DBSCAN+, RealVecLen-DBSCAN+, ZPNVecLen-DBSCAN+ with respect to the Tanimoto measure

Project Report

Data Mining

Kacper Kania

Contents

1 Description of the task

The goal of the task was to gain new knowledge about one of the most popular clustering algorithm - DBSCAN [1]. To familiarize myself with the algorithm, I implemented a modification, dubbed DBSCAN+. It is the main task in the project. In comparison to the basic form[1], "+" in the name stands for the following modification:

A border point of a single cluster can be associated with multiple other clusters for which it is also a border point.

Since the clustering assumes a particular distance measure, I used Tanimoto measure proposed by Kryszkiewicz [2]. Additionally, finding neighbors of core points in the algorithm can be sped up by using triangular inequality. However, Tanimoto measure is not a proper distance metric, therefore triangle inequality cannot be applied in this case. It can be approximated approximated however using methods RealVecLen [2] (if data points are real valued) and ZPNVecLen [2] (if data points are ternary valued).

Overall, the project contains an implementation of DBSCAN+ algorithm with respect to the Tanimoto measure with neighborhood search improvements. The task was done by implementing the following constituents:

- Base form of DBSCAN algorithm
- DBSCAN+ modification
- Tanimoto measure
- Filtering neighbors according to the RealVecLen and ZPNVecLen heuristics
- Experimentation procedure that included loading and processing the data, calculating quality measures (purity, silhouette, RAND, Davies–Bouldin index) and saving obtained results to files.[2]

2 Made assumptions

To solve the described task, I assumed the following:

- To accurately represent data, it is sufficient to use `double` data type for real valued vectors and to calculate similarity measures and values of quality metrics.
- For each dataset, I used its original values without any preprocessing.
- Both purity and RAND are calculated assuming that "noise" label of points creates yet another cluster to be considered.
- RAND measure incorporates the fact that a border point can belong to many clusters - true positive count is increased if true labels of both points are the same and the intersection of cluster assignments is not empty for these points. For the true negative increment - their true labels have to differ and the intersection of their cluster assignments has to be empty.
- Since ZPN datasets are not easily obtainable, I converted downloaded real-valued datasets using Python scripts.

[1] I assume that the reader is knowledgeable of the internal workings of the basic DBSCAN.

[2] The project source code is available at: `https://github.com/kacperkan/dbscan-zpn-realvec-len`

3 Tanimoto measure analysis

While the Euclidean measure is natural to humans as a basic distance measure, it is unclear how the Tanimoto measure behaves for different localization of data points. Its form is shown in Equation 1

$$d(u, v) = \frac{u \cdot v}{|u|^2 + |v|^2 - u \cdot v} \tag{1}$$

To understand the Tanimoto measure more closely, I visualized how its value changes depending on lengths of abstract vectors u and v, and angle α between them. In Figure 1, I show Tanimoto measure values for lengths $|u|, |v| \in [0, 50]$ and 6 linearly spaced angles $\angle \in [0, \pi]$.

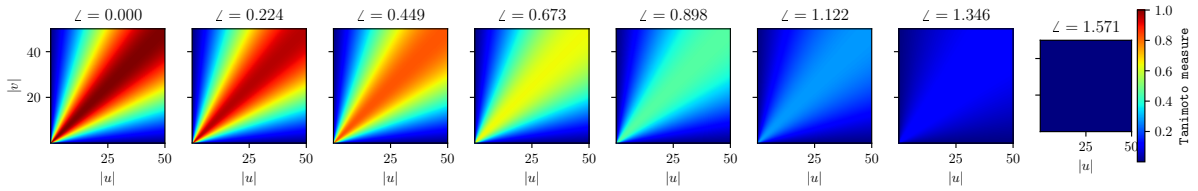


Figure 1: Visualization of the Tanimoto measure values, depending on lengths of two vectors and the angle between them.

It is noticeable, that the Tanimoto measure changes more slowly if two vectors are further away from the point $(0, 0)$. It can be interpreted as being more tolerant to small changes in vector values if they are further away. Additionally, when two vectors are parallel but have opposite directions, their Tanimoto measure value does not change, no matter how long these vectors are.

4 Description of the form of input and output data

To facilitate using existing, well-known benchmark datasets, the implemented program expects a file with data in the following format:

```
x11,x12,x13,...,x1d,label1
x21,x22,x23,...,x2d,label2
...
xn1,xn2,xn3,...,xnd,labeln
```

where `label1,...,labeln` can be in any form. Labels are mapped inside the program to appropriate integer values. While not necessary, it is recommended to store the data in a file with the `*.arff` format. The file may contain comment lines, preceded either by `@` or `%` characters. The program accepts a path to the file and processes it appropriately. To ease using the program, it also handles the file even if labels `label1,...labeln` are not given, *i.e.* the line finishes with the last feature value.

5 Important design and implementation issues

5.1 Class diagram

The main part of the project is implemented according to the class diagram presented in Figure 2.

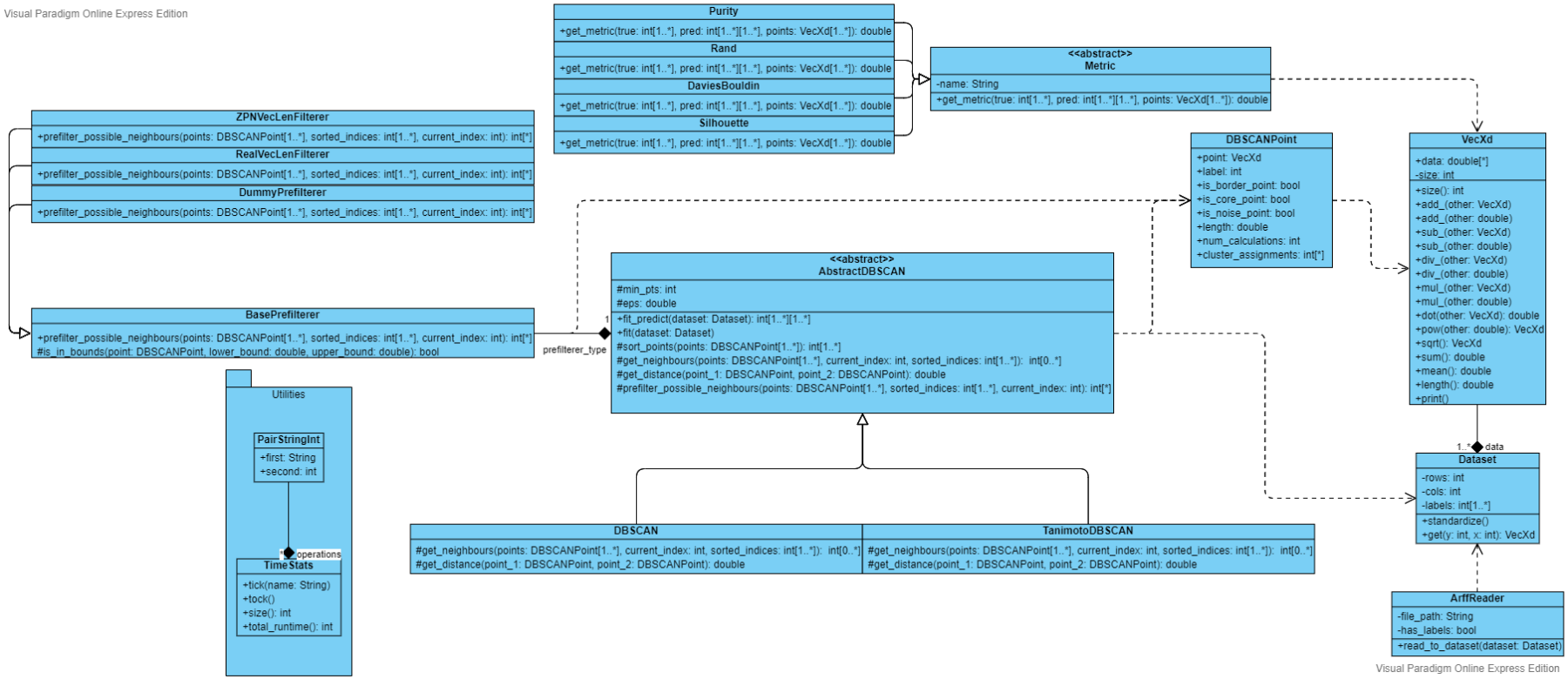


Figure 2: The class diagram of the main part of the project. Auxiliary utilities are omitted for clarity.

5.2 Considering border points

DBSCAN+ differs from the standard DBSCAN algorithm. It allows all border points to have multiple classes assigned to them. It means that a single point can be a border point of multiple clusters. To include that feature, the following changes were necessary: ① A single point has an attribute `cluster_assignments` represented by `std::unordered_set`. This facilitates a fast assignment insertion and deletion. ② When a point is considered as a core point or a noise point, all its previous assignments are removed so it can hold only a single label. ③ When neighborhood of a core point is processed, its label is inserted to the `cluster_assignments` of the neighboring point if the number of that point's neighbors is less then `min_pts`. In principle, instead of assigning a new label to a border point, the new label is put into the list all possible cluster assignments of that point. These changes are summarized in Algorithm 1.

5.3 Points prefiltering

Filtering of points in RealVecLen and ZPNVecLen that can be considered as neighborhood of a point is performed according to Algorithm 2. It is assumed that both heuristics implement `is_in_bounds` method that calculates lower and upper bounds specific to their definition.

Algorithm 2: Filtering points before including them as a neighborhood.

Data: ε, data points $\mathbf{X}$ according to their lengths, current point index i
Result: Indices $\mathbf{Y}$ of points to be considered as a potential neighbors
$stop\_up \leftarrow false$;
$stop\_down \leftarrow false$;
$up\_index \leftarrow i + 1$;
$down\_index \leftarrow i - 1$;
while *true* **do**
  if $up\_index < |\mathbf{X}|$ ***and not*** $stop\_up$ **then**
    if `is_in_bounds`$(X[up\_index], \varepsilon)$ **then**
      `push`$(\mathbf{Y}, up\_index)$;
    else
      $stop\_up \leftarrow true$;
    $up\_index \mathrel{+}= 1$;
  if $down\_index \geq 0$ ***and not*** $stop\_down$ **then**
    if `is_in_bounds`$(X[down\_index], \varepsilon)$ **then**
      `push`$(\mathbf{Y}, down\_index)$;
    else
      $stop\_down \leftarrow true$;
    $down\_index \mathrel{-}= 1$;
  $stop\_up \leftarrow up\_index \geq |\mathbf{X}|$ ***or*** $stop\_up$;
  $stop\_down \leftarrow down\_index < 0$ ***or*** $stop\_down$;
  if $stop\_down$ ***or*** $stop\_up$ **then**
    break;

5.4 Additional information

During the experimentation, I used additional helper scripts written python that facilitated my workflow and debugging C++ code. I used them to: • plot data points and color them according to the assigned cluster label • convert real valued features of data points to ternary valued data points • creating artificial datasets

Algorithm 1: DBSCAN+ algorithm

```
Data: ε, min_pts, X
cls ← −1;
sort_lengths(X);
for x ∈ X do
    if x.label ≠ UNDEFINED then
        continue;
    neigbhors ← get_neighbors(x, X, ε);
    if |neigbhors| < min_pts then
        x.label ← NOISE;
        push(x.cluster_assignments, NOISE);
        continue;
    cls ← cls + 1;
    x.label ← cls;
    x.core ← true;
    push(x.cluster_assignments, cls);
    while |neigbhors| > 0 do
        n ← pop(neighbors);
        if n.label = NOISE then
            n.label ← cls;
            n.core ← false;
            n.border ← true;
            erase(n.cluster_assignments, NOISE);
            push(n.cluster_assignments, cls);
        if n.label ≠ UNDEFINED then
            if not n.core then
                push(n.cluster_assignments, cls);
            continue;
        n.label ← cls;
        neigbhors2 ← get_neighbors(n, X, ε);
        if |neigbhors| ≥ min_pts then
            x.core ← true;
            x.border ← false;
            clear(x.cluster_assignments);
            push(x.cluster_assignments, cls);
            neighbors ← neighbors ∪ neighbors2;
        else
            x.core ← false;
            x.border ← true;
            push(x.cluster_assignments, cls);
```

When calculating quality metrics, I added additional term equal to 10^{-8} in the denominator to avoid dividing by 0.

6 User guide

6.1 Main experiments

The project is prepared in the form of a docker[3] container. The container system facilitates distribution of the code. It allows the user to install all the necessary dependencies of the project without interacting with the host system. To reproduce experiments described in Section 7, the reader has to run the following commands[4]:

```
$ docker build -t dami .
$ docker run -it --rm -v \
    $(pwd)/experiments:/usr/src/dami/experiments dami
```

The command just runs `run_experiments.sh` under the hood. If desired, one can run this script on the host system. Expected results are the same from running the command in docker and running the script on the host.

Once the experiments are finished, results and all associated artifacts will be stored in the `experiments/` folder. Each folder is named with the following convention:

```
<dataset-name>_<epsilon>_<min-pts>_<algorithm-name>_<repeat>_<heuristic>
```

where these parts denote:

- `<dataset-name>` Name of the used dataset, assumed to be the as the folder name where that dataset is located
- `<epsilon>` Value of used ε parameter.
- `<min-pts>` Value of used minimum points in the neighborhood of a point to treat it as a core point.
- `<algorithm-name>` Can be either `base` and `tanimoto`. The former describes that basic DBSCAN+ was used with the Euclidean distance metric, and the latter - assumes using the Tanimoto measure.
- `<repeat>` Ordinal number of the experiment with the same set of parameters.
- `<heuristic>` Used heuristic to filter out considered neighbors during the DBSCAN+ run when the Tanimoto measure is used. Can be either `none` (no prefiltering), `realveclen` (RealVecLen prefiltering is used) and `zpnveclen` (ZPNVecLen prefiltering is used)

Each folder will consist of `OUT` and `STAT` files, containing predictions and run statistics respectively.

6.2 Running individual experiments

If desired, the user can run experiments individually or even without docker. To achieve this, the following steps are required.

[3] `https://www.docker.com/`

[4] At this point, I assume that the user has already installed docker.

6.2.1 Binary compilation

Firstly, the user has to compile the program. Commands are for Linux-based systems. To achieve the same result on Windows, I recommend to use either Windows Subsystem for Linux or one of `cygwin` distributions. The compilation requires that these programs are already installed on the host system:

- `CMake`
- `Boost` and `Boost-IOStreams`
- `GCC` compiler

Then, do the following:

1. Create a build directory with `mkdir build`
2. `cd build`
3. `cmake ..`
4. `make`

These commands will create a binary named `dbscan_zpn_realvec_len` in the project directory.

6.2.2 Executing the binary

Once compiled, the program can be run as:

```
./dbscan_zpn_realvec_len <dataset-file> <experiment-name> \
    --eps <eps> \
    --min_pts <min-pts> \
    --algorithm_name <algorithm-name> \
    --prefiltering_name <prefiltering-name> \
    [--has_labels] \
    [--standardize] \
    [--verbose]
```

where:

- `<dataset-file>` is a path to the dataset file in the `*.arff` format
- `<experiment-name>` is the name of the experiment. The name is used to create the directory in the `experiments/` to put experiment artifacts there
- `<eps>` is the real value of the ε parameter
- `<min-pts>` is a positive integer number of minimum points in the neighborhood of a point to consider it as a core point
- `<algorithm-name>` is the name of the algorithm to use, either basic DBSCAN+ with Euclidean distance or with Tanimoto measure (note that the ε interpretation is different between these cases). Can take value `base` or `tanimoto`
- `<prefiltering-name>` is the heuristic name used to prefilter neighbors when Tanimoto measure is applied (in case of the `base` algorithm, no prefiltering is used). Can be `none`, `realveclen` or `zpnveclen`

- `--has_labels` required if the dataset consists of ground truth labels associated with each point in the dataset
- `--standardize` if used, it normalizes each feature in the dataset to have 0 mean and 1 standard deviation
- `--verbose` optional flag that turns printing all messages on when the algorithm is running

When the experiment is finished, appropriate `OUT` and `STAT` files are put in the `experiments/<experiment-name>` folder.

If desired, a single experiment can be run from the docker environment. To achieve this, the binary running command should be prepended with:

```
docker run -it --rm -v $(pwd)/experiments:/usr/src/dami/experiments dami
```

so the whole command is in the form:

```
docker run -it --rm -v $(pwd)/experiments:/usr/src/dami/experiments dami \
    ./dbscan_zpn_realvec_len <dataset-file> <experiment-name> \
        --eps <eps> \
        --min_pts <min-pts> \
        --algorithm_name <algorithm-name> \
        --prefiltering_name <prefiltering-name> \
        [--has_labels] \
        [--standardize] \
        [--verbose]
```

6.3 Additional scripts

Additional scripts that facilitated preparing the project written in python:

- `python/make_dataset_zpn.py` Processes existing real valued dataset to the one with ternary values. It can be run as:

  ```
  python python/make_dataset_zpn.py \
      <path-to-the-dataset> \
      <output-path-were-processed-data-will-be-stored> \
      --thresholds <min> <max> \
      --labels <neg-label> <zero-label> <pos-label> \
      [--has_labels]
  ```

  where `<min>` and `<max>` denote real values that create 3 bins encompassing ranges of $[-\infty, \texttt{<min>}]$, $[\texttt{<min>}, \texttt{<max>}]$ and $[\texttt{<max>}, \infty]$. Depending where a particular feature value lands, it will be converted to one of labels defined with the flag `--labels`. The optional flag `--has_labels` denotes if a dataset file consists clustering labels for each of the data points. Example usage can be found in `download_data.sh` script.

- `python/plot_single.py` Plots data points on a canvas and colors each data point according to the label associated with it. It works on `OUT` files produced during experiments. Usage:

  ```
  python python/plot_single.py <path-to-the-OUT-file>
  ```

The script produces an image file that is placed in the same folder as the corresponding `OUT` file.

- `python/produce_example_data.py` Creates a dataset consisting of three gaussian blobs placed around the $(0, 0)$ coordinate. Usage:

```
python python/produce_example_data.py <output-folder>
```

The dataset is additionally visualized and saved in the folder denoted in the `<output-folder>` variable. It is in the format expected by the main program.

Dependencies required to use these scripts can be found in `python/requirements.txt` file and installed with the following command:

```
pip install -r python/requirements.txt
```

7 Experiments

The experiments served to analyze and understand the behavior of the DBSCAN+ algorithm with respect to both Euclidean and Tanimoto measures. Firstly, I show qualitative results on an artificial dataset composed of points sampled from a mixture of gaussians. Secondly, I present results on `Cluto`, `Complex9` and `Letter` datasets[5]. These datasets allowed me to explore how values parameters of the DBSCAN algorithm affect obtained results. These parameters include:

- ε - parameter denotes a measure value between two data points to consider them as neighbors
- `minPts` - number of points to consider a point as a core point of any cluster

Then, I show how filtering heuristics influence computational complexity of the DBSCAN+ fitting in terms of number of data point evaluations and the runtime measured in milliseconds. In last two experiments, I show results obtained with the best chosen parameters and comparison between C++ and Python implementations of the same algorithm. Since one of these heuristics requires the datasets to be ternary valued, I explicitly converted each of these datasets so data points features can contain of values from $\{-1, 0, 1\}$. Values of thresholds to obtain ternary valued datasets are presented in Table 1.

Table 1: Thresholds for each datasets that were used to obtain ZPN datasets. Using these thresholds result in each feature having on values from $\{-1, 0, 1\}$. If a feature had a value $v < \epsilon_{\texttt{low}}$ then took value -1, if $\epsilon_{\texttt{min}} \leq v < \epsilon_{\texttt{max}}$ then 0, and 1 if $\epsilon_{\texttt{max}} \leq v$. These values were chosen arbitrarily according to the distribution of values in each dataset.

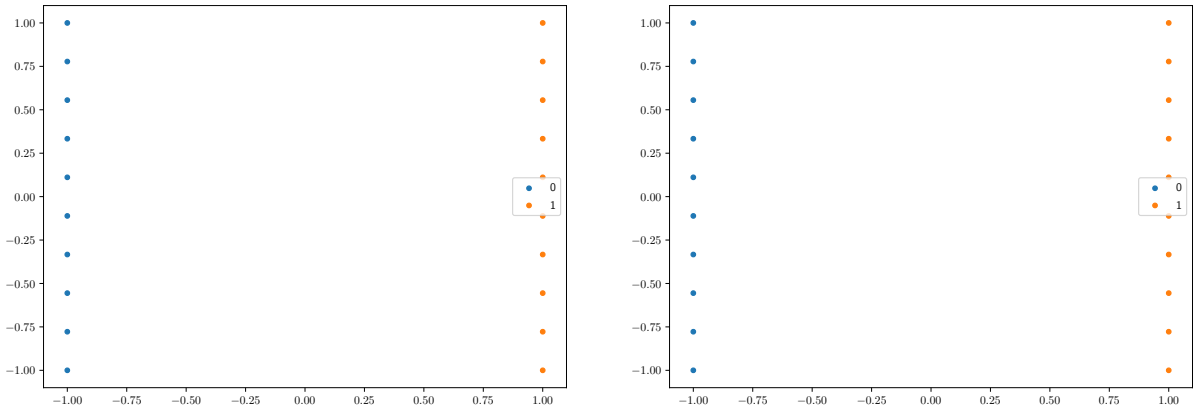
Dataset	$\epsilon_{\texttt{min}}$	$\epsilon_{\texttt{max}}$
`Cluto`	250	500
`Complex9`	100	500
`Letter`	3	7

Quality of the clustering was measured with the following metrics: Purity, RAND, Silhouette, Davis-Bouldin. Each experiment was performed once due to computational limitations.

[5]Available here: `https://github.com/deric/clustering-benchmark/tree/master/src/main/resources/datasets/artificial`

7.1 Empirical validation of the implementation

To validate the implementation, I prepared two two-dimensional datasets. One of them, consists of 10 points that are placed as follows. Five of these points share the same feature value and they are linearly spaced in the range $[-1, 1]$ along the second feature. The other five are mirrored along the axis of the second feature. The second dataset consists of 750 data points. Each data point is sampled from a mixture of three gaussians placed at $\{1, 1\}$, $\{-1, -1\}$ and $\{1, -1\}$ where each gaussian has a standard deviation equal to 0.4. The dataset was standardized during the algorithm run. Clustering results are presented in Figure 3 for the first described dataset and in Figure 4 for the second one. I also show results when ZPN vectors are used in the case of the second dataset. Features were converted to label -2 if their values $v < -1$, to label 0 when $-1 \leq v < 1$, and to label 1 otherwise.



(a) Euclidean, $\varepsilon = 0.3$, `minPts` $= 1$

(b) Tanimoto, $\varepsilon = 0.9$, `minPts` $= 0$

Figure 3: Clustering results on the artificial results with points placed on oposite sides to each other along an axis of one of features for both Euclidean and Tanimoto measures. Parameters of the DBSCAN+ were set appropriately for each measure. Points are colored according the classes assigned to them, described in the displayed legend.

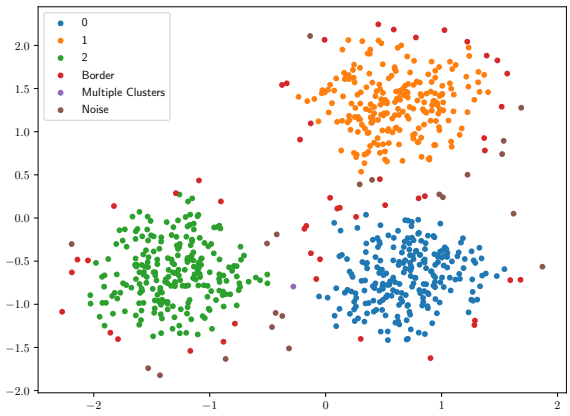
To consider also more structured dataset, I used `Complex9` dataset consisting of 10 000 data points. The clustering result of the dataset is presented in Figure 5.

7.2 Influence of the ε parameter

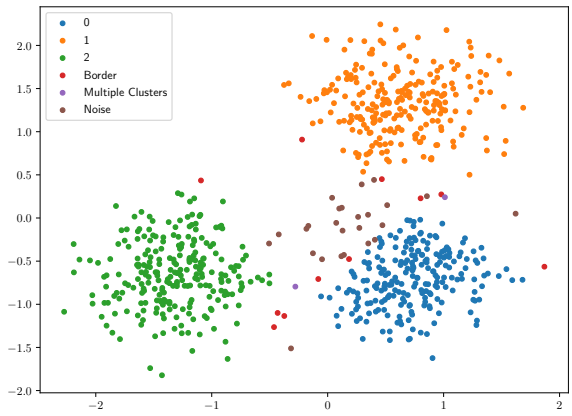
Tables 2,3,4 show obtained results for `Cluto`, `Complex9` `Letter` datasets respectively. In each experiment, only the value of ε was changed and `minPts` was set to 5. It is worth noting that while for the Tanimoto measure $\varepsilon \in [0, 1]$, for the Euclidean distance it is $\varepsilon \in \mathbb{R}_+$.

One can notice that there is no specific rule what value of the ε gives the best result. Moreover, slight changes of the value results also in high variability of the metrics values. These two issues arise from the problem of quality measurement in clustering - there exist no one good metric that can objectively tell about quality of the clustering.

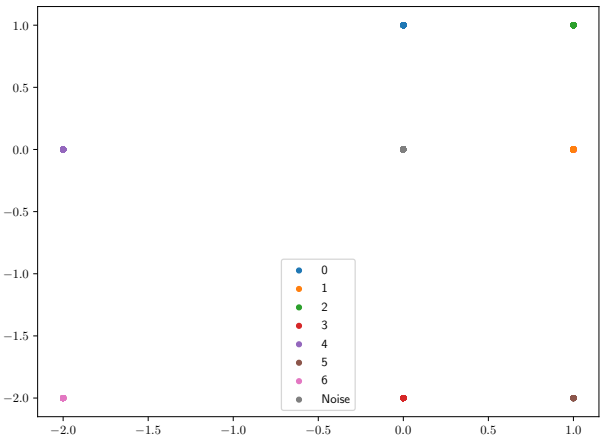
Another issue emerges for using Euclidean distance in clustering. Any value $\varepsilon \in [0, 1]$ leads to obtaining the worst metric, often the same for all parameter values. This comes from the unstandardized data, and due to the scale of the data points, one would have to tune explicitly possible parameters of values. Please note that for the standardized data ε has a similar (but not the same) effect in Euclidean space as $1 - \varepsilon$ in Tanimoto space as was found empirically.



(a) Euclidean, $\varepsilon = 0.3$, `minPts` $= 10$. Number of points with multiple assignments: 0.



(b) Tanimoto, $\varepsilon = 0.9$, `minPts` $= 10$. Number of points with multiple assignments: 2.



(c) Tanimoto with ZPN vectors, $\varepsilon = 0.9$, `minPts` $=$ 10. Number of points with multiple assignments: 0.

Figure 4: Clustering results on the dataset made of 750 points sampled from three for both Euclidean and Tanimoto measures. Parameters of the DBSCAN+ were set appropriately for each measure. Points are colored according the classes assigned to them, described in the displayed legend. The "Multiple Clusters" label means that a border point was assigned to more than one cluster.

Table 2: Evaluation results for different clustering quality measures depending on the value of ε when either Euclidean (Euc) distance measure or Tanimoto (Tan) measure is used for the `Cluto` dataset. It is assumed that `minPts` $= 5$. Best result in each column for each algorithm is written in bold. "#Multi Borders" denotes number of points that were assigned to multiple clusters.

ε	Purity		RAND		Silhouette		Davis-Bouldin		#Multi Borders	
	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
0.0001	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.0010	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.0100	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.0500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.1000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.1500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.2000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.2500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.3000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.3500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.4000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.4500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.5000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.5500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.6500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.7000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.7500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.8000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.8500	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9000	**0.2758**	0.2759	**0.1634**	0.1635	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9500	**0.2758**	0.2765	**0.1634**	0.1645	**0.0000**	0.2095	**0.0000**	0.2815	**0**	0
0.9600	**0.2758**	0.2767	**0.1634**	0.1649	**0.0000**	0.2099	**0.0000**	0.2814	**0**	0
0.9700	**0.2758**	0.2771	**0.1634**	0.1656	**0.0000**	0.1613	**0.0000**	0.3089	**0**	2
0.9800	**0.2758**	0.2774	**0.1634**	0.1661	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9900	**0.2758**	0.2784	**0.1634**	0.1678	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9990	**0.2758**	0.4970	**0.1634**	0.5499	**0.0000**	-0.2863	**0.0000**	**0.5922**	**0**	28
0.9999	**0.2758**	**0.7219**	**0.1634**	**0.8135**	**0.0000**	**0.3047**	**0.0000**	0.5256	**0**	**116**
1.0000	**0.2758**	0.2758	**0.1634**	0.1634	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0

Table 3: Evaluation results for different clustering quality measures depending on the value of ε when either Euclidean (Euc) distance measure or Tanimoto (Tan) measure is used for the `Complex` dataset. It is assumed that `minPts` $= 5$. Best result in each column for each algorithm is written in bold. "#Multi Borders" denotes number of points that were assigned to multiple clusters.

ε	Purity		RAND		Silhouette		Davis-Bouldin		#Multi Borders	
	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
0.0001	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.0010	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.0100	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.0500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.1000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.1500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.2000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.2500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.3000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.3500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.4000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.4500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.5000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.5500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.6500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.7000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.7500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.8000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.8500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9500	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9600	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9700	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0
0.9800	**0.2989**	0.4121	**0.1859**	0.5307	**0.0000**	0.2937	**0.0000**	0.4911	**0**	0
0.9900	**0.2989**	0.4131	**0.1859**	0.5307	**0.0000**	0.2934	**0.0000**	0.4916	**0**	0
0.9990	**0.2989**	**0.6605**	**0.1859**	**0.7649**	**0.0000**	0.0577	**0.0000**	**0.5991**	**0**	12
0.9999	**0.2989**	0.5239	**0.1859**	0.4924	**0.0000**	**0.8703**	**0.0000**	0.4059	**0**	**34**
1.0000	**0.2989**	0.2989	**0.1859**	0.1859	**0.0000**	0.0000	**0.0000**	0.0000	**0**	0

Table 4: Evaluation results for different clustering quality measures depending on the value of ε when either Euclidean (Euc) distance measure or Tanimoto (Tan) measure is used for the `Letter` dataset. It is assumed that `minPts` $= 5$. Best result in each column for each algorithm is written in bold. "#Multi Borders" denotes number of points that were assigned to multiple clusters.

ε	Purity		RAND		Silhouette		Davis-Bouldin		#Multi Borders	
	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
0.0001	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.0010	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.0100	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.0500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.1000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.1500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.2000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.2500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.3000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.3500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.4000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.4500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.5000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.5500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.6500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.7000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.7500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.8000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.8500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.9000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000	0	0
0.9500	0.0575	0.0426	0.0699	0.0421	**1.0000**	0.3199	0.0000	0.3125	0	0
0.9600	0.0575	0.0529	0.0699	0.1156	**1.0000**	0.1808	0.0000	0.6882	0	0
0.9700	0.0575	0.0732	0.0699	0.1527	**1.0000**	-0.0595	0.0000	0.6411	0	0
0.9800	0.0575	0.1479	0.0699	0.2875	**1.0000**	-0.1643	0.0000	0.8461	0	26
0.9900	0.0575	**0.4637**	0.0699	**0.8787**	**1.0000**	0.2262	0.0000	**0.9390**	0	**156**
0.9990	0.0575	0.0575	0.0699	0.0699	**1.0000**	**1.0000**	0.0000	0.0000	0	0
0.9999	0.0575	0.0575	0.0699	0.0699	**1.0000**	**1.0000**	0.0000	0.0000	0	0
1.0000	**0.1209**	0.0575	**0.1847**	0.0699	0.9595	**1.0000**	**0.5308**	0.0000	**26**	0

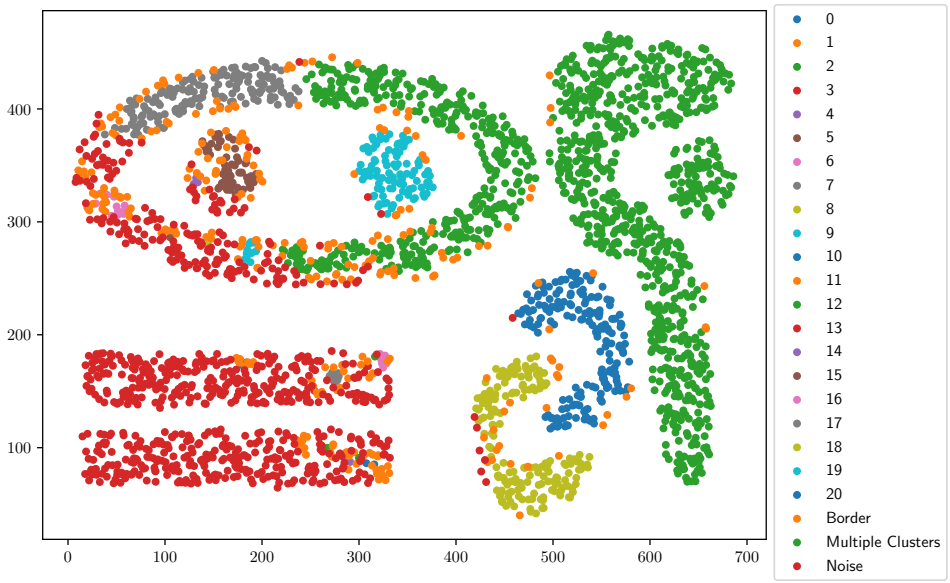


Figure 5: Visualization of the DBSCAN+ algorithm result with parameters $\varepsilon = 0.999$ and `minPts` $= 10$, when Tanimoto measure is used. Points are colored according the classes assigned to them, described in the displayed legend. The "Multiple Clusters" label means that a border point was assigned to more than one cluster. Number of points with multiple assignments: 6.

7.3 Influence of the `minPts` parameter

In this experiment I followed the same procedure as previously - I varied the `minPts` value and analyzed how it affects the clustering. For the Tanimoto measure, I set $\varepsilon = 0.99$ and $\varepsilon = 0.1$ for the Euclidean distance. Obtained results can be found in Table 5. Best results were achievable for moderate values of `minPts`. `minPts` $= 0$ and `minPts` $= 1$ were used as edge-case scenarios to also show drawbacks of using existing metrics. In principle, these values should not be used in real world applications since they lead to each point being assigned to a separate cluster.

Table 5: Evaluation results for different clustering quality measures depending on the value of `minPts` when either Euclidean (Euc) distance measure or Tanimoto (Tan) measure is used for `Letter`, `Complex` and `Cluto` datasets. It is assumed that $\varepsilon = 0.1$ for the Euclidean metric and $\varepsilon = 0.99$ for the Tanimoto measure. Best result in each column for each algorithm is written in bold. "#Multi Borders" denotes number of points that were assigned to multiple clusters.

(a) `Cluto`

`minPts`	Purity		RAND		Silhouette		Davis-Bouldin		#Multi Borders	
	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
0	**1.0000**	0.2776	**0.8366**	0.1664	**0.9999**	0.0909	**0.0005**	0.3333	**0**	0
1	**1.0000**	0.2776	**0.8366**	0.1664	**0.9999**	0.0909	**0.0005**	0.3333	**0**	0
5	0.2758	0.2784	0.1634	0.1678	0.0000	0.0000	0.0000	0.0000	**0**	0
10	0.2758	0.2792	0.1634	0.1691	0.0000	0.0000	0.0000	0.0000	**0**	0
25	0.2758	0.2881	0.1634	0.1918	0.0000	0.1740	0.0000	0.3083	**0**	0
50	0.2758	0.3040	0.1634	0.2266	0.0000	0.0000	0.0000	0.0000	**0**	0
100	0.2758	**0.3945**	0.1634	**0.5131**	0.0000	**0.2566**	0.0000	**0.5696**	**0**	**26**

(b) `Complex`

`minPts`	Purity		RAND		Silhouette		Davis-Bouldin		#Multi Borders	
	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
0	**1.0000**	0.4127	**0.8141**	0.5307	**0.9999**	0.1756	**0.0001**	0.4812	**0**	0
1	**1.0000**	0.4127	**0.8141**	0.5307	**0.9999**	0.1756	**0.0001**	0.4812	**0**	0
5	0.2989	0.4131	0.1859	0.5307	0.0000	0.2934	0.0000	0.4916	**0**	0
10	0.2989	0.4266	0.1859	0.5311	0.0000	0.1679	0.0000	0.5381	**0**	4
25	0.2989	0.4490	0.1859	0.5332	0.0000	0.1975	0.0000	0.5250	**0**	4
50	0.2989	**0.4629**	0.1859	0.5339	0.0000	0.1565	0.0000	**0.7222**	**0**	0
100	0.2989	0.4507	0.1859	**0.6238**	0.0000	**0.6028**	0.0000	0.4380	**0**	**58**

(c) `Letter`

`minPts`	Purity		RAND		Silhouette		Davis-Bouldin		#Multi Borders	
	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
0	**1.0000**	**0.6389**	**0.9616**	**0.9113**	**1.0000**	-0.1171	**0.0000**	0.5469	**0**	0
1	**1.0000**	**0.6389**	**0.9616**	**0.9113**	**1.0000**	-0.1171	**0.0000**	0.5469	**0**	0
5	0.0575	0.4637	0.0699	0.8787	**1.0000**	0.2262	0.0000	**0.9390**	**0**	**156**
10	0.0443	0.4153	0.0452	0.7659	**1.0000**	0.4951	0.0000	0.9308	**0**	139
25	0.0420	0.2397	0.0409	0.4007	0.0000	0.8619	0.0000	0.7850	**0**	52
50	0.0406	0.0748	0.0384	0.1079	0.0000	**0.9859**	0.0000	0.4630	**0**	0
100	0.0406	0.0406	0.0384	0.0384	0.0000	0.0000	0.0000	0.0000	**0**	0

7.4 Influence of applying prefiltering heuristics

Finally, I tested how the application of RealVecLen and ZPNVecLen heuristics affects computation time on each of datasets when ε is varied. These heuristics are applicable only when the Tanimoto measure is used. Table 6 represents how many times a point is compared to other points when a neighborhood of that point is determined. In this case, when no prefiltering is used then, the number of point comparisons should be the same and be equal to the number of points in a dataset.

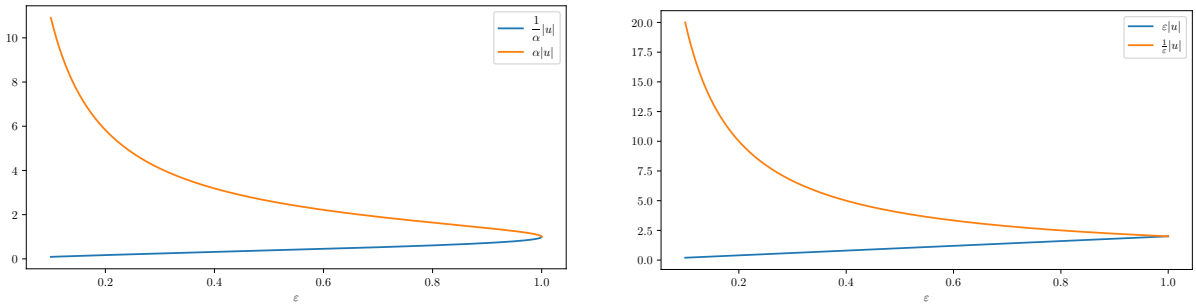
When RealVecLen is used, it can be noticed the number of comparisons converge to 0 each time. It makes sense from the perspective of the heuristic - when $\varepsilon \to 1.0$ then bounds around a particular point converge to the length of that particular point which is visualized in Figure 6. When $\varepsilon = 1.0$ then any other point should be located exactly in the same place as the evaluated point to be considered as its neighborhood. For ZPNVecLen this effect does apply as well. However, since point locations are discretized, they can occupy exactly the same location in the ambient space $\mathbb{R}^d$ where d is number of dimensions of data points. Therefore, number of point

comparison can never decrease to 0.

Lastly, the decreasing number of point comparison correlates with decreased time of the DBSCAN fitting to the data. It means that the lower number of point comparisons, the less time the algorithm spends on finding neighbors of each points hence the program runs faster in general. Table 7 shows how the time spent on the clustering decreases when ε is increased which led to decreased number of comparisons in Table 6. While for the case where no heuristic is applied the time should be constant. However, it varies due to fluctuations in availability of computational resources when experiments were running.

Table 6: Number of average point comparisons for different filtering methods: *None* (no prefiltering), *Real* (RealVecLen), *ZPN* (ZPNVecLen), depending on the value of ε for all datasets. It is assumed that `minPts` = 5. Best result in each column for each algorithm is written in bold (except for *None* where the number of evaluations is the same).

ε	`Cluto`			`Complex9`			`Letter`		
	None	Real	ZPN	None	Real	ZPN	None	Real	ZPN
0.0001	10000	9999	7027	3031	3030	1522	20000	19999	19999
0.0010	10000	9999	7027	3031	3030	1522	20000	19999	19999
0.0100	10000	9999	7027	3031	3030	1522	20000	19999	19999
0.0500	10000	9998	7027	3031	3030	1522	20000	19999	19999
0.1000	10000	9992	7027	3031	3030	1522	20000	19999	19998
0.1500	10000	9962	7027	3031	3020	1522	20000	19999	19996
0.2000	10000	9866	7027	3031	2992	1522	20000	19999	19993
0.2500	10000	9731	7027	3031	2950	1522	20000	19999	19986
0.3000	10000	9543	7027	3031	2893	1522	20000	19999	19970
0.3500	10000	9309	7027	3031	2818	1522	20000	19999	19923
0.4000	10000	9058	7027	3031	2738	1522	20000	19999	19786
0.4500	10000	8793	7027	3031	2659	1522	20000	19999	19591
0.5000	10000	8514	**3895**	3031	2574	**1398**	20000	19999	18990
0.5500	10000	8200	**3895**	3031	2480	**1398**	20000	19998	18256
0.6500	10000	7421	**3895**	3031	2246	**1398**	20000	19990	15963
0.7000	10000	6968	**3895**	3031	2108	**1398**	20000	19964	14582
0.7500	10000	6471	**3895**	3031	1960	**1398**	20000	19900	12429
0.8000	10000	5897	**3895**	3031	1790	**1398**	20000	19773	10560
0.8500	10000	5209	**3895**	3031	1586	**1398**	20000	19527	7541
0.9000	10000	4328	**3895**	3031	1320	**1398**	20000	18904	5547
0.9500	10000	3153	**3895**	3031	965	**1398**	20000	16703	**2590**
0.9600	10000	2846	**3895**	3031	873	**1398**	20000	15745	**2590**
0.9700	10000	2495	**3895**	3031	767	**1398**	20000	14414	**2590**
0.9800	10000	2060	**3895**	3031	635	**1398**	20000	12485	**2590**
0.9900	10000	1474	**3895**	3031	455	**1398**	20000	9402	**2590**
0.9990	10000	494	**3895**	3031	155	**1398**	20000	3159	**2590**
0.9999	10000	158	**3895**	3031	50	**1398**	20000	1006	**2590**
1.0000	10000	**0**	**3895**	3031	**0**	**1398**	20000	**36**	**2590**



(a) RealVecLen (b) ZPNVecLen

Figure 6: Behavior of lower and upper bounds for each of heuristics for the Tanimoto measure. Note they diverge when $\varepsilon \to 0.0$ and converge to a single point when $\varepsilon \to 1.0$. Bounds follow the notation presented by Kryszkiewicz [2].

Table 7: Average time in milliseconds spent on model fitting for different filtering methods: *None* (no prefiltering), *Real* (RealVecLen), *ZPN* (ZPNVecLen), depending on the value of ε for all datasets. It is assumed that `minPts` $= 5$. Best result in each column for each algorithm is written in bold (except for *None* where the average time spent should be the same and any difference is caused by fluctuations of available compute resources at the time of performing experiments). The last column shows Pearson's correlation between the execution time and the number of point comparisons. "–" denotes the undefined correlation due to the constant number of point comparisons.

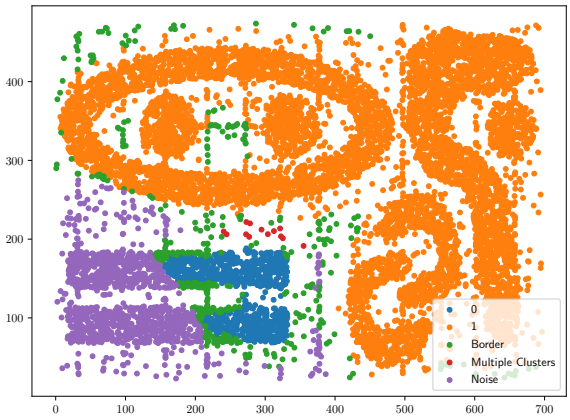
ε	`Cluto`			`Complex9`			`Letter`		
	None	Real	ZPN	None	Real	ZPN	None	Real	ZPN
0.0001	10837	10964	6989	858	881	426	73128	76309	72484
0.0010	11570	11241	7043	845	893	416	74109	75159	71809
0.0100	10425	10518	6816	845	873	431	69350	71479	68305
0.0500	10380	10702	6772	823	852	407	69659	70431	68038
0.1000	10445	10547	6704	821	847	402	69524	70866	67528
0.1500	10490	10607	6704	843	860	409	69756	70758	67433
0.2000	10643	10777	6876	851	884	416	81118	85692	74569
0.2500	10684	10674	6884	851	851	419	83525	82664	75227
0.3000	10703	10140	6596	820	797	403	76446	79099	71212
0.3500	9928	9638	6430	791	754	387	66257	68784	63435
0.4000	9698	9526	6430	791	730	406	67646	68128	62732
0.4500	10175	9257	6381	788	721	390	66227	67992	61546
0.5000	9739	8934	**3613**	799	694	**357**	66237	67904	58324
0.5500	9516	8450	3694	791	677	358	68036	69436	55873
0.6500	9946	7855	3766	809	652	369	67808	68850	51067
0.7000	10270	7574	3872	890	612	379	74981	75526	49276
0.7500	9998	6841	3875	839	567	388	71160	71820	40565
0.8000	10135	6264	3830	824	539	386	71718	71996	34684
0.8500	9849	5564	3821	834	462	380	72001	68926	24668
0.9000	9709	4503	3890	805	395	387	67631	65055	18108
0.9500	9624	3231	3851	798	282	380	67045	57284	9139
0.9600	9432	2939	3855	788	254	382	67099	54883	8967
0.9700	9565	2558	3899	803	227	381	68371	49133	**8743**
0.9800	9393	2091	3946	778	188	380	72533	42938	8962
0.9900	9427	1528	3845	772	137	380	67173	32874	8860
0.9990	9201	579	3844	787	56	395	66948	10857	8968
0.9999	9172	272	3868	772	27	383	66902	3759	8749
1.0000	9072	**120**	3711	763	**13**	367	66183	**816**	8809
ρ	–	0.9988	0.9944	–	0.9973	0.8067	–	0.9816	0.9921

7.5 Final experiment

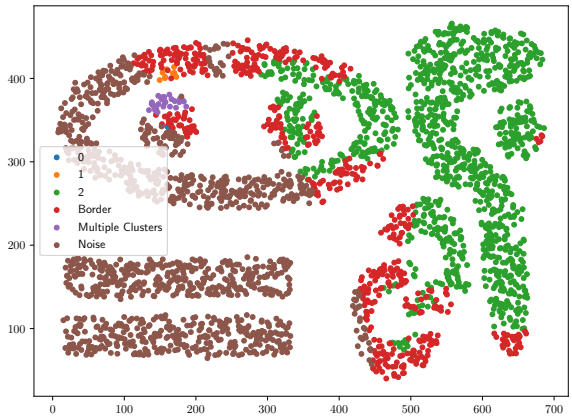
In the final experiment, I present results with parameters set to values found empirically as the best ones according to different clustering quality measures. I consider DBSCAN+ algorithm with the Tanimoto measure RealVecLen heuristic. Results are presented in Table 8. I also visualized clustering results for `Cluto` and `Complex` datasets[6] in Figure 7. One can notice, that the algorithm finds sensible clusters.

Table 8: Results obtained with DBSCAN+ algorithm with the Tanimoto measure and RealVecLen heuristic. "#Multi Borders" denotes number of points that were assigned to multiple clusters. "Time" is a number of milliseconds spent on clustering and "Calculations" is an average number of point comparisons to other points to find neighbors.

Dataset	minPts	ε	Purity	RAND	Silhouette	Davis-Bouldin	#Multi Borders	Time	Calculations
`Cluto`	100	0.99	0.3944	0.5131	0.2565	0.5698	26.0	1536.0	1474.530
`Complex`	100	0.99	0.4513	0.6231	0.6126	0.4183	56.0	137.0	455.898
`Letter`	25	0.99	0.2288	0.3830	0.8721	0.7715	30.0	31989.0	9402.200



(a) `Cluto`. Number of points with multiple assignments: 26.



(b) `Complex`. Number of points with multiple assignments: 56.

Figure 7: Clustering results using DBSCAN+ algorithm with the Tanimoto measure and RealVecLen heuristic for `Cluto` and `Complex` datasets. Points are colored according the classes assigned to them, described in the displayed legend. Parameters of the algorithm for each dataset are presented in Table 8.

7.6 Comparison to Python implementation

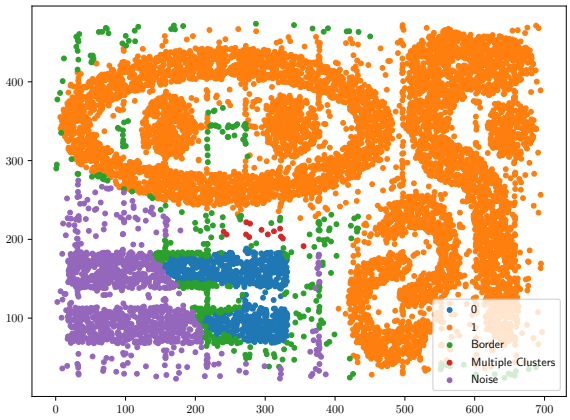
Since the goal of the project was to implement the DBSCAN+ algorithm with RealVecLen and ZPNVecLen heuristics in two languages, I compared my C++ implementation with its Python version in terms of RAND metric and inference time. For unbiased results and true comparison of the algorithm itself, I compared both methods without any heuristics. Results are presented in Table 9. Minute differences between statistics outputted from both algorithms comes from interpretation details, namely: • I treated noise points as a separate class • numbers are represented as `double` where errors of rounding can accumulate over multiple calculations. Additional results of clustering are presented in Figure 8. One can notice additional "red" dots for C++ implementation as border points were marked explicitly in that case. Overall, the clustering seems to be the same for both implementations but differences in the execution time

[6] `Letter` datatset consists of 16 features, hence it was omitted.

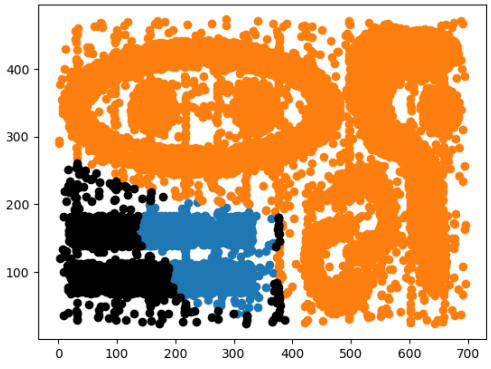
is significant. It comes from the fact that Python is an interpreted language instead of being compiled. Additionally, every element is an object that needs instantiation and freeing. These and many other structural difference influence the time of execution.

Table 9: Results obtained with DBSCAN+ algorithm with the Tanimoto measure with comparison between C++ and Python versions of the implementation.

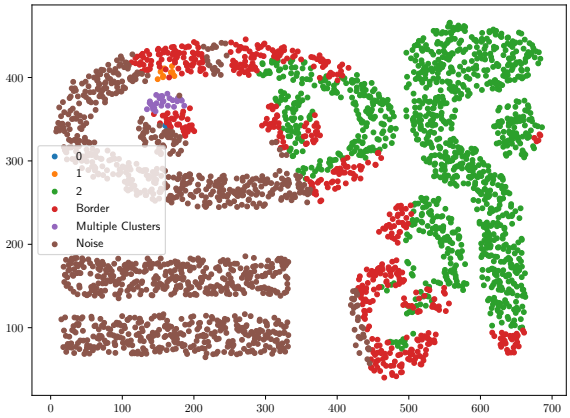
Dataset	`minPts`	ε	RAND		Total Runtime $[s]$	
			C++	Python	C++	Python
`Cluto`	100	0.99	0.513	0.513	9.50	880.86
`Complex`	100	0.99	0.624	0.624	0.81	13.77
`Letter`	25	0.99	0.401	0.401	67.51	636.49



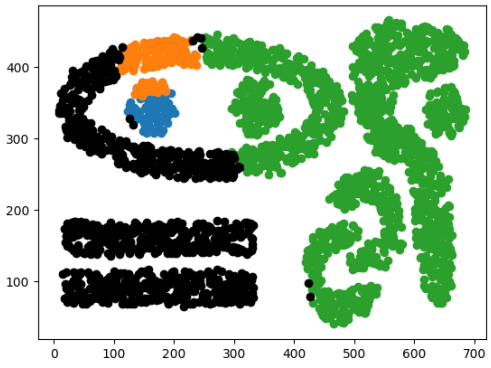
(a) `Cluto` - C++



(b) `Cluto` - Python



(c) `Complex` - C++



(d) `Complex` - Python

Figure 8: Clustering results using DBSCAN+ algorithm with the Tanimoto measure for `Cluto` and `Complex` datasets for C++ and Python implements. Points are colored according the classes assigned to them. Parameters of the algorithm for each dataset are presented in Table 9.

8 Conclusions

I presented results of the project encompassing the implementation of the DBSCAN+ algorithm with respect to the Tanimoto measure with the application of RealVecLen and ZPNVecLen heuristics. Firstly, I described the modification of the original DBSCAN algorithm which allows border points to be assigned to multiple clusters. I introduced tasks made during the project,

made assumptions about the project, data, algorithm and the evaluation. I also showed the expected the data format that the compiled binary of the program expects. I included an UML class diagram used for the implementation of the algorithm. To reproduce obtained results, I described thoroughly the procedure of the installation and usage of the program. Finally, I presented results from the experiments that included initial attempts of the clustering on an artificial datasets, empirical analysis of influence of the ε and `minPts` parameters. Then, I showed that application of prefiltering heuristics significantly reduces runtime of the algorithm. The algorithm with the best values of parameters was used in the final evaluation and compared with its version implemented in Python.

Applying the Tanimoto measure has two consequences. Firstly, it allows the user to set ε to value in the closed range $[0, 1]$. This feature takes of the burden from the user to rely on careful processing and interpretation of the data just to set an appropriate value of the parameter. Grid search of the best hyperparameter value becomes also easier since we can operate on a closed domain of values. In terms of which measure gives the best result according to quality measures, further research would be necessary to determine ε value for the Euclidean measure which can be costly.

Furthermore, it is possible to apply additional heuristics such as RealVecLen and ZPNVecLen that can speed up processing times significantly which becomes even more noticeable for datasets where computing DBSCAN is prohibitive for real world applications.

References

[1] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

[2] Marzena Kryszkiewicz. “Bounds on Lengths of Real Valued Vectors Similar with Regard to the Tanimoto Similarity”. In: *Intelligent Information and Database Systems - 5th Asian Conference, ACIIDS 2013, Kuala Lumpur, Malaysia, March 18-20, 2013, Proceedings, Part I*. Ed. by Ali Selamat et al. Vol. 7802. Lecture Notes in Computer Science. Springer, 2013, pp. 445–454.