DenMune

Release 0.1

Mohamed Abbas

Aug 15, 2023

CONTENTS

DenMune Clustering Algorithm

A clustering algorithm that can find clusters of arbitrary size, shapes and densities in two-dimensions. Higher dimensions are first reduced to 2-D using the t-sne. The algorithm relies on a single parameter K (the number of nearest neighbors). The results show the superiority of DenMune. Enjoy the simplicty but the power of DenMune.

Note: This documentation associated with the paper "DenMune: Density peak based clustering using mutual nearest neighbors"

DOI: https://doi.org/10.1016/j.patcog.2020.107589

Source code is maintained at https://github.com/scikit-learn-contrib/denmune-clustering-algorithm

---

CHAPTER

ONE

USER GUIDE / TUTORIALS

1.1 DenMune: A density-peak clustering algorithm

DenMune a clustering algorithm that can find clusters of arbitrary size, shapes and densities in two-dimensions. Higher dimensions are first reduced to 2-D using the t-sne. The algorithm relies on a single parameter K (the number of nearest neighbors). The results show the superiority of the algorithm. Enjoy the simplicity but the power of DenMune.

1.1.1 Based on the paper

Paper	Journal
Mohamed Abbas, Adel El-Zoghabi, Amin Ahoukry	
DenMune: Density peak based clustering using mutual nearest neighbors	
In: Journal of Pattern Recognition, Elsevier,	
volume 109, number 107589, January 2021	
DOI: https://doi.org/10.1016/j.patcog.2020 .107589	

1.1.2 Documentation:

Documentation, including tutorials, are available on https://denmune.readthedocs.io

1.1.3 Watch it in action

This 30 seconds will tell you how a density-baased algorithm, DenMune propagates:

1.1.4 When less means more

Most calssic clustering algorithms fail in detecting complex where clusters are of different size, shape, density, and being exist in noisy data. Recently, a density-based algorithm named DenMune showed great ability in detecting complex shapes even in noisy data. it can detect number of clusters automatically, detect both pre-identified-noise and post-identified-noise automatically and removing them.

It can achieve accuracy reach 100% in most classic pattern problems, achieve 97% in MNIST dataset. A great advantage of this algorithm is being single-parameter algorithm. All you need is to set number of k-nearest neighbor and the algorithm will care about the rest. Being Non-senstive to changes in k, make it robust and stable.

Keep in mind, the algorithm reduce any N-D dataset to only 2-D dataset initially, so it is a good benefit of this algorithm is being always to plot your data and explore it which make this algorithm a good candidate for data exploration. Finally, the algorithm comes with neat package for visualizing data, validating it and analyze the whole clustering process.

1.2 Installation and Usage

1.2.1 Install It:

Simply install DenMune clustering algorithm using pip command from the official Python repository

From the shell run the command

```
pip install denmune
```

From jupyter notebook cell run the command

```
!pip install denmune
```

1.2.2 Import It:

Once DenMune is installed, you just need to import it

```
from denmune import DenMune
```

Note: Please note that first `denmune` (the package) is in small letters, while `DenMune` (the class itself) has `D` and `M` in capital case while other letters are small.

1.3 Loading data

There are four possible cases of data:

- only train data without labels
- only labeld train data
- labeled train data in addition to test data without labels
- labeled train data in addition to labeled test data

```
#==========================================
# First scenario: train data without labels
# ==========================================

data_path = 'datasets/denmune/chameleon/'
dataset = "t7.10k.csv"
data_file = data_path + dataset

# train data without labels
X_train = pd.read_csv(data_file, sep=',', header=None)

knn = 39 # k-nearest neighbor, the only parameter required by the algorithm

dm = DenMune(train_data=X_train, k_nearest=knn)
labels, validity = dm.fit_predict(show_analyzer=False, show_noise=True)
```

This is an intutive dataset which has no groundtruth provided

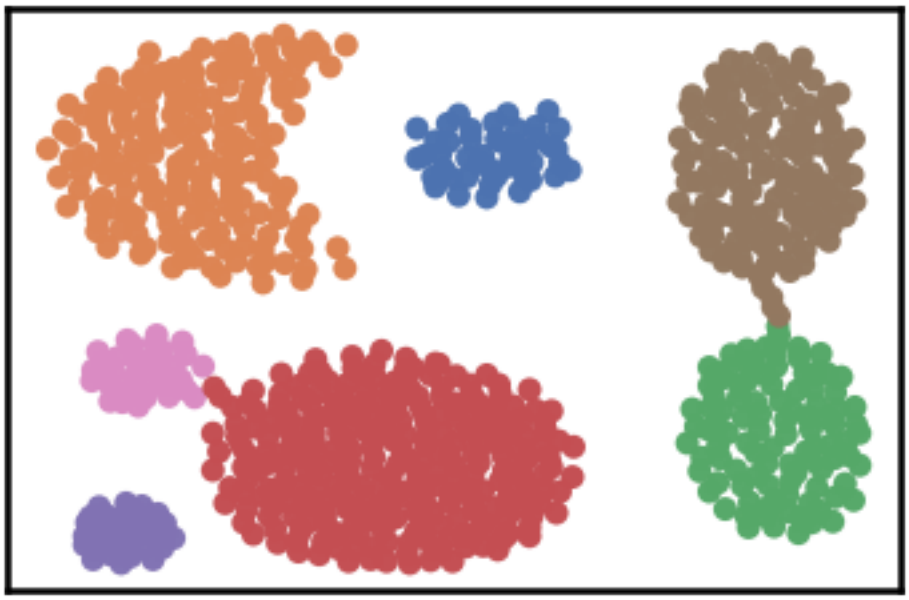
```
#==========================================
# Second scenario: train data with labels
# ==========================================

data_path = 'datasets/denmune/shapes/'
dataset = "aggregation.csv"
data_file = data_path + dataset

# train data with labels
X_train = pd.read_csv(data_file, sep=',', header=None)
y_train = X_train.iloc[:, -1]
X_train = X_train.drop(X_train.columns[-1], axis=1)

knn = 6 # k-nearest neighbor, the only parameter required by the algorithm
```

(continues on next page)

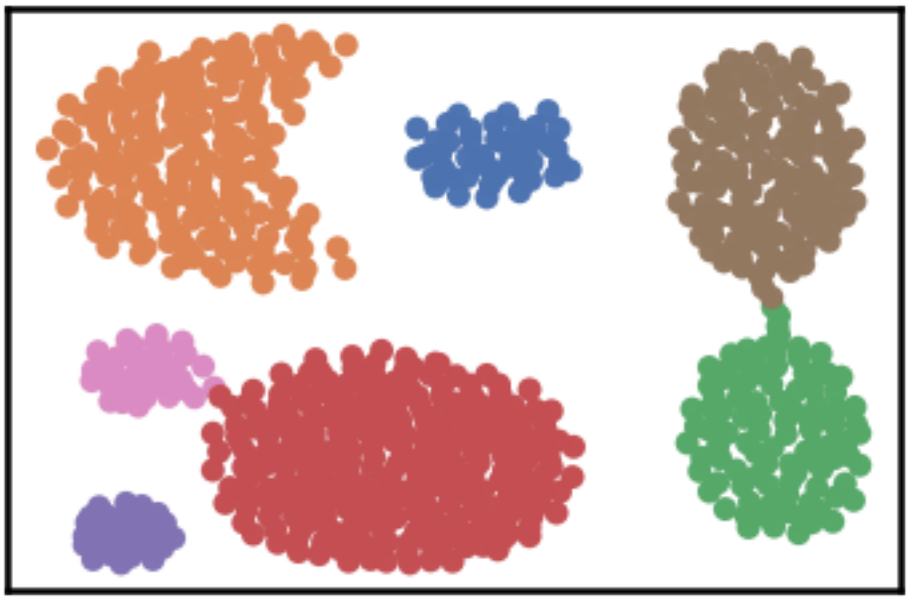
(continued from previous page)

```
dm = DenMune(train_data=X_train, train_truth= y_train, k_nearest=knn)
labels, validity = dm.fit_predict(show_analyzer=False, show_noise=True)
```

Datset groundtruth



Datset as detected by DenMune at k=6



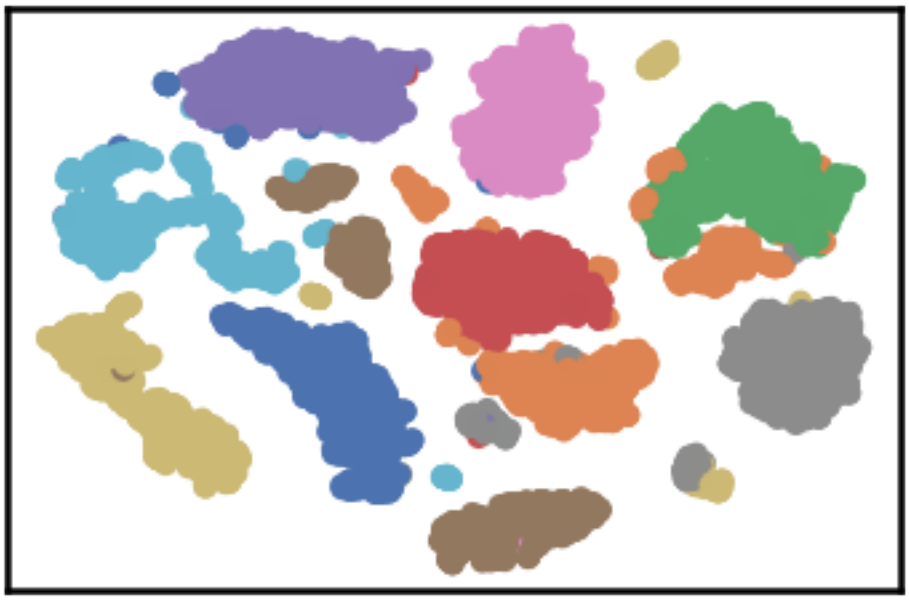
```
#=================================================================
# Third scenario: train data with labels in addition to test data
```

(continues on next page)

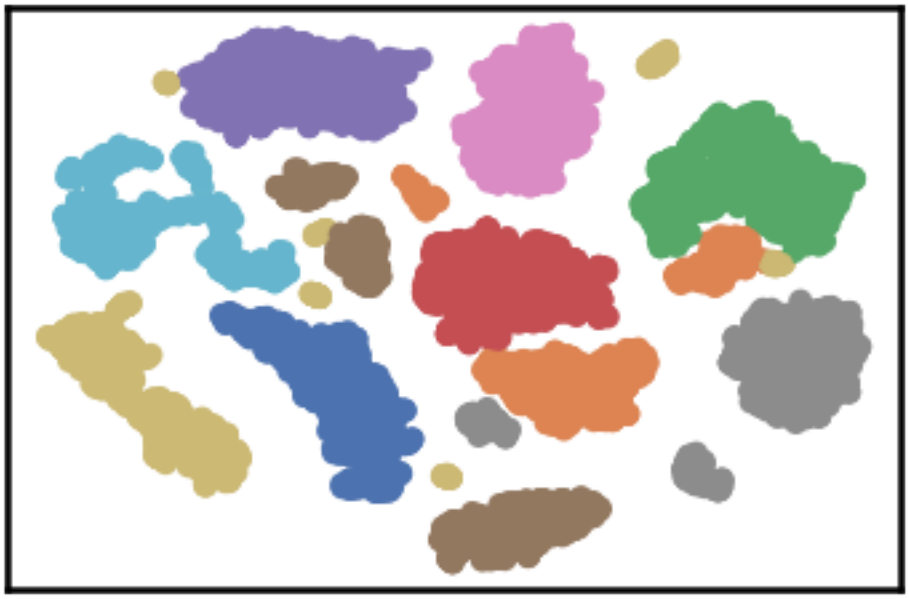
(continued from previous page)

```
# ============================================================

data_path = 'datasets/denmune/pendigits/'
file_2d = data_path + 'pendigits-2d.csv'

# train data with labels
X_train = pd.read_csv(data_path + 'train.csv', sep=',', header=None)
y_train = X_train.iloc[:, -1]
X_train = X_train.drop(X_train.columns[-1], axis=1)

# test data without labels
X_test = pd.read_csv(data_path + 'test.csv', sep=',', header=None)
X_test = X_test.drop(X_test.columns[-1], axis=1)

knn = 50 # k-nearest neighbor, the only parameter required by the algorithm

dm = DenMune(train_data=X_train, train_truth= y_train,
             test_data= X_test,
             k_nearest=knn)
labels, validity = dm.fit_predict(show_analyzer=True, show_noise=True)
```

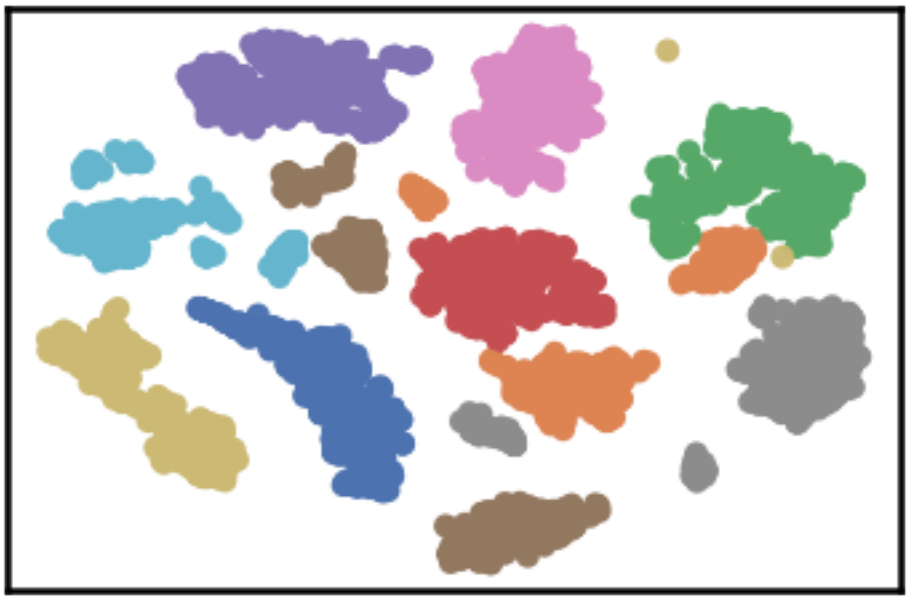
dataset groundtruth



dataset as detected by DenMune at k=50



test data as predicted by DenMune on training the dataset at k=50



1.4 Algorithm's Parameters

1. Parameters used within the initialization of the DenMune class

```
def __init__ (self,
                  train_data=None, test_data=None,
                  train_truth=None, test_truth=None,
                  file_2d ='_temp_2d', k_nearest=10,
                  rgn_tsne=False, prop_step=0,
                  ):
```

- train_data:
  - data used for training the algorithm
  - default: None. It should be provided by the use, otherwise an error will riase.
- train_truth:
  - labels of training data
  - default: None
- test_data:
  - data used for testing the algorithm
- test_truth:
  - labels of testing data
  - default: None
- k_nearest:
  - number of nearest neighbor
  - default: 10. It should be provided by the user.
- rgn_tsn:
  - when set to True: It will regenerate the reduced 2-D version of the N-D dataset each time the algorithm run.
  - when set to False: It will generate the reduced 2-D version of the N-D dataset first time only, then will reuse the saved exist file
  - default: True
- file_2d: name (include location) of file used save/load the reduced 2-d version
  - if empty: the algorithm will create temporary file named '_temp_2d'
  - default: _temp_2d
- prop_step:
  - size of increment used in showing the clustering propagation.
  - leave this parameter set to 0, the default value, unless you are willing intentionally to enter the propagation mode.
  - default: 0

2. Parameters used within the fit_predict function:

```
def fit_predict(self,
                    validate=True,
                    show_plots=True,
                    show_noise=True,
                    show_analyzer=True
                    ):
```

- validate:
  - validate data on/off according to five measures integrated with DenMUne (Accuracy. F1-score, NMI index, AMI index, ARI index)
  - default: True
- show_plots:
  - show/hide plotting of data
  - default: True
- show_noise:
  - show/hide noise and outlier
  - default: True
- show_analyzer:
  - show/hide the analyzer
  - default: True

1.5 Features

1.5.1 The Analyzer

The algorithm provide an intutive tool called analyzer, once called it will provide you with in-depth analysis on how your clustering results perform.

```
Validating train data
├── exec_time
│   ├── DenMune: 8.878
│   ├── NGT: 0.994
│   └── t_SNE: 115.073
├── n_clusters
│   ├── actual: 10
│   └── detected: 10
├── n_points
│   ├── dim: 16
│   ├── noise
│   │   ├── type-1: 0
│   │   └── type-2: 0
│   ├── plot_size: 7494
│   ├── size: 10992
│   ├── strong: 5870
│   └── weak
│       ├── all: 5122
│       ├── failed to merge: 0
│       └── succeeded to merge: 5122
└── validity
    └── train
        ├── ACC: 7245
        ├── AMI: 0.936
        ├── ARI: 0.929
        ├── F1: 0.967
        ├── NMI: 0.936
        ├── completeness: 0.937
        └── homogeneity: 0.936

Validating test data
├── ACC: 3441
├── AMI: 0.965
├── ARI: 0.964
├── F1: 0.984
├── NMI: 0.966
├── completeness: 0.966
└── homogeneity: 0.965
```

1.5.2 Noise Detection

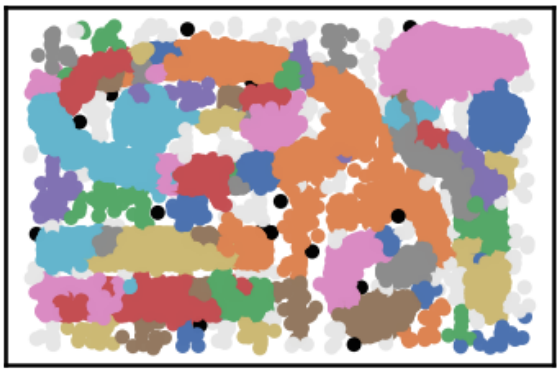
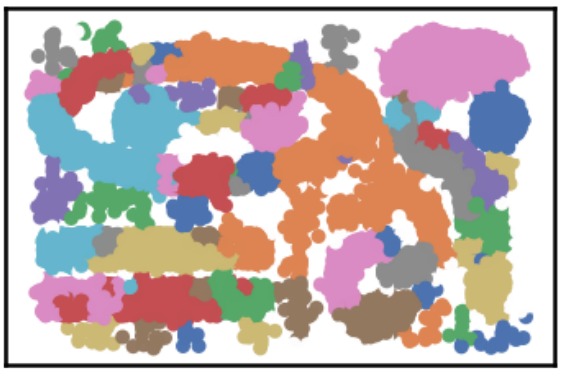
DenMune detects noise and outlier automatically, no need to any further work from your side.

- It plots pre-identified noise in black
- It plots post-identified noise in light grey

You can set show_noise parameter to False.

```
# let us show noise

m = DenMune(train_data=X_train, k_nearest=knn)
labels, validity = dm.fit_predict(show_noise=True)
```

```
# let us show clean data by removing noise

m = DenMune(train_data=X_train, k_nearest=knn)
labels, validity = dm.fit_predict(show_noise=False)
```

noisy data	clean data
	

1.5.3 Validatation

You can get your validation results using 3 methods

- by showing the Analyzer
- extract values from the validity returned list from fit_predict function
- extract values from the Analyzer dictionary

There are five validity measures built-in the algorithm, which are:

- ACC, Accuracy
- F1 score
- NMI index (Normalized Mutual Information)
- AMI index (Adjusted Mutual Information)
- ARI index (Adjusted Rand Index)

```
# secondly, we can extract validity returned list from fit_predict function
dm = DenMune(train_data=X_train, train_truth=y_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_plots=False, show_noise=True, show_analyzer=False)
validity
```

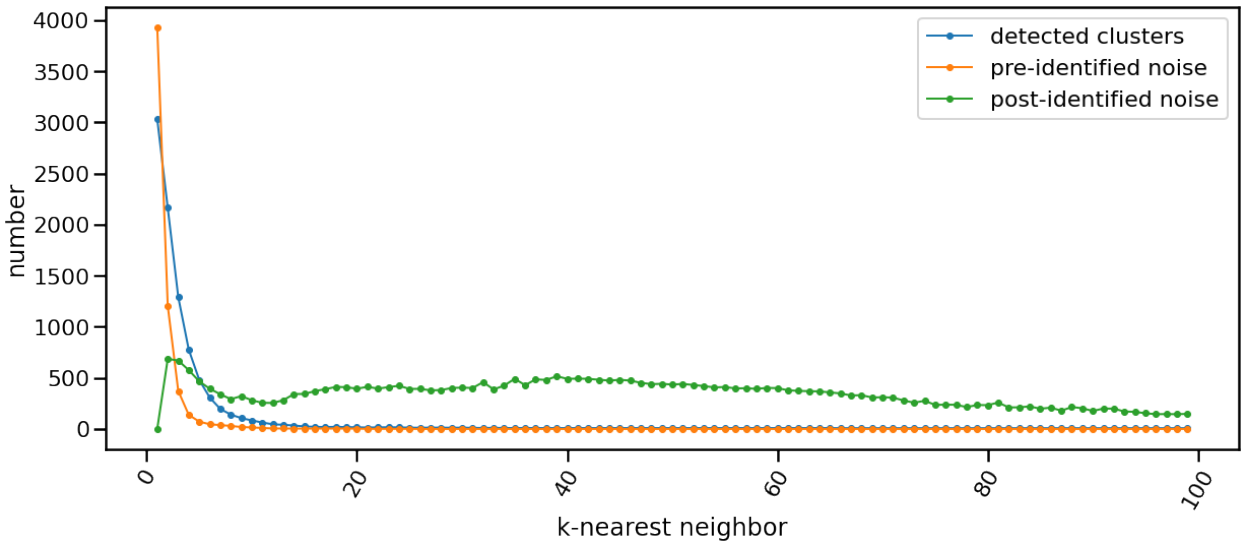
```
{'train': {'ACC': 785,
  'AMI': 0.9880984055236919,
  'ARI': 0.9927076502018027,
  'F1': 0.9962034083064701,
  'NMI': 0.9882680312048461,
  'completeness': 0.9873385166573364,
  'homogeneity': 0.9891992975556994}}
```

```
Accuracy = validity['train']['ACC']
print ('Accuracy:',Accuracy, 'correctely identified points')
F1_score = validity['train']['F1']
print ('F1 score:', round(F1_score*100,2), '%')
NMI = validity['train']['NMI']
print ('NMI index:', round(NMI*100,2), '%')
AMI = validity['train']['AMI']
print ('AMI index:', round(AMI*100,2), '%')
ARI = validity['train']['ARI']
print ('ARI index:', round(ARI*100,2), '%')
```

```
Accuracy: 785 correctely identified points
F1 score: 99.62 %
NMI index: 98.83 %
AMI index: 98.81 %
ARI index: 99.27 %
```

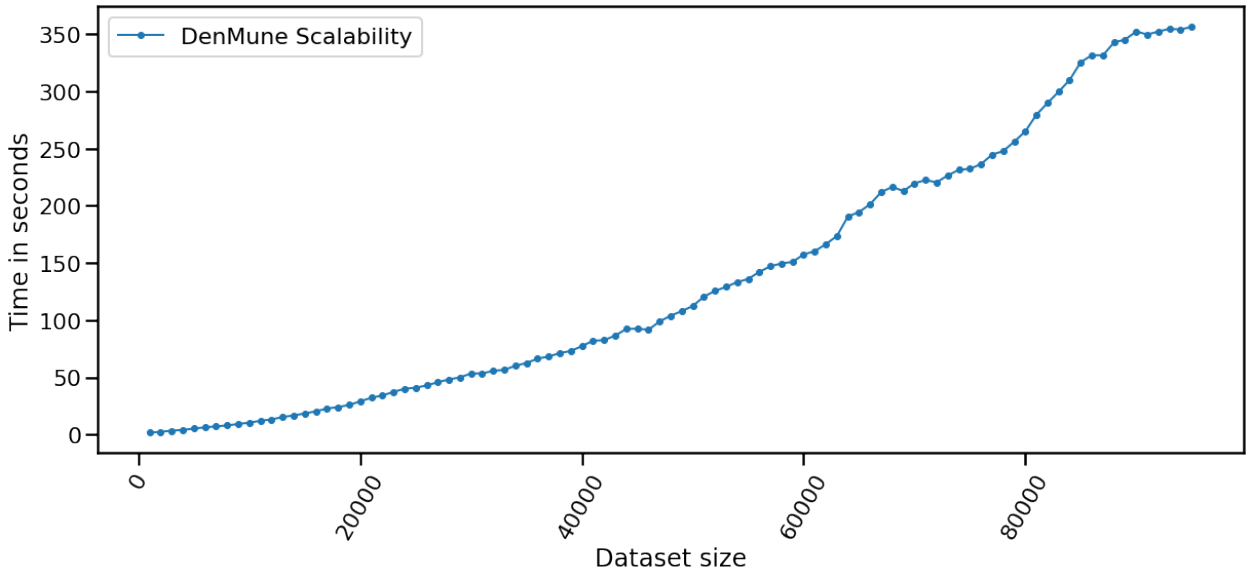
1.5.4 K-nearest Evolution

The following chart shows the evolution of pre and post identified noise in correspondence to increase of number of knn. Also, detected number of clusters is analyzed in the same chart in relation with both types of identified noise.



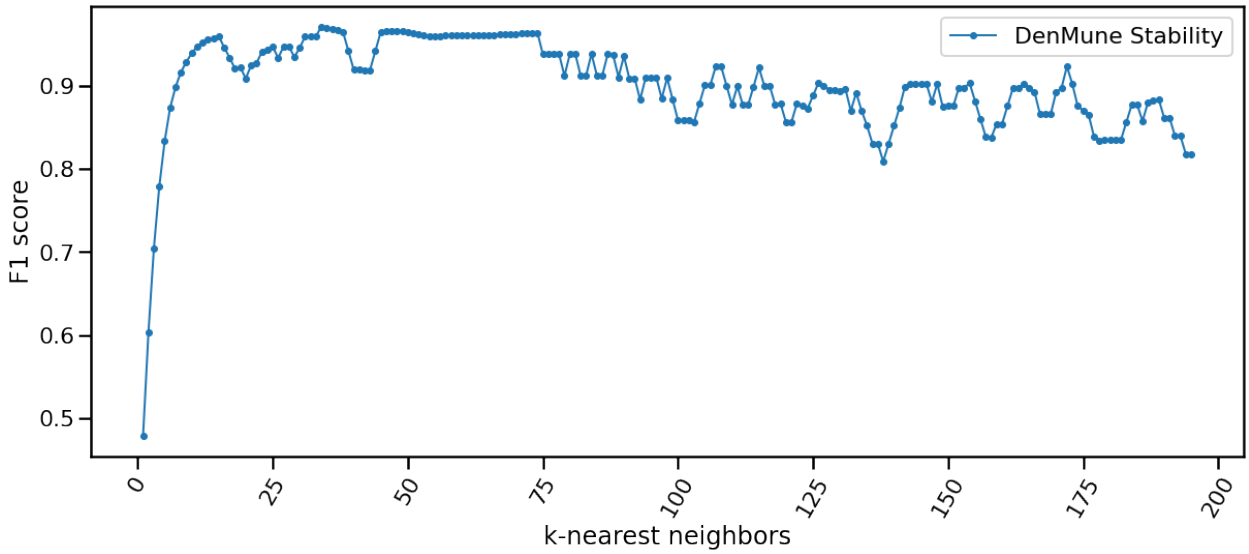
1.5.5 The Scalability

data size	time
data size: 5000	time: 2.3139 seconds
data size: 10000	time: 5.8752 seconds
data size: 15000	time: 12.4535 seconds
data size: 20000	time: 18.8466 seconds
data size: 25000	time: 28.992 seconds
data size: 30000	time: 39.3166 seconds
data size: 35000	time: 39.4842 seconds
data size: 40000	time: 63.7649 seconds
data size: 45000	time: 73.6828 seconds
data size: 50000	time: 86.9194 seconds
data size: 55000	time: 90.1077 seconds
data size: 60000	time: 125.0228 seconds
data size: 65000	time: 149.1858 seconds
data size: 70000	time: 177.4184 seconds
data size: 75000	time: 204.0712 seconds
data size: 80000	time: 220.502 seconds
data size: 85000	time: 251.7625 seconds
data size: 100000	time: 257.563 seconds



1.5.6 The Stability

The algorithm is only single-parameter, even more it not sensitive to changes in that parameter, k. You may guess that from the following chart yourself. This is of greate benfit for you as a data exploration analyst. You can simply explore the dataset using an arbitrary k. Being Non-senstive to changes in k, make it robust and stable.



1.5.7 Reveal the propagation

one of the top performing feature in this algorithm is enabling you to watch how your clusters propagate to construct the final output clusters. just use the parameter 'prop_step' as in the following example:

```
dataset = "t7.10k" #
data_path = 'datasets/denmune/chameleon/'

# train file
data_file = data_path + dataset +'.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)


from itertools import chain

# Denmune's Paramaters
knn = 39 # number of k-nearest neighbor, the only parameter required by the algorithm

# create list of differnt snapshots of the propagation
snapshots = chain(range(2,5), range(5,50,10), range(50, 100, 25), range(100,500,100),␣
↪range(500,2000, 250), range(1000,5500, 500))

from IPython.display import clear_output
for snapshot in snapshots:
    print ("itration", snapshot )
    clear_output(wait=True)
    dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False, prop_step=snapshot)
    labels, validity = dm.fit_predict(show_analyzer=False, show_noise=False)
```

1.6 How to Run and Test

1.6.1 Interact with the algorithm

```
#@title  { run: "auto", vertical-output: true, for
chameleon_dataset = "t7.10k.dat" #@param ["t4.8k.d
show_noize_checkbox = True #@param {type:"boolean"
show_analyzer_checkbox = True #@param {type:"boole
k_nearest_slider = 39 #@param {type:"slider", min:

# train file
data_file = data_path + chameleon_dataset
X_train = pd.read_csv(data_file, sep=',', header=N

dm = DenMune(train_data=X_train, k_nearest=k_neare
labels, validity = dm.fit_predict(show_noise=show_
                                  show_analyzer=sh
```

This notebook allows you interact with the algorithm in many asspects:

- you can choose which dataset to cluster (among 4 chameleon datasets)
- you can decide which number of k-nearest neighbor to use
- show noise on/off; thus you can invesitigate noise detected by the algorithm
- show analyzer on/off

1.6.2 Repo2Docker Binder

Launch Examples in Repo2Docker Binder

> Simply use our repo2docker offered by mybinder.org, which encapsulate the algorithm and all required data in one virtual machine instance. All jupter notebooks examples found in this repository will be also available to you in action to practice in this respo2docer. Thanks mybinder.org, you made it possible!

1.6.3 Kaggle workspace

Launch each Example in Kaggle workspace

> If you are a kaggler like me, then Kaggle, the best workspace where data scientist meet, should fit you to test the algorithm with great experince.

Dataset	Kaggle URL
When less means more - kaggle	
Non-groundtruth datasets - kaggle	
2D Shape datasets - kaggle	
MNIST dataset kaggle	
Iris dataset kaggle	
Training MNIST to get 97%	
Noise detection - kaggle	
Validation - kaggle	|
The beauty of propagation - kaggle	
The beauty of propagation part2 - kaggle	
Snapshots of propagation -kaggle	
Scalability kaggle	
Stability - kaggle	
k-nearest-evolution - kaggle	

1.6.4 Google Research, CoLab

Launch each Example in Google Research, CoLab

Need to test examples one by one, then here another option. Use colab offered by google research to test each example individually.

Here is a list of Google CoLab URL to use the algorithm interactively

Dataset	CoLab URL
How to use it - colab	
Chameleon datasets - colab	
2D Shape datasets - colab	
MNIST dataset - colab	
iris dataset - colab	
Get 97% by training MNIST dataset - colab	
Non-groundtruth datasets - colab	
Noise detection - colab	
validation - colab	
How it propagates - colab	
Snapshots of propagation - colab	
Scalability - colab	
Stability vs number of nearest neighbors - colab	
k-nearest-evolution - colab	

1.7 How to cite

If you have used this codebase in a scientific publication and wish to cite it, please use the Journal of Pattern Recognition article

```
Mohamed Abbas McInnes, Adel El-Zoghaby, Amin Ahoukry, *DenMune: Density peak based␣
↪clustering using mutual nearest neighbors*
In: Journal of Pattern Recognition, Elsevier, volume 109, number 107589.
January 2021
```

```
@article{ABBAS2021107589,
title = {DenMune: Density peak based clustering using mutual nearest neighbors},
journal = {Pattern Recognition},
volume = {109},
pages = {107589},
year = {2021},
issn = {0031-3203},
doi = {https://doi.org/10.1016/j.patcog.2020.107589},
url = {https://www.sciencedirect.com/science/article/pii/S0031320320303927},
author = {Mohamed Abbas and Adel El-Zoghabi and Amin Shoukry},
keywords = {Clustering, Mutual neighbors, Dimensionality reduction, Arbitrary shapes,␣
↪Pattern recognition, Nearest neighbors, Density peak},
abstract = {Many clustering algorithms fail when clusters are of arbitrary shapes, of␣
↪varying densities, or the data classes are unbalanced and close to each other, even in␣
↪two dimensions. A novel clustering algorithm “DenMune” is presented to meet this␣
↪challenge. It is based on identifying dense regions using mutual nearest neighborhoods␣
↪of size K, where K is the only parameter required from the user, besides obeying the␣
↪mutual nearest neighbor consistency principle. The algorithm is stable for a wide␣
↪range of values of K. Moreover, it is able to automatically detect and remove noise␣
↪from the clustering process as well as detecting the target clusters. It produces␣
↪robust results on various low and high dimensional datasets relative to several known␣
↪state of the art clustering algorithms.}
}
```

1.7.1 Licensing

The DenMune algorithm is 3-clause BSD licensed. Enjoy.

1.7.2 Task List

- Update Github with the DenMune sourcode
- create repo2docker repository
- Create pip Package
- create CoLab shared examples
- create documentation
- create Kaggle shared examples
- create conda package

CHAPTER

TWO

EXAMPLES

2.1 Iris Dataset

```
import pandas as pd
import numpy as np
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 57, done.[K
remote: Counting objects: 100% (57/57), done.[K
remote: Compressing objects: 100% (46/46), done.[K
remote: Total 57 (delta 9), reused 54 (delta 9), pack-reused 0[K
Unpacking objects: 100% (57/57), done.
```

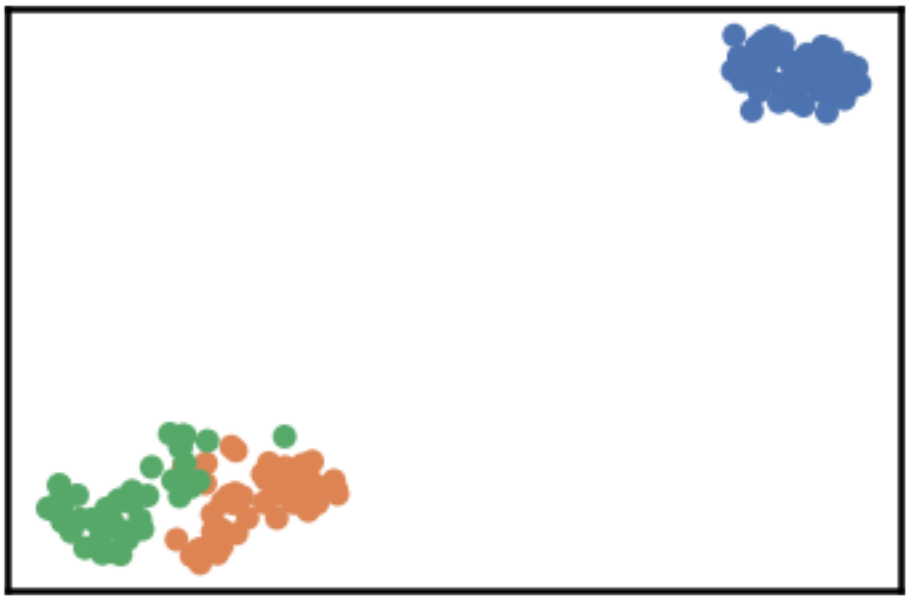
```
data_path = 'datasets/denmune/uci/'
dataset='iris'
data_file = data_path + dataset + '.csv'

X_train = pd.read_csv(data_file, sep=',', header=None)
y_train = X_train.iloc[:, -1]
X_train = X_train.drop(X_train.columns[-1], axis=1)

knn = 11 # k-nearest neighbor, the only parameter required by the algorithm
```

(continues on next page)

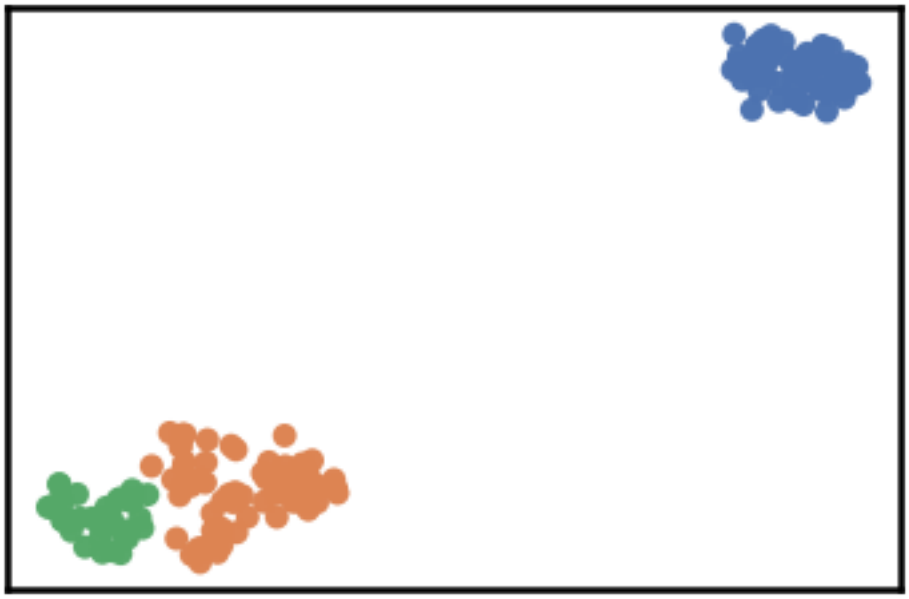
(continued from previous page)

```
dm = DenMune(train_data=X_train,
             train_truth=y_train,
             k_nearest=knn,
             rgn_tsne=False)

labels, validity = dm.fit_predict(show_noise=True, show_analyzer=True)
```

```
Plotting dataset Groundtruth
```



```
Plotting train data
```



```
Validating train data
├── exec_time
    ├── DenMune: 0.019
    ├── NGT: 0.002
    └── t_SNE: 0.85
├── n_clusters
    ├── actual: 3
    └── detected: 3
├── n_points
    ├── dim: 4
    ├── noise
        ├── type-1: 0
        └── type-2: 0
    ├── plot_size: 150
    ├── size: 150
    ├── strong: 84
    └── weak
        ├── all: 66
        ├── failed to merge: 0
        └── succeeded to merge: 66
└── validity
    └── train
        ├── ACC: 135
        ├── AMI: 0.795
        ├── ARI: 0.746
        ├── F1: 0.898
        ├── NMI: 0.798
        ├── completeness: 0.809
        └── homogeneity: 0.787
```

2.2 Chameleon Dataset

```
import pandas as pd
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 52, done.[K
remote: Counting objects: 100% (52/52), done.[K
remote: Compressing objects: 100% (43/43), done.[K
remote: Total 52 (delta 8), reused 49 (delta 8), pack-reused 0[K
Unpacking objects: 100% (52/52), done.
```

```
data_path = 'datasets/denmune/chameleon/'

#@title  { run: "auto", vertical-output: true, form-width: "50%" }
chameleon_dataset = "t7.10k" #@param ["t4.8k", "t5.8k", "t7.10k", "t8.8k"]
show_noize_checkbox = True #@param {type:"boolean"}
show_analyzer_checkbox = True #@param {type:"boolean"}
k_nearest_slider = 39 #@param {type:"slider", min:1, max:100, step:1}

# train file
data_file = data_path + chameleon_dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)

dm = DenMune(train_data=X_train, k_nearest=k_nearest_slider, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=show_noize_checkbox,
                                  show_analyzer=show_analyzer_checkbox)
```

```
Plotting train data
```

```
Validating train data
├── exec_time
│   ├── DenMune: 9.7
│   ├── NGT: 0.612
│   └── t_SNE: 0
├── n_clusters
```

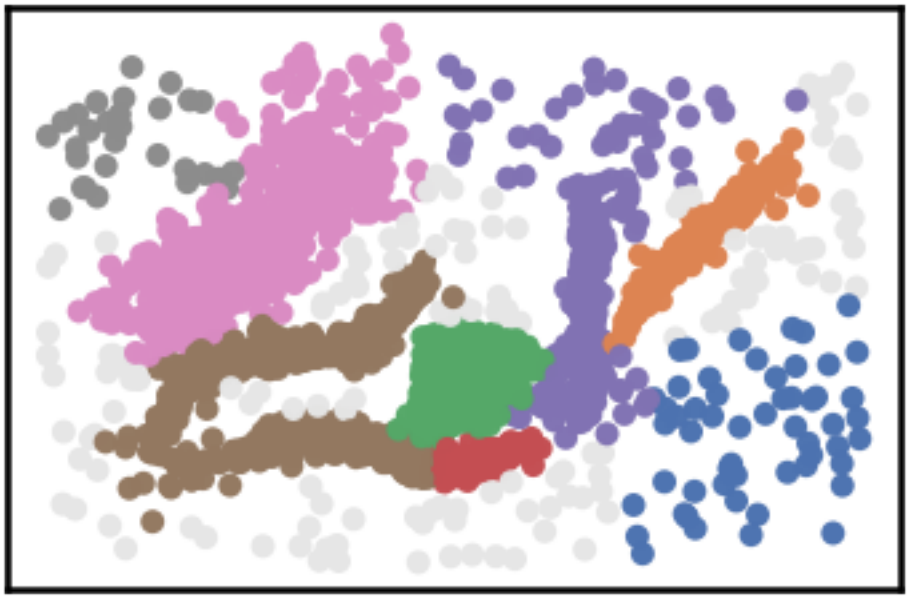
(continues on next page)

(continued from previous page)

```
    ├── actual: 0
    └── detected: 9
└── n_points
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 516
    ├── plot_size: 10000
    ├── size: 10000
    ├── strong: 5860
    └── weak
        ├── all: 4140
        ├── failed to merge: 516
        └── succeeded to merge: 3624
```

```
data_path = 'datasets/denmune/chameleon/'

#@title  { run: "auto", vertical-output: true, form-width: "50%" }
chameleon_dataset = "clusterable" #@param ["t4.8k", "t5.8k", "t7.10k", "t8.8k",
↪"clusterable"]
show_noize_checkbox = True #@param {type:"boolean"}
show_analyzer_checkbox = True #@param {type:"boolean"}
k_nearest_slider = 24 #@param {type:"slider", min:1, max:100, step:1}

# train file
data_file = data_path + chameleon_dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)

dm = DenMune(train_data=X_train, k_nearest=k_nearest_slider, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=show_noize_checkbox,
                                  show_analyzer=show_analyzer_checkbox)
```

```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 1.393
│   ├── NGT: 0.121
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 0
│   └── detected: 8
└── n_points
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 141
    ├── plot_size: 2309
    ├── size: 2309
    ├── strong: 1352
    └── weak
        ├── all: 957
        ├── failed to merge: 141
        └── succeeded to merge: 816
```

2.3 2D Shapes Datasets

```
import pandas as pd
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

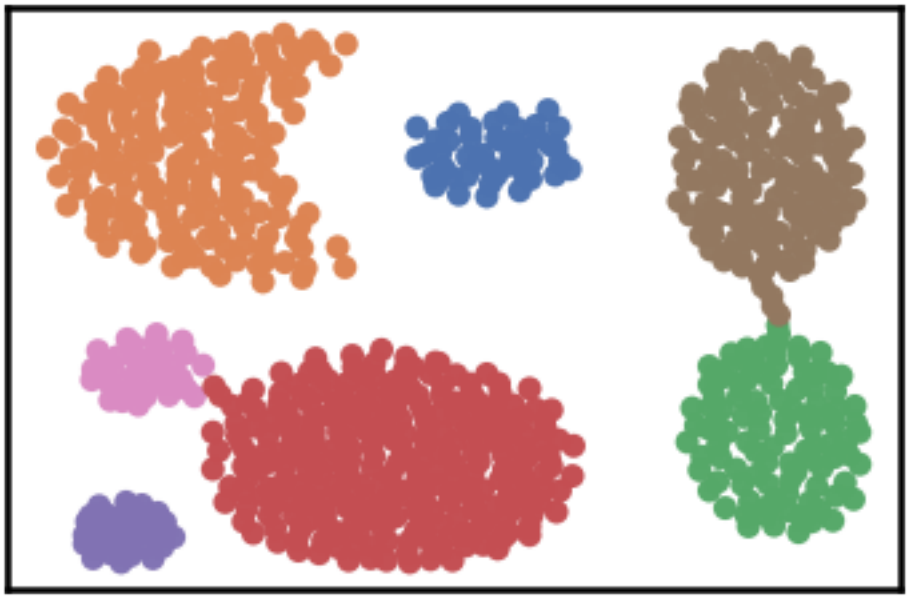
```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

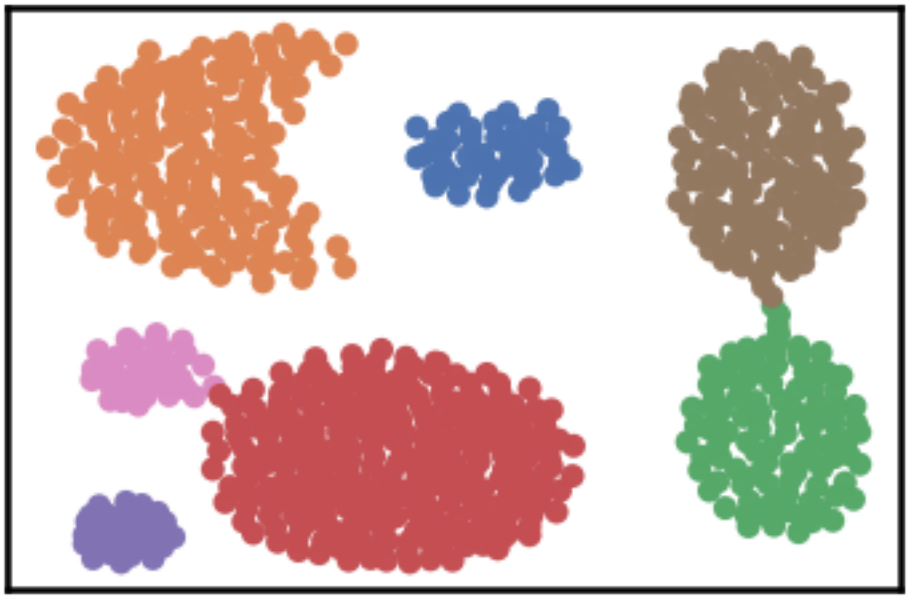
```
Cloning into 'datasets'...
remote: Enumerating objects: 52, done.[K
remote: Counting objects: 100% (52/52), done.[K
remote: Compressing objects: 100% (43/43), done.[K
remote: Total 52 (delta 8), reused 49 (delta 8), pack-reused 0[K
Unpacking objects: 100% (52/52), done.
Checking out files: 100% (20/20), done.
```

```
data_path = 'datasets/denmune/shapes/'
datasets = {"aggregation": 6, "jain": 15, "flame": 8, "compound": 13, "varydensity": 23,
            "unbalance": 8, "spiral": 6, "pathbased": 6, "mouse": 11}

for dataset in datasets:


  data_file = data_path + dataset + '.csv'
  X_train = pd.read_csv(data_file, sep=',', header=None)
  y_train = X_train.iloc[:, -1]
  X_train = X_train.drop(X_train.columns[-1], axis=1)

  print ("Dataset:", dataset)
  dm = DenMune(train_data=X_train,
               train_truth=y_train,
               k_nearest=datasets[dataset],
               rgn_tsne=False)

  labels, validity = dm.fit_predict(show_noise=True, show_analyzer=True)
```

```
Dataset: aggregation
Plotting dataset Groundtruth
```



```
Plotting train data
```

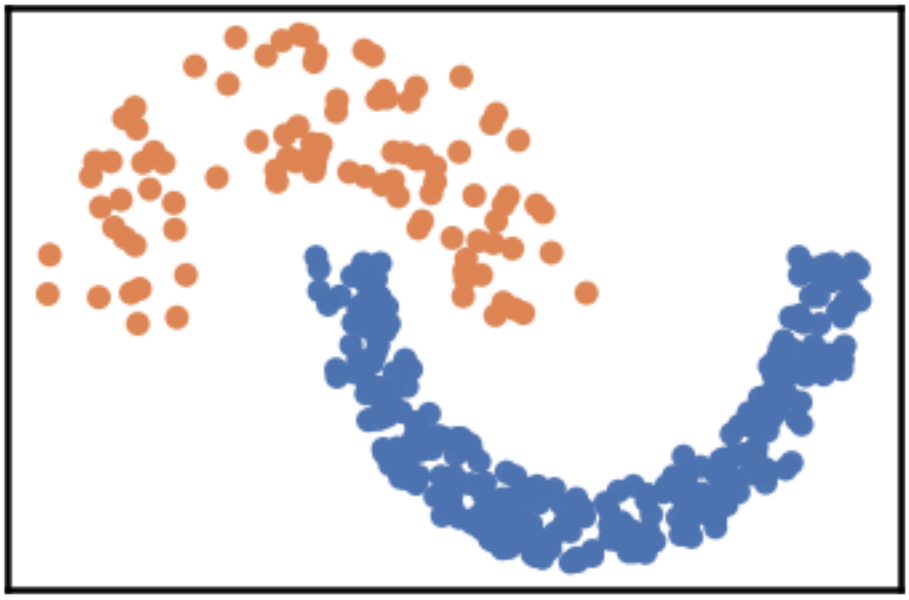


```
Validating train data
├── exec_time
│   ├── DenMune: 0.17
│   ├── NGT: 0.016
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 7
│   └── detected: 7
├── n_points
```

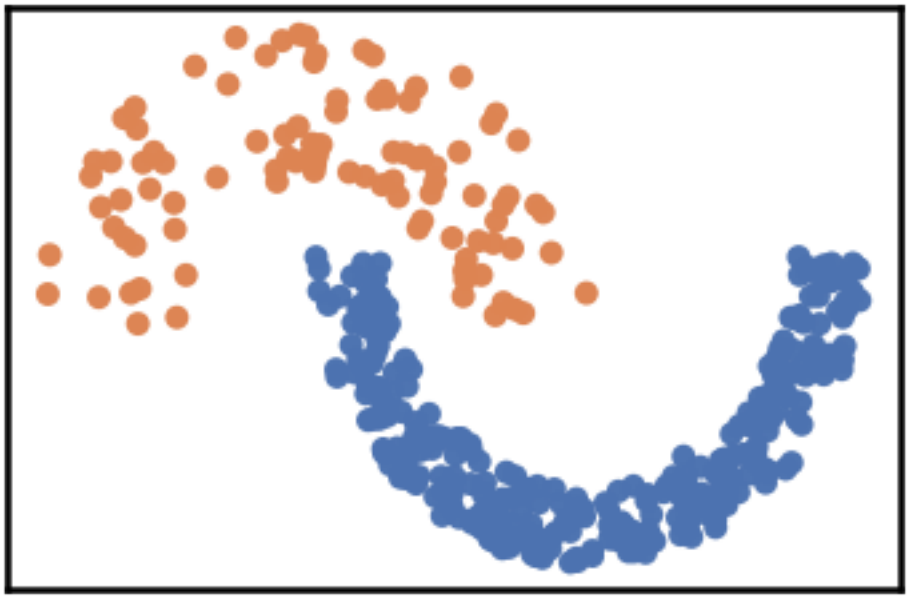
(continues on next page)

(continued from previous page)

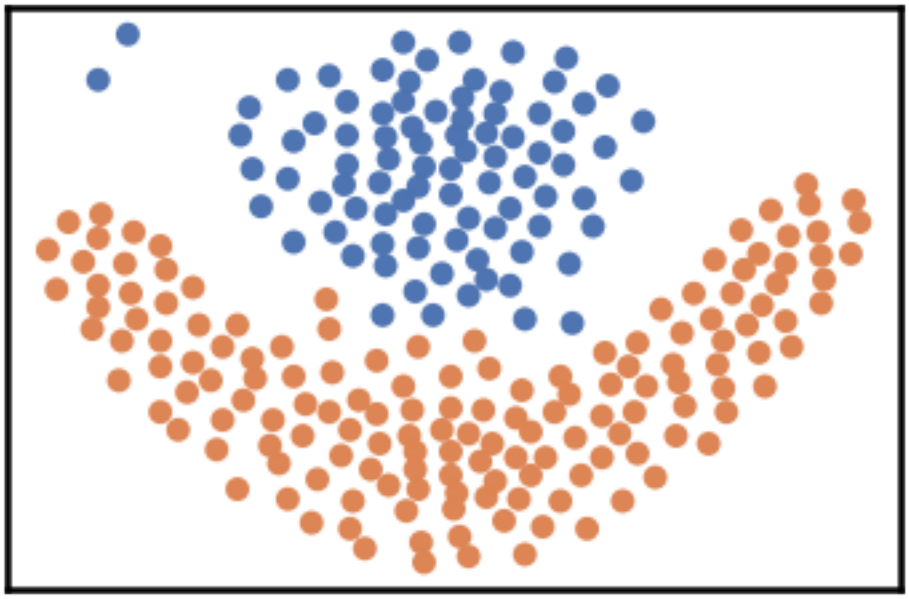
```
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 0
    ├── plot_size: 788
    ├── size: 788
    ├── strong: 492
    └── weak
        ├── all: 296
        ├── failed to merge: 0
        └── succeeded to merge: 296
└── validity
    └── train
        ├── ACC: 785
        ├── AMI: 0.988
        ├── ARI: 0.993
        ├── F1: 0.996
        ├── NMI: 0.988
        ├── completeness: 0.987
        └── homogeneity: 0.989

Dataset: jain
Plotting dataset Groundtruth
```



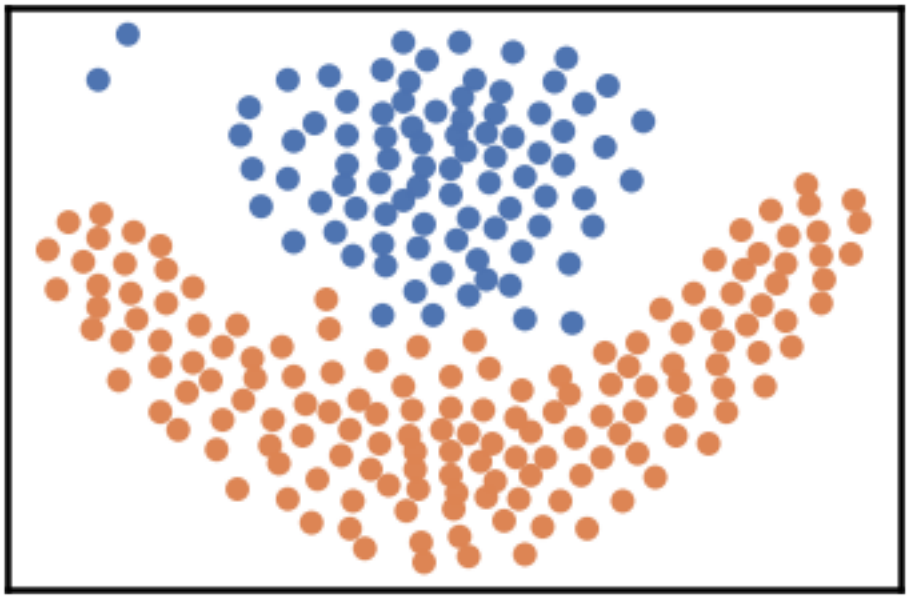
```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 0.097
│   ├── NGT: 0.01
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 2
│   └── detected: 2
├── n_points
│   ├── dim: 2
│   ├── noise
│   │   ├── type-1: 0
│   │   └── type-2: 0
│   ├── plot_size: 373
│   ├── size: 373
│   ├── strong: 198
│   └── weak
│       ├── all: 175
│       ├── failed to merge: 0
│       └── succeeded to merge: 175
└── validity
    └── train
        ├── ACC: 373
        ├── AMI: 1.0
        ├── ARI: 1.0
        ├── F1: 1.0
        ├── NMI: 1.0
        ├── completeness: 1.0
        └── homogeneity: 1.0

Dataset: flame
Plotting dataset Groundtruth
```



```
Plotting train data
```

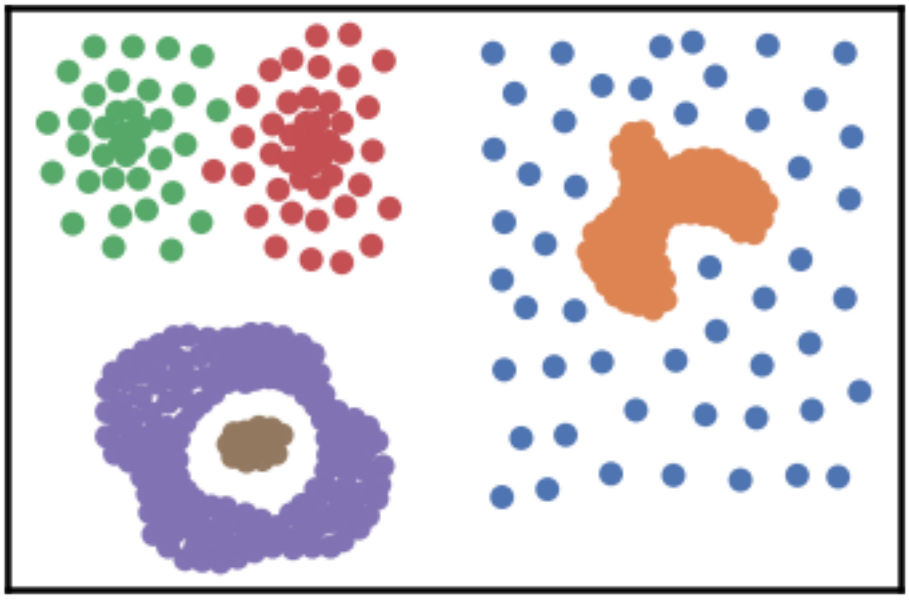


```
Validating train data
├── exec_time
    ├── DenMune: 0.059
    ├── NGT: 0.01
    └── t_SNE: 0
├── n_clusters
    ├── actual: 2
    └── detected: 2
├── n_points
```

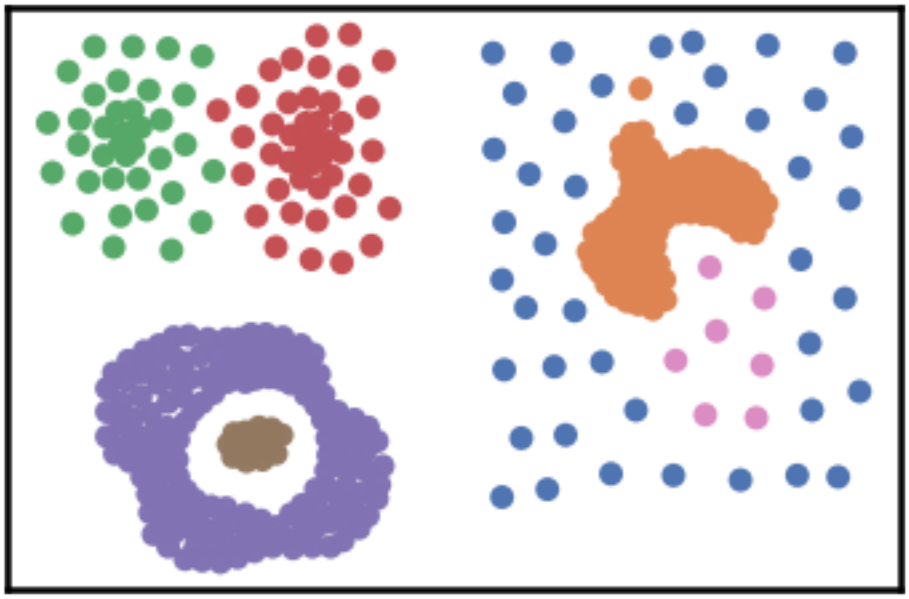
(continues on next page)

(continued from previous page)

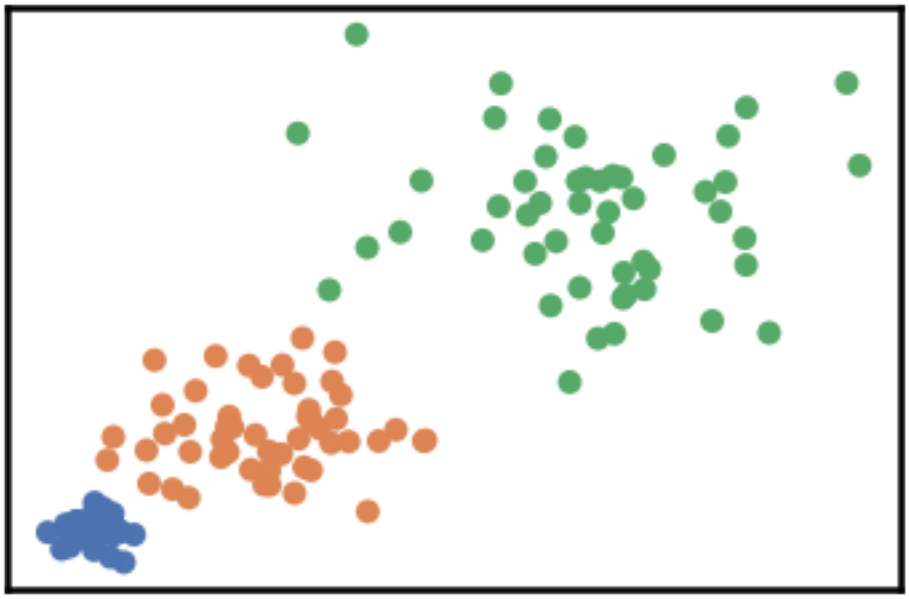
```
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 0
    ├── plot_size: 240
    ├── size: 240
    ├── strong: 150
    └── weak
        ├── all: 90
        ├── failed to merge: 0
        └── succeeded to merge: 90
└── validity
    └── train
        ├── ACC: 240
        ├── AMI: 1.0
        ├── ARI: 1.0
        ├── F1: 1.0
        ├── NMI: 1.0
        ├── completeness: 1.0
        └── homogeneity: 1.0

Dataset: compound
Plotting dataset Groundtruth
```



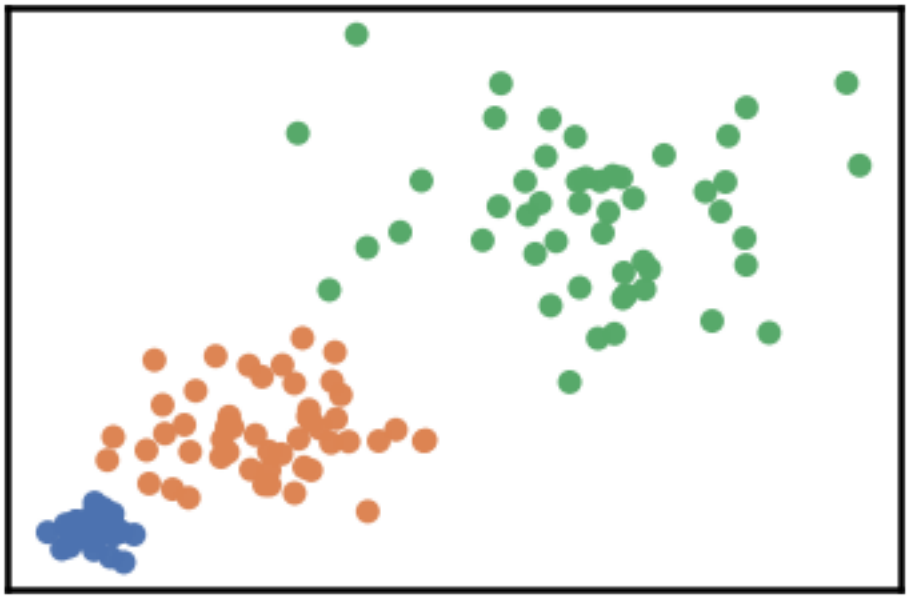
```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 0.077
│   ├── NGT: 0.013
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 6
│   └── detected: 7
├── n_points
│   ├── dim: 2
│   ├── noise
│   │   ├── type-1: 0
│   │   └── type-2: 0
│   ├── plot_size: 399
│   ├── size: 399
│   ├── strong: 218
│   └── weak
│       ├── all: 181
│       ├── failed to merge: 0
│       └── succeeded to merge: 181
└── validity
    └── train
        ├── ACC: 389
        ├── AMI: 0.96
        ├── ARI: 0.98
        ├── F1: 0.983
        ├── NMI: 0.961
        ├── completeness: 0.947
        └── homogeneity: 0.976

Dataset: varydensity
Plotting dataset Groundtruth
```



```
Plotting train data
```

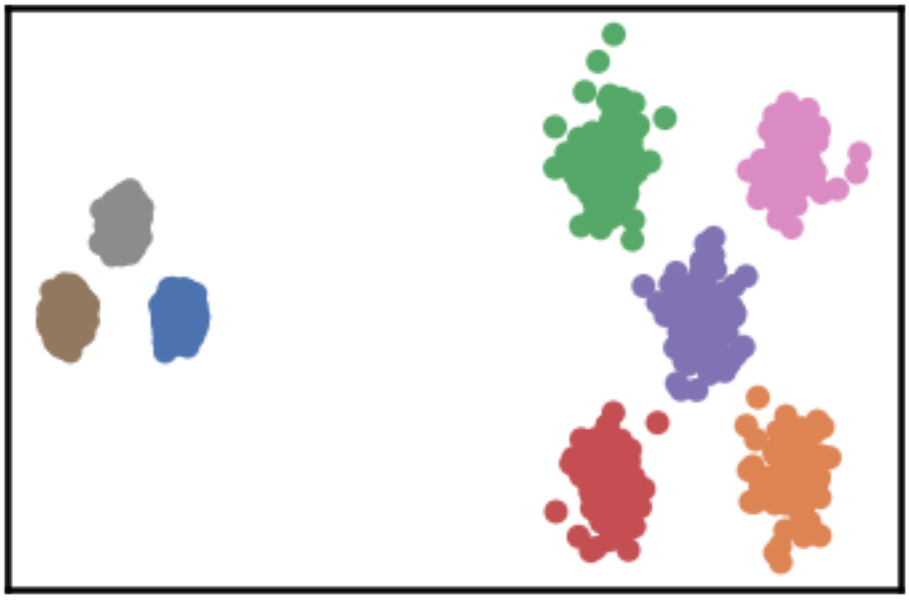


```
Validating train data
├── exec_time
│   ├── DenMune: 0.035
│   ├── NGT: 0.004
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 3
│   └── detected: 3
├── n_points
```

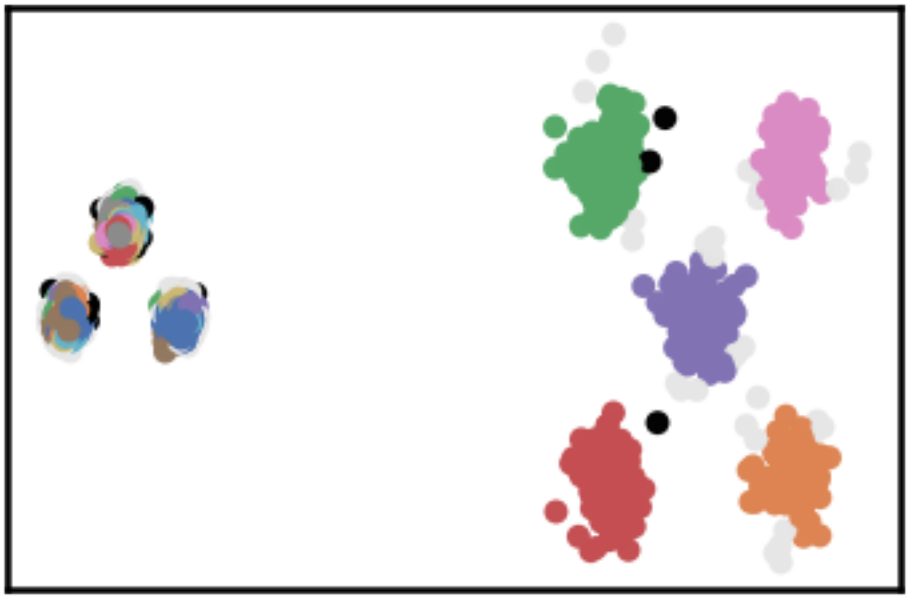
(continues on next page)

(continued from previous page)

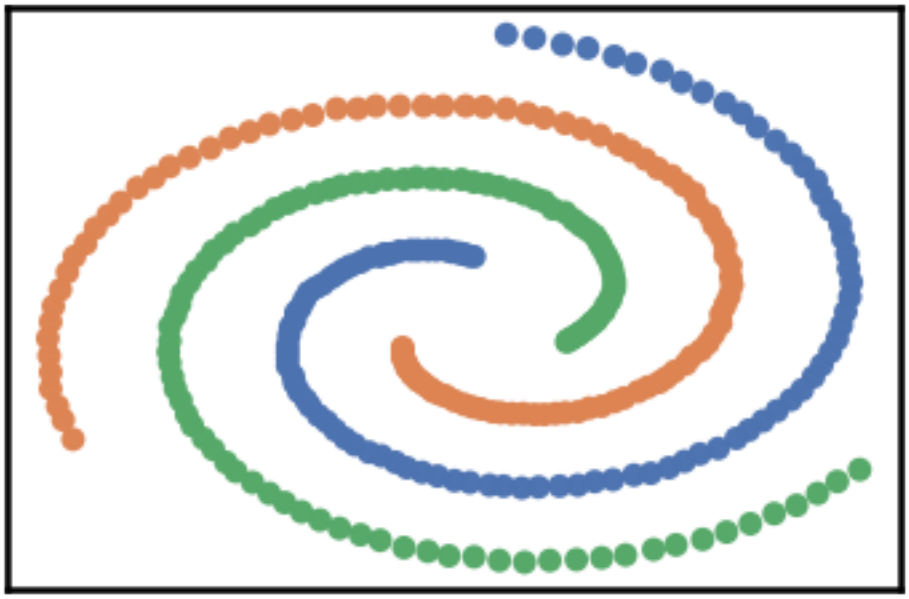
```
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 0
    ├── plot_size: 150
    ├── size: 150
    ├── strong: 76
    └── weak
        ├── all: 74
        ├── failed to merge: 0
        └── succeeded to merge: 74
└── validity
    └── train
        ├── ACC: 150
        ├── AMI: 1.0
        ├── ARI: 1.0
        ├── F1: 1.0
        ├── NMI: 1.0
        ├── completeness: 1.0
        └── homogeneity: 1.0

Dataset: unbalance
Plotting dataset Groundtruth
```



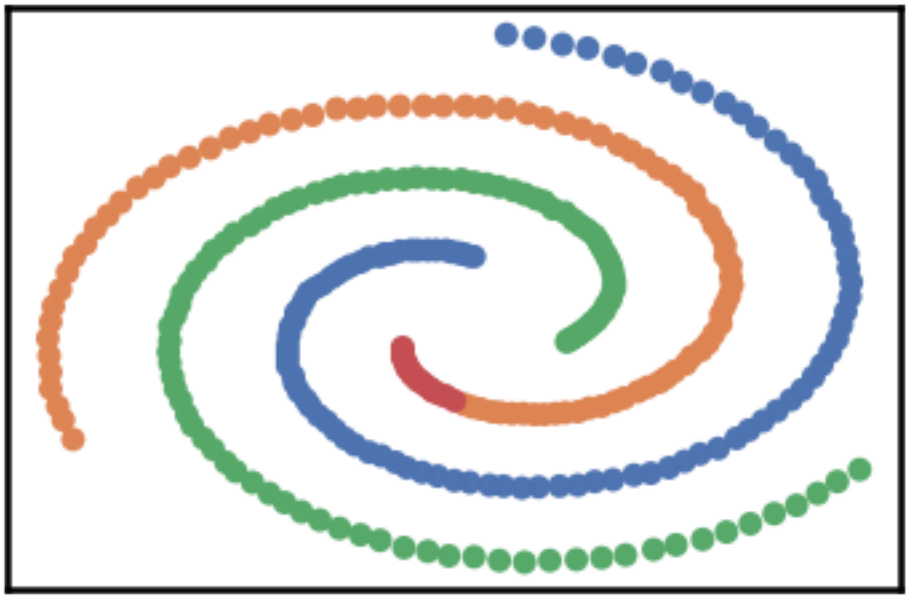
```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 1.028
│   ├── NGT: 0.123
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 8
│   └── detected: 91
├── n_points
│   ├── dim: 2
│   ├── noise
│   │   ├── type-1: 21
│   │   └── type-2: 127
│   ├── plot_size: 6500
│   ├── size: 6500
│   ├── strong: 3910
│   └── weak
│       ├── all: 2590
│       ├── failed to merge: 127
│       └── succeeded to merge: 2463
└── validity
    └── train
        ├── ACC: 2562
        ├── AMI: 0.534
        ├── ARI: 0.272
        ├── F1: 0.519
        ├── NMI: 0.542
        ├── completeness: 0.376
        └── homogeneity: 0.974

Dataset: spiral
Plotting dataset Groundtruth
```



```
Plotting train data
```

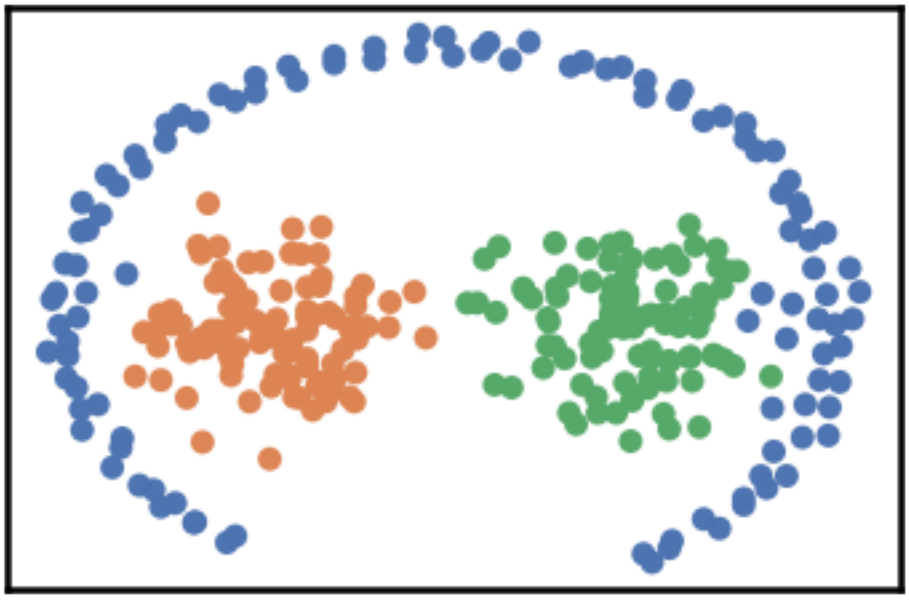


```
Validating train data
├── exec_time
│   ├── DenMune: 0.036
│   ├── NGT: 0.003
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 3
│   └── detected: 4
├── n_points
```

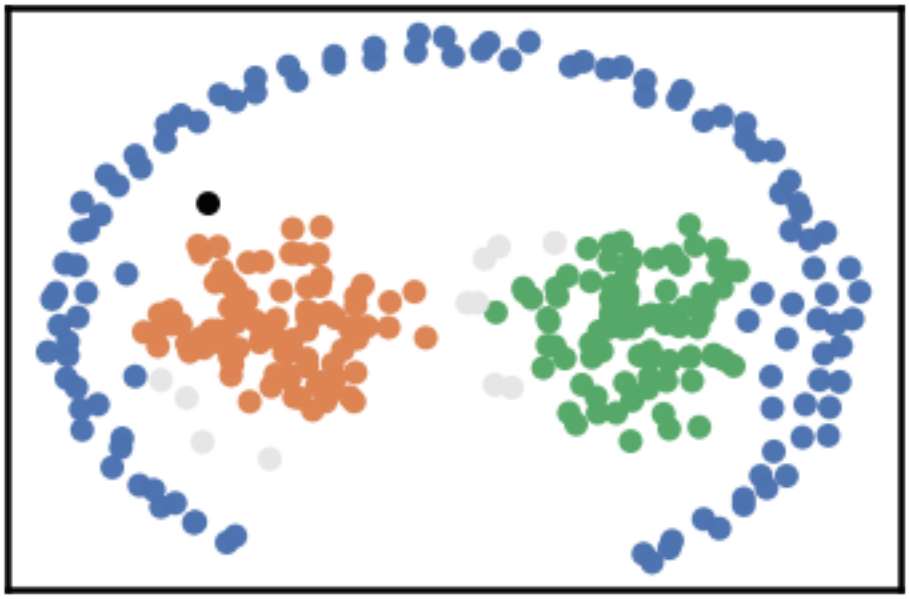
(continues on next page)

(continued from previous page)

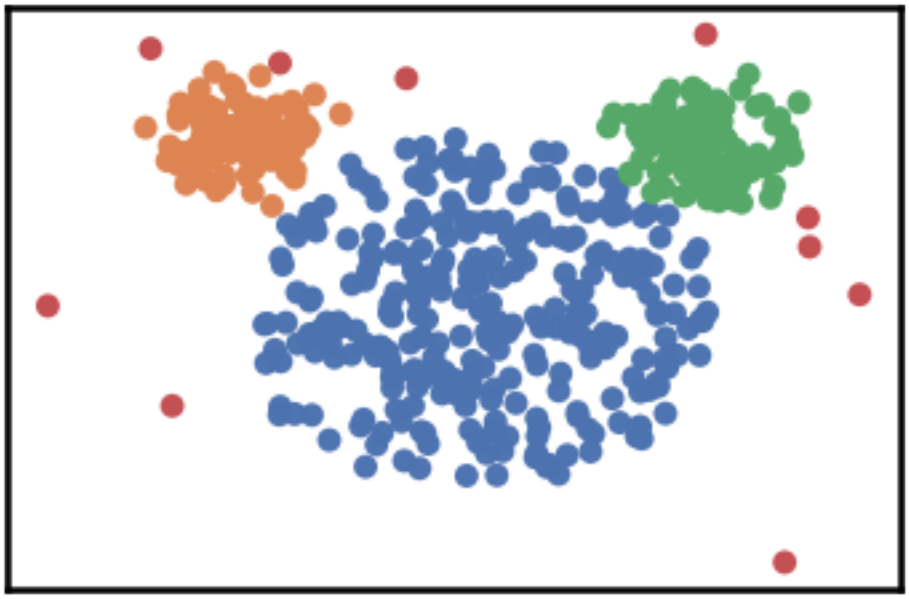
```
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 0
    ├── plot_size: 312
    ├── size: 312
    ├── strong: 285
    └── weak
        ├── all: 27
        ├── failed to merge: 0
        └── succeeded to merge: 27
└── validity
    └── train
        ├── ACC: 293
        ├── AMI: 0.932
        ├── ARI: 0.922
        ├── F1: 0.967
        ├── NMI: 0.932
        ├── completeness: 0.873
        └── homogeneity: 1.0

Dataset: pathbased
Plotting dataset Groundtruth
```



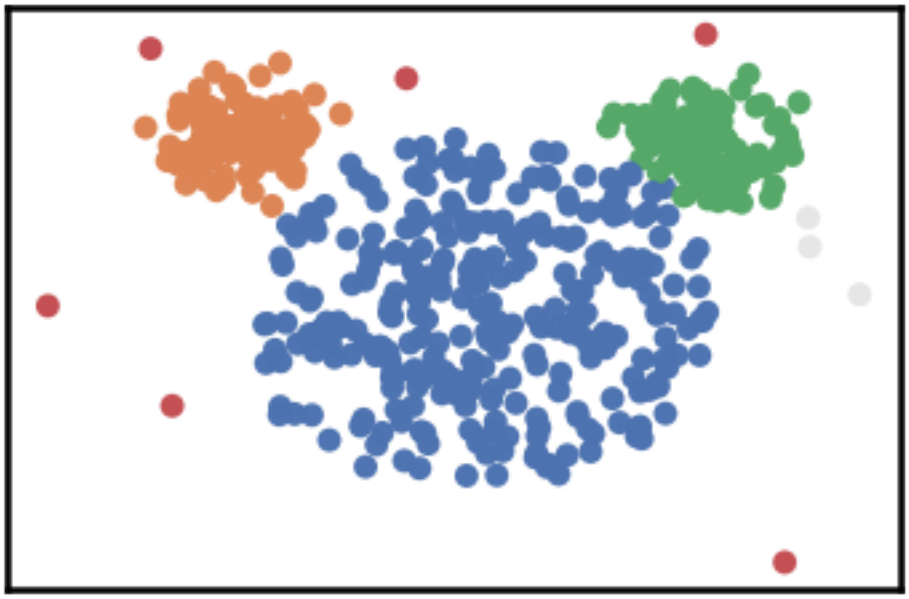
```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 0.072
│   ├── NGT: 0.007
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 3
│   └── detected: 3
├── n_points
│   ├── dim: 2
│   ├── noise
│   │   ├── type-1: 1
│   │   └── type-2: 11
│   ├── plot_size: 300
│   ├── size: 300
│   ├── strong: 198
│   └── weak
│       ├── all: 102
│       ├── failed to merge: 11
│       └── succeeded to merge: 91
└── validity
    └── train
        ├── ACC: 286
        ├── AMI: 0.889
        ├── ARI: 0.924
        ├── F1: 0.972
        ├── NMI: 0.89
        ├── completeness: 0.842
        └── homogeneity: 0.943

Dataset: mouse
Plotting dataset Groundtruth
```



```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 0.082
│   ├── NGT: 0.015
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 4
│   └── detected: 4
├── n_points
```

(continues on next page)

(continued from previous page)

```
    ├── dim: 2
    ├── noise
    │   ├── type-1: 0
    │   └── type-2: 3
    ├── plot_size: 500
    ├── size: 500
    ├── strong: 302
    └── weak
        ├── all: 198
        ├── failed to merge: 3
        └── succeeded to merge: 195
└── validity
    └── train
        ├── ACC: 492
        ├── AMI: 0.949
        ├── ARI: 0.972
        ├── F1: 0.986
        ├── NMI: 0.949
        ├── completeness: 0.95
        └── homogeneity: 0.948
```

2.4 MNIST Dataset

```
import pandas as pd
import numpy as np
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

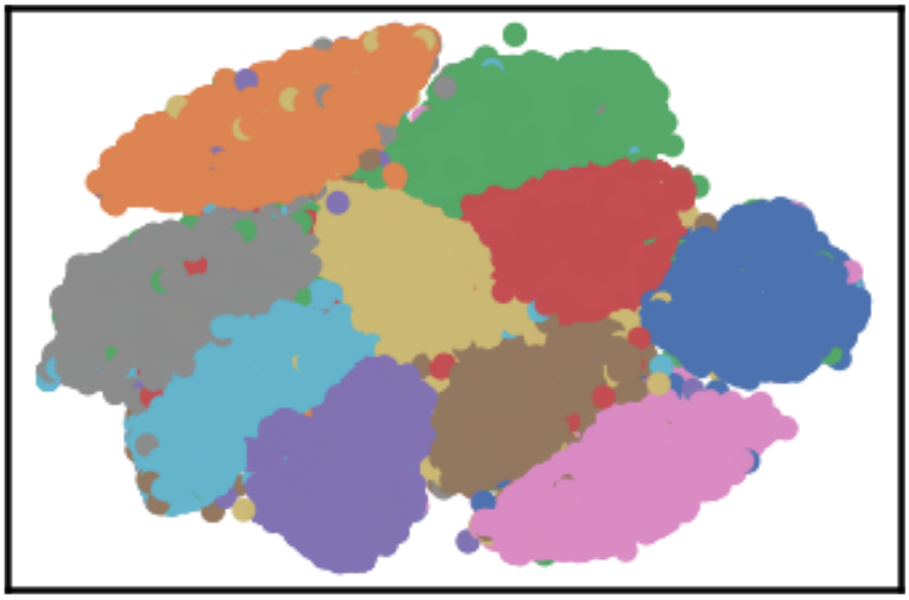
```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 52, done.[K
remote: Counting objects: 100% (52/52), done.[K
remote: Compressing objects: 100% (43/43), done.[K
remote: Total 52 (delta 8), reused 49 (delta 8), pack-reused 0[K
Unpacking objects: 100% (52/52), done.
```

```
data_path = 'datasets/denmune/mnist/'
file_2d = data_path + 'mnist-2d.csv'

X_train = pd.read_csv(data_path + 'train.csv', sep=',')
X_test = pd.read_csv(data_path + 'test.csv',  sep=',')
y_train = X_train['label']
X_train = X_train.drop(['label'], axis=1)

dm = DenMune(train_data=X_train,
             train_truth=y_train,
             test_data=X_test,
             k_nearest=93,
             file_2d=file_2d,
             rgn_tsne=False)

labels, validity = dm.fit_predict(show_noise=True, show_analyzer=True)
```

```
Plotting dataset Groundtruth
```



```
Plotting train data
```

```
Validating train data
├── exec_time
│   ├── DenMune: 340.29
│   ├── NGT: 15.154
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 10
│   └── detected: 10
├── n_points
│   ├── dim: 784
│   ├── noise
│   │   ├── type-1: 2
│   │   └── type-2: 0
│   ├── plot_size: 42000
│   ├── size: 70000
│   ├── strong: 38267
│   └── weak
│       ├── all: 31733
│       ├── failed to merge: 0
│       └── succeeded to merge: 31733
└── validity
    └── train
        ├── ACC: 40564
        ├── AMI: 0.913
        ├── ARI: 0.926
        ├── F1: 0.966
        ├── NMI: 0.913
        ├── completeness: 0.913
        └── homogeneity: 0.913

Plotting test data
```

```
# prepare our output to be submitted to the dataset kaggle competition
ImageID = np.arange(len(X_test))+1
Out = pd.DataFrame([ImageID,labels['test']]).T
Out.to_csv('submission.csv', header =  ['ImageId', 'Label' ], index = None)
```

CHAPTER

THREE

CHARACTERISTICS

3.1 Noise Detection

```
import pandas as pd
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 63, done.[K
remote: Counting objects: 100% (63/63), done.[K
remote: Compressing objects: 100% (52/52), done.[K
remote: Total 63 (delta 10), reused 59 (delta 9), pack-reused 0[K
Unpacking objects: 100% (63/63), done.
Checking out files: 100% (23/23), done.
```

DenMune detects noise and outlier automatically, no need to any further work from your side. * It plots pre-identified noise in black * It plots post-identified noise in light grey

You can set show_noise parameter to False to show clean data as identified by the algorithm

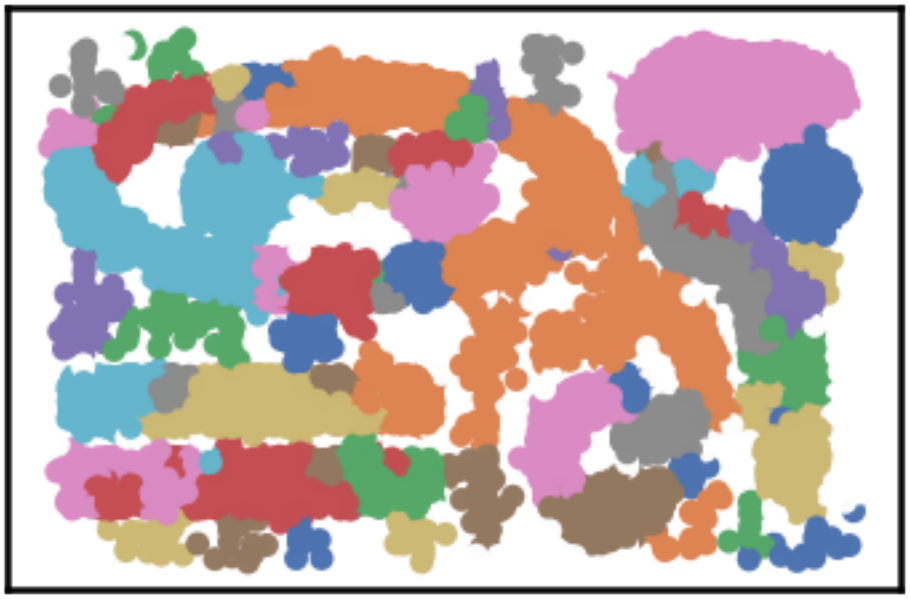
```
data_path = 'datasets/denmune/chameleon/'

chameleon_dataset = "t7.10k" #["t4.8k", "t5.8k", "t7.10k", "t8.8k"]
```

(continues on next page)

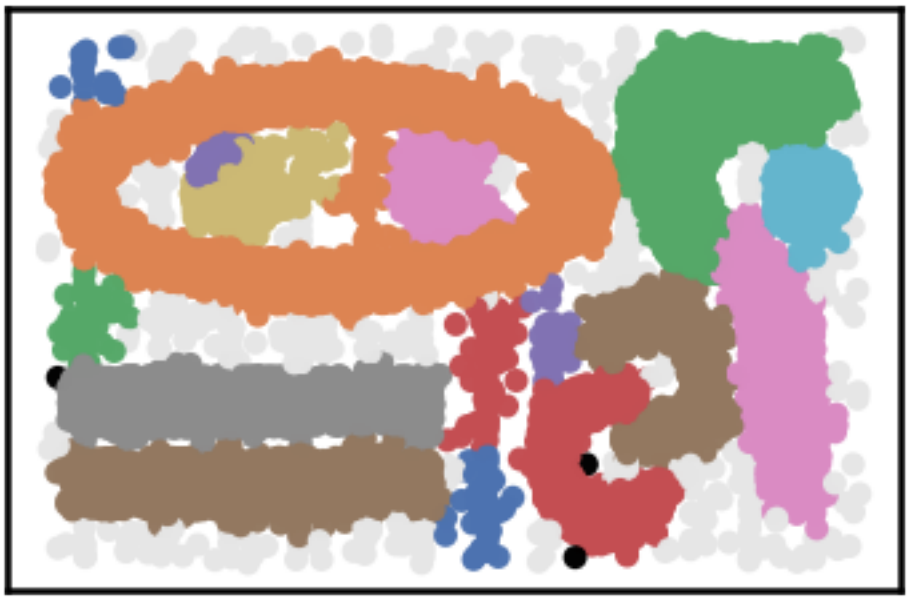
(continued from previous page)

```
knn = 10
# train file
data_file = data_path + chameleon_dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)

dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=True, show_analyzer=False)
```

```
# let us show clean data by removing noise

m = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=False, show_analyzer=False)
```



```
knn = 20
# train file
data_file = data_path + chameleon_dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)

dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=True, show_analyzer=False)
```

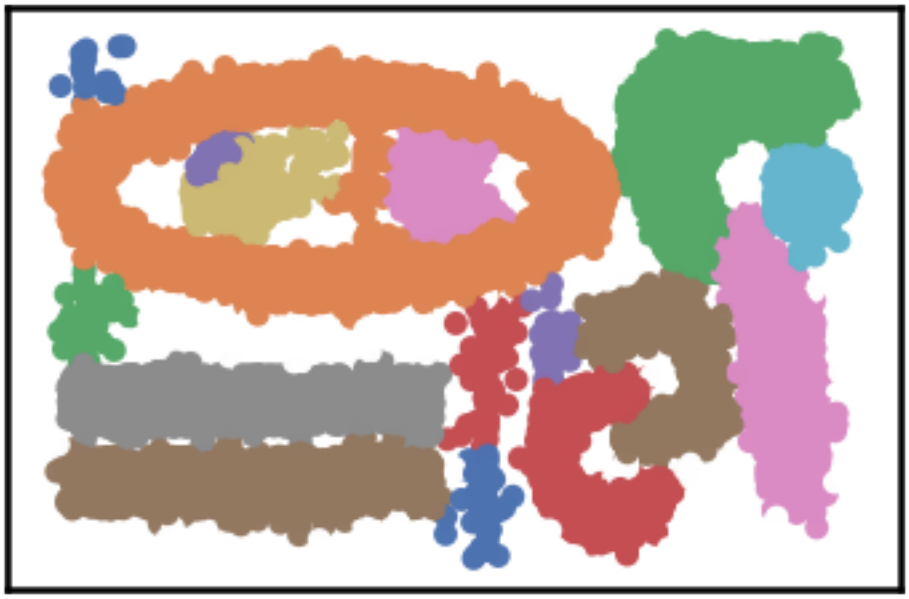


```
# let us show clean data by removing noise

m = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False)
```

(continues on next page)

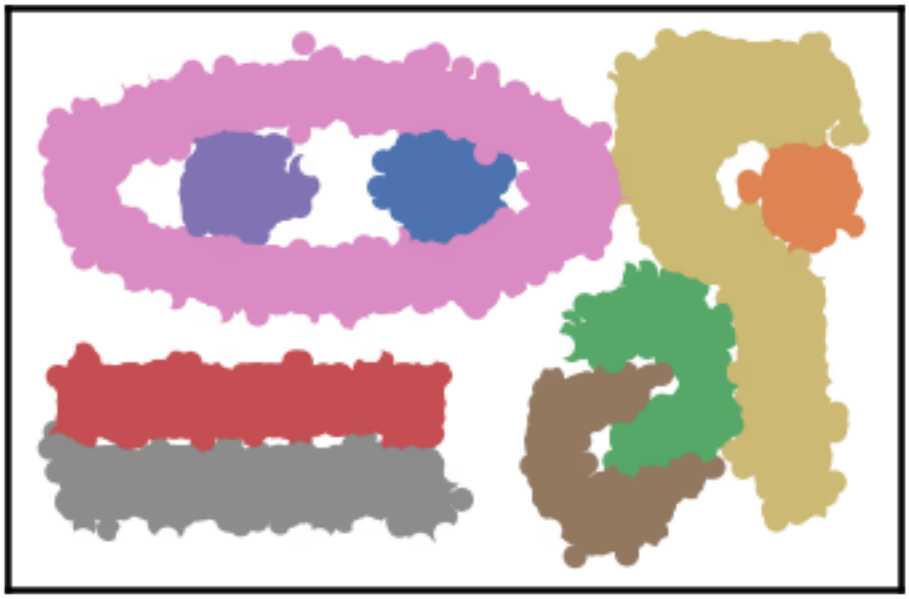
(continued from previous page)

```
labels, validity = dm.fit_predict(show_noise=False, show_analyzer=False)
```



```
knn = 39
# train file
data_file = data_path + chameleon_dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)

dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=True, show_analyzer=False)
```

```
# let us show clean data by removing noise

m = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_noise=False, show_analyzer=False)
```



3.2 Clustering Propagation

```
import pandas as pd
import time
import os.path
import glob

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 52, done.[K
```

(continues on next page)

(continued from previous page)

```
remote: Counting objects: 100% (52/52), done.[K
remote: Compressing objects: 100% (43/43), done.[K
remote: Total 52 (delta 8), reused 49 (delta 8), pack-reused 0[K
Unpacking objects: 100% (52/52), done.
```

```
#@title  { run: "auto", vertical-output: true, form-width: "50%" }
dataset = "t7.10k" #@param ["t4.8k", "t5.8k", "t7.10k", "t8.8k"]
show_noize_checkbox = True #@param {type:"boolean"}
data_path = 'datasets/denmune/chameleon/'

# train file
data_file = data_path + dataset +'.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)
```

```
from itertools import chain

# Denmune's Paramaters
knn = 39 # number of k-nearest neighbor, the only parameter required by the algorithm

# create list of differnt snapshots of the propagation
snapshots = chain([0], range(2,5), range(5,50,5), range(50, 100, 10), range(100,500,50),␣
↪range(500,1000, 100), range(1000,3000, 250),range(3000,5500,500))

from IPython.display import clear_output
for snapshot in snapshots:
    print ("itration", snapshot )
    clear_output(wait=True)
    dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False, prop_step=snapshot)
    labels, validity = dm.fit_predict(show_analyzer=False, show_noise=False)
```

```
from PIL import Image

# collect immages for each snapshot automatically by the algorithm in a folder named␣
↪propagation
images = []
prop_folder = 'propagation'
img_files = os.listdir(prop_folder)
img_files = [os.path.join(prop_folder, f) for f in img_files]
sorted_files = sorted (img_files, key=os.path.getmtime)
for filename in sorted_files:
  im = Image.open(filename)
  images.append(im)

# create annimated gif to show evolution of the propagation
images[0].save('propagation.gif', save_all=True, append_images=images[1:],␣
↪optimize=False, duration=800, loop=1)
```

3.3 Clustering Propagation Snapshots

```
import pandas as pd
import time
import os.path
import glob

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 52, done.[K
remote: Counting objects: 100% (52/52), done.[K
remote: Compressing objects: 100% (43/43), done.[K
remote: Total 52 (delta 8), reused 49 (delta 8), pack-reused 0[K
Unpacking objects: 100% (52/52), done.
```

```
#@title  { run: "auto", vertical-output: true, form-width: "50%" }
dataset = "t7.10k" #@param ["t4.8k", "t5.8k", "t7.10k", "t8.8k"]
show_noize_checkbox = True #@param {type:"boolean"}
data_path = 'datasets/denmune/chameleon/'

# train file
data_file = data_path + dataset +'.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)
```

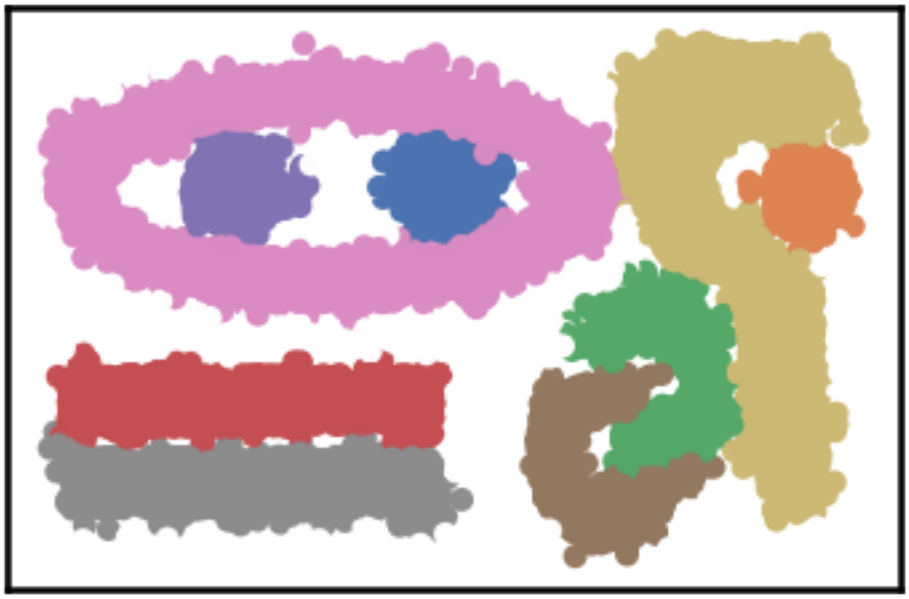
```
from itertools import chain

# Denmune's Paramaters
knn = 39 # number of k-nearest neighbor, the only parameter required by the algorithm

# create list of differnt snapshots of the propagation
snapshots = chain([0], range(2,5), range(5,50,5), range(50, 100, 10), range(100,500,50),␣
↪range(500,1000, 100), range(1000,3000, 250),range(3000,5500,500))

from IPython.display import clear_output
for snapshot in snapshots:
    print ("itration", snapshot )
    #clear_output(wait=True)
```

(continues on next page)

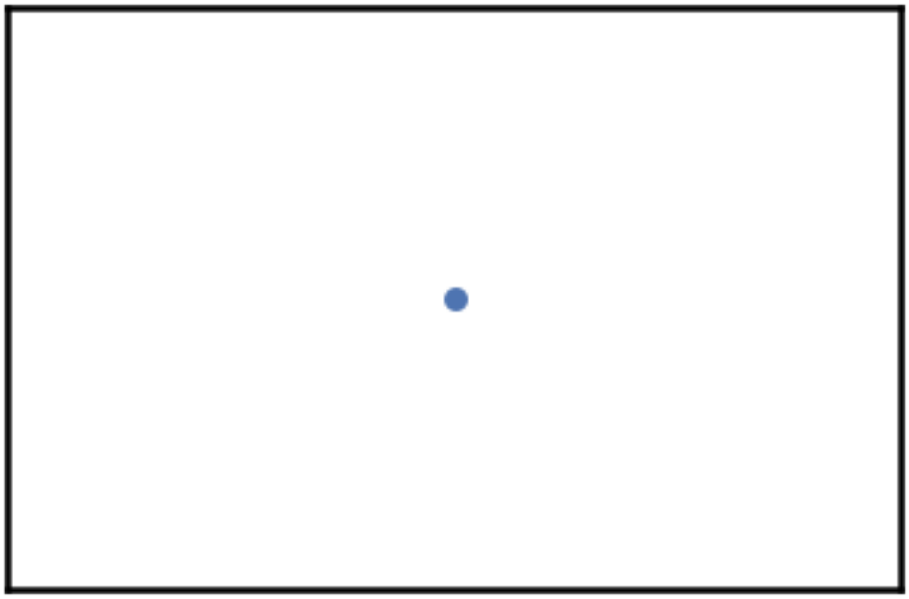
(continued from previous page)

```
    dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False, prop_step=snapshot)
    labels, validity = dm.fit_predict(show_analyzer=False, show_noise=False)
```

```
itration 0
```

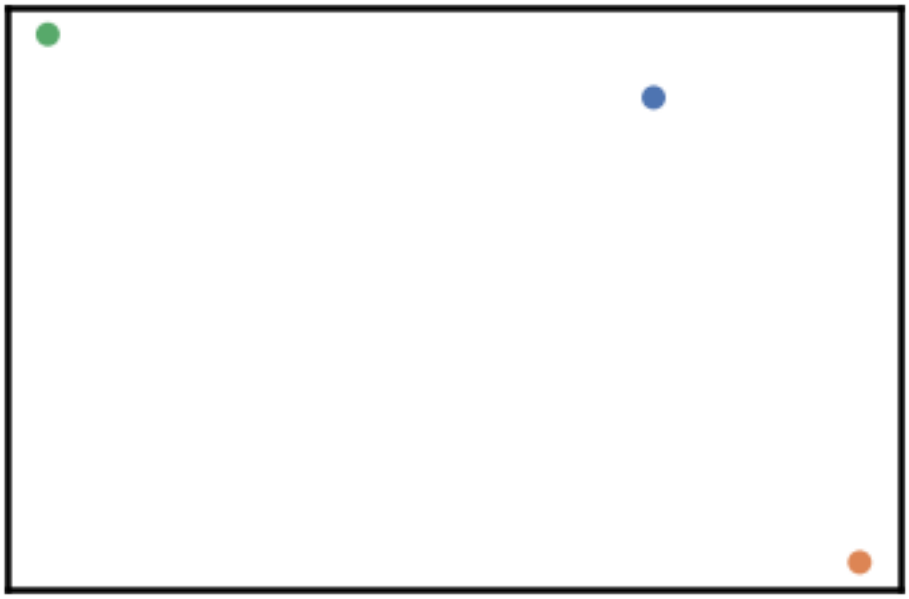


```
itration 2
```

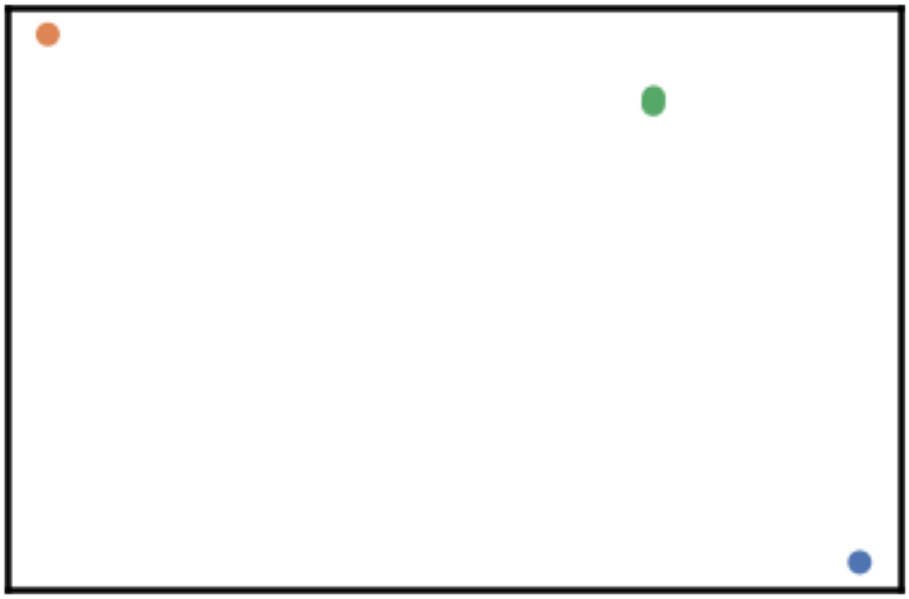


```
itration 3
```

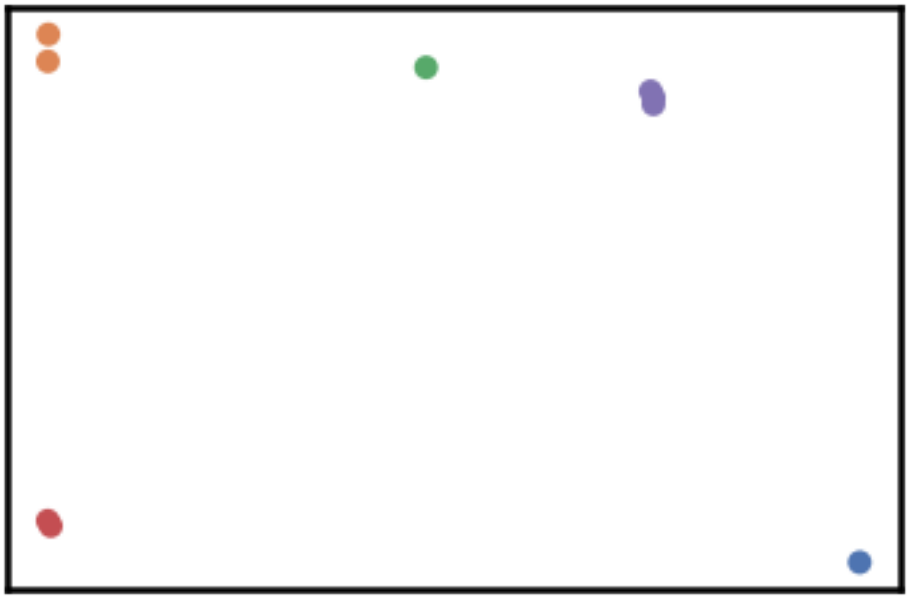
```
itration 4
```



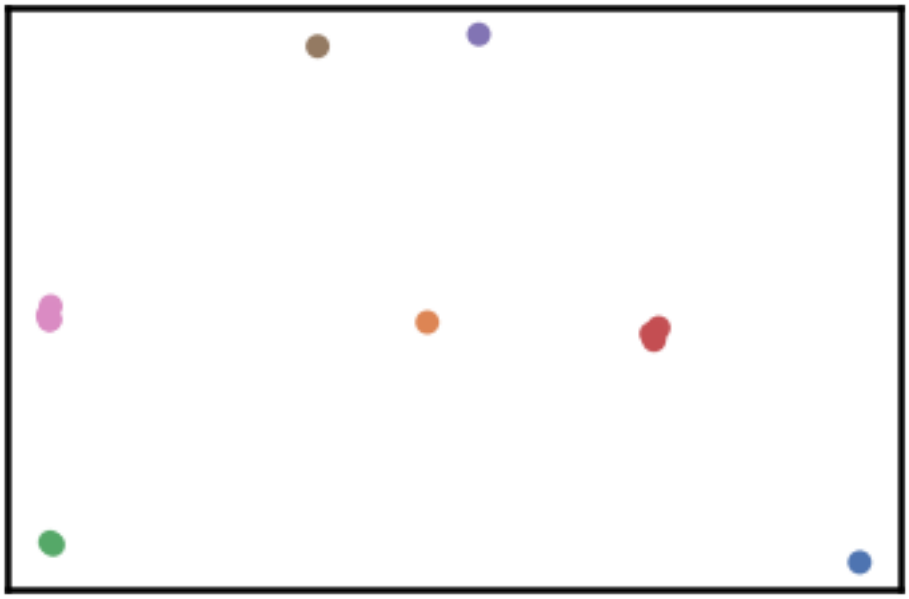
```
itration 5
```



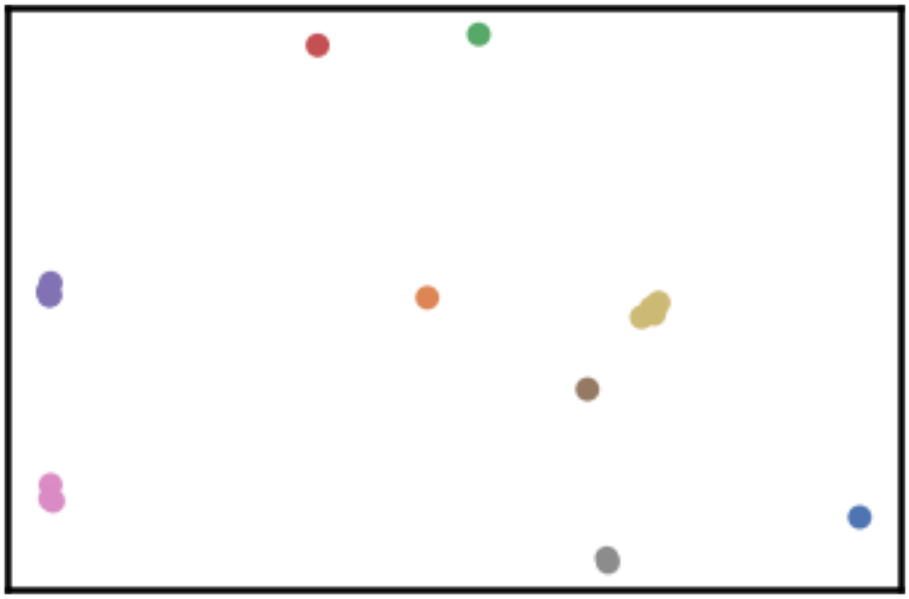
```
itration 10
```



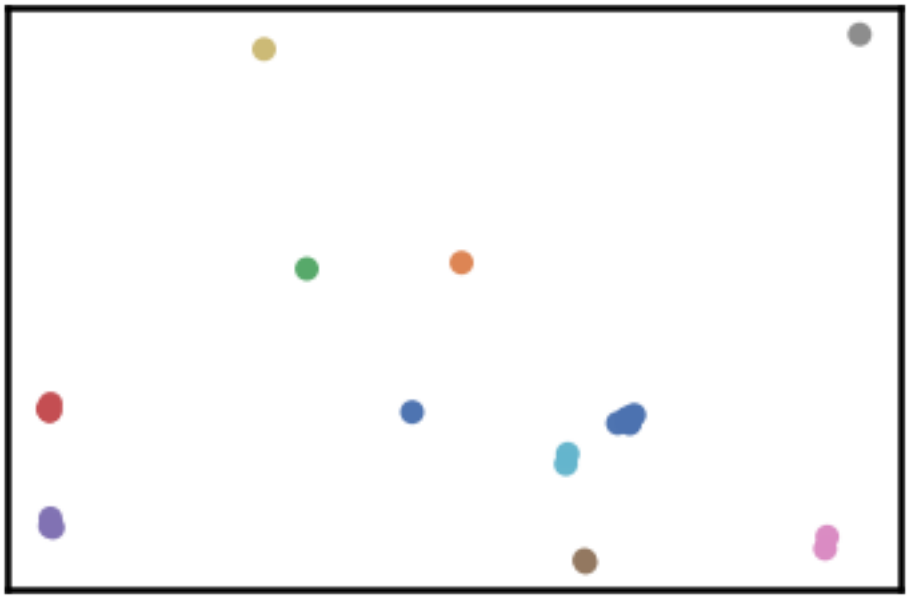
```
itration 15
```



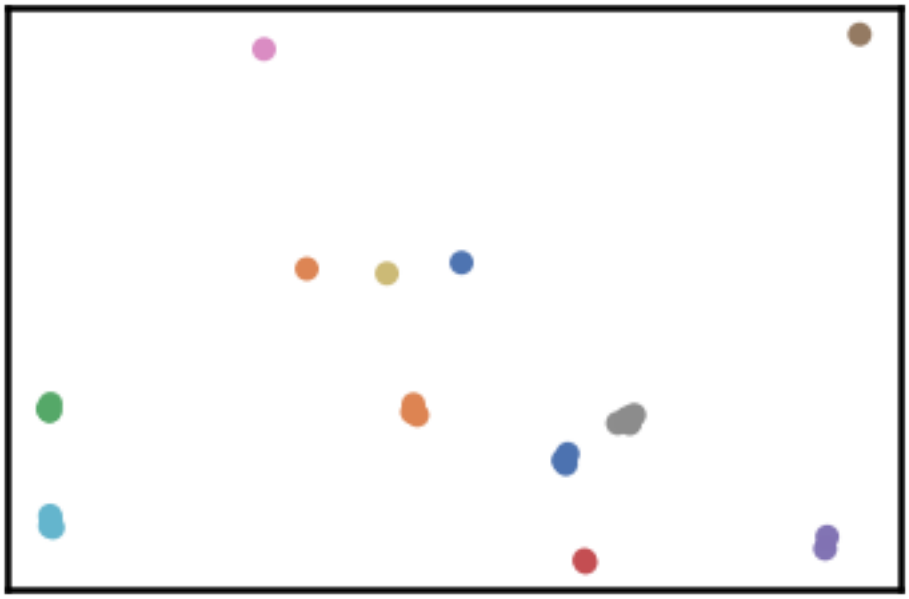
```
itration 20
```



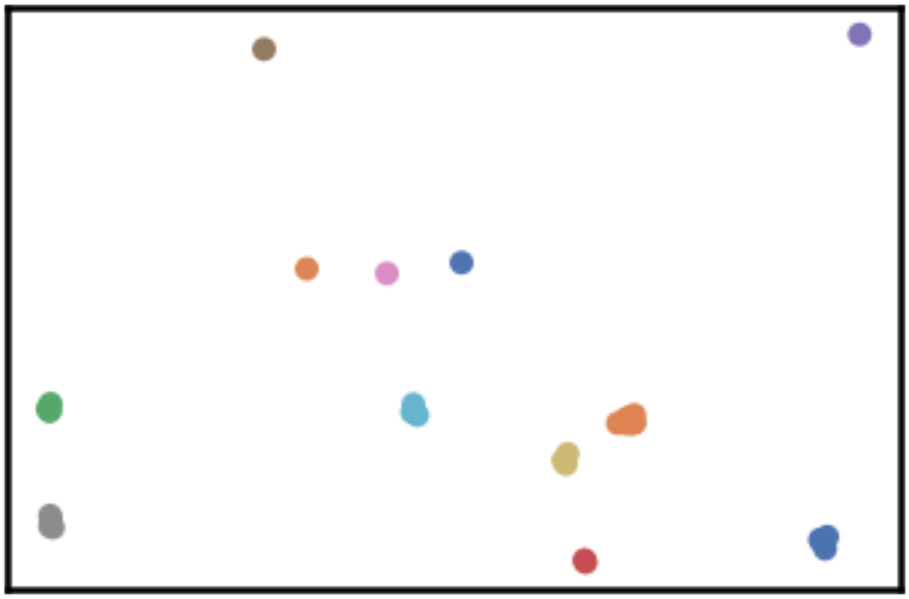
```
itration 25
```



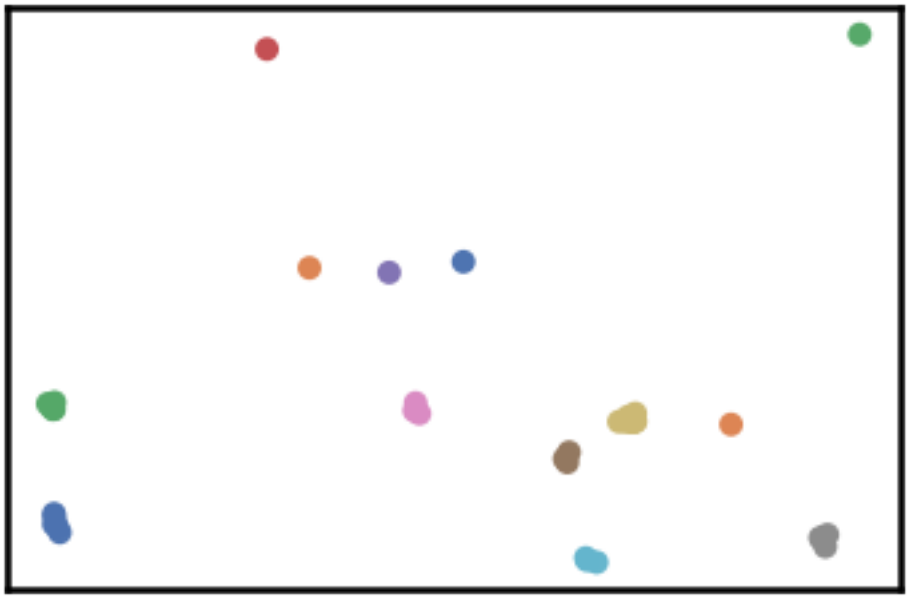
```
itration 30
```



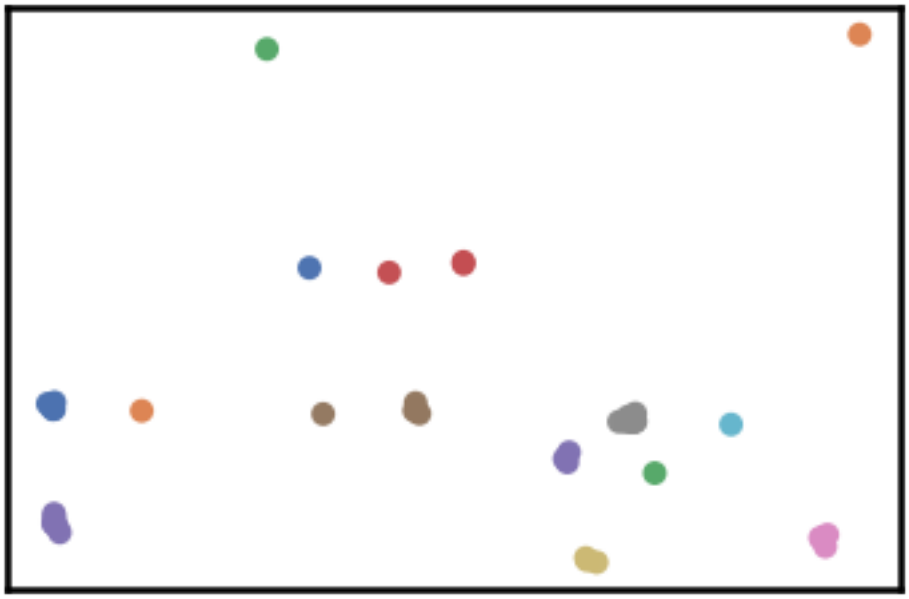
```
itration 35
```



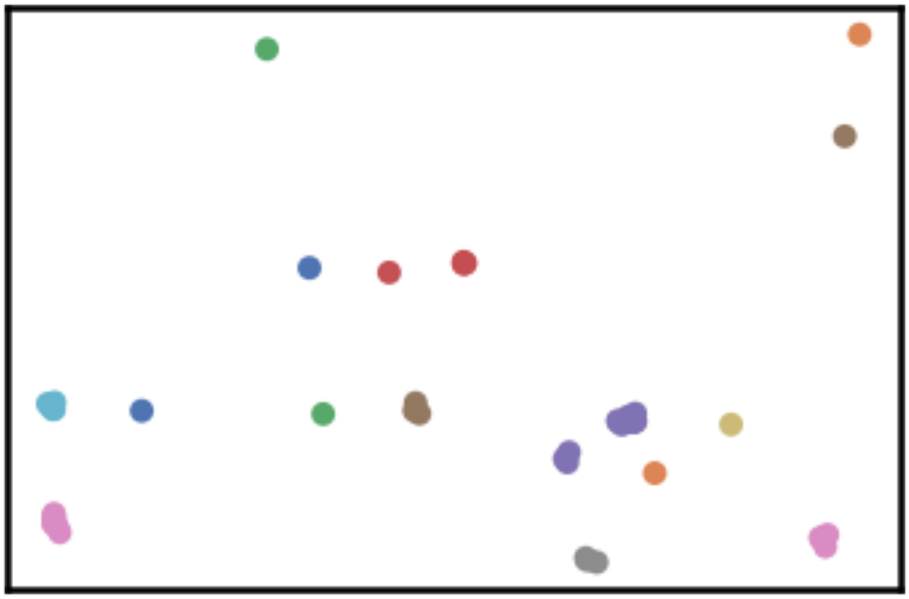
```
itration 40
```



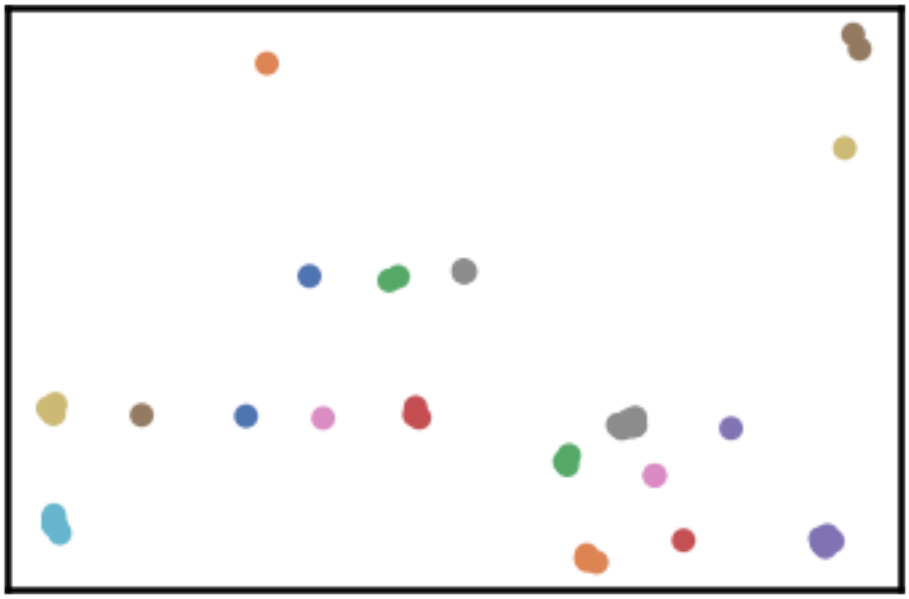
```
itration 45
```



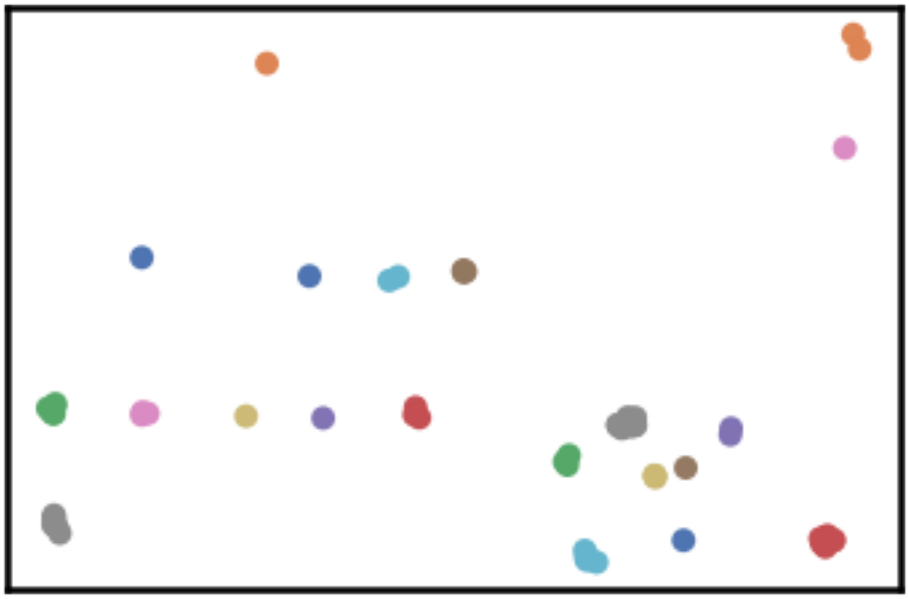
```
itration 50
```



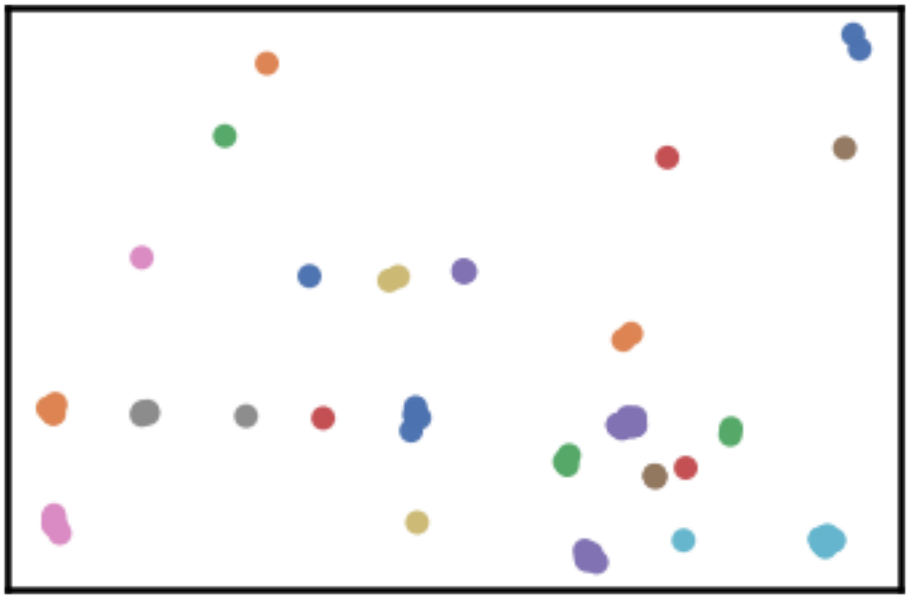
```
itration 60
```



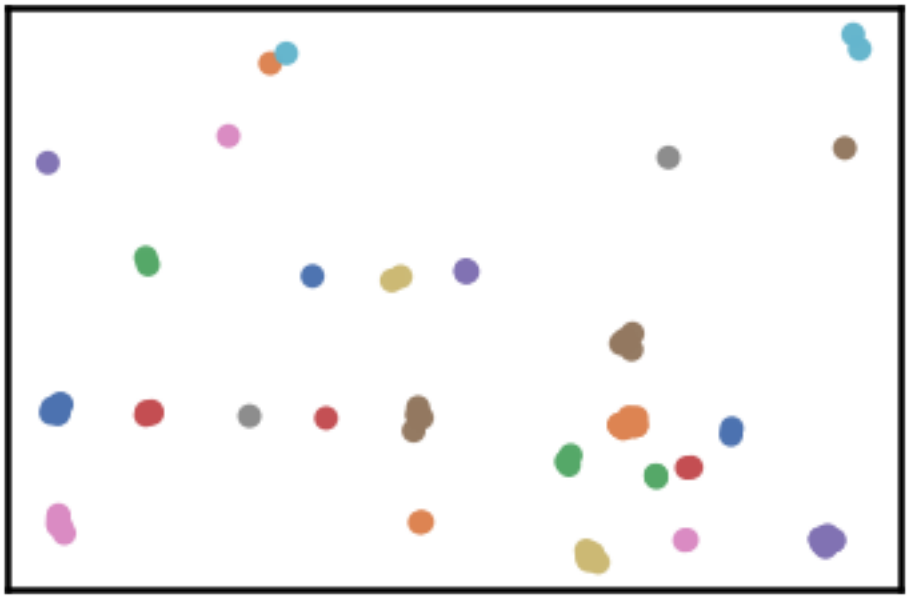
```
itration 70
```



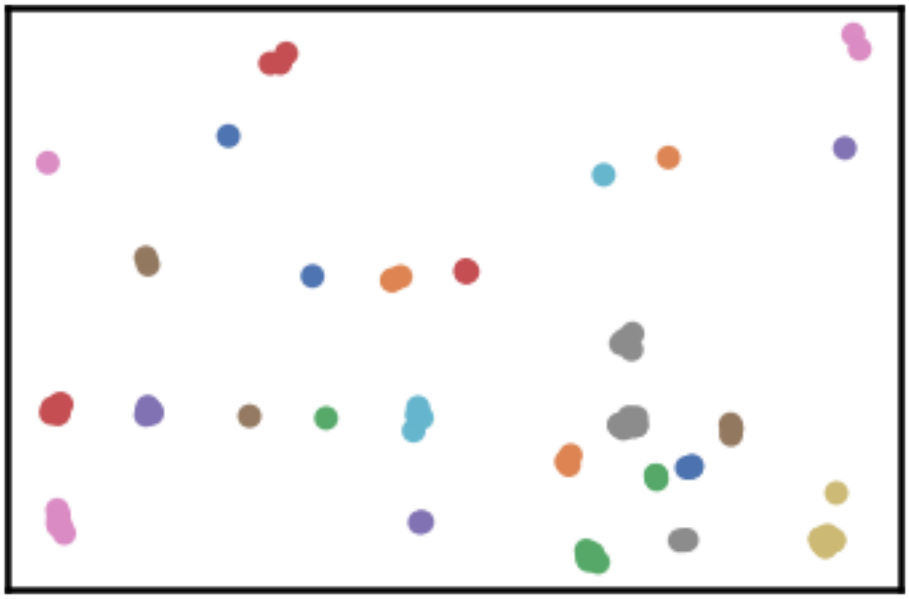
```
itration 80
```



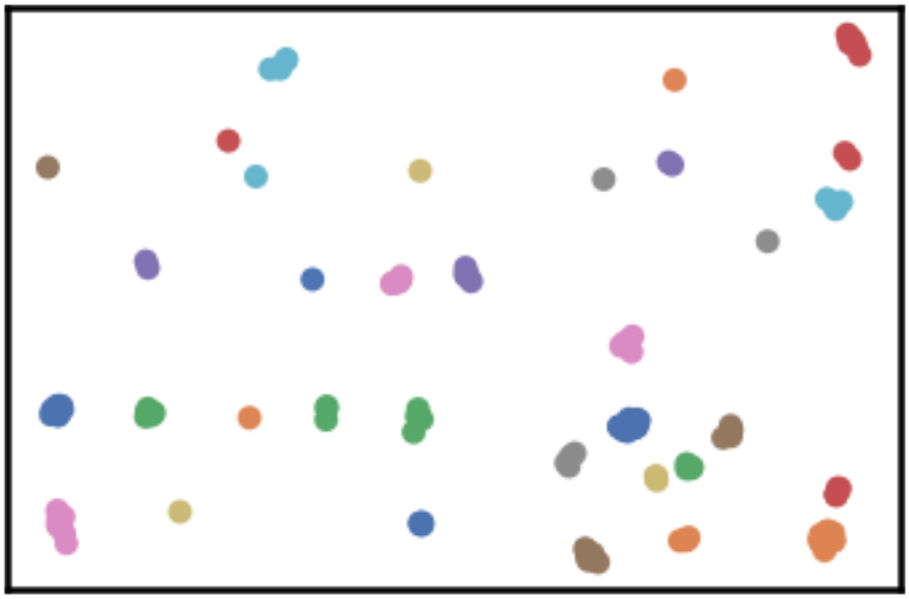
```
itration 90
```



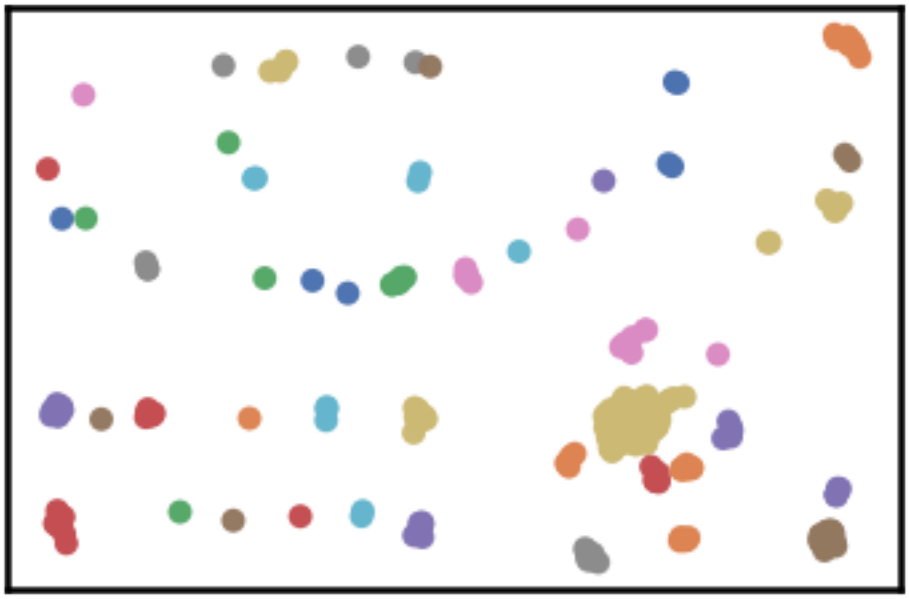
```
itration 100
```



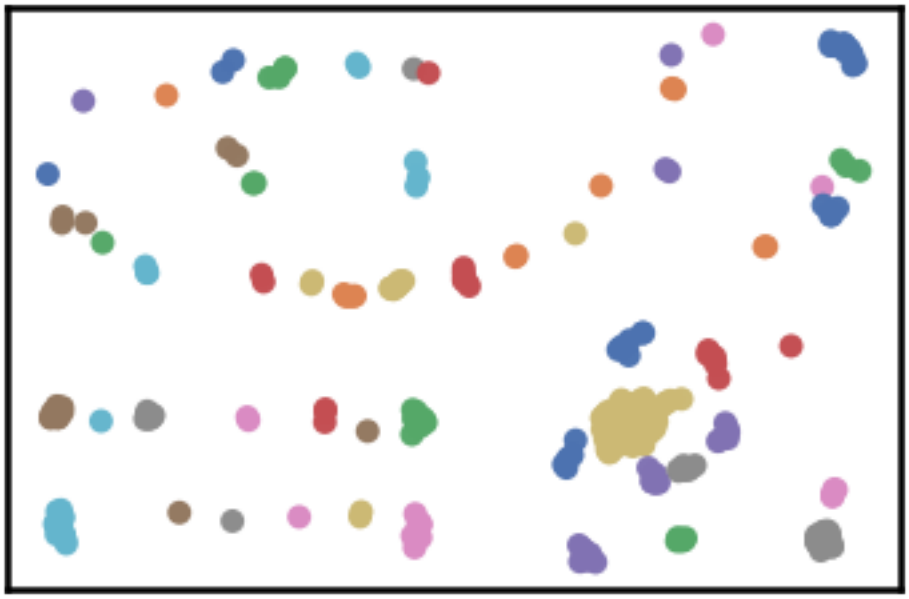
```
itration 150
```



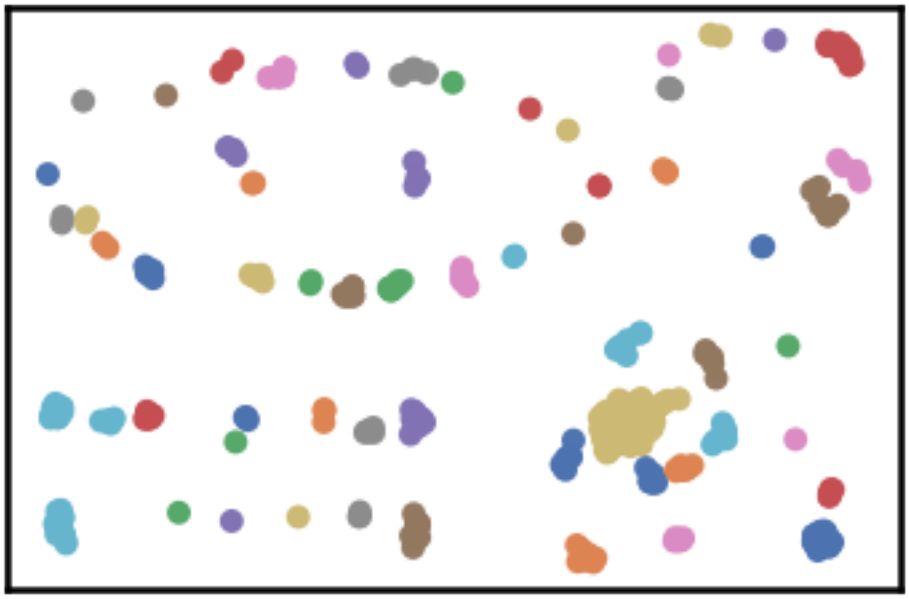
```
itration 200
```



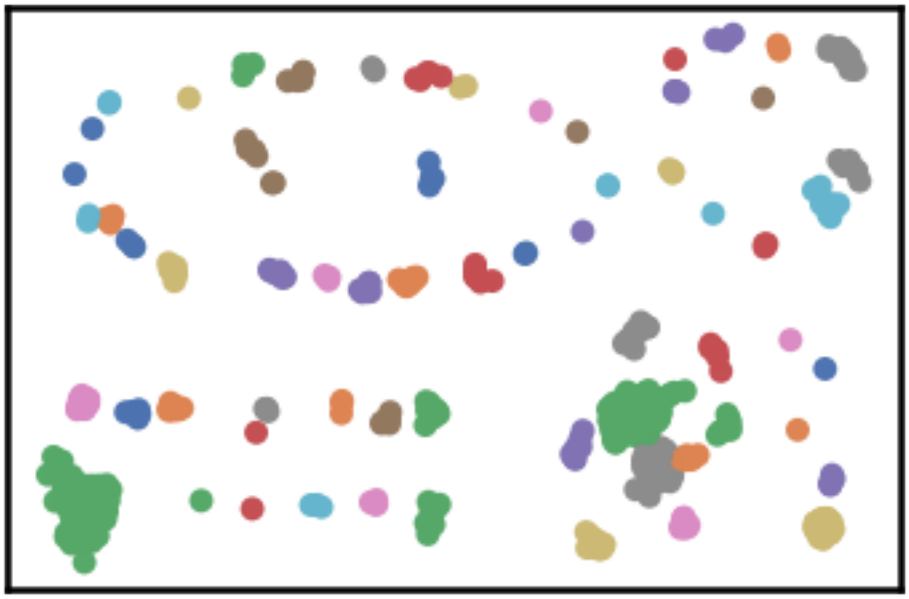
```
itration 250
```



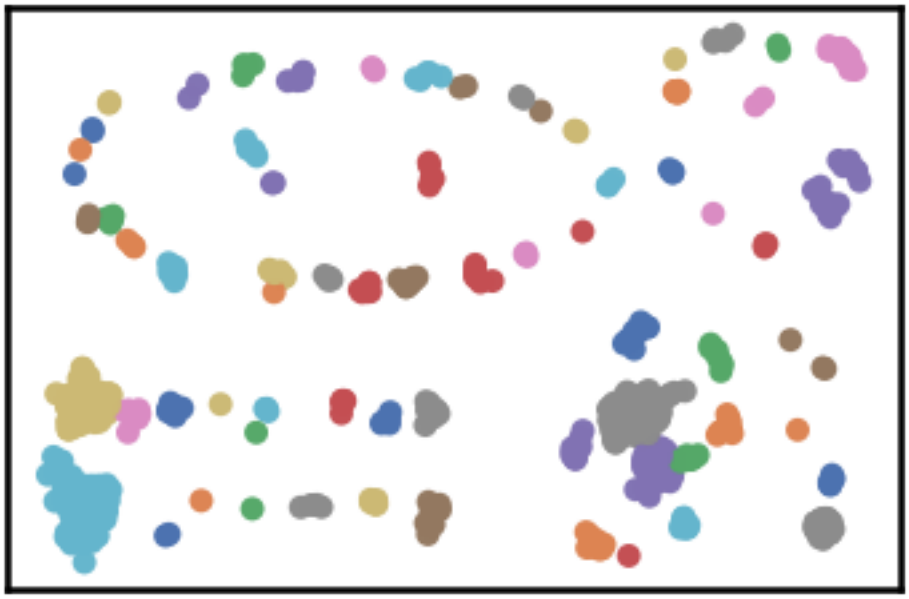
```
itration 300
```



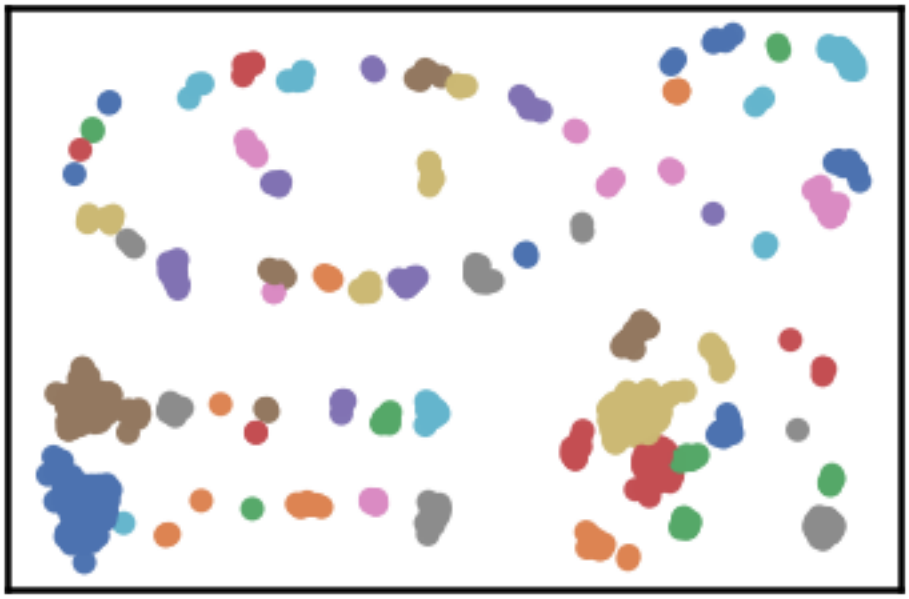
```
itration 350
```



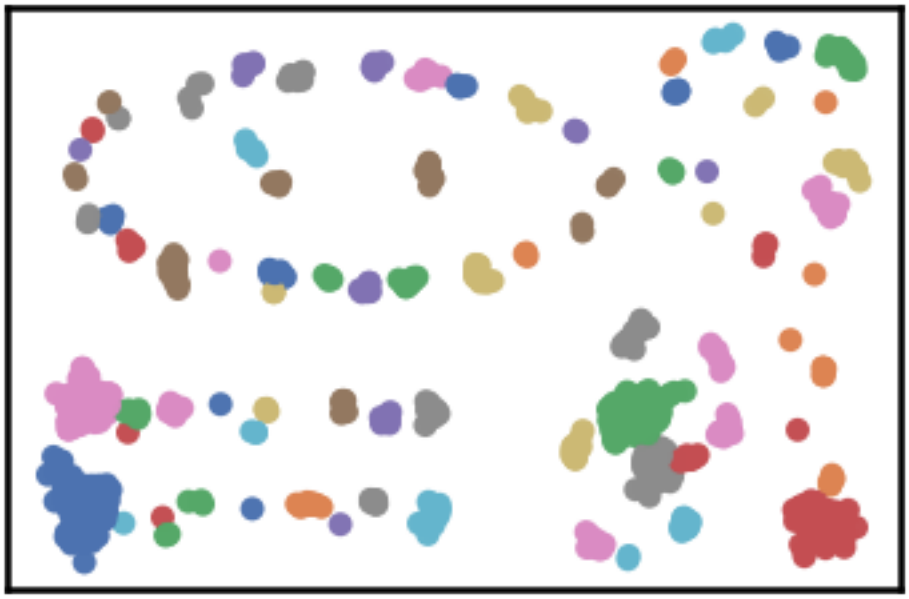
```
itration 400
```



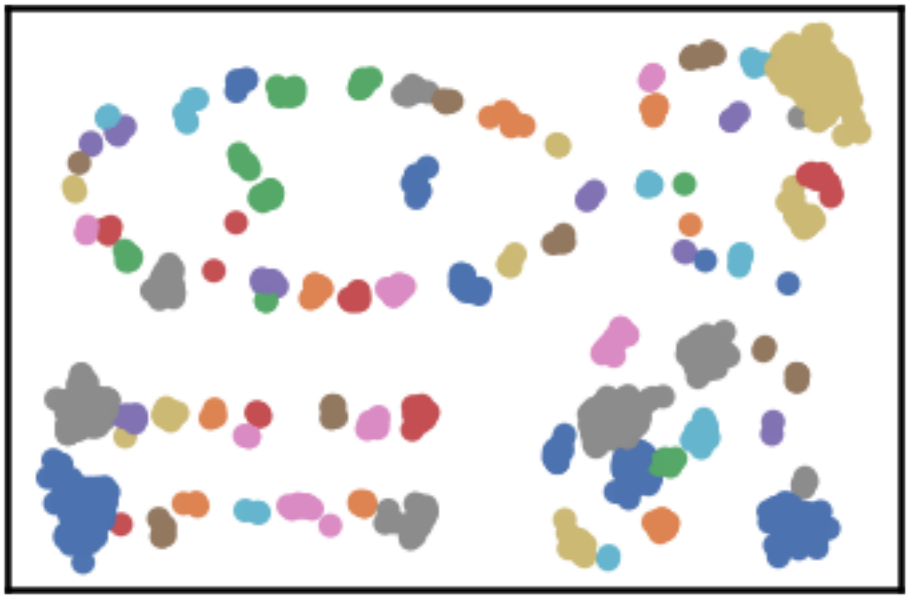
```
itration 450
```



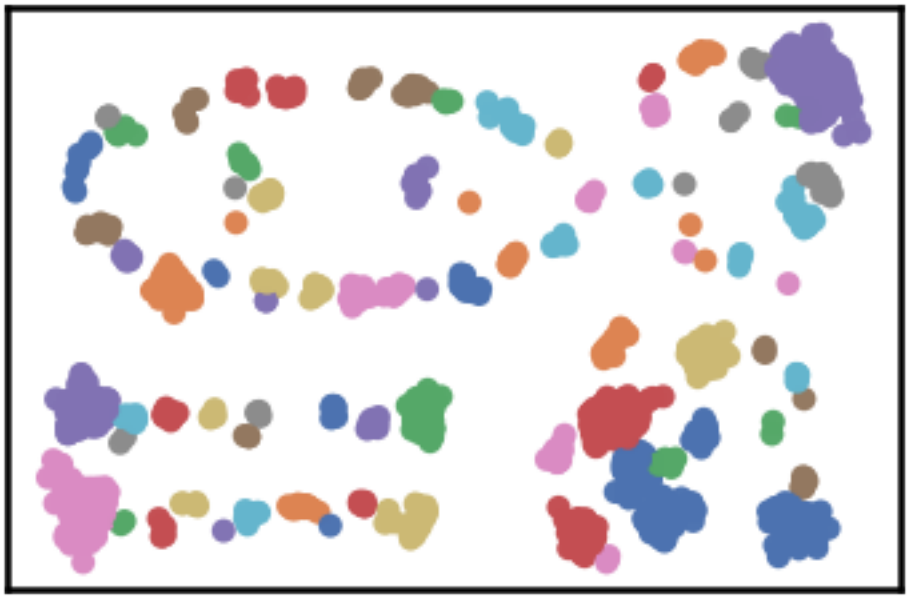
```
itration 500
```



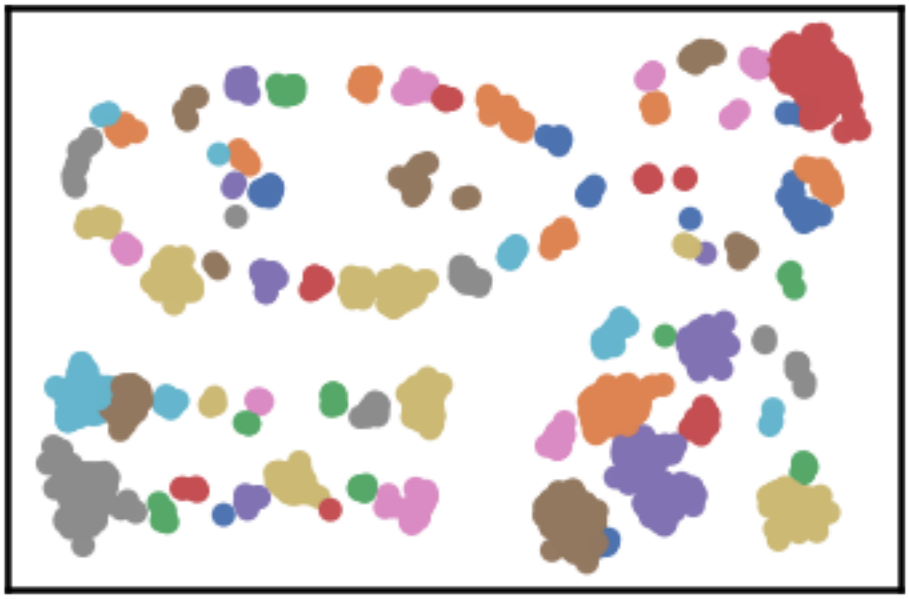
```
itration 600
```



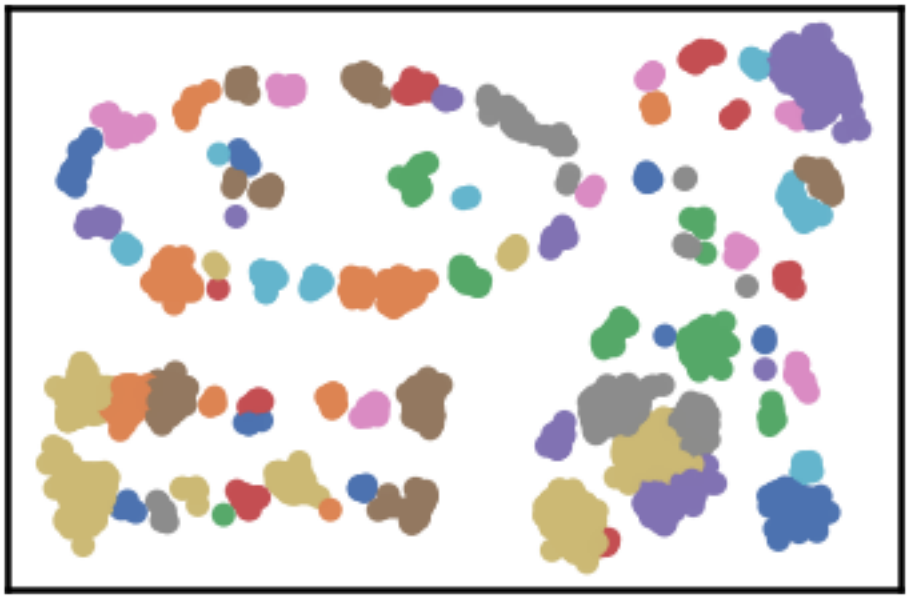
```
itration 700
```



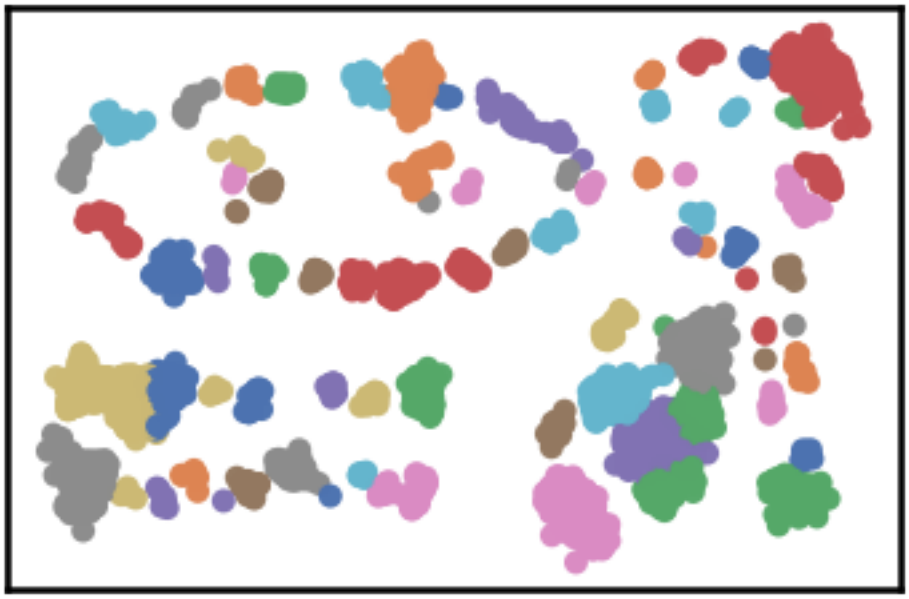
```
itration 800
```



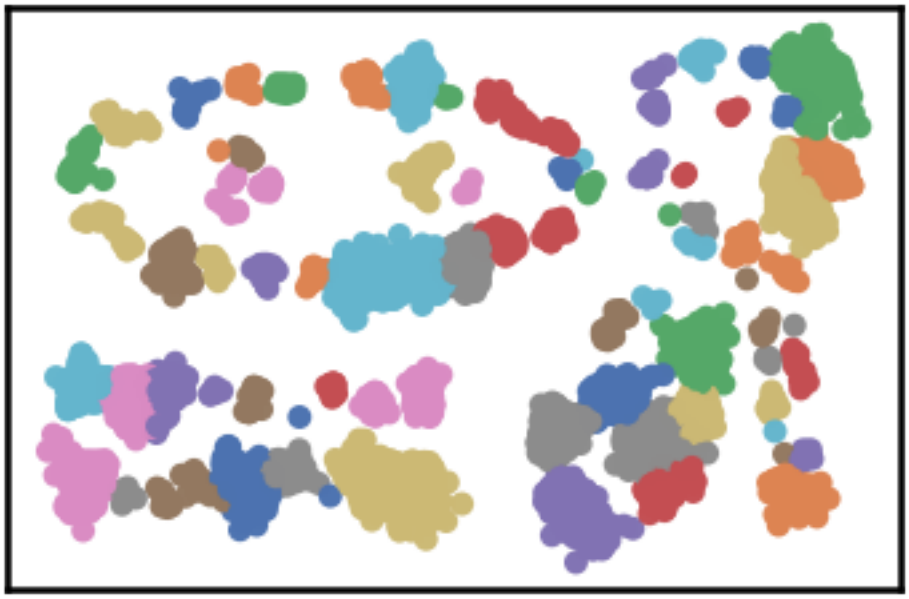
```
itration 900
```



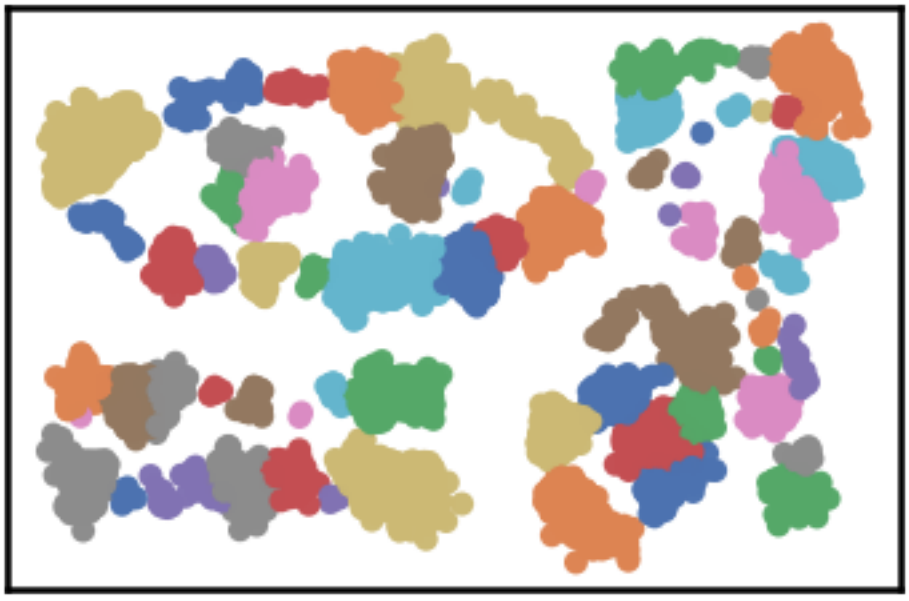
```
itration 1000
```



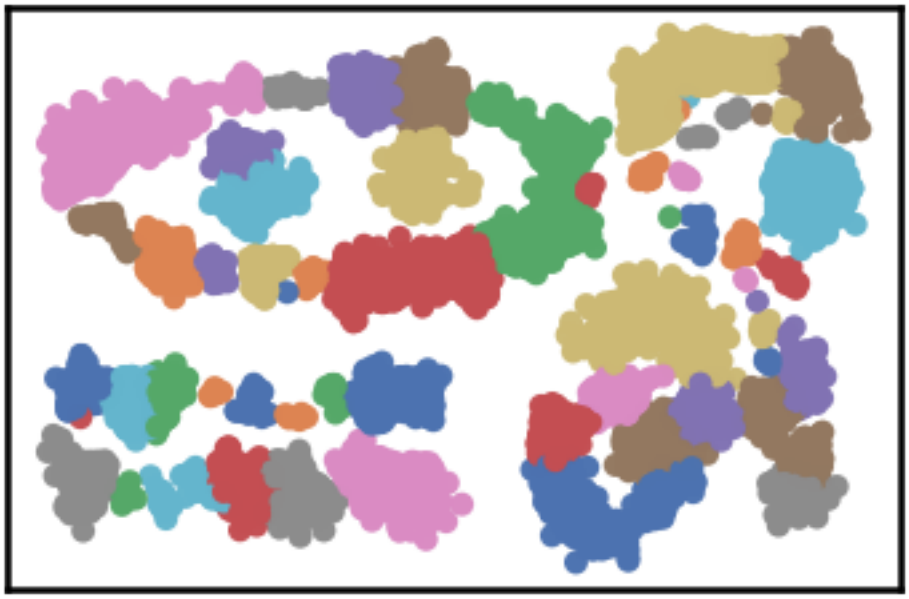
```
itration 1250
```



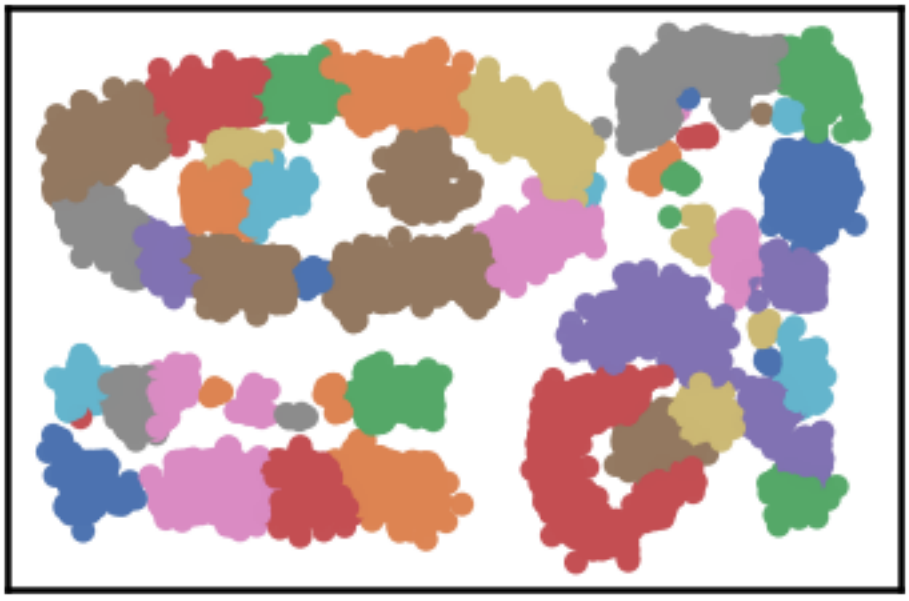
```
itration 1500
```



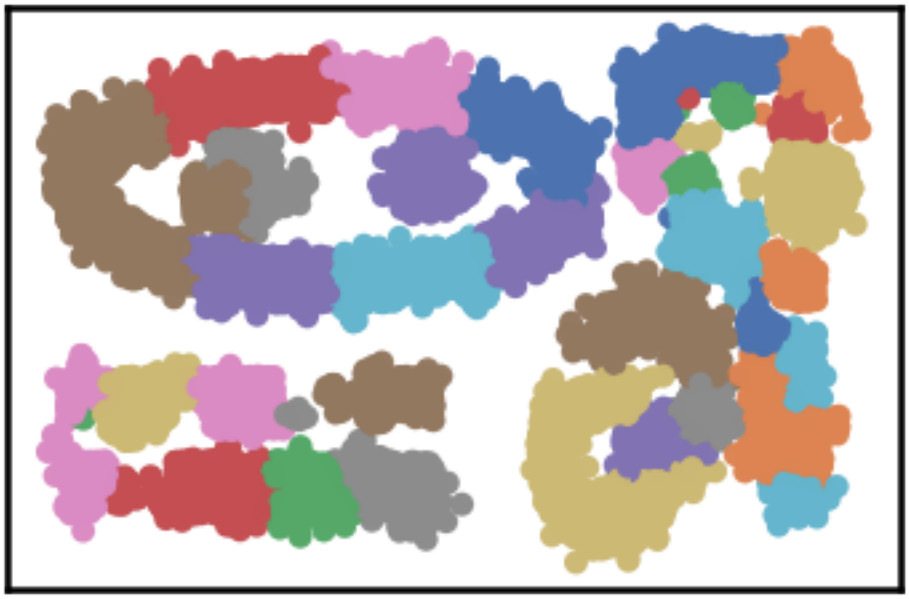
```
itration 1750
```



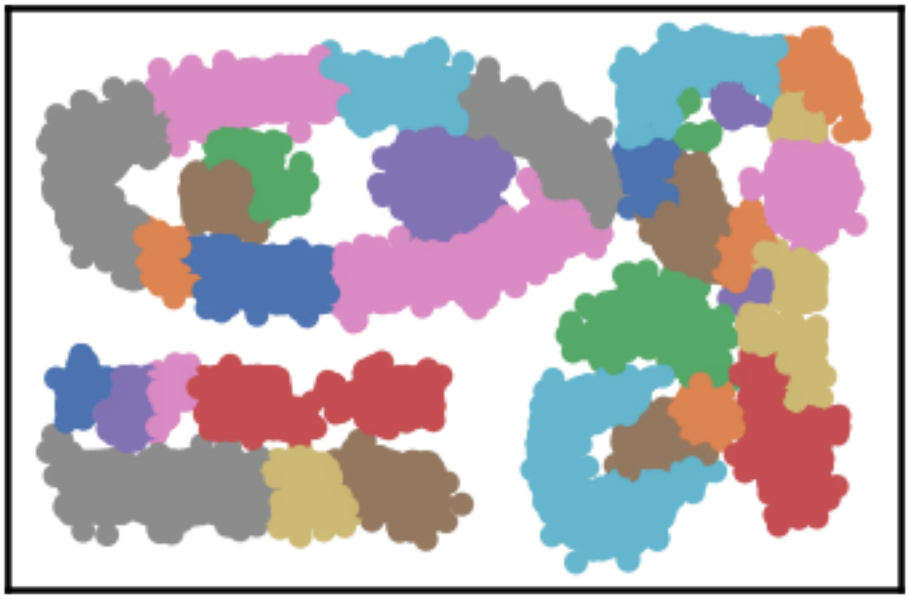
```
itration 2000
```



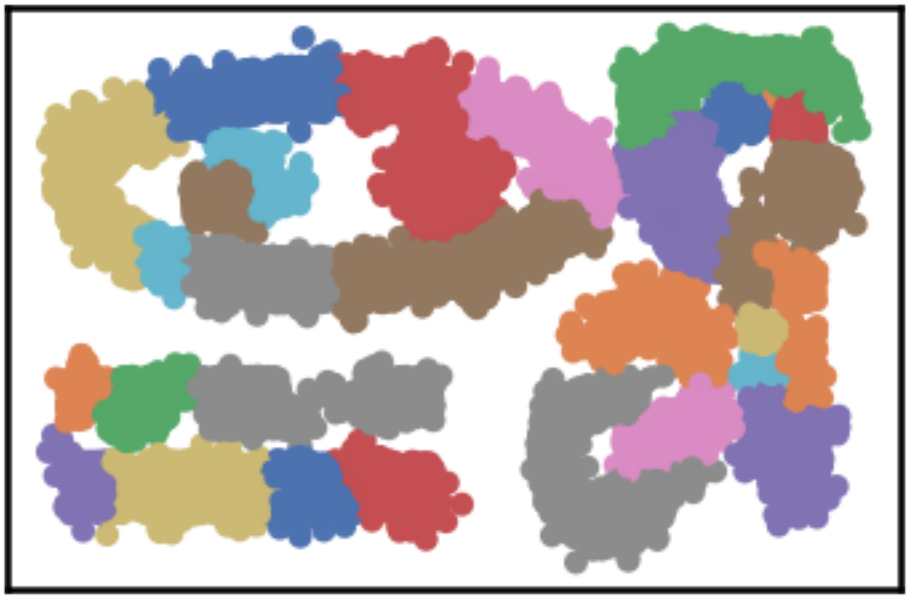
```
itration 2250
```



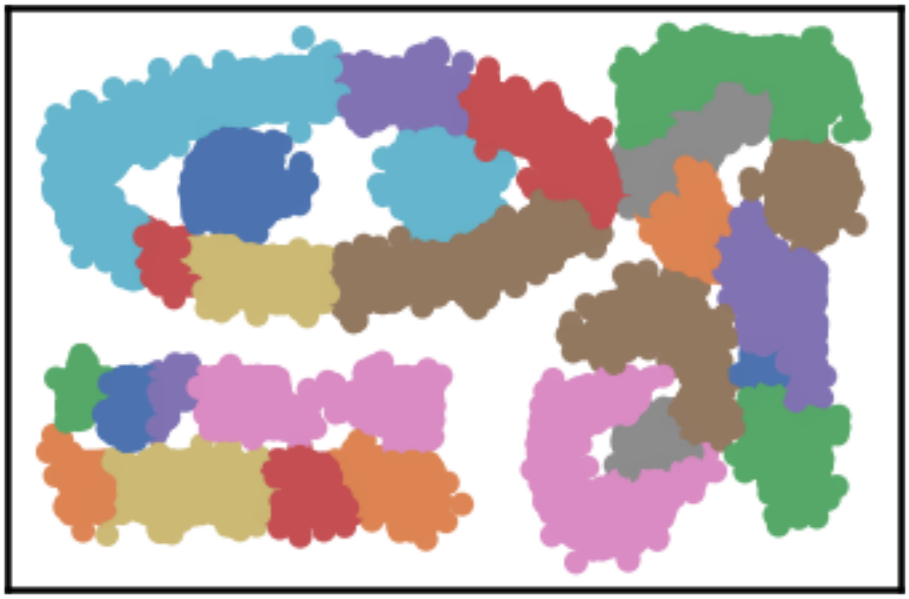
```
itration 2500
```



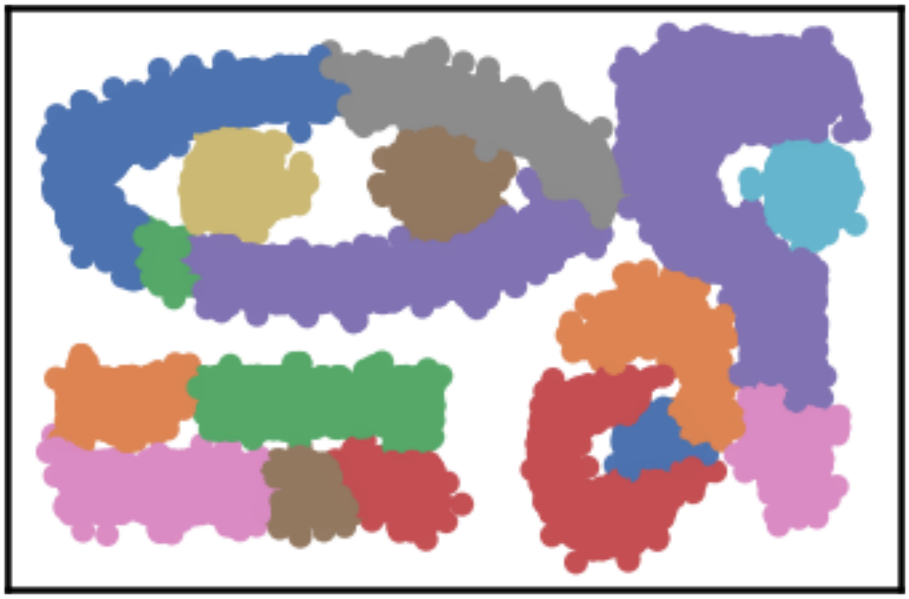
```
itration 2750
```



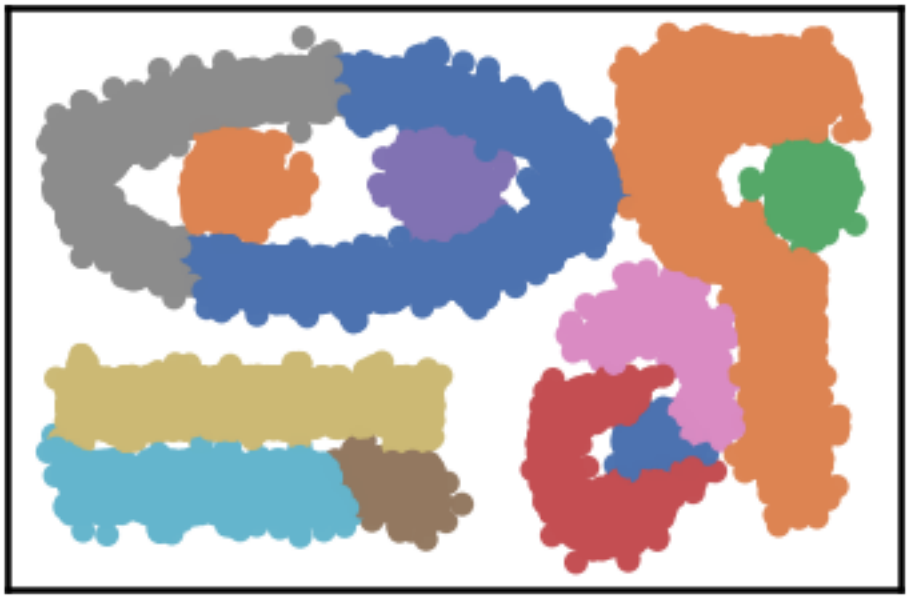
```
itration 3000
```



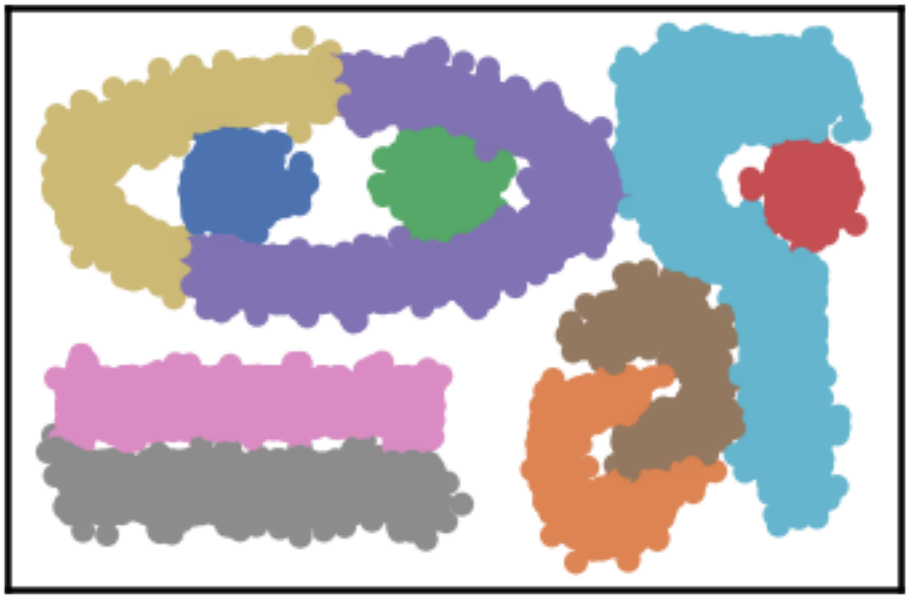
```
itration 3500
```



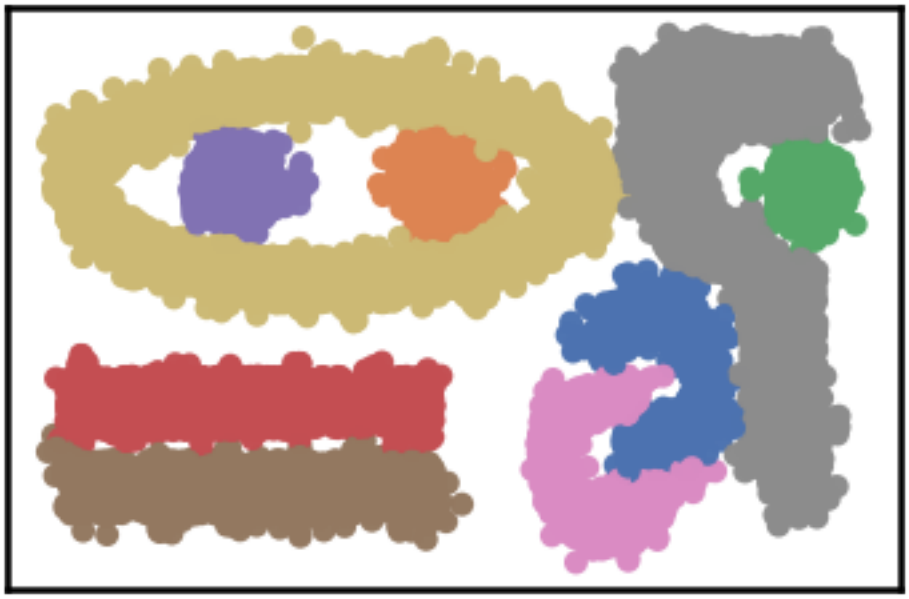
```
itration 4000
```



```
itration 4500
```



```
itration 5000
```



```
from PIL import Image

# collect immages for each snapshot automatically by the algorithm in a folder named␣
↪propagation
images = []
prop_folder = 'propagation'
img_files = os.listdir(prop_folder)
img_files = [os.path.join(prop_folder, f) for f in img_files]
sorted_files = sorted (img_files, key=os.path.getmtime)
```

(continues on next page)

(continued from previous page)

```
for filename in sorted_files:
  im = Image.open(filename)
  images.append(im)

# create annimated gif to show evolution of the propagation
images[0].save('propagation.gif', save_all=True, append_images=images[1:],␣
↪optimize=False, duration=800, loop=1)
```

3.4 Scalability

```
from sklearn import cluster, datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# Denmune's Paramaters
knn = 25 # k-nearest neighbor, the only parameter required by the algorithm
data_scale = []

for n in range(1000, 100000, 1000):
    n_samples = n
    noisy_circles = datasets.make_circles(n_samples=n_samples, factor=0.5, noise=0.05)

    data= noisy_circles[0]
    data_labels = noisy_circles[1]
    dm = DenMune(train_data=data, k_nearest=knn, rgn_tsne=True)
    labels, validity = dm.fit_predict(show_noise=True, show_analyzer=False, show_
↪plots=False)
    time_exec = dm.analyzer['exec_time']['DenMune']
    data_scale.append([n, time_exec ])

    print('data size:',n ,  'time:' , round(time_exec,4), 'seconds')
```

```
data size: 1000 time: 0.4518 seconds
data size: 2000 time: 0.93 seconds
data size: 3000 time: 1.3754 seconds
```

(continues on next page)

(continued from previous page)

```
data size: 4000 time: 2.0891 seconds
data size: 5000 time: 2.8772 seconds
data size: 6000 time: 4.5046 seconds
data size: 7000 time: 5.7184 seconds
data size: 8000 time: 4.836 seconds
data size: 9000 time: 7.793 seconds
data size: 10000 time: 8.3138 seconds
data size: 11000 time: 8.7401 seconds
data size: 12000 time: 9.8531 seconds
data size: 13000 time: 11.2796 seconds
data size: 14000 time: 13.4036 seconds
data size: 15000 time: 16.6113 seconds
data size: 16000 time: 14.4252 seconds
data size: 17000 time: 20.697 seconds
data size: 18000 time: 18.1152 seconds
data size: 19000 time: 22.1096 seconds
data size: 20000 time: 25.8013 seconds
data size: 21000 time: 26.6907 seconds
data size: 22000 time: 27.0235 seconds
data size: 23000 time: 27.3918 seconds
data size: 24000 time: 38.0108 seconds
data size: 25000 time: 41.3266 seconds
data size: 26000 time: 36.7593 seconds
data size: 27000 time: 42.6916 seconds
data size: 28000 time: 41.0344 seconds
data size: 29000 time: 42.878 seconds
data size: 30000 time: 50.9385 seconds
data size: 31000 time: 51.326 seconds
data size: 32000 time: 54.6266 seconds
data size: 33000 time: 50.0233 seconds
data size: 34000 time: 59.8251 seconds
data size: 35000 time: 51.2865 seconds
data size: 36000 time: 62.331 seconds
data size: 37000 time: 59.3316 seconds
data size: 38000 time: 67.6423 seconds
data size: 39000 time: 72.0803 seconds
data size: 40000 time: 70.2149 seconds
data size: 41000 time: 71.7297 seconds
data size: 42000 time: 74.2931 seconds
data size: 43000 time: 76.8439 seconds
data size: 44000 time: 93.3641 seconds
data size: 45000 time: 93.5944 seconds
data size: 46000 time: 74.4506 seconds
data size: 47000 time: 94.0584 seconds
data size: 48000 time: 105.9512 seconds
data size: 49000 time: 94.7943 seconds
data size: 50000 time: 88.705 seconds
data size: 51000 time: 110.4996 seconds
data size: 52000 time: 119.498 seconds
data size: 53000 time: 125.8244 seconds
data size: 54000 time: 117.446 seconds
data size: 55000 time: 129.3928 seconds
```

(continues on next page)

(continued from previous page)

```
data size: 56000 time: 135.6952 seconds
data size: 57000 time: 137.2575 seconds
data size: 58000 time: 146.3149 seconds
data size: 59000 time: 131.5086 seconds
data size: 60000 time: 160.2656 seconds
data size: 61000 time: 160.4223 seconds
data size: 62000 time: 149.5028 seconds
data size: 63000 time: 153.2287 seconds
data size: 64000 time: 163.9789 seconds
data size: 65000 time: 174.5982 seconds
data size: 66000 time: 190.9555 seconds
data size: 67000 time: 185.289 seconds
data size: 68000 time: 238.9069 seconds
data size: 69000 time: 182.9564 seconds
data size: 70000 time: 209.4002 seconds
data size: 71000 time: 243.5443 seconds
data size: 72000 time: 208.053 seconds
data size: 73000 time: 220.6103 seconds
data size: 74000 time: 217.0223 seconds
data size: 75000 time: 223.4226 seconds
data size: 76000 time: 233.5772 seconds
data size: 77000 time: 238.5042 seconds
data size: 78000 time: 245.0213 seconds
data size: 79000 time: 221.3705 seconds
data size: 80000 time: 244.24 seconds
data size: 81000 time: 274.9946 seconds
data size: 82000 time: 253.6804 seconds
data size: 83000 time: 286.6225 seconds
data size: 84000 time: 266.2466 seconds
data size: 85000 time: 317.2572 seconds
data size: 86000 time: 325.8073 seconds
data size: 87000 time: 303.0139 seconds
data size: 88000 time: 338.6655 seconds
data size: 89000 time: 343.8219 seconds
data size: 90000 time: 347.6194 seconds
data size: 91000 time: 325.1733 seconds
data size: 92000 time: 359.6347 seconds
data size: 93000 time: 350.8712 seconds
data size: 94000 time: 379.433 seconds
data size: 95000 time: 334.2113 seconds
data size: 96000 time: 338.2628 seconds
data size: 97000 time: 371.5167 seconds
data size: 98000 time: 347.7595 seconds
data size: 99000 time: 391.2152 seconds
```

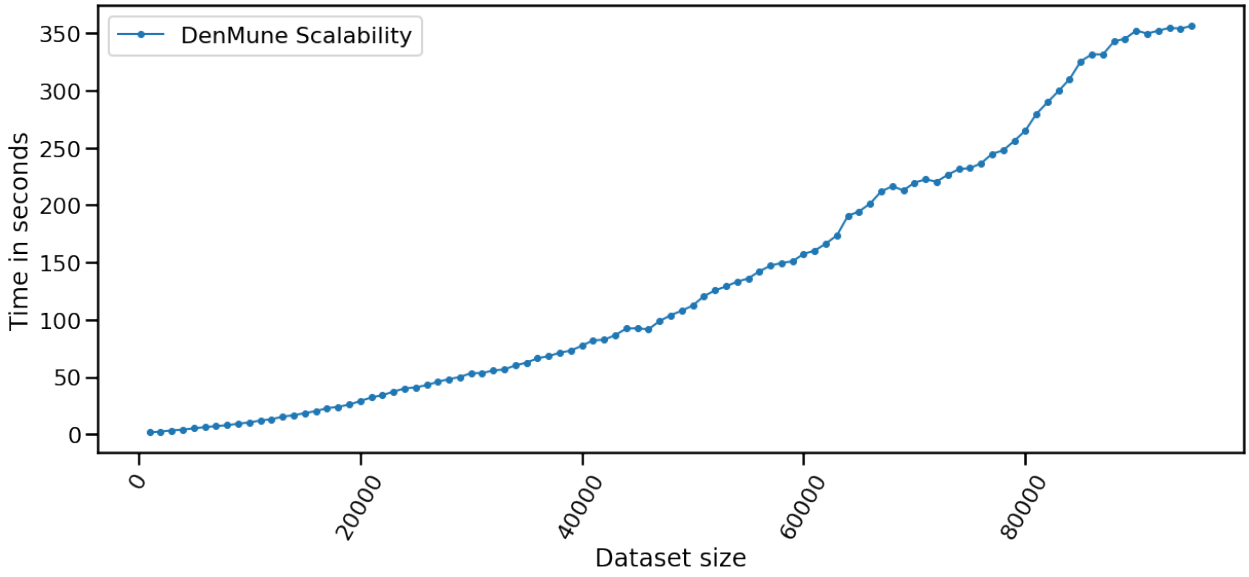
```
# compouting moving average to smooth the curve
x, y = zip(*data_scale)
window = 5
cumsum, moving_aves = [0], []

for i, n in enumerate(y, 1):
    cumsum.append(cumsum[i-1] + n)
```

(continues on next page)

(continued from previous page)

```
    if i>=window:
        moving_ave = (cumsum[i] - cumsum[i-window])/window
        #can do stuff with moving_ave here
        moving_aves.append(moving_ave)
y = moving_aves
```

```
# Creating figure and axis objects using subplots()
fig, ax = plt.subplots(figsize=[20, 8])
ax.plot(x[:-window+1], y, marker='.', linewidth=2, label='DenMune Scalability')
plt.xticks(rotation=60)
ax.set_xlabel('Dataset size')
ax.set_ylabel('Time in seconds')
plt.legend()
plt.show()
```



3.5 Stability

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune
```

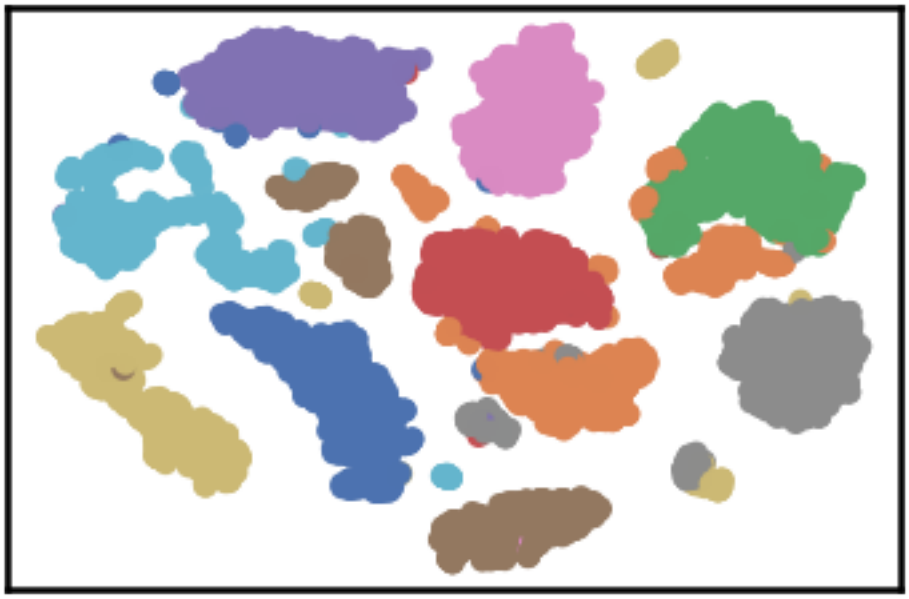
(continues on next page)

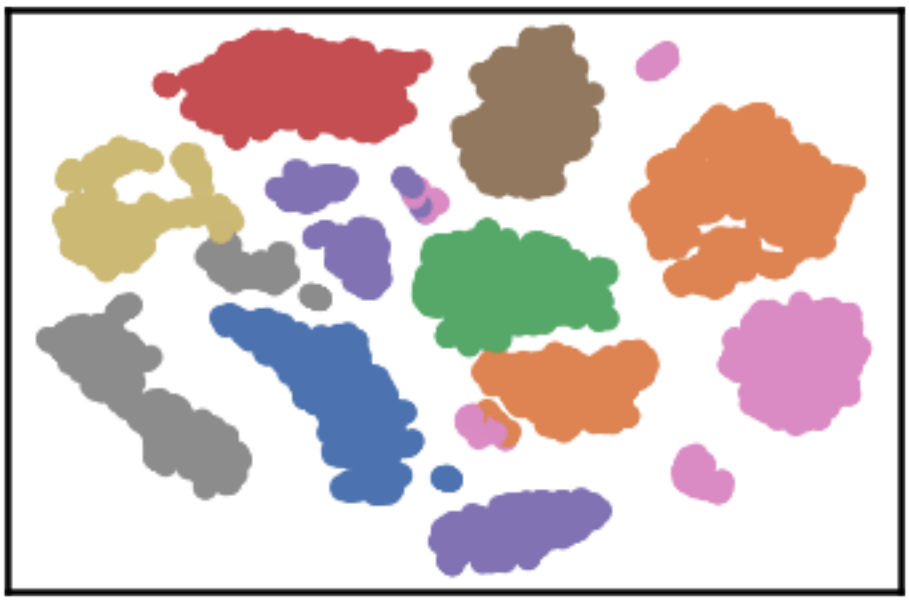
(continued from previous page)

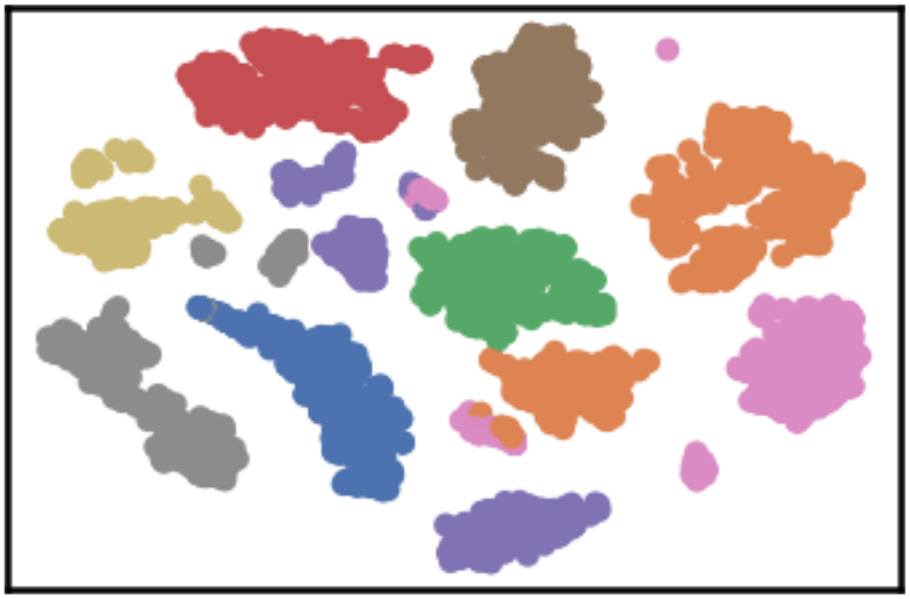
```
# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
data_path = 'datasets/denmune/pendigits/'
file_2d = data_path + 'pendigits-2d.csv'

X_train = pd.read_csv(data_path + 'train.csv', sep=',', header=None)
y_train = X_train.iloc[:, -1]
X_train = X_train.drop(X_train.columns[-1], axis=1)
X_test = pd.read_csv(data_path + 'test.csv', sep=',', header=None)
X_test = X_test.drop(X_test.columns[-1], axis=1)

data_stability = []

from IPython.display import clear_output
for knn in range (1, 200):
  clear_output(wait=True)
  dm = DenMune(train_data=X_train,
              train_truth=y_train,
              test_data=X_test,
              k_nearest=knn,
              file_2d=file_2d,
              rgn_tsne=False)

  labels, validity = dm.fit_predict(show_plots=True, show_analyzer=False)

  validity_key = "F1"
  print ('k=' , knn, validity_key , 'score is:', validity['train'][validity_key])
  data_stability.append([knn, validity['train'][validity_key]])
```



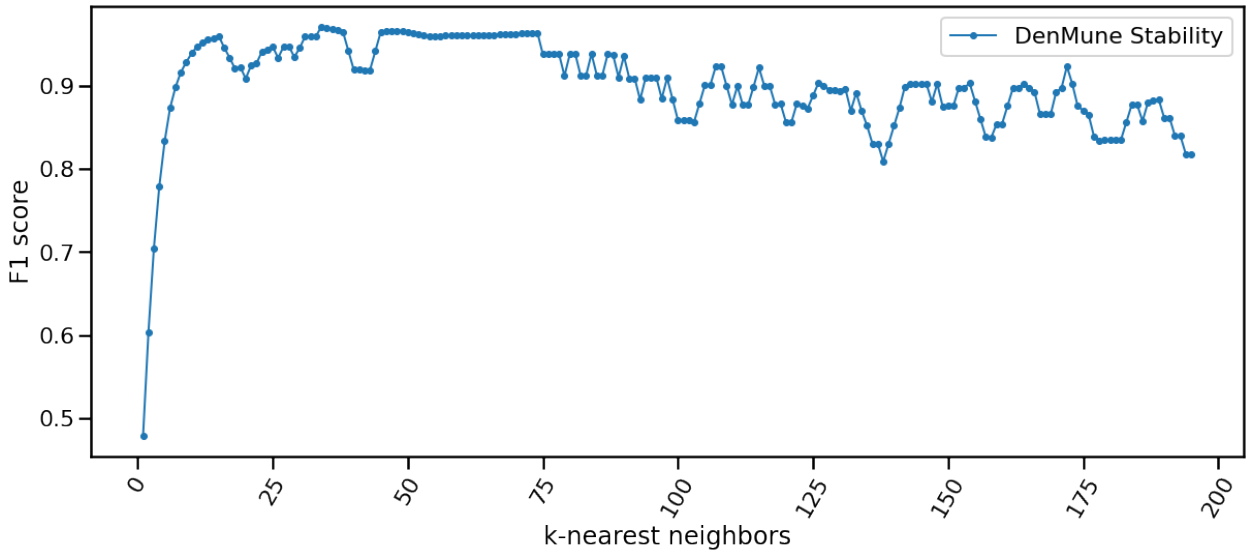




```
k= 199 F1 score is: 0.7960390378193465
```

```
#computing moving average to smoth the curve
x, y = zip(*data_stability)
window = 5
cumsum, moving_aves = [0], []

for i, n in enumerate(y, 1):
    cumsum.append(cumsum[i-1] + n)
    if i>=window:
        moving_ave = (cumsum[i] - cumsum[i-window])/window
        #can do stuff with moving_ave here
        moving_aves.append(moving_ave)
y = moving_aves
```

```
# Creating figure and axis objects using subplots()
fig, ax = plt.subplots(figsize=[20, 8])
ax.plot(x[:-window+1], y, marker='.', linewidth=2, label='DenMune Stability')
plt.xticks(rotation=60)
ax.set_xlabel('k-nearest neighbors')
ax.set_ylabel(validity_key + ' score')
plt.legend()
plt.show()
```



3.6 K-nearest Neighbor Evolution

```
import pandas as pd
import matplotlib.pyplot as plt
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

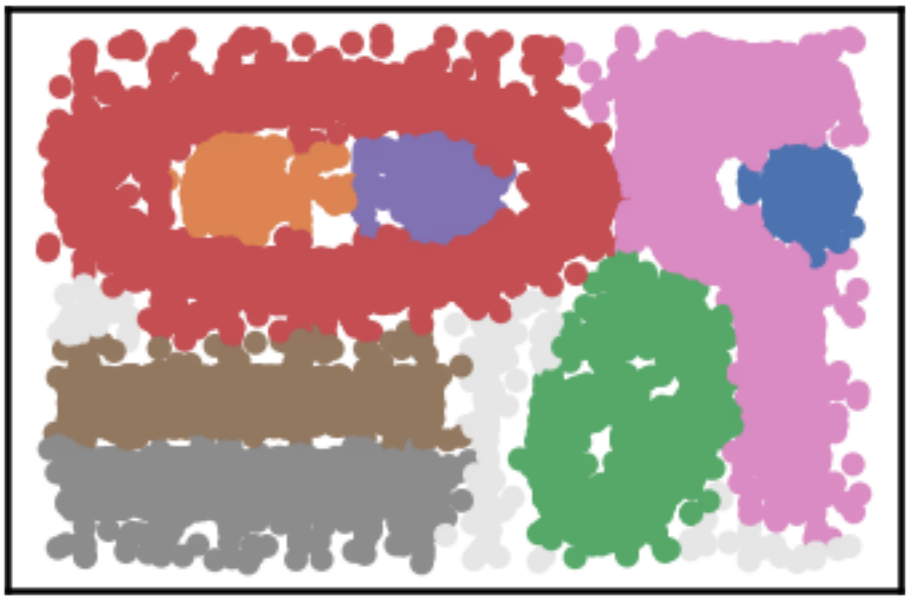
```
Cloning into 'datasets'...
remote: Enumerating objects: 63, done.[K
remote: Counting objects: 100% (63/63), done.[K
remote: Compressing objects: 100% (52/52), done.[K
remote: Total 63 (delta 10), reused 59 (delta 9), pack-reused 0[K
Unpacking objects: 100% (63/63), done.
```

```
data_path = 'datasets/denmune/chameleon/'
chameleon_dataset = "t7.10k" #["t4.8k", "t5.8k", "t7.10k", "t8.8k"]
```

(continues on next page)

(continued from previous page)

```
# train file
detected_clusers = []
noise_type1 = []
noise_type2 = []

data_file = data_path + chameleon_dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)

from IPython.display import clear_output
for knn in range (1, 100, 1):
    print ("knn", knn )
    clear_output(wait=True)
    dm = DenMune(train_data=X_train, k_nearest=knn, rgn_tsne=False )
    labels, validity = dm.fit_predict(show_analyzer=False)
    n_clusters = dm.analyzer['n_clusters']['detected']
    pre_noise = dm.analyzer['n_points']['noise']['type-1']
    post_noise = dm.analyzer['n_points']['noise']['type-2']
    detected_clusers.append([knn, n_clusters ])
    noise_type1.append([knn, pre_noise ])
    noise_type2.append([knn, post_noise ])

    print('knn:',knn ,  ' :: we detected', n_clusters, 'clusters:' , ' :: pre-noise:',␣
↪pre_noise, 'post_noise', post_noise)
    time.sleep(0.2)
```



```
knn: 99  :: we detected 8 clusters:  :: pre-noise: 0 post_noise 148
```

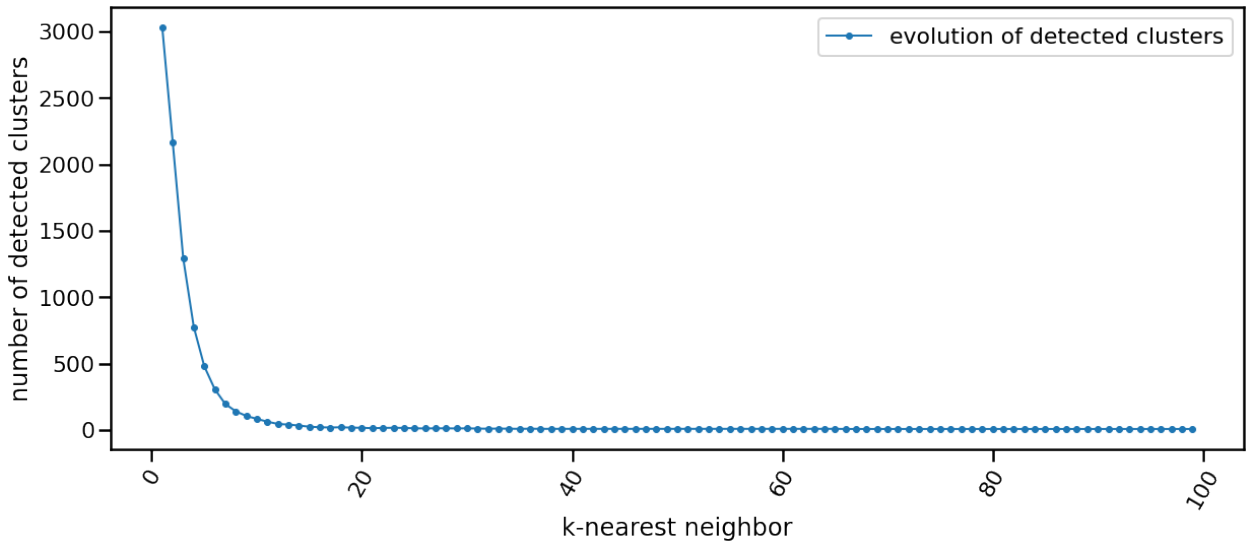
```
x, y = zip(*detected_clusers)
f1 = plt.figure(1)
# Creating figure and axis objects using subplots()
```

(continues on next page)

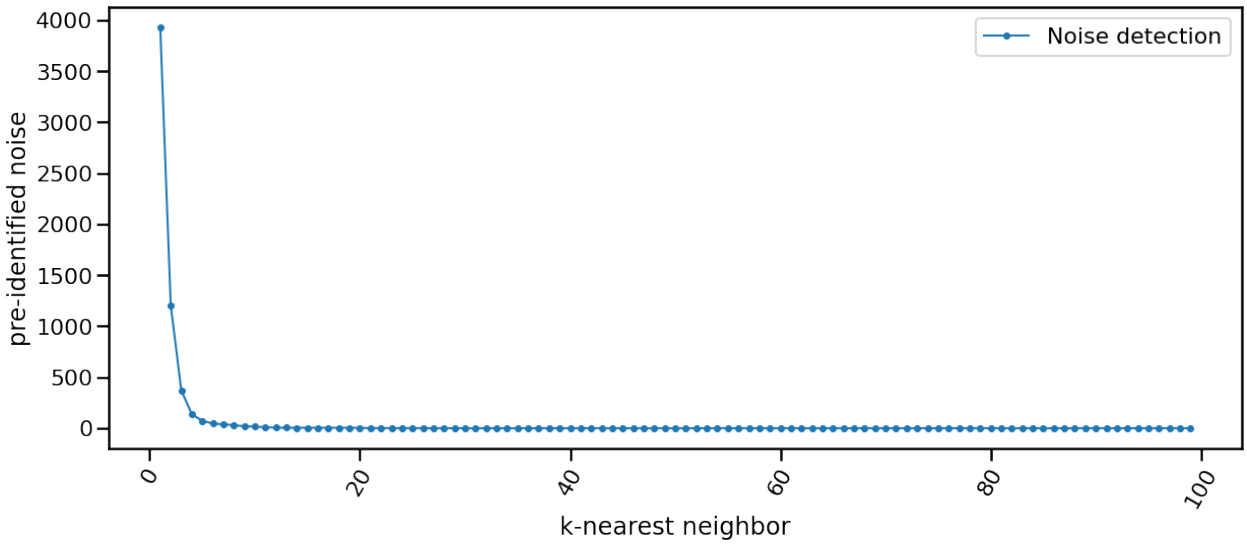
(continued from previous page)

```
fig, ax = plt.subplots(figsize=[20, 8])
ax.plot(x, y, marker='.', linewidth=2, label='evolution of detected clusters')
plt.xticks(rotation=60)
ax.set_xlabel('k-nearest neighbor')
ax.set_ylabel('number of detected clusters')
plt.legend()
plt.show()
```

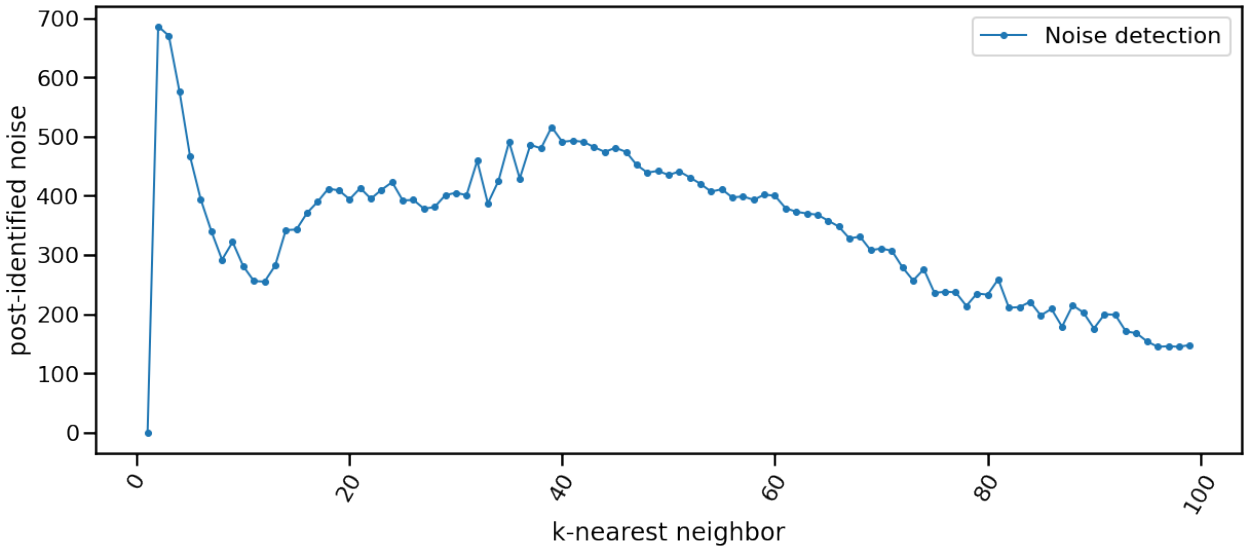
```
<Figure size 432x288 with 0 Axes>
```




```
x, y = zip(*noise_type1)

# Creating figure and axis objects using subplots()
fig, ax = plt.subplots(figsize=[20, 8])
ax.plot(x, y, marker='.', linewidth=2, label='Noise detection')
plt.xticks(rotation=60)
ax.set_xlabel('k-nearest neighbor')
ax.set_ylabel('pre-identified noise')
plt.legend()
plt.show()
```




```
x, y = zip(*noise_type2)

# Creating figure and axis objects using subplots()
fig, ax = plt.subplots(figsize=[20, 8])
ax.plot(x, y, marker='.', linewidth=2, label='Noise detection')
plt.xticks(rotation=60)
ax.set_xlabel('k-nearest neighbor')
ax.set_ylabel('post-identified noise')
plt.legend()
plt.show()
```

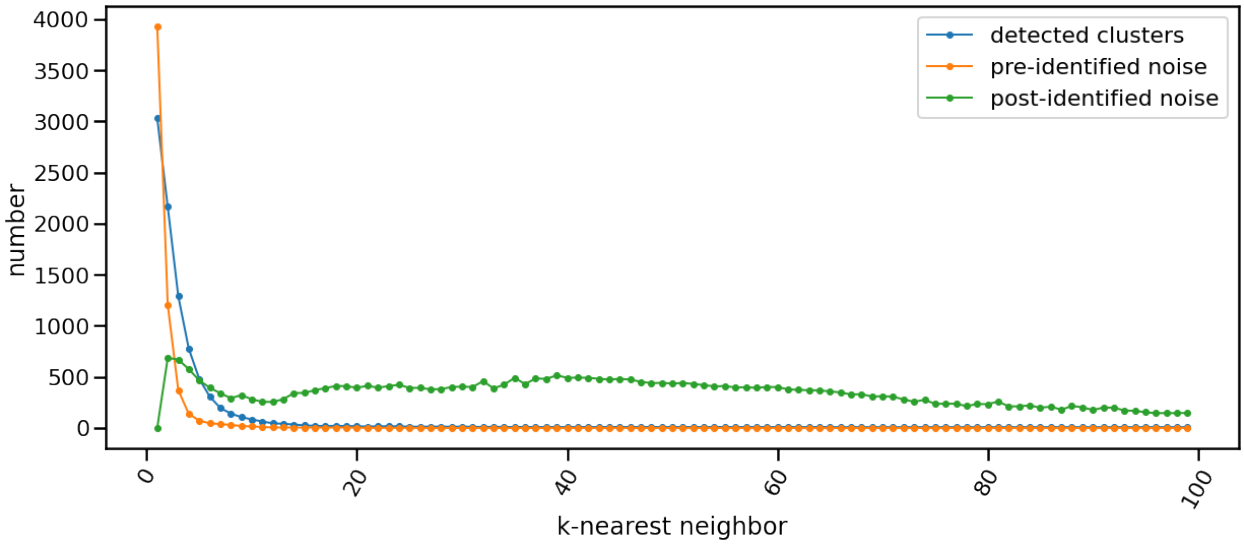



```
# Creating figure and axis objects using subplots()
fig, ax = plt.subplots(figsize=[20, 8])

x, y = zip(*detected_clusers)
ax.plot(x, y, marker='.', linewidth=2, label='detected clusters')
```

(continues on next page)

(continued from previous page)

```
x, y = zip(*noise_type1)
ax.plot(x, y, marker='.', linewidth=2, label='pre-identified noise')

x, y = zip(*noise_type2)
ax.plot(x, y, marker='.', linewidth=2, label='post-identified noise')


plt.xticks(rotation=60)
ax.set_xlabel('k-nearest neighbor')
ax.set_ylabel('number')
plt.legend()
plt.show()
```



CHAPTER

FOUR

PARTICIPATE IN COMPETITIONS

4.1 Validate Your Results

```
import pandas as pd
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 63, done.[K
remote: Counting objects: 100% (63/63), done.[K
remote: Compressing objects: 100% (52/52), done.[K
remote: Total 63 (delta 10), reused 59 (delta 9), pack-reused 0[K
Unpacking objects: 100% (63/63), done.
Checking out files: 100% (23/23), done.
```

You can get your validation results using 3 methods - by showing the Analyzer - extract values from the validity returned list from fit_predict function - extract values from the Analyzer dictionary

The algorithm is associated with five built-in validity measures, which are: - ACC, Accuracy - F1 score - NMI index (Normalized Mutual Information) - AMI index (Adjusted Mutual Information) - ARI index (Adjusted Rand Index)

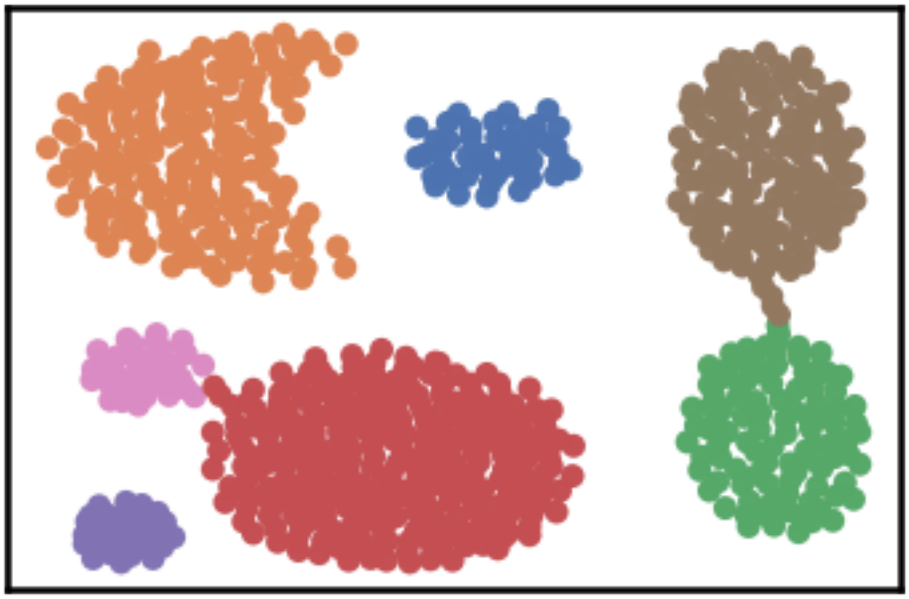
```
# Let us show the analyzer by set show_analyzer to True, which is actually the default␣
↪parameter's value

data_path = 'datasets/denmune/shapes/'
```

(continues on next page)

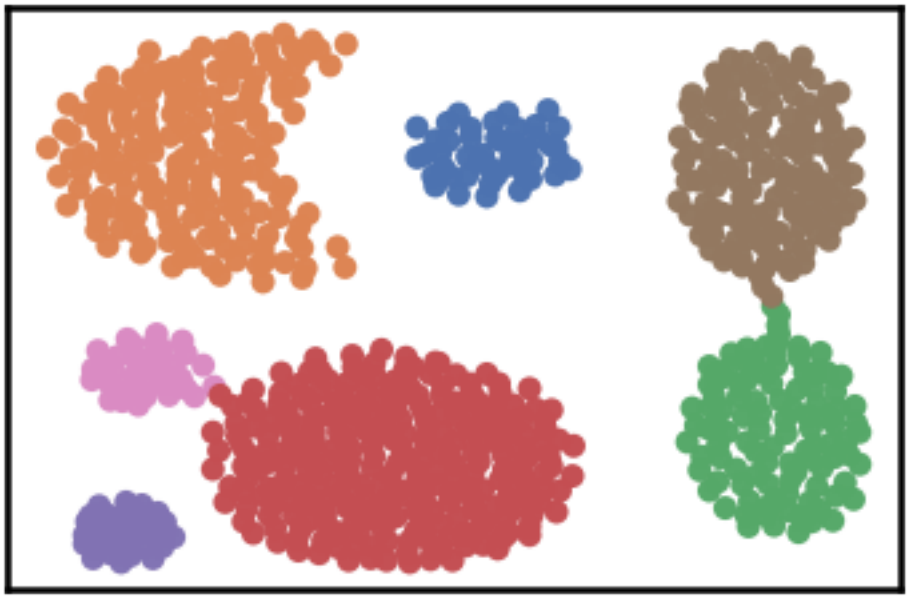
(continued from previous page)

```
dataset = "aggregation"
knn = 6

data_file = data_path + dataset + '.csv'
X_train = pd.read_csv(data_file, sep=',', header=None)
y_train = X_train.iloc[:, -1]
X_train = X_train.drop(X_train.columns[-1], axis=1)

print ("Dataset:", dataset)
dm = DenMune(train_data=X_train,
             train_truth=y_train,
             k_nearest=knn,
             rgn_tsne=False)

labels, validity = dm.fit_predict(show_noise=True, show_analyzer=True)
```

```
Dataset: aggregation
Plotting dataset Groundtruth
```



```
Plotting train data
```



```
Validating train data
├── exec_time
│   ├── DenMune: 0.322
│   ├── NGT: 0.038
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 7
│   └── detected: 7
├── n_points
│   ├── dim: 2
│   ├── noise
│   │   ├── type-1: 0
│   │   └── type-2: 0
│   ├── plot_size: 788
│   ├── size: 788
│   ├── strong: 492
│   └── weak
│       ├── all: 296
│       ├── failed to merge: 0
│       └── succeeded to merge: 296
└── validity
    └── train
        ├── ACC: 785
        ├── AMI: 0.988
        ├── ARI: 0.993
        ├── F1: 0.996
        ├── NMI: 0.988
        ├── completeness: 0.987
        └── homogeneity: 0.989
```

```
# secondly, we can extract validity returned list from fit_predict function
dm = DenMune(train_data=X_train, train_truth=y_train, k_nearest=knn, rgn_tsne=False)
```

(continues on next page)

(continued from previous page)

```
labels, validity = dm.fit_predict(show_plots=False, show_noise=True, show_analyzer=False)
validity
```

```
{'train': {'ACC': 785,
  'AMI': 0.9880984055236919,
  'ARI': 0.9927076502018027,
  'F1': 0.9962034083064701,
  'NMI': 0.9882680312048461,
  'completeness': 0.9873385166573364,
  'homogeneity': 0.9891992975556994}}
```

```
Accuracy = validity['train']['ACC']
print ('Accuracy:',Accuracy, 'correctely identified points')

F1_score = validity['train']['F1']
print ('F1 score:', round(F1_score*100,2), '%')

NMI = validity['train']['NMI']
print ('NMI index:', round(NMI*100,2), '%')

AMI = validity['train']['AMI']
print ('AMI index:', round(AMI*100,2), '%')

ARI = validity['train']['ARI']
print ('ARI index:', round(ARI*100,2), '%')
```

```
Accuracy: 785 correctely identified points
F1 score: 99.62 %
NMI index: 98.83 %
AMI index: 98.81 %
ARI index: 99.27 %
```

```
# Third, we can extract extract values from the Analyzer dictionary
dm = DenMune(train_data=X_train, train_truth=y_train, k_nearest=knn, rgn_tsne=False)
labels, validity = dm.fit_predict(show_plots=False, show_noise=True, show_analyzer=False)
dm.analyzer
```

```
{'exec_time': {'DenMune': 0.12747693061828613,
  'NGT': 0.016164064407348633,
  't_SNE': 0},
 'n_clusters': {'actual': 7, 'detected': 7},
 'n_points': {'dim': 2,
  'noise': {'type-1': 0, 'type-2': 0},
  'plot_size': 788,
  'size': 788,
  'strong': 492,
  'weak': {'all': 296, 'failed to merge': 0, 'succeeded to merge': 296}},
 'validity': {'train': {'ACC': 785,
   'AMI': 0.9880984055236919,
   'ARI': 0.9927076502018027,
   'F1': 0.9962034083064701,
```

(continues on next page)

(continued from previous page)

```
   'NMI': 0.9882680312048461,
   'completeness': 0.9873385166573364,
   'homogeneity': 0.9891992975556994}}}
```

```
Accuracy = dm.analyzer['validity']['train']['ACC']
print ('Accuracy:',Accuracy, 'correctely identified points')

F1_score = dm.analyzer['validity']['train']['F1']
print ('F1 score:', round(F1_score*100,2), '%')

NMI = dm.analyzer['validity']['train']['NMI']
print ('NMI index:', round(NMI*100,2), '%')

AMI = dm.analyzer['validity']['train']['AMI']
print ('AMI index:', round(AMI*100,2), '%')

ARI = dm.analyzer['validity']['train']['ARI']
print ('ARI index:', round(ARI*100,2), '%')
```

```
Accuracy: 785 correctely identified points
F1 score: 99.62 %
NMI index: 98.83 %
AMI index: 98.81 %
ARI index: 99.27 %
```

4.2 Trining MNIST Dataset

```
import pandas as pd
import numpy as np
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 63, done.[K
remote: Counting objects: 100% (63/63), done.[K
remote: Compressing objects: 100% (52/52), done.[K
remote: Total 63 (delta 10), reused 59 (delta 9), pack-reused 0[K
Unpacking objects: 100% (63/63), done.
```

```
# let us train the dataset

data_path = 'datasets/denmune/mnist/'
file_2d = data_path + 'mnist-2d.csv'

X_train = pd.read_csv(data_path + 'train.csv', sep=',')
X_test = pd.read_csv(data_path + 'test.csv',  sep=',')
y_train = X_train['label']
X_train = X_train.drop(['label'], axis=1)

validity_key = "F1"
scores = []
score = 0
best_knn = 0
best_score = -1

for knn in range (10, 50):   # knn ==> k-nearest neighbor, the only parameter required␣
↪by the algorithm
  dm = DenMune(train_data=X_train,
               train_truth=y_train,
               test_data=X_test,
               k_nearest=knn,
               file_2d=file_2d,
               rgn_tsne=False)

  labels, validity = dm.fit_predict(show_plots=False, show_analyzer=False)
  score = validity['train'][validity_key]

  if score > best_score:
    best_score = score
    best_knn = knn

  print ('k=' , knn, validity_key , 'score:', round(score*100,2) , '%, best score:', ␣
↪round(best_score*100,2) , '% at k=', best_knn)
  scores.append([knn, score])
```

```
k= 10 F1 score: 88.92 %, best score: 88.92 % at k= 10
k= 11 F1 score: 90.14 %, best score: 90.14 % at k= 11
k= 12 F1 score: 90.97 %, best score: 90.97 % at k= 12
k= 13 F1 score: 91.97 %, best score: 91.97 % at k= 13
k= 14 F1 score: 92.3 %, best score: 92.3 % at k= 14
k= 15 F1 score: 92.58 %, best score: 92.58 % at k= 15
k= 16 F1 score: 91.77 %, best score: 92.58 % at k= 15
k= 17 F1 score: 94.07 %, best score: 94.07 % at k= 17
k= 18 F1 score: 92.89 %, best score: 94.07 % at k= 17
```

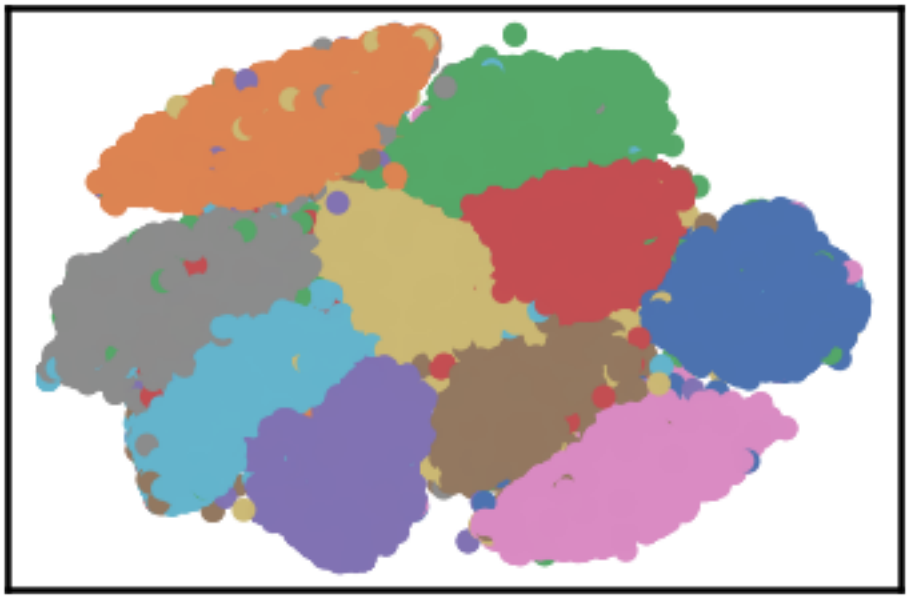
(continues on next page)

(continued from previous page)

```
k= 19 F1 score: 93.7 %, best score: 94.07 % at k= 17
k= 20 F1 score: 94.57 %, best score: 94.57 % at k= 20
k= 21 F1 score: 80.54 %, best score: 94.57 % at k= 20
k= 22 F1 score: 93.18 %, best score: 94.57 % at k= 20
k= 23 F1 score: 94.91 %, best score: 94.91 % at k= 23
k= 24 F1 score: 94.43 %, best score: 94.91 % at k= 23
k= 25 F1 score: 95.1 %, best score: 95.1 % at k= 25
k= 26 F1 score: 93.68 %, best score: 95.1 % at k= 25
k= 27 F1 score: 93.09 %, best score: 95.1 % at k= 25
k= 28 F1 score: 83.83 %, best score: 95.1 % at k= 25
k= 29 F1 score: 83.42 %, best score: 95.1 % at k= 25
k= 30 F1 score: 84.41 %, best score: 95.1 % at k= 25
k= 31 F1 score: 76.45 %, best score: 95.1 % at k= 25
k= 32 F1 score: 65.25 %, best score: 95.1 % at k= 25
k= 33 F1 score: 64.28 %, best score: 95.1 % at k= 25
k= 34 F1 score: 64.12 %, best score: 95.1 % at k= 25
k= 35 F1 score: 73.24 %, best score: 95.1 % at k= 25
k= 36 F1 score: 74.07 %, best score: 95.1 % at k= 25
k= 37 F1 score: 84.26 %, best score: 95.1 % at k= 25
k= 38 F1 score: 96.14 %, best score: 96.14 % at k= 38
k= 39 F1 score: 96.16 %, best score: 96.16 % at k= 39
k= 40 F1 score: 85.34 %, best score: 96.16 % at k= 39
k= 41 F1 score: 72.7 %, best score: 96.16 % at k= 39
k= 42 F1 score: 85.17 %, best score: 96.16 % at k= 39
k= 43 F1 score: 85.19 %, best score: 96.16 % at k= 39
k= 44 F1 score: 85.46 %, best score: 96.16 % at k= 39
k= 45 F1 score: 84.78 %, best score: 96.16 % at k= 39
k= 46 F1 score: 84.55 %, best score: 96.16 % at k= 39
k= 47 F1 score: 84.47 %, best score: 96.16 % at k= 39
k= 48 F1 score: 96.22 %, best score: 96.22 % at k= 48
k= 49 F1 score: 84.11 %, best score: 96.22 % at k= 48
```

```
# now let us use our best_knn which corresponds to best_score for our test data␣
↪prediction
#best_knn ==> 48

dm = DenMune(train_data=X_train,
             train_truth=y_train,
             test_data=X_test,
             k_nearest=best_knn,
             file_2d=file_2d,
             rgn_tsne=False)

labels, validity = dm.fit_predict(show_plots=True, show_analyzer=True)
```

```
Plotting dataset Groundtruth
```



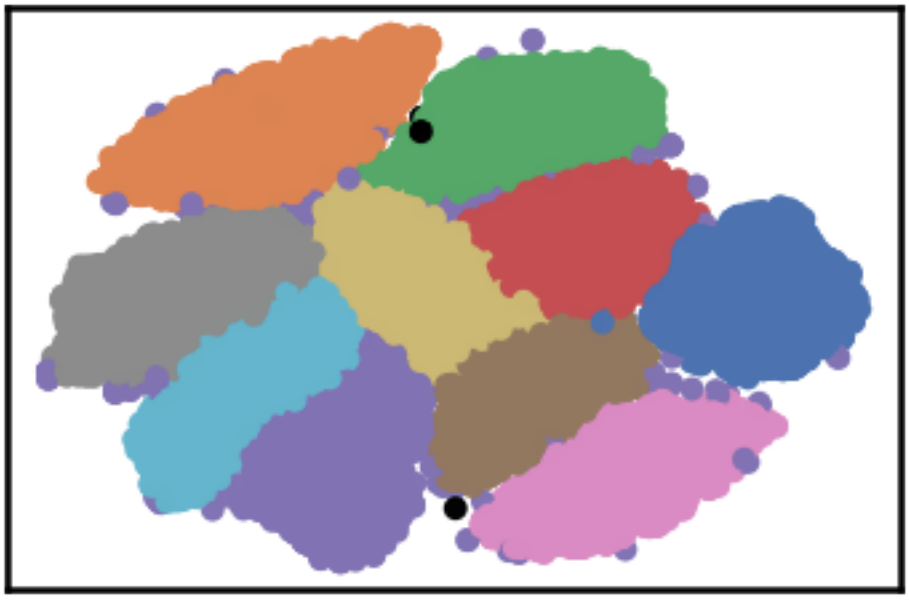
```
Plotting train data
```

```
Validating train data
├── exec_time
│   ├── DenMune: 95.723
│   ├── NGT: 8.561
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 10
│   └── detected: 11
├── n_points
```

(continues on next page)

(continued from previous page)

```
    ├── dim: 784
    ├── noise
    │   ├── type-1: 5
    │   └── type-2: 0
    ├── plot_size: 42000
    ├── size: 70000
    ├── strong: 37447
    └── weak
        ├── all: 32553
        ├── failed to merge: 0
        └── succeeded to merge: 32553
└── validity
    └── train
        ├── ACC: 40386
        ├── AMI: 0.905
        ├── ARI: 0.919
        ├── F1: 0.962
        ├── NMI: 0.905
        ├── completeness: 0.904
        └── homogeneity: 0.906

Plotting test data
```



```
# prepare our output to be submitted to the dataset kaggle competition
ImageID = np.arange(len(X_test))+1
Out = pd.DataFrame([ImageID,labels['test']]).T
Out.to_csv('submission.csv', header =  ['ImageId', 'Label' ], index = None)
```

4.3 Become a Kaggler: Get 97% on MNIST Dataset

```
import pandas as pd
import numpy as np
import time
import os.path

import warnings
warnings.filterwarnings('ignore')
```

```
# install DenMune clustering algorithm using pip command from the offecial Python␣
↪repository, PyPi
# from https://pypi.org/project/denmune/
!pip install denmune

# then import it
from denmune import DenMune
```

```
# clone datasets from our repository datasets
if not os.path.exists('datasets'):
  !git clone https://github.com/egy1st/datasets
```

```
Cloning into 'datasets'...
remote: Enumerating objects: 63, done.[K
remote: Counting objects: 100% (63/63), done.[K
remote: Compressing objects: 100% (52/52), done.[K
remote: Total 63 (delta 10), reused 59 (delta 9), pack-reused 0[K
Unpacking objects: 100% (63/63), done.
```

```
# let us train the dataset

data_path = 'datasets/denmune/mnist/'
file_2d = data_path + 'mnist-2d.csv'

X_train = pd.read_csv(data_path + 'train.csv', sep=',')
X_test = pd.read_csv(data_path + 'test.csv',  sep=',')
y_train = X_train['label']
X_train = X_train.drop(['label'], axis=1)

validity_key = "F1"
scores = []
score = 0
best_knn = 0
best_score = -1

for knn in range (10, 50):   # knn ==> k-nearest neighbor, the only parameter required␣
↪by the algorithm
  dm = DenMune(train_data=X_train,
               train_truth=y_train,
               test_data=X_test,
               k_nearest=knn,
```

(continues on next page)

(continued from previous page)

```
            file_2d=file_2d,
            rgn_tsne=False)

  labels, validity = dm.fit_predict(show_plots=False, show_analyzer=False)
  score = validity['train'][validity_key]

  if score > best_score:
    best_score = score
    best_knn = knn

  print ('k=' , knn, validity_key , 'score:', round(score*100,2) , '%, best score:', ␣
↪round(best_score*100,2) , '% at k=', best_knn)
  scores.append([knn, score])
```

```
k= 10 F1 score: 88.92 %, best score: 88.92 % at k= 10
k= 11 F1 score: 90.14 %, best score: 90.14 % at k= 11
k= 12 F1 score: 90.97 %, best score: 90.97 % at k= 12
k= 13 F1 score: 91.97 %, best score: 91.97 % at k= 13
k= 14 F1 score: 92.3 %, best score: 92.3 % at k= 14
k= 15 F1 score: 92.58 %, best score: 92.58 % at k= 15
k= 16 F1 score: 91.77 %, best score: 92.58 % at k= 15
k= 17 F1 score: 94.07 %, best score: 94.07 % at k= 17
k= 18 F1 score: 92.89 %, best score: 94.07 % at k= 17
k= 19 F1 score: 93.7 %, best score: 94.07 % at k= 17
k= 20 F1 score: 94.57 %, best score: 94.57 % at k= 20
k= 21 F1 score: 80.54 %, best score: 94.57 % at k= 20
k= 22 F1 score: 93.18 %, best score: 94.57 % at k= 20
k= 23 F1 score: 94.91 %, best score: 94.91 % at k= 23
k= 24 F1 score: 94.43 %, best score: 94.91 % at k= 23
k= 25 F1 score: 95.1 %, best score: 95.1 % at k= 25
k= 26 F1 score: 93.68 %, best score: 95.1 % at k= 25
k= 27 F1 score: 93.09 %, best score: 95.1 % at k= 25
k= 28 F1 score: 83.83 %, best score: 95.1 % at k= 25
k= 29 F1 score: 83.42 %, best score: 95.1 % at k= 25
k= 30 F1 score: 84.41 %, best score: 95.1 % at k= 25
k= 31 F1 score: 76.45 %, best score: 95.1 % at k= 25
k= 32 F1 score: 65.25 %, best score: 95.1 % at k= 25
k= 33 F1 score: 64.28 %, best score: 95.1 % at k= 25
k= 34 F1 score: 64.12 %, best score: 95.1 % at k= 25
k= 35 F1 score: 73.24 %, best score: 95.1 % at k= 25
k= 36 F1 score: 74.07 %, best score: 95.1 % at k= 25
k= 37 F1 score: 84.26 %, best score: 95.1 % at k= 25
k= 38 F1 score: 96.14 %, best score: 96.14 % at k= 38
k= 39 F1 score: 96.16 %, best score: 96.16 % at k= 39
k= 40 F1 score: 85.34 %, best score: 96.16 % at k= 39
k= 41 F1 score: 72.7 %, best score: 96.16 % at k= 39
k= 42 F1 score: 85.17 %, best score: 96.16 % at k= 39
k= 43 F1 score: 85.19 %, best score: 96.16 % at k= 39
k= 44 F1 score: 85.46 %, best score: 96.16 % at k= 39
k= 45 F1 score: 84.78 %, best score: 96.16 % at k= 39
k= 46 F1 score: 84.55 %, best score: 96.16 % at k= 39
k= 47 F1 score: 84.47 %, best score: 96.16 % at k= 39
```

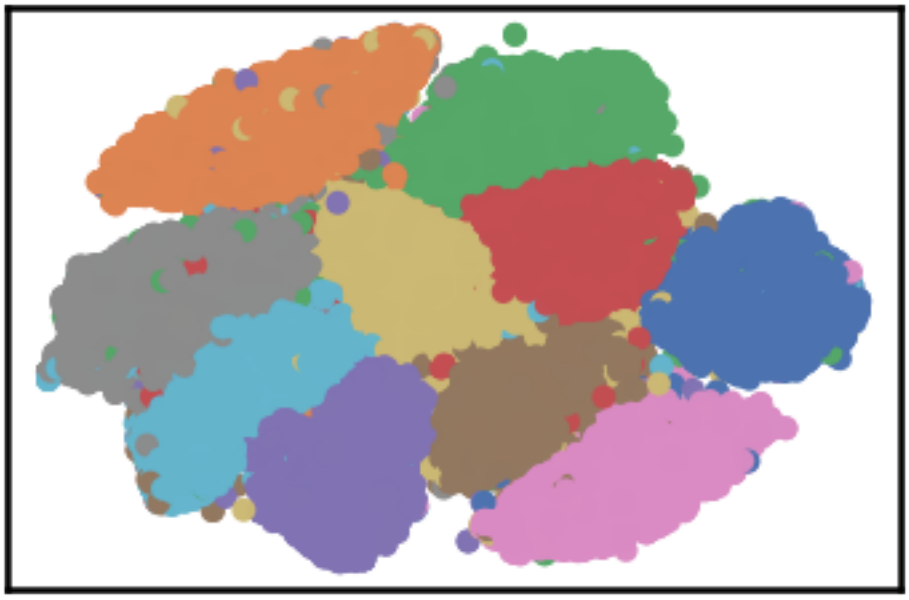
(continues on next page)

(continued from previous page)

```
k= 48 F1 score: 96.22 %, best score: 96.22 % at k= 48
k= 49 F1 score: 84.11 %, best score: 96.22 % at k= 48
```

```
# now let us use our best_knn which corresponds to best_score for our test data␣
↪prediction
#best_knn ==> 48

dm = DenMune(train_data=X_train,
             train_truth=y_train,
             test_data=X_test,
             k_nearest=best_knn,
             file_2d=file_2d,
             rgn_tsne=False)

labels, validity = dm.fit_predict(show_plots=True, show_analyzer=True)
```

```
Plotting dataset Groundtruth
```



```
Plotting train data
```

```
Validating train data
├── exec_time
│   ├── DenMune: 95.723
│   ├── NGT: 8.561
│   └── t_SNE: 0
├── n_clusters
│   ├── actual: 10
│   └── detected: 11
├── n_points
│   ├── dim: 784
│   ├── noise
│   │   ├── type-1: 5
│   │   └── type-2: 0
│   ├── plot_size: 42000
│   ├── size: 70000
│   ├── strong: 37447
│   └── weak
│       ├── all: 32553
│       ├── failed to merge: 0
│       └── succeeded to merge: 32553
└── validity
    └── train
        ├── ACC: 40386
        ├── AMI: 0.905
        ├── ARI: 0.919
        ├── F1: 0.962
        ├── NMI: 0.905
        ├── completeness: 0.904
        └── homogeneity: 0.906

Plotting test data
```

```
# prepare our output to be submitted to the dataset kaggle competition
ImageID = np.arange(len(X_test))+1
Out = pd.DataFrame([ImageID,labels['test']]).T
Out.to_csv('submission.csv', header =  ['ImageId', 'Label' ], index = None)
```