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 $\angle \in [0, \pi]$.

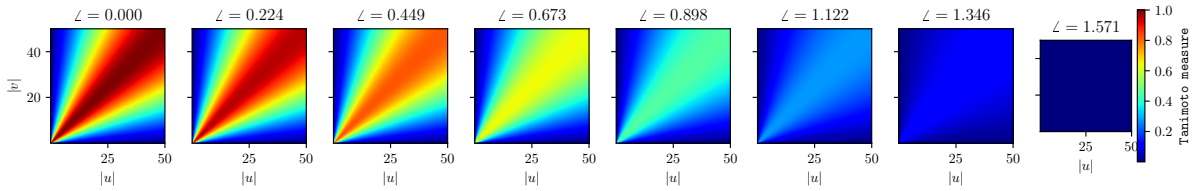


Figure 1: Visualization of the Tanimoto measure values, depending on lengths of two vectors and the angle between them.

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.

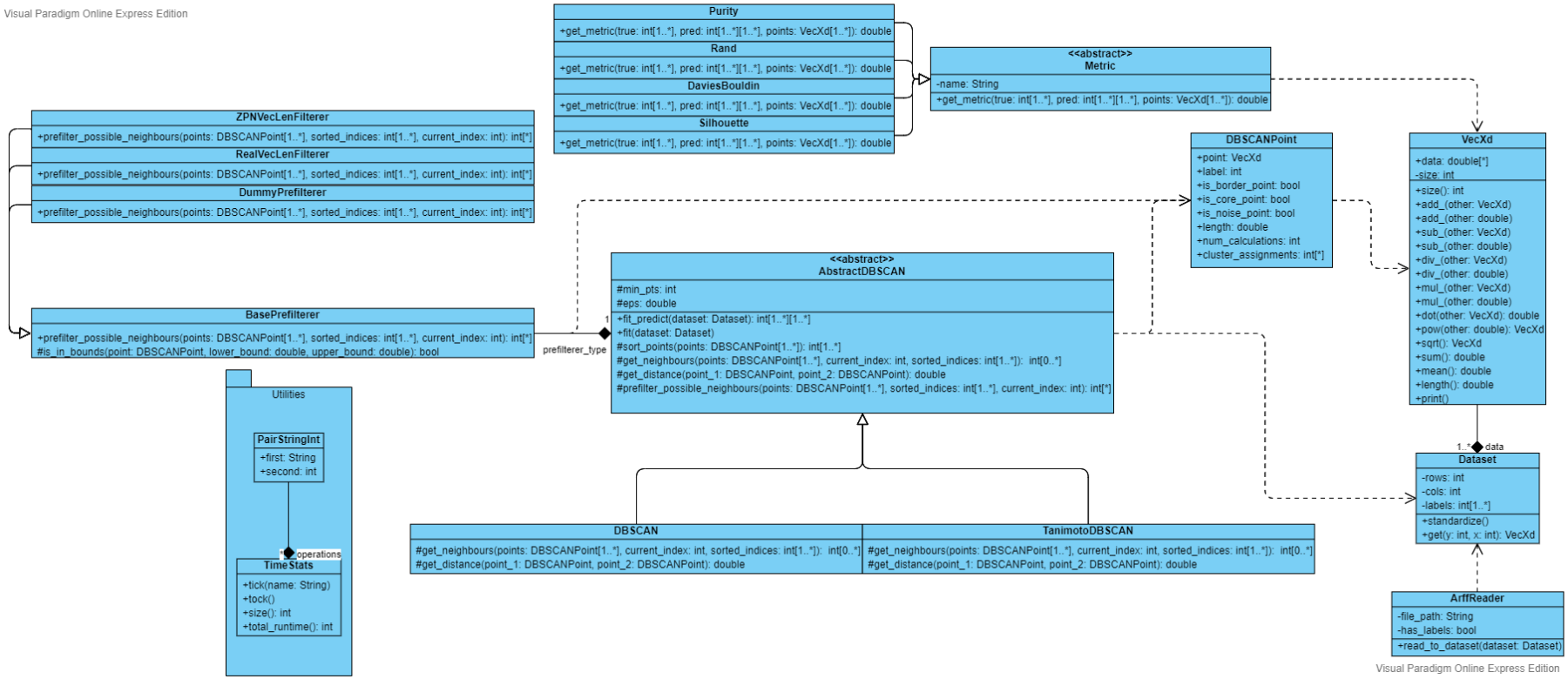


Figure 2: The class diagram of the main part of the project. Auxiliary utilities are omitted for clarity.

5.2 Sorting of points

Instead of sorting points in the dataset by their lengths directly, I hold a separate array of identifiers of these points. This way, no copying of points is performed and querying points during calculations is efficient.

5.3 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.

5.4 Calculation of metrics

As a nature of border points assignment changes in DBSCAN+, a true positive is counted if both true labels for two distinct points are the same and an intersection cluster assignments of these points is not empty. When two labels are different, a true negative is counted if the intersection is empty. The rule applies when RAND metric is calculated.

5.5 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.

5.6 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

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]:

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

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

Algorithm 1: DBSCAN+ algorithm

Data: ε, `min_pts`, $\mathbf{X}$
`cls` $\leftarrow -1$;
`sort_lengths`($\mathbf{X}$);
for `x` $\in \mathbf{X}$ **do**
  if `x.label` $\neq$ `UNDEFINED` **then**
    continue;
  `neigbhors` $\leftarrow$ `get_neighbors`(`x`, $\mathbf{X}$, ε);
  if |`neigbhors`| $<$ `min_pts` **then**
    `x.label` $\leftarrow$ `NOISE`;
    `push`(`x.cluster_assignments`, `NOISE`);
    continue;
  `cls` $\leftarrow$ `cls` $+ 1$;
  `x.label` $\leftarrow$ `cls`;
  `x.core` $\leftarrow$ *true*;
  `push`(`x.cluster_assignments`, `cls`);
  while |`neigbhors`| > 0 **do**
    `n` $\leftarrow$ `pop`(`neighbors`);
    if `n.label` $=$ `NOISE` **then**
      `n.label` $\leftarrow$ `cls`;
      `n.core` $\leftarrow$ *false*;
      `n.border` $\leftarrow$ *true*;
      `n.noise` $\leftarrow$ *false*;
      `erase`(`n.cluster_assignments`, `NOISE`);
      `push`(`n.cluster_assignments`, `cls`);
    if `n.label` $\neq$ `UNDEFINED` **then**
      if ***not*** `n.core` **then**
        `push`(`n.cluster_assignments`, `cls`);
      continue;
    `n.label` $\leftarrow$ `cls`;
    `neigbhors2` $\leftarrow$ `get_neighbors`(`n`, $\mathbf{X}$, ε);
    if |`neigbhors`| $\geq$ `min_pts` **then**
      `x.core` $\leftarrow$ *true*;
      `x.border` $\leftarrow$ *false*;
      `x.noise` $\leftarrow$ *false*;
      `clear`(`x.cluster_assignments`);
      `push`(`x.cluster_assignments`, `cls`);
      `neighbors` $\leftarrow$ `neighbors` $\cup$ `neighbors2`;
    else
      `x.core` $\leftarrow$ *false*;
      `x.border` $\leftarrow$ *true*;
      `push`(`x.cluster_assignments`, `cls`);

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;

```
$ 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.

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> \
      [--has_labels]
  ```

  The scripts converts the original features values to have one values from $\{-1, 0, 1\}$, depending what bin is assigned to a particular value. The bins have the following boundaries: $(-\infty, \mu - 0.5\sigma]$, $(\mu - 0.5\sigma, \mu + 0.5\sigma]$ and $(\mu + 0.5\sigma, \infty)$ where μ and σ are mean and standard deviation calculated for each feature independently. 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 an artificial dataset consisting of 850 points. 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 structure probability distribution. 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 one of values from $\{-1, 0, 1\}$. To obtain zpn-valued vectors, I converted each feature depending on what bin they fall in. The bins have the following boundaries: $(-\infty, \mu - 0.5\sigma]$, $(\mu - 0.5\sigma, \mu + 0.5\sigma]$ where μ and σ are mean and standard deviation calculated for each feature independently.

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

7.1 Empirical validation of the implementation

To validate the implementation, I prepared two two-dimensional datasets. One of them, consists of 21 points that are placed as follows. Ten of these points share the same feature value and they are linearly spaced in the range $[-1, 1]$ along the second feature. The other ten are mirrored along the axis of the second feature. I added one additional point at (0.0, 1.0) that is supposed to be either a noise point or a point classified to two clusters (depending on parameters). The second dataset consists of 850 data points. 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. These vectors were obtained according to the rule described above.

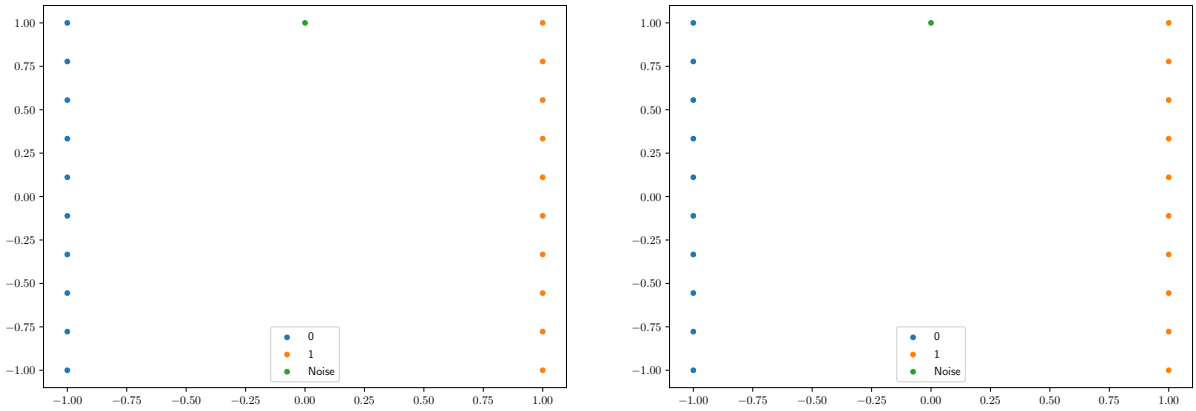
To consider also more structured dataset, I used `Complex9` dataset consisting of 10 000 data points. The clustering result of the dataset is presented in Figure 5.

7.2 Influence of the ε parameter

Tables 1,2,3 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.

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



(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.

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.

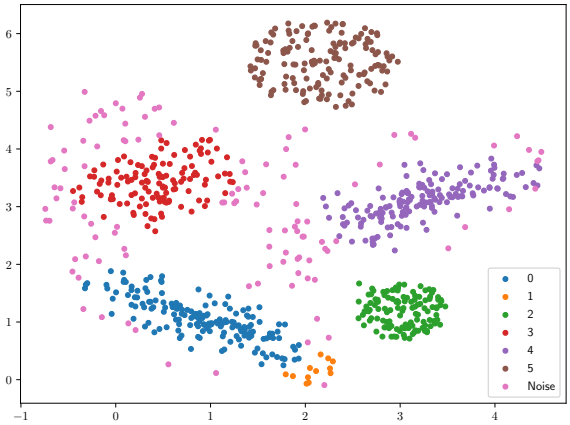
7.3 Influence of the `minPts` parameter

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 4. 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.

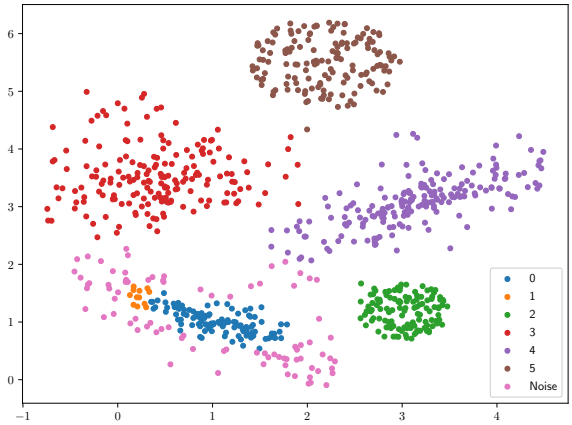
7.4 Influence of applying prefiltering heuristics

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. As a sanity check for debugging purposes, I show in Figure 6 that RealVecLen does not change clustering results when it is used. Tables 5, 6 represents how many times a point is compared to other points when a neighborhood of that point is determined for real-valued and zpn-value datasets respectively. Note that ZPNVecLen cannot be applied for real-valued vectors, thus it was omitted for the first table. In the 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. Results in Table 6 show that leveraging properties of ZPN vectors can speed up calculations in comparison to RealVecLen.

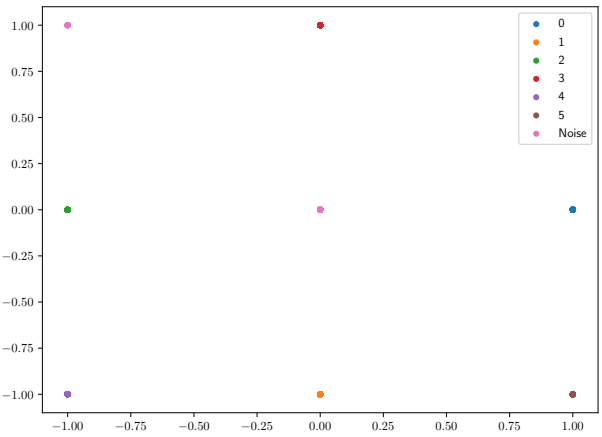
When RealVecLen is used, it can be noticed the number of comparisons converge to 0 each time. It makes sense from the perspective of the heuristic - when $\varepsilon \to 1.0$ then bounds around a particular point converge to the length of that particular point which is visualized in Figure 7. When $\varepsilon = 1.0$ 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.



(a) Euclidean, $\varepsilon = 0.3$, `minPts` $= 15$. Number of points with multiple cluster assignments:



(b) Tanimoto, $\varepsilon = 0.98$, `minPts` $= 15$. Number of points with multiple cluster assignments:



(c) Tanimoto with ZPN vectors, $\varepsilon = 0.98$, `minPts` $= 15$. Number of points with multiple cluster assignments:

Figure 4: Clustering results on the dataset made of 850 points 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 1: 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 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 `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 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 `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

Table 4: 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	**0.0000**	0
1	**1.0000**	0.2776	**0.8366**	0.1664	**0.9999**	0.0909	0.0005	0.3333	**0.0000**	0
5	0.2758	0.2784	0.1634	0.1678	0.0000	0.0000	**0.0000**	**0.0000**	**0.0000**	0
10	0.2758	0.2792	0.1634	0.1691	0.0000	0.0000	**0.0000**	**0.0000**	**0.0000**	0
25	0.2758	0.2881	0.1634	0.1918	0.0000	0.1740	**0.0000**	0.3083	**0.0000**	0
50	0.2758	0.3040	0.1634	0.2266	0.0000	0.0000	**0.0000**	**0.0000**	**0.0000**	0
100	0.2758	**0.3945**	0.1634	**0.5131**	0.0000	**0.2566**	**0.0000**	0.5696	**0.0000**	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	**0.0000**	0
1	**1.0000**	0.4127	**0.8141**	0.5307	**0.9999**	0.1756	0.0001	0.4812	**0.0000**	0
5	0.2989	0.4131	0.1859	0.5307	0.0000	0.2934	**0.0000**	0.4916	**0.0000**	0
10	0.2989	0.4266	0.1859	0.5311	0.0000	0.1679	**0.0000**	0.5381	**0.0000**	4
25	0.2989	0.4490	0.1859	0.5332	0.0000	0.1975	**0.0000**	0.5250	**0.0000**	4
50	0.2989	**0.4629**	0.1859	0.5339	0.0000	0.1565	**0.0000**	0.7222	**0.0000**	0
100	0.2989	0.4507	0.1859	**0.6238**	0.0000	**0.6028**	**0.0000**	**0.4380**	**0.0000**	**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

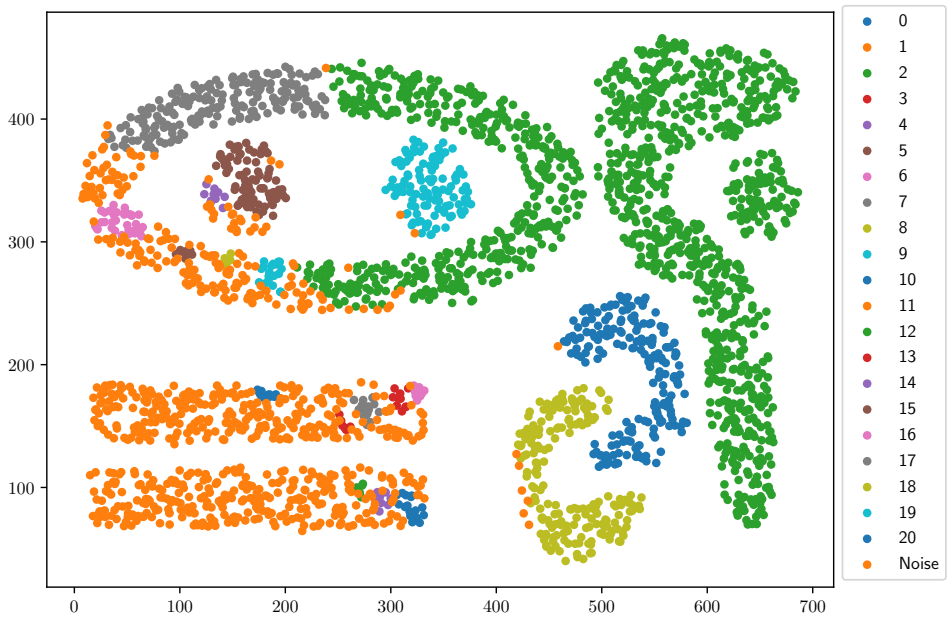


Figure 5: Visualization of the DBSCAN+ algorithm result with parameters $\varepsilon = 0.999$ and `minPts` $= 10$, when Tanimoto measure is used. Points are colored according the classes assigned to them, described in the displayed legend. The "Multiple Clusters" label means that a border point was assigned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

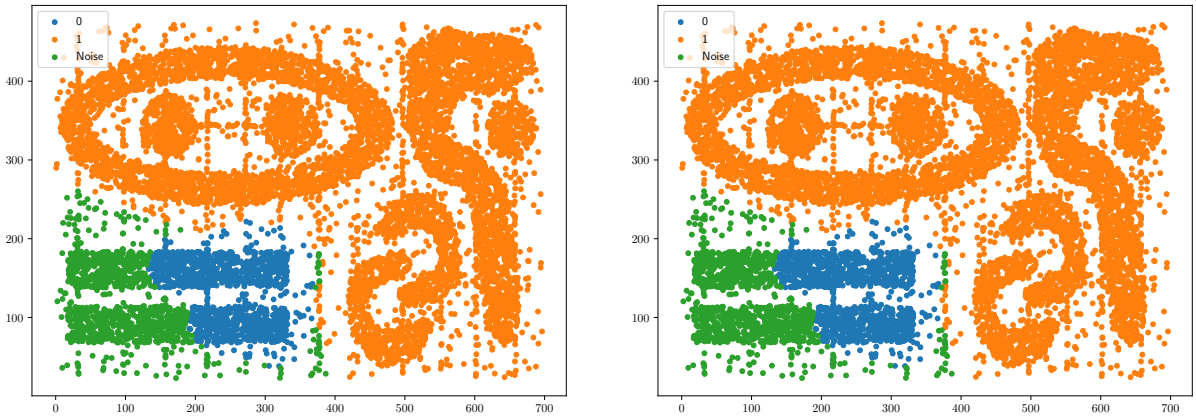
However, since point locations are discretized, they can occupy exactly the same location in the ambient space $\mathbb{R}^d$ where d is number of dimensions of data points. Therefore, number of point comparison can never decrease to 0.

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. Note that I consider only clustering operation (main loop of the DBSCAN algorithm). Any other operations such as loading datasets and sorting of points occupy negligible runtimes, ex. for the `Letter` dataset with 20000 data points both operations took 45 milliseconds and 1 millisecond respectively. Table 7 shows how the time spent on the clustering decreases when ε is increased which led to decreased number of comparisons in Table 5. The same rule applies for zpn-valued vectors in Table 8. 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.

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 9. I also visualized clustering results for `Cluto` and `Complex` datasets[6] in Figure 8. One can notice, that the algorithm finds sensible clusters.

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



(a) Without RealVecLen (b) With RealVecLen

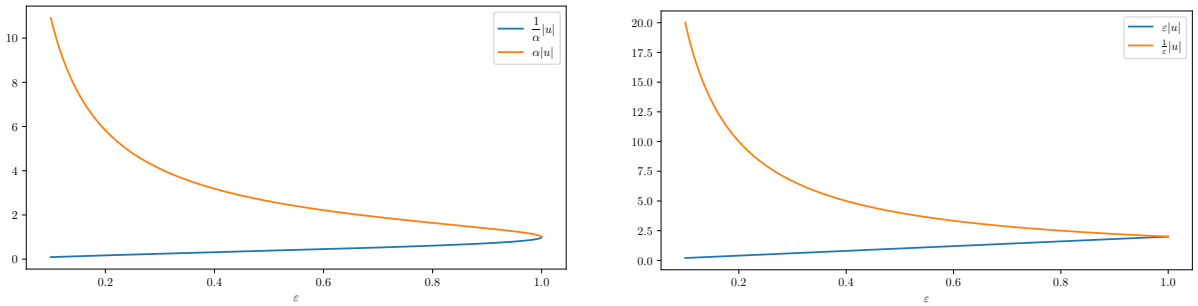
Figure 6: Visualization of clustering results with and without application of the RealVecLen heuristic.

Table 5: Number of average point comparisons for different filtering methods: *None* (no pre-filtering), *Real* (RealVecLen), depending on the value of ε for all real-valued 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	None	Real	None	Real
0.0001	10000	9999	3031	3030	20000	19999
0.0010	10000	9999	3031	3030	20000	19999
0.0100	10000	9999	3031	3030	20000	19999
0.0500	10000	9998	3031	3030	20000	19999
0.1000	10000	9992	3031	3030	20000	19999
0.1500	10000	9962	3031	3020	20000	19999
0.2000	10000	9866	3031	2992	20000	19999
0.2500	10000	9731	3031	2950	20000	19999
0.3000	10000	9543	3031	2893	20000	19999
0.3500	10000	9309	3031	2818	20000	19999
0.4000	10000	9058	3031	2738	20000	19999
0.4500	10000	8793	3031	2659	20000	19999
0.5000	10000	8514	3031	2574	20000	19999
0.5500	10000	8200	3031	2480	20000	19998
0.6500	10000	7421	3031	2246	20000	19990
0.7000	10000	6968	3031	2108	20000	19964
0.7500	10000	6471	3031	1960	20000	19900
0.8000	10000	5897	3031	1790	20000	19773
0.8500	10000	5209	3031	1586	20000	19527
0.9000	10000	4328	3031	1320	20000	18904
0.9500	10000	3153	3031	965	20000	16703
0.9600	10000	2846	3031	873	20000	15745
0.9700	10000	2495	3031	767	20000	14414
0.9800	10000	2060	3031	635	20000	12485
0.9900	10000	1474	3031	455	20000	9402
0.9990	10000	494	3031	155	20000	3159
0.9999	10000	158	3031	50	20000	1006
1.0000	10000	**0**	3031	**0**	20000	**36**

Table 6: Number of average point comparisons for different filtering methods: *None* (no pre-filtering), *Real* (RealVecLen), , *ZPN* (ZPNVecLen), depending on the value of ε for all zpn-valued 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	6914	6914	3031	2087	2087	20000	19957	19957
0.0010	10000	6914	6914	3031	2087	2087	20000	19957	19957
0.0100	10000	6914	6914	3031	2087	2087	20000	19957	19957
0.0500	10000	6914	6914	3031	2087	2087	20000	19957	19957
0.1000	10000	6914	6914	3031	2087	2087	20000	19957	19915
0.1500	10000	6914	6914	3031	2087	2087	20000	19957	19802
0.2000	10000	6914	6914	3031	2087	2087	20000	19957	19658
0.2500	10000	6914	6914	3031	2087	2087	20000	19957	19416
0.3000	10000	6914	6914	3031	2087	2087	20000	19957	18945
0.3500	10000	6914	6914	3031	2087	2087	20000	19949	18295
0.4000	10000	6914	6914	3031	2087	2087	20000	19925	17647
0.4500	10000	6914	6914	3031	2087	2087	20000	19878	16647
0.5000	10000	6914	**3685**	3031	2087	**1109**	20000	19802	15489
0.5500	10000	6914	**3685**	3031	2087	**1109**	20000	19716	14541
0.6500	10000	6914	**3685**	3031	2087	**1109**	20000	19416	11666
0.7000	10000	6914	**3685**	3031	2087	**1109**	20000	19095	10267
0.7500	10000	6914	**3685**	3031	2087	**1109**	20000	18752	8068
0.8000	10000	6914	**3685**	3031	2087	**1109**	20000	18087	6917
0.8500	10000	6914	**3685**	3031	2087	**1109**	20000	16989	5174
0.9000	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	15006	2841
0.9500	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	12083	**2075**
0.9600	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	11666	**2075**
0.9700	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	9770	**2075**
0.9800	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	7735	**2075**
0.9900	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	6180	**2075**
0.9990	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	**2075**	**2075**
0.9999	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	**2075**	**2075**
1.0000	10000	**3685**	**3685**	3031	**1109**	**1109**	20000	**2075**	**2075**



(a) RealVecLen (b) ZPNVecLen

Figure 7: 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), depending on the value of ε for all real-valued 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).

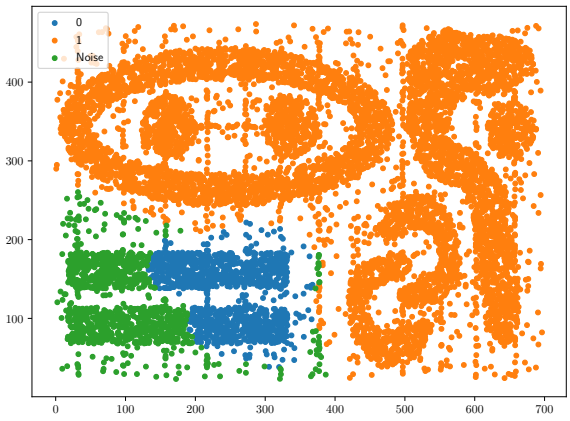
ε	`Cluto`		`Complex9`		`Letter`	
	None	Real	None	Real	None	Real
0.0001	10418	10525	847	865	69511	70783
0.0010	10556	10543	844	865	68883	70304
0.0100	10498	10555	834	865	69869	70538
0.0500	10684	10798	850	866	68832	70343
0.1000	10493	10674	846	867	68874	70681
0.1500	10513	10623	843	865	68895	71991
0.2000	10461	10640	849	872	69191	70648
0.2500	10415	10446	849	857	68895	70588
0.3000	10505	10214	878	829	69228	71687
0.3500	10391	10398	850	817	68978	70720
0.4000	10200	9741	846	791	69714	70001
0.4500	10093	9425	844	761	69360	70272
0.5000	10287	9720	856	742	69059	71300
0.5500	10021	8891	848	713	69327	70383
0.6500	9991	7812	842	654	70419	71301
0.7000	10130	7276	839	607	68676	71393
0.7500	9840	6625	833	563	68153	69450
0.8000	9667	6024	826	525	68169	69519
0.8500	9727	5245	807	457	66471	67199
0.9000	9511	4254	809	375	67941	62722
0.9500	9392	3055	789	276	64633	54680
0.9600	9643	2746	793	245	64564	52457
0.9700	9808	2417	831	214	67944	48129
0.9800	9333	1923	785	179	65434	41282
0.9900	9264	1343	793	129	64609	30595
0.9990	9057	454	773	44	64368	9415
0.9999	9034	143	779	15	64882	2660
1.0000	9002	**2**	777	**1**	64535	**101**

Table 8: 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 zpn-valued 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).

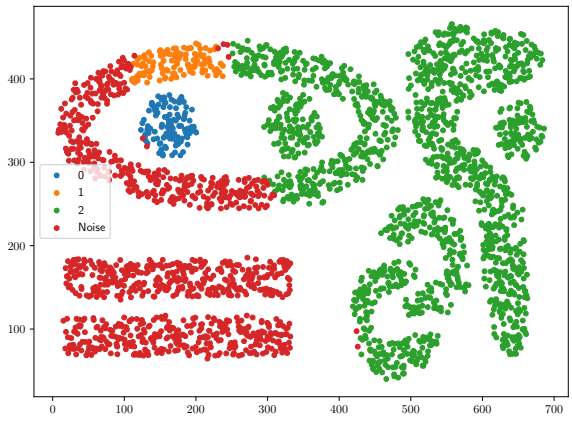
ε	`Cluto`			`Complex9`			`Letter`		
	None	Real	ZPN	None	Real	ZPN	None	Real	ZPN
0.0001	9239	6623	6600	798	564	568	66544	66839	66445
0.0010	9518	6629	7001	789	560	570	68117	67381	67468
0.0100	9143	6600	6618	785	570	570	66811	66463	66324
0.0500	9122	6621	6583	796	567	566	67034	66597	66219
0.1000	9204	6606	6918	794	573	569	65844	66187	65465
0.1500	9140	6634	6612	791	576	575	65562	65097	64434
0.2000	9443	6599	6609	790	566	569	65856	64737	63516
0.2500	9173	6611	6605	792	573	572	64892	64759	64253
0.3000	9208	6613	6626	804	562	650	65802	65137	61091
0.3500	9116	6600	6660	787	582	570	64315	63696	58824
0.4000	9288	6774	6729	794	573	564	64051	64163	57384
0.4500	9267	6673	6690	794	581	570	65259	64622	54197
0.5000	9102	6708	3447	797	574	305	65307	64196	49227
0.5500	8900	6338	3483	782	565	302	63957	63541	46205
0.6500	9019	6446	3492	796	575	299	64771	62978	37657
0.7000	9006	6619	3726	790	559	301	64588	62060	32928
0.7500	8896	6338	3492	785	558	305	63456	59821	25331
0.8000	8965	6346	3443	788	569	299	63716	57662	21656
0.8500	8894	6527	3452	780	558	309	63492	53904	15856
0.9000	8861	3477	3506	780	341	304	63347	47825	8301
0.9500	9173	**3436**	3443	780	301	298	63480	38228	**5805**
0.9600	9036	3504	3497	777	302	307	63787	37489	5841
0.9700	8894	3455	**3442**	780	**300**	301	63613	30655	5816
0.9800	8897	3443	3472	786	307	307	63590	24123	5827
0.9900	8870	3442	3444	885	304	**296**	63584	19097	5839
0.9990	9052	3461	3486	781	**300**	302	63466	**5839**	6201
0.9999	8887	3444	3449	787	308	300	64893	6056	5882
1.0000	8935	3491	3467	788	305	302	63488	5878	5840

Table 9: 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: 58.

Figure 8: 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 9.

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 10. Minute differences between statistics outputted from both algorithms comes from interpretation details, namely:

- Python implementation leverages the fact that there are $\frac{n(n-1)}{2}$ calculations of distance to be done across the dataset with n points since the distance calculation is symmetric. I do not use that fact and perform n^2 instead. That difference is especially significant for the Letter dataset in Table 10 where one would suspect total runtimes to be proportional across all datasets. However, the current C++ implementation does not allow to leverage that feature and would require significant memory burden to remember distances between points.
- In C++ version, when processing neighbors of a point p_c it is determined that its neighbor p_n is also a core point, then all previous cluster assignments of that point are removed and the label of the current cluster is assigned. That does not happen in the Python version where p_n can have multiple assignments because it was found as a neighbor of multiple

other points. Overall, it leads to the lower number of cluster assignments in C++ version on average.

- When processing a neighbor of a particular points and count of its neighbors is higher or equal to `minPts`, then it should be set as a core point as well. It is done in that way in the C++ version, however that fact was not included in the Python version. The difference is visible for the `Letter` dataset where 2 additional points were assigned as core points instead of as border points.
- numbers are represented as `double` where errors of rounding can accumulate over multiple calculations.

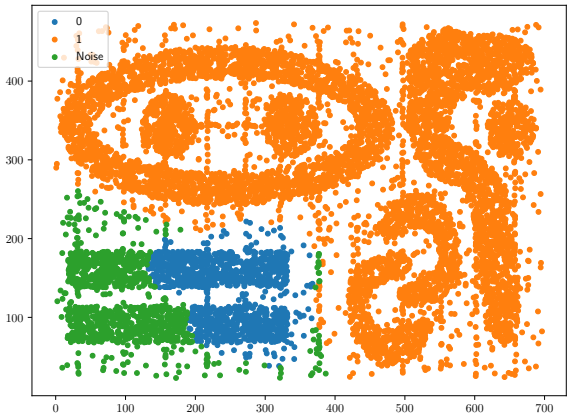
Additional results of clustering are presented in Figure 9. Overall, the clustering are the same for both implementations but differences in the execution time is significant. It comes from the fact that Python is an interpreted language instead of being compiled. Additionally, every element is an object that needs instantiation and freeing. These and many other structural difference influence the time of execution.

Table 10: Results obtained with DBSCAN+ algorithm with the Tanimoto measure with comparison between C++ and Python versions of the implementation for different datasets. Note, that no prefiltering heuristic was used.

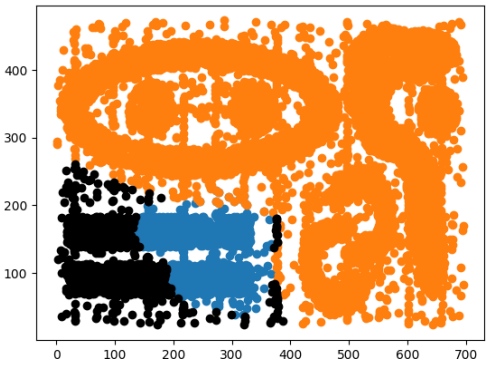
Dataset	`Cluto`		`Complex`		`Letter`	
	Python	C++	Python	C++	Python	C++
ε	0.99		0.99		0.99	
`minPts`	100		100		25	
Total runtime $[s]$	880.86	9.45	13.77	0.80	636.49	66.79
Number of discovered clusters	2	2	3	3	68	68
Number of core points	8359	8359	1245	1245	1862	1864
Number of border points	519	519	728	727	2513	2511
Number of noise points	1122	1122	1059	1059	15625	15625
Average number of distance calculation for a point	4995.00	10000	1515.00	3031.00	9999.50	20000.00
RAND TP	7529422	7529422	601659	601659	5060527	5060527
RAND TN	18123146	18123146	2262835	2262835	75078564	75078564
RAND value	0.513	0.513	0.624	0.624	0.401	0.401
Average number of clusters for a border point	1.03	1.0013	1.04	1.0096	1.01	1.0013

8 Conclusions

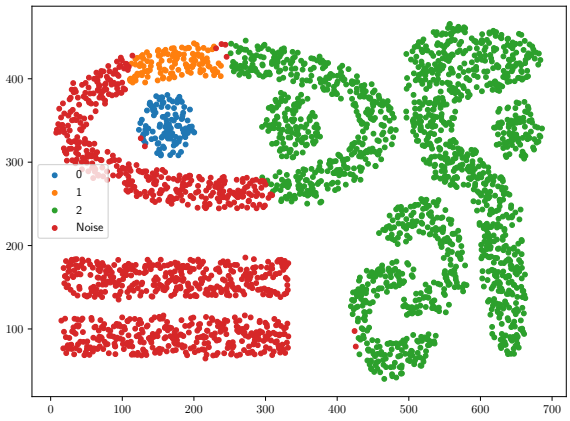
I presented results of the project encompassing the implementation of the DBSCAN+ algorithm with respect to the Tanimoto measure with the application of RealVecLen and ZPNVecLen heuristics. Firstly, I described the modification of the original DBSCAN algorithm which allows border points to be assigned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



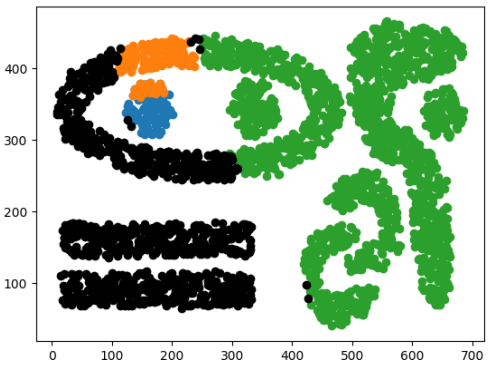
(a) `Cluto` - C++



(b) `Cluto` - Python



(c) `Complex` - C++



(d) `Complex` - Python

Figure 9: 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 10.

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.