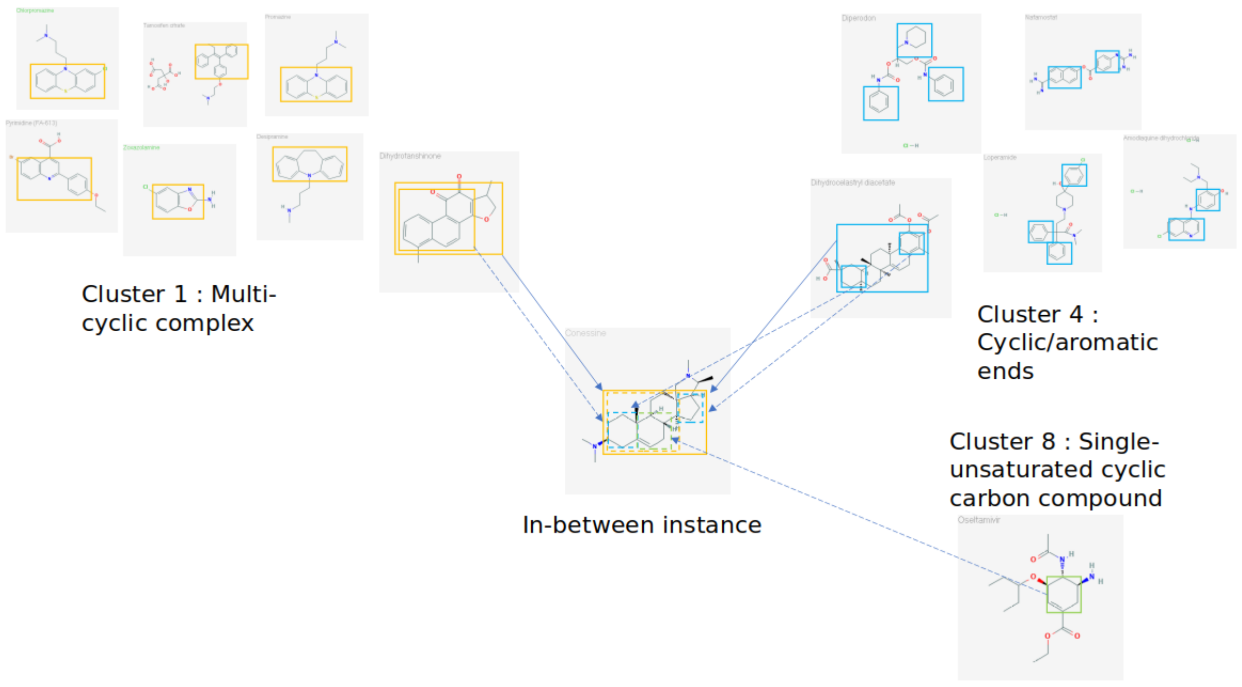

Cluster 1 : Multi-cyclic complex
Dihydrotanshinone
Conessine
Dihydrocelastryl diacetate
Diperodon
Loperamide
Cluster 4 : Cyclic/aromatic ends
In-between instance
Cluster 8 : Single-unsaturated cyclic carbon compound
Oseltamivir

Figure 4.3.1: Neighbor clusters are cluster-1, 4 and 8 for In-between instance

Here in the above figure 4.3.1, we can see that the outlier *Conessine* molecule is an *In-between instance* of cluster-1, cluster-4, and cluster-8. According to the domain experts, cluster-1 is characterized through so-called **Multi-cyclic complexes** [orange boxes] (or in simpler terms: several rings that are close to each other) that are also visible in our detected **In-between instance**. While cluster-4 also consists of several cyclic structures, they have the characteristics of being prominently visible at the 'ends' of the respective molecules. In terms of the domain scientists, the compounds in cluster-4 exhibit **Cyclic/aromatic end** that are also contained in **In-between instance**. The single compound in cluster-8, exhibits a **single unsaturated cyclic carbon compound** that can be seen in **In-between instance** as well. Upon request, if the domain experts could tell us why this cyclic carbon compound with a single unsaturated site is of interest/importance, their answer was that by having only a single unsaturated site, the risk of having a mixture of different variants of a drug is reduced, meaning that it can be used for more specific targetting. At this point, we want to highlight an observation: namely that a single characteristic of the different clusters may overlap. This renders it, like in the customer data, significantly difficult to assess the true unique property of a cluster that is embodied in the in-between instance. In that particular case of the molecular data, we were only able to achieve this through input from domain experts.

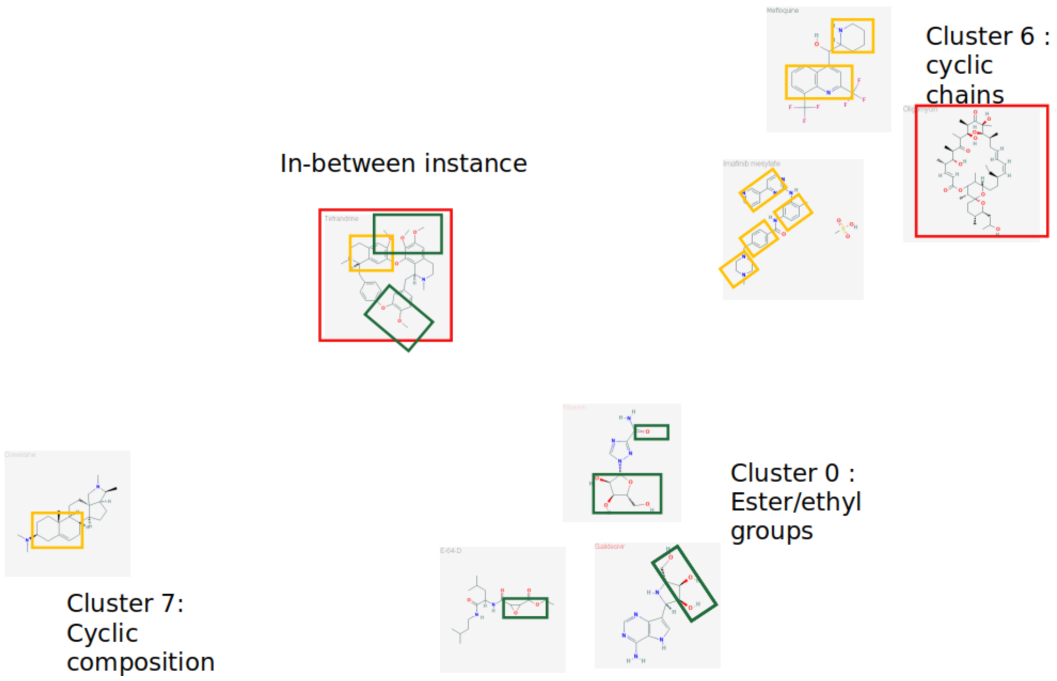

Cluster 6 : cyclic chains
In-between instance
Cluster 0 : Ester/ethyl groups
Cluster 7: Cyclic composition

Figure 4.3.2:Neighbor clusters are cluster-0, 6, and 7 for In-between instance

In another set of clusters and detected in-between instance of compounds we can see in figure 4.3.2, that the outlier *Tetrandrine* is an *In-between instance* of cluster-0, cluster-6, and cluster-7. According to domain scientists, it could be observed that cluster-0 exhibits through its **Ester/ethyl groups** [green boxes] properties that are also contained in the **In-between instance**. In cluster-7 and overlappingly in cluster-6 **cyclic composition** could be observed [orange boxes] that are also contained in the **In-between instance**. Lastly, the observed forming of a ring-like structure from a chain (cyclic chains) as seen in cluster-6 [red box] is also prominent in the discovered In-between instance. By domain scientists, it is assumed that this may act as a barrier, preventing other molecules to dock at the site.

In conclusion to the conducted experiments, there are the following insights that were gained: (1) The angle-based concept is effective in the case of multiple clusters surrounding an in-between instance, capturing also the semantics of a cluster being "behind" another cluster. (2) The choice of the angle threshold is crucial when it comes to deciding how different/opposing the surrounding clusters should be. (3) We have not investigated the impact of overlapping clusters in high-dimensional settings. This however is approached in the real-world setting, since we could observe strong overlappings in 300-dimensional data. (4) The overlapping of clusters, especially the semantic overlapping (similar texts in customer complain data, similar molecular groups etc.)

5 Conclusion and Future Prospect

In this paper, a novel approach is proposed for detecting objects that lie in between clusters, which can provide crucial information for finding relationships among the clusters

based on the distance of a point from its k-nearest neighbor. The proposed method involves the use of a hierarchical clustering technique to identify outliers, followed by the definition of a threshold to obtain outliers. The clustering method, however, can be exchanged for any other algorithm (that provides outlier detection), since the proposed in-between instance detection method is clustering model agnostic. Subsequently, the k-nearest cluster is mined using both euclidean and cosine distances to achieve more precise results. The effectiveness of the proposed approach is tested on both real-world and synthetic data-set, and the results demonstrate that the proposed method is capable of discovering in-between instances. Furthermore, experiments conducted on synthetic data-set provide a more detailed understanding of how the proposed method works including its strengths and limitations. The paper concludes with a study of the sensitivity of the parameter settings, which shows that the proposed method can be fine-tuned to achieve even better results. In summary, the paper highlights the importance of identifying objects in between clusters and suggests that the proposed method is effective in doing so, providing valuable insights into relationships among clusters.
Furthermore, this work discusses the challenges faced in detecting in-between instances in a data-set with clusters and outliers. Initially, the K-means clustering algorithm was used, but it couldn't identify outliers. The Agglomerative clustering algorithm was then applied, and a distance threshold was selected by observing the dendrogram. However, the initial implementation was flawed because the origin was set at (0,0) instead of the outlier. To rectify this, the outlier was translated to the origin, and other elements were compensated to maintain relative properties such as distances and angles. Finally, the cosine distance between an outlier and a cluster with respect to another cluster was calculated to detect whether a cluster was behind another cluster. This thesis concludes that the geometry-oriented in-between instance detection is non-trivial yet effective with respect to the in-between instance detection as well as permitting scientists to understand the in-between instance discovery in a transparent manner.

For future work, one primary target is to investigate if one can go beyond the angle-only criteria by relying on e.g. euclidean distance. More strictly speaking one can consider a scenario where four clusters are close to an in-between instance where the 5th nearest cluster is far away from the in-between instance and four clusters at the same time. While from an angular point-of-view, our 5th cluster may satisfy the criteria it is however too distant and may be questioned to be an actual 'neighboring' cluster surrounding the in-between instance. This raises, for future work, the need to incorporate straight distances i.e. euclidean distance in the detection of surrounding clusters. Additionally one may go in different directions than relying on the k-nearest-neighbor criterion i.e. an eps-based neighborhood.

We are convinced that the first endeavours made in the field of in-between instances detection may pave the path for future research in this field, recognizing the paramount importance of not only distinguishing between clusters and outliers but also of what is "between" them.

References

Bezdek, J. C., Ehrlich, R., & Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, *10*(2-3), 191–203.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). Lof: identifying density-based local outliers. In *Proceedings of the 2000 acm sigmod international conference on management of data* (pp. 93–104).

Chalapathy, R., Menon, A. K., & Chawla, S. (2018). Anomaly detection using one-class neural networks. *arXiv preprint arXiv:1802.06360*.

Chandola, V., Banerjee, A., & Kumar, V. (2007). Outlier detection: A survey. *ACM Computing Surveys*, *14*, 15.

Grover, N. (2014). A study of various fuzzy clustering algorithms. *International Journal of Engineering Research*, *3*(3), 177–181.

Hawkins, D. M. (1980). *Identification of outliers* (Vol. 11). Springer.

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., & Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their applications*, *13*(4), 18–28.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*(2), 251–257.

Islam, S. R., Eberle, W., Ghafoor, S. K., & Ahmed, M. (2021). Explainable artificial intelligence approaches: A survey. *arXiv preprint arXiv:2101.09429*.

Jaeger, S., Fulle, S., & Turk, S. (2018). Mol2vec: unsupervised machine learning approach with chemical intuition. *Journal of chemical information and modeling*, *58*(1), 27–35.

Kazempour, D., Beer, A., Oelker, M., Kröger, P., & Seidl, T. (2021). Compound segmentation via clustering on mol2vec-based embeddings. In *2021 ieee 17th international conference on escience (escience)* (pp. 60–69).

Khan, S. S., & Madden, M. G. (2010). A survey of recent trends in one class classification. In *Artificial intelligence and cognitive science: 20th irish conference, aics 2009, dublin, ireland, august 19-21, 2009, revised selected papers 20* (pp. 188–197).

Kim, H. K., Kim, H., & Cho, S. (2017). Bag-of-concepts: Comprehending document representation through clustering words in distributed representation. *Neurocomputing*, *266*, 336–352.

Kriegel, H.-P., Schubert, M., & Zimek, A. (2008). Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th acm sigkdd international conference on knowledge discovery and data mining* (pp. 444–452).

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern recognition*, *36*(2), 451–461.

Maćkiewicz, A., & Ratajczak, W. (1993). Principal components analysis (pca). *Computers & Geosciences*, *19*(3), 303–342.

Muja, M., & Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, *36*(11), 2227–2240.

Ogbechie, A., Díaz-Rozo, J., Larrañaga, P., & Bielza, C. (2017). Dynamic bayesian network-based anomaly detection for in-process visual inspection of laser surface heat treatment. In *Machine learning for cyber physical systems: Selected papers from the international conference ml4cps 2016* (pp. 17–24).

O'Shea, T. J., Clancy, T. C., & McGwier, R. W. (2016). Recurrent neural radio anomaly detection. *arXiv preprint arXiv:1611.00301*.

Ramaswamy, S., Rastogi, R., & Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 acm sigmod international conference on management of data* (pp. 427–438).

Rashidi, L., Hashemi, S., & Hamzeh, A. (2011). Anomaly detection in categorical datasets using bayesian networks. In *Artificial intelligence and computational intelligence: Third international conference, aici 2011, taiyuan, china, september 24-25, 2011, proceedings, part ii 3* (pp. 610–619).

Rokach, L., & Maimon, O. (2005). Decision trees. *Data mining and knowledge discovery handbook*, 165–192.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., . . . Kloft, M. (2018). Deep one-class classification. In *International conference on machine learning* (pp. 4393–4402).

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, *42*(3), 1–21.

Sibson, R. (1973). Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, *16*(1), 30–34.

Singh, K., & Upadhyaya, S. (2012). Outlier detection: applications and techniques. *International Journal of Computer Science Issues (IJCSI)*, *9*(1), 307.

Struyf, A., Hubert, M., & Rousseeuw, P. (1997). Clustering in an object-oriented environment. *Journal of Statistical Software*, *1*(4), 1–30. DOI: 10.18637/jss.v001.i04

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(11).

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, *11*(12).