Detecting Rectangular Hotspots in Offline and Streaming Models

Chen, Yuxin

Award date:
2021

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Algorithms, Geometry and Applications Group

Detecting Rectangular Hotspots in Offline and Streaming Models

Master Thesis

Yuxin Chen

Supervisors:
Morteza Monemizadeh

Eindhoven, November, 2021

Abstract

Hotspots are parts of data that are denser (or hotter) than their surrounding sparse (or cool) neighborhoods. In real world scenarios when data is constantly changing, hotspots form, fade and change rapidly. Often retail companies, ride hailing services, financial services and airlines are interested in identifying and tracking hotspots of their data. In particular, they study hotspots to classify and forecast significant changes in their customer shopping patterns. This in turn helps them to adjust their production and marketing plans accordingly.

We model hotspots as a set of points that are generated from a mixture of uniformly distributed (axis-alinged) rectangles and a background noise (which could be a significant portion of data) and our goal is to efficiently and approximately find these rectangular hotspots. We propose two efficient algorithms for detecting hotspots. Indeed, we test these algorithms on synthetic as well as real datasets and we show that the precision, recall and the F-score of these algorithms are very high (often, above 75%). Our first algorithm is based on random sampling pairs of points and our second algorithm is an extension of the classical DBScan algorithm [13]. We show that our former algorithm is very fast, but the latter one has better performance.

We also apply our algorithms in the streaming setting where points are given in a streaming fashion and we are interested in detecting hotspots that are forming, fading and changing dynamically.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation and Problem Statement

A common scenario in cloud computing[1] and streaming[2] is that there is a big network of small devices sending their current state (such as their position in terms of latitude and longitude, battery charge level, and so on) to the cloud. Servers in the cloud analyze the information that is gathered from all devices and inform devices what they should do next. Any sudden change in a big set of devices could potentially be interesting. As an example, let us see how ride-hailing services such as Uber and Lyft operate say in Manhattan New York[3]. Uber cars and cabs keep sending their locations (latitude and longitude), pick-up and drop-off dates/times, pick-up and drop-off locations to Uber servers. The servers continuously monitor these incoming data and if a large set of ride requests coming from somewhere in Manhattan inform Uber cars to drive to nearby locations in order to have a higher chance of receiving a request.

As an example suppose that many soccer fans are watching World cup final 2018 at sports bars. One can guess that French and Croatian restaurants and bars[4] in New York were packed watching the final and right after the match, fans would like to leave bars and take Uber and Lyft cars and cabs going back home. Soon there will be few hotspots around these bars and Uber and Lyft services want to provide services as fast as they can. Ride-hailing services monitor the number of requests that are coming from these bars and detect the surrounding areas of these hotspots and inform drivers to drive and wait around those hotspots. In reality, small hot areas of people form and then they get cooler as fans start moving to neighbor blocks since they think they have a better chance of getting a ride if they walk a few blocks. All these things happen in a very short amount of time.

Almost the same scenario can happen in many real-life phenomena such as

- when a computer crashes and this failure[5] spreads out across the whole network [22],
- traffic congestion after an accident is spreading to the surrounding streets, sensors of cars [18] that are in the vicinity, and of that traffic starts sending data to servers in the cloud, or
- when a large portion of the population of a city subscribes to a new service that a competitor retail company[6][7] provides or unsubscribe an old service that a retail company provides.

In fact, there is a study related to analyzing the hotspot cases mentioned above, which is on the real-time hotspot detection at Amazon Kinesis Analytics[8]. Amazon Kinesis is a software that provides streaming algorithms to aggregate statistics of data that is given in a streaming fashion. It makes it easier for users

[1]https://aws.amazon.com/
[2]https://aws.amazon.com/kinesis/
[3]http://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/
[4]https://www.amny.com/things-to-do/world-cup-final-viewing-parties-nyc-1.19767224
[5]https://en.wikipedia.org/wiki/Cascading_failure
[6]https://www.retaildive.com/news/how-amazon-and-walmarts-head-to-head-competition-is-changing-retail/595762/
[7]https://www.infoentrepreneurs.org/en/guides/understand-your-competitors/
[8]https://aws.amazon.com/blogs/aws/real-time-hotspot-detection-in-amazon-kinesis-analytics/

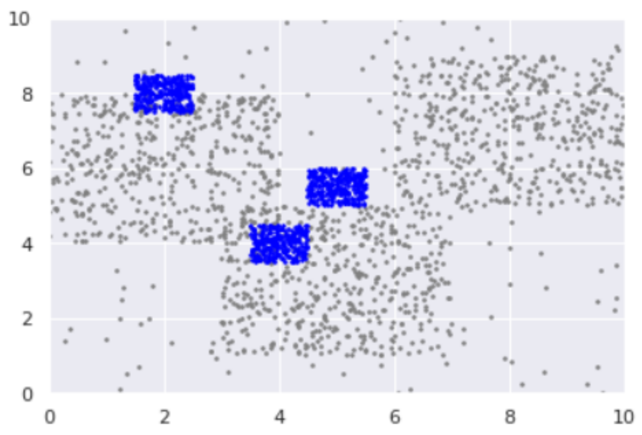


Figure 1.1: Illustration of the mixture of hotspots and background noise. Hotspot points are in blue and the noise points are in gray.

to write SQL queries with less domain knowledge to derive information from data. It is able to detect the subsection of data that needs attention right in time.

In all these scenarios, the big question is how to find areas of hotspots and how fast we can detect that new hotspots are forming or old hotspots are fading, shrinking or expanding. We model the hotspot problem as a mixture of hyper-rectangular regions that are endowed with uniform distributions and this mixture is mixed with a noise which is itself a distribution that is defined for the whole space $\mathbb{R}^d$. Our goal is to extract the hotspots (approximately) when the background noise could be a big portion of this mixture. An example of it could be depicted by the rectangle dataset in Figure 1.1.

Avoding Gerrymandering Phenomenon. One may think that hotspots can have arbitrary shapes and so, it may be hard to detect them. Although this is a fair argument, we think due to the following reasons, reporting simple-shape hotspots is worth exploring.

- Hotspots that are basic geometric plane shapes such as circle, triangle, rectangle, rhombus, square and trapezoid are easy to explain and interpret.
- Simple-shape hotspots can avoid *overfitting problem*[9]. Complex hotspots often correspond too closely to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.
- In geographical maps, often residence blocks/neighborhoods or streets are simple shapes such as a rectangle or a square.

Therefore, we think avoiding the gerrymandering phenomenon is a good way to handle hotspots. The gerrymandering phenomenon– drawing political boundaries to give your party a numeric advantage over an opposing party– is a classic example to explain an unfair election. In this phenomenon, one (arguably in an unfair manner) changes the result of an election to the opposite by setting boundaries of electoral districts into a more complex way. For example, two subgraphs on the left in Figure 1.2 depicted that the blue party will win the election, by either the votes in total or separating the electoral districts evenly in a rectangular shape. Nevertheless, the third case in Figure 1.2 provides a different means to divide the electoral districts in order to make the red party win, which means the minority wins. The same situation could happen when detecting hotspots, when the blue party represents the correct hotspot points and the red ones are the non-hotspot points, shapes of high complexity will end up in a case like the third subgraph in Figure 1.2, a wrong prediction of hotspots. The above case tells that shapes of simple complexity provide results that can explain a phenomenon better and hopefully better handle the overfitting problem[10]. To this

[9]https://en.wikipedia.org/wiki/Overfitting
[10]https://en.wikipedia.org/wiki/Overfitting

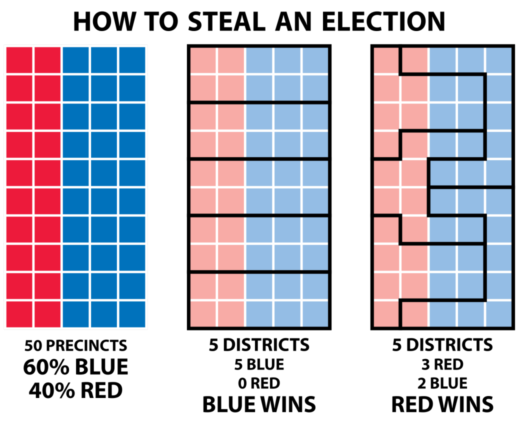


Figure 1.2: Illustration of the gerrymandering phenomenon in the US election [16]. From left to right, the first plot shows the actual votes, the second one shows a fair division of electoral districts, and the third one divides the electoral districts to steal the election from the blue party.

end, in this thesis, we focus on detecting hotspots that are rectangles and we think these types of hotspots can resemble residence blocks or streets in a geographical map.

Hotspot Definition. More formally, for a given parameter $k \in \mathbb{N}$, let $\mathcal{D} = w_1\mu(H_1) + \cdots + w_k\mu(H_k) + w_{k+1}\mu(N)$ be a mixture of a set of axis-aligned rectangular hotspots $H_1, \ldots, H_k \subseteq N$ and a background noise $N \subseteq \mathbb{R}^d$ that is also a rectangle. Here, $w_1, \ldots, w_{k+1}$ are the mixing weights with $\sum_{i=1}^{k+1} w_i = 1$ and k is a natural number. We denote by $vol(H)$ the volume of a rectangle $H \subseteq \mathbb{R}^d$, which is area in the two-dimension case. In the d-dimensional Euclidean space $\mathbb{R}^d$ a rectangle is in fact a hyper-rectangle, but abusing the notations, we use the word rectangle for hyper-rectangles even in $\mathbb{R}^d$.

Intuitively, when we think of hotspots of data, we think of regions of data that are denser and much smaller than their sparser and bigger surrounding neighborhoods. Also, hotspots shall not cover whole data, otherwise, there is no point of reporting hotspots. Moreover, hotspots should not completely overlap, otherwise it will be very hard (if not impossible) to detect them separately.

We define hotspots as multi-dimensional heavy hitters of data as follows.

Definition 1 ((α, β)-Hotspot)**.** *Let $0 < \alpha, \beta \leq 1$. Let $\mathcal{D} = w_1\mu(H_1) + \cdots + w_k\mu(H_k) + w_{k+1}\mu(N)$ be a mixture of a set of axis-aligned rectangular hotspots $H_1, \ldots, H_k \subseteq N$ and a background noise $N \subseteq \mathbb{R}^d$ that is also a rectangle. We say H_i is an (α, β)-hotspot for the mixture $\mathcal{D}$ if*

1. ***Dense Enough:** $w_i \geq \alpha$, and*
2. ***Not Completely Overlapped:** for each two α-hotspots H_i and H_j we have $vol(H_i) \cap vol(H_j) \leq \beta \cdot \min(vol(H_i), vol(H_j))$.*

Observe that there are at most $\min(k, \frac{1}{\alpha})$ hotspots for a mixture $\mathcal{D}$. We should mention that we do not try to optimize the parameters of Definition 1 as the focus of this thesis is in defining and capturing hotspot formations as a real-life phenomenon. To some extend the dynamic scenario that we think can capture and explain hotspot forming and fading is based on Maxwell's demon [11].

After defining the (α, β)-Hotspot, we next define the density of a rectangular region. This will help us later once we develop our algorithm, to rank a set of computed candidate rectangles and decide which one could be a potential hotspot.

Definition 2 (Density of a rectangle)**.** *Let $R \subseteq \mathbb{R}^2$ be an arbitrary rectangle. Let $n(R) = |R \bigcap P|$ denote the number of points covered in rectangle R. The area of a rectangle is computed by its width multiplied by its*

[11] https://www.sciencenews.org/article/scientists-peek-inside-mind-maxwells-demon

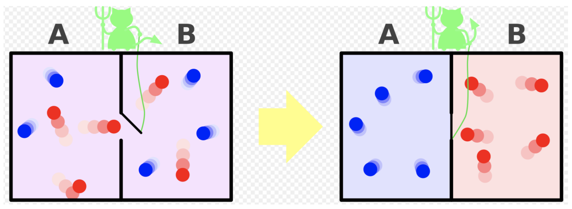


Figure 1.3: Illustration of Maxwell's demon, from https://www.sciencenews.org/article/scientists-peek-inside-mind-maxwells-demon. A demon controls a small massless door between two chambers of gas. As individual gas molecules (or atoms) approach the door, the demon quickly opens and closes the door to allow only fast-moving molecules to pass through in one direction, and only slow-moving molecules to pass through in the other. Because the kinetic temperature of a gas depends on the velocities of its constituent molecules, the demon's actions cause one chamber to warm up and the other to cool down. This would decrease the total entropy of the two gases, without applying any work, thereby violating the second law of thermodynamics.

height, i.e. $\mathrm{area}(R) = w_R \cdot h_R$. *Then, the density of a rectangle that is denoted by* $\mathrm{density}(R)$ *is defined as*

$$\mathrm{density}(R) = \frac{n(R)}{\mathrm{area}(R)} \ .$$

We assume that hotspots cannot be extremely small like a single point of high weight in a 2-dimensional Euclidean space, otherwise, the density of a point is extremely high since its area is almost zero. To avoid these abnormal cases, we assume $\mathrm{vol}(H_i) \geq \gamma \cdot \mathrm{vol}(N)$ for some parameter $0 < \gamma \leq 1$.

There is still one ambiguous place in the concept of hotspots, how to assign points to such regions. The dataset only contains points and their own contributes, efforts should be made to divide regions. In order to divide points into regions, clustering methods are to used.

Furthermore, if the dataset is massive enough, it will take a huge amount of time to process. The solution to it is to adopt a kind of method that is able to receive data as data streams continuously, which is the sliding window method in this thesis.

Our Contribution. We develop two offline algorithms for detecting axis-aligned rectangular hotspots and a streaming algorithm.

1. **Random Sampling:** The first algorithm is a random sampling based algorithm that samples a small subset of pairs of points and for each pair considers the minimum enclosing rectangle that covers the pair. It then ranks the candidate rectangles that are computed in this way and greedily finds and reports those rectangles that could be potential hotspots.

2. **Adapted DBScan:** The second algorithm is an extension of the classic DBScan [13] algorithm. DBScan has two parameters. We discretize the space of these parameters and enumerate these discretized choices. For each such choice, we invoke the DBScan algorithm to report arbitrary clusters and compute enclosing rectangles for the best clusters. The rest of the algorithm would be the same as the first algorithm. That is, we then rank the candidate rectangles that are computed in this way and greedily find and report those rectangles that could be potential hotspots.

3. **Streaming Hotspots:** Given a set of points that are revealed in a streaming fashion, we run either of these algorithms on sliding windows that consist of windows of W latest (i.e., newest) points of the stream, for a parameter $W \in \mathbb{N}$. To speed up the algorithms, we run them on sampled subsets of points that are in the sliding windows.

We test our hotspot detection algorithms on synthetic as well as real datasets. We observe that the random sampling algorithm is very fast compared to the adapted DBScan algorithm. However, the adapted DBScan algorithm has better performance (in terms of precision, recall, and F-score) compared to the random sampling algorithm. We also see that our streaming algorithm performs well in detecting hotspots of data that are forming, fading and changing.

1.2 Related Works

DBScan Clustering As for clustering, density-based clustering is one of the popular clustering methods, and DBScan is the most representative one. The DBScan clustering algorithm was proposed by M. Ester et al.[9] as an efficient solution to find clusters with arbitrary shapes on spatial data and use only one input parameter epsilon (MinPts is fixed). The detailed algorithm is explained in section 2.2. Gan and Tao [10] discussed that though a large amount of time is used to run all DBScan algorithms it can be decreased to $O(n)$ with slight inaccuracy in the clustering results, and DBScan with dimension more than 3 is computationally intractable even on moderately large n. Another further research on work of Gan and Tao is discussed by E. Schubert et al. [21], they claimed that the proposed algorithm in [10] to replace DBScan on big data is not conclusive, and DBScan in [9] with proper parameters and indexes is sufficiently competitive. A faster DBScan algorithm proposed by A. Gunawan in [13] and [7] uses box graphs and computes the algorithm in $O(n \log n)$ time in $\mathbb{R}^2$, where epsilon is to be determined and MinPts is fixed to be 4, details will be shown in section 2.2.2.

DBScan in data streams F. Cao et al. proposed a DBScan algorithm[3], DenStream, to discover arbitrary shapes of clusters in an evolving data stream, it is noise-sensitive as well. The pruning strategy proposed in it to limit the memory consumption is to get rid of the outliers and keep the potential clusters in the meantime. Based on the work of Cao et al., Kumar and Sharma proposed a DDenStream algorithm[14]. It is based on a fading window model where the weight of the data points is deduced over time. It also improves the quality when multiple clusters overlap by using DCQ-Means algorithms.

Agglomerative Clustering Hierarchical clustering (HC) includes two different methods, one of which is from top to bottom, and the other one is from bottom to top. The bottom-up clustering is called agglomerative clustering, the decomposition of it is represented by the dendrogram. It begins with one cluster of all points as the root node of a tree, which is divided into sub-clusters, and the cluster at the leaf node contains only one object. Hierarchical clustering research of Charikar et al.[5] proposed two new hierarchical algorithms that outperform the average-linkage method, a similarity HC and a dissimilarity HC. In [4], they also proposed an algorithm on Euclidean data under vector-based similarity measures which the main focus is on the Gaussian kernel, it outperforms the average-linkage method and can be scaled to high dimensions.

Clustering in the Streaming Model An algorithm, CURE, of clustering for large datasets is proposed by Guha et al.[12]. CURE can identify non-spherical and even other shapes of clusters and can process large databases by using random sampling and partitioning. But it is not able to deal with data streams. Another work from Guha et al.[11] explained a clustering algorithm that can handle large data streams to use less memory.

Chapter 2

Preliminaries

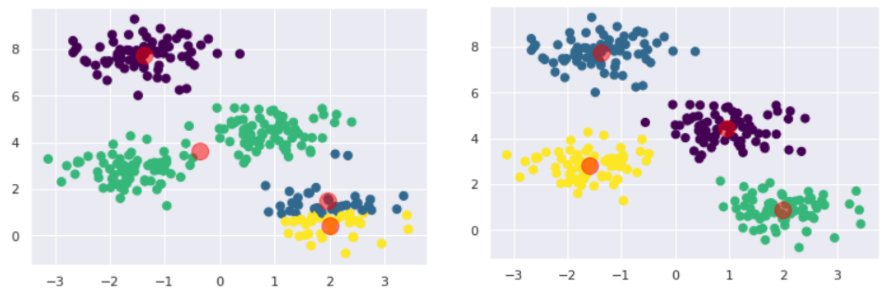
In this chapter, we define the notations that we use through this thesis. We also explain the clustering algorithms that we use and implement in this thesis. In particular, we define the k-means algorithm [15] and the k-means++ clustering [2]. We later introduce the known DBScan clustering [9] and its fast implementation that is known as DBScan with Box-graphs [13]. Later, we define the sliding window model. To compare different clustering algorithms we also introduce the notions of precision, recall, and F-score that we define next. A nice way of representing the distribution of data is what is called *Kernel Density Estimation (KDE)* that we define at the end of this chapter. We first expalin the k-means problem and k-means algorithm.

2.1 Centroid Based Clustering

2.1.1 k-means

The k-means clustering algorithm is a classic centroid-based clustering algorithm, which refers to Lloyd's algorithm [15]. It defines a cluster by a center point, also called centroid, therefore each cluster is defined by a cenroid and a set of points that are close to the centroid. The main idea of k-means algorithm is to minimize the squared Euclidean distance between cluster points and their nearest centroids.

Let $P \subseteq \mathbb{R}^d$ be a set of points of size n given in a d-dimensional Euclidean space and a parameter $k \in \mathbb{N}$. In the k-means clustering, we would like to report k center points in $\mathbb{R}^d$ such that the sum of squared Euclidean distances of each point to its nearest center is minimum. Suppose we have the optimal centers $\mathcal{C}^* = \{c_1^*, \cdots, c_k^*\}$, then in the k-means problem we assign each point to its nearest center in $\mathcal{C}^*$. Thus, we can define the cluster C_i^* for $i \in \{1, 2, \cdots, k\}$ as all the points that are closer to the center c_i^* than the other



(a) An example when the initialization of centroids is random, so the clusters are not divided as expected. (b) Good clustering result can be obtained after a good centroid initialzation.

Figure 2.1: Illustrations of k-means clustering when the initialization of the centroids is random, results could be different. Clusters are in different colors, red points are the centroids.

centers, where the ties are broken arbitrarily. In this way, we can say the center c_i^* is the center of the cluster C_i^*. It is known that c_i^* is the center of mass or the center of gravity of the cluster C_i^*. That is,

$$c_i^* = \frac{\sum_{p \in C_i^*} p}{|C_i^*|} \quad .$$

where $|C_i^*|$ denotes the number of points in the i-th cluster.

Since the optimal clustering have the minimum sum of squared Euclidean distances of all points to their assigned centroids, the sum of the distances of the i-th cluster $\mathtt{cost}(C_i^*)$ is introduced as the cost,

$$\mathtt{cost}(C_i^*) = \sum_{p \in C_i^*} \| p - c_i^* \|_2^2 \quad .$$

Thus, for each cluster the optimal centroid should have the minimal distances to all of its assigned points, that is,

$$c_1^*, \cdots, c_k^* = \arg\min \sum_{j=1}^{k} \mathtt{cost}(C_j^*) = \arg\min \sum_{j=1}^{k} \sum_{p \in C_j^*} \| p - c_j \|^2$$

The pseudo-code in Algorithm 1 explains the whole process. The algorithm is initialized by randomly choosing k points $\mathcal{C} = \{c_1, \ldots, c_k\}$, $\mathcal{C} \subseteq P$, as the set of centroids. Then, each point $p \in P$ is assigned to a centroid closest to p. After all points in P are assigned, the centroid of each cluster will be updated. The process repeats assigning each point p to the new centroids that are nearest to p and computing new centroids again, then stops when all k clusters converge. At the end, we have the optimal clusters $\{C_1^*, \cdots, C_k^*\}$.

Algorithm 1 KMEANS(k, P)

```
Require: k: the number of required clusters; P: set of data points;
Ensure: every point is assigned to one of k clusters, the distances between points and their assigned
    centroids are minimized.
 1: Initialize cluster centroids $c_1, c_2, \ldots, c_k \in \mathbb{R}^2$ randomly;
 2: repeat
 3:     for each $i \in \{1, \ldots, n\}$ do
 4:         find the closest centroid for point $p_i$, $\text{label}_i = \arg\min_{j \in \{1, \cdots, k\}} \| p_i - c_j \|_2^2$;
 5:         assign point $p_i$ to its nearest centroid, $C_{\text{label}_i} = C_{\text{label}_i} \bigcap p_i$;
 6:     end for
 7:     for each $j \in \{1, \ldots, k\}$ do
 8:         compute a new centroid for every cluster after assigning points to their closest centroids,
            $c_j = \sum_{i \in \{1, \ldots, n\} p_i \in C_j} p_i / |C_j|$;
 9:     end for
10: until the cluster assignment converges.
```

The shortcoming of it is obvious that the random initialization of the centroids could lead to bad clustering as shown in Figure 2.1(a), since the result of the k-means algorithm relies heavily on the initialization of the centroids at the beginning.

2.1.2 k-means++

The k-means++ algorithm [2] adapts the k-means algorithm on the initialization of the centroids. Instead of choosing all the centroids randomly at one time, it picks one point $p \in P$ randomly as the first centroid c_1 and adds it in the set of selected centroids $\mathcal{C}$. Then, the next centroid is randomly chosen with a set of weights, the weight of a point p_i is the proportion of the distance between p_i and its closest centroid in $\mathcal{C}$ to the sum of the distance between all points and their closest centroids in $\mathcal{C}$. Each weight can be computed by

$$\text{weight}_i = \frac{\text{dist}_{min}^i}{\sum_{i=1}^{n} \text{dist}_{min}^i} \quad ,$$

where dist^i_{min} is the notion of the distance between each point $p_i \in P$ and its nearest centroid $c_j \in S$, and it is computed by

$$\mathrm{dist}^i_{min} = \min_{j=\{1,\dots,k\}} \mathrm{dist}_{p_i,c_j} = \min_{j=\{1,\dots,k\}} \| p_i - c_j \|_2^2 \quad .$$

According to the function of weights, it is observed that the further a point is to its closest centroid in $\mathcal{C}$, the bigger chance a point has to be chosen as the next centroid. It repeats appending centroids into $\mathcal{C}$ using weights computed in the way described before until there are already k centroids in $\mathcal{C}$. The resulting set of centroids $\mathcal{C}$ is regarded as the optimal set of centroids, so $\mathcal{C}^* = \mathcal{C}$. Then it runs the k-means algorithm with the optimal centroid set $\mathcal{C}^* = c^*_1, \dots, c^*_k$ as the initial centroid set, the pseudo-code is shown in Algorithm 3.

Algorithm 2 CENTROIDINITIALIZATION(k, P)

```
Require: $k$: the number of required clusters; $P$: set of data points;
Ensure: $\mathcal{C}^*$, a set of $k$ optimal centroids
 1: distance $\leftarrow$ empty list;
 2: pick one point $p \in P$ as the first centroid $c_1$ in a uniformly random way, append it in the set of centroids
    $\mathcal{C} = \{c_1\}$;
 3: repeat
 4:     for each $i \in \{1, \dots, n\}$ do
 5:         for each $c_j \in \mathcal{C}$ do                    $\triangleright$ the number of centroids in $\mathcal{C}$ will increase.
 6:             compute the distance $\mathrm{dist}_{p_i,c_j} = \| p_i - c_j \|^2$ between $p_i \in P$ and $c_j \in \mathcal{C}$;
 7:             $d^i_{min} = \min(\mathrm{dist}_{p_i,c_j}, d^i_{min})$;
 8:         end for
 9:         append the distance $d^i_{min}$ between point $p_i$ and its nearest centroid to distance;
10:     end for
11:     choose the next centroid with the probability of $\frac{d^i_{min}}{\sum_{i=1}^n d^i_{min}}$;
12: until $|\mathcal{C}| = k$
```

Algorithm 3 KMEANSPLUSPLUS(k, P)

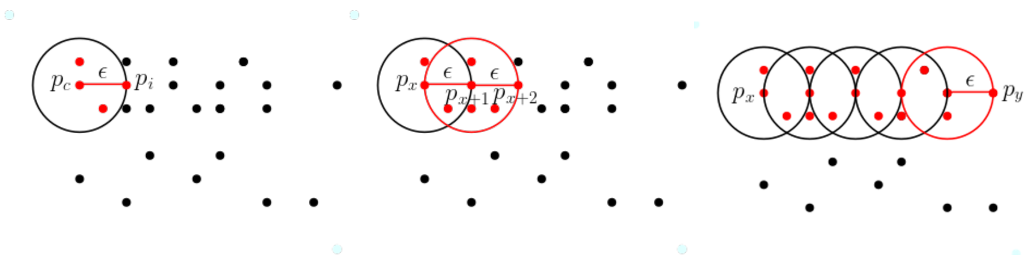
```
Require: $k$: the number of required clusters; $P$: set of data points;
Ensure: correct labeled and clustered points
 1: invoke CENTROIDINITIALIZATION($k$, $P$);
 2: invoke KMEANS($k$, $P$);
```

2.2 Density Based Clustering

2.2.1 DBScan

A representative density-based clustering algorithm is the DBScan algorithm [9]. The purpose of it is to identify points that are in the dense regions, and each region that has sufficiently high density is considered as a candidate for clusters. Moreover, compared to the centroid-based clustering as mentioned in section 2.1, the resulting clustering of DBScan is able to be in arbitrary shapes instead of only linearly separated.

There are two important parameters in DBScan clustering. The important parameters are ϵ and *MinPts*, respectively. A point $p \in P$ is defined as a **CorePoint** if within the distance of ϵ around p there exists at least **MinPts** points (p is also counted). A few concepts should be introduced as well before we explain the DBScan algorithm. A core point p_c is **DirectlyReachable** to a point p_i in P if the distance between them is within ϵ, which is depiected in Figure 2.2(a). For the core point p_x and point p_y, if there is a path $p_x, \dots, p_y$ and each point p_i is directly reachable to p_{i+1}, then p_x is *Reachable* to p_y, as shown in Figure 2.2(c). Other than core points, there are other kinds of points, point p is a **BorderPoint** if there are less than *MinPts* points within distance ϵ from p, otherwise, it is a **NoisePoint**.



(a) A core points p_c is directly reachable to a point p_i, which means that the distance between p_c and p_i is less than or equal to ϵ.

(b) From p_x to p_{x+2}, there is a path $\{p_x, p_{x+1}, p_{x+2}\}$, p_x is directly reachable to p_{x+1}, p_{x+1} is directly reachable to p_{x+2}. So, p_x is reachable to p_{x+2}.

(c) There is a path $\{p_x, \ldots, p_y\}$, and each point p_i is directly reachable to p_{i+1}. So, p_x is reachable to p_y.

Figure 2.2: Directly reachable and reachable.

The abstract process of the DBScan algorithm is described as follows. With points in P and parameters ϵ and *MinPts*, a point $p \in P$ is chosen randomly. If there are *MinPts* points within distance of ϵ from the point p, p is a core point. If p is not a core point, then continue to search for the next point. What comes next is to find all points that are directly reachable from point p and not visited yet, add them into a set of neighbor points of p, denoted as NeighborPts, and assign these points to a new cluster C_1. Next, find directly-reachable points from each of those newly-assigned points in NeighborPts and add them into the set NeighborPts. If these neighbor points in NeighborPts are not assigned to any cluster yet, append them to C_1. Then keep expanding the cluster C_1 until all points that are reachable from p are visited. A new random point is selected to do the same process. The algorithm repeats the above steps until all points are visited.

2.2.2 Boxgraph

Due to the fact that the DBScan algorithm computes the distance between every two points, more time is spent on calculating the edges of the neighborhood graph, a faster DBScan algorithm was proposed to improve that by using box graphs. The faster DBScan algorithm using box graphs costs $O(n \cdot \log n)$ running time in the worst case and needs $O(n \cdot \log n)$ space.

In terms of box graph, vertices and edges should be introduced. Points in P are divided into boxes $\mathcal{B} = \{B_1, \ldots, B_m\}$, each box is regarded as a vertice. When there is a core point in B_1 has a distance within ϵ to a core point in B_2, then B_1 and B_2 are neighbors, an edge between B_1 and B_2 is formed.

The algorithm is composed of four phases: (i) divide points in disjoint boxes; (ii) identify core points; (iii) compute clusters for core points; (iv) assign border points in clusters.

Algorithm 4 COMPUTESTRIPS(ϵ, P)

```
Require: $\epsilon$: within distance $\epsilon$ from a point $p$, if there are *MinPts* points, $p$ is a core point; $P$: a set of points;
Ensure: ordered strips list with width of at most $\epsilon/\sqrt{2}$
 1: sort $P$ by x-coordinates and denote $P = \{p_1, \cdots, p_n\}$ as a set of points in increasing order;
 2: $\mathcal{L} \leftarrow \phi$;
 3: $S \leftarrow p_1$; $left_S \leftarrow x_1$;
 4: for each $i \in [2, n]$ do
 5:     if $x_i \leq left_S + \epsilon/\sqrt{2}$ then
 6:         append $p_i$ to $S$;
 7:     else
 8:     end if
 9: end for
10: append $S$ to $\mathcal{L}$;
11: return $\mathcal{L}$
```

Algorithm 5 COMPUTEBOXES($\epsilon, \mathcal{L}$)

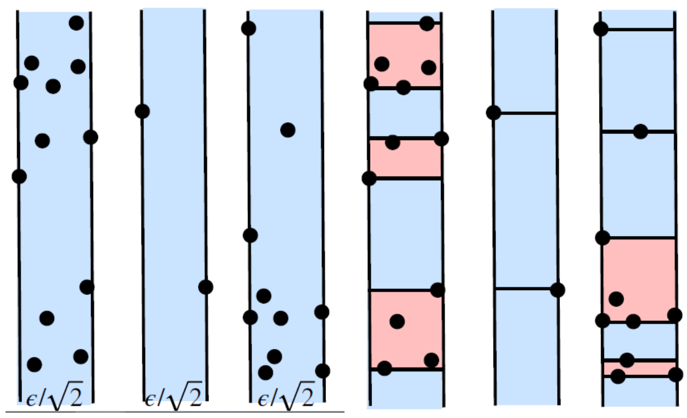
```
Require: $\epsilon$: within distance $\epsilon$ from a point $p$, if there are MinPts points, $p$ is a core point; $\mathcal{L}$: a list of ordered strips;
Ensure: a list of boxes with width and height of at most $\epsilon/\sqrt{2}$
 1: boxes $\leftarrow \phi$;
 2: for each strip $S_i \in \mathcal{L}$ do
 3:     sort points in $S_i$ by y-coordinates and denote $S_i = \{p_i1, \cdots\}$ as a set of points in increasing order;
 4:     $\mathcal{B} \leftarrow \phi$;
 5:     Q $\leftarrow$ a set with only one strip $S_1$; bottom$_Q \leftarrow y_{i1}$;
 6:     for j $\leftarrow$ 2 to |S| do
 7:         if $y_{ij} \leq \mathrm{bottom}_Q + \epsilon/\sqrt{2}$ then
 8:             append $S_j$ to Q;
 9:         else
10:             create box B; $n(B) \leftarrow$ Q where $n(B) = |B \bigcap P|$; rectangle$(B) \leftarrow$ bounding box of Q; neighbors$(B) \leftarrow \phi$;
11:             append B to $\mathcal{B}$; Q $\leftarrow$ a set with $S_j$; $\mathrm{bottom}_Q \leftarrow y_{ij}$);
12:         end if
13:     end for
14:     create box B; $n(B) \leftarrow$ Q; rectangle$(B) \leftarrow$ bounding box of Q; neighbors$(B) \leftarrow \phi$;
15:     append B to $\mathcal{B}$; append $\mathcal{B}$ to boxes;
16: end for
17: return boxes
```

Firstly, points are divided into grids with disjoint rectangular boxes with side length of at most ϵ as in Algorithm 4. Points are sorted by their x-coordinate at the beginning. The leftmost point that has the minimum x-coordinate left$_S$ is denoted as $p_{\mathtt{left}}$, points having x-coordinates within [left$_S$, left$_S + \epsilon/\sqrt{2}$] are divided into a strip of at most $\epsilon/\sqrt{2}$ in width. The starting point of the next strip is the first point $p'_{\mathtt{left}}$ having x-coordinate left$'_S >$ left$_S + \epsilon/\sqrt{2}$, set left$_S =$ left$'_S$. Repeat this until there are points left and the width of the next strip is less than or equal to $\epsilon/\sqrt{2}$, divide all points left into one strip. After all points are divided into strips as in Figure 2.3, strips are divided into boxes of the height of at most $\epsilon/\sqrt{2}$ as in Algorithm 5. Points in one strip are sorted by their y-coordinates. The first box of one strip S_i starts from the bottom point p_b that has the minimum y-coordinate y_b. All points in S_i with y-coordinates within $[y_b, y_b + \epsilon/\sqrt{2}]$ are assigned into S_i. The starting point of the next box $p'_b \in S_i$ is the first point in S_i that has y-coordinate $y'_b > y_b + \epsilon/\sqrt{2}$, so let $p_b = p'_b$. Repeat dividing points in S_i into boxes of at most $\epsilon/\sqrt{2}$ in height until there are points left and the height of the next box is less than or equal to $\epsilon/\sqrt{2}$, divide all points left into one box.

After dividing points in boxes, it is necessary to compute the neighbor of boxes, its pseudo-code is shown in Algorithm 6. It can be observed that neighbors of a box can only be boxes in the two strips on the left and two strips on the right, 5 neighbor strips in total. That is because two boxes are neighbors if they have a distance of at most ϵ which is less than the width of two strips $\sqrt{2}\epsilon$. Then, a sweepline approach is used to find the neighbors for boxes. The sweepline is actually a horizontal line moving from bottom to top, every time it approaches the lower bound of a box, the box and its neighbor are added to the sweepline. For boxes in 5 neighbor strips of a box, all boxes that are within distance ϵ and lower bound of it is below the sweepline will be added to each other's neighbors. If the upper bound of a box is more than ϵ away from the sweepline, the box will be removed from the sweepline.

The second phase is to compute the core points, as in Algorithm 7. The iteration goes over all the boxes, if the number of points of a box is bigger than or equal to *MinPts*, all points in that box are labeled as core points since the distances between any two points in a box are less than ϵ. Else, if the number of points in a box is less than *MinPts*, the points in the box can also be labeled as core points if they have more than *MinPts* neighbors which are also within distance ϵ. The running time of Algorithm 7 is $O(n)$.

In the third phase, clusters are computed, the algorithm is depicted in Algorithm 8. Like the classic DBScan algorithm, if a core point p_1 of B_1 and a core point p_2 of B_2 have distance within ϵ between them,



(a) Points are divided into strips by width of $\epsilon/\sqrt{2}$. (b) Each strip is divided into boxes by the height of $\epsilon/\sqrt{2}$.

Figure 2.3: The points are divided into boxes in the DBScan algorithm using box graphs.

these two boxes should belong to one cluster. It starts from a random box and all of its neighbors and iterates over all boxes. The running time for Algorithm 8 is $O(n \cdot \log n)$.

Only the core points are assigned in the previous phases, finally, the border points are assigned to the nearest cluster in the last phase, pseudo-code is shown as Algorithm 9. It iterates over all the points in all boxes, and computes the distance between the border points and their neighbor core points, each border point will be assigned to a cluster of the nearest core point.

After all, the DBScan algorithm using box graphs is composed of all phases mentioned above, as shown in Algorithm 10. The required inputs are a point dataset P, the distance ϵ, the minimum number of points *MinPts* within ϵ (but *MinPts* is often set to be 4 and not regarded as an input). The points in the dataset are first divided into strips, each of which is divided into boxes. Then neighbors of all boxes are computed. With boxes computed before, the algorithm is able to find all core points, which enables the clusters computing phase. In the end, the border points are also assigned.

Algorithm 6 COMPUTEBOXNEIGHBORS($\epsilon, \mathcal{B}$)

```
Require: ϵ: within distance ϵ from a point p, if there are MinPts points, p is a core point; $\mathcal{B}$: a list of boxes
    in strips $\mathcal{L}$;
Ensure: a list of boxes with correct neighbors
 1: $Q \leftarrow$ empty priority queue;
 2: for $i \leftarrow 1$ to $|\mathcal{L}|$ do
 3:     for each box $B \in S_i$ do
 4:         $y_{top}, y_{bottom} \leftarrow$ minimum and maximum y-coordinates of the bounding box of a box denoted
    as rectangle$(B)$;
 5:         add $(y_{bottom}, B, i)$ to Q with priorith $y_{bottom}$;
 6:         add $(y_{top} + \epsilon, B, i)$ to Q with priority $y_{top}$;
 7:     end for
 8: end for
 9: sweepline $\leftarrow$ an array of $|\mathcal{L}|$ empty lists;
10: while Q is not empty do
11:     $(y, B, i)$ the last row in Q, which has the minimum priority;
12:     if y is the bottom coordinate of rectangle$(B)$ then
13:         for each $j \in \{i-2, i-1, i, i+1, i+2\}$ do
14:             if $1 \le j \le$ size(sweepline) then
15:                 for each box B in sweepline$_j$ do
16:                     if minimum distance between B and $B'$ is at most $\epsilon$ then
17:                         add B to neighbors$(B')$; add $B'$ to neighbors$(B)$;
18:                     end if
19:                 end for
20:             end if
21:         end for
22:     else
23:         remove B from sweepline$_i$;
24:     end if
25: end while
26: $\mathcal{B} \leftarrow$ empty list;
27: for each strip $S \in \mathcal{L}$ do
28:     for each box $B \in S$ do
29:         append B to $\mathcal{B}$
30:     end for
31: end for
32: return $\mathcal{B}$
```

Algorithm 7 IDENTIFYCOREPOINTS(ϵ, *MinPts*, $\mathcal{B}$)

```
Require: ϵ:within distance ϵ from a point p, if there are MinPts points, p is a core point; MinPts: within
    distance ϵ, if there are MinPts points, it is defined as the Core Point; $\mathcal{B}$: a list of boxes;
Ensure: each core points will be labeled;
 1: for each box B in $\mathcal{B}$ do
 2:     if $n(B) \geq MinPts$ then
 3:         mark each point in points(B) as core point;
 4:     else
 5:         for each point p in B do
 6:             count $\leftarrow n(B)$;
 7:             for each box $B'$ in neighbors(B) do
 8:                 for each point q in $B'$ do
 9:                     if $dist(p, q) \leq \epsilon$ then count $\leftarrow$ count + 1
10:                     end if
11:                 end for
12:             end for
13:             if count $\geq MinPts$ then
14:                 mark p as core point;
15:             end if
16:         end for
17:     end if
18: end for
```

Algorithm 8 COMPUTECLUSTERS(ϵ, $\mathcal{B}$)

```
Require: ϵ:within distance ϵ from a point p, if there are MinPts points, p is a core point; $\mathcal{B}$: a list of boxes;
Ensure: correct cluster labels for each core points
 1: label $\leftarrow 1$;
 2: for each box B in $\mathcal{B}$ do
 3:     if cluster(B) = none and B contains a core piont then
 4:         cluster(B) $\leftarrow$ label; front $\leftarrow \{B\}$;
 5:         while front is not empty do
 6:             remove a box B from front;
 7:             for each point p in B do
 8:                 if p is a core point then
 9:                     cluster(p) = label;
10:                 end if
11:             end for
12:             for each box $B'$ in neighbors(B) do
13:                 if cluter($B'$) = none and $B'$ contains a core point then
14:                     if coredistance($B, B'$) $\leq \epsilon$ then
15:                         cluster($B'$) = label; add $B'$ to front;
16:                     end if
17:                 end if
18:             end for
19:         end while
20:         label $\leftarrow$ label + 1;
21:     end if
22: end for
```

Algorithm 9 ASSIGNBORDERPOINTS($\epsilon, \mathcal{B}$)

Require: ϵ: within distance ϵ from a point p, if there are *MinPts* points, p is a core point; $\mathcal{B}$: a list of boxes with core points;

Ensure: assignment of border points;

```
 1: for each box B in $\mathcal{B}$ do
 2:     for each point p in B do
 3:         if cluster$(p)$ = none then
 4:             nearest $\leftarrow$ none; distance $\leftarrow \infty$;
 5:             for each point q in B do
 6:                 if dist$(p, q) \leq$ distance and cluster$(q) \neq$ none then
 7:                     nearest $\leftarrow q$; distance $\leftarrow \mathtt{dist}(p, q)$;
 8:                 end if
 9:             end for
10:             for each box $B'$ in neighbors$(B)$ do
11:                 for each point q in B do
12:                     if dist$(p, q) \leq$ distance and cluster$(q) \neq$ none then
13:                         nearest $\leftarrow q$; distance $\leftarrow$ dist$(p, q)$;
14:                     end if
15:                 end for
16:             end for
17:             if distance $\leq \epsilon$ then
18:                 cluster$(p) \leftarrow$ cluster(nearest);
19:             end if
20:         end if
21:     end for
22: end for
```

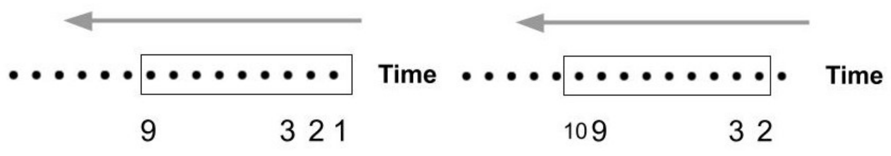
Algorithm 10 DBSCANBOXGRAPHS($P, \epsilon,$ *MinPts*)

Require: P: dataset, a set of points; ϵ: within distance ϵ; *MinPts*: if there are *MinPts* points within distance ϵ from point p, p is defined as a core point;

Ensure: correct cluster labels for each points;

```
1: $\mathcal{L} = \mathtt{ComputeStrips}(\epsilon, P)$;
2: $\mathcal{B} = \mathtt{ComputeBoxes}(\epsilon, \mathcal{L})$;
3: $\mathcal{B} = \mathtt{ComputeBoxNeighbors}(\epsilon, \mathcal{B})$;
4: invoke $\mathtt{IndentifyCorePoints}(\epsilon, MinPts, \mathcal{B})$;
5: invoke $\mathtt{ComputeClusters}(\epsilon, \mathcal{B})$;
6: invoke $\mathtt{AssignBorderPoints}(\epsilon, \mathcal{B})$;
```

2.3 Sliding Window



(a) The window size here is 9 hours, and it moves per 1 hour. If the data in the sliding window is not full, it waits until all data in the sliding window is received from nine minutes ago till now $t = 9$.

(b) The sliding window is full and it moves to the next hour. In the meanwhile, it deletes data received at the oldest minute, and append data received from $t = 9$ to $t = 10$.

Figure 2.4: Illustration of sliding window.

We develop offline algorithms for detecting rectangular hotspots. Later we show how to implement these algorithms in the sliding windows model as shown in Figure 2.4. The sliding windows model was introduced by Datar, Gionis, Indyk, and Motwani [6] as is as follows. We are given a stream of points possibly of infinite length and a window size W, and we are interested in computing a function (either exactly or approximately) for a window of size W of latest (newest) points in the stream. At any time t, we call the window that consists of points with arrival time between $[\max(1, t - W), t]$ the *current window*. Here we assume that at any time t only one point arrives, but in reality it can happen that at any timestamp t more than one point arrives, say a burst of points arrive. The idea is at any time t, the number of points that arrive is less than W and the current window covers W points that has the latest arrival time. For any point p, we denote by $t_a(p)$ the arrival time of p. Here we assume that one point arrive at a time.

After an interval v, the time now is $t' = t + v$, therefore, the current window deletes oldest points having arrival time between $[\max(1, t - W), \max(1, t' - W)]$ and adds points arriving between $[t, t']$.

2.4 F-score

F-score [19], also as known as F1-score, is an evaluation measure often used in information retrieval systems, it is also used to evaluate the binary classification. Whether hotspots are detected or not could be considered as a binary classification problem as well, so F-score is a useful evaluation method. The F-score can be computed with respect to precision and recall:

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad .$$

The precision mentioned above could be computed by the following function,

$$\text{precision} = \frac{TP}{TP + FP}$$

where TP is the number of true positives, FP is the number of false positives. It ranges from 0 to 1. A true positive point is when a point in hotspot region is predicted as a hotspot point. A false positive point is when a noise point is predicted as a hotspot point. Precision represents the rate of the correct prediction over all predictions.

The recall ranging from 0 to 1 is the probability of points that are detected as the hotspot points over all actual hotspot points, which shows the relation between actual hotspot points. It can be formulated as follows.

$$\text{recall} = \frac{TP}{TP + FN}$$

The number of false negatives denoted as FN is the number of points in hotspot region that are predicted to be background noise.

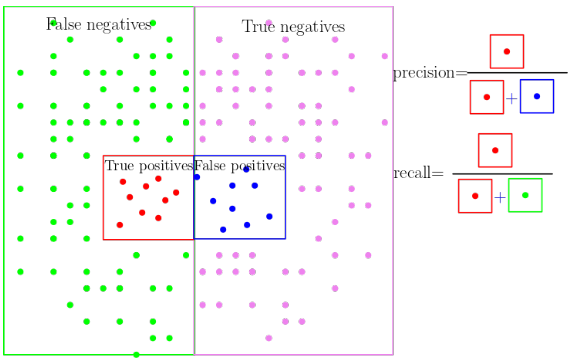


Figure 2.5: To show F-score in a direct way, it can be visualized. The points in green and red are the actual positive points while the points in purple and blue are the actual negative points. Precision can be represented by the red points divided by red and blue points, recall can be represented by the red points divided by red and green points.

There is often a negative relation between precision and recall, improving one could decrease the other one in the meantime. Hence, they always show up together as balanced measures. When precision equals 1, all points that are predicted as hotspot points are indeed hotspot points, but it does not promise all the other actual hotspot points can be detected correctly. The recall of 1 indicates that all hotspot points are detected when it does not reflect the situation that noise points are detected by mistake as well. The precision of 1 could be achieved by only detecting a small proportion of real hotspot points, and it is also easy to attain recall of 1 by detecting an actual hotspot point as one hotspot. Thus, a perfect precision or recall does not guarantee a perfect performance of an algorithm, F-score is come up with as a combination of precision and recall to balance the impact of both of them.

2.5 Kernel Density Estimate

In order to obtain the probability density function of the dataset, two density estimations can be used, which are non-parametric density estimation and parametric density estimation. The former one assumes that the dataset follows a certain probability distribution, such as likelihood estimation and Gaussian estimation, then it fits the parameters of the distribution according to the dataset. However, because it requires prior knowledge and there are big differences between the assumption and the real model, the result of this kind of estimation is not always satisfactory. The parametric density estimation does not involve any prior knowledge and assumption, instead, it estimates the density according to the characteristics and property of the data. A classic parametric density estimation, kernel density estimation (KDE), was first proposed by Rosenblatt [20] and Parzen [17] as the Parzen window. Before introducing the KDE, the histogram is unavoidable to talk about. The histogram is a widely used density estimation method. Intuitively, it shows the percentage of points that fall in a certain interval, such as $[y_i, y_{i+1})$, which is called a bin as well. The range of the histogram is between the minimum and the maximum value of the data. Given the number of bins, the width of the bin is $\frac{\text{range}}{\text{number of bins}}$. Then the probability density function is computed by the average of a fraction, which is the number of observations that fall in one specific bin as x divided by the width of the bin mentioned above. It is formulated as follows, where b stands for the bandwidth of the bin.

$$\hat{f}^{hist}(x) = \frac{1}{n}\frac{\text{\# of } X_i \text{ in same bin as } x}{\text{width of bin containing } x} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{b}\mathbf{1}\{x_i \in [x - \frac{1}{2}b; x + \frac{1}{2}b)\} \ .$$

Finally, one of the parametric estimations, KDE is introduced. The difference between KDE and the histogram method is the kernel function, which histogram uses a constant and KDE uses a continuous function. If the denominator nb is ignored, at the observed point, histogram adds one while KDE adds a kernel function, as shown in Figure 2.6. Therefore, KDE smooths the density function by adding value on not only the observed point but also its neighbors, which is the reason why the curve of probability density function estimated by KDE covers the values that the data does not include, such as when $x \leq -4$ and $x \geq 8$. The probability density function is estimated by KDE with the formula as follows, where b is

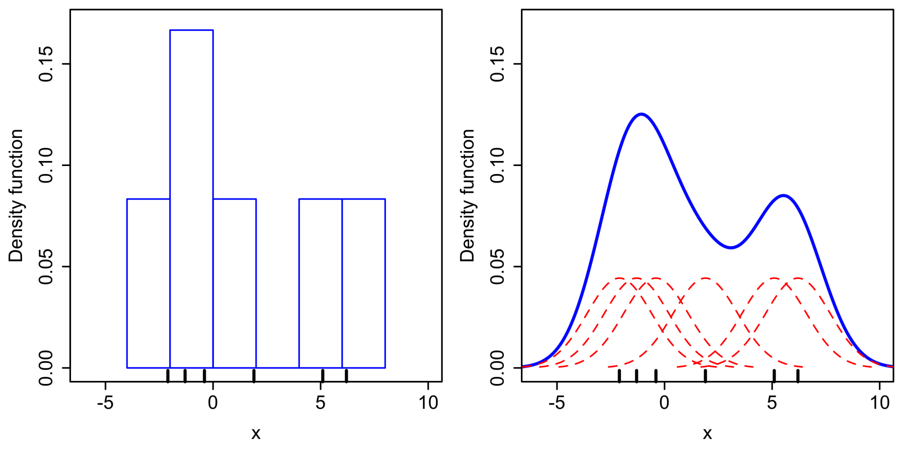


Figure 2.6: Difference of hitsogram and KDE, from [8]. Data points used are $\{-2.1, -1.3, -0.4, 1.9, 5.1, 6.2\}$, which are marked on the x-axis.

the bandwidth of the kernel function, $K(\cdot)$ is the kernel. The kernel weights differently depending on the distance between the observation and the x.

$$\hat{f}^{kde}(x) = \frac{1}{n}\sum_{i=1}^{N} K(x - x_i) = \frac{1}{nb}\sum_{i=1}^{N} K\left(\frac{x - x_i}{b}\right)$$

The kernel $K(\cdot)$ has lots of choices, the Gaussian function is being chosen here, therefore, the above function can be expressed by

$$\hat{f}_g^{kde}(x) = \frac{1}{nb}\sum_{i=1}^{n}\left(\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x-x_i{}^2}{2\sigma^2 b^2}}\right).$$

As for the bandwidth b, it uses $b = n^{\frac{-1}{d+4}}$, where d is the dimension.

Other than that, KDE is extended to estimate multivariate densities, and 2-dimension KDE is used in the following sections and thus is elaborated. It can be easily shown in Figure 2.7, which is contour map intuitively. The probability density function then needs to be extended to two dimensions, which is

$$\hat{f}^{kde}(x; H) = \frac{1}{n|H|^{1/2}}\sum_{i=1}^{n} K\left(H^{-1/2}(x - X_i)\right)$$

where $K(\cdot)$ is the multivariate kernel, and H is the bandwidth matrix, which also can be regarded as a covariance matrix of a multivariate normal density with mean X_i. The kernel function used here is the two-dimensional Gaussian kernel, i.e. $K(x) = (2\pi)^{-d/2}H^{-1}/2e^{-\frac{1}{2}x^T H^{-1} x}$

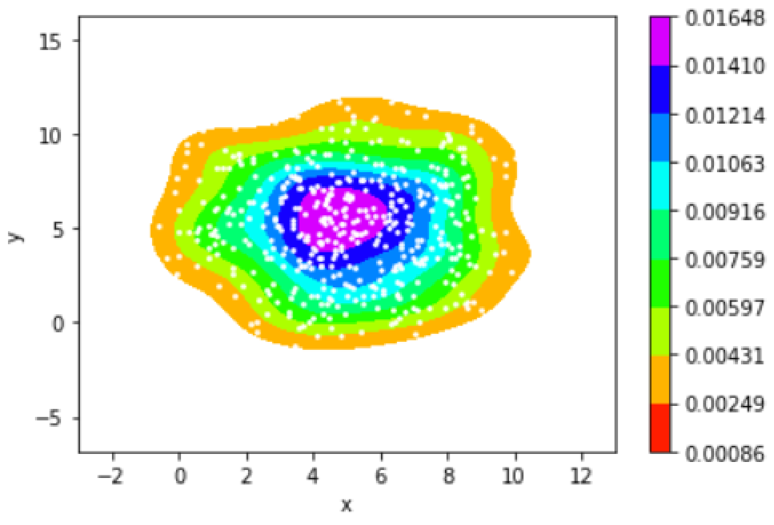


Figure 2.7: An example of 2-dimension KDE for a normally distributed dataset, and the data points are in white.

Chapter 3

Proposed Algorithms

There are a large number of clustering algorithms suitable for numerous datasets and problems. In this thesis, the hotspot problem is the main focus. The concept of the hotspot was first come up with by J. Tuzo Wilson in [23], hotspots are defined as the volcanic regions that are assumed to be fed by the underlying mantle that is anomalously hot. The word hotspot is then known for some famous places for tourism or entertainment. Nevertheless, in the computing field, hotspot stands for a region of a computer program that occurs a large number of executed instructions or that costs most of the time of the execution of the whole program, as defined in Wikipedia description [1]. In this thesis, a $(\alpha, \beta)-$hotspot is defined in Definition 1 as the region that covers points of higher density than its neighbors.

In this chapter, we propose two offline algorithms for rectangular hotspots detecting, and a stream algorithm. They are all based on density. The first one, the random sampling based algorithm, is a simple clustering algorithm that outputs k densest rectangles it samples as resulting hotspots. The limitation of it is that it depends more on the initial set of randomly sampled rectangles, so the results are relatively random. The second one is based on the DBScan clustering algorithm with box graphs [13], after choosing the optimal ϵ and *MinPts*, the top k densest rectangle are reported among all clusters where k is given in advance.

3.1 Random Sampling Based Algorithm

The primary idea of the random sampling based (RSB) algorithm is to select two points $p_1, p_2 \in P$ randomly, construct a minimum rectangle R_1 with p_1, p_2, then compute the density of R_1 as in Definition 2, repeat selecting the remaining $z-1$ rectangles by two random points, then return k rectangles that have the highest density and overlap with each other at most β of the area.

There are 2 main phases in this algorithm:

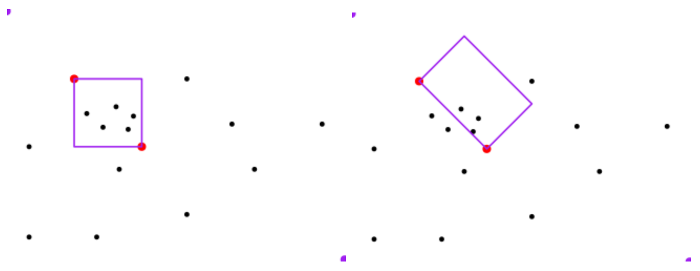
- **Sampling:** Randomly sample z pairs of points and construct z minimum rectangles with these z pairs of points. Append z minimum rectangles into a set $\mathcal{R}$.
- **Sorting and reporting:** Compute the densities of z rectangles in $\mathcal{R}$, sort rectangles in $\mathcal{R}$ by densities computed. At the end, report k rectangles in $\mathcal{R}$ as the hotspots $\{H_1, \cdots, H_k\}$. These k hotspots have highest density in $\mathcal{R}$ and any reported hotspot H_i overlap with any H_j at most β of the minimum area between H_i and H_j where $i, j \in \{1, \cdots, k\}$.

The pseudo-code is shown in Algorithm 11.

At the sampling stage, two points p_1, p_2 are selected randomly from P ($p_1 = p_2$ is possible), then construct a minimum rectangle with p_1 and p_2 as two of the corners, as shown in Figure 3.1 and defined in Definition 3.

Definition 3 (Minimum rectangle)**.** *A minimum rectangle R of two points p_1, p_2 can be represented*

$$R(x_{mid}; y_{mid}; w_R; h_R) \ ,$$


(a) A rectangle is constructed by two red points as two of the corner points, in the meantime, the rectangle is the minimum enclosing rectangle that is axis-aligned.

(b) Another rectangle that is also constructed by these two points as two of the corner points, but this rectangle is not the minimum rectangle, the area of the rectangle is random as the other two corner points could be random. Therefore, this case is not used.

Figure 3.1: Illustration of the minimum rectangle constructed by two points as the corner points.

where $x_{mid} = \frac{1}{2}(x_1 + x_2)$ *is the x-coordinate of the center point of* R, $y_{mid} = \frac{1}{2}(y_1 + y_2))$ *is y-coordinate of the center point of* R, $w_R = |x_1 - x_2|$ *stands for the width of* R, $h_R = |y_1 - y_2|$ *represents the height of* R.

The rest of the rectangles are chosen by repeating the same process. At the end of this phase, a list of rectangles $\mathcal{R} = \{R_1, \ldots, R_z\}$ of size z is selected and stored as the sampled rectangles for the next phase. Any rectangle $R_i \in \mathcal{R}$ can overlap with other rectangles in $\mathcal{R}$ at most β of the minimal area between R_i and other rectangles in $\mathcal{R}$. The sampled number z is actually a function of k, i.e.$z = f(k)$, and $z \geq k$. As for the selection of z, it is a trade-off between the precision and time consumed, which would be chosen through experiments in section 4.3 and k is given in advance.

Apparently, the random selection brings up the overlapping problem. As points are sampled randomly, there is a certain probability that a rectangle constructed by a pair of points will be overlapped with another sampled rectangle. The overlapping rate of two rectangles R_1 and R_2, denoted as $\beta_{1,2}$, is computed as

$$\beta_{1,2} = \frac{\text{area}(R_1) \bigcap \text{area}(R_2)}{\min(\text{area}(R_1), \text{area}(R_2))} \ .$$

The maximum overlapping rate is denoted as β as in a (α, β)—hotspot, an upper bound of the overlapping rate of all reported rectangles, which will be chosen with the highest F-score in section 4.3.

Definition 4 (Maximum overlapping rate)**.** *Let* $H_i, H_j \in \mathbb{R}^2$ *be two overlapping rectangualr hotspots. Let* $\beta_{i,j}$ *denote the overlapping rate of* H_i *and* H_j. *For any two rectangular hotspots* H_i *and* H_j, H_i *can intersect with* H_j *at most* β, *that is*

$$\text{area}(H_i) \bigcap \text{area}(H_j) \leq \beta \cdot \min(\text{area}(H_i), \text{area}(H_j)) \ .$$

A higher proportion of overlapping areas may result in the detection of fewer hotspots, because when they overlap, all reported rectangles are very close to each other. Though repeating detection increases the reliability of the repeating area to be a correct hotspot prediction, the algorithm may have less information of the neighbor region or the other places. Nevertheless, a smaller rate of the overlapping area increases the probability to search for hot spots at more places.

An explicit problem also can not be overlooked, a tiny area of sampled rectangles causes false-high density. For instance, when two close points are selected, or even worse, the same point is selected twice, the area of the minimum rectangle R constructed by these two close points is extremely small, or even zero, causing the density of R incredibly huge even when there is only one point in R. It is definitely not an expected case when a tiny place having one point is defined as a dense region. Therefore, it is of great importance to set up the minimum area S_{min} expected in advance, which should be at least 1.0 to avoid the false high-density problem. If the S_{min} is set to be smaller than 1.0, then the mentioned problem of false high-density has the ability to have a fairly bad influence on clustering, but if it is too big, R covering too many non-hotspot points will result in lower precision. Hence, setting a suitable modest value for S_{min} is essential. The selection for S_{min} is done by experiments in section 4.3.

At the sorting and reporting stage, the densities of the selected rectangles $\mathcal{R}$ are calculated first as it is the criterion for sorting. Next, z rectangles are sorted based on their density decreasingly, other than reporting the first k rectangles, it selects the first densest rectangle R_1 at first, and then append the next rectangle R_2 that $\beta_{1,2} \leq \beta$, and repeat selecting the rectangles with β until there are k rectangles. Then report the k densest rectangles that having maximum overlapping rate β as k rectangular hotspots.

Algorithm 11 RandomSamplingBasedClustering(P, z, k, β)

```
Require: $P$: the set of points; $z$: the number of pairs of points that needs to be sampled; $k$: the number of
    clusters; $\beta$: the optimal overlapping proportion of area two rectangura hotspots can at most intersect.
1: for each $i \in \{1, \dots, z\}$ do
2:     From $P$ sample a pair of points $(p_{i1}, p_{i2})$ randomly;
3:     Construct a rectangle $R_i$ with width, height, and the average of sampled two points as its coordinate,
    i.e. width=$|x_{i1} - x_{i2}|$, height=$|y_{i1} - y_{i2}|$, $(x_i, y_i) = (\frac{1}{2}(x_{i1} + x_{i2}), (\frac{1}{2}(y_{i1} + y_{i2}))$;
4:     Compute the density of $R_i$ by density$(R_i) = \frac{n(R_i)}{\text{area}(R_i)}$, where $n(R_i) = |R_I \cap P|$;
5: end for
6: Sort $z$ rectangles decreasingly according to their densities, so the first rectangle $R_1$ of the sorted rectangle
    set $\mathcal{R}$ has the highest density;
7: Initialize the set of Hotspots $\mathcal{H} = \{R_1\}$;
8: repeat
9:     compute the overlapping rate $\beta_{i,j}$ between the next rectangle $R_i \in \mathcal{R}$ and the rectangles $R_j \in \mathcal{H}$,
    the next rectangle $R_i$ is selected in decreasing order of sorted $\mathcal{R}$;
10:    if $\forall R_j \in \mathcal{H}$, the overlapping rate $\beta_{i,j} \leq \beta$, append $R_i$ to $\mathcal{H}$;
11: until ($|\mathcal{H}| \geq k$)
```

Lemma 5. *The running time complexity of Algorithm 11 is* $O(n \log n + z(n + k))$.

Proof. Line 2 and 3, 10 can be done in $O(1)$ time, computing density involving counting points in a rectangle takes $O(n)$ time, so line 1 to 5 takes $O(nz)$ time. The sorting at line 6 takes $O(n \log n)$ time, and line 7 takes only $O(1)$ time. Line 9 needs to compare each rectangles in $\mathcal{R}$ and rectangles in $\mathcal{H}$, which takes $O(|\mathcal{R}| \cdot |\mathcal{H}|) = O(zk)$. Therefore, the result can be obtained by summing up all of them, which is $O(nz + n \log n + zk) = O(n \log n + z(n + k))$. □

3.2 Adapted DBScan algorithm

The adapted DBScan runs a bit differently in the offline model and the sliding window model. In the offline model, the input dataset is fixed while in the sliding window model it is continuously changing. Also, to speed up the adapted DBScan algorithm in the sliding window model, a sampling trick is also used. The details can be seen in the following subsections.

3.2.1 Adapted DBScan in Offline Model

The main idea of the adapted DBScan algorithm is to first find the optimal ϵ and *MinPts* for DBScan, then run the DBScan algorithm using box graphs with these parameters on a dataset. After that, points are assigned into several clusters, the bounding box of a cluster is regarded as a rectangle. These rectangles are sorted by their densities defined in Defintition 2, then top k rectangles are reported as the rectangular hotspots, hence points covered in those regions will be reported as hotspot points. The pseudo-code elaborates it in Algorithm 13.

At first, the optimal ϵ and *MinPts* for DBScan should be found with the F-score used as the evaluation measure of experiments. To select the best ϵ and *MinPts*, the algorithm first enumerates a list of ϵ and a list of *MinPts*. The list of ϵ is generated with respect to the maximum value of width and height of the whole region. A parameter λ between $[0, 1]$ is used, $1 + \lambda$ is used as the base. The biggest ϵ could be as big as the bigger value between width and height of the canvas, that is LongSide $= \max\{width, height\}$, hence the maximum value of ϵ is given by

$$\epsilon_{max} = \text{LongSide} = \log_{1+\lambda}(1+\lambda)^{\text{LongSide}}.$$

Set the minimum ϵ to be the side length of the minimum area of the box graphs S_{min}, that is,

$$\epsilon_{min} = \sqrt{S_{min}} = \log_{1+\lambda}(1+\lambda)^{\sqrt{S_{min}}}.$$

Therefore, the reasonable range of epsilon $\mathcal{E}$ is defined as $(1+\lambda)^j$, where $j \in \{\epsilon_{min}, \epsilon_{min}+1, \dots, \epsilon_{max}\}$. The same procedure could be done for the list of *MinPts*, denoted as MP, the corresponding parameter is defined as $\gamma \in [0, 1]$, which means the base is $1 + \gamma$. The maximum value of *MinPts* will be the number of points of the dataset n, thus, $\mathit{MinPts}_{max} = n = \log_{1+\gamma}(1+\gamma)^n$. So the list of *MinPts* is defined as $(1+\gamma)^i$, where $i \in \{0, 1, \dots, \mathit{MinPts}_{max}\}$.

However, the optimal pair of ϵ and MinPts is not fixed forever, they could be different over time since the data changes over time. Hence, it is essential to update the optimal pair of ϵ and *MinPts* at a certain interval since data in a specific time range could be extremely dense or sparse.

While enumerating each ϵ in $\mathcal{E}$, each *MinPts* in MP, DBScan algorithm returns $z_{i,j}$ clusters in each iteration using ϵ_i and MinPts_j. For any cluster C in $\{C_1, \cdots, C_{z_{i,j}}\}$, find the bounding box $R(C)$ of cluster C, that is, $\{x_{lower} = x_{min}, x_{upper} = x_{max}, y_{lower} = y_{min}, y_{upper} = y_{max}\}$, and let it represent the cluster C. Then compute the density of this rectangular cluster density$(R(C))$, and append the tuple of rectangle and its density to a list

$$\text{RectDenList} = \{(R(C_1), \text{density}(R(C_1))), \dots, (R(C_{z_{i,j}}), \text{density}(R(C_{z_{i,j}})))\}.$$

The RectDenList is then sorted by the density density$(R(C_1))$. Among $z_{i,j}$ rectangles, the algorithm first picks the first rectangle that has the highest density in a set of rectangles which is denoted by kRectList. Then compute the overlapping rate $\beta_{i,j}$ between each rectangle $R(C_i) \in$ RectDenList and rectangles $R(C_j)$ already in kRectList, if the overlapping rate $\beta_{i,j} \leq \beta$ and area$(R(C_i)) \geq S_{min}$, append rectangle $R(C_i)$ and its density as a pair to the kRectList. Each iteration with different pair of epsilon and MinPts results in k different clusters, let the $(\epsilon, \mathit{MinPts})$ be the key of a dictionary Dic which has size of $(i \times j)$, and the list kRectList that has k hotspots be the value of Dic. Then, find the optimal hotspot detection $\mathcal{H}^*$ among all hotspots of different epsilon and MinPts in Dic by comparing the average of density. In the end, the set of hotspots that has highest average density is returned as the best hotspot detection result.

Algorithm 12 FINDPOINTSINRECTS(RECTS, P)

```
Require: $rects$: a list of rectangles; $P$: a list of points;
Ensure: $P_{rects} = rects \bigcap P$: a list of points that are covered in the rectangles in $rects$.
 1: $P_{rects} \leftarrow$ empty list;
 2: for each $r \in rects$ do
 3:     find the bounding edges of $r$, they are $x_{min}, x_{max}, y_{min}, y_{max}$;
 4:     for each $p \in P$ do
 5:         if $x_{min} \leq x(p) \leq x_{max}$ and $y_{min} \leq y(p) \leq y_{max}$ then
 6:             append point $p$ in $P_{rects}$, i.e. $P_{rects} = P_{rects} \bigcup p$;
 7:         end if
 8:     end for
 9: end for
10: return $P_{rects}$;
```

Lemma 6. *The running time complexity of Algorithm 12 is* $O(|\text{rects}|n)$.

Proof. Line 3 and Line 5 to 6 take $O(1)$ running time, so from line 4 to 8, it takes $O(n)$. Therefore, the algorithm takes $O(|\text{rects}|(n+1)) = O(|\text{rects}|n)$. □

Algorithm 13 $\textsc{AdaptedDBScan}(\mathcal{E}, MP, S_{min}, \beta, P)$

```
Require: $\mathcal{E}$: a list of $\epsilon$; $MP$: a list of $MinPts$; $S_{min}$: minimum area of the hotspot; $\beta$: the optimal
    overlapping proportion of area two rectangles can at most intersect; $P$: a list of points;
Ensure: optimal hotspot detection $\mathcal{H}^*$
 1: $Dic \leftarrow$ empty dictionary; RectDenList $\leftarrow$ empty list; kRectList $\leftarrow$ empty list;
 2: for $\epsilon$ in $\mathcal{E}$ do
 3:     for MinPts in $MP$ do
 4:         Invoke DBScanBixGraphs($P$, $\epsilon$,MinPts), then we have $z$ clusters $\{C_1, \dots, C_z\}$;
 5:         for any cluster $C$ in $\{C_1, \dots, C_z\}$ do
 6:             construct the bounding rectangle of the cluster $R(C)$ and compute the density of $R(C)$,i.e.
    density($R(C)$), with the points covered in $R(C)$ by invoking FindPointsInRects($R(C)$, $P$) ;
 7:             append ($R(C)$, density($R(C)$)) to RectDenList;
 8:         end for
 9:         sort RectDenList by density($R(C)$) decreasingly so that the first item in RectDenList has the
    highest density;
10:         add the first $R(C_1)$ in RectDenList to kRectList;
11:         repeat
12:             if next $R(C_i)$ in RectDenList intersect rectangles $R(C_j)$ in kRectList at most $\beta$ of
    min(area($R(C_i)$), area($R(C_j)$)), and area of $R(C_i)$ is at least $S_{min}$ then
13:                 add ($R(C_i)$, density($R(C_i)$)) in RectDenList to kRectList;
14:             end if
15:         until there are $k$ items in kRectList
16:         Add ($\epsilon$,MinPts) , kRectList to $Dic$;
17:     end for
18: end for
19: The optimal set of hotspots $\mathcal{H}^* \leftarrow$ all rectangles in kRectList of the first item in $Dic$;
20: for ($\epsilon$, MinPts), kRectList in $Dic$ do
21:     Rects $\leftarrow$ all $R(C)$ in kRectList;
22:     compute the average density of Rects, $D_{avg}$(Rects);
23:     if $D_{avg}$(kRectList) $> D_{avg}(\mathcal{H}^*)$ then
24:         $\mathcal{H}^* \leftarrow$ Rects;
25:     end if
26: end for
27: return $\mathcal{H}^*$
```

Lemma 7. *The running time complexity of Algorithm 13 is* $O(|\mathcal{E}| \cdot |MP|(n \log n + z(k+n)))$.

Proof. As mentioned in the preliminaries section, DBScan using box graphs takes $O(n \log n)$ running time, which is at line 4. Because computing the density of a rectangle involves Algorithm 12, which takes $O(n)$ time when the number of the required rectangle is one at each iteration. The computing time of line 5 to 8 is $O(zn)$ with line 7 taking onely $O(1)$. The sorting phase in line 9 takes $O(n \log n)$ and $O(1)$ for line 10. The progress in Line 11 to 15 uses two loops, one for each rectangle in RectDenList, one for each rectangle in kRectList, so it costs $O(kz)$. The adding action in line 16 takes $O(1)$. So for the above iterations, the running time is

$$O(|\mathcal{E}| \cdot |MP|(n \log n + zn + O(n \log n) + kz)) = O(|\mathcal{E}| \cdot |MP|(n \log n + z(k+n))).$$

Line 19, 21, 22 and 24, each of them costs $O(1)$ time. So from line 19 to 26, it takes $O(|\mathcal{E}| \cdot |MP|)$ time. Overall, this algorithm takes $O(|\mathcal{E}| \cdot |MP|(n \log n + z(k+n)))$ running time. □

3.2.2 Adapted DBScan in sliding window model

Before running the DBScan algorithm using box graphs on line 4 in Algorithm 13, a sampling trick is used to speed up the algorithm in the sliding window model. When we sample a proportion of data to run the DBScan, the top k densest rectangular hotspots on sampling data are returned, which can be considered as the hotspots on the whole dataset. This probability of sampling is denoted as the sampling rate $Sr \in (0, 1)$, which has effects on the speed of the algorithm. When $Sr = 0.5$, it samples half of the data points randomly and runs the DBScan algorithm on the sampled dataset. The smaller the Sr is, the faster the algorithm runs. Although a low Sr could speed up the algorithm, it may end up with horrible cluster results since a low Sr fraction of data could hardly represent the whole dataset. Here, the Sr is set to be 0.25. After this, the sampled data is used as the point set P in Algorithm 13.

Also, the sliding window shifts for intervals of length W and we move it every time for a time interval ν. Random sampling is used at every interval so that when the window moves, as new data is appended and some of the old data is deleted, data is always sampled before adding to the window. Next, DBScan is used on sampled points inside each window. However, there are two ways to tune the parameters of DBScan:

- Parameters are only updated once at the beginning and it does not change further.
- Parameters are updated once in a while after observation has been made to the result of the algorithm in terms of F-score is deteriorating.

Besides the parameter tuning, the number of rectangular hotspots is also changed over time, which is for the purpose of checking the performance of the algorithm when hotspots change.

Chapter 4

Experiments and Evaluation

4.1 Setting

Experiments are implemented on a laptop with Intel i7-3687 CPU and 8 GB RAM, programmed via Python using Jupyter Notebook.

4.2 Dataset

Two datasets are used to evaluate the algorithms, they are the rectangle dataset and the New York taxicab dataset of TLC record data from the public data of the New York government[1], which is abbreviated as the taxi dataset in this thesis. The generated rectangle dataset could contain an arbitrary number of rectangular hotspots, background noise data (non-hotspot points), say generate points in three rectangular hotspots and three background rectangles, as shown in Figure 1.1.

The first dataset introduced is the rectangle dataset. As shown in Figure 1.1, the number of points of the rectangle dataset used in this thesis is 2300 in total, including 1500 background/noise points and 800 hotspot points. The whole canvas has a width of 10 and a height of 10. All of the background rectangles have both width and height of 4 (gray rectangles) while those of rectangular hotspots are both 1 (blue rectangles). The center point of each background rectangle is at $(2, 6), (5, 3), (8, 7)$, respectively. Background points have a probability of 0.1 to be anywhere on the canvas (could be in the hotspots), and a probability of 0.3 to be in each of the background rectangles. The center point of each rectangular hotspot is at $(2, 8), (4, 4), (5, 5)$, respectively. Hotspot points have an even probability to be in each of the rectangular hotspots, which is $\frac{1}{3}$.

Points in blue are the generated hotspot points, they are normally distributed. The non-hotspot points are in gray, some of them are in the rectangular bounding box and some of them are scattered. Let m be the number of generated hotspot points, each point has different probabilities to be in different rectangular hotspots, which is $Pb_{hotspot} = \{pb_1, pb_2, \ldots, pb_f\}$, where f is the number of rectangular hotspot. Those m points are also possible to be scattered and the probability is pb_s. Thus $\sum_{i=1}^{f} pb_i + pb_s = 1$. Therefore, the number of points in each rectangular hotspot can be computed approximately, i.e. $\{m \cdot pb_1, m \cdot pb_2, \ldots, m \cdot pb_f, m \cdot pb_s\}$, but these values are not the exact number since they are randomly generated. Other than the coordinates, points are also generated with other information, such as datetime and ID. The datetime is generated randomly within a specific required time (Python DateTime format), and ID is the generated order (an integer).

Hotspot points and background points can have three spatial relations, which are completely overlapped, partly overlapped, and disjoint. As shown in Figure 4.1, the grey points are the background points, the blue points are the hotspot points. The hybrid of three relations in Figure 4.1 is already shown in Figure 1.1, which is also the rectangle dataset that will be used in this thesis. It is obvious that if hotspot points intersect with background points (Figure 4.1(c)) it will be easier for clustering algorithms to divide clusters while it is

[1]https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

harder for the other two relations since they may be seen as one cluster if the clustering algorithm is not good at separating denser clusters or parameters of it are not suitable.

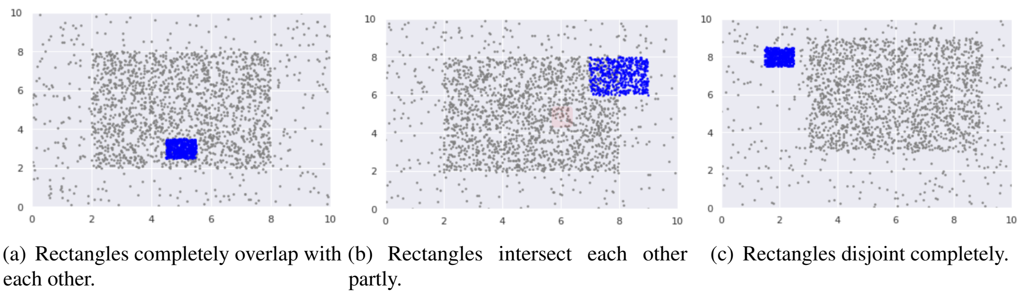

(a) Rectangles completely overlap with each other.
(b) Rectangles intersect each other partly.
(c) Rectangles disjoint completely.

Figure 4.1: Different spatial relations that rectangles have.

Another dataset is the New York taxicab dataset, in which only the part of points that are in the Manhattan region is used. Points are generated in several rectangular regions to simulate hotspot points. As shown in Figure 4.2, there are 3000 generated hotspot points in blue, and points in gray are the non-hotspot points from the New York taxicab dataset (filtered by datetime). The New York taxi dataset contains 19 columns, which are vendorID, tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, store_and_fwd_flag, RateCodeID, payment_type, fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge, total_amount. The useful columns are the ones related to pickup or drop-off datetime, pickup or drop-off longitude and latitude. For convenience, The longitude and latitude attributes are transformed into x and y coordinates. Before running algorithms, the radius of the Earth denoted as r_E is required for all coordinates. The rightmost x-coordinate is based on longitude and cosine of the average of the range of latitude, that is $x_{max} = \text{longitude} \cdot r_E \cdot \cos(\text{average}(\text{latitude}))$. The uppermost y-coordinate is related to the latitude, that is $y_{max} = \text{latitude} \cdot r_E$. The coordinates of points in between can be easily computed by using the proportion.

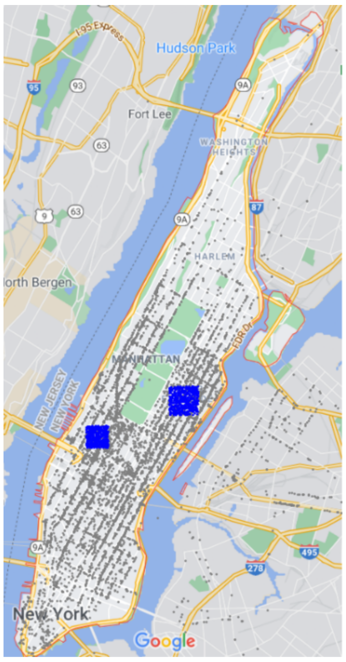


Figure 4.2: New York taxi dataset, this figure shows pickup data of January 10^{th}, 2015, from 7 am to 8 am. Two rectangles are planted as hotspots, which include 3000 points with a half of the chance to be in each of rectangular hotspots. Hotspot points are shown in blue while the noise/non-hotspot points are in gray.

4.3 Parameter Selection

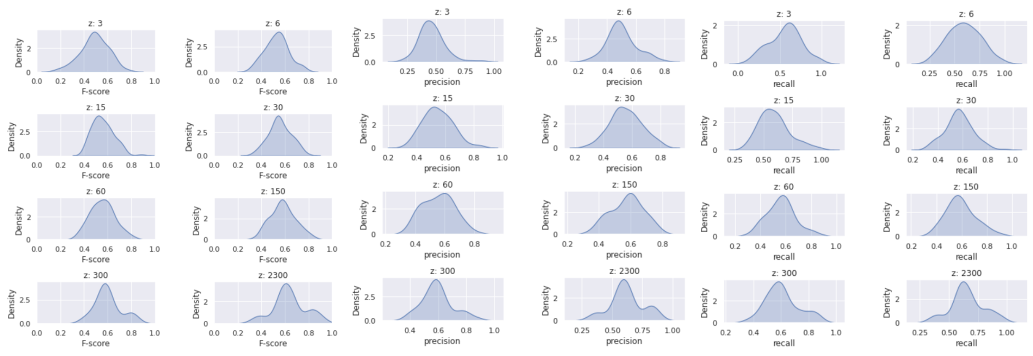
In the proposed algorithm section, there are several parameters that are to be determined by experiments, which will be discussed in this section. For different datasets, parameters for different algorithms are discussed.

4.3.1 Rectangle Dataset

As for the parameter selection for the RSB algorithm, the number of rectangular hotspots k is set to be 3, the other three parameters involved are: the sampled number z, the minimum area of the sampled rectangle $S_{min}^{(RSB)}$, the maximum overlapping rate $\beta^{(RSB)}$. They are chosen by experiments and comparisons between different values of the same parameter while other parameters stay still.

RSB algorithm

Sampled number z The number of rectangles that RSB samples is denoted as z, which is actually the number of sampled pairs of points. It is a function of k, which is the number of clusters. Several linear functions are considered, they are: $z = \{k, 2k, 5k, 10k, 50k, 100k, n\}$ where n is the number of points. Since k is set to be 3 in advance, comparison between different $z = f(k) = \{3, 6, 15, 30, 150, 300, 2300\}$ of the same $k = 3$ could be shown in Figure 4.3.



(a) The density function of F-score, each curve has different number of sampled pair of points z on rectangle set.

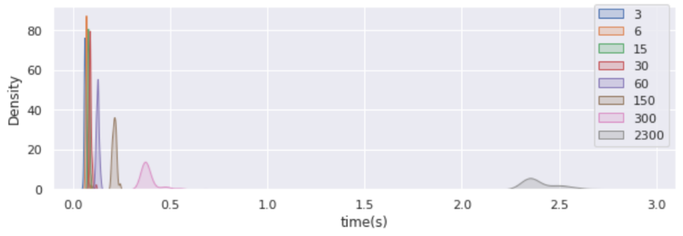
(b) The density function of precision of different number of sampled pair of points z on rectangle set.

(c) The density function of recall of different number of sampled pair of points z on rectangle set.

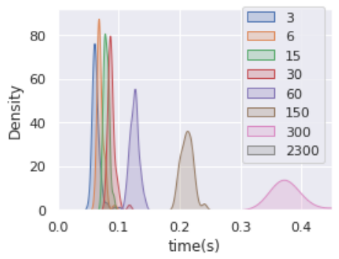
Figure 4.3: RSB samples different z pairs of points depending on k, comparisons are made between them and measured by F-score, precision and recall. We run the algorithm 100 times with the same parameters to obtain density functions. The z of each curve is described in the caption.

Although lots of pairs of points are sampled, the F-score is not increased rapidly. It can be seen in Figure 4.3, sampling 2300 pairs of points (rectangles) does not change the majority of F-score, though there are some having high scores, they only account for a small percentage. That is because the randomness of sampling points has a deep influence. Although having a large number of sampling pairs does not increase the F-score greatly, it still increases the probability of having a higher F-score, so the purpose is to have as many sampled pairs as possible. However, the number of sampled pairs affects the running time greatly, as shown in Figure 4.4(b), the density functions of the running time of $z \leq 30$ are very close. Here $z = 30 = 10k$ is chosen in order to trade off the requirements that z should be as big as possible to be more likely to gain a higher F-score and as small as possible to speed up the algorithm.

Minimum area $S_{min}^{(RSB)}$ As shown in Figure 4.5, the F-score, precision, and recall are compared between different minimum areas. Each of the returned k rectangles should have an area of at least S_min. The majority of the F-score of a higher $S_{min}^{(RSB)}$ is lower than those having a smaller minimum area, which is

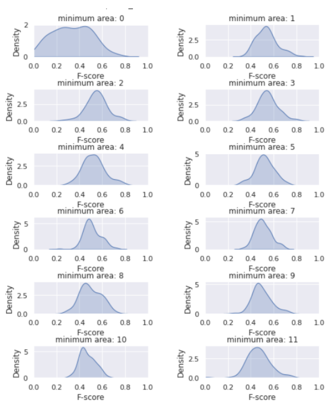


(a) The density function of running time of different number of sampled pair of points z on rectangle set.

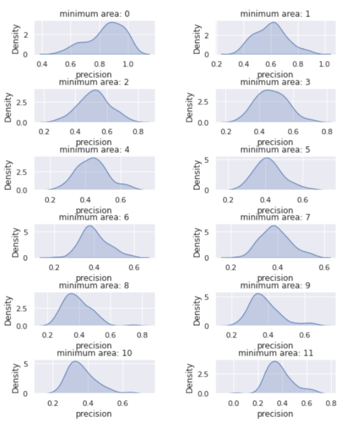


(b) Let us zoom in the subgraph(a) to take a close look since curves of smaller z are not clear.

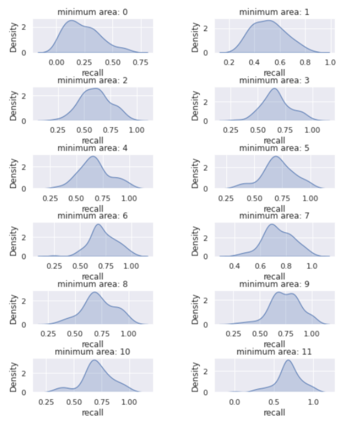
Figure 4.4: RSB samples different z pair of points depending on k, run this algorithm 100 times with the same parameters, the density function of running time of each is shown together.



(a) The density function of F-score of different minimum area $S^{(RSB)}_{min}$ on rectangle set.



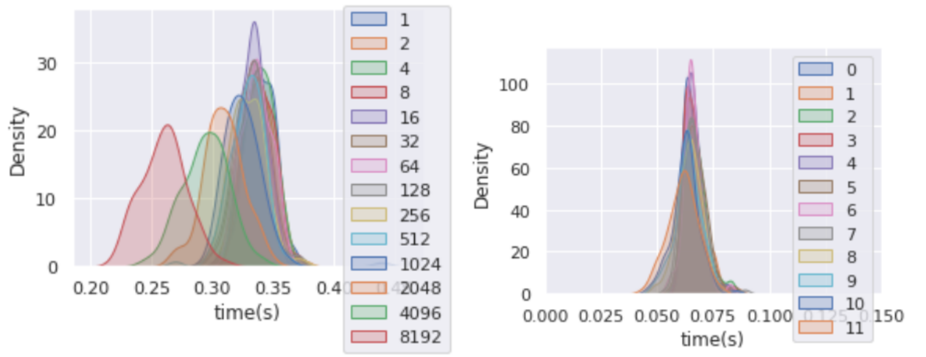
(b) The density function of precision of different minimum area $S^{(RSB)}_{min}$ on rectangle set.



(c) The density function of recall of different minimum area $S^{(RSB)}_{min}$ on rectangle set.

Figure 4.5: The density functions for choosing the minimum area parameter on rectangle set shown in Figure 4.1, measured by F-score, precision, and recall. Each density function curves are obtained by running RSB algorithm 100 times with the same parameters.

quite understandable since larger rectangles have more probability to cover more non-hotspot points and thus lower the precision, especially when the sampled rectangle is much larger than the generated rectangle. A fun fact could be found that the performance of the algorithm is extremely random when there is no limit on $S_{min}^{(RSB)}$, which is caused by the randomness of the process that chooses pairs of points. According to Figure 4.5, the majority of F-score of smaller $S_{min}^{(RSB)}$ is higher, especially when $S_{min} = \{1, 2, 3\}$. Here $S_{min} = 1$ is used to reduce the probability to contain more noise points.



(a) The density function of F-score of different minimum area $S_{min}^{(RSB)}$ on New York taxi set.

(b) The density function of F-score of different minimum area $S_{min}^{(RSB)}$ on rectangle set.

Figure 4.6: The density function of F-score for choosing the minimum area parameter on the rectangle dataset and the taxi dataset, measured by running time.

However, the running time of every dataset is not affected by the variant of minimum area, which can be illustrated by Figure 4.6.

Maximum Overlapping Rate $\beta^{(RSB)}$ Set $k = 3$, $z = 10 \cdot k = 30$, $S_{min} = 1$, comparison is made over $\beta^{(RSB)} = \{0, 0.1, \dots, 1\}$, when $\beta^{(RSB)} = 0$ it means all rectangles of selected clusters could not intersect at all, and $\beta^{(RSB)} = 1$ means they could be completely overlapped or could have inclusion relation. The choice of this parameter is given by experiments as shown in Figure 4.7. The density functions of F-score or precision or recall of different $\beta^{(RSB)}$ are really similar. It is apparent that there is no such relation between overlapping rate and F-score, therefore, a compromised value, 0.5 is chosen.

From Figure 4.8, it can also tell that different value of $\beta^{(RSB)}$ does not affect the running time at all.

Adapted DBScan

The minimum area and maximum overlapping area are also parameters in the adapted DBScan algorithm. To be clear, superscripts are used to indicate the parameter is for which algorithm such as minimum area denoted as $S_{min}^{(D)}$ is for the adapted DBScan and $S_{min}^{(RSB)}$ is for the RSB algorithm.

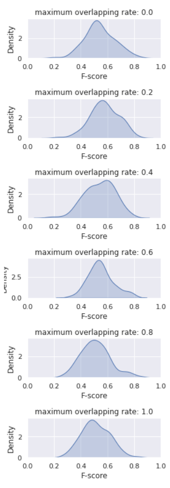
Minimum area $S_{min}^{(D)}$ Observation can be made from Figure 4.9 that the best $S_{min}^{(D)}$ is 1 when it has the highest F-score and a relatively low running time. This implies that the size of hotspot detecting results is quite close to the actual hotspots which is a 1×1 rectangle.

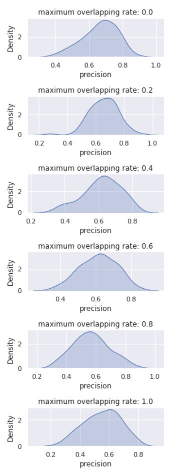
Maximum overlapping rate $\beta^{(D)}$ Observation can be made from Figure 4.10 that the best maximum overlapping rate $\beta^{(D)}$ is 0.9 or 1, to avoid completely overlapping, $\beta^{(D)} = 0.9$ is used.

4.3.2 Taxi dataset

RSB algorithm

Sampled number z It uses the same value as the rectangle dataset, $z = 10k = 30$.


(a) The density function of F-score of different minimum area $\beta^{(RSB)}$ on rectangle set.


(b) The density function of precision of different minimum area $\beta^{(RSB)}$ on rectangle set.

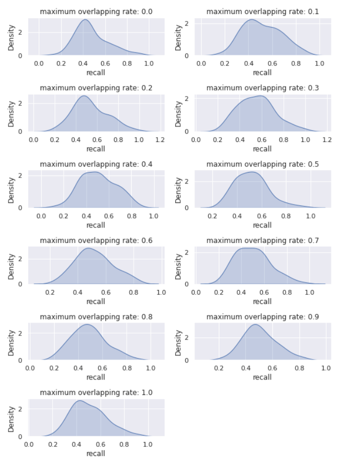

(c) The density function of recall of different minimum area $\beta^{(RSB)}$ on rectangle set.

Figure 4.7: Using the rectangle dataset, and set $k = 3$, $z = 10 \cdot k = 30$, $S_{min} = 1$, the relation between overlapping rate and F-score is shown. The density functions are attained by running the RSB with the parameters mentioned 100 times.

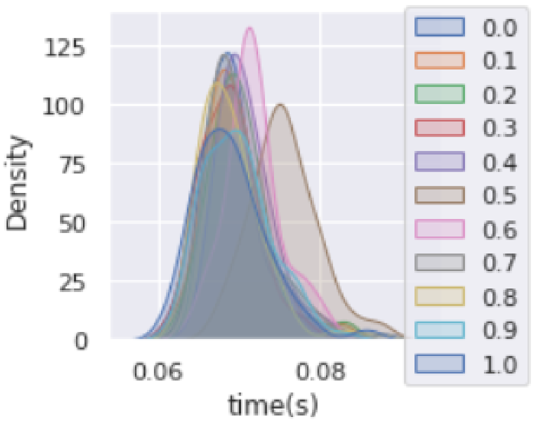
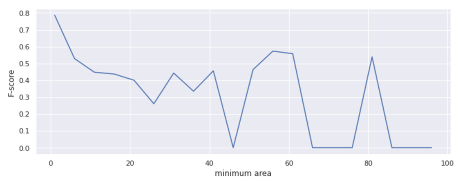
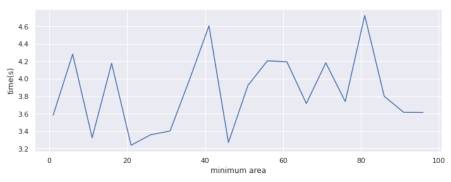


Figure 4.8: Using the rectangle dataset, and set $k = 3$, $z = 10 \cdot k = 30$, $S_{min} = 1$, the density functions of running time of different maximum overlapping rate are shown. They are attained by running the RSB with the parameters mentioned 100 times.

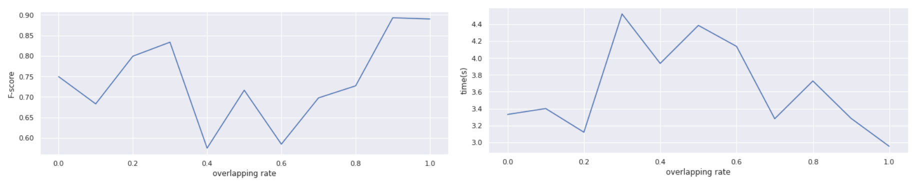


(a) The F-score curve of adapted DBScan on the rectangle dataset with different $S^{(D)}_{min}$.



(b) The running time curve of adapted DBScan on the rectangle dataset with different $S^{(D)}_{min}$.

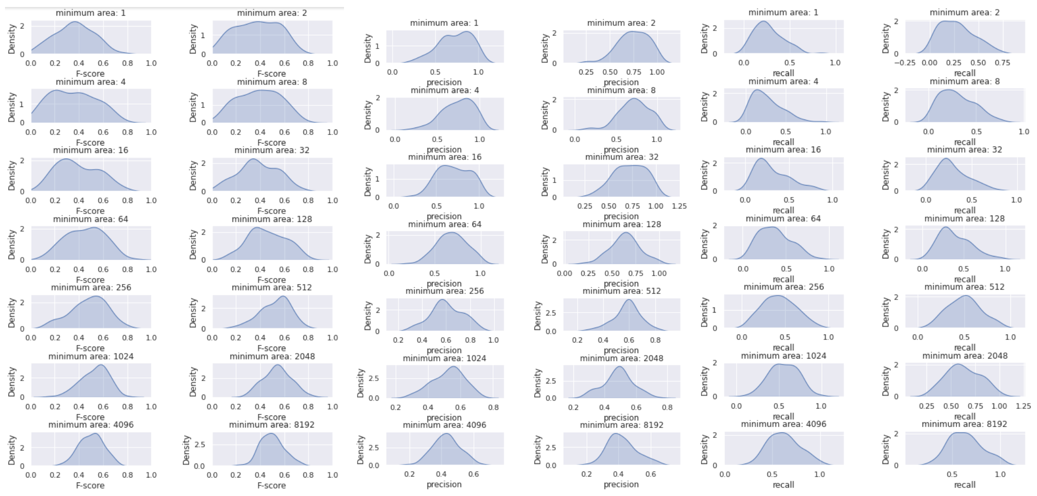
Figure 4.9: The F-score and running time curves of adapted DBScan on the rectangle dataset with different $S^{(D)}_{min}$. The maximum overlapping rate using here is 1.



(a) The F-score curve of adapted DBScan on the rectangle dataset with different $\beta^{(D)}$.
(b) The running time curve of adapted DBScan on the rectangle dataset with different $\beta^{(D)}$.

Figure 4.10: The F-score and running time curves of adapted DBScan on the rectangle dataset with different $\beta^{(D)}$. The minimum area using here is 1.

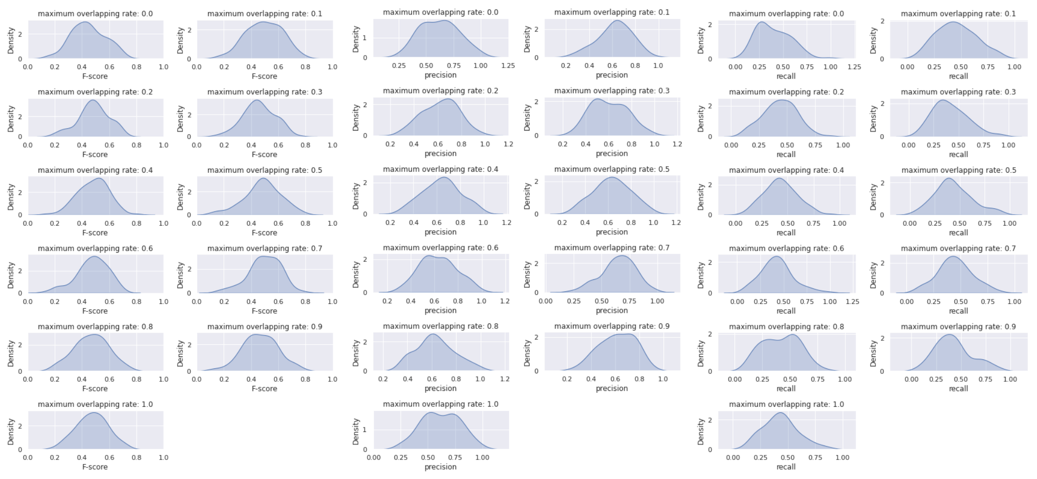
Minimum area $S_{min}^{(RSB)}$ The performance of the RSB algorithm is largely affected by the minimum area $S_{min}^{(RSB)}$ of the sampling rectangles. Since this parameter highly depends on the dataset, this parameter is set differently for each dataset. The Algorithm 11 shows that it samples k rectangles having an area of at least $S_{min}^{(RSB)}$, so a lower $S_{min}^{(RSB)}$ could result in a very small recall since it only finds a small part of points, also a worse precision because of the false high-density problem brought by the small area. In another way, a smaller $S_{min}^{(RSB)}$ is possible to lead to a high F-score when it finds correct larger rectangles by accident though the $S_{min}^{(RSB)}$ is set low, therefore, a smaller $S_{min}^{(RSB)}$ will lead to random performance, as seen in Figure 4.11(a) for New York taxi dataset, where the KDE plots of F-score of the lower minimum area have wider bandwidth. A thinner bandwidth of the F-score curve implies a more stable performance because that means the F-score is more concentrated. Also, the KDE curve chart has thinner bandwidth and it peaks at a higher F-score when the $S_{min} = \{1024, 2048\}$, but the peak of $S_{min} = 2048$ is higher. It is in accord with the actual area of the generated hotspot rectangles, which are $35 \times 35 = 1225$ and $40 \times 40 = 1600$. In Figure



(a) F-score of different minimum area $S_{min}^{(RSB)}$.
(b) Precision of different minimum area $S_{min}^{(RSB)}$.
(c) Recall of different minimum area $S_{min}^{(RSB)}$.

Figure 4.11: KDE plotting for choosing the minimum area parameter on New York taxi dataset, measured by F-score, precision, and recall.

4.11(b), RSB using lower minimum area often gains higher precision, if the sampled rectangle is small enough, say covers only one point, which happens to be a hotspot point, then the precision reaches 1.0. But in return, they will achieve lower recall, as shown in Figure 4.11(c), since the rectangles cover fewer correct points.



(a) The F-score curve of adapted DBScan on the rectangle dataset with different $\beta^{(RSB)}$.

(b) The precision curve of adapted DBScan on the rectangle dataset with different $\beta^{(RSB)}$.

(c) The recall curve of adapted DBScan on the rectangle dataset with different $\beta^{(RSB)}$.

Figure 4.12: Using data from 7 a.m. to 8 a.m. on Jan. 10th, 2015, the density functions of F-score and precison, recall are shown.

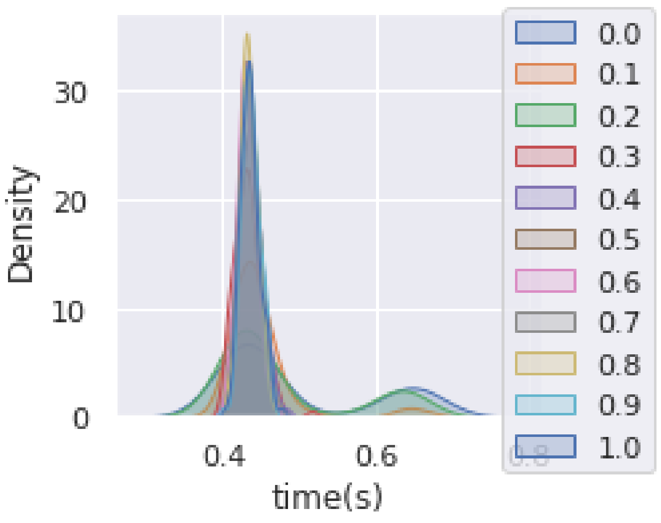
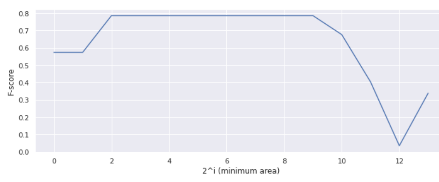


Figure 4.13: Using data from 7 a.m. to 8 a.m. on Jan. 10th, 2015, the density function of running time of different maximum overlapping rate is shown.

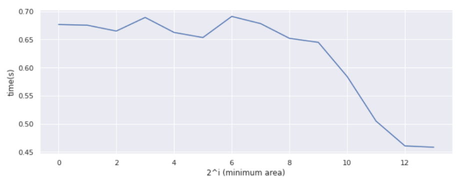
Maximum overlapping rate β^R From Figure 4.12, it is shown that the maximum overlapping rate does not affect the F-score since the density functions of all maximum overlapping rates are quite similar. Also, Figure 4.13 shows that the running time is not influenced by β^R as well. Therefore, any β^R will be fine, $\beta^R = 0.5$ is chosen here.

Adapted DBScan

Minimum area $S^{(G)}_{min}$ In Figure 4.14(a), there is a horizontal line that can not be overlooked, that is when the minimum area is from 2^2 to 2^9. The F-score of them holds still while the minimum area increases and it peaks at that range. The reason for that is that returned rectangles that have the highest density are the same when the minimum area is required to be bigger than 2^2 or even 2^9. Therefore, when the minimum area $S^{(G)}_{min}$ is set to be smaller than 4, many small rectangles that will cause false high density are able to influence the F-score to a great extent. That is also the reason and necessity of selecting this parameter. When the algorithm is able to rule out those tiny rectangles to prevent the false high-density cases, it is sufficient for it to obtain better performance. It is observed in Figure 4.14(b) that when the minimum area is



(a) F-score of different minimum area $S^{(G)}_{min}$.

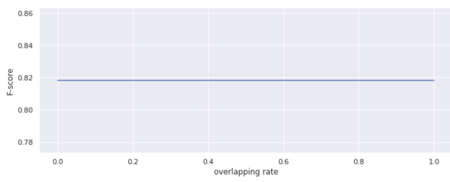


(b) Running time of different minimum area $S^{(G)}_{min}$.

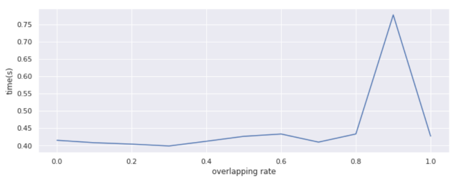
Figure 4.14: Comparison between different value of minimum area measured by F-score and running time.

bigger than 2^9, the running time decreases, while the F-score with a minimum area smaller than 2^9 fluctuates around 0.67 seconds. This is understandable since when the minimum area is big, the rectangles that meet the requirements are less, the time needed to find the rectangles is less. When the minimum area is set to be big, the F-score reduces, though running time reduces as well, it is not the best choice. Therefore, the only requirement for choosing this parameter is to exclude values that are too small. Thus, set it as the mid-value of the range from 2^2 to 2^9, since it is not an integer, take the bigger one, that is 2^6.

Maximum overlapping rate $\beta^{(RSB)}$ An interesting fact is found when running adapted DBScan algorithm on taxi set with different maximum overlapping rate, which means that any two of the rectangles can not have overlapping area more that $\beta^{(RSB)}$. The experiments are done running on the same result set



(a) F-score of different maximum overlapping rate $\beta^{(RSB)}$.



(b) Running time of different maximum overlapping rate $\beta^{(RSB)}$.

Figure 4.15: Comparison between different value of maximum overlapping rate measured by F-score and running time.

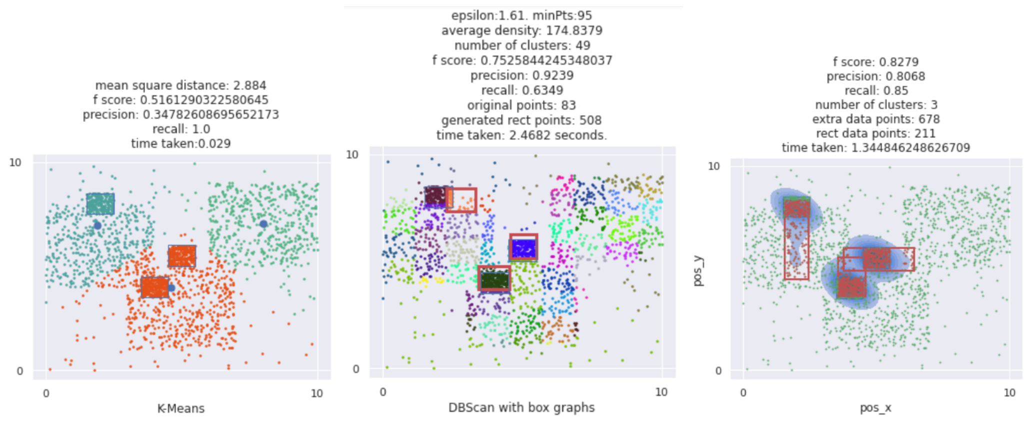
of z rectangles, and only maximum overlapping rate changes. In Figure 4.15, the F-score is a horizontal line, that is because DBScan clusters each point in one cluster so that the clusters do not overlap at all, especially when DBScan using box graphs finds more rectangular clusters, the probability for two clusters to overlap is tiny. Therefore, the $\beta^{(RSB)}$ is not one of the contributing factors. So, let $\beta^{(RSB)} = 0.5$.

4.4 Offline models

From the previous section, several datasets needed are introduced for the upcoming experiments. In this section, datasets will be used on some offline models where "offline" means that the algorithm receives data in advance when the sliding window model receives data constantly.

Here, the parameters of rectangle dataset use the values that are evaluated as the best. For instance, for apdapted DBScan algorithm on the rectangle dataset, $S^{(D)}_{min} = 1$, $\beta^{(D)} = 0.9$, for RSB algorithm on the rectangle dataset, $z = 10k = 30$, $S^{(RSB)}_{min} = 1$, $\beta^{(RSB)} = 0.5$; for apdapted DBScan algorithm on the taxi dataset, $S^{(D)}_{min} = 64$, $\beta^{(D)} = 0.5$, for RSB algorithm on the taxi dataset, they are $S^{(RSB)}_{min} = 2048$, $\beta^{(RSB)} = 0.5$, $z = 10 * k = 30$.

The performance of three clustering algorithm using the rectangle dataset is compared by comparing the F-score of each algorithm, which is shown in Figure 4.16. Different colors of points in 4.16 (a) and (b) represent various clusters, each blue big point in 4.16 (a) is the center point of each cluster. The k parameter for every algorithm is set to be 3, hence, there are three colors in 4.16 (a) and three rectangles in 4.16 (b) and (c).

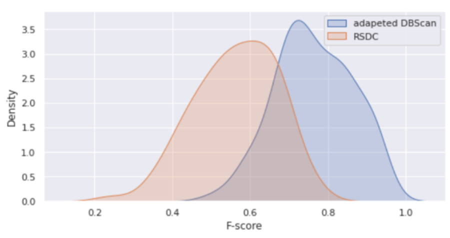


(a) k-means++ algorithm. (b) Adapted DBScan algorithm. (c) RSB algorithm.

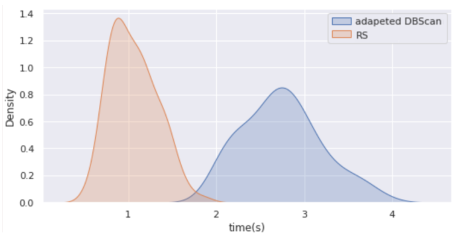
Figure 4.16: Comparison between k-means++ algorithm, adapted DBScan algorithm, RSB algorithm.

Dataset used here consists of two non-hotspot rectangles and two smaller hotspot rectangles, and they could be completely or partly overlapped, or disjoint. It is observed that the F-scores of these three algorithms are between $(0.61, 0.82)$ at this particular experiment. However, the F-score of k-means++ is only related to the proportion of hotspot points and the other points, because k-means++ algorithm clusters all points including noise points. Therefore, the recall for it is always 1.0 and the precision of it only relates to the proportion of hotspot points among all points. The DBScan using box graphs has the highest F-score, as the best clustering it gets finds the hotspots, and the rectangles are smaller so that they will not include too many non-hotspot points, so the precision and recall are both high. The result of the RSB algorithm is quite random since it samples rectangles randomly, so the performance is random. The running time taken by each algorithm is completely different, the k-means++ algorithm takes the least amount of time, the adapted DBScan algorithm comes next, and the RSB algorithm is the most time-consuming one. Nevertheless, Figure 4.16 shows one specific case, it may not tell the whole story, in Figure 4.17, the density of F-score and time cost for the adapted DBScan algorithm are given by blue shadowed curves, and those for the RSB algorithm are given in red shadowed curves.

The F-score of k-means++ algorithm holds still as a constant when the number of points does not change, so there is no need to analyze the F-score of it. The other two algorithms are compared with respect to F-score and running time. It can be observed that the F-score of the adapted DBScan algorithm is concentrated between 0.6 and 0.95, and it peaks at around 0.7. Also, the running time of it is around $[2, 4]$ seconds. At the same time, the F-score of the RSB algorithm has a similar distribution as the adapted DBScan algorithm, the majority of them are between 0.4 and 0.75, and the density peaks at around 0.6,



(a) Density of all F-score running adapted DBScan algorithm and RSB algorithm 100 times on the rectangle dataset.



(b) Density of time cost when running adapted DBScan algorithm and RSB algorithm 100 times on the rectangle dataset.

Figure 4.17: The density and time cost contribution of adapted DBScan algorithm and RSB algorithm.

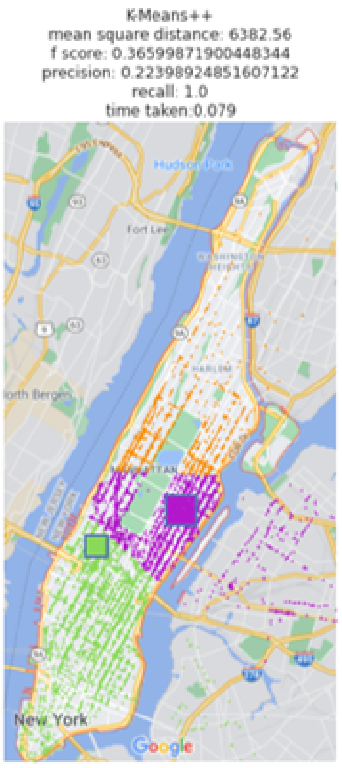
which is less than that for the adapted DBScan algorithm. However, most of them take around $[0.75, 1.5]$ seconds of running time, which is obviously less than the adapted DBScan algorithm. Therefore, the adapted DBScan algorithm clearly outperforms the other two algorithms, RSB comes next, which still has better performance than k-means++, but they both take more running time than k-means++.

Also, when searching for the best pair of ϵ and $MinPts$, the DBScan using box graphs can always find the ϵ that is very close to the width or height of the actual hotspots.

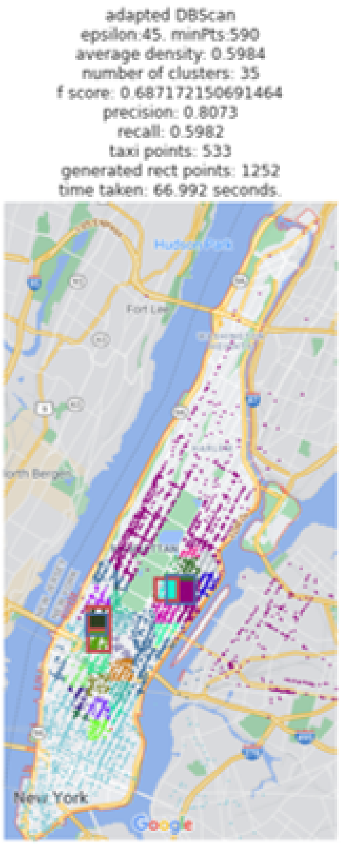
Now that the observation has been made, validation could be made by comparing three algorithms using the New York taxi dataset. For convenience, the dataset in Figure 4.2 is employed. The hotspot area simulates cases that there is an event at that particular region which leads to a large number of people calling a taxi to leave or to go because it uses the pickup data here. For example, the hotspot area is the residential area and the residents need to go to work or school during rush hour in the morning, or it is a stadium and people tend to call a taxi to leave when the concert ends. Here show the clustering results of three algorithms in Figure 4.18. It is easy to tell that the clustering result of k-means++ algorithm is less than satisfactory, it linearly separates k clusters among all points. Using the optimal parameters from the previous section, the adapted DBScan algorithm and the RSB algorithm both perform better than k-means++ algorithm. Although in 4.18(b), only one hotspot is detected completely while a small part of the other one is detected, the high precision also causes a better F-score.

Similarly, more results could provide sufficient evidence for the performance of all algorithms. The KDE plot of the F-score of running the adapted DBScan and RSB is shown for this taxi dataset in Figure 4.19. It is illustrated in 4.19(a) that the majority of the F-score of the adapted DBScan algorithm is concentrated at around 0.7, and the running time is between 46 and 61 seconds while some outliers take over 70 seconds. Conclusions can be drawn that the performance of the adapted DBScan algorithm remains stably high, but it takes more time when confronted with a larger number of data points.

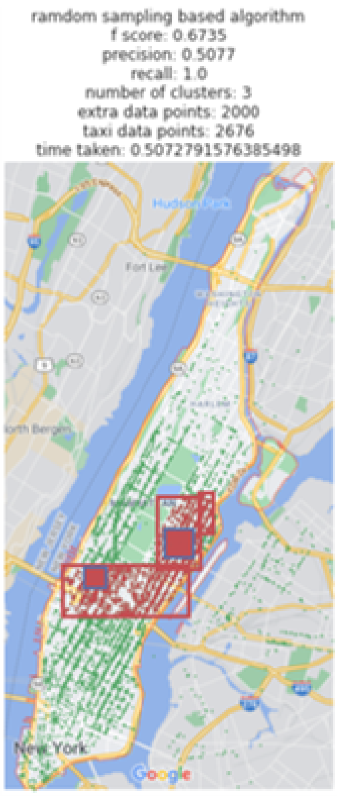
Then, the running time of a different number of points is shown in Figure 4.20, which shows that when the number of points increases, the running time of all three algorithms increases. However, the running time of adapted DBScan increases rapidly as the number of points increases.



(a) k-means++ algorithm.

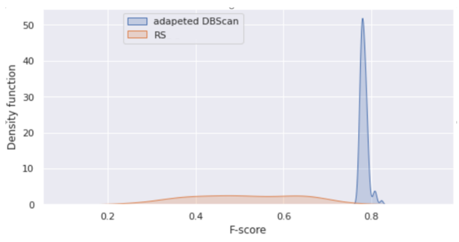


(b) Adapted DBScan algorithm.

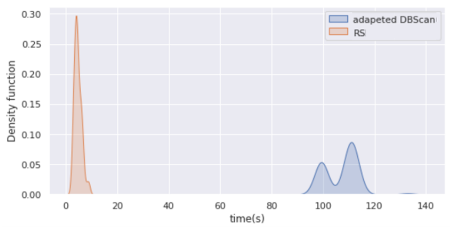


(c) RSB algorithm.

Figure 4.18: Comparison between k-means++ algorithm, adapted DBScan algorithm, RSB algorithm.



(a) The KDE plot of the F-score of running the adapted DBScan and RSB 100 times on the taxi dataset.



(b) The KDE plot of the running time of running the adapted DBScan and RSB 100 times on the taxi dataset.

Figure 4.19: The F-score and running time of data using adapted DBScan and RSB on the taxi dataset.

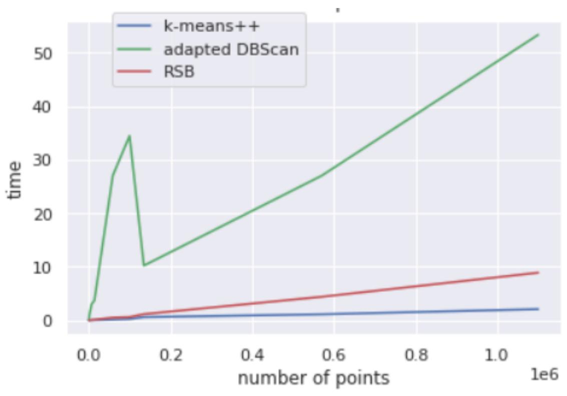


Figure 4.20: Running time for three algorithms with different number of points.

4.5 Sliding Window Model

In this section, instead of fixing the dataset in a specific time range, it uses the sliding window and moves to the next time slot every interval, so that it is able to receive data continuously. Compared to the generated rectangle dataset, the taxi dataset is more suitable to present the result for the sliding window since it collects the real data and is able to reflect some fun facts, while the timestamp of the points in the rectangle dataset is randomly generated. It is used here for the purpose of getting to know the trend of the changing distribution of taxis at a specific range of time and how the algorithm will perform when the data is changing.

To speed up the process, the dataset was sampled by a sampling rate $\mathtt{Sr_{opt}}$, set it to be 0.25, which means that approximately 25% of the data is used in the algorithm. After the algorithm running on a fraction of the data, a few rectangular hotspots are reported. In the end, F-score is computed on all points instead of only the sampled points. For example, there are 1000 points in total in $P = \{p_1, \ldots, p_{1000}\}$, which is approximately 250 points after sampling, $P_{\mathtt{sample}}$. The algorithm clusters on $P_{\mathtt{sample}}$ and returns a predicted rectangle list $\mathcal{R}_{\mathtt{predicted}}$ of 3 rectangles as hotspots. Say the generated rectangle list is $\mathcal{R}_{\mathtt{generated}}$, and points from 1000 points in all generated rectangles are presented as $\mathtt{points}(\mathcal{R}_{\mathtt{generated}})$. The precision is computed by,

$$\text{precision} = \frac{\mathtt{points}(\mathcal{R}_{\text{predicted}} \bigcap \mathcal{R}_{\text{generated}})}{\mathtt{points}(\mathcal{R}_{\text{predicted}})}.$$

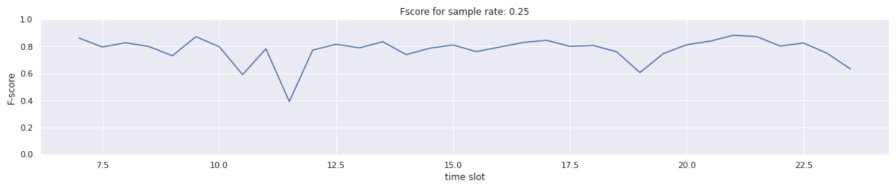
And the recall is computed by,

$$\text{recall} = \frac{\mathtt{points}(\mathcal{R}_{\text{predicted}} \bigcap \mathcal{R}_{\text{generated}})}{\mathtt{points}(\mathcal{R}_{\text{generated}})}.$$

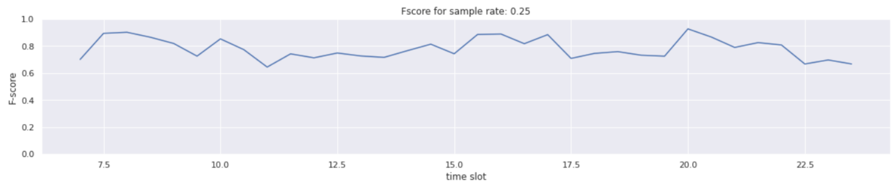
So the F-score is computed on 1000 points instead of only 250 points.

Several special cases should be shown when one attempts to figure out the hotspot condition of taxi pickups, such as at the rush hour or the off-peak hour of weekdays and weekends. Thus, four days are used as representatives, they are Friday, Saturday, Sunday, and Monday. The rush hours could be very different between weekdays and weekends, for example, taxis are often to be called at around 7 to 10 in the morning and 4 to 7 in the afternoon on a weekday, but on weekends or holidays, people tend to obtain more sleep in the morning and have fun at night. There are also some tiny differences from each day. Confronted with different densities of data of the different times of days, algorithms should have the ability to recognize the real hotspots or even the potential ones.

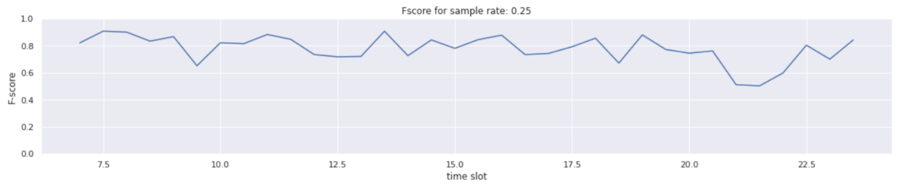
Take four specific days as examples, the F-score of data using adapted DBScan algorithm without the sliding window is shown in Figure 4.21, from which bigger fluctuations can be observed. That is because the optimal ϵ and *MinPts* are chosen only for that specific distribution of data at the beginning, but as the window slides, data changes, so the parameters are no longer suitable. Thus, another condition is that parameters are updated a few times after the observation of results that only update parameters at the beginning. The pruning time could be set to the time when the taxi data fluctuates a lot during the rapid



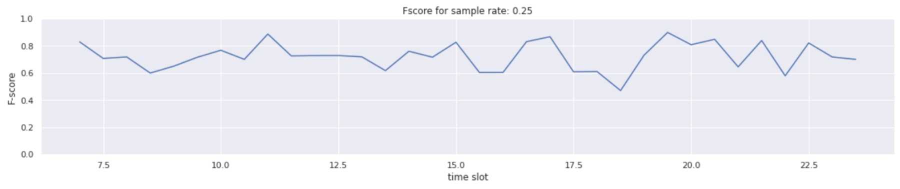
(a) Using data on Friday, January 9th, 2015.



(b) Using data on Saturday, January 10th, 2015.



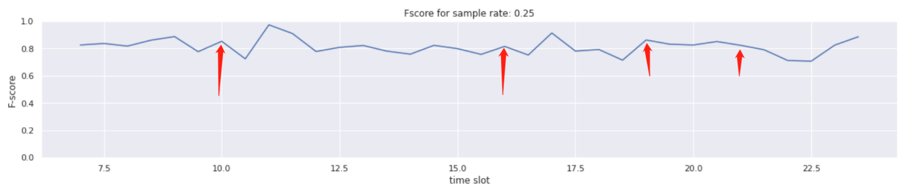
(c) Using data on Sunday, January 11th, 2015.



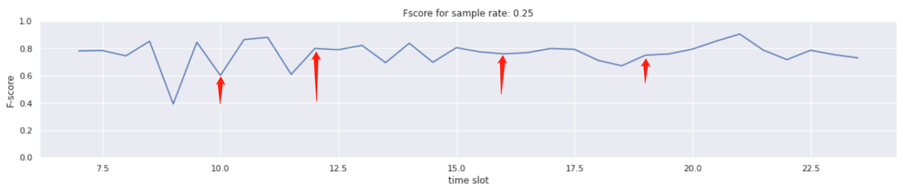
(d) Using data on Monday, January 12th, 2015.

Figure 4.21: The F-score of data using adapted DBScan in sliding window. Sliding window moves every 30 minutes, the window size is one hour. Parameters are only updated at the beginning and it does not change from then on.

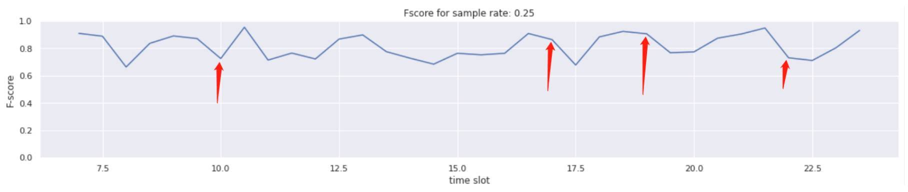
change of rush hour and after it, or the point that the F-score is incredibly low. As shown in Figure 4.22, different update time is customized for each day.



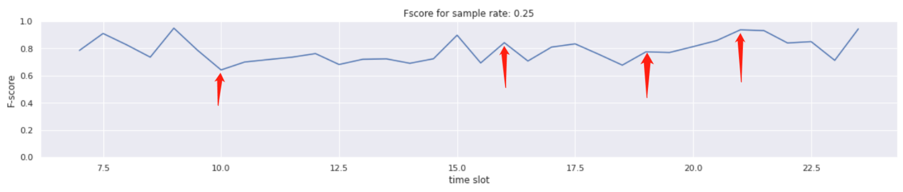
(a) Using data on Friday, January 9th, 2015. The ϵ and *MinPts* update at 10 am, 4 pm, 7 pm, 9 pm, which are indicated at the red arrows.



(b) Using data on Saturday, January 10th, 2015. The ϵ and *MinPts* update at 10 am, 12 pm, 4 pm, 7 pm.



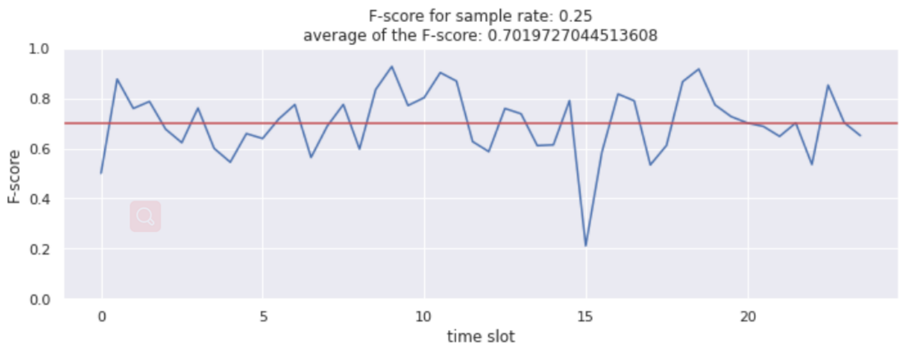
(c) Using data on Sunday, January 11th, 2015. The ϵ and *MinPts* update at 10 am, 4 pm, 7 pm, 9 pm.



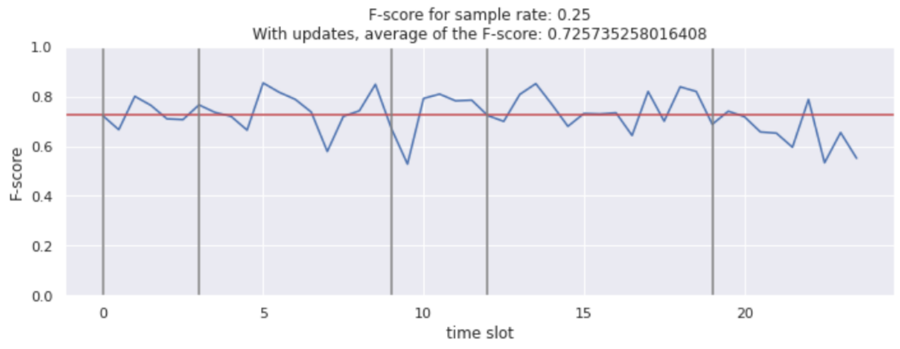
(d) Using data on Monday, January 12th, 2015. The ϵ and *MinPts* update at 10 am, 4 pm, 7 pm, 9 pm.

Figure 4.22: The F-score of data using adapted DBScan in sliding window with updating ϵ and *MinPts*. Sliding window moves every 30 minutes, the window size is one hour.

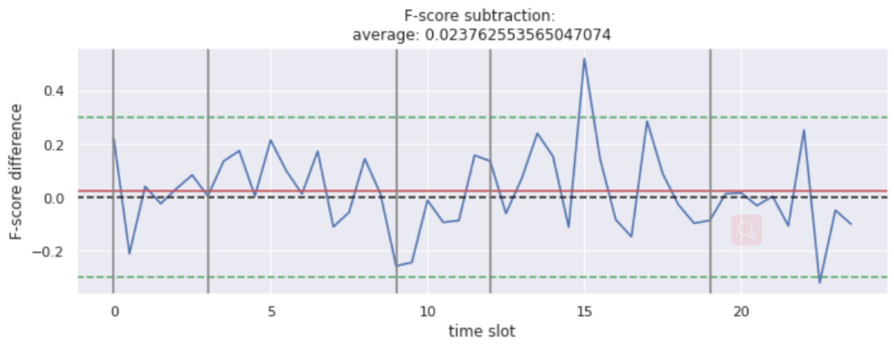
Next, taxi data of festivals, such as Christmas or new year, are also worthwhile parts to make an observation. The F-score line chart of Christmas in 2015 without updates is shown in Figure 4.23(a), from which can be seen that the F-score fluctuates more wildly within some periods, such as from 2:30 am to 8 am, from 3 pm to 9 pm, thus parameters should be updated at those specific moments in order to make fluctuations smaller, as shown in Figure 4.23(b). Despite of the fluctuation, the average F-score is high as well even without updating parameters, which means that the hotspot regions on that day change gently or even slightly, so using the same parameters is sufficient. In Figure 4.23(c), the F-score list with parameter updates subtracts that without parameter updates. The positive value in Figure 4.23(c) illustrates that parameters after updating are more suitable for that period of data, and negative values mean that the parameters without updates are better. It is clearly depicted that updating parameters improve the F-score to a great extent since lots of subtractions are bigger than zero, and most of the absolute value of positive subtractions is a lot bigger than that of negative subtractions. Moreover, most of them increase by more than 0.1. Instead of updating the parameters at some specific time, what if they are pruned at every time point, which is shown in Figure 4.24. It shows clearly that the F-score line is much smoother than that in Figure 4.23(b). Although it performs better, the running time is too much and unaffordable, especially in the sliding window model.



(a) Using taxi data on Christmas in 2015. The ϵ and $MinPts$ is not updated. A red horizontal line of the average F-score is added, which is $y = 0.7019$.



(b) Using taxi data on Christmas in 2015. The ϵ and $MinPts$ update at 0 am, 3 am, 9 am, 12 am, 7 pm, which are indicated by the gray vertical lines. A red horizontal line of the average F-score is added, which is $y = 0.7257$.



(c) Subtract the F-score list without updates from that with updates. Gray vertical lines are the update time, the black dashed horizontal line indicates $y = 0$, green dashed horizontal lines are $y = 0.3$ and $y = -0.3$, and the average of all is added as the red horizontal line, which is $y = 0.0238$.

Figure 4.23: The F-score of data on Christmas, 2015 using adapted DBScan in sliding window with updating ϵ and $Minpts$. Sliding window moves every 30 minutes, the window size is one hour.

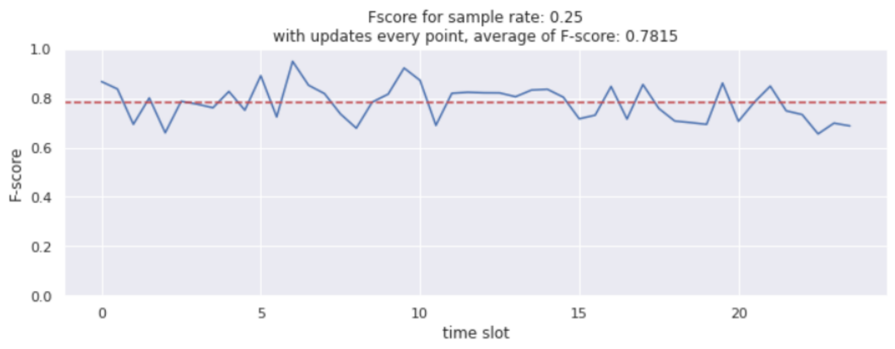
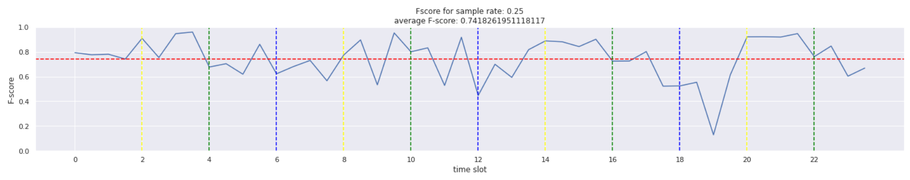


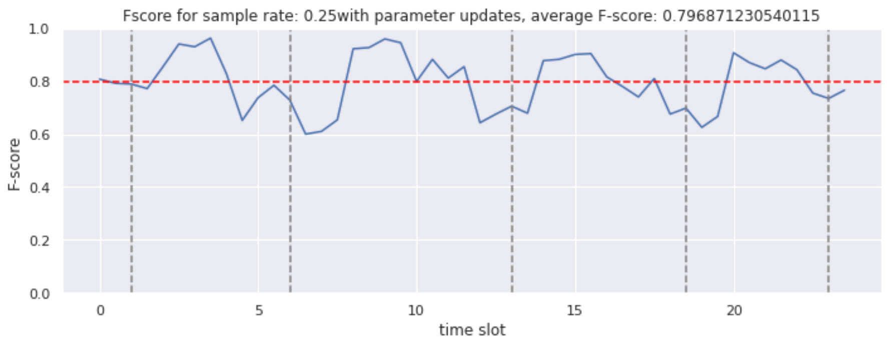
Figure 4.24: Update parameters at every point, the F-score of data on Christmas, 2015 using adapted DBScan in sliding window is much smoother.

Besides a fixed dataset of generated rectangles, experiments are done on a dataset that is planted a various number of rectangles, the F-score comparison is shown in Figure 4.25. Three of the clustering results of the adapted DBScan on Christmas, 2015 is shown, the one with only one rectangular hotspot is shown in Figure 4.26, the one with two and three rectangular hotspots are shown in Figure 4.27 and Figure 4.28.

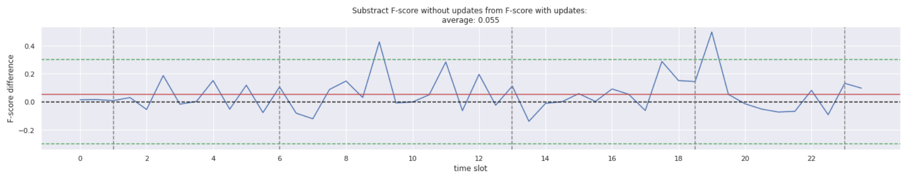
To have a better view of the hotspots of the taxi dataset, a 2-D kernel density estimate is a fairly good visualization tool. The data in Figure 4.2 is used to create a KDE plot, and some of them can be seen in Figure 4.29. Red rectangles of the figure are the predicted hotspots using sampled data points, and the F-score is computed on the whole dataset without sampling. Black rectangles are the generated hotspots, the F-score is high when all red rectangles cover only and completely all black rectangles. The main focus of Figure 4.29 is the rainbow contour. It can be noticed in 4.29(c) that there are other regions that have the same color as hotspot rectangles, which means that they have the same high density as hotspots. This is due to the fact that there are events in those regions if the density is high the whole day. If the density of those regions is high for a certain period, then they are most likely to be the business or office districts. Hence, it detects potential hotspots and it is of great importance.



(a) The dashed vertical lines are the timing that the number of rectangular hotspots changes. At the beginning, there are 2 rectangular hotspots. At the time marked by yellow, green, and blue lines, the number of hotspots changes to be 1, 2, 3, respectively.



(b) The F-score after updating at the dashed vertical lines.



(c) The subtration between the previous two figures.

Figure 4.25: Using dynmaic planted rectangle dataset on the taxi dataset on Christmas, 2015, the comparison measured by F-score is shown.

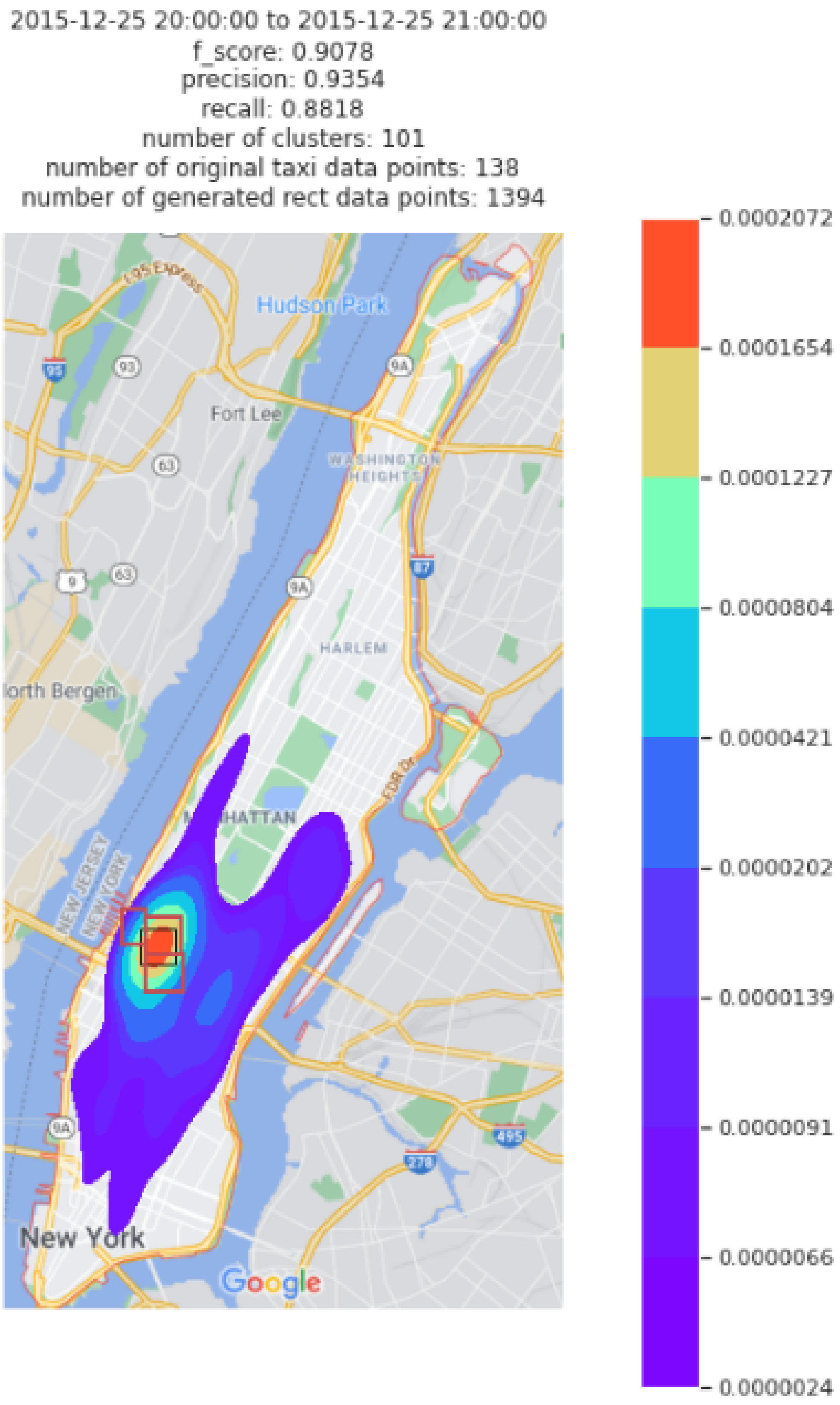


Figure 4.26: Using dynmaic planted rectangle dataset on the taxi dataset. The number of hotspot in the dataset here is onely one. In this sliding window, data is from 8 p.m. to 9 p.m.on Christmas, 2015. A 2-dimension KDE is shown.

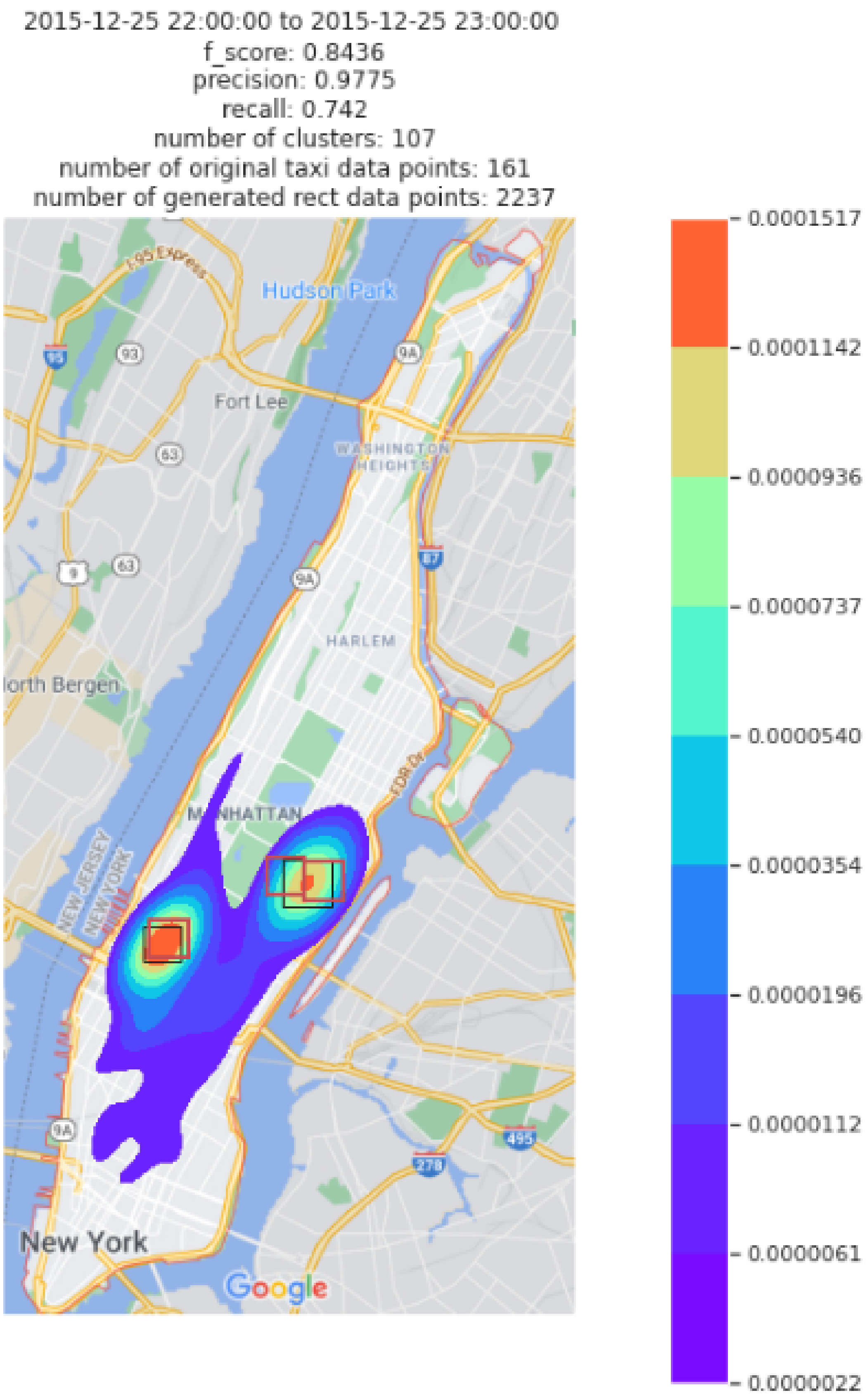


Figure 4.27: Using dynmaic planted rectangle dataset on the taxi dataset. The number of hotspot in the dataset here is two. In this sliding window, data is from 8 p.m. to 9 p.m on Christmas, 2015. A 2-dimension KDE is shown.

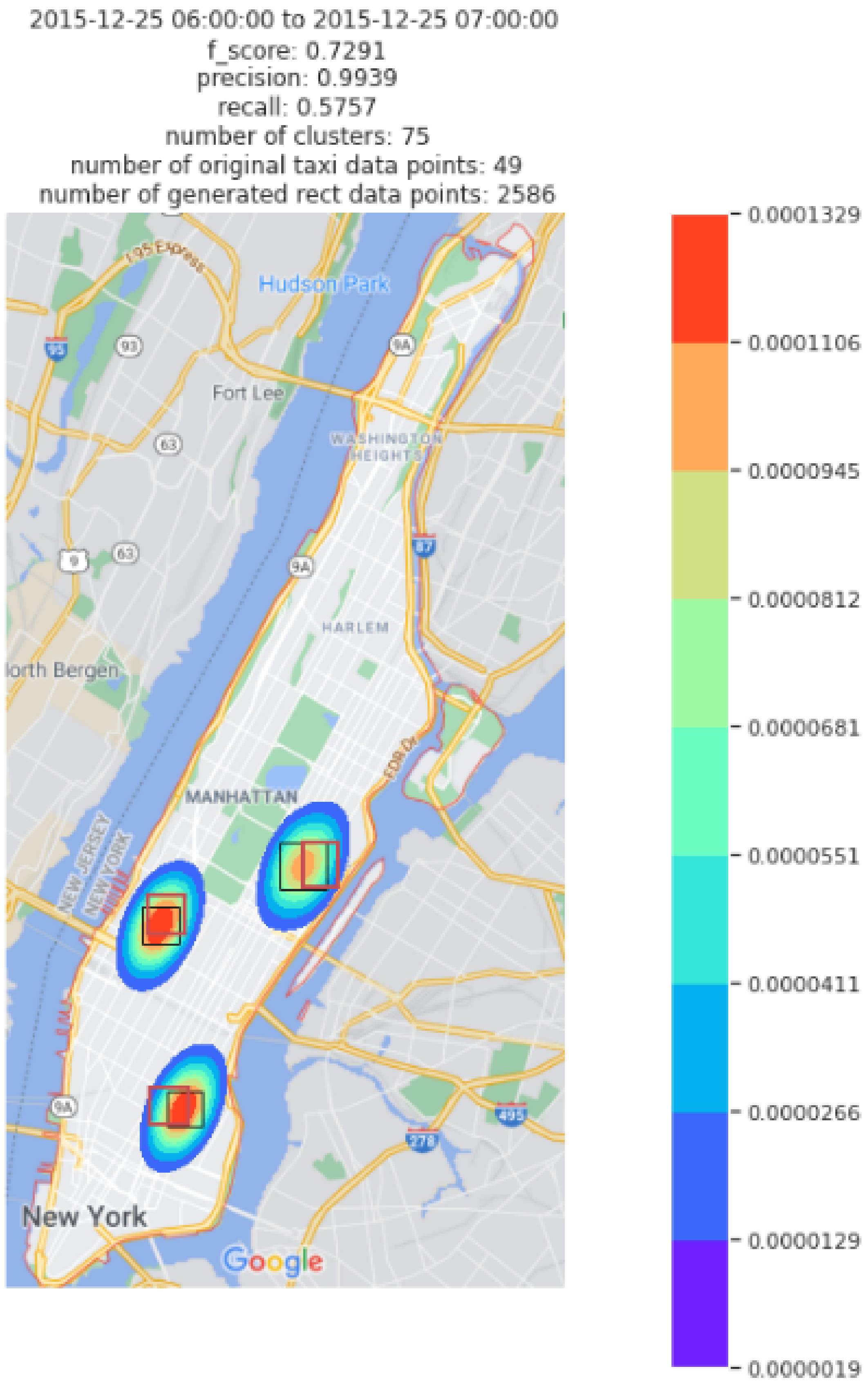
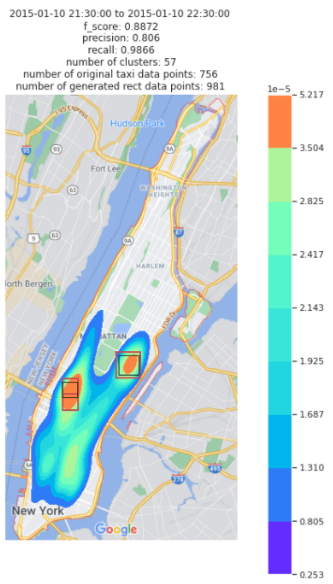
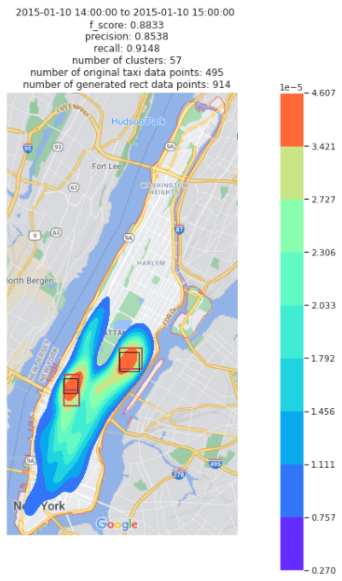


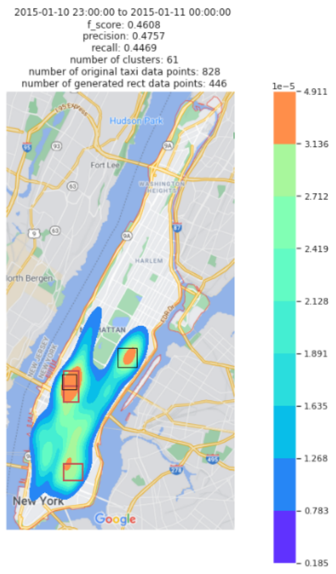
Figure 4.28: Using dynmaic planted rectangle dataset on the taxi dataset. The number of hotspot in the dataset here is three. In this sliding window, data is from 8 p.m. to 9 p.m.on Christmas, 2015. A 2-dimension KDE is shown.



(a) The starting time of this sliding window is 9:30 p.m., the window size is one hour. The red color represents the densest area. In this subgraph, three predicted rectangles highly overlapped with the generated ones, which means that they have higher precision and recall, which results in a higher F-score.



(b) The starting time of this sliding window is 2 p.m., the window size is one hour. In this subgraph, the yellow/orange area covers more area, which means that lots of the places have higher density.



(c) The starting time of this sliding window is 11 p.m., the window size is one hour. In this subgraph, there is another area as dense as the generated hotspots, which is the potential hotspot. It is detected by the algorithm, though it leads to a low F-score.

Figure 4.29: Using data on Saturday, January 10th, 2015, without the updates. Red rectangles are the predicted rectangular hotspots, and the black rectangles are the generated hotspots.

Chapter 5

Conclusions

By the result of thesis we have shown that

- **Random Sampling vs. Adapted DBSCan:** Random sampling where we sample a set of pair of points and then find the most qualitative rectangles is faster than the adapted DBScan that we propose in this thesis. However, the result that we obtain using the adapted DBScan has a better performance in terms of precision, recall.
- **Randomness in Random Sampling:** In the random sampling approach we randomly sample pairs of points and so, the result of multiple runs might not be the same. On the other hand, the adapted DBScan is a deterministic algorithm and thus, the result of multiple runs are similar.
- **Adapted DBScan in the sliding windows model:** We observe that if we frequently learn the parameters of the adapted DBScan, the performance of the adapted DBScan is much better than if we train its parameters only at the beginning and fix it during the stream. This is mainly because different windows in the sliding windows model may have different hotspots (e.g., the hotposts during the runsh hour might be different than the hotspots during the normal hours, therefore, if we train the parameters of the adapted DBScan, the algorithm can adapt itself to dynamic changes). However, it is not clear to us how often we need to train the parameters of the adapted DBScan for different datasets. One approach would be if we see that the reported hotspots are deviating from ones we have been reporting, this may be a hint that we should train the parameters again.
- **Semi-supervised learning:** In general, detecting hotspots is an unsupervised learning where we do not have any label for the correct and incorrect hotspots. But we could use labels when training to evaluate the algorithms. We think having a partial data that our algorithm can use would help the performance of our algorithms.

Acknowledgment

I want to give big thanks to my supervisor Morteza Monemizadeh who has been provided lots of advice and I have learned a lot from him. Also, I am grateful to the committee members Marcel Roeloffzen and Nikolay Yakovets, thank you for attending my defense. Last but not least, thanks to my super supportive families who have offered me so much patience and love, and always care about me even if I was abroad. Especially my mother, Hongjuan Song, it is her strong willpower that influences me and also pushes me all the time.

Bibliography

[1] Hot Spot wikipedia description. https://en.wikipedia.org/wiki/Hot_spot_(computer_programming). Accessed: 2021-09-30. 19

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006. 6, 7

[3] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 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, 2006. 5

[4] Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2291–2304. SIAM, 2019. 5

[5] Moses Charikar, Vaggos Chatziafratis, Rad Niazadeh, and Grigory Yaroslavtsev. Hierarchical clustering for euclidean data. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 2721–2730. PMLR, 2019. 5

[6] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. 15

[7] Mark de Berg, Ade Gunawan, and Marcel Roeloffzen. Faster db-scan and hdb-scan in low-dimensional euclidean spaces. *arXiv preprint arXiv:1702.08607*, 2017. 5

[8] Drleft at English Wikipedia. Comparison of 1d histogram and kde, 2010. [Online; accessed November, 2021]. iv, 17

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996. 5, 6, 8

[10] Junhao Gan and Yufei Tao. Dbscan revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 519–530, 2015. 5

[11] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE transactions on knowledge and data engineering*, 15(3):515–528, 2003. 5

[12] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. *ACM Sigmod record*, 27(2):73–84, 1998. 5

[13] Ade Gunawan and M de Berg. A faster algorithm for dbscan. *Master's thesis*, 2013. ii, 4, 5, 6, 19

[14] Manoj Kumar and Ashish Sharma. Mining of data stream using “ddenstream” clustering algorithm. In *2013 IEEE International Conference in MOOC, Innovation and Technology in Education (MITE)*, pages 315–320. IEEE, 2013. 5

[15] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. 6

[16] Nass, Steven. How to steal a election, 2015. [Online; accessed November, 2021]. iv, 3

[17] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962. 16

[18] Patrizio Pelliccione, Eric Knauss, S. Magnus Ågren, Rogardt Heldal, Carl Bergenhem, Alexey Vinel, and Oliver Brunnegård. Beyond connected cars: A systems of systems perspective. *Science of Computer Programming*, 191:102414, 2020. 1

[19] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020. 15

[20] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956. 16

[21] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017. 5

[22] Julius Strake, Franz Kaiser, Farnaz Basiri, Henrik Ronellenfitsch, and Dirk Witthaut. Non-local impact of link failures in linear flow networks. 21(5):053009, may 2019. 1

[23] J Tuzo Wilson. A possible origin of the hawaiian islands. *Canadian Journal of Physics*, 41(6):863–870, 1963. 19