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

Project Report

Data Mining

Kacper Kania

Contents

1 Description of the task

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+. 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. It can be approximated using methods RealVecLen [2] (if data points are real valued) and ZNPVecLen [2] (if data points are ternary valued).
Overall, the project contains an implementation of DBSCAN+ algorithm with respect to 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.
- Each dataset is z-standardized, *i.e.* each feature is normalized to have the mean equal to and the standard deviation equal to 1.
- Both purity and RAND are calculated assuming that "noise" label of points create yet another cluster to consider.
- RAND measure incorporates the fact that a border point can belong to many clusters - true positive count is increase 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 explicitly 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-znp-realvec-len`

3 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.

4 Important design and implementation issues

The main part of the project is implemented according to the class diagram presented in Figure 1.
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
- create an artificial dataset consisting of three gaussian blobs placed around the $(0, 0)$ coordinate.

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

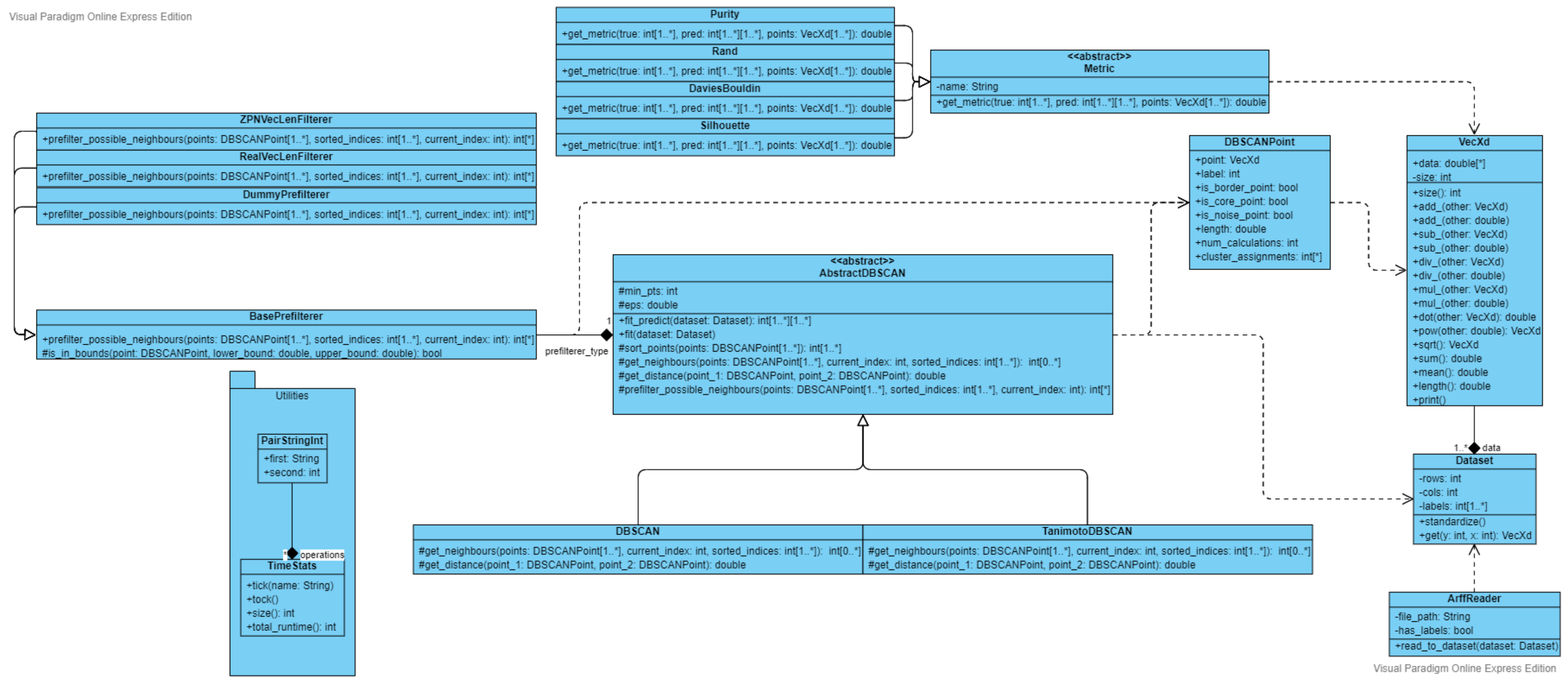

Visual Paradigm Online Express Edition
Purity
+get_metric(true: int[1..*], pred: int[1..*][1..*], points: VecXd[1..*]): double
Rand
+get_metric(true: int[1..*], pred: int[1..*][1..*], points: VecXd[1..*]): double
DaviesBouldin
+get_metric(true: int[1..*], pred: int[1..*][1..*], points: VecXd[1..*]): double
Silhouette
+get_metric(true: int[1..*], pred: int[1..*][1..*], points: VecXd[1..*]): double
<<abstract>> Metric
-name: String
+get_metric(true: int[1..*], pred: int[1..*][1..*], points: VecXd[1..*]): double
ZPNVecLenFilterer
+prefilter_possible_neighbours(points: DBSCANPoint[1..*], sorted_indices: int[1..*], current_index: int): int[*]
RealVecLenFilterer
+prefilter_possible_neighbours(points: DBSCANPoint[1..*], sorted_indices: int[1..*], current_index: int): int[*]
DummyPrefilterer
+prefilter_possible_neighbours(points: DBSCANPoint[1..*], sorted_indices: int[1..*], current_index: int): int[*]
BasePrefilterer
+prefilter_possible_neighbours(points: DBSCANPoint[1..*], sorted_indices: int[1..*], current_index: int): int[*]
#is_in_bounds(point: DBSCANPoint, lower_bound: double, upper_bound: double): bool
prefilterer_type
1
DBSCANPoint
+point: VecXd
+label: int
+is_border_point: bool
+is_core_point: bool
+is_noise_point: bool
+length: double
+num_calculations: int
+cluster_assignments: int[*]
VecXd
+data: double[*]
-size: int
+size(): int
+add_(other: VecXd)
+add_(other: double)
+sub_(other: VecXd)
+sub_(other: double)
+div_(other: VecXd)
+div_(other: double)
+mul_(other: VecXd)
+mul_(other: double)
+dot(other: VecXd): double
+pow(other: double): VecXd
+sqrt(): VecXd
+sum(): double
+mean(): double
+length(): double
+print()
<<abstract>> AbstractDBSCAN
#min_pts: int
#eps: double
+fit_predict(dataset: Dataset): int[1..*][1..*]
+fit(dataset: Dataset)
#sort_points(points: DBSCANPoint[1..*]): int[1..*]
#get_neighbours(points: DBSCANPoint[1..*], current_index: int, sorted_indices: int[1..*]): int[0..*]
#get_distance(point_1: DBSCANPoint, point_2: DBSCANPoint): double
#prefilter_possible_neighbours(points: DBSCANPoint[1..*], sorted_indices: int[1..*], current_index: int): int[*]
Utilities
PairStringInt
+first: String
+second: int
* operations
TimeStats
+tick(name: String)
+tock()
+size(): int
+total_runtime(): int
DBSCAN
#get_neighbours(points: DBSCANPoint[1..*], current_index: int, sorted_indices: int[1..*]): int[0..*]
#get_distance(point_1: DBSCANPoint, point_2: DBSCANPoint): double
TanimotoDBSCAN
#get_neighbours(points: DBSCANPoint[1..*], current_index: int, sorted_indices: int[1..*]): int[0..*]
#get_distance(point_1: DBSCANPoint, point_2: DBSCANPoint): double
1..* data
Dataset
-rows: int
-cols: int
-labels: int[1..*]
+standardize()
+get(y: int, x: int): VecXd
ArffReader
-file_path: String
-has_labels: bool
+read_to_dataset(dataset: Dataset)
Visual Paradigm Online Express Edition

Figure 1: The class diagram of the main part of the project. Auxiliary utilities are omitted for clarity.

5 User guide

5.1 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 6, 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 (if multiple repeats are performed).
- `<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.

5.2 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.

5.2.1 Binary compilation

Firtly, 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_znp_realvec_len` in the project directory.

5.2.2 Executing the binary

Once compiled, the program can be run as:

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

5.3 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 $[-\infty,$ `<min>`$]$, $[$`<min>`, `<max>`$]$ and $[$`<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
```

6 Experiments

The experiments served to analyze and understand the behavior of DBSCAN+ algorithm with respect to the 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

Lastly, I show how filtering heuristics influence computational complexity of the DBSCAN+ fitting in terms of number of data point evaluations and runtime measured in milliseconds. Since 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 value in each dataset.

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

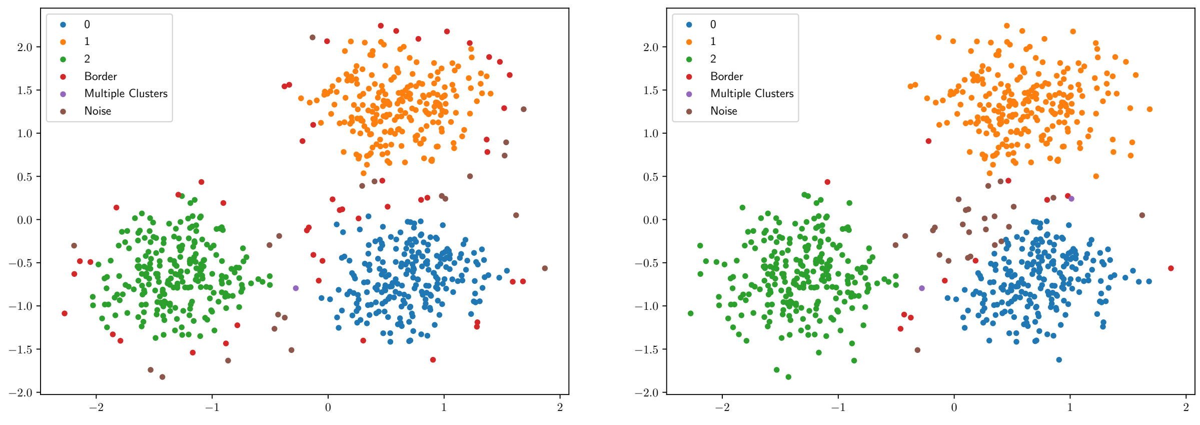
Quality of the clustering was measured with the following metrics: Purity, RAND, Silhouette, Davis-Bouldin. Each experiment was performed once due to computational limitations.

6.1 Empirical validation of the implementation

To validate the implementation, I prepared a simple dataset consisting of 750 data points. Each data point is sampled from a mixture of three gaussians placed at $\{1, 1\}$, $\{-1, -1\}$ and $\{1, -1\}$

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

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 2.



(a) Euclidean, $\varepsilon = 0.3$, $\texttt{minPts} = 10$ (b) Tanimoto, $\varepsilon = 0.9$, $\texttt{minPts} = 10$

Figure 2: Clustering results on an artificial datasets made of three gaussian blobs for both Euclidean and Tanimoto measures. Parameters of the DBSCAN+ were set appropriately for each measure. The "Multiple Clusters" label means that a border point was assigned to more than one cluster.

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 3.

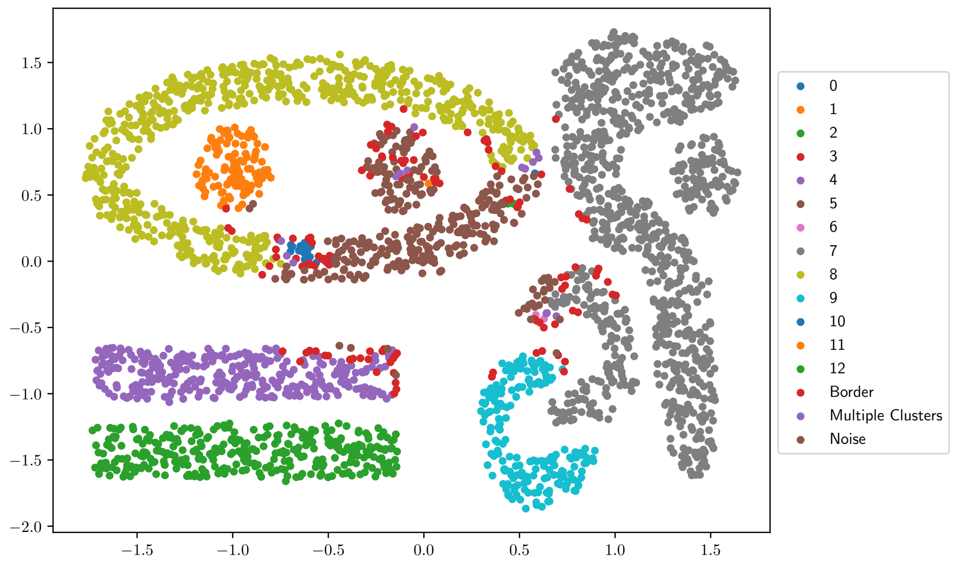


Figure 3: Visualization of the DBSCAN+ algorithm result with parameters $\varepsilon = 0.99$ and $\texttt{minPts} = 10$, when Tanimoto measure is used. The "Multiple Clusters" label means that a border point was assigned to more than one cluster.

6.2 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. These two issues arise

from the problem of quality measurement in clustering - there exist no one good metric that is able objectively say about quality of the clustering.

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.

ε	Purity		RAND		Silhouette		Davis-Bouldin	
	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
0.0010	0.2758	0.2758	0.1634	0.1634	0.0000	0.0000	0.0000	0.0000
0.0100	0.2785	0.2758	0.1683	0.1634	**1.0000**	0.0000	0.0091	0.0000
0.0500	**0.9890**	0.2758	**0.9941**	0.1634	-0.2239	0.0000	**1.2158**	0.0000
0.1000	0.2894	0.2758	0.1865	0.1634	-0.4623	0.0000	0.2939	0.0000
0.1500	0.2761	0.2758	0.1639	0.1634	0.0000	0.0000	0.0000	0.0000
0.2000	0.2758	0.2758	0.1634	0.1634	0.0000	0.0000	0.0000	0.0000
0.2500	0.2758	0.2758	0.1634	0.1634	0.0000	0.0000	0.0000	0.0000
0.3000	0.2758	0.2758	0.1634	0.1635	0.0000	0.0000	0.0000	0.0000
0.3500	0.2758	0.2758	0.1634	0.1635	0.0000	0.0000	0.0000	0.0000
0.4000	0.2758	0.2758	0.1634	0.1635	0.0000	0.0000	0.0000	0.0000
0.4500	0.2758	0.2758	0.1634	0.1635	0.0000	0.0000	0.0000	0.0000
0.5000	0.2758	0.2758	0.1634	0.1635	0.0000	0.0000	0.0000	0.0000
0.5500	0.2758	0.2758	0.1634	0.1635	0.0000	0.0000	0.0000	0.0000
0.6500	0.2758	0.2758	0.1634	0.1636	0.0000	0.0000	0.0000	0.0000
0.7000	0.2758	0.2758	0.1634	0.1639	0.0000	0.0000	0.0000	0.0000
0.7500	0.2758	0.2758	0.1634	0.1643	0.0000	0.0000	0.0000	0.0000
0.8000	0.2758	0.2758	0.1634	0.1654	0.0000	-0.2755	0.0000	5.5803
0.8500	0.2758	0.2758	0.1634	0.1656	0.0000	-0.2753	0.0000	5.5801
0.9000	0.2758	0.2758	0.1634	0.1668	0.0000	-0.2738	0.0000	5.5779
0.9500	0.2758	0.2758	0.1634	0.1709	0.0000	-0.2670	0.0000	**5.5971**
0.9600	0.2758	0.2773	0.1634	0.1745	0.0000	-0.3461	0.0000	2.9339
0.9700	0.2758	0.2765	0.1634	0.1783	0.0000	-0.3425	0.0000	5.3170
0.9800	0.2758	0.2764	0.1634	0.1879	0.0000	-0.3792	0.0000	5.5343
0.9900	0.2758	0.2783	0.1634	0.2084	0.0000	-0.4709	0.0000	3.9114
0.9990	0.2758	**0.8577**	0.1634	**0.8904**	0.0000	-0.0293	0.0000	0.7386
0.9999	0.2758	0.3440	0.1634	0.2679	0.0000	**0.9772**	0.0000	0.2488
1.0000	0.2758	0.2758	0.1634	0.1634	0.0000	0.0000	0.0000	0.0000

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.

ε	Purity		RAND		Silhouette		Davis-Bouldin	
	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
0.0010	0.2989	0.2989	0.1859	0.1859	0.0000	0.0000	0.0000	0.0000
0.0100	0.2989	0.2989	0.1859	0.1859	0.0000	0.0000	0.0000	0.0000
0.0500	**0.8657**	0.2989	0.7945	0.1859	**0.5001**	0.0000	0.5366	0.0000
0.1000	0.7552	0.2989	**0.8536**	0.1859	-0.1045	0.0000	**0.9477**	0.0000
0.1500	0.5490	0.2989	0.5983	0.1861	-0.0444	0.0000	0.6614	0.0000
0.2000	0.4121	0.2989	0.5307	0.1861	0.3287	0.0000	0.4582	0.0000
0.2500	0.4121	0.2989	0.5307	0.1867	0.3287	0.0000	0.4582	0.0000
0.3000	0.4121	0.2989	0.5307	0.1867	0.3287	0.0000	0.4582	0.0000
0.3500	0.4121	0.2989	0.5307	0.1867	0.3287	0.0000	0.4582	0.0000
0.4000	0.4121	0.2989	0.5307	0.1869	0.3287	0.0000	0.4582	0.0000
0.4500	0.4121	0.2989	0.5307	0.1869	0.3287	0.0000	0.4582	0.0000
0.5000	0.2989	0.2989	0.1859	0.1869	0.0000	0.0000	0.0000	0.0000
0.5500	0.2989	0.2989	0.1859	0.1875	0.0000	0.0000	0.0000	0.0000
0.6500	0.2989	0.2989	0.1859	0.1891	0.0000	0.0000	0.0000	0.0000
0.7000	0.2989	0.2989	0.1859	0.1893	0.0000	0.0000	0.0000	0.0000
0.7500	0.2989	0.2989	0.1859	0.1893	0.0000	0.0000	0.0000	0.0000
0.8000	0.2989	0.2989	0.1859	0.1896	0.0000	0.0000	0.0000	0.0000
0.8500	0.2989	0.2989	0.1859	0.1904	0.0000	0.0000	0.0000	0.0000
0.9000	0.2989	0.2989	0.1859	0.1962	0.0000	-0.2825	0.0000	**7.9699**
0.9500	0.2989	0.4121	0.1859	0.5387	0.0000	-0.0341	0.0000	1.1656
0.9600	0.2989	0.4121	0.1859	0.5399	0.0000	-0.0309	0.0000	1.1755
0.9700	0.2989	0.4121	0.1859	0.5413	0.0000	0.3463	0.0000	0.4620
0.9800	0.2989	0.4121	0.1859	0.5424	0.0000	-0.0499	0.0000	2.5284
0.9900	0.2989	0.6269	0.1859	**0.7191**	0.0000	-0.0956	0.0000	1.5798
0.9990	0.2989	**0.7245**	0.1859	0.7084	0.0000	0.6489	0.0000	0.4690
0.9999	0.2989	0.3006	0.1859	0.1891	0.0000	**1.0000**	0.0000	0.0052
1.0000	0.2989	0.2989	0.1859	0.1859	0.0000	0.0000	0.0000	0.0000

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.

ε	Purity		RAND		Silhouette		Davis-Bouldin	
	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
0.0010	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.0100	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.0500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.1000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.1500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.2000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.2500	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.3000	0.0575	0.0406	0.0699	0.0384	**1.0000**	0.0000	0.0000	0.0000
0.3500	0.0639	0.0406	0.0815	0.0384	0.9976	0.0000	0.1321	0.0000
0.4000	0.0701	0.0406	0.0926	0.0384	0.9957	0.0000	0.1865	0.0000
0.4500	0.0945	0.0407	0.1370	0.0385	0.9806	0.0000	0.4034	0.0000
0.5000	0.1120	0.0407	0.1683	0.0385	0.9683	0.0000	0.4989	0.0000
0.5500	0.1382	0.0407	0.2147	0.0387	0.9435	0.0000	0.5847	0.0000
0.6500	0.1868	0.0410	0.2981	0.0422	0.9042	0.0000	0.6828	0.0000
0.7000	0.2150	0.0420	0.3433	0.0512	0.8818	0.0000	0.7102	0.0000
0.7500	0.2549	0.0459	0.4120	0.0829	0.8409	-0.2899	0.7567	0.7729
0.8000	0.2944	0.0637	0.4735	0.1746	0.8097	-0.2903	0.7853	0.9759
0.8500	0.3320	0.1387	0.5409	0.4120	0.7578	-0.1555	0.8119	0.9742
0.9000	0.3716	**0.3810**	0.6045	**0.8319**	0.7028	0.3026	0.8343	**1.0071**
0.9500	0.3975	0.3342	0.6644	0.5395	0.6422	0.7680	0.8376	0.7784
0.9600	0.4010	0.2770	0.6756	0.4479	0.6285	0.8210	0.8369	0.7692
0.9700	0.3966	0.2099	0.6849	0.3444	0.6144	0.8846	0.8349	0.6954
0.9800	0.3942	0.1561	0.6959	0.2466	0.5951	0.9320	0.8444	0.6090
0.9900	0.4033	0.0921	0.7080	0.1331	0.5837	0.9790	0.8590	0.4183
0.9990	0.4100	0.0575	0.7184	0.0699	0.5714	**1.0000**	0.8617	0.0000
0.9999	**0.4104**	0.0575	**0.7194**	0.0699	0.5701	**1.0000**	**0.8639**	0.0000
1.0000	**0.4104**	0.0575	**0.7194**	0.0699	0.5701	**1.0000**	**0.8639**	0.0000

6.3 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.

Dataset	`minPts`	Purity		RAND		Silhouette		Davis-Bouldin	
		Euc	Tan	Euc	Tan	Euc	Tan	Euc	Tan
`Cluto`	0	0.2819	0.2834	0.1737	0.1987	-0.7206	-0.5605	0.2743	**11.3573**
	1	0.2819	0.2834	0.1737	0.1987	-0.7206	-0.5605	0.2743	**11.3573**
	5	0.2894	0.2783	0.1865	0.2084	-0.4623	-0.4709	0.2939	3.9114
	10	0.4190	0.2772	0.5487	0.2305	-0.1387	-0.3988	0.5063	2.8378
	25	0.6575	0.4210	0.8160	0.6207	-0.0134	0.0485	**1.2228**	1.1707
	50	**0.9216**	0.6168	**0.8863**	**0.8191**	**0.3995**	0.2391	0.6533	1.0067
	100	0.2758	**0.6588**	0.1634	0.7239	0.0000	**0.6409**	0.0000	0.7205
`Complex`	0	0.7552	0.6292	0.8536	0.7149	-0.1045	-0.1965	0.9477	**1.7904**
	1	0.7552	0.6292	0.8536	0.7149	-0.1045	-0.1965	0.9477	**1.7904**
	5	0.7552	0.6269	0.8536	0.7191	-0.1045	-0.0956	0.9477	1.5798
	10	**0.9983**	**0.8961**	**0.9993**	**0.9110**	-0.0090	0.0121	**1.0659**	1.0610
	25	0.4193	0.8773	0.3919	0.8612	**0.9328**	0.3894	0.3566	0.7472
	50	0.2989	0.5470	0.1859	0.5444	0.0000	**0.8932**	0.0000	0.2882
	100	0.2989	0.2989	0.1859	0.1859	0.0000	0.0000	0.0000	0.0000
`Letter`	0	**1.0000**	**1.0000**	**0.9616**	**0.9616**	**1.0000**	0.9445	0.0000	0.1556
	1	**1.0000**	**1.0000**	**0.9616**	**0.9616**	**1.0000**	0.9445	0.0000	0.1556
	5	0.0575	0.0921	0.0699	0.1331	**1.0000**	0.9790	0.0000	**0.4183**
	10	0.0443	0.0595	0.0452	0.0731	**1.0000**	**0.9950**	**0.0000**	0.3202
	25	0.0420	0.0428	0.0409	0.0425	0.0000	0.0000	0.0000	0.0000
	50	0.0406	0.0406	0.0384	0.0384	0.0000	0.0000	0.0000	0.0000
	100	0.0406	0.0406	0.0384	0.0384	0.0000	0.0000	0.0000	0.0000

6.4 Influence of applying prefiltering heuristics

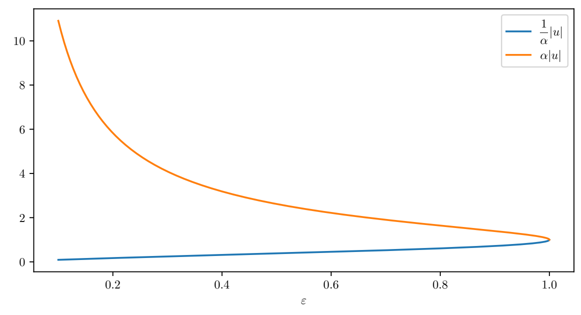
Finally, I tested how the application of RealVecLen and ZPNVecLen heuristics affects computation time on each of datasets when ε is varied. These heuristics are applicable only when the Tanimoto measure is used. Table 6 represents how many times a point is compared to other points when a neighborhood 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 4. 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.

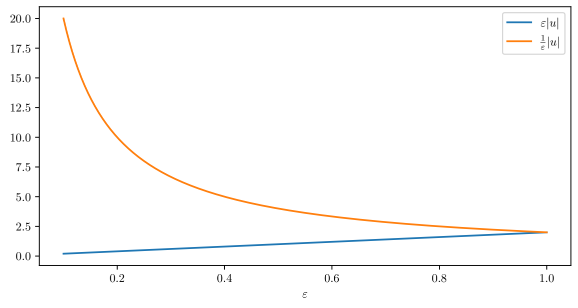
Lastly, the decreasing number of point comparison correlates with decreased time of the DB-SCAN fitting to the data. It means that the lower number of point comparisons, the less time the algorithm spends on finding neighbors of each points hence the program runs faster in general. Table 7 shows how 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 pre-filtering), *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	9999	3031	3030	3030	20000	19999	19999
0.0010	10000	9999	9999	3031	3029	3030	20000	19999	19999
0.0100	10000	9996	9999	3031	3024	3030	20000	19999	19999
0.0500	10000	9945	9999	3031	3007	2683	20000	19999	19998
0.1000	10000	9839	9999	3031	2974	2407	20000	19999	19979
0.1500	10000	9724	9999	3031	2940	2407	20000	19999	19868
0.2000	10000	9602	9999	3031	2906	1443	20000	19998	19608
0.2500	10000	9474	9999	3031	2870	1287	20000	19994	19156
0.3000	10000	9340	9511	3031	2831	1287	20000	19982	18504
0.3500	10000	9200	8203	3031	2791	1287	20000	19949	17658
0.4000	10000	9048	7532	3031	2748	1275	20000	19881	16640
0.4500	10000	8879	7158	3031	2699	1076	20000	19759	15456
0.5000	10000	8686	6684	3031	2643	1076	20000	19566	14137
0.5500	10000	8462	6158	3031	2580	1027	20000	19281	12724
0.6500	10000	7895	5139	3031	2423	1027	20000	18343	9771
0.7000	10000	7526	5139	3031	2317	1017	20000	17632	8283
0.7500	10000	7082	5139	3031	2185	1017	20000	16710	6808
0.8000	10000	6538	4642	3031	2018	1015	20000	15516	5358
0.8500	10000	5857	3501	3031	1808	**1004**	20000	13956	3946
0.9000	10000	4965	2854	3031	1534	**1004**	20000	11842	2580
0.9500	10000	3662	**1694**	3031	1136	**1004**	20000	8711	1264
0.9600	10000	3306	**1694**	3031	1027	**1004**	20000	7854	1007
0.9700	10000	2890	**1694**	3031	899	**1004**	20000	6856	755
0.9800	10000	2384	**1694**	3031	743	**1004**	20000	5643	504
0.9900	10000	1704	**1694**	3031	533	**1004**	20000	4023	255
0.9990	10000	545	**1694**	3031	171	**1004**	20000	1283	37
0.9999	10000	172	**1694**	3031	54	**1004**	20000	406	12
1.0000	10000	**0**	**1694**	3031	**0**	**1004**	20000	**0**	**9**



(a) RealVecLen



(b) ZPNVecLen

Figure 4: 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 miliseconds 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	12181	13498	12177	984	1013	964	83947	88587	79361
0.0010	12397	13196	12077	1043	1063	973	85855	86377	84341
0.0100	12538	13206	12033	1031	1082	969	86620	86702	83895
0.0500	12149	12956	12627	1046	1073	885	84618	84202	81611
0.1000	11540	12031	11680	985	996	762	79142	77576	76097
0.1500	11356	11630	11068	967	970	759	74230	77140	73424
0.2000	11049	11293	11224	945	941	510	72827	74016	71014
0.2500	11004	11148	11083	965	910	468	71285	71962	68169
0.3000	10758	10797	10786	898	900	472	70937	71010	66014
0.3500	10839	10544	8681	910	873	463	71449	70364	62079
0.4000	10670	10118	8110	892	858	479	69382	69627	57768
0.4500	10624	10011	7729	880	828	414	68501	68945	54711
0.5000	10466	9845	6921	881	791	414	76170	76734	53663
0.5500	10206	9297	6572	859	766	403	75801	75350	48522
0.6500	10197	8716	5582	843	710	402	68307	64783	36109
0.7000	10047	7995	5458	840	678	413	69980	63180	29839
0.7500	9931	7602	5753	830	633	400	68247	58456	24567
0.8000	9939	7361	5222	820	594	407	68366	54799	19191
0.8500	11649	6091	4091	824	521	400	68389	48782	13874
0.9000	9624	5170	3484	829	457	402	69857	41824	9113
0.9500	9590	3754	2361	801	331	401	68467	30194	4684
0.9600	9644	3372	2669	799	294	**395**	69100	27628	3817
0.9700	9364	2982	2326	794	263	400	68187	24126	3296
0.9800	9530	2437	2506	786	214	396	68338	19484	2211
0.9900	9478	1746	2319	809	157	**395**	68290	14109	1492
0.9990	9354	634	2325	783	62	400	69262	4812	821
0.9999	9466	294	**2292**	784	31	408	68515	1929	748
1.0000	9407	**124**	2330	789	**13**	407	68802	**688**	**728**
ρ	–	0.9880	0.9920	–	0.9899	0.9986	–	0.9830	0.9943

7 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.

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.