University of Zurich^UZH

Thesis June 13, 2021

Continuous Semi-Supervised Binary Classification of Data Streams

Patrick Muntwyler
of Wohlen AG, Switzerland

Student-ID: 14-711-337
patrick.muntwyler@uzh.ch

Advisor: **D. Dell'Aglio, PhD**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

I want to express my deepest thanks to Daniele Dell'Aglio, PhD who supervised this thesis. I am grateful for his valuable inputs, constructive criticism and guidance. I would like to thank Professor Abraham Bernstein, PhD for making this research project possible. Further on, I want to thank my family and friends who supported me during this time. Last but not least, I would like to thank Christoph Weber for proofreading this thesis.

Zusammenfassung

Die Anzahl der Datenströme wächst täglich und damit auch ihre Bedeutung in unserem täglichen Leben. Es ist wichtig, Datenströme automatisch analysieren zu können, zum Beispiel um verdächtige Aktivitäten in einem System zu erkennen oder interessante Datenpunkte zu filtern. Viele Systeme setzen heute auf überwachte Ansätze. Diese haben jedoch den Nachteil, dass sie sich nicht an neue Trends in den Datenströmen anpassen können. Hierfür werden teilüberwachte Ansätze benötigt. Dieser Bereich ist jedoch weniger gut erforscht. Wir entwickeln daher SSDenStream. SSDenStream basiert auf DenStream, einem unüberwachten dichte-basierten Stream-Clustering-Algorithmus, und ist in der Lage, Online-Klassifikation durchzuführen. Wir geben einen Überblick über dichte-basiertes Stream-Clustering und teilüberwachte Erweiterungen davon. Wir führen mehrere Experimente mit synthetischen und realen Datensätzen durch, um die Funktionalität von SSDenStream zu beweisen. Die Experimente zeigen, dass SSDenStream in der Lage ist, mit überlappenden Clustern umzugehen und bei realen Daten gute Ergebnisse erzielt.

Abstract

The number of data streams is growing every day, and so is their importance in our daily lives. It is important to be able to analyze data streams automatically, for example to find suspicious activities in a system or to filter interesting data points. Many systems today rely on supervised approaches. However, these have the disadvantage that they cannot adapt to new trends in the data streams. Semi-supervised stream approaches are needed for this. However, this area is not yet well explored. We therefore develop SSDenStream. SSDenStream is based on DenStream, an unsupervised density-based stream clustering algorithm, and is able to perform online classification. We give an overview of density-based stream clustering and semi-supervised extensions of it. We perform several experiments on synthetic and real-world data sets to prove the functionality of SSDenStream. The experiments show that SSDenStream is able to handle overlapping clusters and performs well on real-world data.

Contents

Abbreviations

1-NN	K-Nearest Neighbors | $k = 1$.
2-NN	K-Nearest Neighbors | $k = 2$.
3-NN	K-Nearest Neighbors | $k = 3$.
5-NN	K-Nearest Neighbors | $k = 5$.
CWD	Class Weight Dictionary.
k-NN	K-Nearest Neighbors.
MST	Minimum Spanning Tree.
ROC AUC	Receiver Operating Characteristic Area Under Curve.

1

Introduction

A data stream is a continuous flow of data which can be of high volume and potentially infinite. Data streams are becoming increasingly important, whether in everyday life, in industry or in research. There are countless examples: Netflix, video conferences, results from sensors, for example at CERN or from telescopes, monitoring of financial assets or the monitoring of collaborative knowledge graphs (vandalism detection).

Wikidata[1] is a well known example for a knowledge graph. Multiple studies exist that address the topic of Wikidata vandalism detection, most notably [Sarabadani et al., 2017] and [Heindorf et al., 2019]. This topic was also covered at the WSDM Cup 2017[2]. All solutions build on a supervised learning algorithm.

There exist other fields of activity where algorithms are required to filter large data streams for interesting samples which are then investigated manually. One example is astronomy where telescopes gather huge amount of data and algorithms determine candidates for certain phenomena. [Lyon et al., 2016] presents a semi-supervised approach for finding pulsar candidates (a certain type of star) in data streams.

For such tasks, a semi-supervised learning approach has some advantages compared to supervised approaches: Training a supervised model normally requires a large amount of labeled data which is expensive to create. Once it is trained, it cannot adapt to changes in the data, i.e. to drifting data. Semi-supervised models are able to learn continually from only a few labeled data samples over time and thus adapting to evolving data streams. In both mentioned fields of activities new labeled data is available over time as experts review suspicious data samples and evaluate them.

We want to tackle such problems by using a density-based stream clustering algorithm. Density-based clustering algorithms have certain advantages, e.g. they can process data of various shape and can handle noise while no previous knowledge about the number of clusters is required. We use DenStream as base algorithm which was proposed by [Cao et al., 2006]. It is an unsupervised algorithm and is able to work in a streaming environment. There exist many extensions of DenStream for different use cases. But there exists only little work on semi-supervised extensions. CDenStream applies instance-based constraints to define if certain samples belong to the same cluster [Ruiz et al., 2009]. It lacks a mechanism to forget old constraints, as data samples lose importance

[1]https://www.wikidata.org/wiki/Wikidata:Main_Page (18.05.2021)
[2]https://www.wsdm-cup-2017.org/vandalism-detection.html (05.05.2021)

over time. SemiStream tackles these drawbacks by applying a cost function based on the constraint when searching for the best cluster a sample can be assigned to [Halkidi et al., 2012]. But neither work directly with labels. The instance-based constraints must first be derived from the labels. SSE-Stream, an extension of a different stream clustering algorithm called SED-Stream, is an approach that works with labels instead of instance-based constraints [Treechalong et al., 2015]. But all of these algorithms focus on building clusters in an offline step after accumulating data in the online phase. None of them can do online classification.

To tackle challenges like vandalism detection or finding pulsar candidates with Den-Stream we need a semi-supervised extension that is able to do online classification. We develop such an extension and call the resulting algorithm SSDenStream. We introduce a mechanism that stores the class labels of the labeled data samples. Based on the collected class labels, unlabeled data points are classified. We apply a decaying function to the class labels so that old class labels have less influence than new ones. This allows the algorithm to learn new trends. We also present different approaches to propagate class label information into previously unallocated parts of the vector space. We perform an exploratory analysis on the hyper parameters and show their impact on the classification quality. We show that proper hyper parameter tuning is essential for SSDenStream. We conduct various experiments with synthetic and real-world data sets and compare them to state-of-the-art algorithms. Our experiments show that SSDenStream can handle overlapping clusters and achieves good results on real-world data.

Chapter 2 presents a motivating example for this thesis: vandalism detection. Chapter 3 shows related work and introduces the necessary background knowledge. Chapter 4 explains the functionality of SSDenStream. Chapter 5 explains the experiments and their results. Chapter 6 presents the limitations and future work. Chapter 7 draws the conclusion about this thesis.

2

Motivating Example - Vandalism Detection

Knowledge graphs play a major role in our daily lives. Google maintains a knowledge graph which is used to improve search results[1]. BBC connects news items to knowledge graph entity identifiers to be able to query their content meaningfully and find related content[2]. Wikidata is a collaborative system that is used by countless applications and people every day. Wikidata may be edited by anyone. This has the great advantage that anyone can contribute their knowledge to the knowledge graph. However, there are also people who deliberately or unintentionally enter incorrect data. To maintain high data quality of Wikidata or other collaborative knowledge graphs, changes must be quality checked. This requires automatic systems that report suspicious changes.

This is the motivation of this thesis. We first had to evaluate state-of-the-art research to find out where there is room for improvement. This led to the realization that semi-supervised approaches are not being pursued for this use case. Therefore, we developed SSDenStream to counter vandalism with a new approach. Below we present recent literature in the field of vandalism detection which can serve as a basis for further research on this topic.

2.1 Wikidata Vandalism Detection

When conducting research on a certain topic it is beneficial to have a public data corpus which people can use to engineer features, train algorithms and compare their results on. [Heindorf et al., 2015] presents the creation of a vandalism corpus for Wikidata revisions. The corpus comprises all Wikidata revisions which were manually done by users since the launch of Wikidata in October 2012 until November 2014. Altogether the corpus includes about 24 million samples of which all are labeled as vandalism or not. The labeling is done by a machine as labeling this amount of samples manually is infeasible.

There exist two types of operations on Wikidata to correct wrong revisions. The *rollback* operation can only be used by administrators and privileged users and "should

[1]https://www.blog.google/products/search/introducing-knowledge-graph-things-not/ (05.05.2021)
[2]https://www.bbc.co.uk/blogs/internet/entries/d6d2e984-1acd-30dd-a75a-afe9f12f5b46 (05.05.2021)

only be used to revert vandalism and test edits"[3]. The *undo* and *restore* functionalities can also be used to revert revisions but they are available to all non-blocked users. [Heindorf et al., 2015] shows that revisions which were rolled back can be seen as vandalism where as undone or restored revisions were often no vandalism attempts. In the end, the automated labeling mechanism only uses the *rollback* operation to find vandalism attempts and leaves the *undo* and *restore* revisions to be examined further in future work.

This vandalism corpus is used by [Heindorf et al., 2016] to train a supervised learning algorithm which categorizes Wikidata revisions as vandalism or non-vandalism attempts. [Heindorf et al., 2016] describes all 47 features that were used for the final classifier. The features are divided into content and context features. Content features are created from the content of revision at either character, word, sentence or statement level. Examples are *digitRatio* (ratio of digits to all characters) or $longestWord$ (length of longest word in the revision). Contextual features describe either the user (e.g. country of a user), the revised item (e.g. the frequency of that item to be part of a revision) or the revision itself (e.g. the type of a revision). The features do not require high computational power and thus are suitable for online classification.

In 2017, there was a conference called WSDM Cup 2017[4]. One of the available tasks was vandalism detection on Wikidata revisions[5]. The provided training corpus follows the same creation logic as described by [Heindorf et al., 2015] but is a more up-to-date version containing data from October 2012 until June 2016 which makes around 65 million revisions. The intention of the task chair and committee was to transfer the gained knowledge to Wikimedia Germany by inviting the winning team for a couple of days. [Crescenzi et al., 2017] presents the features and algorithm used by the winning team called Buffaloberry. The corresponding repository is available on GitHub[6]. [Heindorf et al., 2017] gives an overview over the data, all the features and the algorithms used by the participating teams.

[Sarabadani et al., 2017] criticises the corpus presented by [Heindorf et al., 2015] because it only considers revisions reverted by the *rollback* operation as vandalism attempts. [Sarabadani et al., 2017] presents their own corpus which contains 500'000 samples and is created by a different strategy. According to the authors only 63% of the vandalism attempts are reverted using the *rollback* operation. The remaining 37% of the vandalism attempts are reverted using the *restore* or other operations. This means that the corpus of [Heindorf et al., 2015] overlooks a big part of the vandalism attempts. The model trained on the new corpus performs well, but is strongly influenced by the features that reflect the user who performed the edit.

[3]https://www.wikidata.org/wiki/Wikidata:Rollbackers (03.05.2021)
[4]https://www.wsdm-cup-2017.org/ (03.05.2021)
[5]https://www.wsdm-cup-2017.org/vandalism-detection.html (03.05.2021)
[6]https://github.com/wsdm-cup-2017/buffaloberry (03.05.2021)

[Nishioka and Scherp, 2018] describes a different approach. The data set contains the Wikidata changes between April 2014 and August 2016. The incorrect changes are identified using a simple heuristic: every change that is reverted up to four weeks after publication is categorized as vandalism. Compared to the other two corpora this approach results in a higher rate of incorrect changes. The data set consists of subject-predicate-object triples with a timestamp and a flag. The flag defines if the change was a deletion or an addition of information. The features created from this data set solely base on graph metrics such as node in-degree, node out-degree and node age. No user metrics are used at all.

[Heindorf et al., 2019] states that the vandalism detection algorithms used at Wikidata heavily rely on user metadata features. These features induce user bias and thus encourage discrimination against new users and anonymous users. The authors present two algorithms, called FAIR-E and FAIR-S, which evaluate Wikidata revisions based on their content without taking into account the reputation of the users. FAIR-E is based on graph embedding using the subject-predicate-object triples mirroring the Wikidata edits. FAIR-S uses already existing features from state-of-the-art Wikidata vandalism detection frameworks and a few newly created features leaving out all features that are not linked to the content of an edit.

2.2 Vandalism Detection for Other Knowledge Bases

Vandalism detection is an important topic for other knowledge bases as well. [Tan et al., 2014] presents work about vandalism detection on Freebase[7]. The created features can be split into three categories: user contribution history, triple features and user contribution expertise. Two out of three categories rely only on the information about the users who edit the knowledge base. Therefore this work has the disadvantage of discriminating new users.

[Wienand and Paulheim, 2014] presents an approach that concentrates on numerical data in DBpedia[8]. The authors create subject-predicate-object (SPO) triple data sets for different predicates, for example *dbpedia-owl:populationTotal*, *dbpedia-owl:height* or *dbpedia-owl:elevation*. In other words, the first data set contains all SPO triples with *dbpedia-owl:populationTotal* as predicate. In a second step the data sets are split by the type of the subject. A resulting sub data set contains for example all SPO triples representing the population of cities. Different unsupervised outlier algorithms are applied to these sub data sets to find out if the information is correct or not. This approach has the advantage that the learned algorithms are not biased against the user who created a statement as it is solely based on the statement's content. One disadvantage of this work is that the applied algorithms cannot be used with data streams. They are not

[7]https://en.wikipedia.org/wiki/Freebase_(database) (04.05.2021)
[8]https://www.dbpedia.org/ (03.05.2021)

able to do online computation.

Wikidata can also be used as external information source to verify data in other databases. In [Olivieri et al., 2017] SPO triples in the form of *actor-act-movie* from the movie database LinkedMDB[9] are checked for truthfulness. This is achieved by querying Wikidata. There are six queries defined which should find out if the actor is an actor, the movie is a name of a movie and if the actor was part of that movie. This principle can be transferred to other databases to verify if their content is true.

OpenStreetMap[10] is a map project that relies on volunteers like Wikidata. As a consequence there is vandalism. [Truong et al., 2018] describes how geometric data about buildings can be checked for vandalism. The features are constructed to mirror the geometric data and to work with the unsupervised clustering algorithm called DENCLUE. [Neis et al., 2012] is an older work which concentrates on vandalism detection in OpenStreetMap. The applied features only rely on user data and thus is biased against certain user groups.

2.3 Wrap-Up

The literature research shows that a wide variety of approaches are being pursued to counter the vandalism problem. However, all approaches have their limits. We see great opportunities for improvement in three different directions.

(1) Corpus creation: [Sarabadani et al., 2017] criticizes the mechanism used to create the corpus presented by [Heindorf et al., 2015] which is also used at WSDM Cup 2017. Many vandalism cases are not marked as such in this corpus. [Sarabadani et al., 2017] presents its own approach, but it relies heavily on users' metadata which induces bias against certain user groups. [Nishioka and Scherp, 2018] presents a different approach. It would be of great use to compare the three different approaches and take the best parts of each to generate a new corpus. It is important that a new corpus is made available from time to time, as user behavior can also change over time. However, none of the work presented includes data that is more recent than 2016.

(2) Using SPO triples: The SPO triples in a knowledge graph can be divided into two groups. (i) A triple can connect two entities with a predicate and thus create a semantic link between the two entities, for example France - capital - Paris. (ii) The object is a literal and thus an attribute is assigned to an entity, for example France - population - 66,628,000. [Heindorf et al., 2019] focuses on group (i). [Nishioka and Scherp, 2018] examines both groups, but cannot present good results on group (ii). We see potential for improvement here. A solution would be based purely on content and thus not contain user bias. Perhaps an approach similar to [Wienand and Paulheim, 2014] can be successfully applied to Wikidata.

[9] Original URL is not valid anymore (04.05.2021)
[10] https://www.openstreetmap.org/ (04.05.2021)

(3) Semi-supervised learning: None of the works studied use a semi-supervised approach. Some use unsupervised algorithms, but the majority use supervised algorithms. These algorithms cannot adapt to trends in the data and therefore have to be retrained from time to time. We want to contribute to this problem with our work. We want to develop a semi-supervised algorithm that can be used in future work to detect vandalism, among other things.

3

Background and Related Work

3.1 Clustering and Stream Mining

Clustering and classification are similar tasks, differing mainly in whether labeled samples are available or not. Classification is a supervised learning approach in which samples must be assigned to classes. The classes are known in advance. Clustering, on the other hand, is unsupervised. The goal is to divide the data points into "natural" groups. There are many different definitions of the term *cluster*. However, they all have in common that a cluster should contain similar samples and separate them from dissimilar samples [Xu and Wunsch, 2008].

There exist different approaches how clustering algorithms compute clusters. Five different groups of clustering algorithms are defined by [Mousavi et al., 2015]:

- **Hierarchical methods:** Usually in a top down approach, where in the beginning all data points belong to the same cluster. Then the cluster is divided iteratively into several sub-clusters until an optimal structure is found.

- **Partitioning methods:** Require a predefined number of cluster centers and their position in the data space. Then all data samples are assigned to the closest cluster center according to some distance function.

- **Grid-based methods:** They place a grid over the vector space and thus divide it into cells. The density of a cell depends on the number of points in it. Clustering is performed with the cells instead with the data points.

- **Density-based methods:** Work with a density threshold (i.e. number of data points in a certain radius) to define which areas in the vector space can be seen as clusters.

- **Model-based methods:** Statistical approach that runs a hypothesized model (a probability distribution) for every cluster. Goal is to find a model that fits the data points in a cluster as good as possible and thus decides which points belong to which cluster.

In this thesis we work with a density-based clustering algorithm. Therefore, a suitable definition for the term *cluster* is *a region in the data space with a high density of data samples separated by low density areas.*

There is also the distinction between traditional and stream clustering. In a traditional setup a data set is static, and therefore can be processed multiple times. The amount of data does not exceed the storage capacities and thus can be stored. In stream clustering, algorithms are able to work with data streams, i.e. perform stream mining. Data streams are (i) continuous and potentially infinite; (ii) they contain evolving data and response is required in real-time; (iii) data is available only once for processing; (iv) due to huge amount of data only part of it can be stored so the important part of the data must be found during processing; and, (v) data streams can be multidimensional which requires advanced algorithms for the mining task [Mousavi et al., 2015].

According to [Mousavi et al., 2015], density-based clustering algorithms can handle data of various shape, are able to handle noise, and need to process the data only once (no iterative mechanisms). These properties make density-based clustering a good choice for a streaming environment.

3.2 DenStream

Data streams are potentially infinite, they can include huge amounts of data and can evolve over time. Due to these properties, traditional clustering algorithms cannot handle data streams. [Cao et al., 2006] introduces a density-based clustering algorithm for data streams, called DenStream, that can tackle these challenges. As data streams are potentially infinite, the algorithm cannot store all data samples. At some point the computer will run out of storage. A data structure called micro-cluster summarizes the incoming data samples. There are more micro-clusters than final clusters but still much fewer micro-clusters than data samples. DenStream distinguishes between two phases: the *online* phase and the *offline* phase. During the online phase, the algorithm maintains the micro-clusters to summarize the incoming data. At some point the user requires a final clustering of the current situation. This is done in the offline phase, where DenStream applies a variant of DBSCAN to the micro-clusters to create clusters. DBSCAN was introduced by [Ester et al., 1996].

DenStream works with the damped window model. An old data sample will have less influence on the clustering than new data samples. An exponential decaying function decreases the weight of a data sample over time.

Definition 1. (decay factor λ [Cao et al., 2006]): The exponential decaying function is defined as $f(t) = 2^{-\lambda * t}$, where $\lambda > 0$. The higher λ, the lower the influence of old data samples.

We work with the DenStream implementation of [Weber, 2019] which uses a slightly different definition of micro-clusters compared to the work of [Cao et al., 2006].

Definition 2. (micro-cluster [Weber, 2019]): A micro-cluster is defined by a weight w, mean $\bar{x}$ (which presents the cluster's center) and a variance σ^2.

The following three equations define how a micro-cluster's properties are updated when a new samples s is assigned to it at timestamp t. Timestamp t_{last} denotes the timestamp when the last sample was added. Sample s has a value s_x and a weight s_w.

$$w_t = w_{t_{last}} * 2^{-\lambda*(t-t_{last})} + s_w \tag{3.1}$$

$$\bar{x}_t = \bar{x}_{t_{last}} + \frac{s_w * (s_x - \bar{x}_{t_{last}})}{w_t} \tag{3.2}$$

$$\sigma_t^2 = \sigma_{t_{last}}^2 * \frac{w_t - s_w}{w_{t_{last}}} + s_w * (s_x - \bar{x}_t) * (s_x - \bar{x}_{t_{last}}) \tag{3.3}$$

Figure 3.1 illustrates the process of adding samples to a micro-cluster. Table 3.1 contains the corresponding key figures. In the example, λ reduces the influence of past samples by half every time unit. There is one time unit between each new sample and each sample owns a weight s_w of 1. Illustration (i) shows the situation when the first sample $S1$ is added to the micro-cluster. $S1$ has the coordinates [0, 0] and is marked as blue dot. As this is the first sample of this micro-cluster, the micro-cluster's center has the same coordinates.

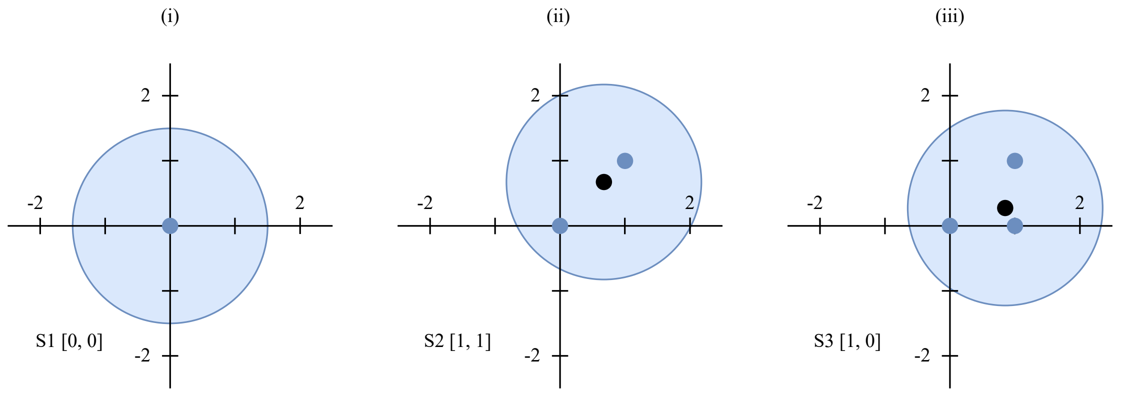

Figure 3.1: Illustration of the process of adding samples to a micro-cluster. Blue dots are samples and the black dot denotes the micro-cluster's center

	s_x	s_w	t	λ	w	$\bar{x}$	σ^2
(i)	[0, 0]	1	1	1	1	[0, 0]	0
(ii)	[1, 1]	1	2	1	1.5	[0.67, 0.67]	[0.33, 0.33]
(iii)	[1, 0]	1	3	1	1.75	[0.86, 0.29]	[0.21, 0.36]

Table 3.1: Key figures of the process illustrated in Figure 3.1

In (ii) sample $S2$ appears and is located at [1, 1]. The micro-cluster's weight w does not double now. The decaying function reduces the existing weight and adds the new additional weight. Also, the center of the micro-cluster is not moved to the middle between the two samples, but closer to $S2$. This is because $S1$ has already lost influence. In (iii) sample $S3$ appears at [1, 0]. Current weight w is halved and the weight s_w of $S3$ is added. The center $\bar{x}$ moves into direction of $S3$. Due to the decaying function, the past is gradually forgotten and the micro-cluster follows the new samples. Variation σ^2 is required to describe the scattering of the samples.

There is a parameter required which limits the size of a micro-cluster. Otherwise, all samples can be assigned to the same micro-cluster.

Definition 3. (maximum micro-cluster radius ϵ [Weber, 2019]): A micro-cluster has an upper threshold for its radius r, such that $r \leq \epsilon$. It cannot absorb data samples where the distance from the cluster's center to the data sample is bigger than ϵ.

[Cao et al., 2006] defines different types of micro-clusters depending on their weights.

Definition 4. (core-micro-cluster [Cao et al., 2006]): A core-micro-cluster (c-micro-cluster) is a micro-cluster which has a weight higher than some threshold μ, simultaneously not violating the maximum radius restriction, i.e., $w \geq \mu$ and $r \leq \epsilon$. A c-micro-cluster is what can be referred to as a dense micro-cluster.

[Putina et al., 2018] shows that μ can be automatically computed only depending on the decaying factor λ: $\mu = \frac{1}{1-2^{-\lambda}}$

As the data streams evolve, outliers can become cluster members and vice versa. C-micro-clusters are only formed gradually. Therefore [Cao et al., 2006] defines the *potential c-micro-cluster* (p-micro-cluster) and *outlier-micro-cluster* (o-micro-cluster). O-micro-clusters are used to define regions where outlier points lie. The only difference between the two types of micro-clusters is their weight.

Definition 5. (p-micro-cluster and o-micro-cluster [Cao et al., 2006]): If the weight w of a micro-cluster is lower than a certain threshold, then it is defined as a o-micro-cluster, i.e., $w < \beta * \mu$. Otherwise the micro-cluster is classified as p-micro-cluster. β is a hyper parameter of DenStream.

The offline phase of DenStream uses the p-micro-clusters as input for DBSCAN to create final clusters. [Weber, 2019] applies DenStream for outlier detection instead of creating final clusters. Therefore, the offline phase is not implemented and we do not describe it further at this point. If a new data sample cannot be assigned to a p-micro-cluster then it is labeled as outlier. In a certain time interval the algorithm checks all micro-clusters if their weight is higher or below the weight threshold $\beta * \mu$. If a o-micro-cluster becomes a p-micro-cluster, we can say that those points were early movers, i.e. they set a trend. It is also possible that p-micro-clusters become o-micro-clusters if that region is against the current trend of the data samples.

[Weber, 2019] introduces DenStream*, an extension of Denstream. It contains two additional hyper parameters which are called drift-distance-influence δ and drift-weight-influence ω. As mentioned, some outliers are early movers and will not be outliers in the future and some cluster points will become outliers. This is due to the evolving

```
def DenStream*($\epsilon$, $\beta$, $\mu$, $\lambda$, $\delta$, $\omega$):
    for $s$ in data stream:
        # try to merge sample into nearest p-micro-cluster $c_p$
        $c_p$ = get_nearest_p_micro_cluster($s$)
        # $s_w$ is sample weight
        $c_p^+$ = merge($c_p$, $s$, $\lambda$, $s_w$ * (1 + $\delta$ * $\omega$))
        if radius($c_p^+$) $\leq$ $\epsilon$ * (1 + $\delta$ * $\omega$):
            $c_p = c_p^+$
        else:
            # try merge sample into nearest o-micro-cluster $c_o$
            $c_o$ = get_nearest_or_new_o_micro_cluster($s$)
            $c_o^+$ = merge($c_o$, $s$, $\lambda$, $s_w$ * (1 + $\delta$ * $\omega$))
            if radius($c_o^+$) $\leq$ $\epsilon$ * (1 + $\delta$ * $\omega$):
                # check if big enough for promotion
                if weight($c_o^+$) > $\beta * \mu$:
                    p_micro_clusters.append($c_o^+$)
                    o_micro_clusters.remove($c_o$)
                else:
                    $c_o = c_o^+$
                    write_real_time_outlier($s$)

            else:
                # new o-micro-cluster
                o_micro_clusters.append($c_o^+$)
                write_real_time_outlier($s$)
```

Algorithm 3.1: DenStream* Implementation

nature of data streams. The two parameters δ and ω include the trend of a stream into the allocation of data points to micro-clusters. The drift-distance-influence parameter adapts the distance a point can have from a cluster's center. If the data point lies in direction of the trend, then the distance from the cluster center can be higher as ϵ. The drift-weight-influence adapts the weight a data sample adds to a micro-cluster. If the data point lies in the direction of the trend, its weight is higher and vice versa.

Algorithm 3.1 shows the pseudo-code of DenStream*. When a new sample s arrives, the closest p-micro-cluster c_p is retrieved. Sample s is merged into c_p and if the updated radius does not exceed maximum radius threshold ϵ, the algorithm ends here. If the radius is too large then the closest o-micro-cluster c_o has to be retrieved. If sample s can be merged into c_o, there has to be evaluated if c_o has enough weight now to be categorized as p-micro-cluster. If yes, c_o is now a p-micro-cluster, else sample s is marked as outlier. If s cannot be merged into c_o, a new micro-cluster is generated, which is automatically categorized as o-micro-cluster and thus s is defined as outlier.

The two hyper parameters β and λ are in the usual case very incomprehensible numbers. To make it easier to set them correctly, they are computed from minimum cluster size ζ and half-life time $t_{1/2}$ in the following way:

$$\lambda = \frac{-log_2(0.5)}{t_{1/2}} \tag{3.4}$$

$$\beta = \zeta * (1 - 2^{-\lambda}) \tag{3.5}$$

The following example illustrates the simplification through these pre-processing steps. We want each p-micro-cluster to have a minimum cluster size ζ of 5 and that the weights lose 50% of their influence after 100 time units. In the example we calculate λ, β and μ and show the outcome of weight threshold $\beta * \mu$ (see Line 15 of Algorithm 3.1 or see Definition 5):

$$\zeta = 5;\ t_{1/2} = 100 \tag{3.6}$$

$$\lambda = \frac{-log_2(0.5)}{t_{1/2}} = \frac{-log_2(0.5)}{100} = 0.01 \tag{3.7}$$

$$\beta = \zeta * (1 - 2^{-\lambda}) = 5 * (1 - 2^{-0.01}) = 0.0345 \tag{3.8}$$

$$\mu = \frac{1}{1 - 2^{-\lambda}} = \frac{1}{1 - 2^{-0.01}} = 144.7701 \tag{3.9}$$

$$\beta * \mu = 0.0345 * 144.7701 = 5 \tag{3.10}$$

3.3 Further Work Based on DenStream

DenStream is a popular density-based stream clustering algorithm and thus there exist many different extensions. DenStream is only able to process numerical features. But categorical features can be as critical as numerical features to determine the correct cluster structure. [Lin and Lin, 2009] proposes an extension of the DenStream algorithm called HDenStream that can handle categorical data. HDenStream uses different distance functions for categorical and numerical features which are summed up to determine the final distance of a sample to a cluster. The distance between two data points for a categorical dimension is 0 if they both own the same category label or 1 otherwise. A micro-cluster is able to sum up samples with different categories. It maintains a two-dimensional array that stores the frequency for each valid label of a categorical feature. The distance of a sample to a micro-cluster for a categorical feature is the sum of the weights for all category labels that are different to the samples label. [Chen and

He, 2016] introduces different distance functions for numerical and categorical features depending on their proportion in the data set.

DenStream is an unsupervised clustering algorithm. But there may exist data sets which have partially labeled samples. [Ruiz et al., 2009] introduces C-DenStream, a semi-supervised approach of DenStream. C-DenStream can handle samples with instance-based constraints. There exist must-link and cannot-link constraints, i.e., the constraint describes if two samples belong to the same cluster or not. The algorithm takes the constraints into account when adding samples to clusters. It does not allow for two samples with a cannot-link constraint to be in the same cluster. [Halkidi et al., 2012] criticises C-DenStream for creating many small micro-clusters, due to satisfying all constraints. Their approach, called SemiStream, uses instance-based constraints as well. But SemiStream has not to satisfy all constraints; it rather uses them as penalty when computing the distance from sample to cluster.

[Treechalong et al., 2015] proposes a semi-supervised stream clustering algorithm, called SSE-Stream, which works with data labels instead of constraints. They argue that using instance-based constraints in a streaming environment has some drawbacks, e.g. not all constraints can be evaluated at the same time. Thus it is hard to find an optimal solution. The algorithm tries to build micro-clusters so that a micro-cluster does not contain different labels. It is using split and merge operations to keep the micro-clusters as pure as possible. If there exist impure micro-clusters when the offline clustering is done, the majority class label of a micro-cluster is used.

As mentioned in Section 3.2 DenStream uses DBSCAN for creating the final clusters. DBSCAN has one big disadvantage: it cannot find clusters of different densities. Based on the hyper parameters, DBSCAN computes a static density threshold that defines how many data points must exist in a certain radius to form a cluster. [Campello et al., 2013] introduces HDBSCAN, a hierarchical density-based clustering algorithm, that overcomes that limitation.

[Hassani, 2015] presents the combination of DenStream and HDBSCAN, resulting in a hierarchical density-based stream clustering algorithm called HASTREAM. In the online phase, the streaming data is summarized through micro-clusters, with similar definitions as specified by [Cao et al., 2006]. During the offline phase an adapted version of HDBSCAN is used, which works with micro-clusters instead of single data points. HDBSCAN requires the computation of a Minimum Spanning Tree with data points or micro-clusters respectively as nodes. A Minimum Spanning Tree is a weighted graph that connects all its nodes such that the sum of all used edges is minimal. Computing a MST is an expensive task. Thus [Hassani et al., 2016] introduces I-HASTREAM, an improved version of HASTREAM. The focus of I-HASTREAM lies on the computation of the Minimum Spanning Tree. Instead of computing the whole graph each time the offline phase is triggered, I-HASTREAM can recognize which micro-clusters have changed. Only the affected regions of the graph are recomputed.

4

SSDenStream: An Approach for Semi-Supervised Binary Stream Classification

SSDenStream is a semi-supervised extension of the DenStream algorithm. It is based on the DenStream implementation of [Weber, 2019] called DenStream*. [Weber, 2019] uses the implementation of [Putina et al., 2018]. [Putina et al., 2018] introduces an extended version of DenStream which detects outliers. The implementation of [Weber, 2019] extends the outlier detection with two additional hyper parameters δ and ω, which improve the algorithm by following drifting data. For more information about the hyper parameters see Section 3.2.

SSDenStream works with partially labeled data streams. Class labels from labeled data samples are stored in the corresponding micro-clusters. When a new micro-cluster is created, the class label information of the neighboring micro-clusters is used to initialize class label information for the new micro-cluster. An unlabeled data sample is classified by the class label information of the micro-cluster it is assigned to.

Algorithm 4.1 shows DenStream* with the extensions that result in SSDenStream. The black pseudo-code represents DenStream* and the red lines are the extensions for SSDenStream. The added mechanisms are explained in more details in the following sections. Section 4.1 explains how the available information of the labeled data is stored (Line 36 in Algorithm 4.1). Section 4.2 describes what happens when a new micro-cluster is created which does not own any class information (Lines 30 & 31 in Algorithm 4.1). Section 4.3 elucidates how unlabeled samples are classified (Line 37 in Algorithm 4.1). Section 4.4 explains the advantages and disadvantages between different approaches for finding the neighbors of a micro-cluster (Line 30 in Algorithm 4.1) in more details.

4.1 Storing Labels

In a semi-supervised setup, some data samples are labeled and some are not. SSDenStream stores the available labels and uses them to classify unlabeled samples. It uses a dictionary where the class label is the key and the value is the weight of the corresponding class. We call it the Class Weight Dictionary (CWD). Every micro-cluster owns a

```
def SSDenStream($\epsilon$, $\beta$, $\mu$, $\lambda$, $\delta$, $\omega$):
    for $s$ in data stream:
        # read the sample label $s_l$. Can be None
        $s_l$ = s.label
        # variable to access final micro-cluster mc of s
        $mc$ = None
        # try to merge sample into nearest p-micro-cluster $c_p$
        $c_p$ = get_nearest_p_micro_cluster($s$)
        # $s_w$ is sample weight
        $c_p^+$ = merge($c_p$, $s$, $\lambda$, $s_w$ * (1 + $\delta$ * $\omega$))
        if radius($c_p^+$) $\leq$ $\epsilon$ * (1 + $\delta$ * $\omega$):
            $c_p = c_p^+$
            $mc = c_p^+$
        else:
            # try merge sample into nearest o-micro-cluster $c_o$
            $c_o$ = get_nearest_or_new_o_micro_cluster($s$)
            $c_o^+$ = merge($c_o$, $s$, $\lambda$, $s_w$ * (1 + $\delta$ * $\omega$))
            if radius($c_o^+$) $\leq$ $\epsilon$ * (1 + $\delta$ * $\omega$):
                # check if big enough for promotion
                if weight($c_o^+$) > $\beta * \mu$:
                    p_micro_clusters.append($c_o^+$)
                    o_micro_clusters.remove($c_o$)
                else:
                    $c_o = c_o^+$
                    write_real_time_outlier($s$)

            else:
                # new o-micro-cluster
                if $s_l$ is None:
                    $n$ = find_neighbors($c_o^+$)
                    propagate_labels($c_o^+, n$)
                o_micro_clusters.append($c_o^+$)
                write_real_time_outlier($s$)
            $mc = c_o^+$
        if $s_l$ is not None:
            $mc$.update_class_weight_dictionary($s_l$)
        classify_sample($mc$)
```

Algorithm 4.1: SSDenStream Implementation

CWD. Whenever a labeled data sample is assigned to a micro-cluster, the CWD stores this information and accumulates the labels for each class. Thus, we need an extended definition for micro-clusters.

Definition 6. (micro-cluster): A micro-cluster is defined by a weight w, mean $\bar{x}$ (which presents the cluster's center), a variance σ^2 and a Class Weight Dictionary *cwd*.
Sample s is a new sample that is added to the micro-cluster at timestamp t. Sample s owns a value s_x, a weight s_w and a class label s_{cw}. Timestamp t_{last} denotes the point of time when the last sample was added to this micro-cluster.

$$w_t = w_{t_{last}} * 2^{-\lambda*(t-t_{last})} + s_w \tag{4.1}$$

$$\bar{x}_t = \bar{x}_{t_{last}} + \frac{s_w * (s_x - \bar{x}_{t_{last}})}{w_t} \tag{4.2}$$

$$\sigma_t^2 = \sigma_{t_{last}}^2 * \frac{w_t - s_w}{w_{t_{last}}} + s_w * (s_x - \bar{x}_t) * (s_x - \bar{x}_{t_{last}}) \tag{4.3}$$

$$cwd_t = cwd_{t_{last}} * 2^{-\lambda*(t-t_{last})} + s_{cw} \tag{4.4}$$

There is a time decay function applied to the CWD to reduce the influence of old labels. We use the same decay function as it is used for micro-cluster weights. If 100% of the data is labeled, the sum of the weights for all classes together is exactly the weight of the corresponding micro-cluster. This is illustrated in the following example:

For this illustration we choose λ such that a sample loses 50% of its influence after 1 time unit. There are two classes A and B and four samples with the same weight of 1. The first sample is added at time 1, the second at time 2 and so on. We measure the weight of the classes (cw_A, cw_B) and the micro-cluster's weight (w_{mc}) at time 4 which means that the latest weight has not decayed at all and the oldest weight is 3 time units old. The first two samples contain class label A, the last two samples contain class label B. We can see that $w_{mc} = cw_A + cw_B$.

$$\begin{aligned} w_{mc} &= 1 * (0.5)^3 + 1 * (0.5)^2 + 1 * (0.5)^1 + 1 * (0.5)^0 && (4.5)\\ &= 0.125 + 0.25 + 0.5 + 1 && (4.6)\\ &= 1.875 && (4.7) \end{aligned}$$

$$\begin{aligned} cw_A &= 1 * (0.5)^3 + 1 * (0.5)^2 + 0 * (0.5)^1 + 0 * (0.5)^0 && (4.8)\\ &= 0.125 + 0.25 && (4.9)\\ &= 0.375 && (4.10) \end{aligned}$$

$$
\begin{aligned}
cw_B &= 0 * (0.5)^3 + 0 * (0.5)^2 + 1 * (0.5)^1 + 1 * (0.5)^0 && (4.11)\\
&= 0.5 + 1 && (4.12)\\
&= 1.5 && (4.13)
\end{aligned}
$$

The resulting Class Weight Dictionary contains two key-value pairs, one for each class. The class label is the key where as the class weight of the corresponding class is the value.

$$cwd = \{'A' : 0.375, 'B' : 1.5\} \tag{4.14}$$

4.2 Propagating Labels

When a new micro-cluster is created and the first data sample assigned to it does not contain a class label, we must ensure that the micro-cluster gets some class label information from its neighbors. Without that the new micro-cluster is not able to classify the data sample. We implement and evaluate two different approaches. The first one uses K-Nearest Neighbors, the second one uses a Minimum Spanning Tree to find the neighbors of a micro-cluster. Whenever we mention the term "graph approach" we mean K-Nearest Neighbors or Minimum Spanning Tree for label propagation.

4.2.1 Finding neighbors using K-Nearest Neighbors (k-NN)

When a new micro-cluster mc_{new} is created, the algorithm calculates the distances between all existing micro-clusters and mc_{new}. The k micro-clusters with the shortest distance to mc_{new} are defined as its neighbors. k is set as a hyper parameter. The distance between two micro-clusters is the euclidean distance between their centers.

4.2.2 Finding neighbors using a Minimum Spanning Tree (MST)

A Minimum Spanning Tree is a graph that connects all vertices in a way such that the sum of the edge weights is minimal. In our case the graph vertices are the micro-clusters and the edge weights are the euclidean distances between the micro-clusters' centers. In a first step, we compute the distances between every possible pair of micro-clusters and create a fully connected graph. In the second step, we apply the Prim's algorithm to build the MST. Prim's algorithm is faster in dense graphs than Kruskal's algorithm. The neighbors of a micro-cluster are all micro-clusters that are connected to it by an edge.

Figure 4.1 illustrates the differences in defining the neighbors of a micro-cluster for the different graph approaches. The blue micro-clusters are the existing ones, the grey micro-cluster denotes mc_{new}. The black lines are the edges in the graph. The thick lines define the edges to mc_{new}'s neighbors. According to the MST, mc_{new} has two neighbors from which the class label information is inherited. 1-NN only defines the closest micro-cluster as a neighbor, 2-NN the two closest micro-clusters and 3-NN defines all three 3 micro-clusters as neighbors. In this case MST and 2-NN do not create the same graph, but the resulting neighbors are the same.

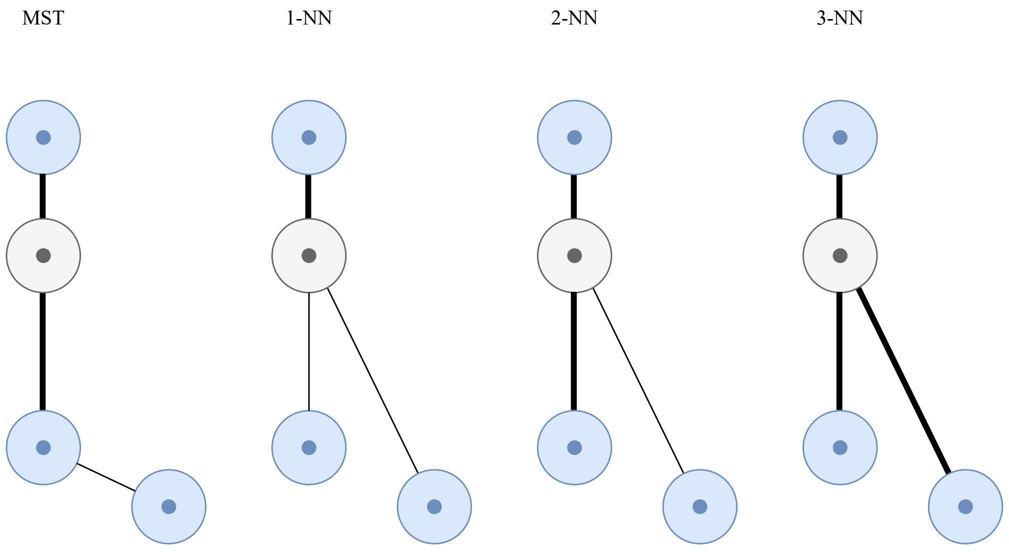

Figure 4.1: Illustration of different graph approaches for finding neighboring micro-clusters. Neighbors are searched for the grey micro-cluster. Found neighbors are connect by a thick line. All lines together denote the graph

4.2.3 Propagating labels from neighbors to new micro-cluster

Once the set of neighbors is defined, the new micro-cluster can inherit the class labels from its neighbors. We apply a distance decay function, so that class labels from close micro-clusters weigh more than those from distant micro-clusters.

Figure 4.2 shows three situations. Situation (1) shows the initial situation before a new micro-cluster is created. There are three named micro-clusters K, L and M. All the micro-clusters contain class weights for two classes . The class weights are denoted by the numbers in the micro-clusters. Let's assume the classes are called class A and class B. Class weights are named cw_A for class A and cw_B for class B or in general cw when referring to all class weights of a micro-cluster. $cw_A(K)$ denotes the class weight A for micro-cluster K, which is 16. If class A outweighs class B the micro-cluster is colored blue. Otherwise it is colored red. Then, a Minimum Spanning Tree with the three micro-clusters is created. Neighbors are connected by an edge with edge weights representing the distance between neighboring micro-clusters.

In Situation (1a) a new micro-cluster mc_{new} is created between the micro-clusters K and L. The distance to micro-cluster K is 1, the distance to micro-cluster L is 2.

mc_{new} was created because a data sample could not been assigned to an existing micro-cluster. This data sample has a weight of 1 and is unlabeled, thus class label information from neighboring micro-clusters is required. The exact calculation steps for the label propagation are illustrated for this example. Figure 4.2 contains a short version of the calculation steps. On the following page, they are explained in more details.

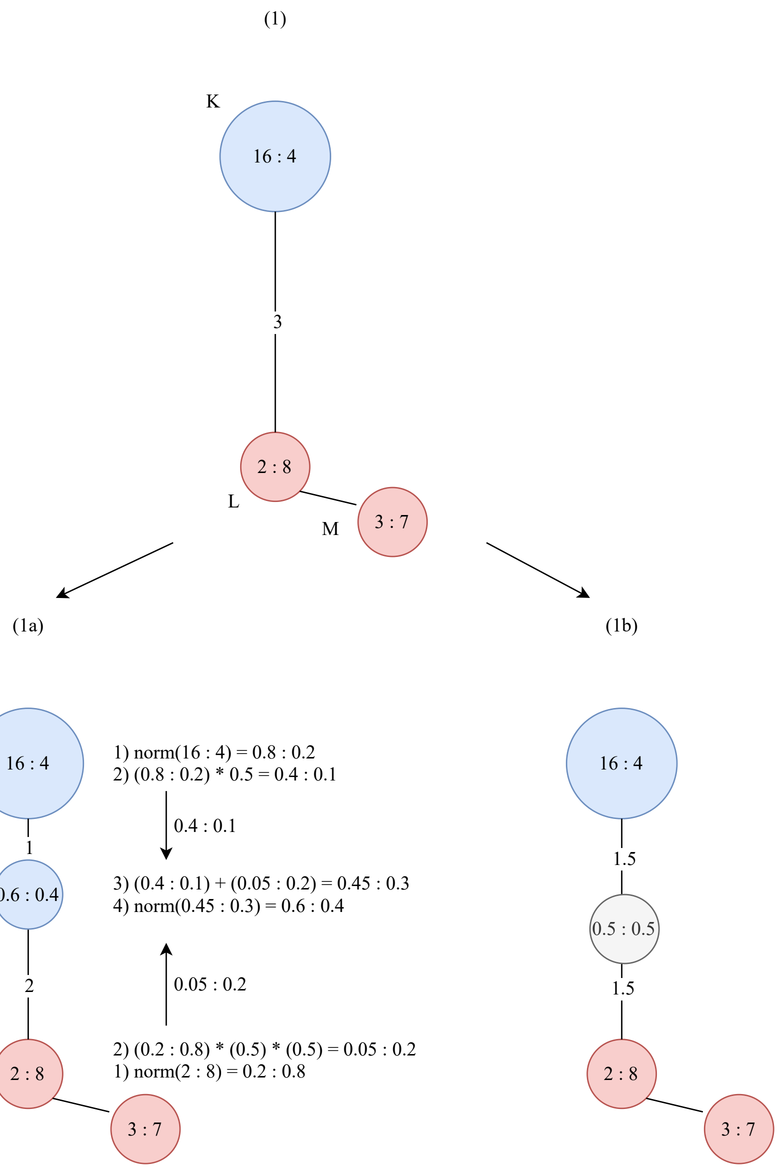

Figure 4.2: Propagating class weights to the new micro-cluster

1. The class weights of the neighbors (micro-clusters K and L) are normalized. Due to the normalization, heavy micro-clusters have not more influence than light ones. Otherwise minor classes in imbalanced data sets will be easily overruled. The terms heavy and light refer to the weight of a micro-cluster.

$$norm(cw(K)) = norm(16 : 4) = 0.8 : 0.2 \tag{4.15}$$
$$norm(cw(L)) = norm(2 : 8) = 0.2 : 0.8 \tag{4.16}$$

2. The normalized class weights are propagated to the new micro-cluster. There is a distance decay function *d_decay* applied to the class weights. Distance decay in this example is 50% after distance d of 1, i.e. the weight loses half of its influence after a distance of 1.

$$d\_decay(d) = 0.5^d \tag{4.17}$$
$$norm(cw(K)) * d\_decay(1) = (0.8 : 0.2) * (0.5)^1 = 0.4 : 0.1 \tag{4.18}$$
$$norm(cw(L)) * d\_decay(2) = (0.2 : 0.8) * (0.5)^2 = 0.05 : 0.2 \tag{4.19}$$

3. The propagated class weights of all neighbors are summed up.

$$(0.4 : 0.1) + (0.05 : 0.2) = 0.45 : 0.3 \tag{4.20}$$

4. The sum is normalized. Total weight must sum up to 1 because this is the weight of the data sample assigned to mc_{new}.

$$norm(0.45 : 0.3) = 0.6 : 0.4 \tag{4.21}$$
$$cw_A(mc_{new}) = 0.6 \tag{4.22}$$
$$cw_B(mc_{new}) = 0.4 \tag{4.23}$$

Situation (1b) is similar to situation (1a) except for the coordinates of mc_{new}. In this example, mc_{new} is located exactly in the middle between the micro-clusters K and L. As these two micro-clusters have an exact opposite class weight ratio and the distances to mc_{new} are identical, the propagated class weights result in 0.5 : 0.5. Calculation steps are not further explained as they follow the same logic as in situation (1a). Label propagation is only depending on the pureness of micro-clusters (ratio of class weights) and the distance between micro-clusters.

For demonstrating how the label propagation works we chose distances and distance decay such that the examples are easy to calculate. The implementation of SSDenStream does not contain any static distance decays. We use two different distance decay approaches for k-NN and MST. When working with k-NN, distance decay is depending on the distance d_{min} of mc_{new} to its closest neighbor. Propagated class weights loose 50% of their influence after a distance of d_{min}. When using a MST, the distance decay is depending on the median length d_{med} of all edges in the MST. After a distance of d_{med} the propagated class weights loose 50% of their weight. We did not investigate further distance decay functions and leave this for future work.

4.3 Classifying unlabeled sample

In Section 4.1 we explained how the micro-clusters store the available class label information. This is done for all incoming labeled samples. If an unlabeled sample is assigned to a micro-cluster, the stored information is used to classify it. The class weights contained in the Class Weight Dictionary of the corresponding micro-cluster are normalized such that the sum of all weights is 1. The resulting weights for each class are the final probabilities of the unlabeled sample belonging to the related class. The following example uses the Class Weight Dictionary of equation 4.14.

$$norm(cw_A) = \frac{0.375}{1.875} = 0.2 \tag{4.24}$$

$$norm(cw_B) = \frac{1.5}{1.875} = 0.8 \tag{4.25}$$

The sample is soft classified with the probability of 0.2 to belong to class A and probability of 0.8 to belong to class B. If hard classification is required the probabilities are rounded and thus the sample will be classified as class B.

4.4 Finding Neighbors Using MST vs. k-NN

There are some important differences in using k-NN or a MST for finding the neighbors of a micro-cluster. We state them in this section and use some examples to illustrate them.

The run time of both approaches is depending on the number of existing micro-clusters and on the frequency of creating new micro-clusters. Searching for K-Nearest Neighbors is done in linear run time because the distance from the new micro-cluster to every other has to be computed. Creating a MST has a quadratic run time because in the first step a fully connected graph has to be built by calculating the distances between every pair of micro-clusters. In addition, the Prim's algorithm for creating the MST from the fully connected graph has a quadratic run time as well. [Hassani et al., 2016] presents a technique which can be used to reduce the computation time for our MST approach. Due to lack of time we leave the implementation of it as future work.

Another difference between MST and k-NN is that k-NN brings an additional hyper parameter that must be tuned. Choosing the correct number of neighbors k has a great impact on the results. Creating a MST is not depending on any further hyper parameters.

Figure 4.3 shows two well separated round clusters and an illustration of micro-clusters connected by a MST. In this example, using 1-NN will always outperform the MST approach. The run time is much better and there won't be any edges connecting two micro-clusters of different classes.

Figure 4.4 shows a more complicated scenario. There are two overlapping clusters. If a new micro-cluster in the overlapping part of the clusters is created and 1-NN is used,

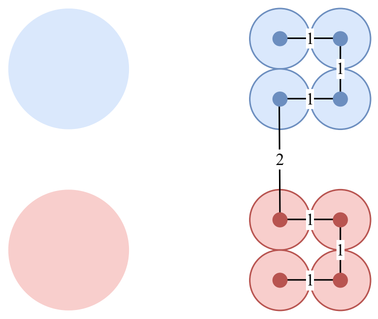

Figure 4.3: Two well separated clusters and its illustration as micro-clusters connected by a MST

there is a high chance that it will be connected to a micro-cluster of the wrong class. If this neighbor has a pure ratio of class weights, the new micro-cluster will be pure as well. This will result in a classification of samples, that look confident but that will be completely wrong. Using a higher number for k can flatten this error by propagating the class weights of several neighbors. But choosing k too high will have a negative influence too, as micro-clusters in non-overlapping areas will be connected to opposite class micro-clusters. A MST does not have this problem of finding the correct hyper parameter k.

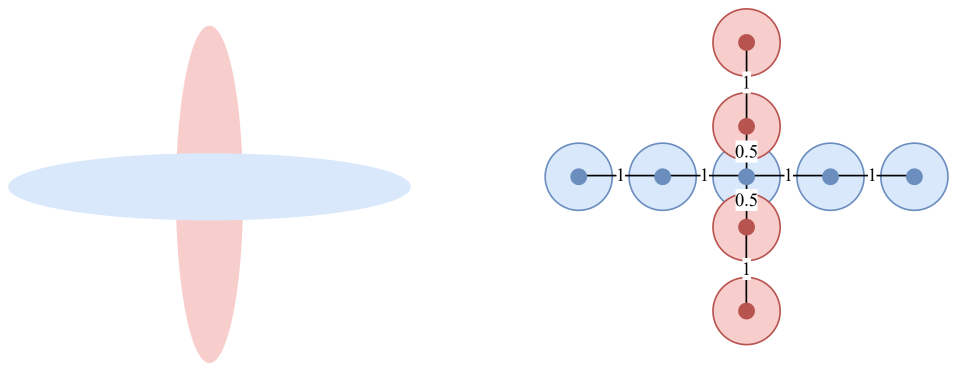

Figure 4.4: Two overlapping clusters and its illustration as micro-clusters connected by a MST

Figure 4.5 shows two clusters with different shapes. Using 1-NN succeeds as long as there is no overlap at all. Choosing a slightly higher value for k will result in wrong micro-cluster connections, as k-NN extends the search for neighbors in a circular form. On the contrary, a MST is able to connect micro-clusters of various shaped clusters. Other density based clustering algorithms use MST exactly for this reason, e.g. [Campello et al., 2013] and [Gertrudes et al., 2019].

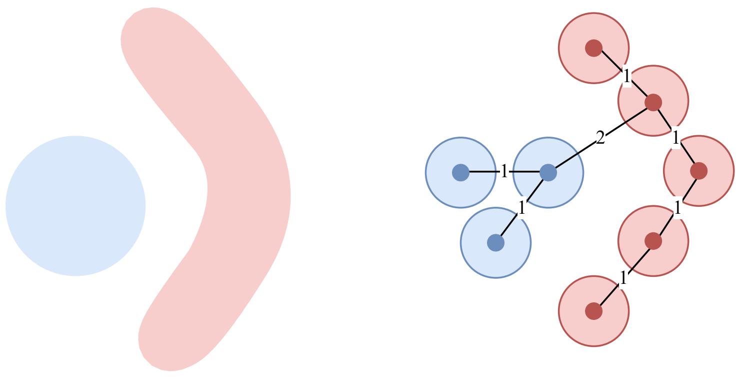

Figure 4.5: Two differently shaped clusters and its illustration as micro-clusters connected by a MST

5 Evaluation

We conduct three experiments. The first experiment explores the effect of hyper parameters on the method performance. The second experiment compares the performance of the different graph approaches for label propagation. The last experiment compares SSDenStream to state-of-the-art algorithms. Section 5.1 describes the experimental setup including a presentation of the used data sets. Sections 5.2, 5.3 and 5.4 present the three experiments and their results. Section 5.5 describes how we apply SSDenStream to the use case of Wikidata vandalism detection.

For the sake of reproducibility the scripts to create and prepare the data sets, the data sets themselves, all hyper parameter combinations used for grid search and the finally chosen hyper parameters are stored in the GitLab repository. We do not provide the URL to the repository because it is not public.

5.1 Experimental Setup

In this section, we present the setup of our experiments. Section 5.1.1 explains how and why we split our data sets. Section 5.1.2 presents the data sets we use for our experiments. In Section 5.1.3 we explain the configuration of SSDenStream. Section 5.1.4 describes the evaluation metrics we use.

5.1.1 Data Set Split

We split all data sets into a train, validation and test set. The train set is relatively small and is used to create initial micro-clusters with class label information. This is achieved by running SSDenStream on the fully labeled train set. The remaining data is split into validation and test set. We run grid search on the validation set to find a proper hyper parameter configuration. The validation set must be big enough to successfully find good hyper parameters. Especially the tuning of decay factor λ needs enough data samples. Another reason we cannot use a small validation set is the threat of over-fitting. If the validation set is small, the best results are achieved when each data sample in the train set belongs to its own micro-cluster. We are able to counteract this trend with a large validation set. After selecting hyper parameter values we run SSDenStream on the test data set to evaluate its performance.

We compare the performance of SSDenStream for certain data sets with the performance of SVM. We run grid search with 5-fold cross-validation to properly tune SVM's hyper parameters. For the 5-fold cross validation we merge the train and validation sets because this split is not necessary when working with cross validation. Test set remains the same as for SSDenStream.

5.1.2 Data Sets

We use three different data sets to evaluate the performance of SSDenStream. We present them in this section.

Overlapping Clusters

We use synthetic 2-dimensional data sets to test the performance of SSDenStream on simple overlapping clusters. The data sets were created by [Mariescu-Istodor and Zhong, 2016] and can be downloaded[1]. They are called the G2 data sets, come in nine different variations and include two clusters with different degrees of overlap. G2-2-10 which has no overlap at all and G2-2-90 which contains the highest amount of overlap are shown in Figure 5.1. Figures A.1 and A.2 visualize all nine data sets. Each data set contains 2048 samples evenly distributed between the two classes.

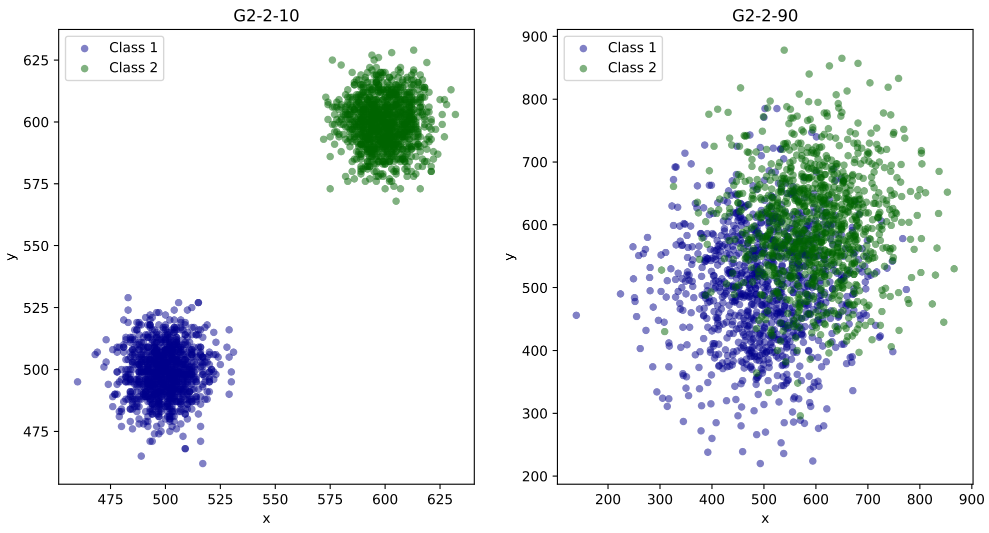

Figure 5.1: Data sets G2-2-10 and G2-2-90 visualized as scatter plots

We split all of the nine data sets with the same approach. Each train set contains 200 samples. The remaining 1848 samples are equally split into validation and test data set. The train sets contain 100% labeled data, each validation and test set contains 1% labeled data. Table 5.1 shows an overview of the data sets' characteristics.

[1]http://cs.joensuu.fi/sipu/datasets/ (31.05.2021)

	# of Samples of Class 1	# of Samples of Class 2	Total Number of Samples	Labeled Data
Train	100	100	200	100%
Validation	462	462	924	1%
Test	462	462	924	1%

Table 5.1: Number of samples in train, validation and test split of G2 data sets and their ratio of labeled data

Wikidata

We use the data sets from the WSDM Cup 2017's vandalism detection challenge[2] to evaluate SSDenStream on the Wikidata vandalism task. The data on the website is split in data sets which contain the chronologically ordered changes done to Wikidata from October 2012 until June 2016. The changes are assigned to one of two classes: vandalism and non-vandalism. The two classes are strongly imbalanced. The most recent data sets have a vandalism rate of about 0.1%.

The data sets contain the Wikidata changes in XML format. We use the code of the WSDM Cup 2017's winning team, called Buffaloberry[3], to create features. They created 95 features, of which 67 are categorical, 4 represent ids and 24 are numerical. We only use numerical features. There are two numerical features that represent meta data about the user that initiated the corresponding change at Wikidata. We do not use them as they induce user bias. These are the remaining 22 features that we use for SSDenStream: *alphanumericRatio, amount, badWordRatio, bracketRatio, commentTailLength, digitRatio, fuzzy_partial, fuzzy_total, json_len, lang_prob, languageWordRatio, latinRatio, longestCharacterSequence, longestWord, lowerCaseRatio, lowerCaseWordRatio, nonLatinRatio, punctuationRatio, simbolRatio, upperCaseRatio, upperCaseWordRatio, whitespaceRatio*. [Crescenzi et al., 2017] explains all features in more detail.

Validation set (wdvc16_2016_03) and test set (wdvc16_2016_05) are already given by the WSDM Cup 2017. We use two different train sets. One contains all the data from the data set called wdvc16_2016_01. The second train set is more balanced. We take all vandalism attempts from wdvc16_2016_01 and randomly select non-vandalism samples such that it results in a data set that contains a class ratio of one to ten. We call it wdvc16_2016_01_10%. Table 5.2 gives an overview of these data sets. For an overview of all data sets published by the WSDM Cup 2017, see Table A.1.

Pulsar Candidates

We test SSDenStream on another real world data set called HUTR2. This data set[4] is introduced by [Lyon et al., 2016]. It contains data about pulsar candidates. A pulsar

[2]https://www.wsdm-cup-2017.org/vandalism-detection.html (01.06.2021)
[3]https://github.com/wsdm-cup-2017/buffaloberry (01.06.2021)
[4]http://archive.ics.uci.edu/ml/datasets/HTRU2 (01.06.2021)

	Total number of Samples	# of Vand. Samples	Vandalism Rate	Labeled Data
Train (wdvc16_2016_01)	9’336’013	9’551	0.10%	100%
Train 1:10 (wdvc16_2016_01_10%)	95’510	9’551	10%	100%
Validation (wdvc16_2016_03)	7’214’141	10’784	0.15%	1%
Test (wdvc16_2016_05)	10’433’991	11’043	0.11%	1%

Table 5.2: Key figures of Wikidata train, validation and test sets

is a certain type of star. The data set contains 17898 samples of which 1639 are pulsar candidates. There are eight features which are all numeric. Further feature processing is not required. [Lyon et al., 2016] presents an experiment that is run on a train set containing 200 samples of both classes. The experiment is repeated 500 times and for each run the train set is sampled randomly. Thus we cannot use the exact same train, validation and test sets because they are not reproducible. We decide to use the same size and same class balance in the train set as is presented by [Lyon et al., 2016]. The test set contains 50% of all data samples. The remaining 8549 samples are used as validation set. Table 5.3 gives an overview of the different data splits.

	Total number of Samples	# of Pulsar Candidates	Pulsar Rate	Labeled Data
Train	400	200	50%	100%
Validation	8549	619	7.2%	1%
Test	8949	820	9.2%	1%

Table 5.3: Overview of HUTR2 train, validation and test sets

5.1.3 SSDenStream Configuration

We evaluate the influence of the hyper parameters half-life time $t_{1/2}$, maximum radius ϵ and minimum cluster size ζ and the different graph approaches on the performance of SSDenStream. The remaining hyper parameters and configuration options are kept constant for all experiments.

We set the drift hyper parameters δ and ω to 0. Our focus is not on evaluating the effectiveness of those two hyper parameters, thus we deactivate the drift hyper parameters to reduce the number of hyper parameters to tune.

SSDenStream has inherited the functionality of DenStream* to scale the data samples on the run. There is the choice between Min-Max and Z-Score normalization. Z-Score normalization works better with normally distributed data. We do not make any assumptions about the data distribution of our data sets and thus choose Min-Max normalization. Parameters for the Min-Max normalization are extracted from the train set and applied to the validation and test sets.

5.1.4 Evaluation Metrics

We select hyper parameter values from the grid search by choosing a combination that results in a high F1 score and in a low run time. Often there are several hyper parameter combinations that result in a similar F1 score but which have different run times. A high run time is the result of having many small micro-clusters or when micro-clusters are created and deleted often. We do not want to have such a configuration as this indicates over-fitting or unstable micro-clusters.

We refer to *soft classification* as when data samples are given a probability of belonging to a class. The probabilities of all classes together sum up to 1. *Hard classification* does not make any use of probabilities.

SSDenStream is able to do soft and hard classification. Soft classification is evaluated on the G2 data sets. As these data sets are balanced, we use ROC AUC. When evaluating SSDenStream's hard classification we use the confusion matrix, precision, recall and F1 score. These are widely used metrics in classification tasks and thus allow us to compare our results to other studies.

5.2 Experiment 1 - Influence of the Hyper Parameters

In this section, we investigate the influence of the hyper parameters on the performance of SSDenStream. There are three hyper parameters to tune, besides selecting a graph approach, namely half-life time $t_{1/2}$, minimum cluster size ζ and maximum radius ϵ. ϵ defines the radius of a micro-cluster, i.e. the distance a data sample is allowed to have to a micro-cluster's center. The bigger ϵ, the larger a micro-cluster's radius.

ζ influences two thresholds: (i) the weight threshold that differentiates between o-micro-clusters and p-micro-clusters and (ii) the weight threshold that defines if an o-micro-cluster gets deleted. The higher ζ, the higher the weight threshold becomes.

$t_{1/2}$ determines how fast a micro-cluster is losing importance in form of weight. A high value for $t_{1/2}$ states a high half-life time which results in low weight decay.

The hyper parameters determine the granularity with which the available information in form of labelled data is stored. If we choose large micro-clusters the class labels are aggregated in few micro-clusters. If we choose small micro-clusters, the vector space is divided more granularly. If the different classes of the data are well separated and occupy large areas in the vector space, big micro-clusters can be used. But if the data is overlapping, smaller micro-clusters are required because we need to distinguish the vector space's regions with a higher granularity. The hyper parameters also determine the adaptability of SSDenStream to changing patterns in the data, for example drifting data. If the data is static or only barely moving, a high value for $t_{1/2}$ can be chosen. But if there is a lot of drift in the data set, the micro-clusters must forget their information faster, thus selecting a low value for $t_{1/2}$ makes sense.

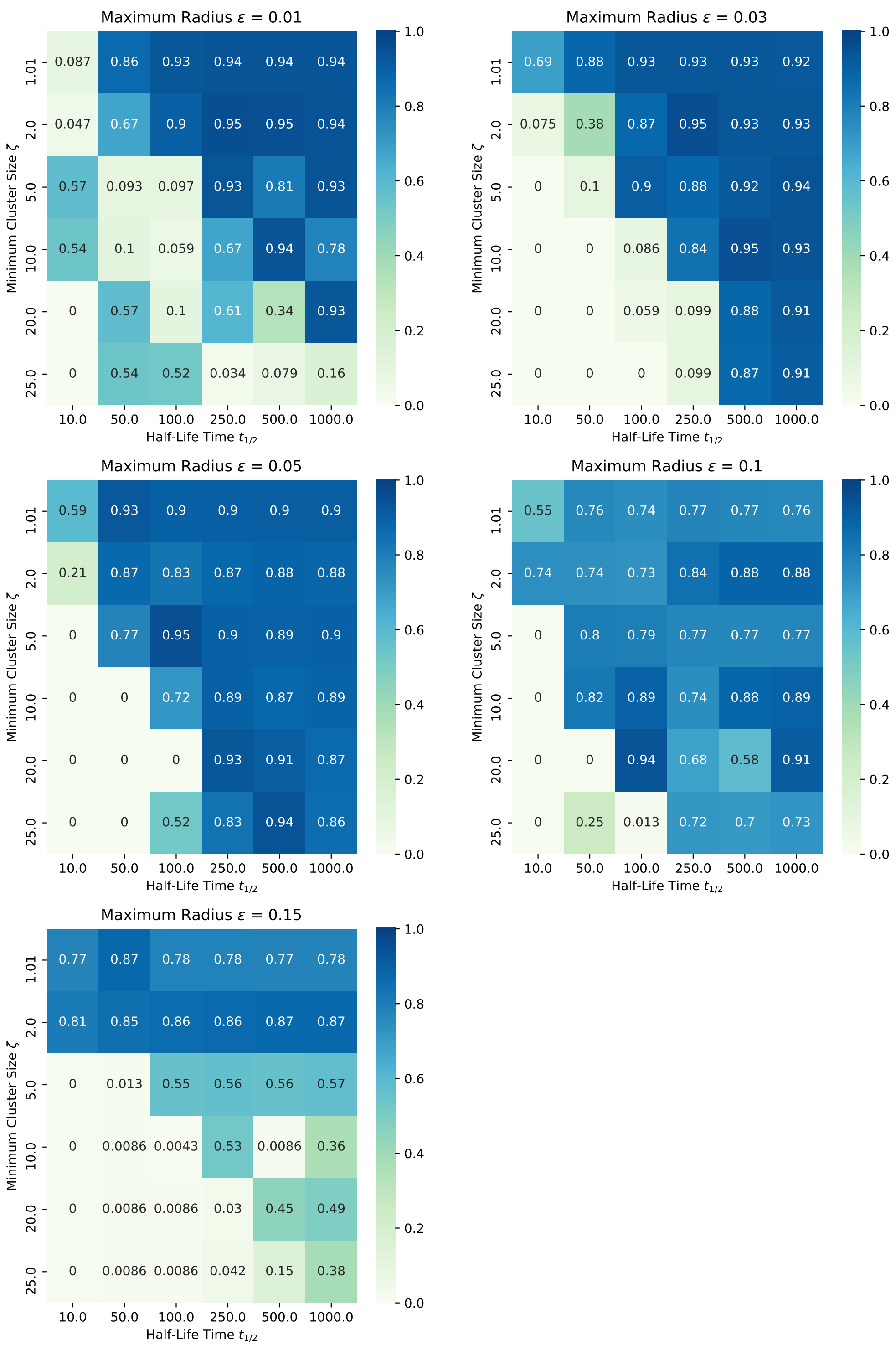

Figure 5.2: F1 scores for all hyper parameter combinations of the grid search done on G2-2-40 with 1-NN

Figure 5.2 shows the different F1 scores for all hyper parameter combinations that were examined during grid search for SSDenStream with 1-NN on the G2-2-40 data set. One heat map shows the results for all combinations of ζ and $t_{1/2}$ for a fixed value of ϵ. One tile contains the result of one hyper parameter configuration. A tile which is located bottom left in the heat map represents a configuration that creates short-living micro-clusters. The more top right a configuration, the more stable and long-living the micro-clusters. They lose weight slowly and must fall below a small weight threshold to be deleted.

We observe that for $\epsilon = 0.01$, especially attempts in the upper right corner are successful. For an increasing ϵ, more short-living micro-clusters are also viable and the heat map turns blue from the top right towards the bottom left, indicating good results. This makes sense because with a larger ϵ the micro cluster's area and thus the inflow of data samples grows. The micro-cluster can thus hold enough weight not to be deleted.

Above a certain value of ϵ, the quality of the classification decreases again. The radius of the micro-clusters becomes too large and thus overlapping regions are no longer clearly delimited. The data set G2-2-40 has only a small area of overlapping clusters. If the micro-clusters are larger than this overlapping area, data points which are actually located in the pure part of the cluster are added to these impure micro-clusters.

There are combinations that let SSDenStream fully fail. This is the case when the micro-clusters are created and deleted more often than new class labels appear. There will be micro-clusters with no class label information at all and therefore the incoming data samples cannot be classified. In this case, the run is considered failed and is assigned an F1 score of 0.

This illustration states the dependence and influence of the different parameters on each other and on the results. A good result can only be achieved if all three parameters are set properly. Working on a 2-dimensional data set is easy, because data can be displayed. With some experience a range of proper hyper parameter values can be estimated. Working with high dimensional data is much harder because we cannot display it and get an estimation of the data distribution. Therefore grid search is required to find optimal values for the hyper parameters.

Table 5.4 shows the hyper parameters for the G2 data sets that were found to work good for all further examined graph approaches. The results of the G2 data sets that are presented in the following sections are computed with these hyper parameters. Half-life time is set to 1000 for the most of the data sets which is high compared to the size of the data sets. This states that there is no drift in the data. Generally, it can be said that the combinations of ζ and ϵ prefer lower values with increasing cluster overlap. This results in higher granularity. More granularity in strongly overlapping clusters makes sense as the regions must be distinguished more precisely.

Our analysis shows that there exists a range for the hyper parameter values for which SSDenStream achieves good results. On all nine G2 data sets SSDenStream performs reliably for $\zeta \leq 5$, $t_{1/2} \geq 500$ and $0.03 \leq \epsilon \leq 0.05$ (see Appendix B.1). But this does not guarantee that they work on other data sets as well. The characteristics of a data set determine which range of values will work. For example the data distribution, number of clusters, dimensionality, the size of the data set, or the degree of drift in the data play

Data Set	Half-Life Time $t_{1/2}$	Minimum Cluster Size ζ	Maximum Radius ϵ
G2-2-10	500	5	0.15
G2-2-20	500	5	0.15
G2-2-30	1000	5	0.15
G2-2-40	1000	10	0.03
G2-2-50	1000	5	0.03
G2-2-60	1000	5	0.03
G2-2-70	1000	5	0.03
G2-2-80	1000	2	0.03
G2-2-90	250	1	0.05

Table 5.4: Selected hyper parameters for the different G2 data sets

all an important role.

We do not discuss the selected hyper parameters for the two other data sets as the data sets are high dimensional and cannot be visualized. Thus we cannot draw a comparison between the parameters and the shape of the clusters in the data sets.

5.3 Experiment 2 - Comparing Different Graph Approaches

In Section 4.4 we argue that using MST prevents new micro-clusters in overlapping areas from being labeled completely wrong by connecting it to several neighbors. We also state that this can be achieved by using k-NN with $k > 1$. But if we choose a value for k that is too big, it results in poor performance as it unites the class label weights of too many micro-clusters and therefore generalizes the distribution of the data samples too much. From these statements, we derive the following three hypotheses:

Hypothesis 1 *MST outperforms 1-NN on overlapping clusters.*

Hypothesis 2 *MST outperforms 5-NN on overlapping clusters.*

Hypothesis 3 *There exist values for k such that MST and k-NN achieve similar results on overlapping clusters.*

We run SSDenStream with four different graph approaches on the G2 data sets and compare their performance. We evaluate their performance using ROC-AUC for soft classification and the F1 score for hard classification. Figures 5.3 and 5.4 show the results for this experiment.

For soft classification MST achieves better results on all nine data sets compared to 1-NN, thus Hypothesis 1 holds. Using 2-NN or 5-NN achieves similar results as MST and therefore Hypothesis 3 holds. We are surprised that 5-NN performs that good. We expected the value to be too high for k and thus it should perform worse than MST or 2-NN. These results indicate that k-NN performs stable with $1 < k <= 5$ on soft

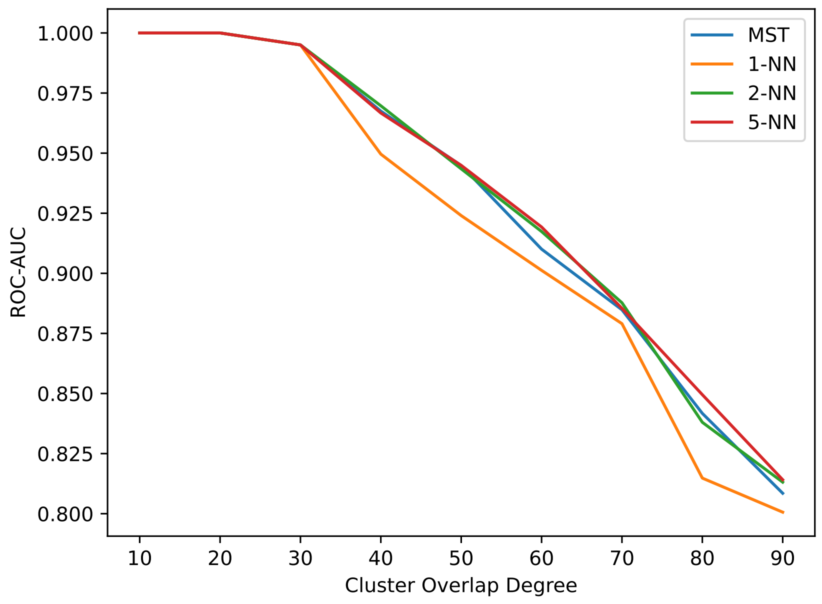

Figure 5.3: ROC-AUC for different graph approaches on the G2 data sets

classification for trivial cluster shapes. Therefore Hypothesis 2 does not hold. Further experiments have to be executed to examine what values of k let the performance of k-NN decrease.

Comparing the performances on hard classification (Figure 5.4) results in a different picture. Surprisingly, 1-NN achieves the best results on all data sets. MST and 2-NN perform similar where as 5-NN performs the worst. Thus Hypothesis 1 does not hold for hard classification but Hypothesis 3 and 2 do.

We cannot explain the differences in the performance of 1-NN with full certainty but present the following hypothesis. In overlapping clusters there exist areas where samples of both classes occur with the same frequency. This leads to micro-clusters with class weight ratios of about $0.5 : 0.5$. The higher the value of k, the larger must this area become because the label weight information of more micro-clusters are aggregated when a new micro-cluster is created. For soft classification this aggregation seems to be of advantage but hard classification achieves higher results when this area is kept as small as possible ($k = 1$). Further analysis is required to fully understand this phenomena. We leave this for future work.

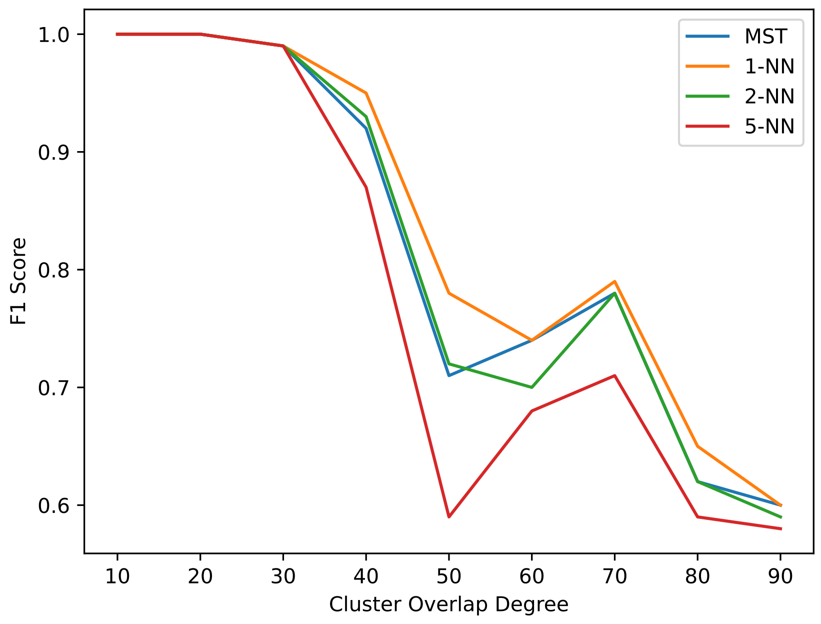

Figure 5.4: F1 scores for different graph approaches on the G2 data sets

5.4 Experiment 3 - Comparing SSDenStream to State-of-the-Art Algorithms

We want to compare the performance of SSDenStream with a supervised offline state-of-the-art algorithm to see if the performances correlate. If they correlate, it means that the principle of SSDenStream works and the performances are not random. Since the G2 and HUTR2 data sets do not contain drifting data, the state-of-the-art algorithm should perform better.

Hypothesis 4 *SSDenStream performs worse than supervised offline state-of-the-art algorithms.*

We compare the results of the G2 data sets with the performance of SVM. SVM works with numerical features and depends on the local separation of the two classes in the vector space. Thus the performance of SVM should decrease with increasing overlap of the clusters. This makes it easy to compare its results with those of SSDenStream. We also use SVM as a comparison to SSDenStream on the HUTR2 data set, as proposed by [Lyon et al., 2016]. We cannot directly compare our results to those of [Lyon et al., 2016] because we cannot reproduce their data split.

Figure 5.5 shows the F1 scores of SSDenStream and SVM on the G2 data sets. The fluctuation in SSDenStream's curve is bigger compared to the one of SVM. But both curves have a similar shape. SVM's curve can be seen as a flattened version of SSDenStream's curve. For example both curves have a peak at cluster overlap degree 70, the

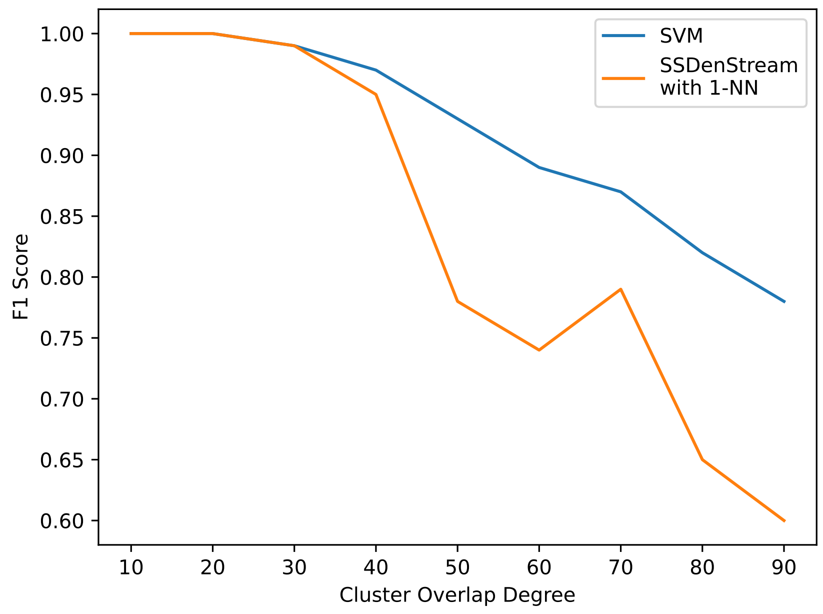

Figure 5.5: F1 scores for SVM and SSDenStream on the G2 data sets

one of SSDenStream is just bigger. There is a correlation between the results of the two algorithms. SVM achieves better results on all nine data sets. The difference in performance becomes greater with increasing overlap.

It is possible that the bigger fluctuation in SSDenStream's curve is linked to the data split. SSDenStream uses 200 data points for the initial clustering and 924 for validation. SVM's cross validation is done on all 1124 data samples and therefore more robust to the stochasticity introduced by data splits.

	F1 Score	Precision	Recall
SSDenStream	0.84	0.86	0.82
SVM	0.90	0.95	0.85

Table 5.5: Result comparison between SVM and SSDenStream with 1-NN on the HUTR2 data set

Table 5.5 shows the results of SVM and of SSDenStream on the HUTR2 data set. SVM produces slightly better results. Especially positive is that SSDenStream realizes only 3% less recall compared to SVM. This means that SSDenStream detects approximately the same amount of interesting data samples, in this case pulsar candidates, as SVM. Again, one has to be aware that SVM uses 8949 data points for the 5-fold cross validation. The initial clustering of SSDenStream uses only 400 samples. This shows that SSDenStream achieves decent results even with a small number of labeled samples.

But it also indicates that the data set's features are of high quality. It seems that a large part of the data points are well separated in the vector space and that it is difficult to classify the remaining points cleanly.

We confirm Hypothesis 4: SVM outperforms SSDenStream on all data sets. The performance difference is small on data sets with well-separated clusters. The greater the overlap (the poorer the quality of the features), the greater the performance difference.

This experiment shows that the principle of SSDenStream works. However, there is still an important experiment missing that examines how it works on drifting data. Such an experiment is required to prove that SSDenStream inherits the characteristics of DenStream which allow to process evolving data. There SSDenStream should have clear advantages over offline supervised state-of-the-arts algorithms like SVM. Due to time constraints of this thesis we could not perform this experiment and have to leave it for future work.

5.5 Use Case - Wikidata Vandalism Detection

The motivation behind this project comes from vandalism detection on knowledge bases. We first apply SSDenStream to the Wikidata data set wdvc16_2016_01. SSDenStream does not find a single vandalism attempt. As we have shown, the reliability of SSDenStream correlates with the quality of a data set's features.

Graph Configuration	Metric	wdvc16_2016_01	wdvc16_2016_01_10%
	F1 Score	0.00	0.02
1-NN	Precision	0.00	0.04
	Recall	0.00	0.02
	F1 Score	-	0.08
SVM	Precision	-	0.05
	Recall	-	0.20

Table 5.6: Resulting F1 score, precision and recall of our experiments on the two Wikidata train sets

Since this data set is very imbalanced, the features must be correspondingly good. We change the imbalance of the train set to focus on the vandalism attempts. Therefore, we create wdvc16_2016_01_10%. SSDenStream can correctly classify some vandalism attempts with this train set, but still performs very poorly.

To check if the poor performance is due to the feature selection or our algorithm, we apply SVM to it. Also SVM performs poorly: Table 5.6 shows the results in form of F1 score, precision and recall; Table 5.7 shows the confusion matrices. We did not apply SVM to wdvc16_2016_01 because of its quadratic run time. We conclude from these results that our feature selection is the reason for the poor performance.

		1-NN		SVM	
		Normal	Vandalism	Normal	Vandalism
wdvc16_2016_01	Normal	10’433’984	0	-	-
	Vandalism	11’043	0	-	-
wdvc16_2016_01_10%	Normal	10’429’000	4’977	10’393’685	40’306
	Vandalism	10’861	182	8’802	2241

Table 5.7: Confusion matrices of our experiments on the two Wikidata train sets

Table 5.8 shows the results of team Buffaloberry on the Wikidata vandalism challenge. Buffaloberry uses a supervised learning algorithm which can process categorical and numerical data. They use a total of 95 features. Buffaloberry is the winning team of the competition but still only achieves 26% recall. Around three quarters of all vandalism attempts are not found. This illustrates how hard the Wikidata vandalism problem is to solve and that it is not solved yet.

	ROC	PR	Accuarcy	Precision	Recall	F1 Score
Buffaloberry	0.94702	0.45757	0.99909	0.68197	0.26370	0.38033

Table 5.8: Performance of the winning team of the WSDM Cup 2017 on the Wikidata data set

We propose a detailed feature analysis and if necessary the creation of new features. Such an evaluation expands the range of algorithms that can be applied to the vandalism problem.

6 Limitations and Future Work

Using SSDenStream with MST is not suitable for data sets where a large number of micro clusters are created. This is due to the quadratic computation time of a MST.

Our experiments do not contain any data sets with evolving data. We were also not able to examine the influence of all hyper parameters (distance decay factor) and the amount of labeled data. There are further experiments required to fully evaluate the potential of SSDenStream.

The performance of SSDenStream was never compared to any other semi-supervised stream algorithm. We did not find any running implementation of the algorithms we introduced in the related work. Thus we compared our results only to SVM.

We use Min-Max normalization in our experiments. Min-Max normalization is not suitable for evolving data because new minimum or maximum values can appear. When conducting experiments with drifting data, either a different normalization method must be used or normalization must be omitted.

We use micro-clusters to map the incoming data stream. However, we have not verified whether there are other options and what advantages or disadvantages micro-clusters have compared to those. We do not know whether micro-clusters perform particularly well in terms of required memory or processing speed.

To overcome some of these limitations and to close knowledge gaps regarding the performance of SSDenStream we propose the following topics for future work.

Drifting Data A characteristic of a data stream is that it can contain evolving data. Our experiments do not involve any drifting data. Thus we cannot present any results of SSDenStream on such data. We propose to apply SSDenStream on synthetic and real world data sets with evolving data. This allows to further investigate the utility of SSDenStream for real world stream applications.

Impact of Labeled Data The validation and test sets of our experiments always contained a labeling rate of 1%. We do not know what impact the amount of labeled data has on the hyper parameter tuning and on the choice of the graph approach. We suggest to add further experiments with varying amount of labeled data samples. We recommend to combine these experiments with the exploration of SSDenStream's utility on drifting data. The amount of labeled data should have an influence on the ability of

SSDenStream to classify drifting clusters correctly.

Distance Decay Function The distance decay function is used during label propagation. It regulates the influence of a micro cluster to its neighbor based on its distance. The influence of this distance decay function has not yet been further investigated. We propose to add further experiments to examine the distance decay function. Maybe it is possible to find any correlation between the data distribution and the distance decay function and thus to automatically tune it.

Use Different Underlying Clustering Algorithm We use DenStream as base algorithm which we extend with a semi-supervised mechanism to classify samples on the run. We use DenStream's definition of micro clusters for data aggregation of a data stream. DenStream has certain drawbacks, for example hierarchical distinction of clusters is not possible. It is also not able to merge or split micro clusters like in evolution-based algorithms. We propose to investigate if our semi-supervised extension can be applied to a more modern density-based stream clustering algorithm. Possible starting point can be [Hassani et al., 2016].

Multiclass Our experiments were conducted only on binary data sets. Theoretically, our approach also fits the application area of multiclass problems. We propose to conduct experiments on multiclass data sets. If the applicability of SSDenStream to multiclass problems were confirmed, the scope of this application would increase significantly.

Categorical Features Many data sets contain categorical features. SSDenStream, however, is only able to handle numerical features. In the related work section, we present [Lin and Lin, 2009] and [Chen and He, 2016] that introduce mechanisms to extend micro clusters to handle categorical features. We recommend analyzing these works and extending SSDenStream accordingly. This will increase the applicability of SSDenStream.

Hyper Parameter Evaluation We use an unconventional train and validation split for our experiments. The validation set is quite large compared to the train set. One reason is that an ordinary train and validation split results in over-fitting. If the validation set is small, the best results can be achieve if each data sample in the train set has its own micro cluster. We propose to investigate if a new metric can be created that compares the number of micro clusters to the quality of classification. Good hyper parameters should produce as precise a classification as possible and at the same time as few micro clusters as possible.

7 Conclusions

Data streams are becoming increasingly important. Methods are needed to evaluate them continuously. This can be to discover interesting data points or to monitor a system. It is possible that patterns in the data change over time. However, today's solutions are often based on a supervised approach. These cannot adapt to evolving data. Semi-supervised algorithms are needed to meet this challenge.

We present in this thesis SSDenStream, a semi-supervised stream clustering algorithm capable of doing online classification. SSDenStream is based on DenStream, an unsupervised stream clustering algorithm. We extend DenStream with a mechanism that is able to process partially labeled data.

DenStream aggregates the samples of a data stream into micro-clusters. It is able to recognize clusters in different shapes and distributions. We extend the structure of micro-clusters to store the class membership of labeled data. This serves as a basis to classify unlabeled data points. We apply a decay function so that class label information loses influence over time. This is required so that SSDenStream can react to changing data and adapt the classifications accordingly. When new micro-clusters emerge, they obtain class label information from neighboring micro-clusters. We present two different approaches to define which micro-clusters are adjacent. One uses a Minimum Spanning Tree, and the second approach relies on K-Nearest Neighbors. We describe the theoretical advantages and disadvantages of these approaches and evaluate some of those in our experiments.

We perform several experiments to evaluate the potential of SSDenStream. We use a synthetic data set with overlapping clusters for this purpose. In addition, we evaluate SSDenStream on two real world data sets. We first show how the hyper parameters are related. It turns out that correct hyper parameter tuning is essential for SSDenStream to work. Then we evaluate the performance of MST and k-NN. It turns out that the higher computation time of MST does not pay off with our data sets.

Finally, we compare the performance of SSDenStream with state-of-the-art algorithms. It is shown that the performance of SSDenStream correlates with the performance of SVM. The better the features distinguish the classes of a data set, the better the results of SSDenStream. SSDenStream performs well on the pulsar candidates data set as well. SVM outperforms SSDenStream. This is not surprising as SVM is an offline supervised algorithm. However, SSDenStream can offer the advantages of a semi-supervised algorithm.

SSDenStream is an interesting approach for semi-supervised stream classification. However, more experiments are needed to further evaluate its performance. We suggest several directions that can be pursued to make further progress in the field of semi-supervised stream classification.

References

[Campello et al., 2013] Campello, R. J., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer.

[Cao et al., 2006] Cao, F., Estert, M., Qian, W., and Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 328–339. SIAM.

[Chen and He, 2016] Chen, J.-Y. and He, H.-H. (2016). A fast density-based data stream clustering algorithm with cluster centers self-determined for mixed data. *Information Sciences*, 345:271–293.

[Crescenzi et al., 2017] Crescenzi, R., Fernandez, M., Calabria, F. A. G., Albani, P., Tauziet, D., Baravalle, A., and D'Ambrosio, A. S. (2017). A production oriented approach for vandalism detection in wikidata-the buffaloberry vandalism detector at wsdm cup 2017. *arXiv preprint arXiv:1712.06919*.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[Gertrudes et al., 2019] Gertrudes, J. C., Zimek, A., Sander, J., and Campello, R. J. (2019). A unified view of density-based methods for semi-supervised clustering and classification. *Data mining and knowledge discovery*, 33(6):1894–1952.

[Halkidi et al., 2012] Halkidi, M., Spiliopoulou, M., and Pavlou, A. (2012). A semi-supervised incremental clustering algorithm for streaming data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 578–590. Springer.

[Hassani, 2015] Hassani, M. (2015). *Efficient clustering of big data streams*. Apprimus Wissenschaftsverlag.

[Hassani et al., 2016] Hassani, M., Spaus, P., Cuzzocrea, A., and Seidl, T. (2016). I-hastream: density-based hierarchical clustering of big data streams and its application to big graph analytics tools. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 656–665. IEEE.

[Heindorf et al., 2017] Heindorf, S., Potthast, M., Engels, G., and Stein, B. (2017). Overview of the wikidata vandalism detection task at wsdm cup 2017. *arXiv preprint arXiv:1712.05956*.

[Heindorf et al., 2015] Heindorf, S., Potthast, M., Stein, B., and Engels, G. (2015). Towards vandalism detection in knowledge bases: Corpus construction and analysis. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 831–834.

[Heindorf et al., 2016] Heindorf, S., Potthast, M., Stein, B., and Engels, G. (2016). Vandalism detection in wikidata. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 327–336.

[Heindorf et al., 2019] Heindorf, S., Scholten, Y., Engels, G., and Potthast, M. (2019). Debiasing vandalism detection models at wikidata. In *The World Wide Web Conference*, pages 670–680.

[Lin and Lin, 2009] Lin, J. and Lin, H. (2009). A density-based clustering over evolving heterogeneous data stream. In *2009 ISECS international colloquium on computing, communication, control, and management*, volume 4, pages 275–277. IEEE.

[Lyon et al., 2016] Lyon, R. J., Stappers, B., Cooper, S., Brooke, J. M., and Knowles, J. D. (2016). Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123.

[Mariescu-Istodor and Zhong, 2016] Mariescu-Istodor, P. F. R. and Zhong, C. (2016). Xnn graph. LNCS 10029:207–217.

[Mousavi et al., 2015] Mousavi, M., Bakar, A. A., and Vakilian, M. (2015). Data stream clustering algorithms: A review. *Int J Adv Soft Comput Appl*, 7(3):13.

[Neis et al., 2012] Neis, P., Goetz, M., and Zipf, A. (2012). Towards automatic vandalism detection in openstreetmap. *ISPRS International Journal of Geo-Information*, 1(3):315–332.

[Nishioka and Scherp, 2018] Nishioka, C. and Scherp, A. (2018). Analysing the evolution of knowledge graphs for the purpose of change verification. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 25–32. IEEE.

[Olivieri et al., 2017] Olivieri, A. C., Shabani, S., Sokhn, M., and Cudré-Mauroux, P. (2017). Assessing data veracity through domain specific knowledge base inspection. In *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 291–296. IEEE.

[Putina et al., 2018] Putina, A., Rossi, D., Bifet, A., Barth, S., Pletcher, D., Precup, C., and Nivaggioli, P. (2018). Telemetry-based stream-learning of bgp anomalies. In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 15–20.

[Ruiz et al., 2009] Ruiz, C., Menasalvas, E., and Spiliopoulou, M. (2009). C-denstream: Using domain knowledge on a data stream. In *International Conference on Discovery Science*, pages 287–301. Springer.

[Sarabadani et al., 2017] Sarabadani, A., Halfaker, A., and Taraborelli, D. (2017). Building automated vandalism detection tools for wikidata. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 1647–1654.

[Tan et al., 2014] Tan, C. H., Agichtein, E., Ipeirotis, P., and Gabrilovich, E. (2014). Trust, but verify: Predicting contribution quality for knowledge base construction and curation. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 553–562.

[Treechalong et al., 2015] Treechalong, K., Rakthanmanon, T., and Waiyamai, K. (2015). Semi-supervised stream clustering using labeled data points. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 281–295. Springer.

[Truong et al., 2018] Truong, Q. T., Touya, G., and de Runz, C. (2018). Towards vandalism detection in openstreetmap through a data driven approach. In *GIScience 2018*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Weber, 2019] Weber, C. (2019). Online anomaly detection on multivariate data streams. Master's thesis, University of Zurich.

[Wienand and Paulheim, 2014] Wienand, D. and Paulheim, H. (2014). Detecting incorrect numerical data in dbpedia. In *European Semantic Web Conference*, pages 504–518. Springer.

[Xu and Wunsch, 2008] Xu, R. and Wunsch, D. (2008). *Clustering*, volume 10. John Wiley & Sons.

A

Experiment Data

This appendix provides a more detailed overview of the G2 and Wikidata data sets. The first section contains the plots for each of the nine G2 data sets. The second section presents a table which presents key figures of all available data sets from the WSDM Cup 2017.

A.1 G2 Data Sets

Figures A.1 and A.2 show the plots for all nine G2 data sets.

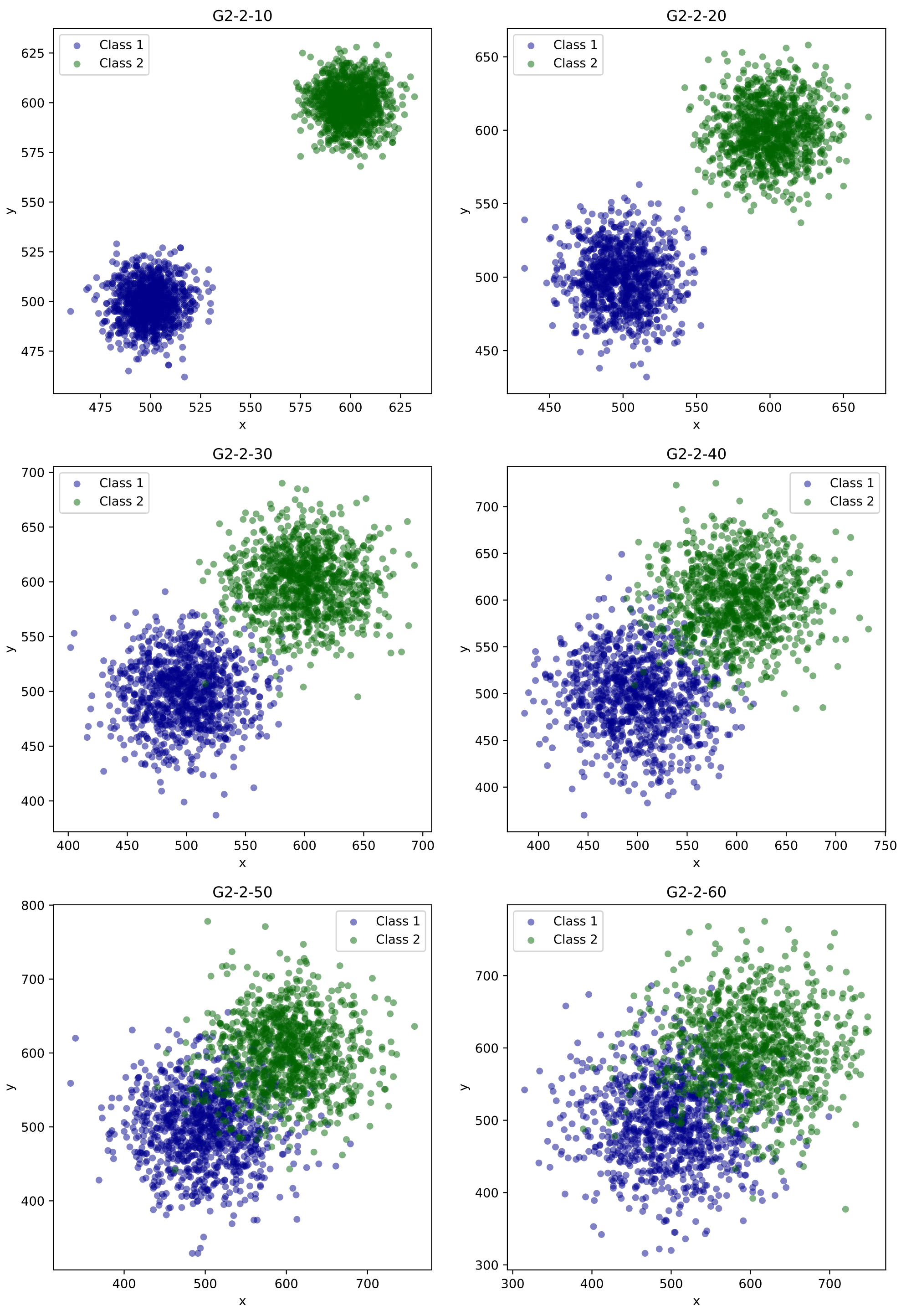

Figure A.1: Data sets G2-2-10 to G2-2-60 visualized as scatter plots

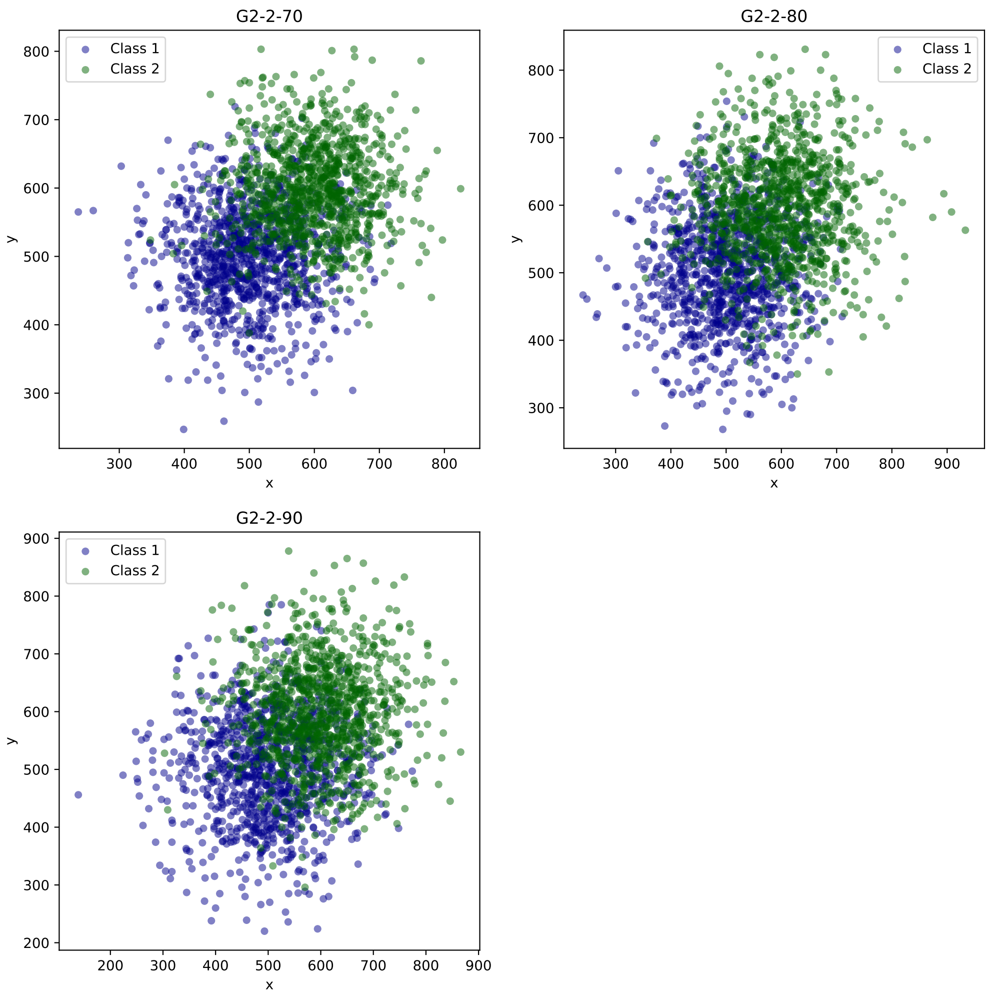

Figure A.2: Data sets G2-2-70 to G2-2-90 visualized as scatter plots

A.2 Wikidata Vandalism Data Sets

Table A.1 shows the number of samples, number of vandalism samples and their ratio for each data set that is part of the WSDM Cup 2017.

Data Set Name	# of Samples	# of Vandalism Samples	Vandalism Rate
wdvc16_2012_10	264'618	18	0.0068%
wdvc16_2012_11	432'420	289	0.0668%
wdvc16_2013_01	562'204	2'628	0.4674%
wdvc16_2013_03	1'623'142	16'580	1.0215%
wdvc16_2013_05	1'766'531	9'614	0.5442%
wdvc16_2013_07	1'530'703	6'848	0.4474%
wdvc16_2013_09	1'322'024	9'733	0.7362%
wdvc16_2013_11	1'716'302	8'004	0.4664%
wdvc16_2014_01	1'731'642	10'153	0.5863%
wdvc16_2014_03	1'786'761	9'481	0.5306%
wdvc16_2014_05	3'388'967	11'004	0.3247%
wdvc16_2014_07	4'159'129	11'217	0.2697%
wdvc16_2014_09	3'997'261	8'133	0.2035%
wdvc16_2014_11	3'406'975	10'820	0.3176%
wdvc16_2015_01	3'909'428	14'703	0.3761%
wdvc16_2015_03	4'257'680	10'609	0.2492%
wdvc16_2015_05	3'650'018	7'053	0.1932%
wdvc16_2015_07	3'543'306	5'868	0.1656%
wdvc16_2015_09	4'864'681	7'434	0.1528%
wdvc16_2015_11	7'760'154	6'580	0.0848%
wdvc16_2016_01	9'336'013	9'551	0.1023%
wdvc16_2016_03	7'214'141	10'784	0.1494%
wdvc16_2016_05	10'433'991	11'043	0.1058%
Overall	82'658'091	198'147	0.2397%

Table A.1: Vandalism rate per Wikidata vandalism data set

B

Evaluation Results

This appendix contains further evaluation results. The first section shows the hyper parameter heat maps for SSDenStream with 1-NN for all nine G2 data sets. The second section contains the precision and recall line charts showing the results of different graph approaches on the G2 data sets. Additionally, all confusion matrices are listed. The third section shows the confusion matrices of SVM run on the G2 and HUTR2 data sets.

B.1 Hyper Parameter Analysis

This section contains the hyper parameter heat maps for SSDenStream with 1-NN which was run on all nine G2 data sets.

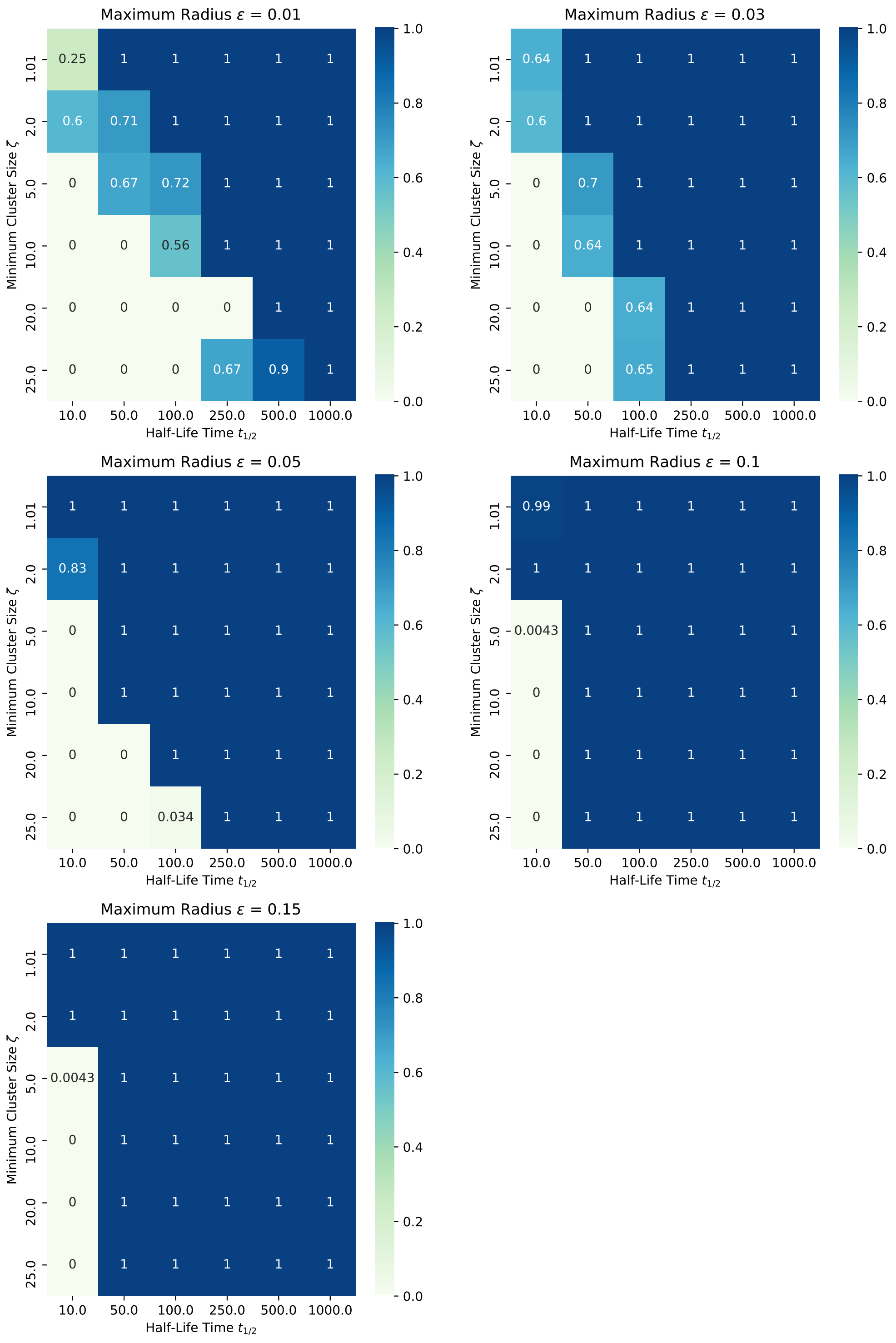

Figure B.1: F1 scores for all hyper parameters combinations of the grid search done on G2-2-10 with k-NN | $k = 1$

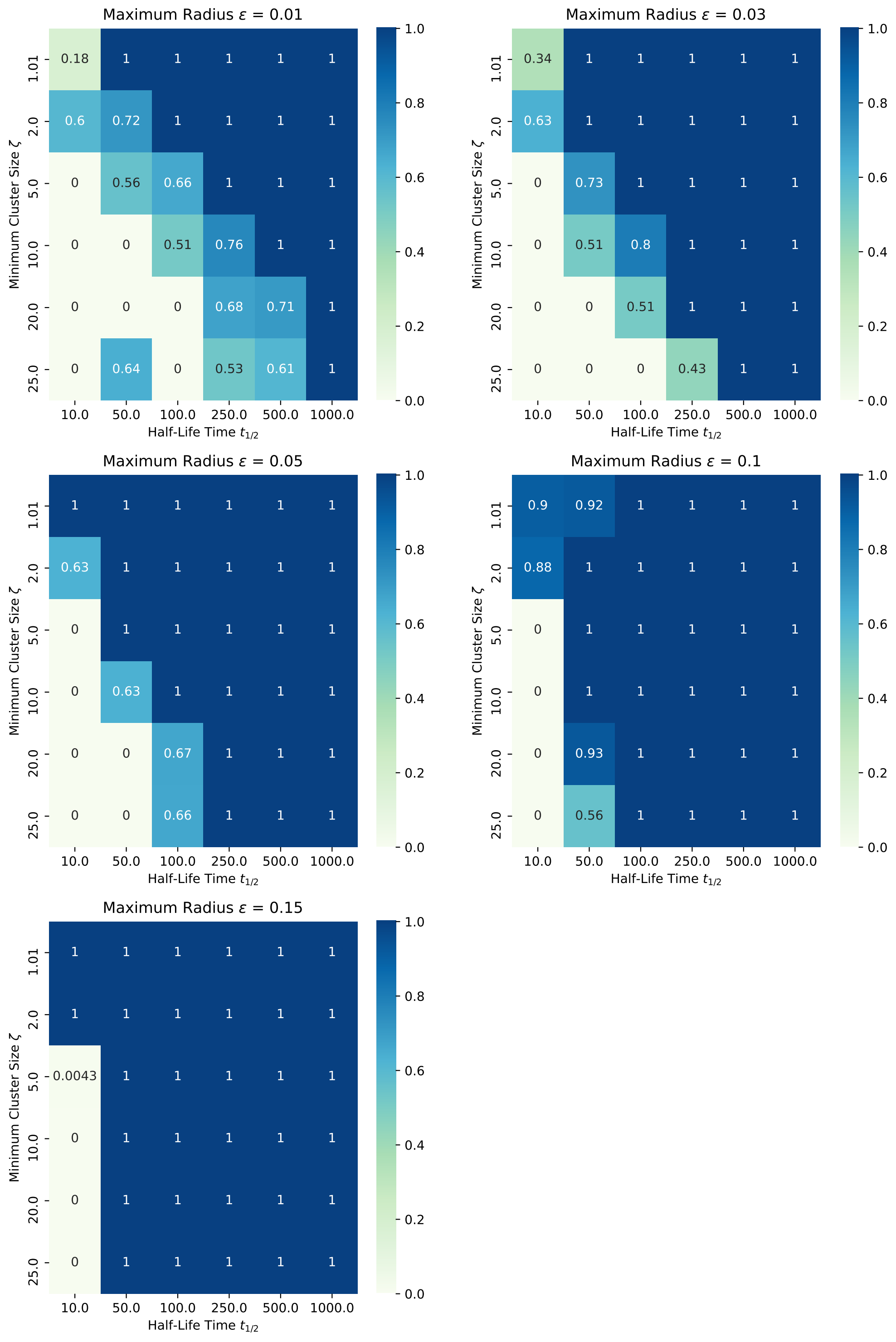

Figure B.2: F1 scores for all hyper parameters combinations of the grid search done on G2-2-20 with k-NN | $k = 1$

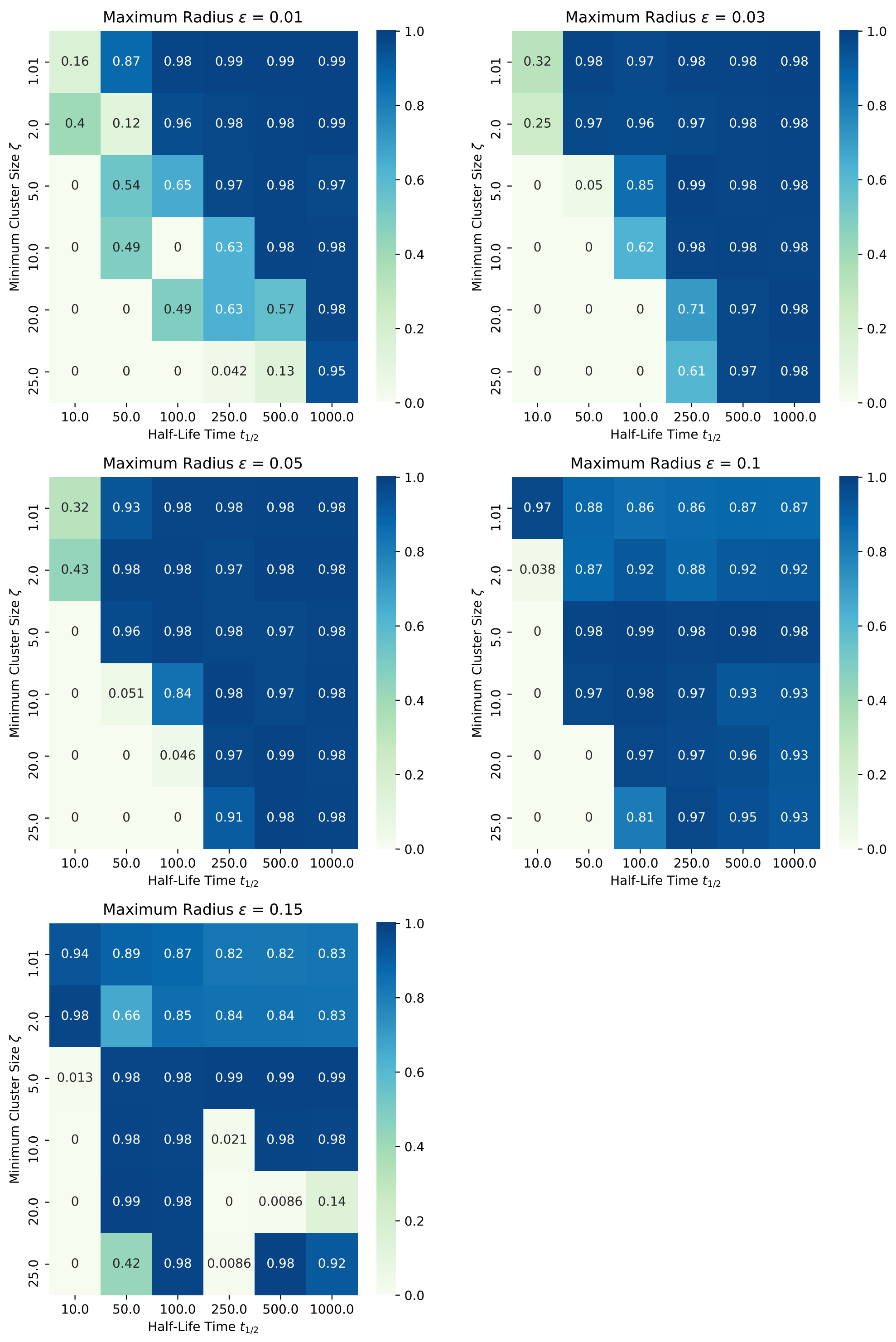

Figure B.3: F1 scores for all hyper parameters combinations of the grid search done on G2-2-30 with k-NN | $k = 1$

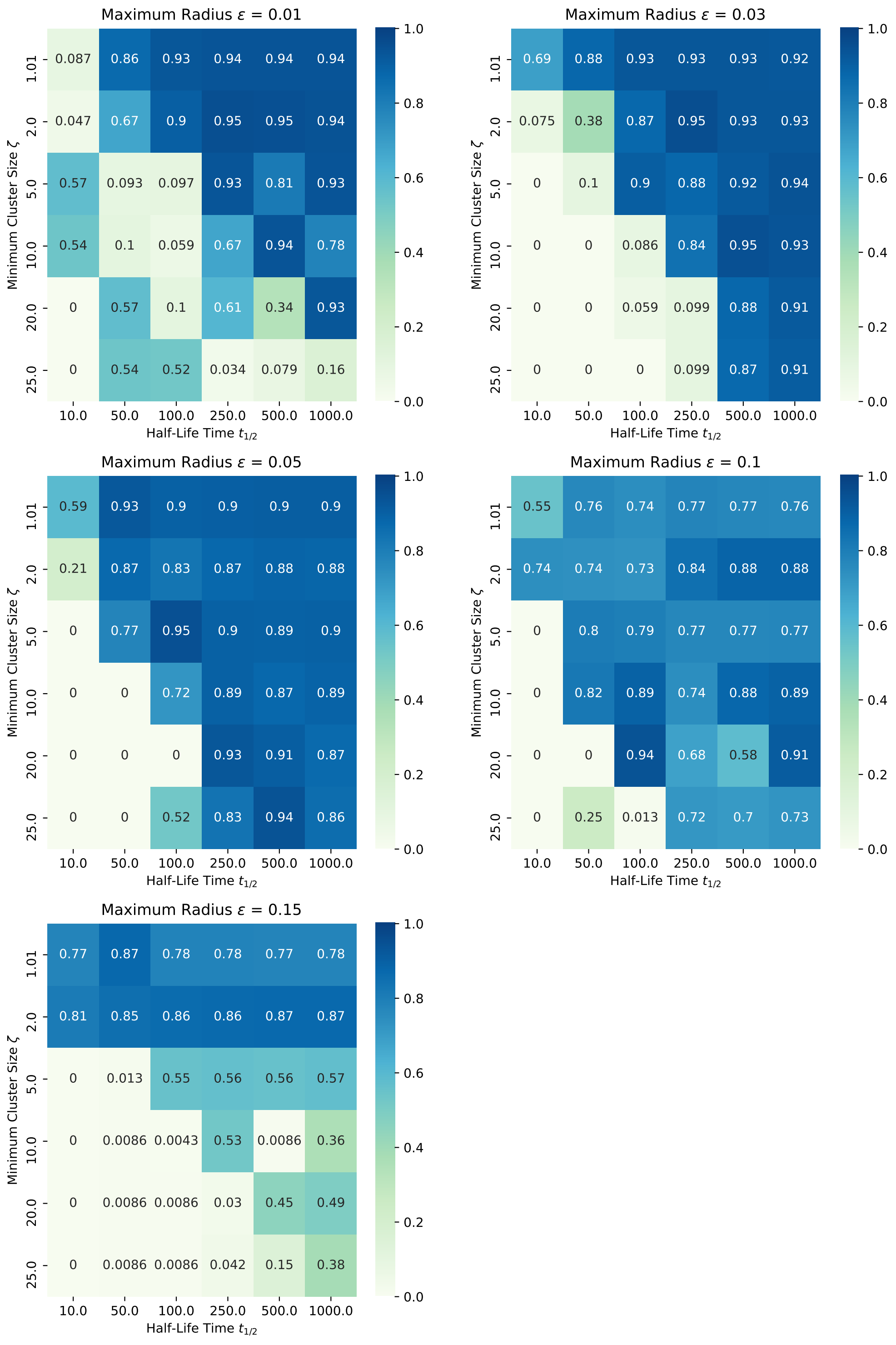

Figure B.4: F1 scores for all hyper parameters combinations of the grid search done on G2-2-40 with k-NN | $k = 1$

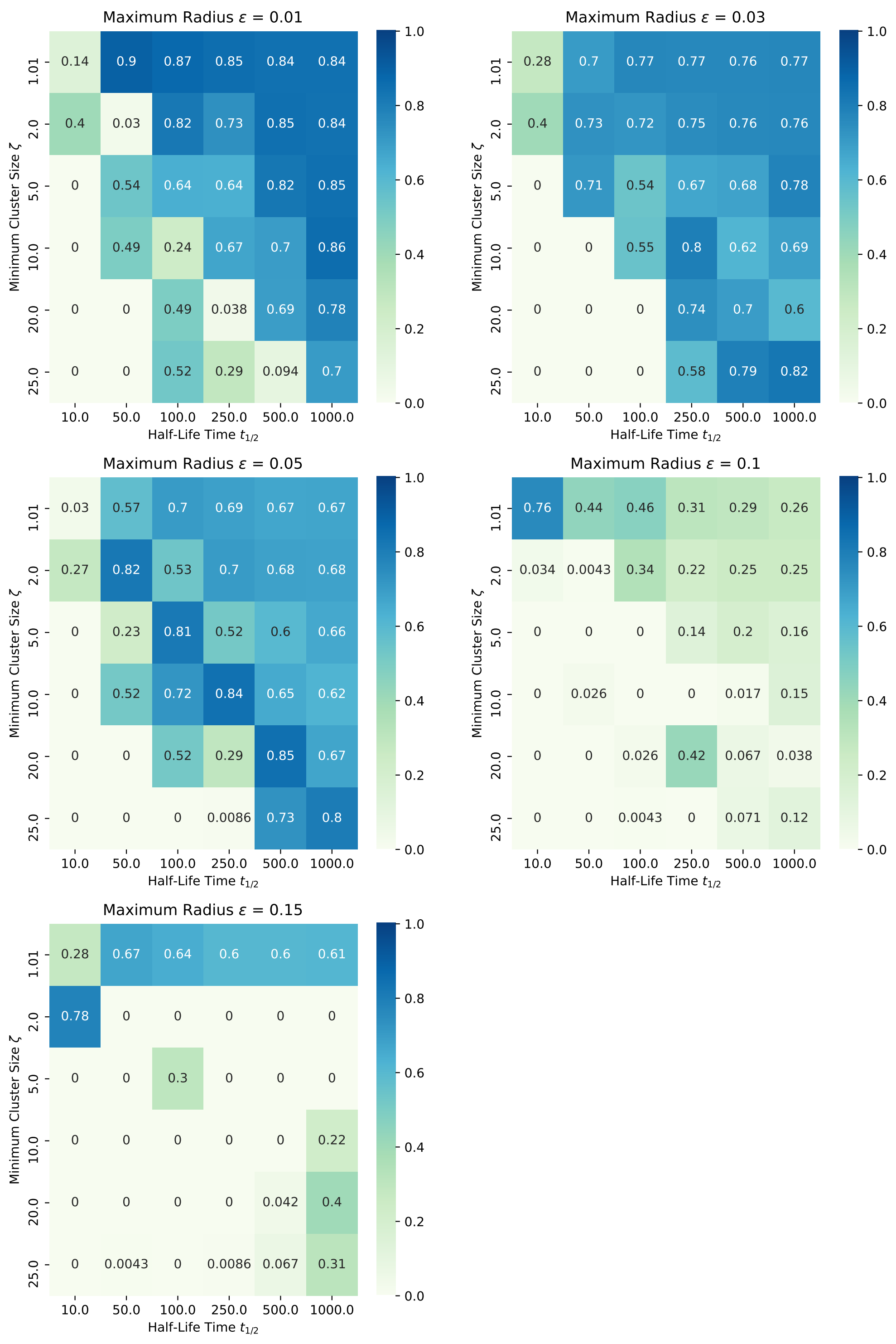

Figure B.5: F1 scores for all hyper parameters combinations of the grid search done on G2-2-50 with k-NN | $k = 1$

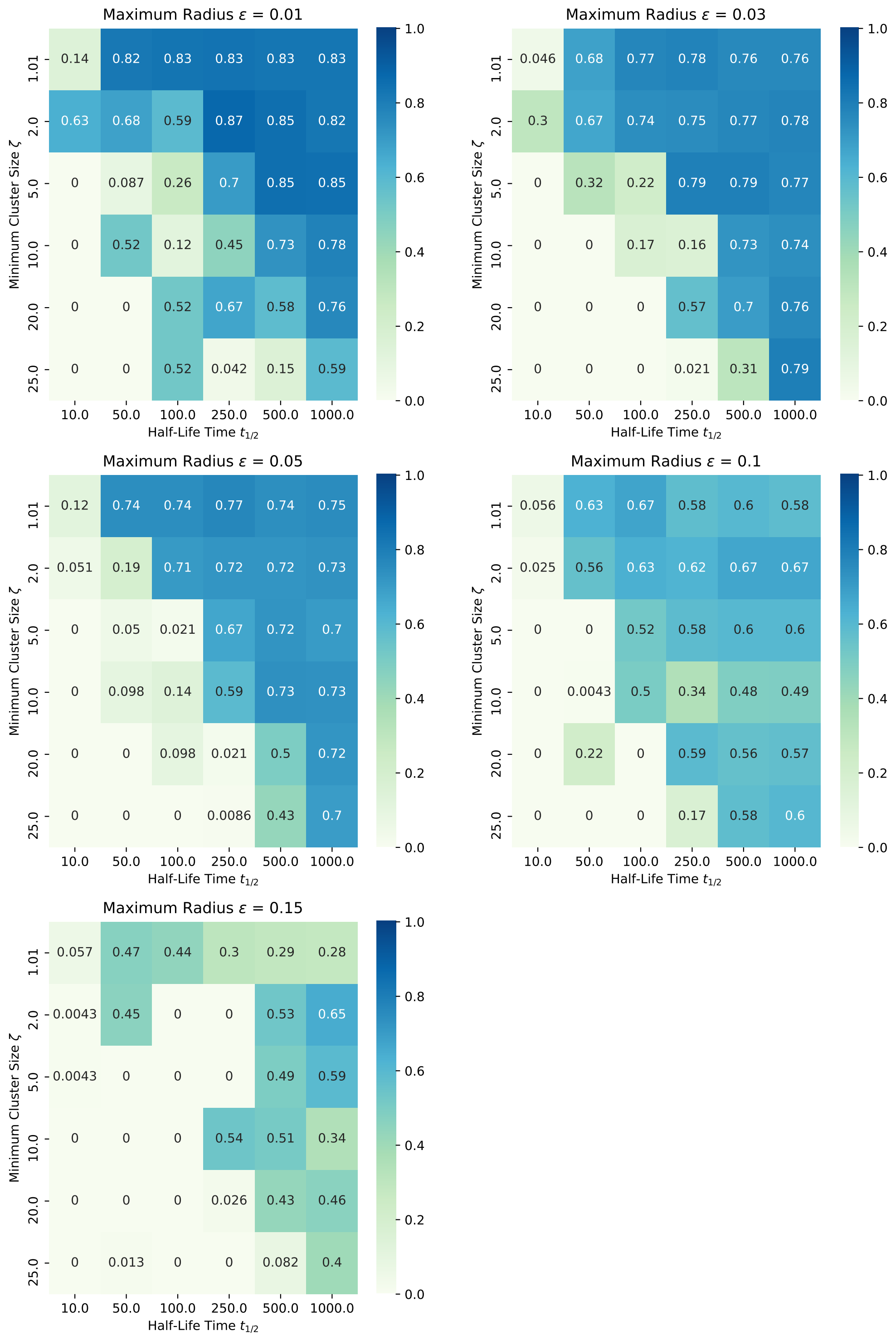

Figure B.6: F1 scores for all hyper parameters combinations of the grid search done on G2-2-60 with k-NN | $k = 1$

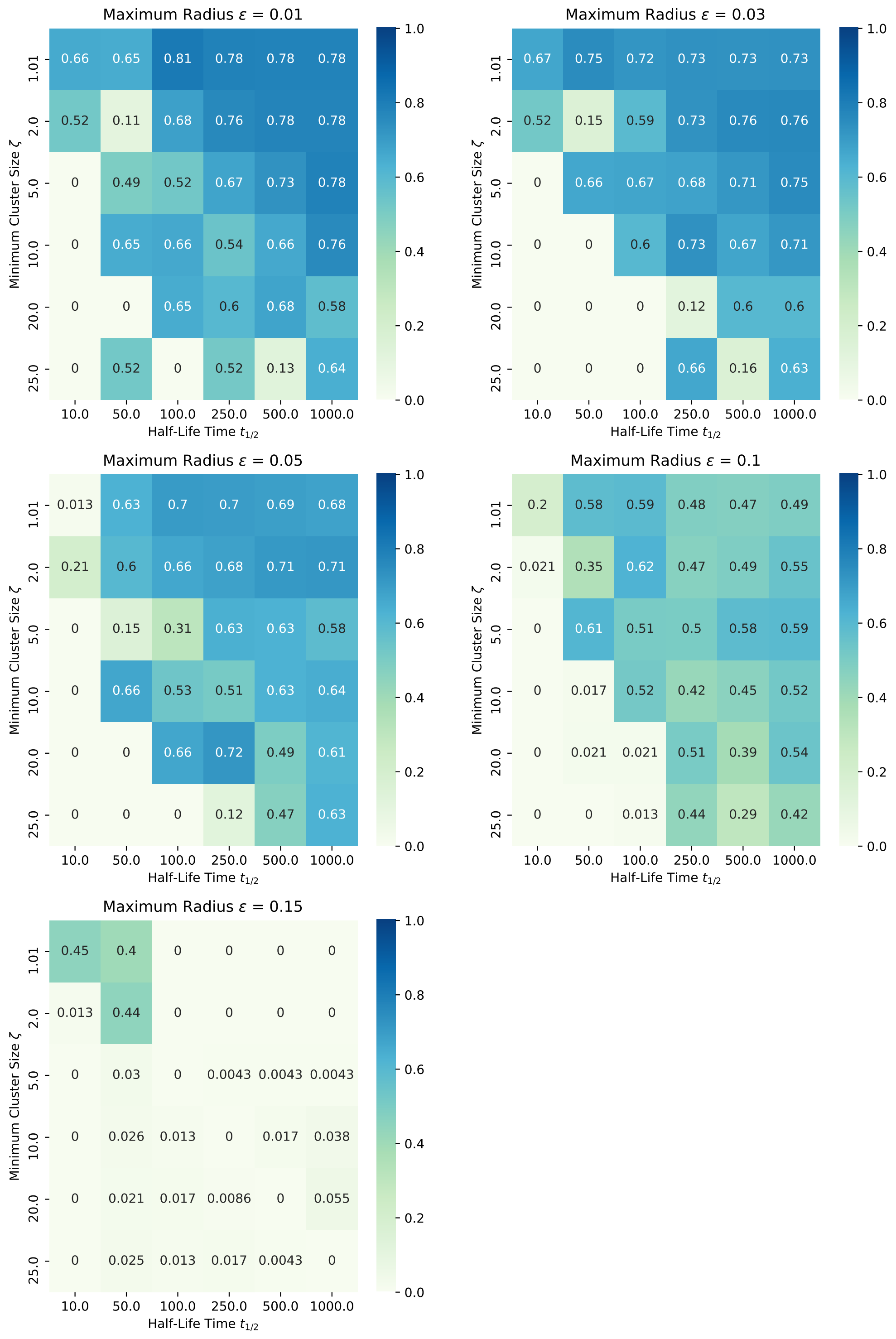

Figure B.7: F1 scores for all hyper parameters combinations of the grid search done on G2-2-70 with k-NN | $k = 1$

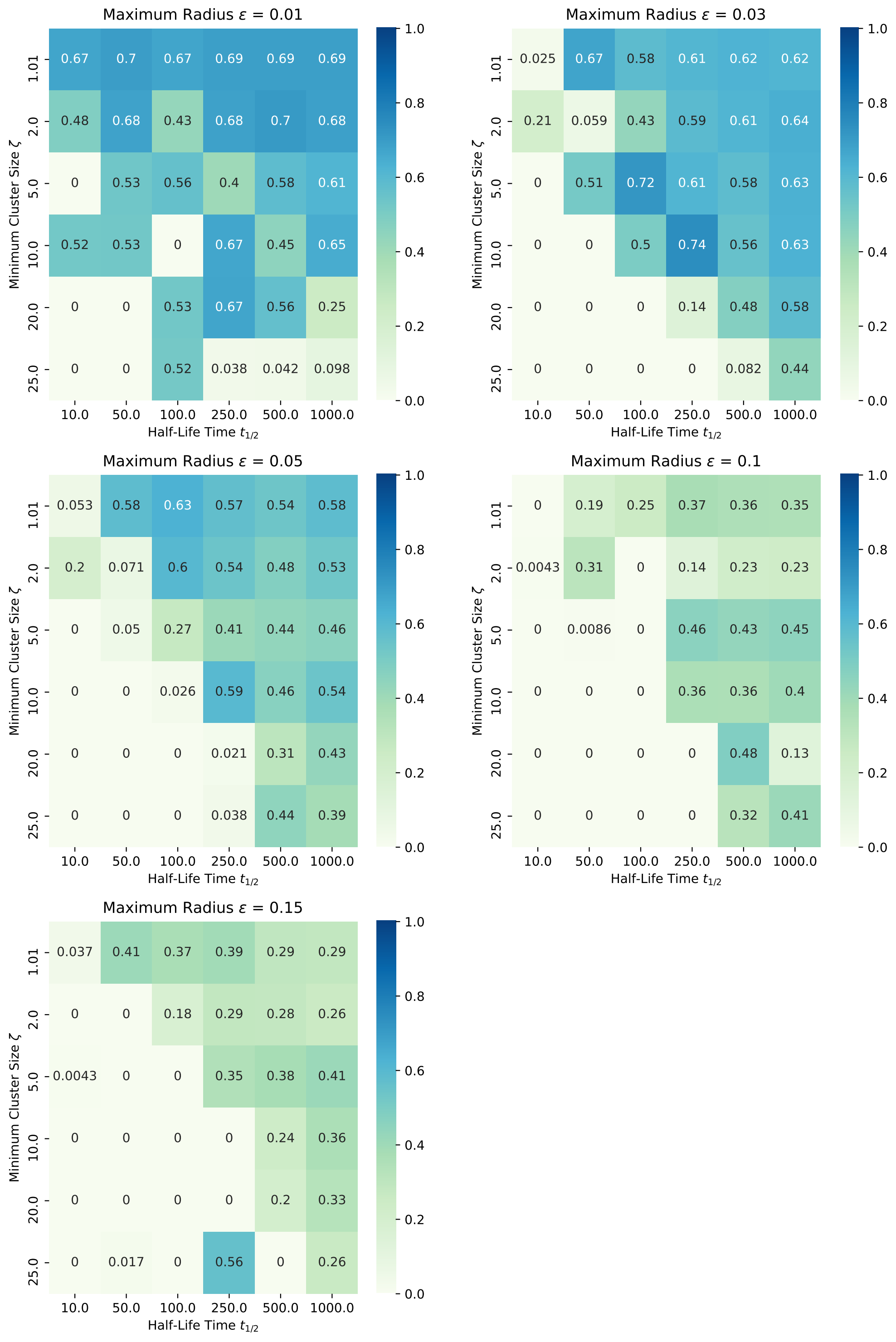

Figure B.8: F1 scores for all hyper parameters combinations of the grid search done on G2-2-80 with k-NN | $k = 1$

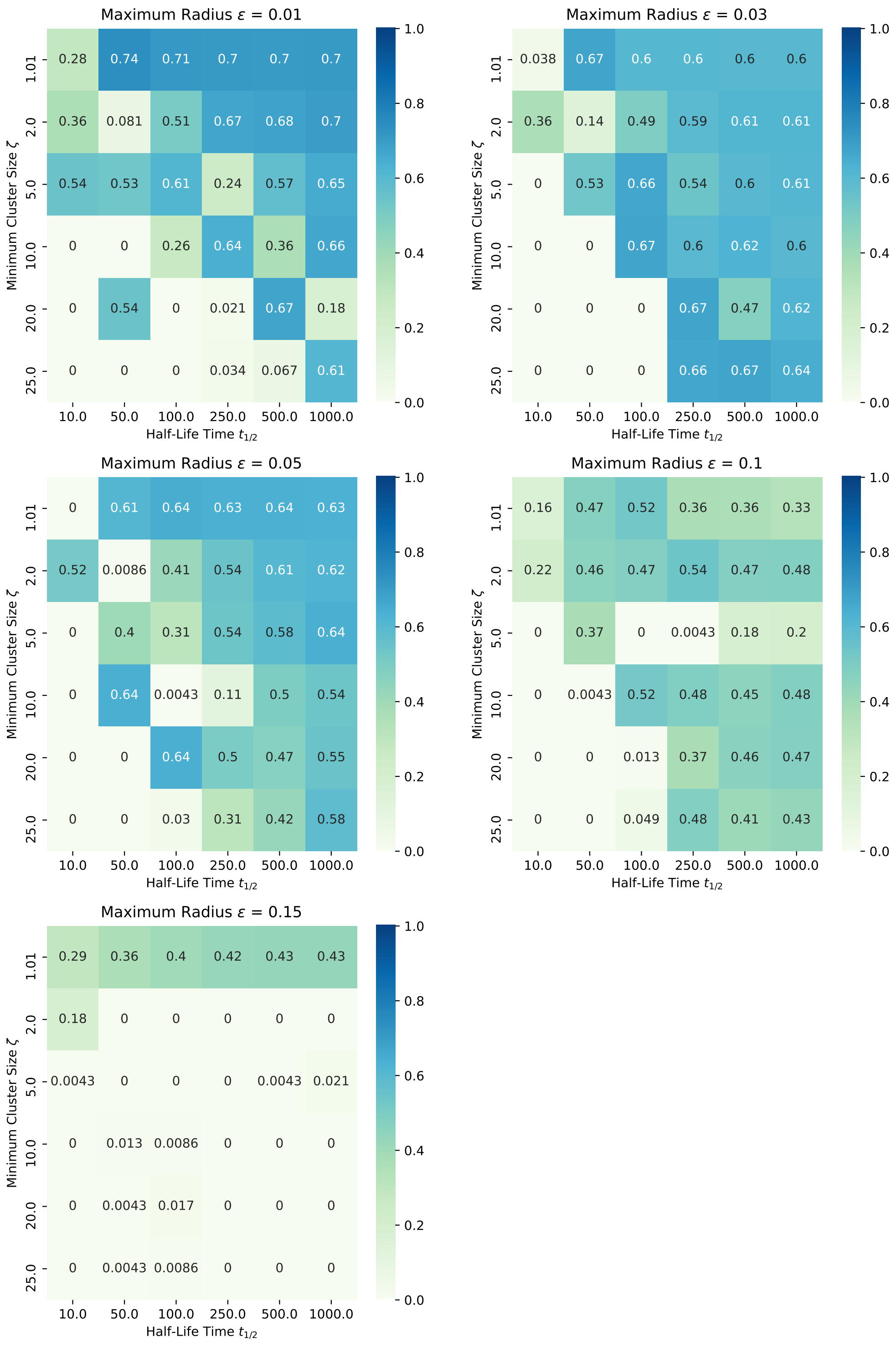

Figure B.9: F1 scores for all hyper parameters combinations of the grid search done on G2-2-90 with k-NN | $k = 1$

B.2 Comparing the Influence of MST and k-NN

This section contains the performance results of different graph approaches on the G2 data sets in form of precision, recall and the confusion matrices.

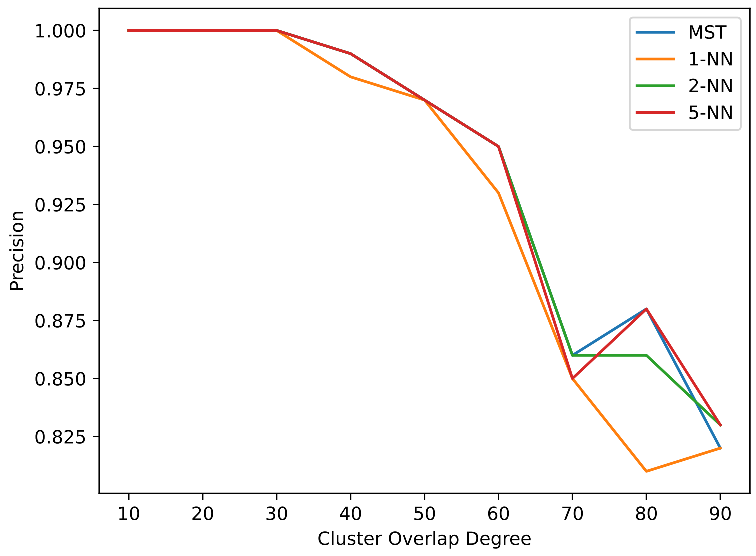

Figure B.10: Precision for different graph approaches on the G2 data sets

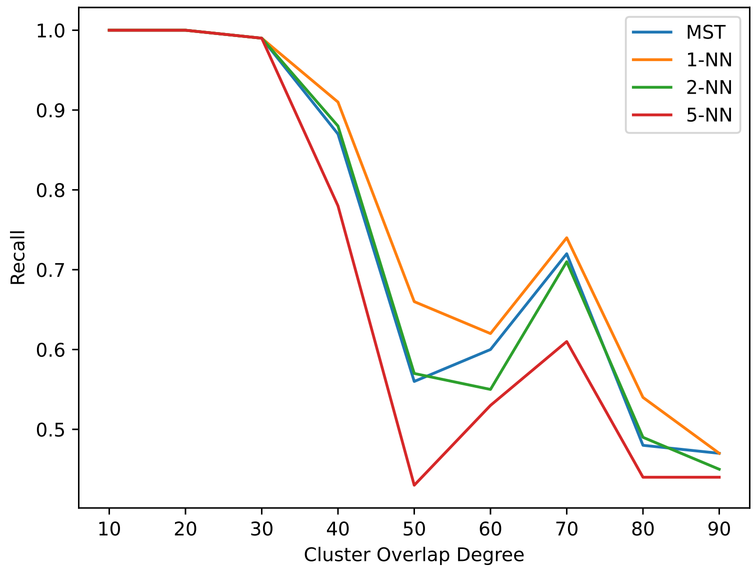

Figure B.11: Recall for different graph approaches on the G2 data sets

		MST		1-NN		2-NN		5-NN	
		0	1	0	1	0	1	0	1
G2-2-10	0	462	0	462	0	462	0	462	0
	1	0	462	0	462	0	462	0	462
G2-2-20	0	462	0	462	0	462	0	462	0
	1	0	462	0	462	0	462	0	462
G2-2-30	0	460	2	460	2	460	2	460	2
	1	5	457	5	457	5	457	5	457
G2-2-40	0	457	5	455	7	457	5	457	5
	1	61	401	41	421	55	407	101	361
G2-2-50	0	454	8	453	9	455	7	455	7
	1	201	261	158	304	199	263	265	197
G2-2-60	0	446	16	440	22	450	12	450	12
	1	183	279	176	286	208	254	216	246
G2-2-70	0	407	55	404	58	408	54	413	49
	1	131	331	122	340	133	329	180	282
G2-2-80	0	431	31	402	60	425	37	433	29
	1	239	223	212	250	237	225	257	205
G2-2-90	0	414	48	414	48	418	44	419	43
	1	245	217	244	218	252	210	258	204

Table B.1: Confusion matrices for the G2 data sets for different graph configurations in SSDenStream

B.3 Comparison to State of the Art

This section contains the confusion matrices showing the performance of SVM on the G2 and HUTR2 data sets.

		SVM	
Data Set		0	1
G2-2-10	0	462	0
	1	0	462
G2-2-20	0	462	0
	1	0	462
G2-2-30	0	456	6
	1	2	460
G2-2-40	0	445	17
	1	14	448
G2-2-50	0	434	28
	1	39	423
G2-2-60	0	407	55
	1	49	413
G2-2-70	0	371	91
	1	37	425
G2-2-80	0	374	88
	1	80	382
G2-2-90	0	347	115
	1	93	369

Table B.2: Confusion matrices for the G2 data sets classified by SVM

		SVM	
Data Set		0	1
HUTR2	0	8094	35
	1	127	693

Table B.3: Confusion matrix for the HUTR2 data set classified by SVM

List of Figures

List of Tables

List of Algorithms