ChemPlot

Release 1.2.1

Dajt Mullaj, Murat Cihan Sorkun

Sep 27, 2022

USER MANUAL:

Date: Sep 27, 2022 **Version**: 1.2.1

In the last decades, Machine Learning (ML) applications have had a great impact on molecular and material science. However, every ML model requires a definition of its applicability domain. We developed a python package, Chemplot, that allows users to plot the chemical space of their datasets. Chemplot contains smart algorithms behind which uses both structural and tailored similarity. Moreover, it is easy to use even for non-experts. For details on the background of ChemPlot you can find here our paper.

This guide provides the user with the explanantion of ChemPlot concepts and functionality.

CHAPTER

ONE

INSTALLATION

1.1 Installing an official release

There two different options you can follow to install ChemPlot.

1.1.1 Option 1: Use conda

You can install ChemPlot using conda. To install ChemPlot, at the command line, run:

```
~$ conda install -c conda-forge chemplot
```

1.1.2 Option 2: Use pip

An alternative method is to install is using pip:

```
~$ pip install chemplot
```

Note: ChemPlot requires RDKit, which cannot be installed using pip. The official RDKit documentation contains installation instructions for multiple platforms.

1.2 Verify Installation

You can verify that ChemPlot was installed on your local computer by running:

```
~$ pip show chemplot
Name: chemplot
...
```

If instead of what is shown above your output is:

```
WARNING: Package(s) not found: chemplot
```

ChemPlot was not installed correctly or your system cannot find the path to it. If ChemPlot is installed correctly you can also test the package by running:

```
~$ pip install pytest
~$ python -m pytest --pyargs chemplot
```

These will run all the library tests against your installation. For every official release from *1.2.0* you can use this command to verify that every function of your local installation of ChemPlot works as expected.

CHAPTER

TWO

HOW TO USE CHEMPLOT

ChemPlot is a cheminformatics tool whose purpose is to visualize subsets of the chemical space in two dimensions. It uses the RDKit chemistry framework, the scikit-learn API and the umap-learn API.

2.1 Getting started

To demonstrate how to use the functions the library offers we will use a BBBP (blood-brain barrier penetration)[1] molecular dataset. This is a set of molecules encoded as SMILES, which have been assigned a binary label according to their permeability properties. This dataset can be loaded as a *pandas <https://pandas.pydata.org/pandas-docs/stable/index.html>* `_ DataFrame object.

```
from pandas import read_csv

data_BBBP = read_csv("BBBP.csv")
```

To visualize the molecules in 2D according to their similarity it is first needed to construct a `Plotter` object. This is the class containing all the functions ChemPlot uses to produce the desired visualizations. A `Plotter` object can be constructed using classmethods, which differentiate between the type of input that is feed to the object. In our example we need to use the method from_smiles. We pass three parameters: the list of SMILES from the BBBP dataset, their target values (the binary labels) and the target type (in this case "C", which stands for "Classification").

```
from chemplot import Plotter

cp = Plotter.from_smiles(data_BBBP["smiles"], target=data_BBBP["target"], target_type="C
↪")
```

2.2 Plotting the results

When the `Plotter` object was constructed, descriptors for each SMILES were calculated, using the library mordred, and then selected based on the target values. We reduce the number of dimensions for each molecule from the number of descriptors selected to only 2. ChemPlot uses three different algorithms in order to achieve this. In this example we will first use t-SNE[2].

```
cp.tsne()
```

[1] **Martins, Ines Filipa, et al.** (2012). A Bayesian approach to in silico blood-brain barrier penetration modeling. Journal of chemical information and modeling 52.6, 1686-1697

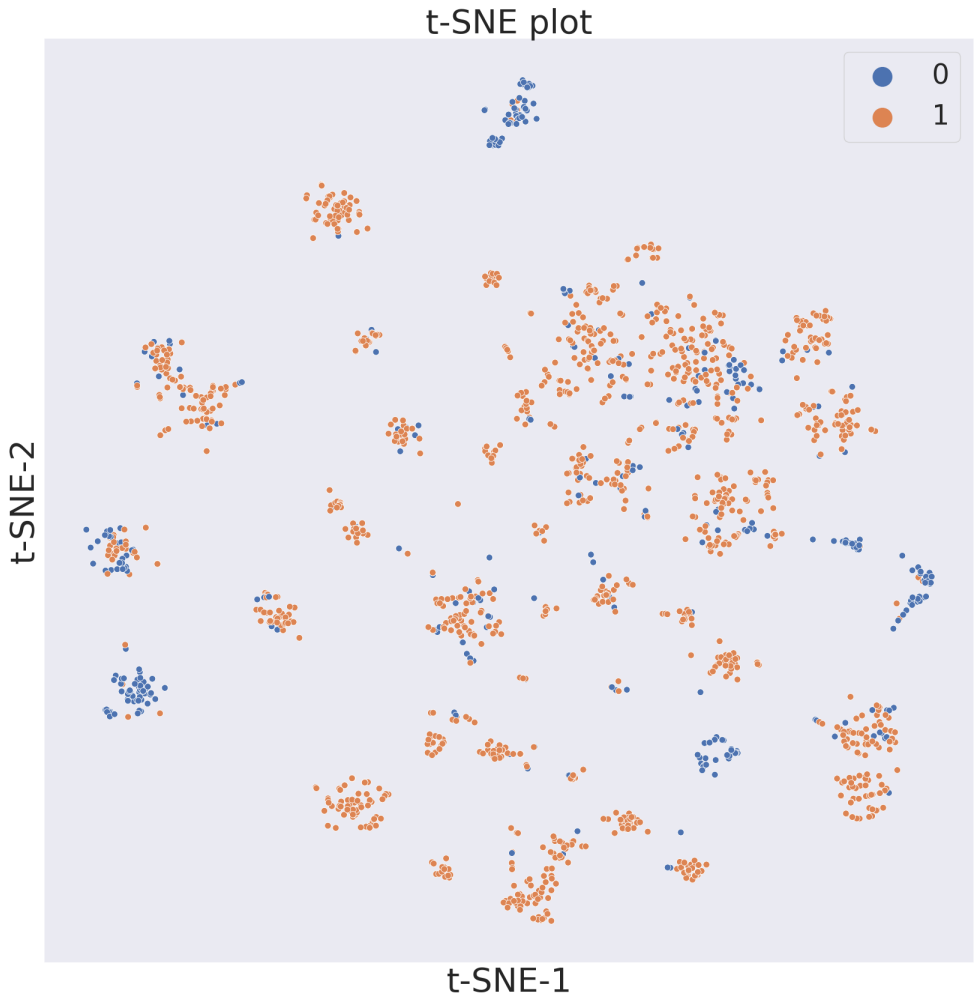
[2] **van der Maaten, Laurens, Hinton, Geoffrey.** (2008). Viualizingdata using t-SNE. Journal of Machine Learning Research. 9. 2579-2605.

The output will be a dataframe containg the reduced dimensions and the target values.

t-SNE-1	t-SNE-2	target
-41.056122	0.355575	1
-35.535915	21.648867	1
23.771597	-14.438373	1

To now visualize the chemical space of the dataset we use `visualize_plot()`.

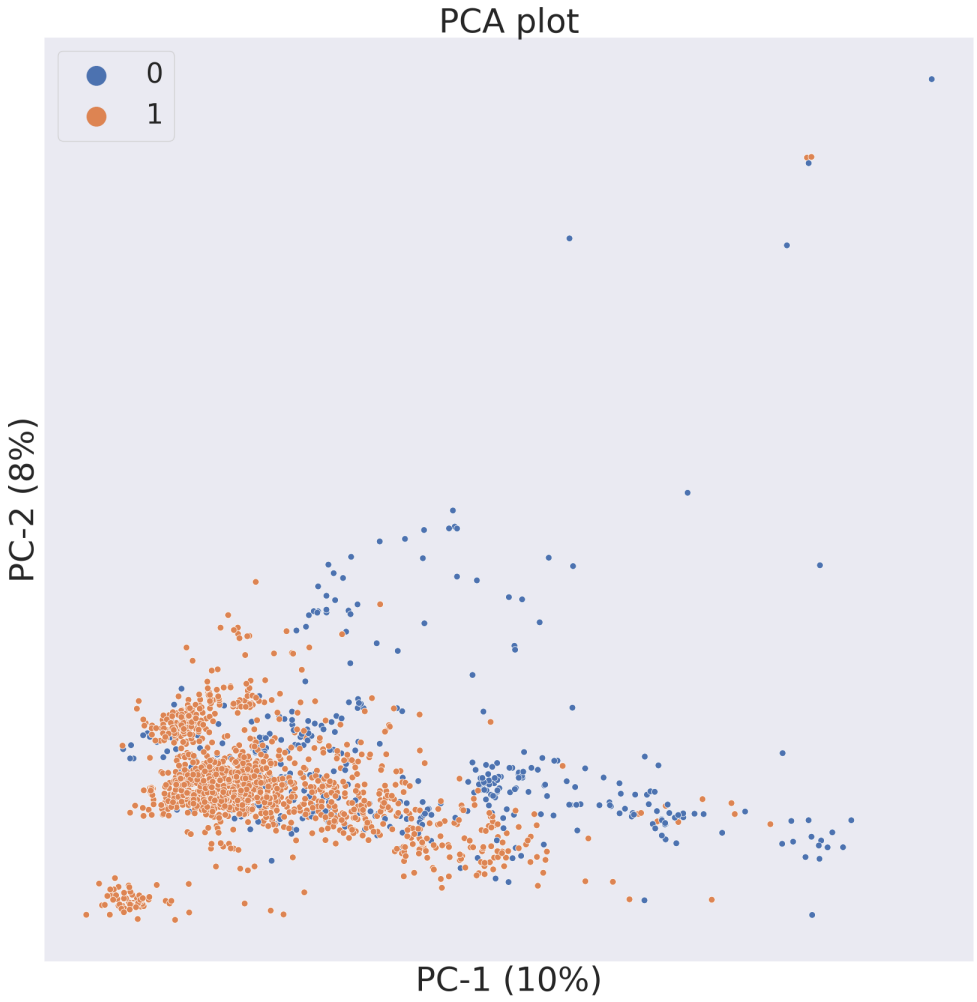
```
import matplotlib.pyplot as plt

cp.visualize_plot()
```



The second figure shows the results obtained by reducing the dimensions of features Principal Component Analysis

(PCA)[3].

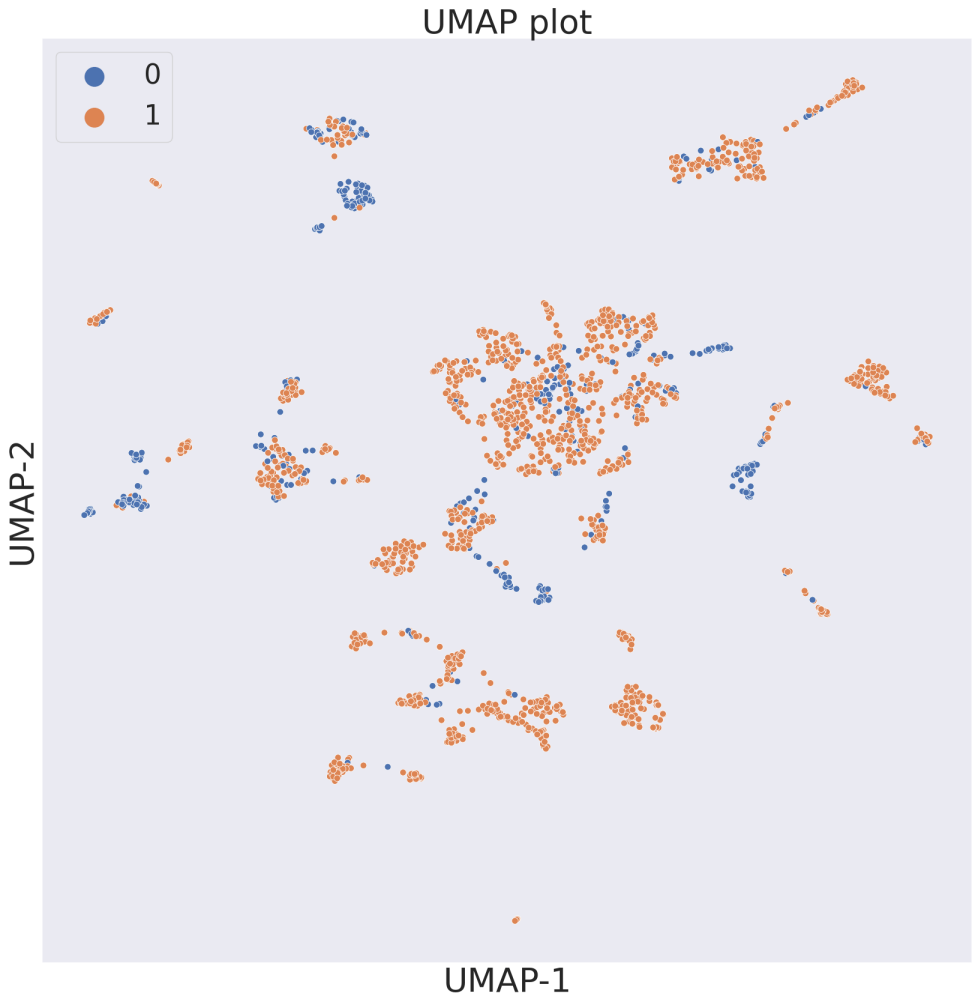
```
cp.pca()
cp.visualize_plot()
```



The third figure shows the results obtained by reducing the dimensions of features by UMAP[4].

```
cp.umap()
cp.visualize_plot()
```

[3] **Wold, S., Esbensen, K., Geladi, P.** (1987). Principal component analysis. Chemometrics and intelligent laboratory systems. 2(1-3). 37-52.
[4] **McInnes, L., Healy, J., Melville, J.** (2018). Umap: Uniform manifold approximation and projection for dimension reduction. arXivpreprint arXiv:1802.03426.



In each figure the molecules are coloured by class value.

References:

CHAPTER

THREE

SIMILARITY ANALYSIS

In this example we will use two molecular datasets: the BBBP (blood-brain barrier penetration) dataset[1], already used in the previous section, and the BACE (-secretase inhibitors) dataset[2]. While the target values of the molecules collected by the BBBP dataset are binary, and therefore discrete, the target values of the molecules collected by the BACE dataset are continuous.

```
from chemplot import Plotter, load_data

data_BBBP = load_data("BBBP")
data_BACE = load_data("BACE")
```

In order to plot a subset of the chemical space over a 2D graph it is necessary to define the metric according to which a certain molecule will be plotted on a certain location of the graph. What ChemPlot uses when deciding which molecules need to be plotted where is the concept of "molecular similarity". Similar molecules will be displayed closer together, while molecules which are less similar will be displayed further apart.

ChemPlot distinguishes between two definitions of molecular similarity: structural and tailored[3].

3.1 Structural

Structural similarity is defined as the number and dimensions of "fragments" different molecules share. Molecular fragments are groups of atoms and bonds which a molecule can be divided into. The higher the number and dimensions of fragments two molecules share the more similar they are according to structural similarity. ChemPlot uses Extended-Connectivity Fingerprints (ECFPs)[4] to define which fragments are present in each molecule. To create a `Plotter` object which visualizes the desired molecules according to structural similarity we need to pass the keyword "structural" as the `sim_type` parameter when constructing the object.

```
cp_BBBP = Plotter.from_smiles(data_BBBP["smiles"], target=data_BBBP["target"], target_
↪type="C", sim_type="structural")
cp_BACE = Plotter.from_smiles(data_BACE["smiles"], target=data_BACE["target"], target_
↪type="R", sim_type="structural")
```

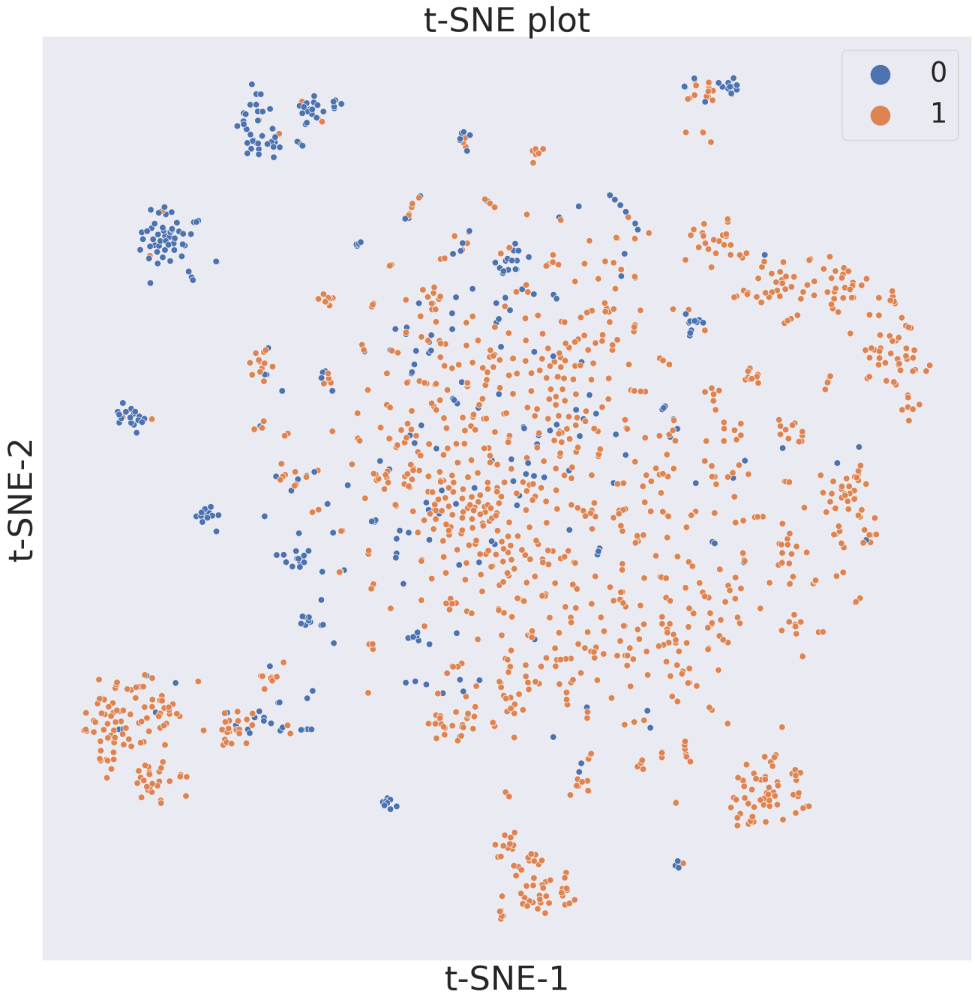
```
cp_BBBP.tsne()
cp_BBBP.visualize_plot()
```

[1] **Martins, Ines Filipa, et al.** (2012). A Bayesian approach to in silico blood-brain barrier penetration modeling. Journal of chemical information and modeling 52.6, 1686-1697

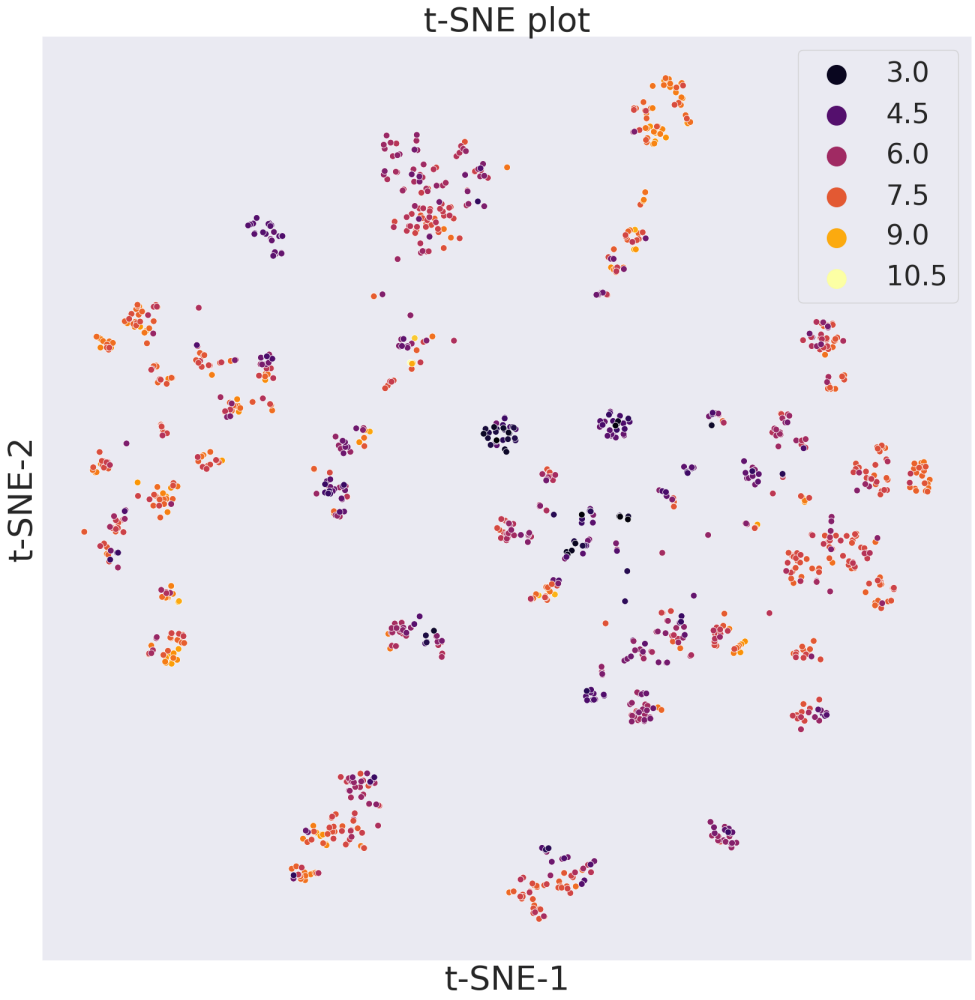
[2] **Subramanian, Govindan, et al.** (2016). Computational modeling of -secretase 1 (BACE-1) inhibitors using ligand based approaches. Journal of chemical information and modeling 56.10, 1936-1949.

[3] **Basak, S.C. and Grunwald, G.D.** (1995) Predicting mutagenicity of chemicals using topological and quantum chemical parameters: a similarity based study. Chemosphere 31, 2529–2546

[4] **Rogers, D., Hahn, M.** (2010).** Extended-connectivity fingerprints. Journal of chemical information and modeling, 50(5), 742-754.




```
cp_BACE.tsne()
cp_BACE.visualize_plot()
```



3.2 Tailored

Tailored similarity is a similarity metric between molecules which takes into account the target property for determining if two molecules are similar or not. Indeed after a general set of descriptors is calculated for each molecule, a subset of those is selected by optimizing for the target property. Finally depending on the values of the subset ChemPlot can decide which molecules are more similar than others. To create a `Plotter` object which visualizes the desired molecules according to structural similarity, we need to pass the keyword "tailored" as the `sim_type` parameter when constructing the object. Since "tailored" is the default value of `sim_type` if a list of target values is passed in construction, in the following example we could have omitted the last parameter and still have got the same objects.

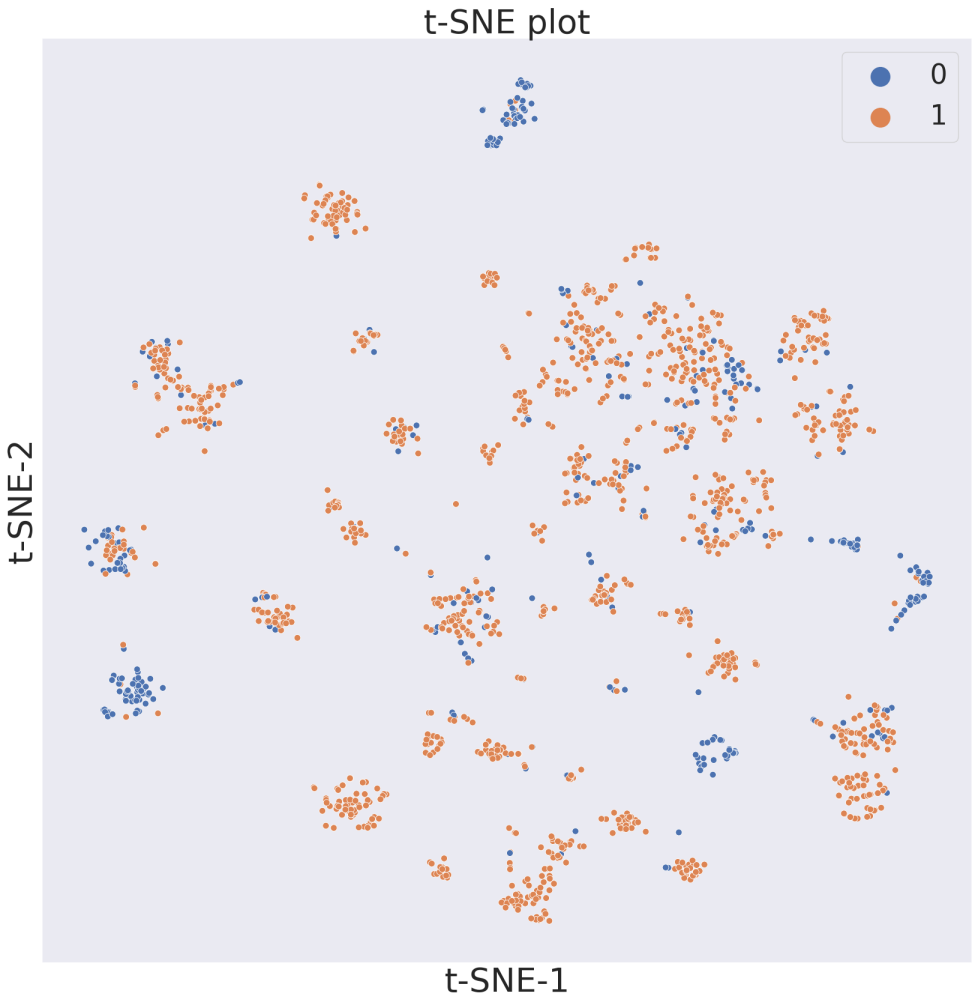
```
cp_BBBP = Plotter.from_smiles(data_BBBP["smiles"], target=data_BBBP["target"], target_
↪type="C", sim_type="tailored")
cp_BACE = Plotter.from_smiles(data_BACE["smiles"], target=data_BACE["target"], target_
```

(continues on next page)

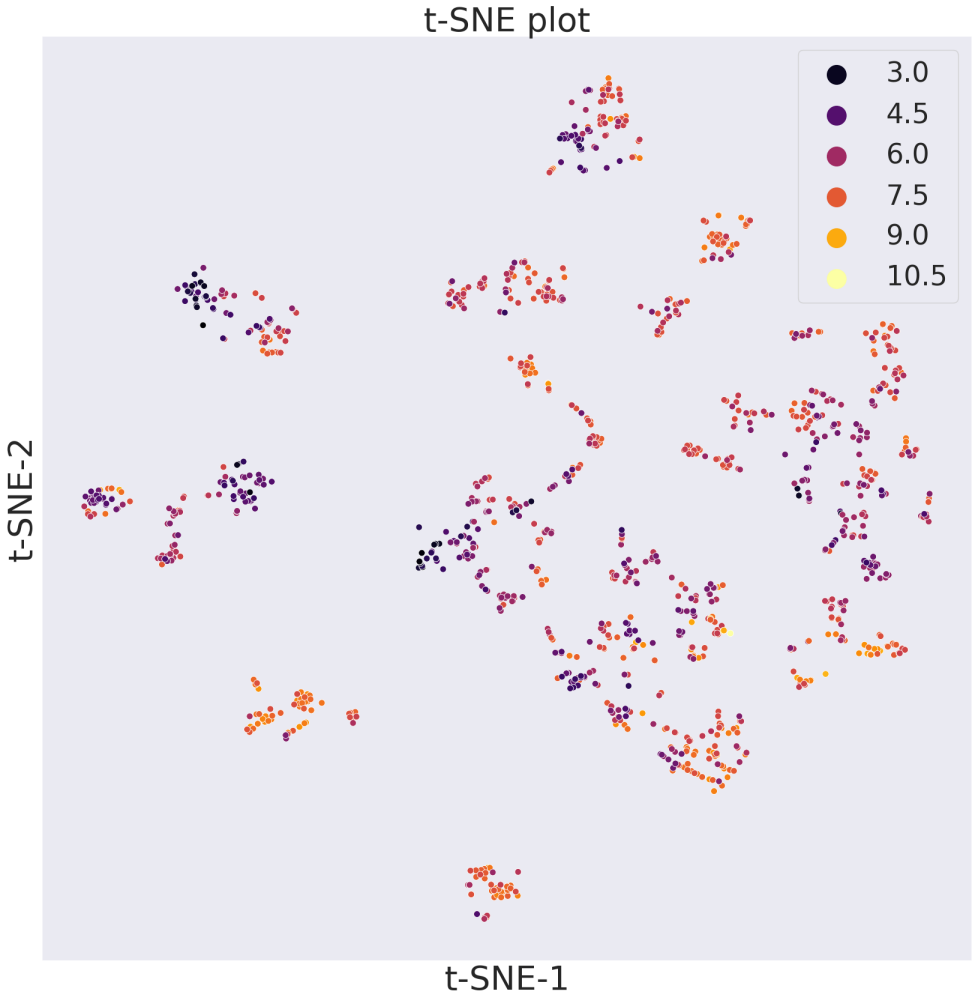
(continued from previous page)

```
↪type="R", sim_type="tailored")
```

```
cp_BBBP.tsne()
cp_BBBP.visualize_plot()
```




```
cp_BACE.tsne()
cp_BACE.visualize_plot()
```



References:

CHAPTER

FOUR

DIMENSIONALITY REDUCTION

ChemPlot uses different machine learning techniques to reduce the number of dimensions, or features, of each molecule to only two in order to then create 2D graphs. These algorithms are: PCA[1], t-SNE[2] and UMAP[3].

For the following examples we will use two molecular datasets, already mentioned in the previous section: the BBBP (blood-brain barrier penetration) dataset[4] and the BACE (-secretase inhibitors) dataset[5].

```
from chemplot import Plotter, load_data

data_BBBP = load_data("BBBP")
data_BACE = load_data("BACE")
cp_BBBP = Plotter.from_smiles(data_BBBP["smiles"], target=data_BBBP["target"], target_
↪type="C")
cp_BACE = Plotter.from_smiles(data_BACE["smiles"], target=data_BACE["target"], target_
↪type="R")
```

4.1 PCA

ChemPlot uses PCA from the scikit-learn package to compute the two principal components of the molecular dataset. PCA allows for time efficient results and for a visualization which gives a global view of the data.

```
cp_BBBP.pca()
cp_BBBP.visualize_plot()
```

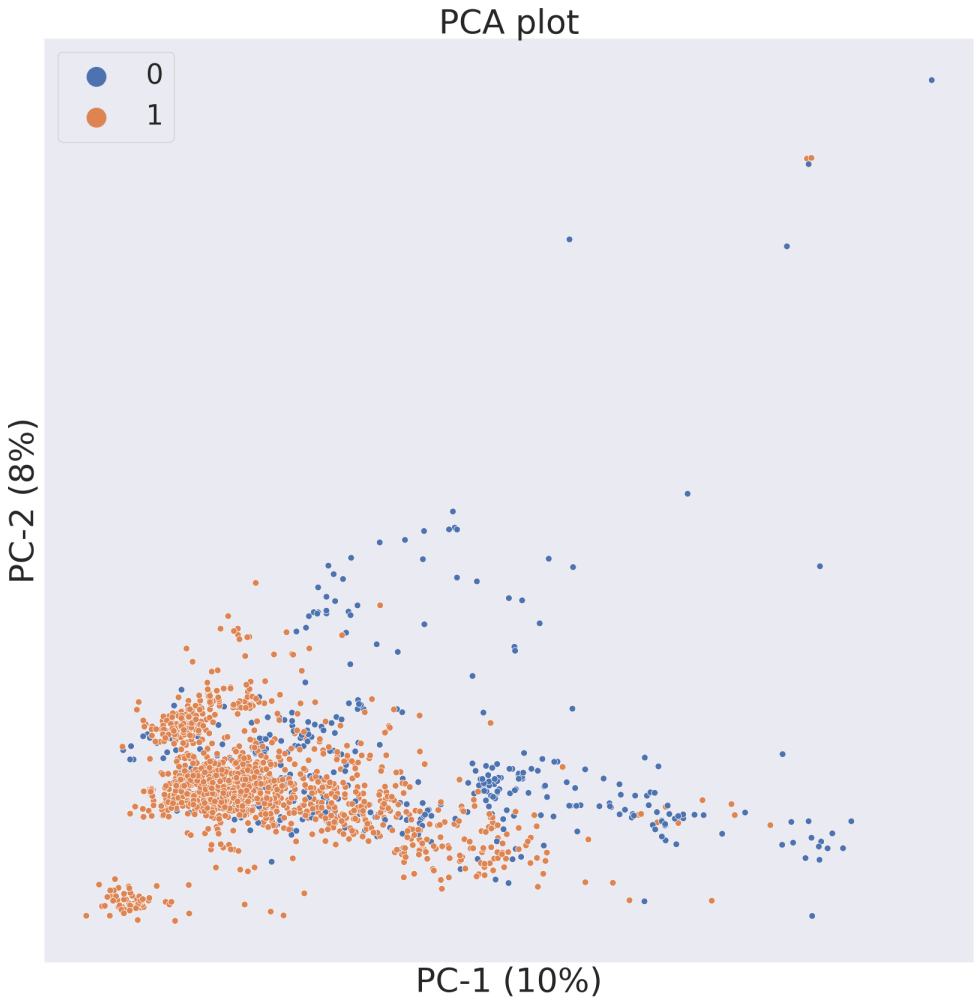
[1] **Wold, S., Esbensen, K., Geladi, P.** (1987). Principal component analysis. Chemometrics and intelligent laboratory systems. 2(1-3). 37-52.

[2] **van der Maaten, Laurens, Hinton, Geoffrey.** (2008). Viualizingdata using t-SNE. Journal of Machine Learning Research. 9. 2579-2605.

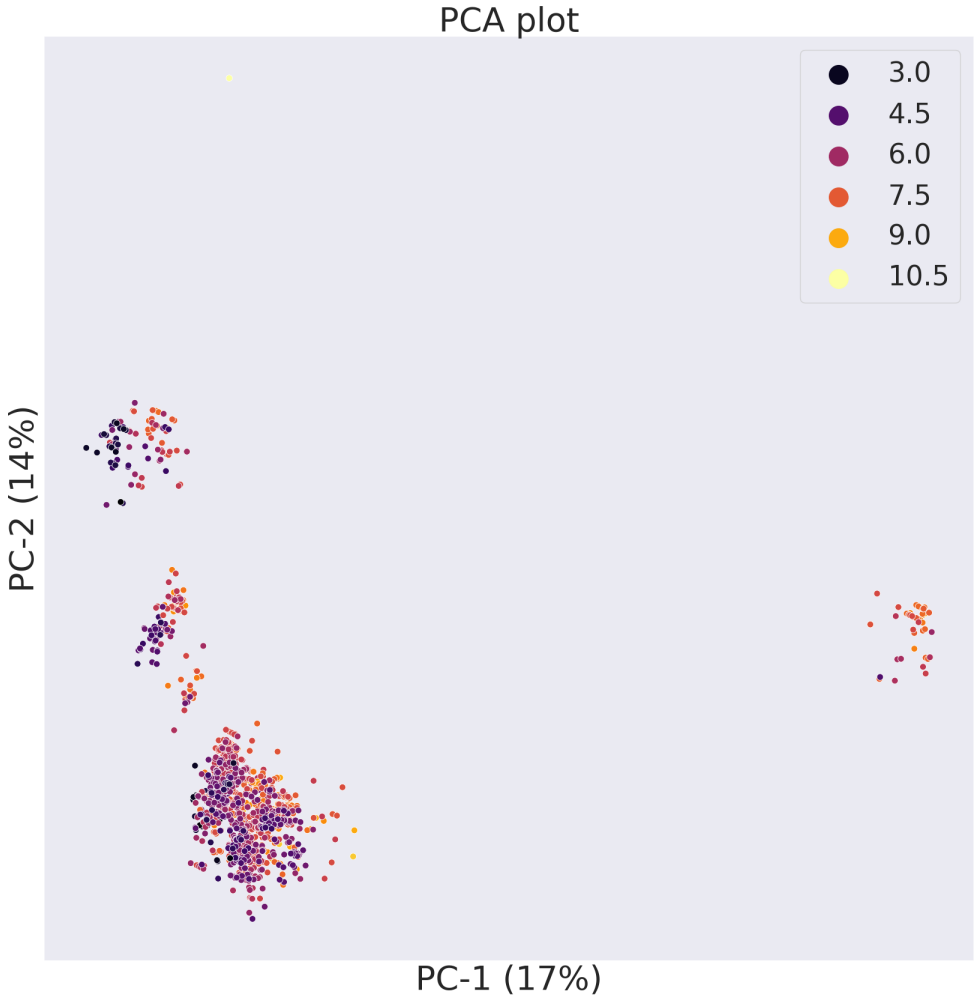
[3] **McInnes, L., Healy, J., Melville, J.** (2018). Umap: Uniform manifold approximation and projection for dimension reduction. arXivpreprint arXiv:1802.03426.

[4] **Martins, Ines Filipa, et al.** (2012). A Bayesian approach to in silico blood-brain barrier penetration modeling. Journal of chemical information and modeling 52.6, 1686-1697

[5] **Subramanian, Govindan, et al.** (2016). Computational modeling of -secretase 1 (BACE-1) inhibitors using ligand based approaches. Journal of chemical information and modeling 56.10, 1936-1949.



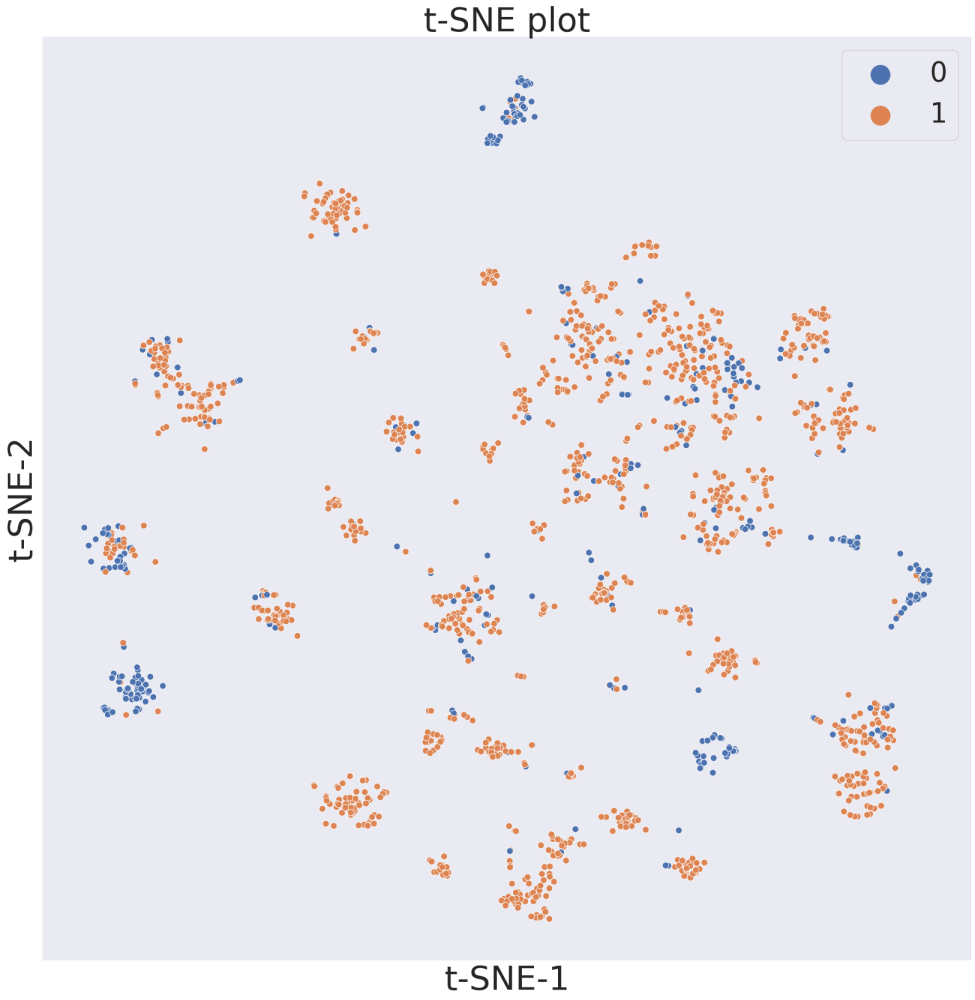

```
cp_BACE.pca()
cp_BACE.visualize_plot()
```



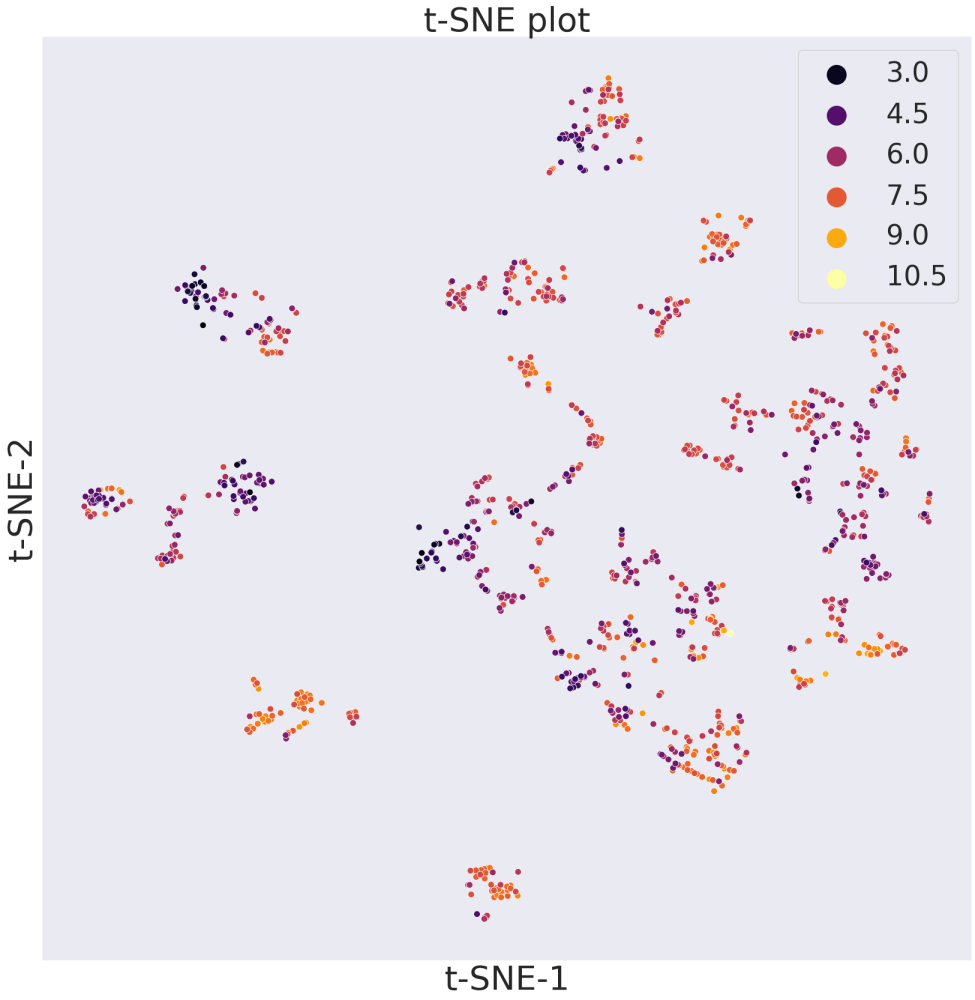
4.2 t-SNE

ChemPlot uses t-SNE from the scikit-learn package to reduce to only 2 the number of features of the molecular dataset. t-SNE looks at local neighbourhoods of molecules when it is reducing their dimensions. In this way the local structure of the dataset is better preserved, while the global structure is mostly lost when plotting the results in a 2D graph. However because of the locality preservation that t-SNE offers it is possible to visualize well-defined clusters of similar molecules that exhibit similar properties.

```
cp_BBBP.tsne()
cp_BBBP.visualize_plot()
```




```
cp_BACE.tsne()
cp_BACE.visualize_plot()
```

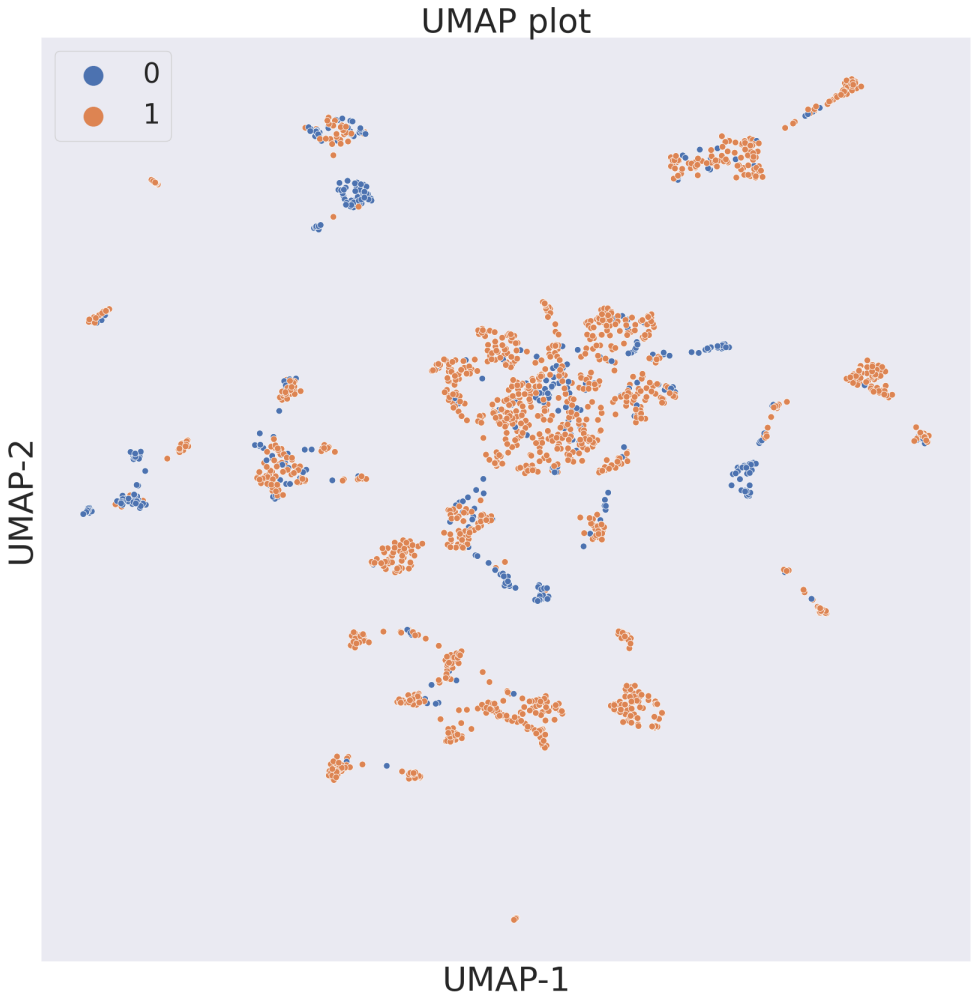


Two important parameters of the `tsne()` method are `perplexity` and `pca`. The former is a positive integer parameter which defines the size of the neighbourhoods the algorithm will look for when analyzing the dataset. The higher the value of `perplexity` the wider the analyzed neighbourhoods. The recommended values for `perplexity` range from 5 to 50. The `pca` parameter is a Boolean value which indicates if the data has to be preprocessed with PCA. Its value is taken into account when plotting according to structural similarities when each molecule is encoded with a long number of features. Since t-SNE is computationally expensive, preprocessing the data can save substantial amounts of time when generating plots, at the cost of losing some of the molecular structural information.

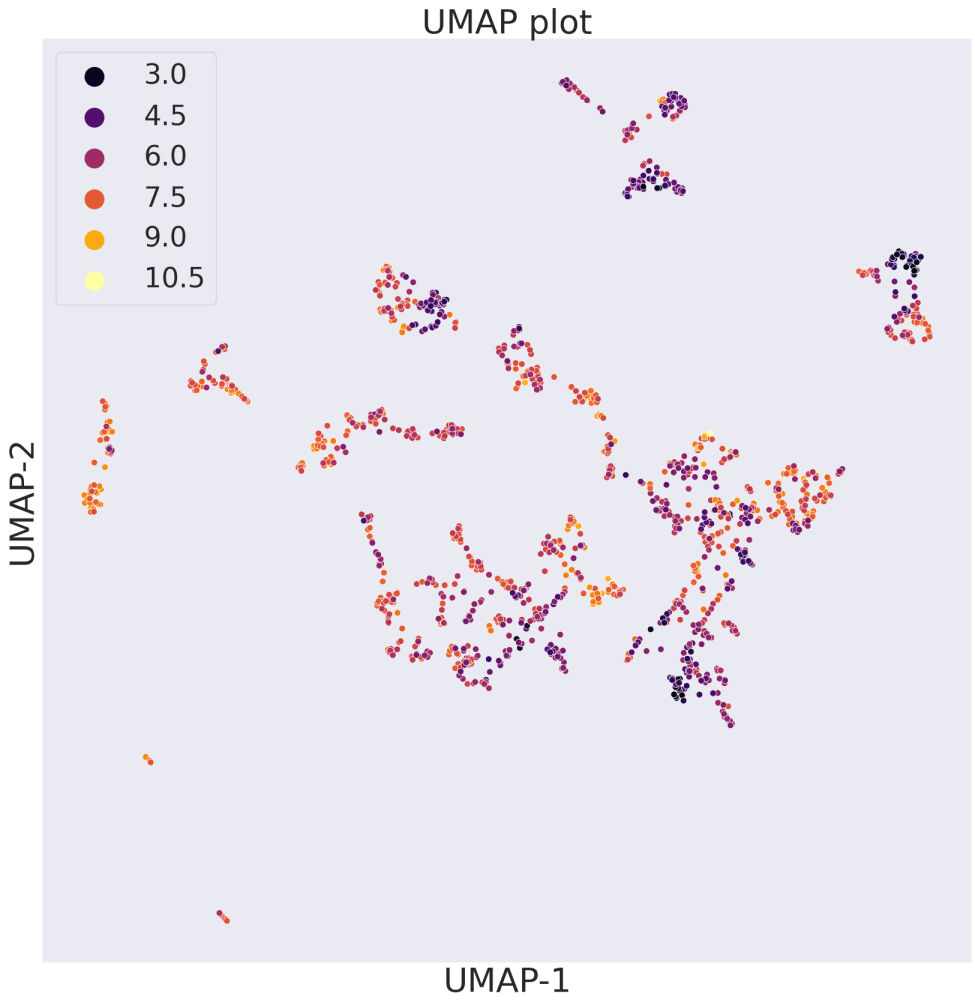
4.3 UMAP

ChemPlot uses UMAP from the umap-learn package to reduce to only 2 the number of features of the molecular dataset. As t-SNE, UMAP looks at local neighbourhoods of molecules when it is reducing their dimensions. While this also results in 2D clusters of locally similar molecules, compared to t-SNE, UMAP retains more of the global structure of the dataset. Compared to t-SNE, furthermore, UMAP is much more computationally efficient and faster.

```
cp_BBBP.umap()
cp_BBBP.visualize_plot()
```




```
cp_BACE.umap()
cp_BACE.visualize_plot()
```



Two important parameters of the `umap()` method are `n_neighbors`, `min_dist` and `pca`. The former is a positive integer parameter which constrains the size of the local neighbourhood the algorithm will look for when analyzing the dataset. Low values of `n_neighbors` will make ChemPlot visualize very local structures. The `min_dist` parameter is a value which ranges from 0.0 to 0.99. It provides the minimum distance apart that points are allowed to be in the 2D graph. The `pca` parameter is a Boolean value which indicates if the data has to be preprocessed with PCA.

References:

CHAPTER

FIVE

VISUALIZE THE CHEMICAL SPACE

ChemPlot can generate two types of plots for a given chemical space: static and interactive.

For the following examples we will use the BBBP (blood-brain barrier penetration) dataset[1].

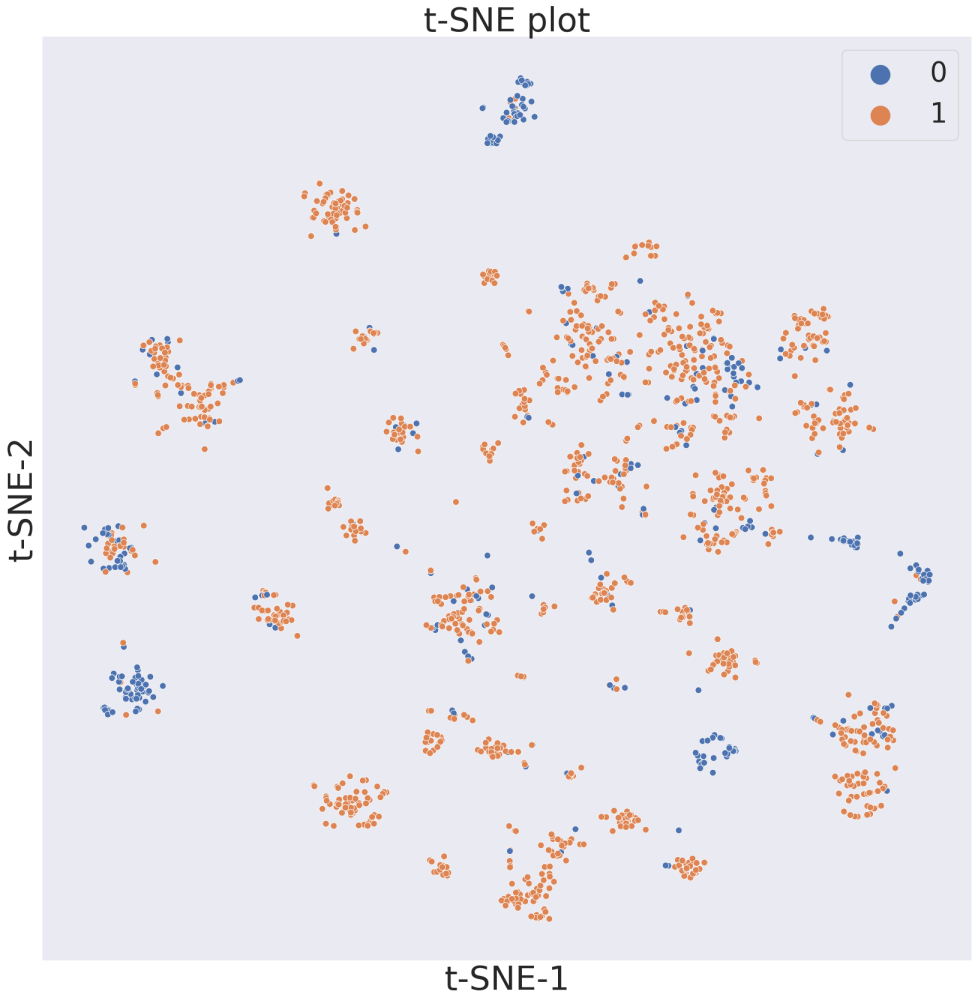
```
from chemplot import Plotter, load_data

data = load_data("BBBP")
cp = Plotter.from_smiles(data["smiles"], target=data["target"], target_type="C")
```

5.1 Static Plot

To generate a static plot first reduce the dimensions of the molecules used to initialize the `Plotter` instance. Then you can use `visualize_plot()` to generate a static visualization of the chemical space.

```
import matplotlib.pyplot as plt

cp.tsne()
cp.visualize_plot()
```

[1] **Martins, Ines Filipa, et al.** (2012). A Bayesian approach to in silico blood-brain barrier penetration modeling. Journal of chemical information and modeling 52.6, 1686-1697



5.2 Interactive Plot

To generate an interactive plot first reduce the dimensions of the molecules used to initialize the `Plotter` instance. Then you can use `interactive_plot()` to generate an interactive visualization of the chemical space.

```
cp.interactive_plot(show_plot=True)
```

The interactive plot is generated using the library bokeh. You can interact with it by using the toolbar displayed on the top right of the visualization. You can navigate across the plot, select group of molecules, zoom in and out the visualization and save the plot as an image. Furthermore you can hover over the molecules to see their 2D image.

References:

CHAPTER

SIX

CLUSTERING DATA

ChemPlot allows you to identify different clusters in you data by making use of the KMeans[1] algorithm as implemented in sklearn. To illustrate its implementation in ChemPlot we will load the LOGP dataset[2], a dataset about Lipophilicity with continuos targets. Let's load the sample dataset and create a `Plotter` object.

```
from chemplot import Plotter, load_data

data = load_data("LOGP")
cp = Plotter.from_smiles(data["smiles"], target=data["target"], target_type="R")
```

Let's then reduce the dimensions of the molecular descriptors.

```
cp.umap(random_state=500)
```

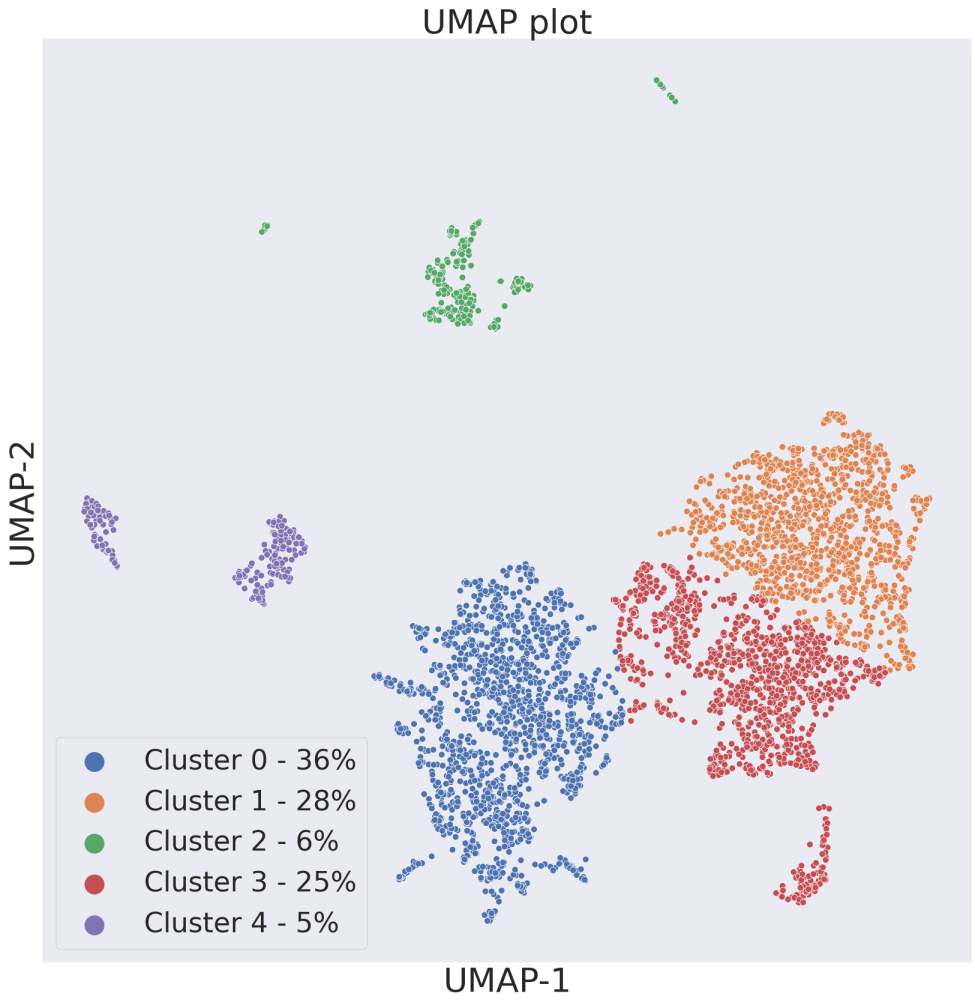
Now that the dimensions are reduced we can plot the image as shown in the previous chapters. We can also, however, identify some clusters in the data by calling this function:

```
cp.cluster()
```

`cluster()` will identify the clusters in our reduced dataset by using KMeans. The function takes one parameter `n_clusters`, identifying the number of clusters we want to see. By default `n_clusters` is 5. Once we clustered the data we can call `visualize_plot(clusters=True)` to see the plot. Notice how we need to pass a parameter `clusters` set to `True` in order to see the clusters in the resulting image.

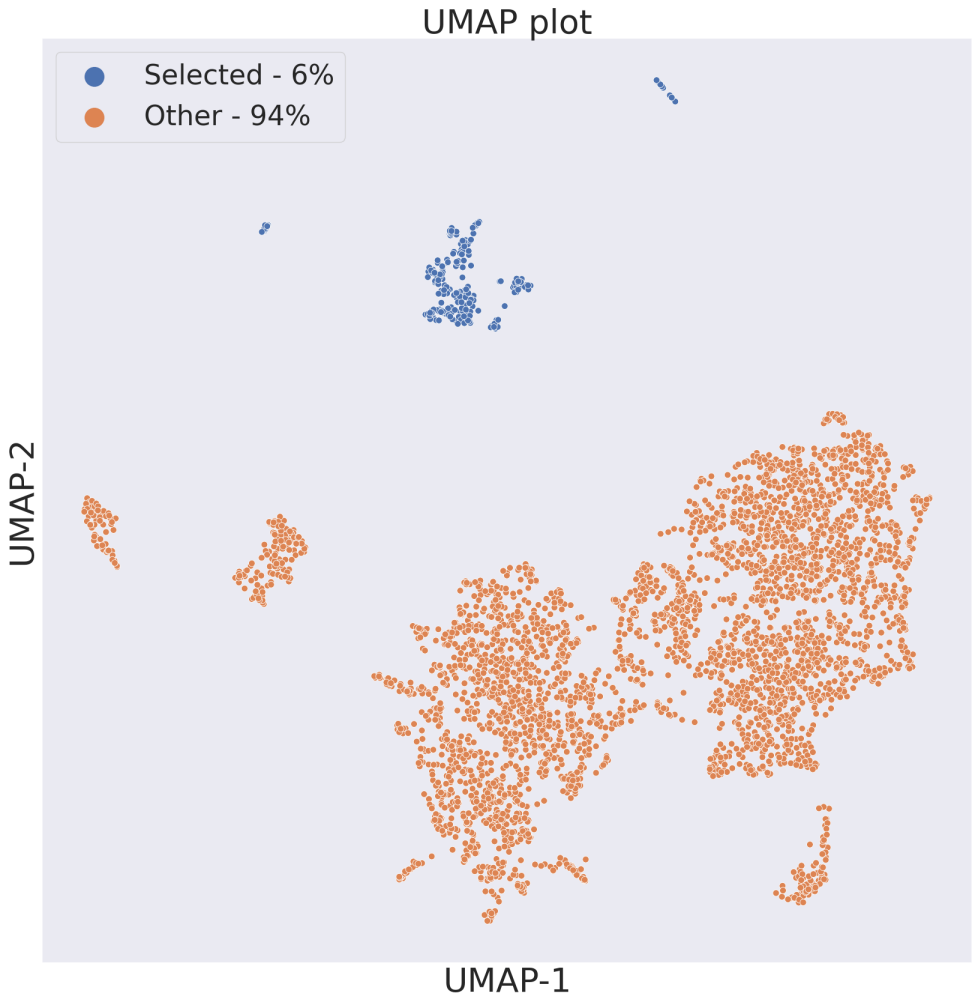
```
cp.visualize_plot(clusters=True)
```

[1] **Lloyd, Stuart P.** (1982). Least square quantization in PCM. IEEE Transactions on Information Theory. 28 (2): 129–137.
[2] **Hersey, A.** (2015) ChEMBL Deposited Data Set - AZ dataset

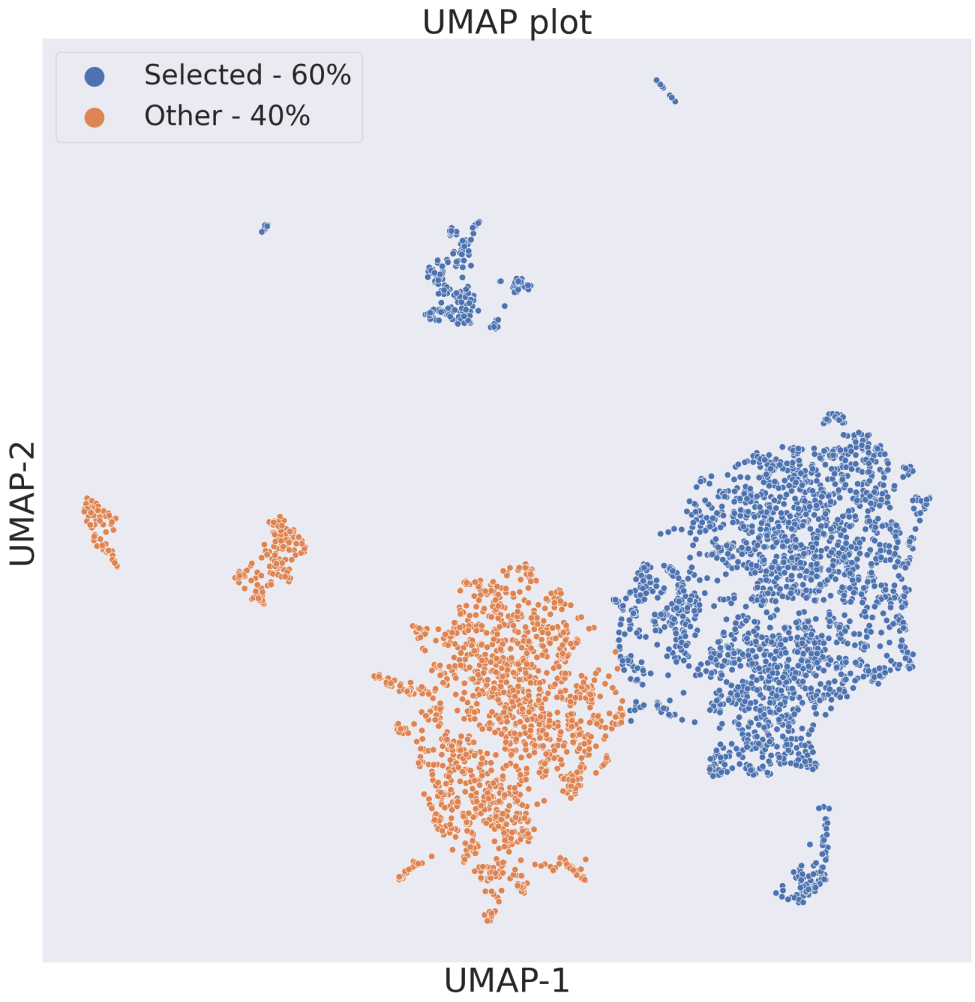


We can however also select a number of clusters we want to highlight. The parameter `clusters` in `visualize_plot()` can indeed also be a `list` of integers or an `int` itself. An integer represents one of the clusters identified in the previous steps. ChemPlot will either read the list or the single number passed as a parameter and highlight those clusters as selected.

```
cp.visualize_plot(clusters=2)
```




```
cp.visualize_plot(clusters=[1,2,3])
```



We can also use `interactive_plot()` to visualize the clusters. In these case pass `clusters=True` to generate a bokeh plot with two tabs. The first tab will contain the plot that would have been generated also without clustering. The second tab will contain a plot showing the different clusters. Click on the elements of the legend to mute a cluster's data points.

```
cp.interactive_plot(clusters=True)
```

References:

CHAPTER

SEVEN

ADDITIONAL FEATURES

ChemPlot offers additional features for chemical space visualization which can improve the understanding of the underlying similarities between the investigated molecules.

Using the `Plotter` object it is possible to create two different kind plots of the chemical space, aside from the scatterplots showed in the previous sections. These plots investigate the density distribution of the chemical space an are hexagonal bin plot and the kernel density estimate plot.

To show the before mentioned features we will use the BBBP (blood-brain barrier penetration) dataset[1], already mentioned in the previous section:

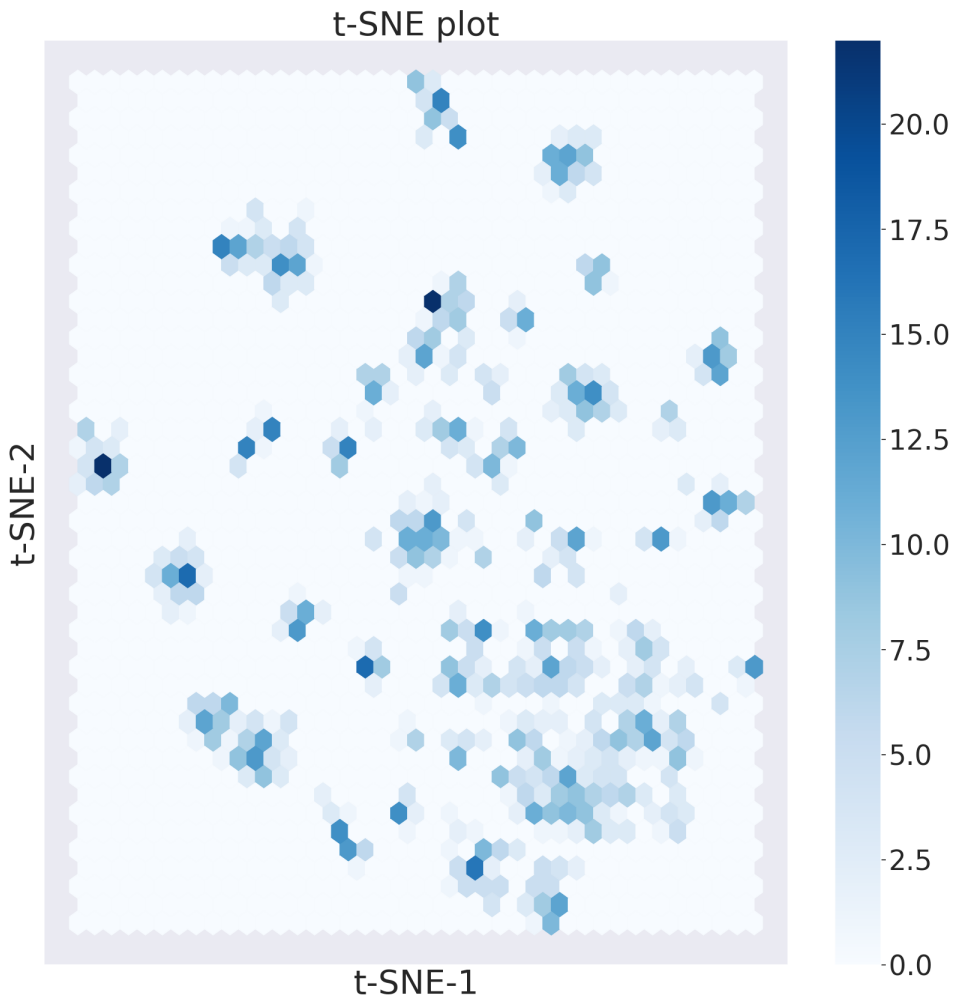
```
from chemplot import Plotter, load_data

data_BBBP = load_data("BBBP")
cp_BBBP = Plotter.from_smiles(data_BBBP["smiles"], target=data_BBBP["target"], target_
↪type="C")
```

7.1 Hexagonal Bin Plot

In a hexagonal bin plot points are binned into hexagons, which in turn are coloured depending on the count of observations they cover. To create a hexagonal bin plot we need to pass the keyword "hex" as the `kind` parameter when visualizing the plot.

```
cp_BBBP.tsne(random_state=0)
cp_BBBP.visualize_plot(kind="hex")
```

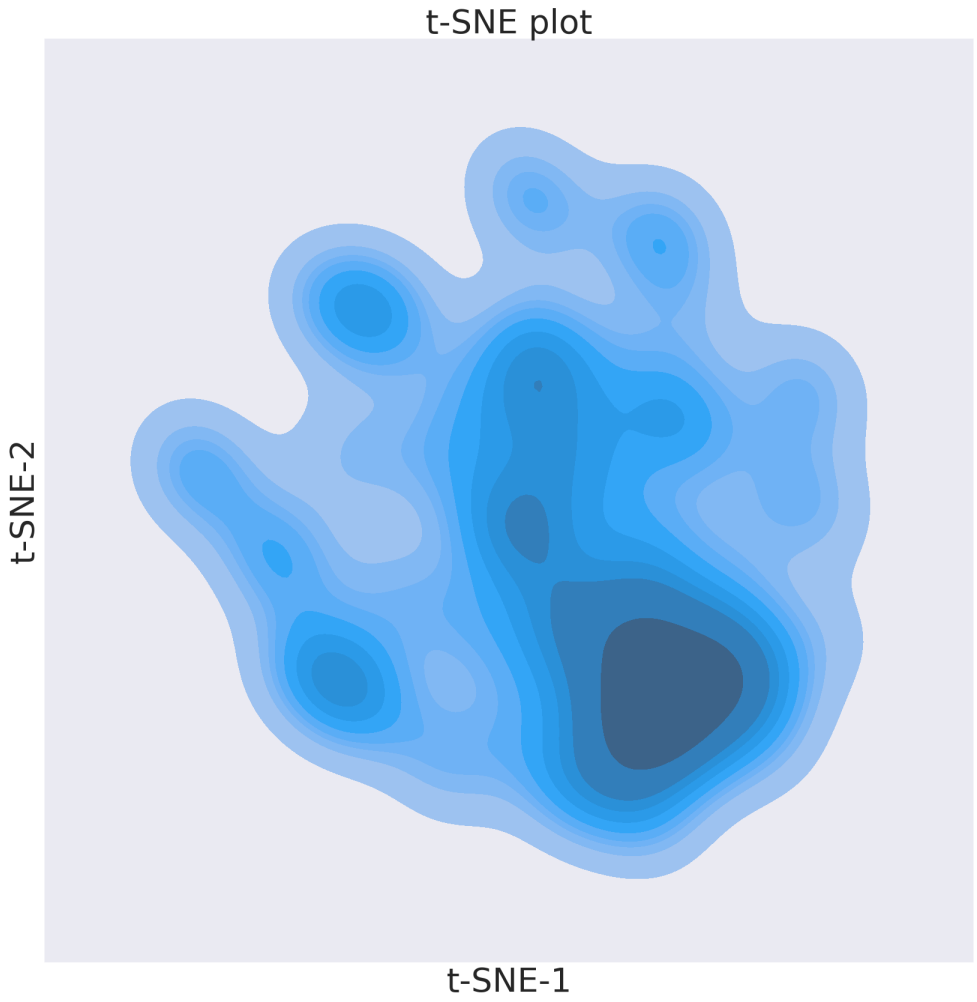
[1] **Martins, Ines Filipa, et al.** (2012). A Bayesian approach to in silico blood-brain barrier penetration modeling. Journal of chemical information and modeling 52.6, 1686-1697



7.2 Kernel Density Estimate Plot

In a kernel density estimate plot, the data distribution is visualized by a continuous probability density curve which in our case is in 2 dimensions. To create a kernel density estimate plot we need to pass the keyword "kde" as the `kind` parameter when visualizing the plot.

```
cp_BBBP.visualize_plot(kind="kde")
```



References:

CHAPTER

EIGHT

SAMPLE DATASETS

ChemPlot provides some sample datasets that can be used to get started right away with exploring the libraries features. These datasets can be loaded with the following function:

```
from chemplot import load_data

df = load_data("BBBP")
```

In these case we are loading the BBBP dataset, used in the previous sections of this manual. `load_data()` returns a pandas DataFrame built using the sample dataset provided as a parameter. Chemplot contains the following sample datasets:

ID	Name	Type	Size
C_1478_CLINTOX_2	Clintox (Toxicity)[1][2][3][4]	Classification	1478
C_1513_BACE_2	BACE (Inhibitor)[5]	Classification	1513
C_2039_BBBP_2	BBBP (Blood-brain barrier penetration)[6]	Classification	2039
C_41127_HIV_3	HIV[7]	Classification	41127
R_642_SAMPL	SAMPL (Hydration free energy)[8]	Regression	642
R_1513_BACE	BACE (Binding affinity)[5]	Regression	1513
R_4200_LOGP	LOGP (Lipophilicity)[9]	Regression	4200
R_1291_LOGS	LOGS (Aqueous Solubility)[10]	Regression	1291
R_9982_AQSOLDB	AQSOLDB (Aqueous Solubility)[11]	Regression	9982

The datasets ID are constructed in the following way:

Name Formatting: type_size_name_num_of_classes.csv

[1] **Gayvert, Kaitlyn M., Neel S. Madhukar, and Olivier Elemento.** (2016) *A data-driven approach to predicting successes and failures of clinical trials.* Cell chemical biology 23.10 1294-1301.

[2] **Artemov, Artem V., et al.** (2016) *Integrated deep learned transcriptomic and structure-based predictor of clinical trials outcomes.* bioRxiv 095653.

[3] **Novick, Paul A., et al.** (2013) *SWEETLEAD: an in silico database of approved drugs, regulated chemicals, and herbal isolates for computer-aided drug discovery.* PloS one 8.11 e79568.

[4] Aggregate Analysis of ClincalTrials.gov (AACT) Database.

[5] **Subramanian, Govindan, et al.** (2016) *Computational modeling of -secretase 1 (BACE-1) inhibitors using ligand based approaches.* Journal of chemical information and modeling 56.10 1936-1949.

[6] **Martins, Ines Filipa, et al.** (2014) *A Bayesian approach to in silico blood-brain barrier penetration modeling.* Journal of chemical information and modeling 52.6 (2012): 1686-1697.

[7] AIDS Antiviral Screen Data.

[8] **Mobley, David L., and J. Peter Guthrie.** *FreeSolv: a database of experimental and calculated hydration free energies, with input files.* Journal of computer-aided molecular design 28.7 711-720.

[9] **Hersey, A.** (2015) ChEMBL Deposited Data Set - AZ dataset

[10] **Huuskonen, J.** (2000) *Estimation of aqueous solubility for a diverse set of organic compounds based on molecular topology.* Journal of Chemical Information and Computer Sciences, 40(3), 773-777.

[11] **Sorkun, M. C., Khetan, A., & Er, S.** (2019) *AqSolDB, a curated reference set of aqueous solubility and 2D descriptors for a diverse set of compounds.* Scientific data, 6(1), 1-8.

- **type:** R->Numerical and C->Categorical
- **size:** Number of instances in the dataset
- **name:** Name of dataset
- **num_of_classes:** Number of classes (Categorical only)

You can retrieve the datasets by passing their ID to `load_data()`.

Note: The first 8 datasets in the table are edited versions of the MoleculeNet repository[12].

You can print the available sample datasets to console with ChemPlot using the following function:

```
from chemplot import info_data

df = info_data()
```

References:

[12] **Wu, Zhenqin, et al.** (2018) *MoleculeNet: a benchmark for molecular machine learning.* Chemical science 9.2 513-530.

CHAPTER

NINE

DEVELOPMENT ENVIRONMENT

The development environment is an installation of ChemPlot on your local computer which can be used for testing existing features or developing new ones in order to contribute to the library.

Start by making sure you have conda installed. This is needed since an important dependency of ChemPlot is RDKit, which can safely be installed only with conda.

Then clone your forked GitHub repository of ChemPlot on your local computer using either HTTPS:

```
~$ git clone https://github.com/<your-username>/ChemPlot.git
```

Or using SSH:

```
~$ git clone git@github.com:<your-username>/ChemPlot.git
```

Then from the terminal navigate to the ChemPlot repository you just created. From there create a new conda environment with all the dependencies needed to work with ChemPlot. Create the environment by running:

```
~/<PATH-TO-CLONE>/ChemPlot$ conda env create -f requirements_conda.yml
```

When conda finishes creating the environment, activate it by running:

```
~/<PATH-TO-CLONE>/ChemPlot$ conda activate chemplot_env
```

You can now install ChemPlot in editable mode. Editable mode will allow your code changes to be propagated through the library code without having to reinstall.

```
~/<PATH-TO-CLONE>/ChemPlot$ pip install -e .
```

You are now ready to develop ChemPlot!

9.1 Testing

To run the unit tests for ChemPlot use this command:

```
~$ python -m pytest --pyargs chemplot
```

On your cloned version of the ChemPlot repository you have two more tests, used to check performance of the library on your machine and to check the figures ChemPlot can generate. You can find these tests inside the performance_tests folder:

```
ChemPlot
├── ...
├── performance_tests/
│   ├── performanceTest.py
│   └── visualplotsTest.py
└── ...
```

You can run these tests by navigating to the performance_test library:

```
~/ChemPlot$ cd performance_tests
~/ChemPlot/performance_tests$ python performanceTest.py
~/ChemPlot/performance_tests$ python visualplotsTest.py
```

If it doesn't work you might have to change `python` with `python3` in the command. `performanceTest.py` will generate a `.csv` file containing all the times taken by ChemPlot to run all the dimensionality reduction methods on your machine. It will use the sample datasets provided with the library. `visualplotsTest.py` will instead create a multipage `.pdf` file containing different figures illustrating all plotting options for ChemPlot. These method as well will use the sample datasets included in the library.

CHAPTER

TEN

CITING CHEMPLOT

If you use ChemPlot for your scientific projects, we would appreciate if you would cite the paper Cihan Sorkun, M., Mullaj, D., Koelman, J., & Er, S. (2022). ChemPlot, a Python Library for Chemical Space Visualization. Chemistry–Methods, 2(7), e202200005].

```
@article{2022ChemPlot,
    author = {Cihan Sorkun, Murat and Mullaj, Dajt and Koelman, J. M. Vianney A. and Er,␣
↪Süleyman},
    title = {ChemPlot, a Python Library for Chemical Space Visualization},
    journal = {Chemistry-Methods},
    volume = {2},
    number = {7},
    pages = {e202200005},
    keywords = {chemical space visualization, cheminformatics, molecular similarity,␣
↪Python, tailored similarity},
    doi = {https://doi.org/10.1002/cmtd.202200005},
    url = {https://chemistry-europe.onlinelibrary.wiley.com/doi/abs/10.1002/cmtd.
↪202200005},
    eprint = {https://chemistry-europe.onlinelibrary.wiley.com/doi/pdf/10.1002/cmtd.
↪202200005},
    abstract = {Visualizing chemical spaces streamlines the analysis of molecular␣
↪datasets by reducing the information
    to human perception level, hence it forms an integral piece of molecular engineering,
↪ including chemical library design,
    high-throughput screening, diversity analysis, and outlier detection. We present␣
↪here ChemPlot, which enables users to
    visualize the chemical space of molecular datasets in both static and interactive␣
↪ways. ChemPlot features structural and
    tailored similarity methods, together with three different dimensionality reduction␣
↪methods: PCA, t-SNE, and UMAP.
    ChemPlot is the first visualization software that tackles the activity/property␣
↪cliff problem by incorporating tailored similarity.
    With tailored similarity, the chemical space is constructed in a supervised manner␣
↪considering target properties. Additionally,
    we propose a metric, the Distance Property Relationship score, to quantify the␣
↪property difference of similar (i.e. close)
    molecules in the visualized chemical space. ChemPlot can be installed via Conda or␣
↪PyPI (pip) and a web application is freely
    accessible at https://www.amdlab.nl/chemplot/.},
    year = {2022}
}
```

CHAPTER

ELEVEN

API DOCUMENTATION

ChemPlot principal class is `Plotter`. It receives a list of molecules as a parameter in order to then use different functions for plotting the data in two dimensions. All the main functions of ChemPlot are part of the `Plotter`. There are however two more functions outside of `Plotter`, which can be used to access the sample datasets.

11.1 chemplot.Plotter

class chemplot.**Plotter**(*encoding_list*, *target*, *target_type*, *sim_type*, *get_desc*, *get_fingerprints*)

A class used to plot the ECFP fingerprints of the molecules used to instantiate it.

Parameters

- **__sim_type** (*string*) – similarity type structural or tailored
- **__target_type** (*string*) – target type R (regression) or C (classificatino)
- **__target** (*list*) – list containing the target values. Is empty if a target does not exist
- **__mols** (*rdkit.Chem.rdchem.Mol*) – list of valid molecules that can be plotted
- **__df_descriptors** (*Dataframe*) – datatframe containing the descriptors representation of each molecule
- **__df_2_components** (*Dataframe*) – dataframe containing the two-dimenstional representation of each molecule
- **__plot_title** (*string*) – title of the plot reflecting the dimensionality reduction algorithm used
- **__data** (*list*) – list of the scaled descriptors to which the dimensionality reduction algorithm is applied
- **pca_fit** (*sklearn.decomposition.TSNE*) – PCA object created when the corresponding algorithm is applied to the data
- **tsne_fit** (*sklearn.manifold.TSNE*) – t-SNE object created when the corresponding algorithm is applied to the data
- **umap_fit** (**umap.umap_**.UMAP) – UMAP object created when the corresponding algorithm is applied to the data
- **df_plot_xy** (*Dataframe*) – dataframe containing the coordinates that have been plotted

classmethod from_smiles(*smiles_list*, *target=[]*, *target_type=None*, *sim_type=None*)

Class method to construct a Plotter object from a list of SMILES.

Parameters

- **smile_list** (*list*) – List of the SMILES representation of the molecules to plot.
- **target** (*list*) – target values
- **target_type** (*string*) – target type R (regression) or C (classificatino)
- **sim_type** (*string*) – similarity type structural or tailored

Returns
A Plotter object for the molecules given as input.

Return type
Plotter

classmethod from_inchi(*inchi_list*, *target=[]*, *target_type=None*, *sim_type=None*)

Class method to construct a Plotter object from a list of InChi.

Parameters

- **inchi_list** (*dict*) – List of the InChi representation of the molecules to plot.
- **target** (*dict*) – target values
- **target_type** (*string*) – target type R (regression) or C (classificatino)
- **sim_type** (*string*) – similarity type structural or tailored

Returns
A Plotter object for the molecules given as input.

Return type
Plotter

pca(***kwargs*)

Calculates the first 2 PCA components of the molecular descriptors.

Parameters
kwargs (*key, value mappings*) – Other keyword arguments are passed down to sklearn.decomposition.PCA

Returns
The dataframe containing the PCA components.

Return type
Dataframe

tsne(*perplexity=None*, *pca=False*, *random_state=None*, ***kwargs*)

Calculates the first 2 t-SNE components of the molecular descriptors.

Parameters

- **perplexity** (*int*) – perplexity value for the t-SNE model
- **pca** (*boolean*) – indicates if the features must be preprocessed by PCA
- **random_state** (*int*) – random seed that can be passed as a parameter for reproducing the same results
- **kwargs** (*key, value mappings*) – Other keyword arguments are passed down to sklearn.manifold.TSNE

Returns
The dataframe containing the t-SNE components.

Return type
Dataframe

umap(*n_neighbors=None*, *min_dist=None*, *pca=False*, *random_state=None*, ***kwargs*)

Calculates the first 2 UMAP components of the molecular descriptors.

Parameters

- **num_neighbors** (*int*) – Number of neighbours used in the UMAP madel.
- **min_dist** (*float*) – Value between 0.0 and 0.99, indicates how close to each other the points can be displayed.
- **random_state** (*int*) – random seed that can be passed as a parameter for reproducing the same results
- **kwargs** (*key, value mappings*) – Other keyword arguments are passed down to umap.UMAP

Returns
The dataframe containing the UMAP components.

Return type
Dataframe

cluster(*n_clusters=5*, ***kwargs*)

Computes the clusters presents in the embedded chemical space.

Parameters

- **n_clusters** (*int*) – Number of clusters that will be computed
- **kwargs** (*key, value mappings*) – Other keyword arguments are passed down to sklearn.cluster.KMeans

Returns
The dataframe containing the 2D embedding.

Return type
Dataframe

visualize_plot(*size=20*, *kind='scatter'*, *remove_outliers=False*, *is_colored=True*, *colorbar=False*, *clusters=False*, *filename=None*, *title=None*)

Generates a plot for the given molecules embedded in two dimensions.

Parameters

- **size** (*int*) – Size of the plot
- **kind** (*string*) – Type of plot
- **remove_outliers** (*boolean*) – Boolean value indicating if the outliers must be identified and removed
- **is_colored** (*boolean*) – Indicates if the points must be colored according to target
- **colorbar** (*boolean*) – Indicates if the plot legend must be represented as a colorbar. Only considered when the target_type is "R".
- **clusters** (*boolean or list or int*) – If True the clusters are shown instead of possible targets. Pass a list or a int to only show selected clusters (indexed by int).
- **filename** (*string*) – Indicates the file where to save the plot
- **title** (*string*) – Title of the plot.

Returns
The matplotlib axes containing the plot.

Return type
Axes

interactive_plot(*size=700*, *kind='scatter'*, *remove_outliers=False*, *is_colored=True*, *clusters=False*, *filename=None*, *show_plot=False*, *title=None*)

Generates an interactive Bokeh plot for the given molecules embedded in two dimensions.

Parameters

- **size** (*int*) – Size of the plot
- **kind** (*string*) – Type of plot
- **remove_outliers** (*boolean*) – Boolean value indicating if the outliers must be identified and removed
- **is_colored** (*boolean*) – Indicates if the points must be colored according to target
- **clusters** – Indicates if to add a tab with the clusters if these have been computed
- **filename** (*string*) – Indicates the file where to save the Bokeh plot
- **show_plot** (*boolean*) – Immediately display the current plot.
- **title** (*string*) – Title of the plot.

Returns
The bokeh figure containing the plot.

Return type
Figure

11.2 Utils

chemplot.**load_data**(*name*)

Returns one of the sample datasets.

Parameters
name (*string*) – Name of the sample dataset

Returns
The Dataframe of the sample dataset

Return type
Dataframe

chemplot.**info_data**()

Prints the metadata relative to the available sample datasets.

INDEX