

Clustering High Dimensional Data

Technical Review

Author: Laurent Querella, PhD, MSc

Senior Data Scientist

Version : draft version 0.1 – Oct 2022

Contents

1	Introduction to clustering	4
1.1	References	4
1.2	Clustering algorithms – key concepts	5
1.3	The curse of dimensionality	6
1.4	Point Assignment	7
1.4.1	K-means algorithms	7
1.4.2	The CURE algorithm	7
1.5	Hierarchical clustering	8
1.5.1	Euclidean space	8
1.5.2	Non-Euclidean spaces	8
1.6	Model-based clustering	8
1.7	Density-based clustering	8
1.7.1	DBSCAN	8
1.7.2	HDBSCAN	9
2	R Implementations	10
2.1	References	10
2.2	Clustering tendency	10
2.3	Mixed-type data	11
2.3.1	Gower’s distance	11
2.3.2	Finite-mixture models	11
2.3.3	Kamila package	11
2.4	Hierarchical clustering	12
2.4.1	Base R packages: stats and cluster	12
2.4.2	Other packages	12
2.5	Partitioning	14
2.5.1	K-means and alternatives	14
2.5.2	Application to PLIF	14
2.6	Model-based	15
2.6.1	ML estimation	15
2.6.2	Bayesian estimation	16
2.7	Density-based	16
2.7.1	DBSCAN algorithms	16
2.7.2	HDBSCAN	16
2.8	Dimension reduction and projections	16

2.8.1	References	16
2.8.2	LargeVis algorithm	16
2.8.3	UMAP algorithm.....	21
3	Python implementations	24
3.1	Density-based clustering.....	24
3.1.1	HDBSCAN.....	24
3.2	Dimension reduction.....	24
3.2.1	LargeVis algorithm	24
3.2.2	Triplet constraints	24
3.2.3	UMAP	24

1 Introduction to clustering

1.1 References

<http://infolab.stanford.edu/~ullman/mmds/ch7.pdf>

Chapter 7

Clustering

Clustering is the process of examining a collection of “points,” and grouping the points into “clusters” according to some distance measure. The goal is that points in the same cluster have a small distance from one another, while points in different clusters are at a large distance from one another. A suggestion of what clusters might look like was seen in Fig. 1.1. However, the intent was that there were three clusters around three different road intersections, but two of the clusters blended into one another because they were not sufficiently separated.

Our goal in this chapter is to offer methods for discovering clusters in data. We are particularly interested in situations where the data is very large, and/or where the space either is high-dimensional, or the space is not Euclidean at all. We shall therefore discuss several algorithms that assume the data does not fit in main memory. However, we begin with the basics: the two general approaches to clustering and the methods for dealing with clusters in a non-Euclidean space.

https://www-users.cs.umn.edu/~kumar001/papers/high_dim_clustering_19.pdf


The Challenges of Clustering High Dimensional Data*

Michael Steinbach, Levent Ertöz, and Vipin Kumar

Abstract

Cluster analysis divides data into groups (clusters) for the purposes of summarization or improved understanding. For example, cluster analysis has been used to group related documents for browsing, to find genes and proteins that have similar functionality, or as a means of data compression. While clustering has a long history and a large number of clustering techniques have been developed in statistics, pattern recognition, data mining, and other fields, significant challenges still remain. In this chapter we provide a short introduction to cluster analysis, and then focus on the challenge of clustering high dimensional data. We present a brief overview of several recent techniques, including a more detailed description of recent work of our own which uses a concept-based clustering approach.

<https://samm.univ-paris1.fr/IMG/pdf/bouveyron.pdf>



Université Paris 1
Panthéon - Sorbonne

HDclassif : an R Package for Model-Based Classification of High-Dimensional Data

Charles BOUYEYRON

Laboratoire SAMM, EA 4543
Université Paris 1 Panthéon-Sorbonne

This joint work with L. Bergé & S. Girard

Charles BOUYEYRON | HDclassif : an R Package for Model-Based Classification of High-Dimensional Data 1/38

1.2 Clustering algorithms – key concepts

Clustering strategies:

- **Hierarchical**: produces a nested sequence of partitions, with a single, all-inclusive cluster at the top and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining (splitting) two clusters from the next lower (next higher) level.
 - Agglomerative: starting with clusters containing a single point, and then merging them
 - Divisive: starting with one large cluster and splitting it
- **Partition** (point assignment): creates a one-level (unnested) partitioning of the data points. If K is the desired number of clusters, then partitional approaches typically find all K clusters at once.
 - This process is normally preceded by a short phase in which initial clusters are estimated
- **Model-based**: assumes that the data were generated by a model and tries to recover the original model from the data. The model that we recover from the data then defines clusters and an assignment of points to clusters. A commonly used criterion for estimating the model parameters is maximum likelihood
- **Other**: fuzzy clustering, hybrid methods, graph and network, etc.
 - Most interesting is density-based clustering (e.g. DBSCAN algorithm)

Algorithms for clustering can also be distinguished by:

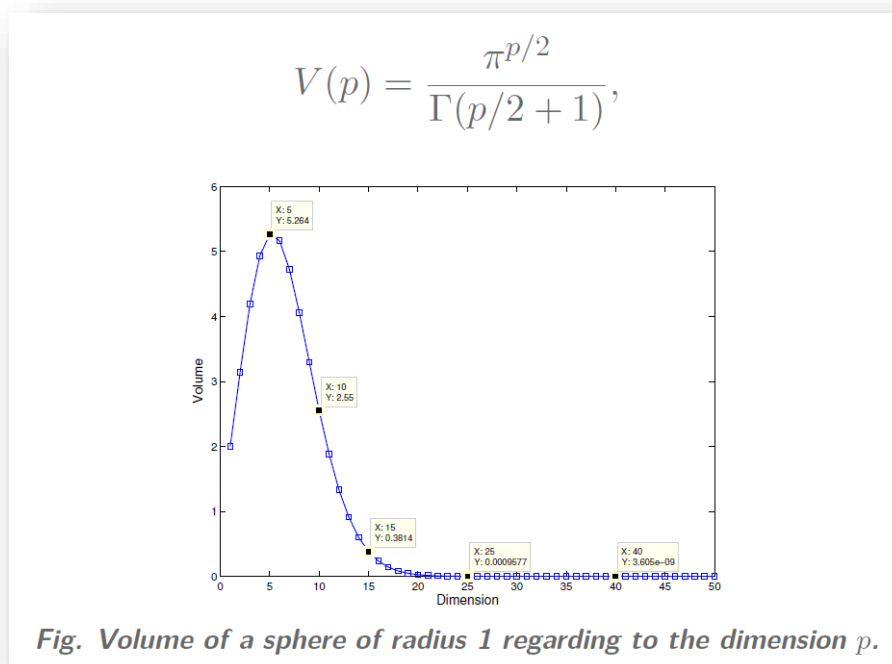
- Whether the algorithm assumes a *Euclidean space*, or whether the algorithm works for an arbitrary distance measure. We shall see that a key distinction is that in a Euclidean space it

is possible to summarize a collection of points by their *centroid* – the average of the points. In a non-Euclidean space, there is no notion of a centroid, and we are forced to develop another way to summarize clusters

- Whether the algorithm assumes that the data is small enough to fit in main memory, or whether data must reside in secondary memory, primarily. Algorithms for large amounts of data often must take shortcuts, since it is infeasible to look at all pairs of points, for example. It is also necessary to summarize clusters in main memory, since we cannot hold all the points of all the clusters in main memory at the same time

1.3 The curse of dimensionality

High-dimensional Euclidean spaces have a number of unintuitive properties that are sometimes referred to as the “curse of dimensionality.” Non-Euclidean spaces usually share these anomalies as well. One manifestation of the “curse” is that in high dimensions, almost all pairs of points are equally far away from one another. Another manifestation is that almost any two vectors are almost orthogonal. In general terms, problems with high dimensionality result from the fact that a fixed number of data points become increasingly “sparse” as the dimensionality increase: *high-dimensional spaces are almost empty!*



How to avoid this curse?

Dimension reduction:

- the problem comes from that p is too large,
- therefore, reduce the data dimension to $d \ll p$,
- such that the curse of dimensionality vanishes!

Parsimonious models:

- the problem comes from that the number of parameters to estimate is too large,
- therefore, make additional assumptions to the model,
- such that the number of parameters to estimate becomes more “decent”!

Regularization:

- the problem comes from that parameter estimates are instable,
- therefore, regularize these estimates,
- such that the parameter are correctly estimated!

1.4 Point Assignment

1.4.1 K-means algorithms

They assume a Euclidean space, and they also assume the number of clusters, k , is known in advance. It is, however, possible to deduce k by trial and error. K-means poorly performs for bigger data sets. The BFR algorithm enables to execute k-means on data that is too large to fit in main memory.

```
Initially choose  $k$  points that are likely to be in
different clusters;
Make these points the centroids of their clusters;
FOR each remaining point  $p$  DO
    find the centroid to which  $p$  is closest;
    Add  $p$  to the cluster of that centroid;
    Adjust the centroid of that cluster to account for  $p$ ;
END;
```

Figure 7.7: Outline of k -means algorithms

1.4.2 The CURE algorithm

This algorithm, called CURE (Clustering Using REpresentatives), assumes a Euclidean space. However, it does not assume anything about the shape of clusters; they need not be normally distributed, and can even have strange bends, S-shapes, or even rings. Instead of representing clusters by their centroid, it uses a collection of representative points, as the name implies.

1.5 Hierarchical clustering

1.5.1 Euclidean space

We assume the space is Euclidean. That allows us to represent a cluster by its **centroid** or average of the points in the cluster. Note that in a cluster of one point, that point is the centroid, so we can initialize the clusters straightforwardly. We can then use the merging rule that the distance between any two clusters is the Euclidean distance between their centroids, and we should pick the two clusters at the shortest distance. Other ways to define intercluster distance are possible, and we can also pick the best pair of clusters on a basis other than their distance.

1.5.2 Non-Euclidean spaces

When the space is non-Euclidean, we need to use some distance measure that is computed from points, such as Jaccard, cosine, or edit distance. That is, we cannot base distances on “location” of points. A problem arises when we need to represent a cluster, because we cannot replace a collection of points by their centroid: our only choice is to pick one of the points of the cluster itself to represent the cluster. Ideally, this point is close to all the points of the cluster, so it in some sense lies in the “center.” We call the representative point the **clustroid**. We can select the clustroid in various ways, each designed to, in some sense, minimize the distances between the clustroid and the other points in the cluster.

1.6 Model-based clustering

<http://www.sthda.com/english/articles/30-advanced-clustering/104-model-based-clustering-essentials/>

Basic idea behind Model-based Clustering

- Sample observations arise from a distribution that is a mixture of two or more components.
- Each component is described by a density function and has an associated probability or “weight” in the mixture.
- In principle, we can adopt any probability model for the components, but typically we will assume that components are p -variate normal distributions. (This does not necessarily mean things are easy: inference is tractable, however.)
- Thus, the probability model for clustering will often be a mixture of multivariate normal distributions.
- Each component in the mixture is what we call a cluster.

1.7 Density-based clustering

1.7.1 DBSCAN

Density-based spatial clustering of applications with noise – **DBSCAN** – is one of the most common clustering algorithms and also most cited in scientific literature. DBSCAN is a density based algorithm; it assumes clusters for dense regions: given a set of points in some space, it groups together points

that are closely packed together (points with many nearby neighbours), marking as outliers points that lie alone in low-density regions (whose nearest neighbours are too far away). It doesn't require that every point be assigned to a cluster and hence doesn't partition the data, but instead extracts the 'dense' clusters and leaves sparse background classified as 'noise'.

In practice DBSCAN is related to agglomerative clustering. As a first step DBSCAN transforms the space according to the density of the data: points in dense regions are left alone, while points in sparse regions are moved further away. Applying single linkage clustering to the transformed space results in a dendrogram, which we cut according to a distance parameter (called epsilon or `eps` in many implementations) to get clusters. Importantly any singleton clusters at that cut level are deemed to be 'noise' and left unclustered. This provides several advantages: we get the manifold following behaviour of agglomerative clustering, and we get actual clustering as opposed to partitioning. Better yet, since we can frame the algorithm in terms of local region queries we can use various tricks such as `kdtrees` to get exceptionally good performance and scale to dataset sizes that are otherwise unapproachable with algorithms other than K-Means. There are some catches however. Obviously epsilon can be hard to pick; you can do some data analysis and get a good guess, but the algorithm can be quite sensitive to the choice of the parameter. The density based transformation depends on another parameter (`min_samples` in `sklearn`). Finally the combination of `min_samples` and `eps` amounts to a choice of density and the clustering only finds clusters at or above that density; if your data has variable density clusters then DBSCAN is either going to miss them, split them up, or lump some of them together depending on your parameter choices.

1.7.2 HDBSCAN

HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise. Performs DBSCAN over varying epsilon values and integrates the result to find a clustering that gives the best stability over epsilon. This allows HDBSCAN to find clusters of varying densities (unlike DBSCAN), and be more robust to parameter selection.

In practice this means that HDBSCAN returns a good clustering straight away with little or no parameter tuning -- and the primary parameter, minimum cluster size, is intuitive and easy to select.

HDBSCAN is ideal for exploratory data analysis; it's a fast and robust algorithm that you can trust to return meaningful clusters (if there are any).

2 R Implementations

2.1 References

<http://cran.cnr.berkeley.edu/web/views/Cluster.html>

CRAN Task View: Cluster Analysis & Finite Mixture Models

Maintainer: Friedrich Leisch and Bettina Gruen

Contact: Bettina.Gruen at jku.at

Version: 2018-05-04

URL: <https://CRAN.R-project.org/view=Cluster>

This CRAN Task View contains a list of packages that can be used for finding groups in data and modeling unobserved cross-sectional heterogeneity, rather than an ultimate categorization. Except for packages stats and cluster (which ship with base R and hence are part of every R installation), each p

<https://www.r-bloggers.com/clustering-mixed-data-types-in-r/>

Clustering Mixed Data Types in R

June 21, 2016

By Wicked Good Data - r

(This article was first published on [Wicked Good Data - r](#), and kindly contributed to [R-bloggers](#))

14
SHARES

f Share

🐦 Tweet

Clustering allows us to better understand how a sample might be comprised of distinct subgroups given a set of variables. While many introductions to cluster analysis typically review a simple application using continuous variables, clustering data of mixed types (e.g., continuous, ordinal, and nominal) is often of interest. The following is an overview of one approach to clustering data of mixed types using Gower distance, partitioning around medoids, and silhouette width.

2.2 Clustering tendency

<http://www.sthda.com/english/articles/29-cluster-validation-essentials/95-assessing-clustering-tendency-essentials/>

Before applying any clustering method on your data, it's important to evaluate whether the data sets contains meaningful clusters (i.e.: non-random structures) or not. If yes, then how many clusters are there. This process is defined as the assessing of clustering tendency or the feasibility of the clustering analysis.

A big issue, in cluster analysis, is that clustering methods will return clusters even if the data does not contain any clusters. In other words, if you blindly apply a clustering method on a data set, it will divide the data into clusters because that is what it supposed to do.

One method consists in checking the Hopkins statistic:

The Hopkins statistic (Lawson and Jurs 1990) is used to assess the clustering tendency of a data set by measuring the probability that a given data set is generated by a uniform data distribution. In other words, it tests the spatial randomness of the data.

2.3 Mixed-type data

Most of the clustering functions assume a numeric data frame or matrix to compute pairwise distances. In some cases one may provide a dissimilarity matrix, thus skipping the data.

When there are categorical variables in the data frame, one may perform *one-hot encoding* (with e.g. {caret}) and use “classical” distances, but this results in much more dimensions to handle and some issues related to the presence or absence thereof of one level in some space region.

Numerous other existing strategies for clustering mixed-type data involve this same problem of requiring a user-specified weight determining the relative contribution of continuous versus categorical variables (next Gower’s distance also exhibits this problem – cf. Kamila package for an extensive review).

2.3.1 Gower’s distance

Using a distance metric compatible with mixed data, such as Gower’s distance and then use a clustering method that depends only on the distances between the data points. However, each variable in Gower’s distance must be assigned a user-specified weight determining its relative contribution to the distance, which presents essentially the same dilemma as the choice of c (“level is present == c ”) in the dummy coding approach. A poor choice of weights will result in certain variables being over- or under-emphasized, and in particular, it is unclear how to properly weigh the continuous variables relative to the categorical variables for a given data set.

- `cluster::daisy()`
 - `diss_matrix <- daisy(x, metric = "gower")`
- Use any algorithm that can take as input a dissimilarity matrix instead of the raw data

2.3.2 Finite-mixture models

Finite mixture modelling is another technique that does not require user-specified weights for the continuous versus categorical variable contribution.

For mixed-type data, a popular model is the joint normal-multinomial mixture model. Finite mixture models are often able to achieve a favourable balance between continuous and categorical variables when their parametric assumptions are reasonably accurate; however, they often perform poorly when their parametric assumptions are strongly violated.

2.3.3 Kamila package

<https://CRAN.R-project.org/package=kamila>

In a manner similar to finite mixture models, the KAMILA method is able to achieve a favourable balance between continuous and categorical variables without requiring user-specified weights. In order to decrease the susceptibility to violations of parametric assumptions, the continuous components are modelled using a general class of elliptical distributions. Categorical variables are modelled as mixtures of multinomial random variables, thus not requiring dummy coding.

Existing clustering software for very large data sets relies heavily on methods designed for continuous data only, and on k-means clustering in particular; they are thus vulnerable to the drawbacks

enumerated above when used with mixed-type data. For large mixed-type data sets the KAMILA clustering algorithm is implemented in Hadoop streaming mode.

NB: testing Kamila package on PLIF data yields a fatal error.

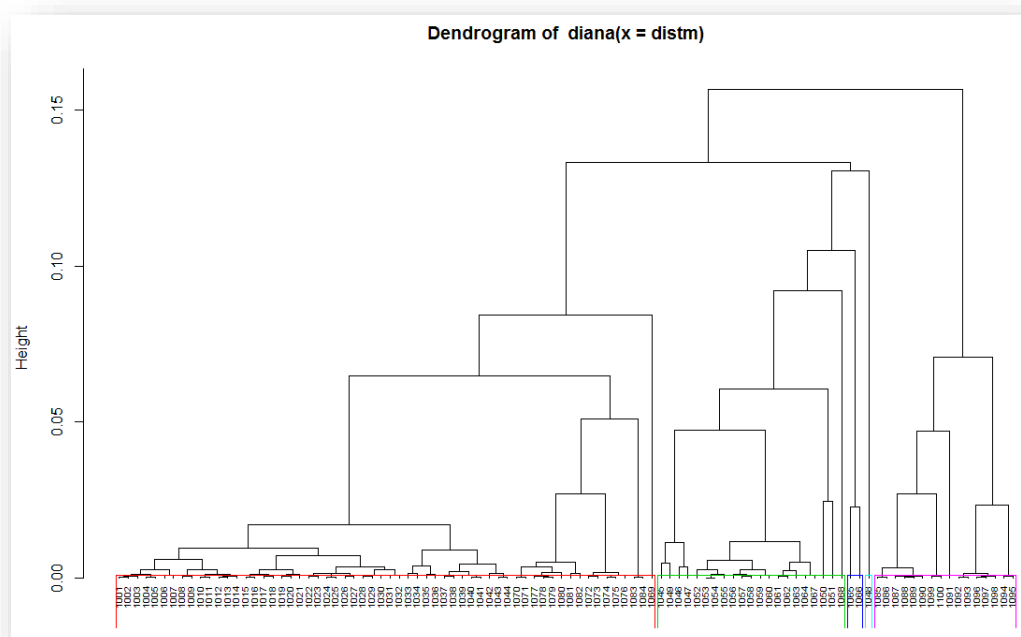
2.4 Hierarchical clustering

2.4.1 Base R packages: `stats` and `cluster`

- *agglomerative*
 - `stats::hclust()`
 - faster alternative: `{fastcluster}` and `{flashClust}`
 - `cluster::agnes()`
 - Ward method seems to perform better
- *divisive*
 - `cluster::diana()`

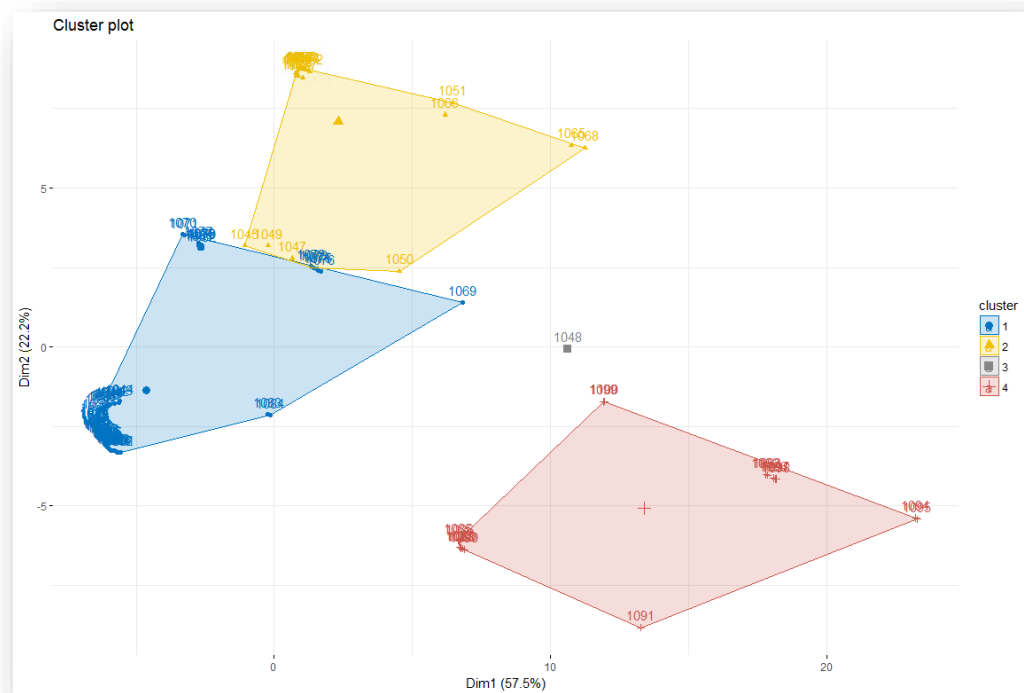
2.4.2 Other packages

- Cutting a tree
 - `stats::cutree()`
 - `stats::rect.hclust()`

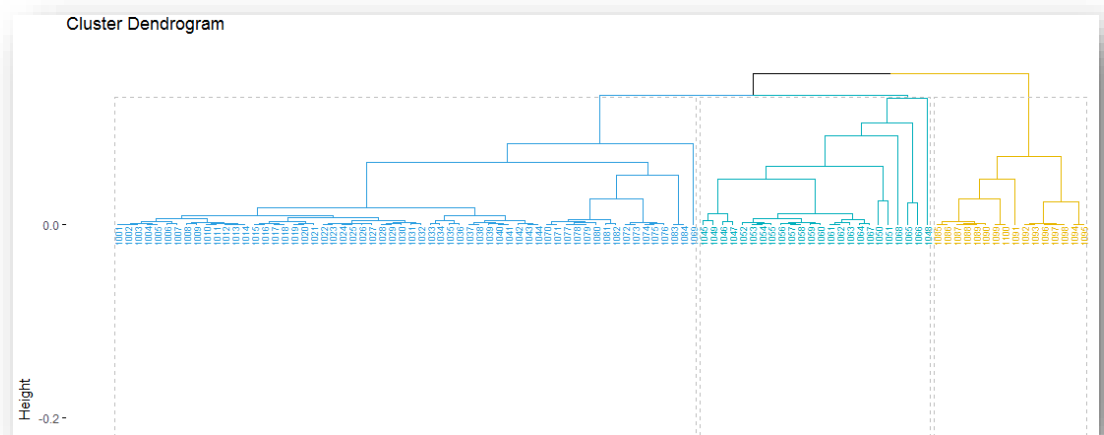


- Scatterplot of clusters in 2D (after projection on principal components axes)

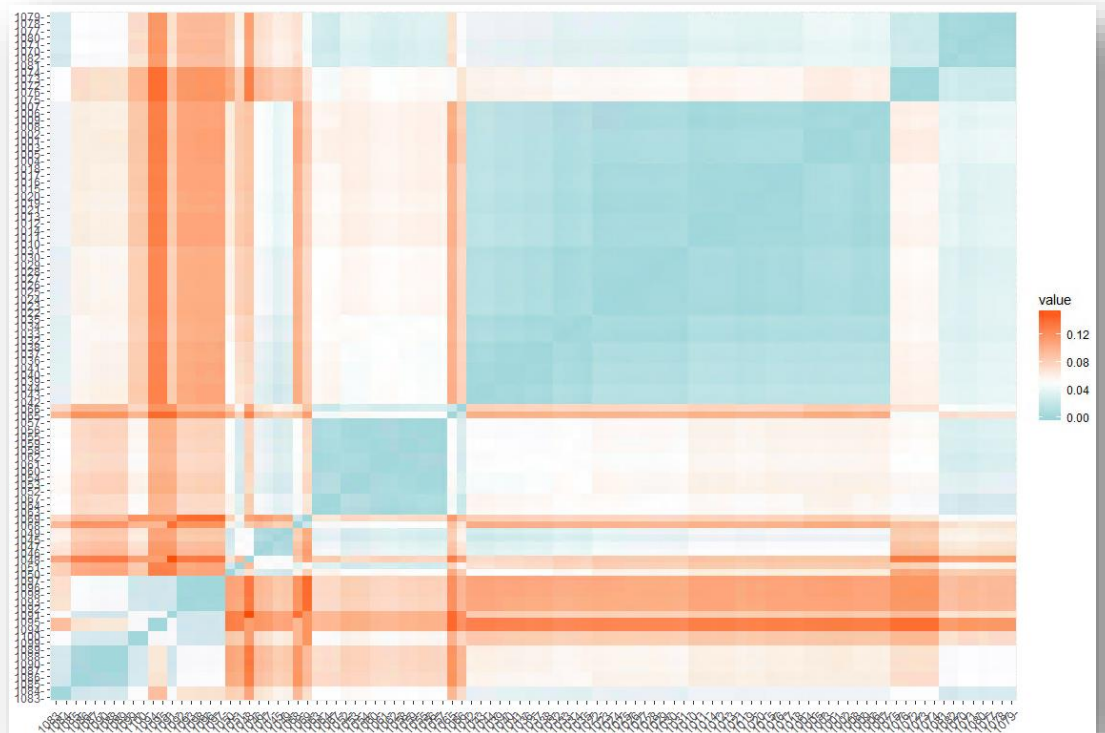
- `factoextra::fviz_cluster()`



- `factoextra::fviz_dend()`



- `factoextra::fviz_dist()`



- Determining the optimal number of clusters
 - `NbClust::NbClust()`
- Visualisation of dendrograms
 - `dendextend`
- (...)

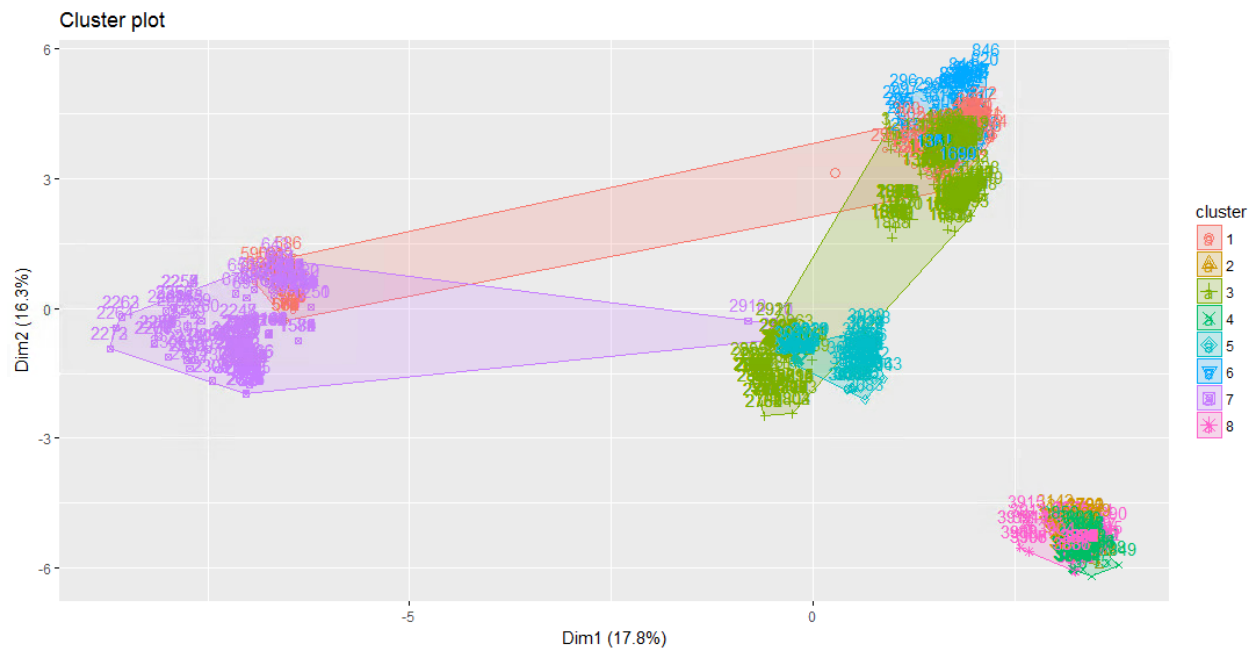
2.5 Partitioning

2.5.1 K-means and alternatives

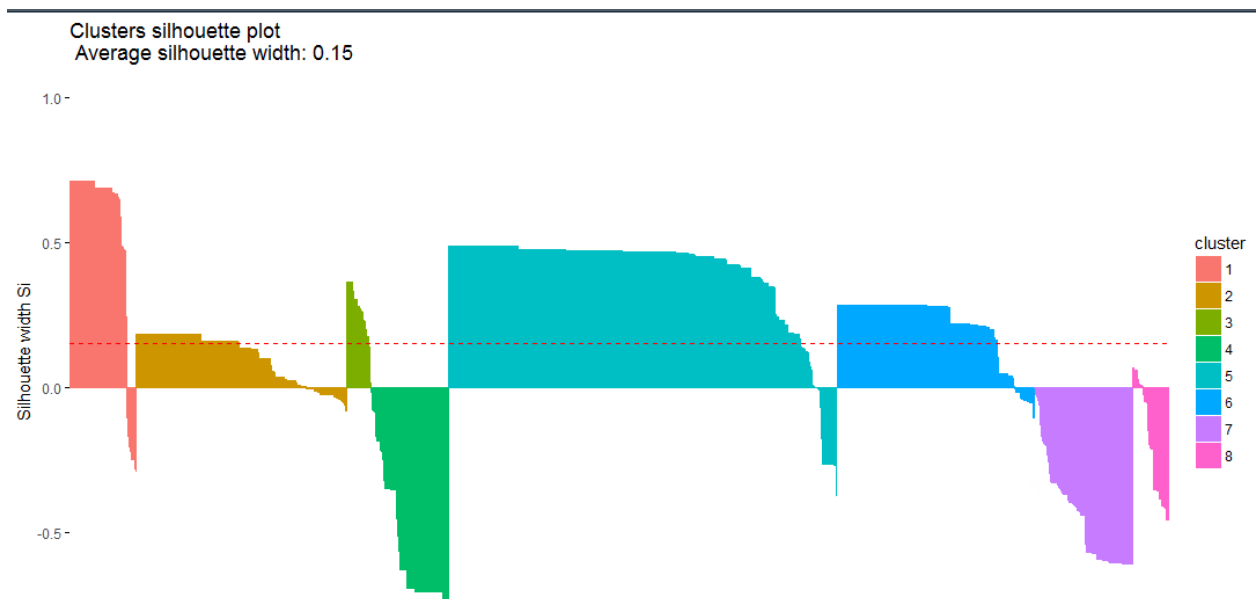
- k-means
 - `stats::kmeans()`
- k-medoids PAM (partitioning around medoids) algorithm
 - `cluster::pam()`
 - `cluster::clara()` the partitioning steps are executed on sub-sampled data and then applied to the entire data set. This efficient algorithmic structure makes clara well suited for large data sets that can be stored in RAM
- Kernel k-means uses kernels to project the data into a non-linear feature space before applying k-means
 - `kernlab::kkmeans()` and `kernlab::specc()` for spectral clustering
- Generalizations of k-means using arbitrary centroid statistics and distance measures
 - `flexclust::kcca()`

2.5.2 Application to PLIF

Df1 zone 1:



Silhouette plot



2.6 Model-based

2.6.1 ML estimation

- **mclust** fits mixtures of Gaussians using the EM algorithm. It allows fine control of volume and shape of covariance matrices and agglomerative hierarchical clustering based on maximum likelihood. It provides comprehensive strategies using hierarchical clustering, EM and the Bayesian Information Criterion (BIC) for clustering, density estimation, and discriminant analysis. It provides all 14 possible variance-covariance structures based on the eigenvalue decomposition
- **HDclassif** provides function **hddc** to fit Gaussian mixture model to high-dimensional data where it is assumed that the data lives in a lower dimension than the original space.
- (...)

2.6.2 Bayesian estimation

- Bayesian estimation of finite mixtures of multivariate Gaussians is possible using package [bayesm](#). The package provides functionality for sampling from such a mixture as well as estimating the model using Gibbs sampling. Additional functionality for analyzing the MCMC chains is available for averaging the moments over MCMC draws, for determining the marginal densities, for clustering observations and for plotting the uni- and bivariate marginal densities
- [bayesmix](#) provides Bayesian estimation using JAGS
- (...)

2.7 Density-based

2.7.1 DBSCAN algorithms

<http://www.sthda.com/english/wiki/print.php?id=246>

- [dbscan](#) provides a fast reimplementation of the DBSCAN (density-based spatial clustering of applications with noise) algorithm using a kd-tree
- [largeVis](#) implements the algorithm of the same name for visualizing very large high-dimensional datasets. Regarding clustering optimized implementations of the HDBSCAN*, DBSCAN and OPTICS algorithms are provided in combination with a very fast search for approximate nearest neighbors and outlier detection. Use of Spotify-annoy package: <https://github.com/spotify/annoy>

2.7.2 HDBSCAN

Accelerated HDBSCAN* algorithm (May 2017): implemented in Python/scikit-learn:

<https://github.com/scikit-learn-contrib/hdbscan>

<http://hdbscan.readthedocs.io/en/latest/>

R version in largeVis: removed from CRAN

<https://cran.r-project.org/web/packages/largeVis/index.html>

2.8 Dimension reduction and projections

2.8.1 References

<https://jlmelville.github.io/uwot/umap-for-tsne.html>

2.8.2 LargeVis algorithm

<https://cran.r-project.org/web/packages/largeVis/vignettes/largeVis.html>

<https://github.com/elbamos/largeVis>

This package provides LargeVis visualizations and fast nearest-neighbour search. The LargeVis algorithm, presented in Tang et al. (2016), creates high-quality low-dimensional representations of large, high-dimensional datasets, similar to t-SNE.

These visualizations are useful for data exploration, for visualizing complex non-linear functions, and especially for visualizing embeddings such as learned vectors for images.

A limitation of t-SNE is that because the algorithm has complexity order $O(n^2)$, it is not feasible for use on even moderately sized datasets. Barnes-Hut, an approximation of t-SNE, has complexity $O(n \log n)$ but also quickly becomes infeasible as the size of data grows. LargeVis is intended to address the issue

by operating in linear $O(n)$ time. It has been benchmarked at more than 30x faster than Barnes-Hut on datasets of approximately 1-million rows, and scaled linearly as long as there is sufficient RAM.

In addition, LargeVis includes an algorithm for finding approximate k-Nearest Neighbours in $O(n)$ time. This algorithm turns out to be faster at finding accurate a-NNs than any other method I was able to test.

The package also includes a function for visualizing image embeddings by plotting images at the locations given by the LargeVis algorithm.

For a detailed description of the algorithm, please see the original paper, Tang et al. (2016).

2.8.2.1 *randomProjectionTreeSearch {largeVis}*

A fast and accurate algorithm for finding approximate k-nearest neighbors.

CAUTION: input data should be in the form of a (potentially sparse) matrix, where examples are columns and features are rows.

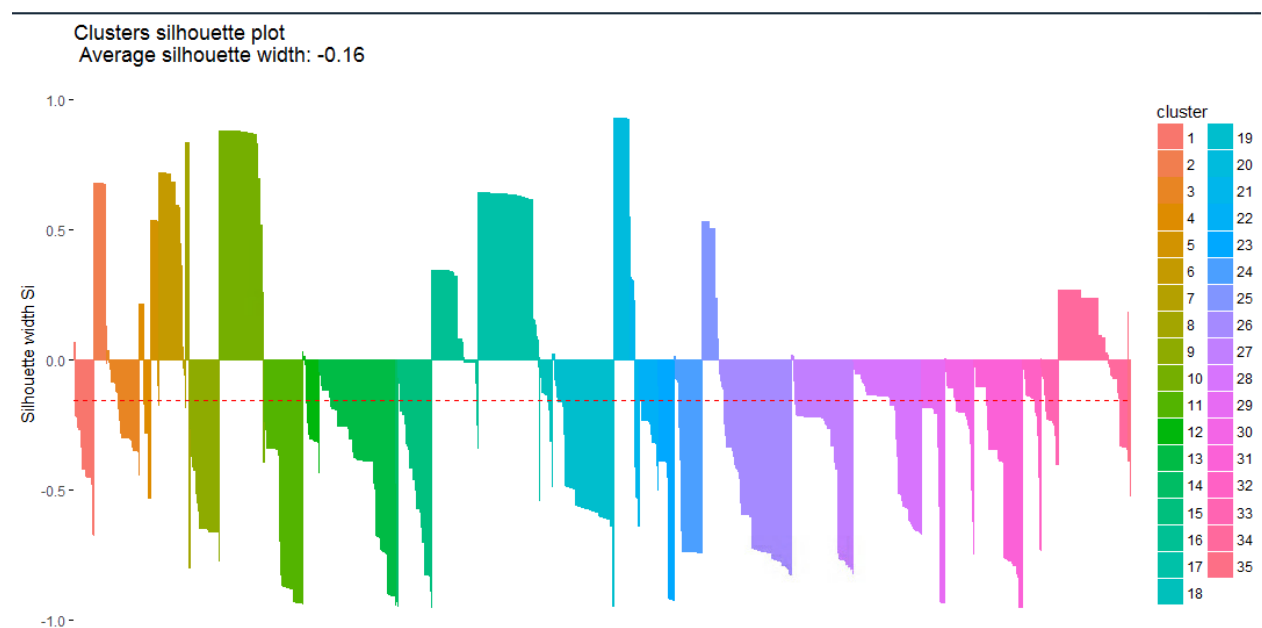
Implementation of HDBSCAN*:

<https://rdr.io/cran/largeVis/man/hdbscan.html>

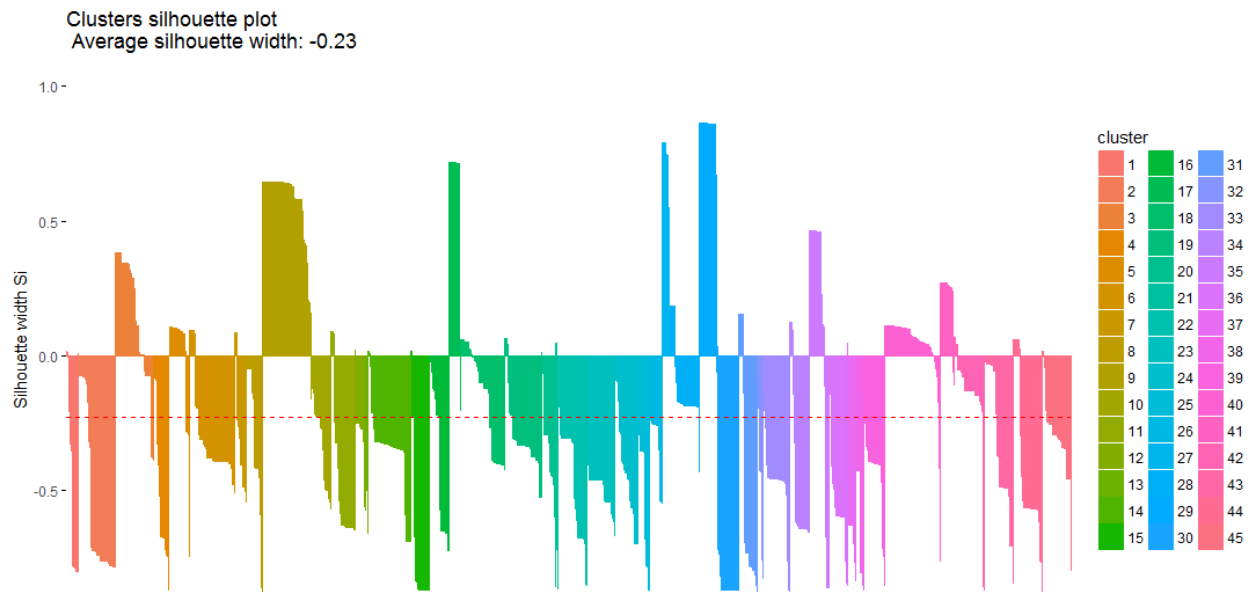
2.8.2.2 *Application to PLIF data*

Zone 1:

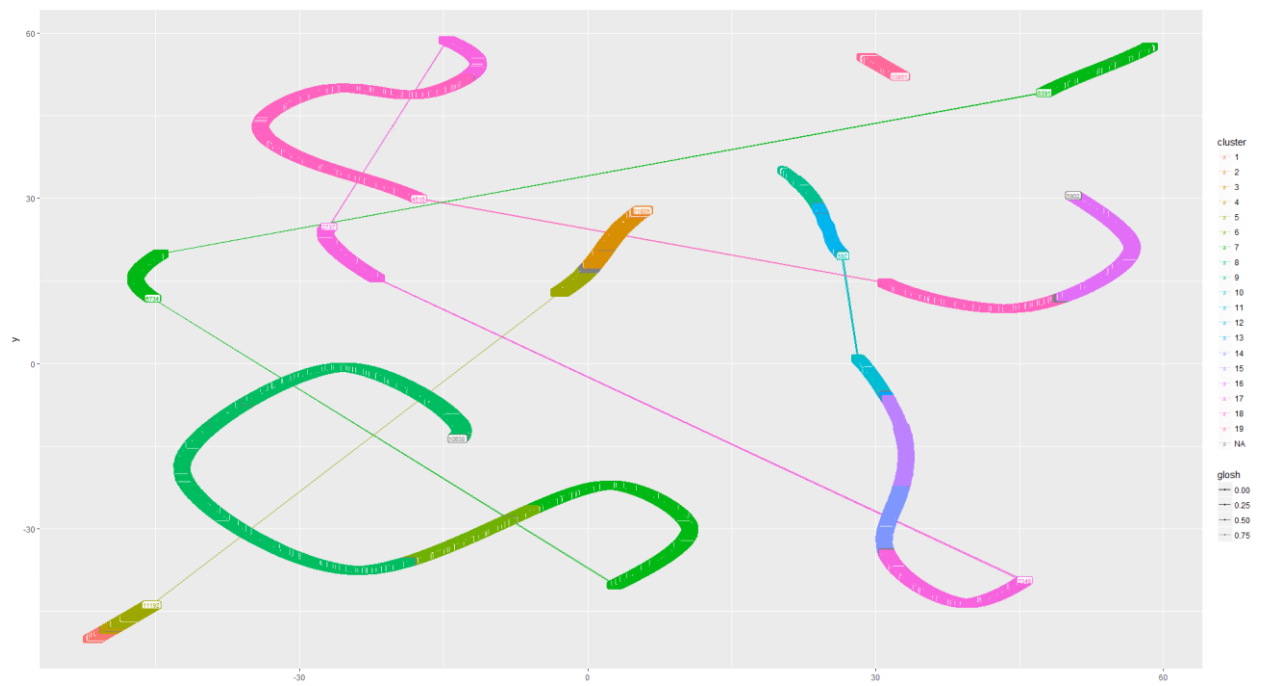
Kmeans – silhouette plot (best)



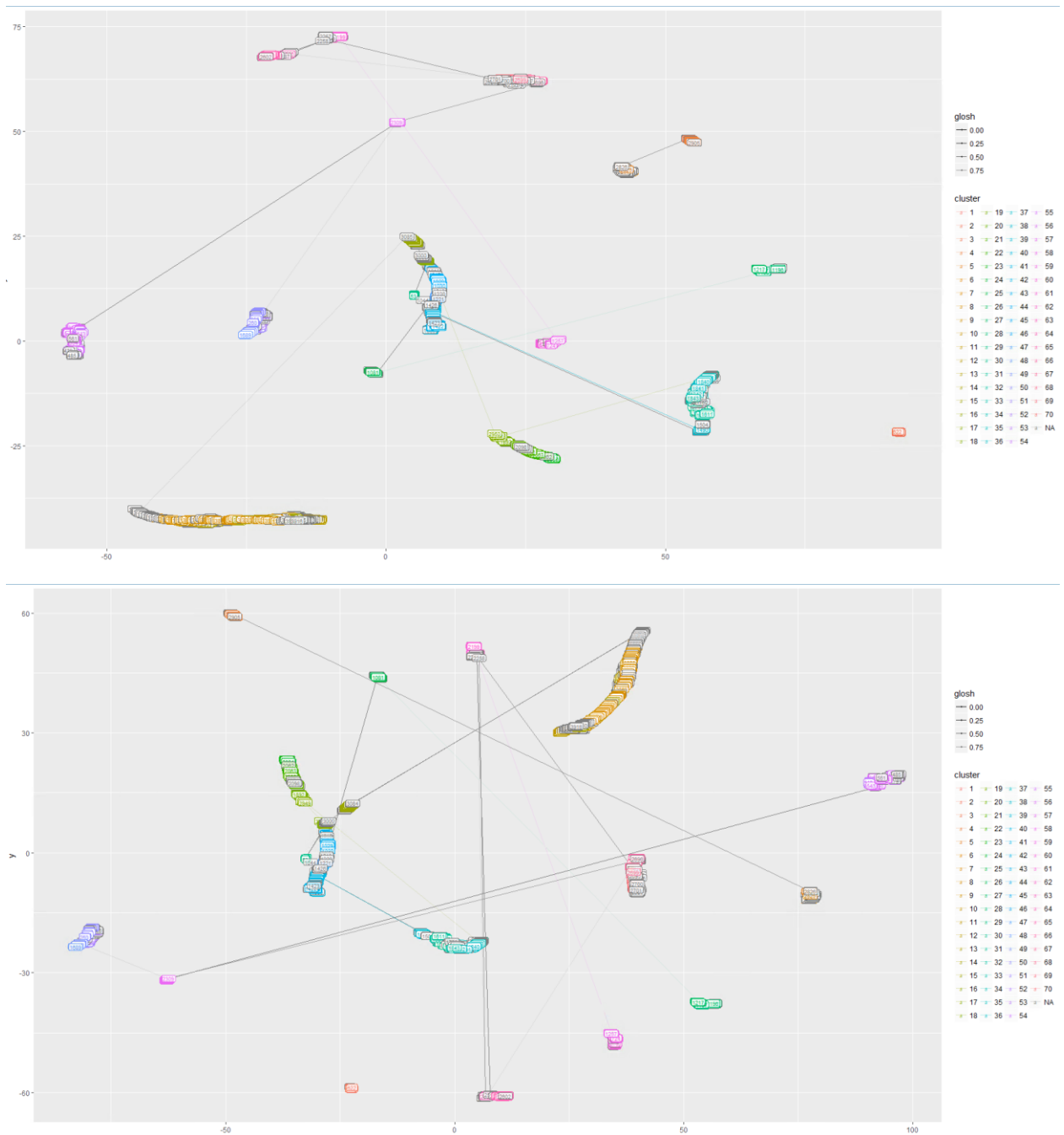
Zone 2



Zone 1 LargeVis

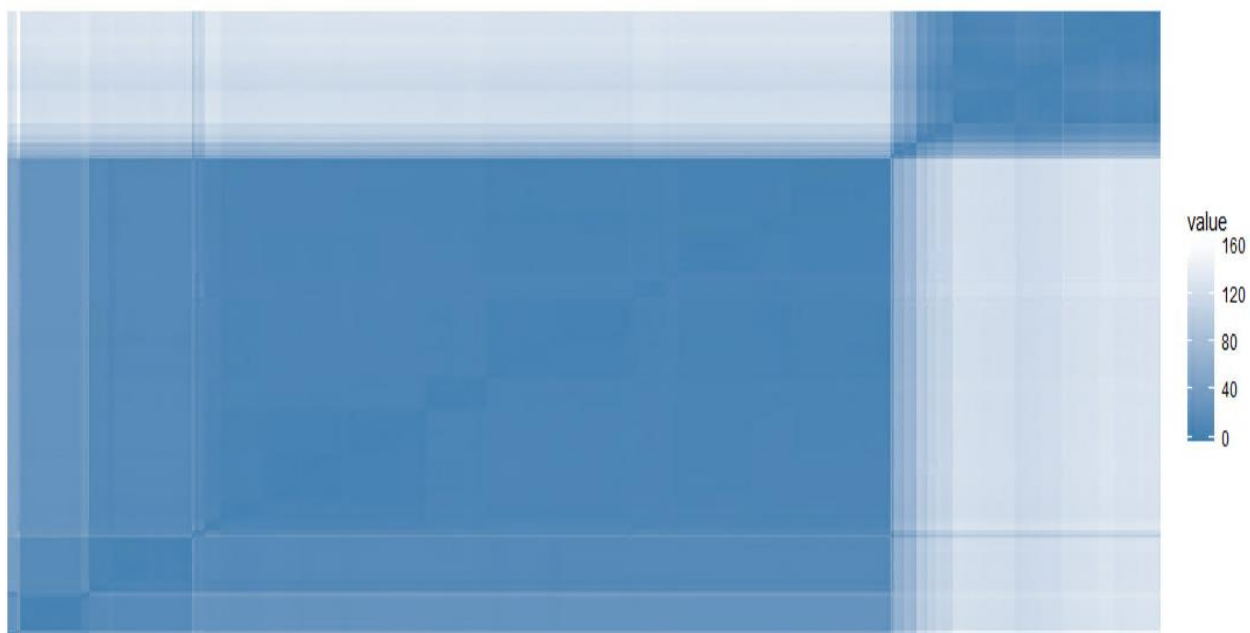


SP1

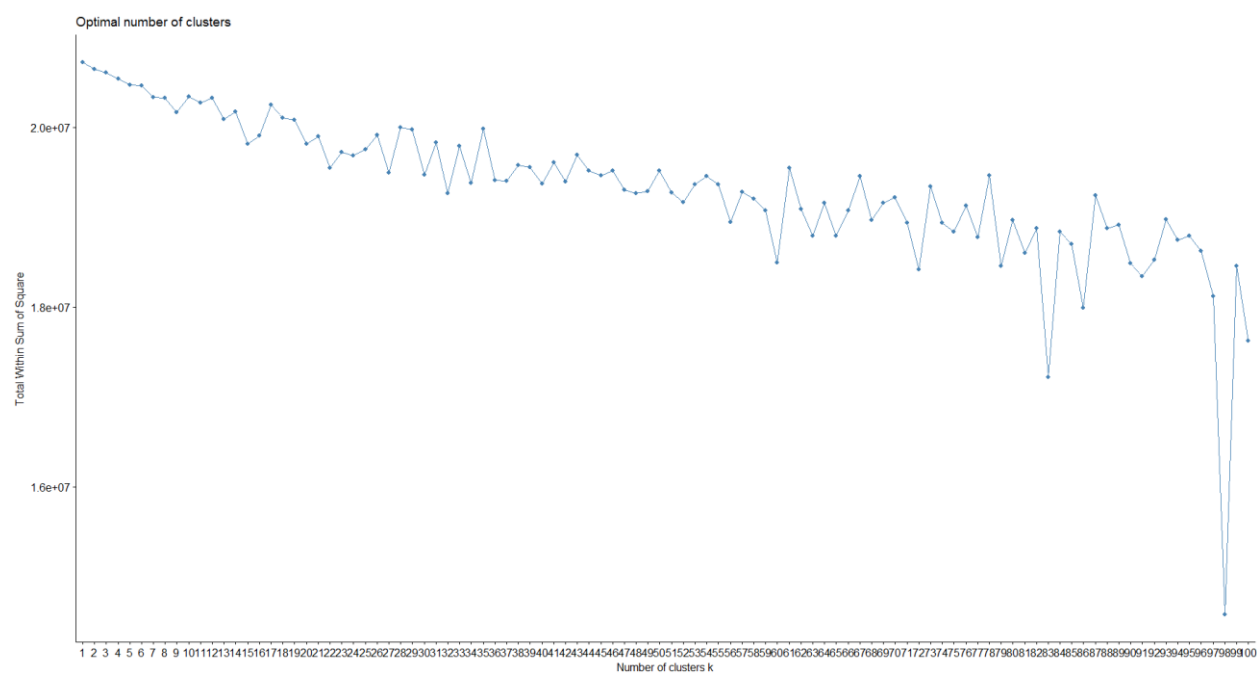


Hopkins statistic: If the value of Hopkins statistic is close to zero (far below 0.5), then we can conclude that the dataset is significantly clusterable: for SP1 it is 0.009420729 on a 500 sample

https://en.wikipedia.org/wiki/Hopkins_statistic

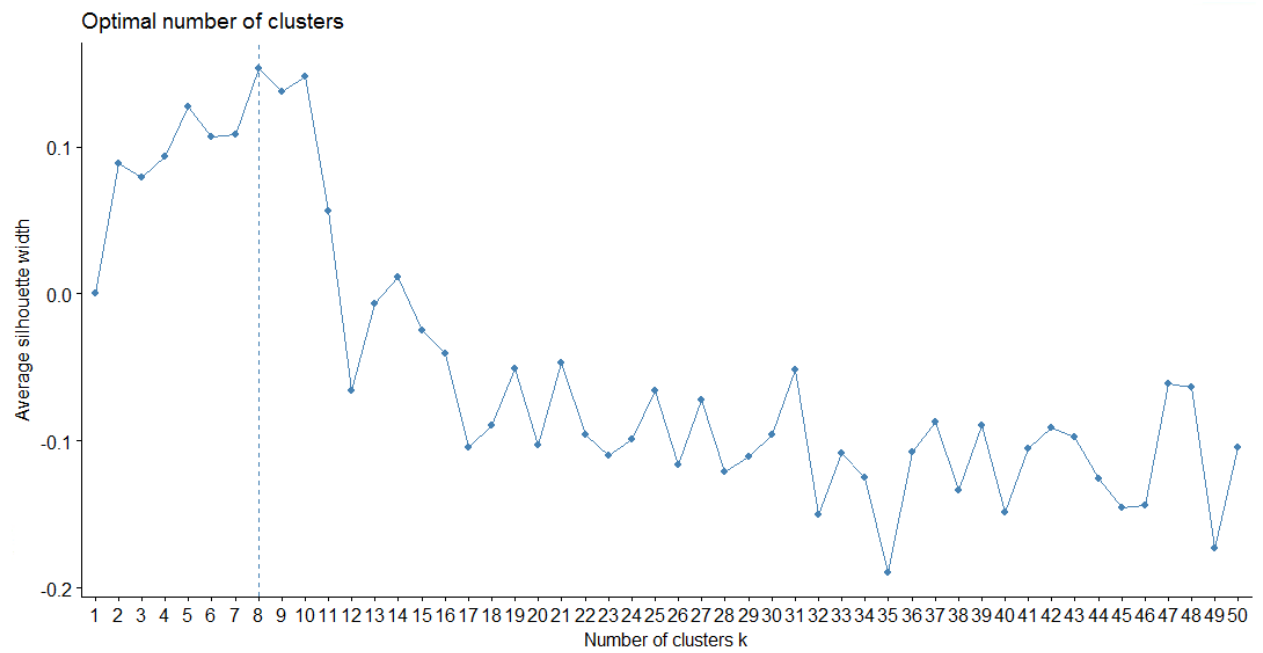


Optimal number of clusters (factoextra-kmeans)



Df1: 98

Silhouette: 8

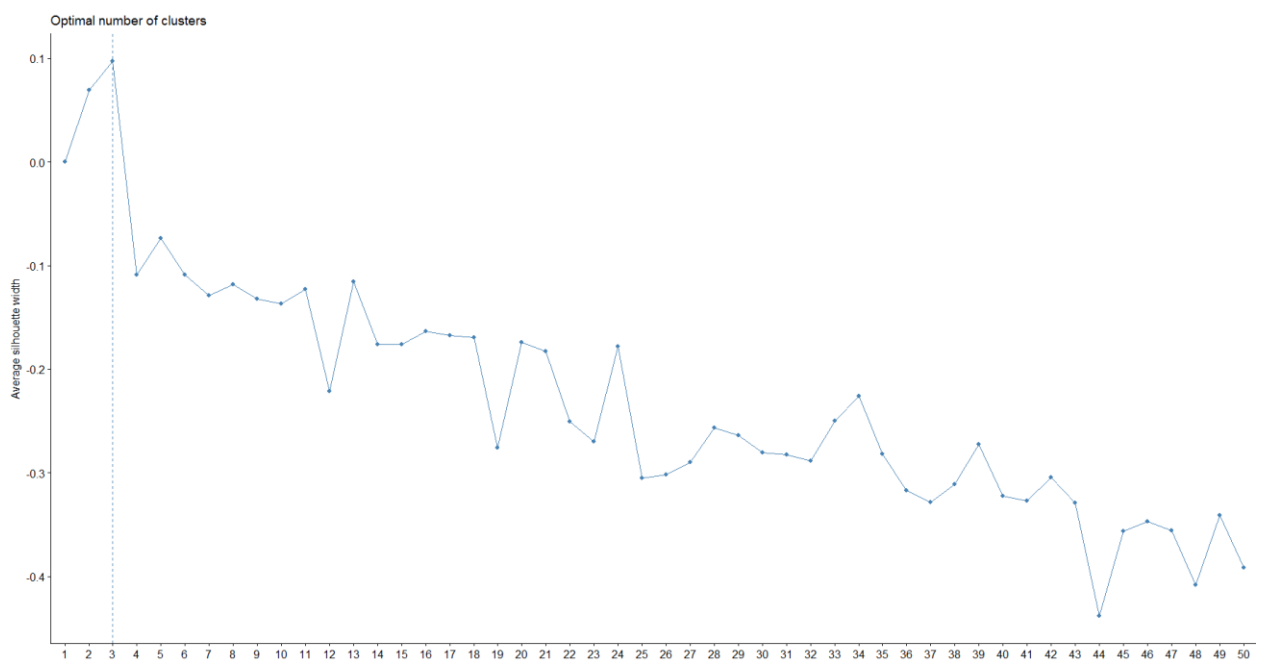


Df2: 94

Silhouette: 2

Df3: 17!?

Silhouette plot



2.8.3 UMAP algorithm

2.8.3.1 UWOT library

<https://github.com/jlmelville/uwot>

PRO = n_threads to use multicore with RccpParallel

CON = umap class is a matrix – no plotting direct feature

Laurent Querella – Ad Infinitum BI – All rights reserved

2.8.3.2 *UMAP library*

<https://cran.r-project.org/web/packages/umap/index.html>

tested on iris data set.

2.8.3.3 *UMPAR library*

<https://github.com/ropenscilabs/umapr>

error during install – 32-bit vs 64-bit

3 Python implementations

3.1 Density-based clustering

3.1.1 HDBSCAN

<https://hdbscan.readthedocs.io/en/latest/>

3.2 Dimension reduction

3.2.1 LargeVis algorithm

<https://github.com/lferry007/LargeVis>

<https://jlorince.github.io/viz-tutorial/>

git clone <https://github.com/lferry007/LargeVis.git>

sudo yum install python36-devel

sudo yum install gsl-devel

version 1.15

g++ LargeVis.cpp main.cpp -o LargeVis -lm -pthread -lgsl -lgslcblas -Ofast -march=native -ffast-math

sudo python setup.py install

Compil done in python 2x but failed in Python 3.5 (Aug18) sudo python3

<https://stackoverflow.com/questions/29109685/error-building-cython-with-python3-error-pystring-asstring-was-not-declared-i>

3.2.2 Triplet constraints

<https://pypi.org/project/trimap/>

TriMap is a dimensionality reduction method that uses triplet constraints to form a low dimensional embedding of a set of points. The triplet constraints are of the form “point i is closer to point j than point k”. The triplets are sampled from the high-dimensional representation of the points and a weighting scheme is used to reflect the importance of each triplet.

3.2.3 UMAP

3.2.3.1 References

<https://arxiv.org/abs/1802.03426>

<https://github.com/lmcinnes/umap>

<https://umap-learn.readthedocs.io/en/latest/>

3.2.3.2 Usage

- Dimension reduction coupled with clustering (HDBSCAN)
- Custom distance function (numba)
- Preprocessing transformer in a scikit-learn pipeline
- Supervised dimension reduction

See also (t-SNE/UMAP clustering):

<https://github.com/lmcinnes/umap/issues/25>

<https://iwatobipen.wordpress.com/2018/02/23/chemical-space-visualization-and-clustering-with-hdbscan-and-rdkit-rdkit/>

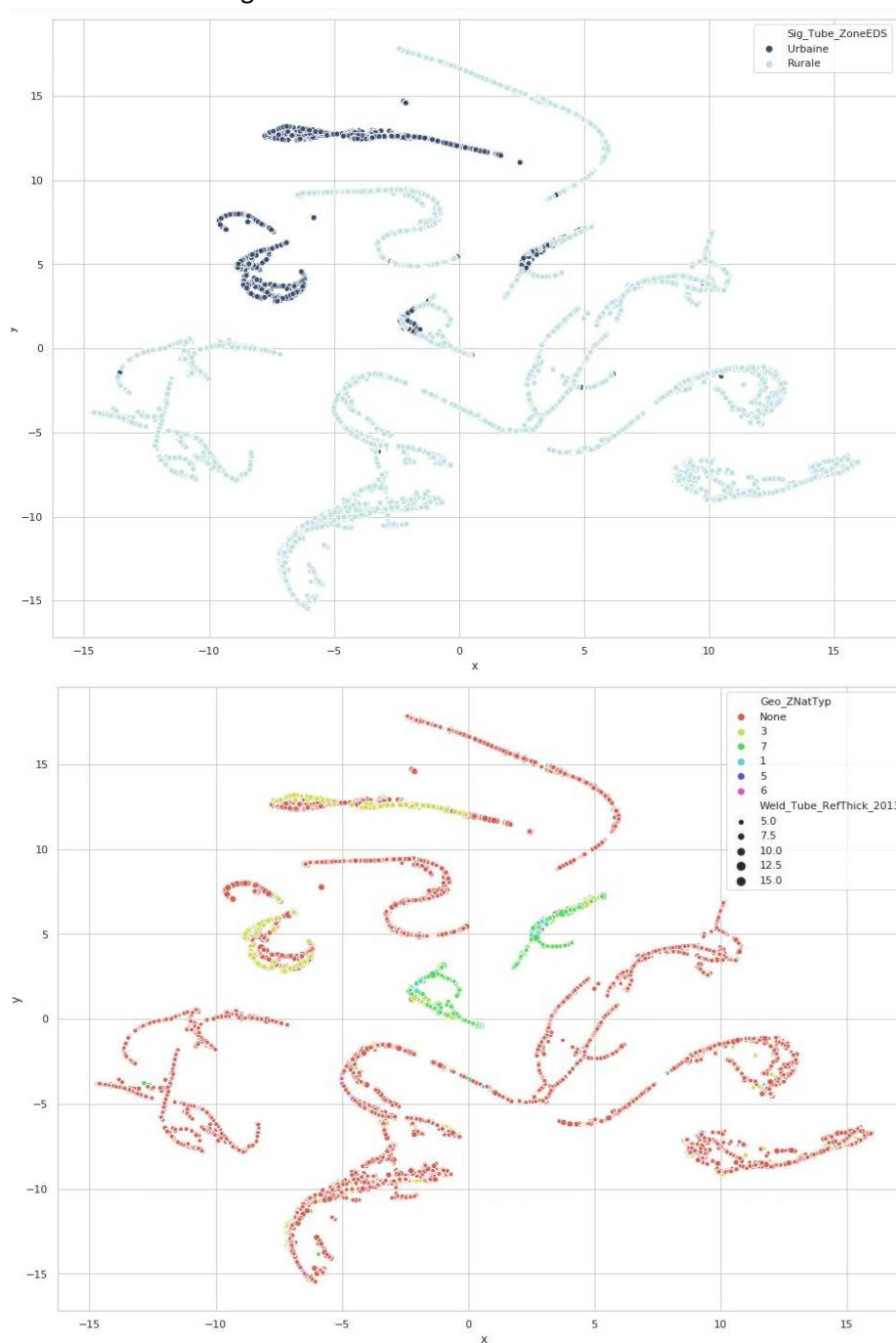
3.2.3.3 PLIF data

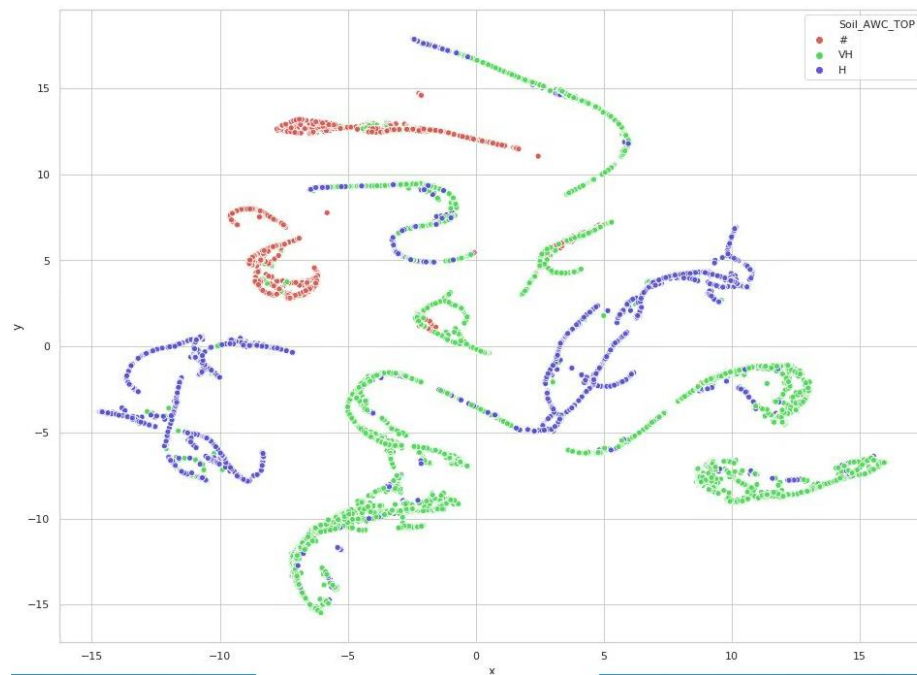
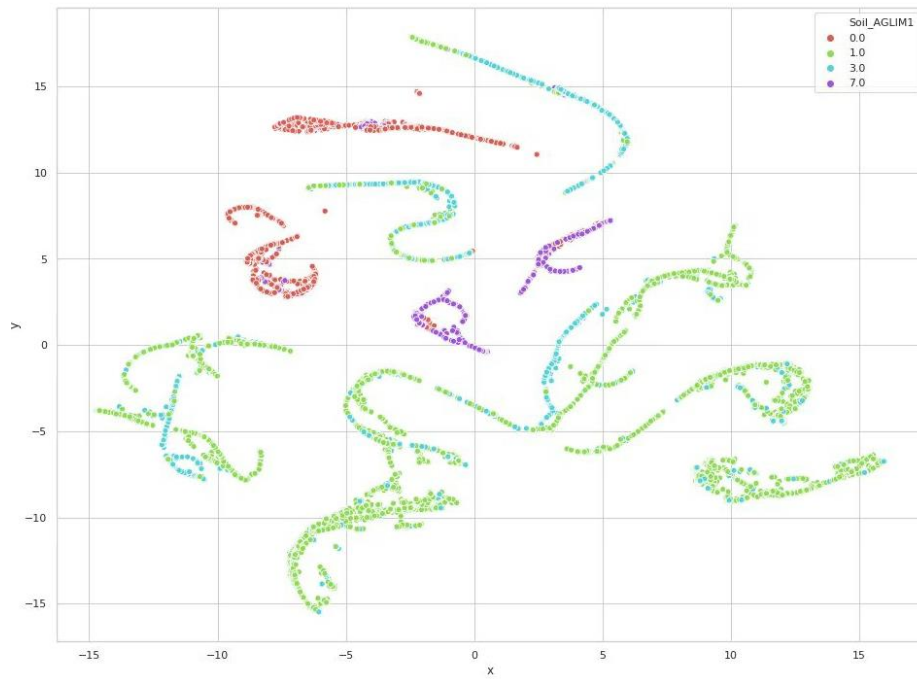
PLIF data: **UMAP - PLIF clustering - 03Aug18**

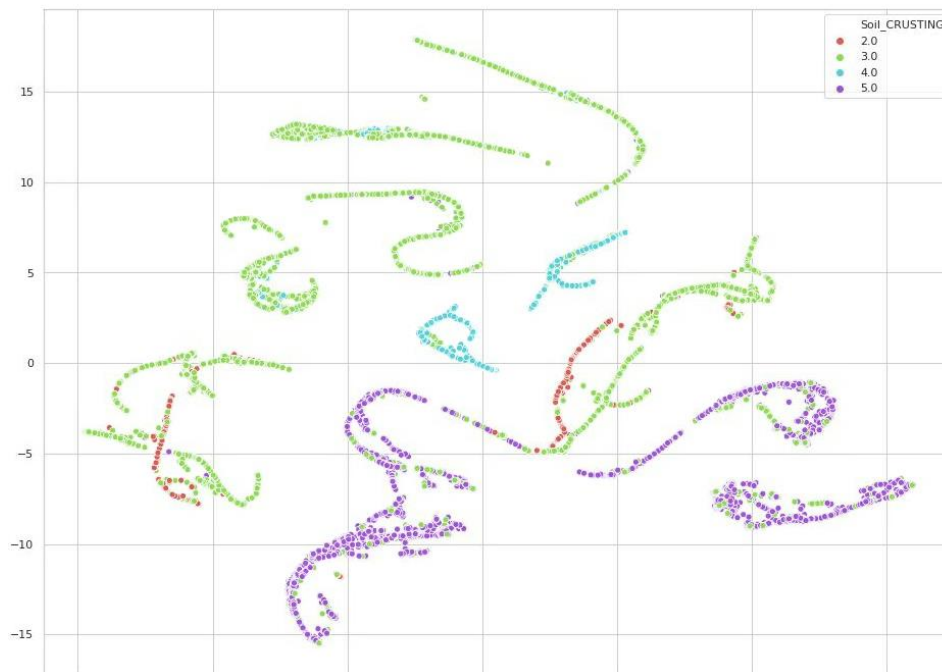
Select environment variables (not inspection)

3.2.3.3.1 ZONE1

1. Standard embedding in 2D





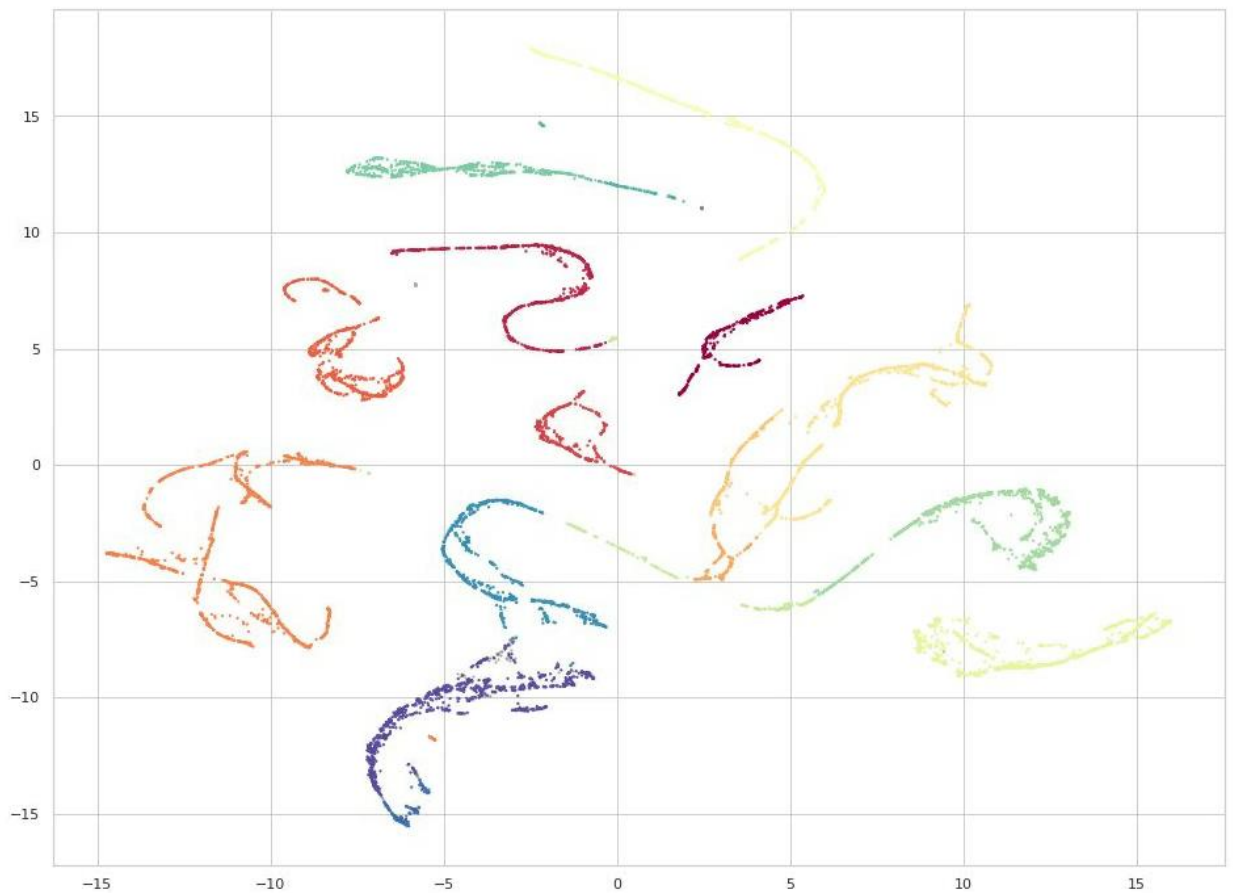


2. UMAP projection onto 20 dim

```
%%time
clusterable_embedding = umap.UMAP(
    n_neighbors=100,
    min_dist=0.0,
    # n_components=2,
    n_components=20,
    random_state=42,
).fit_transform(zone1.enc[col_list[2:]])
```

3. HDBSCAN clustering from projection

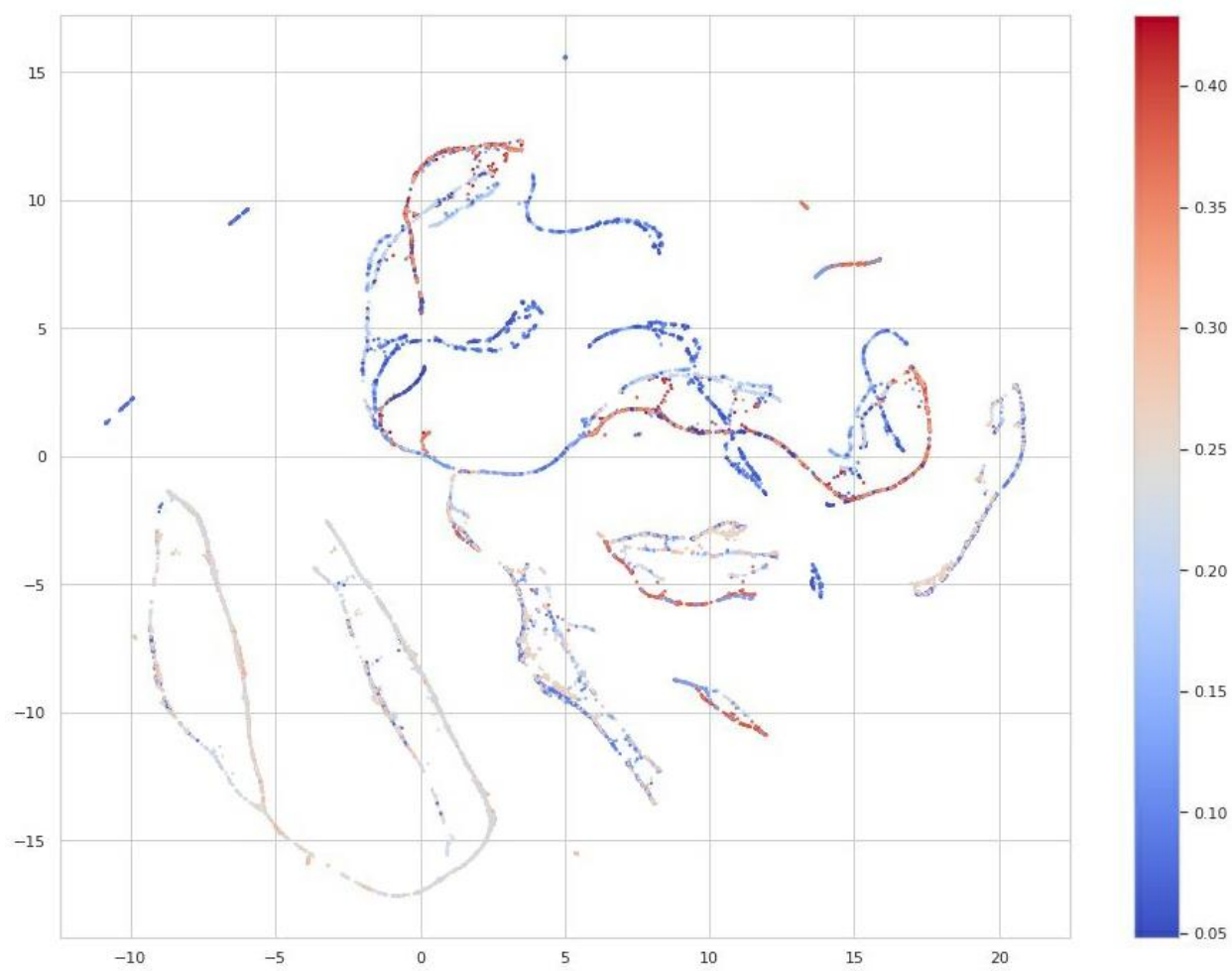
```
%time
labels = hdbscan.HDBSCAN(
    min_samples=15,
    min_cluster_size=100,).fit_predict(clusterable_embedding)
```

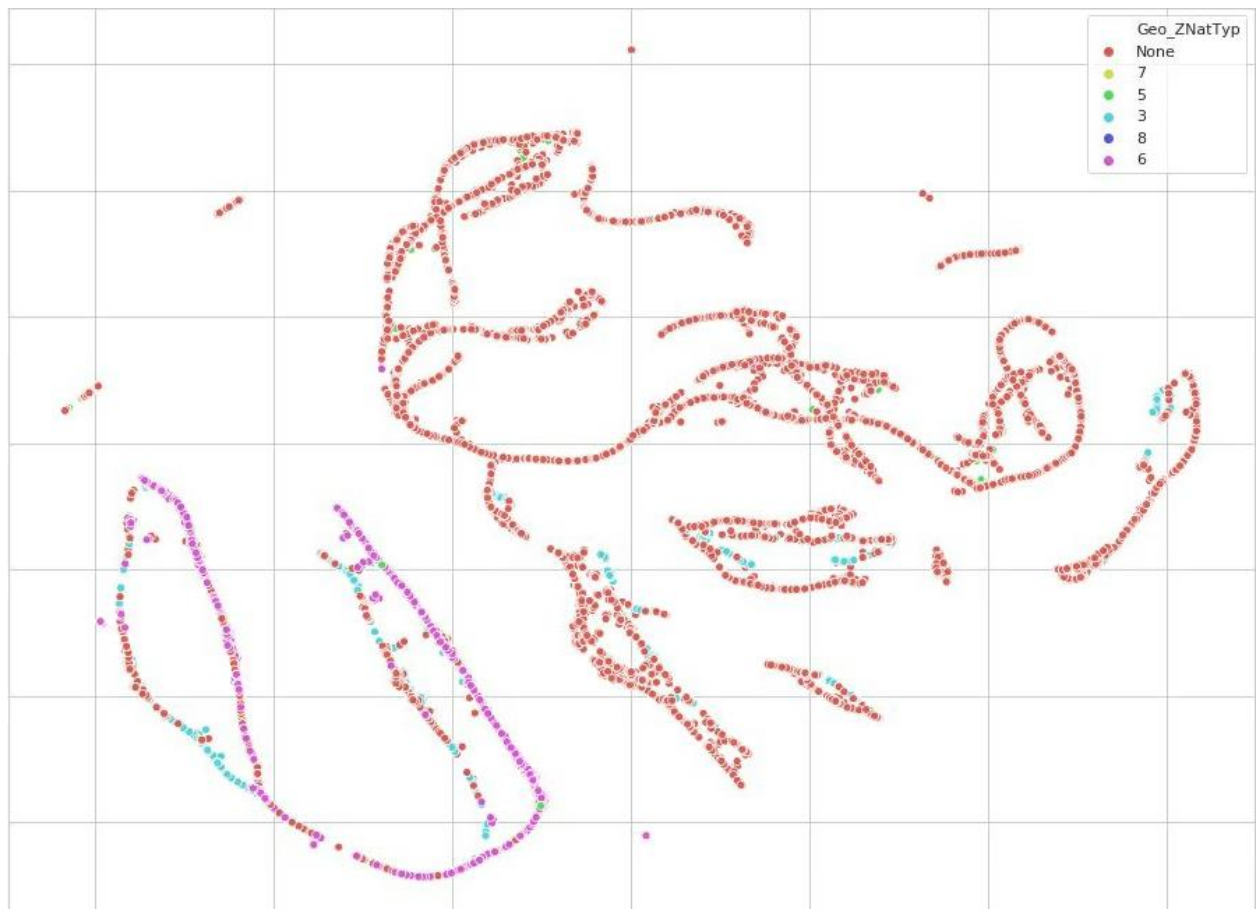
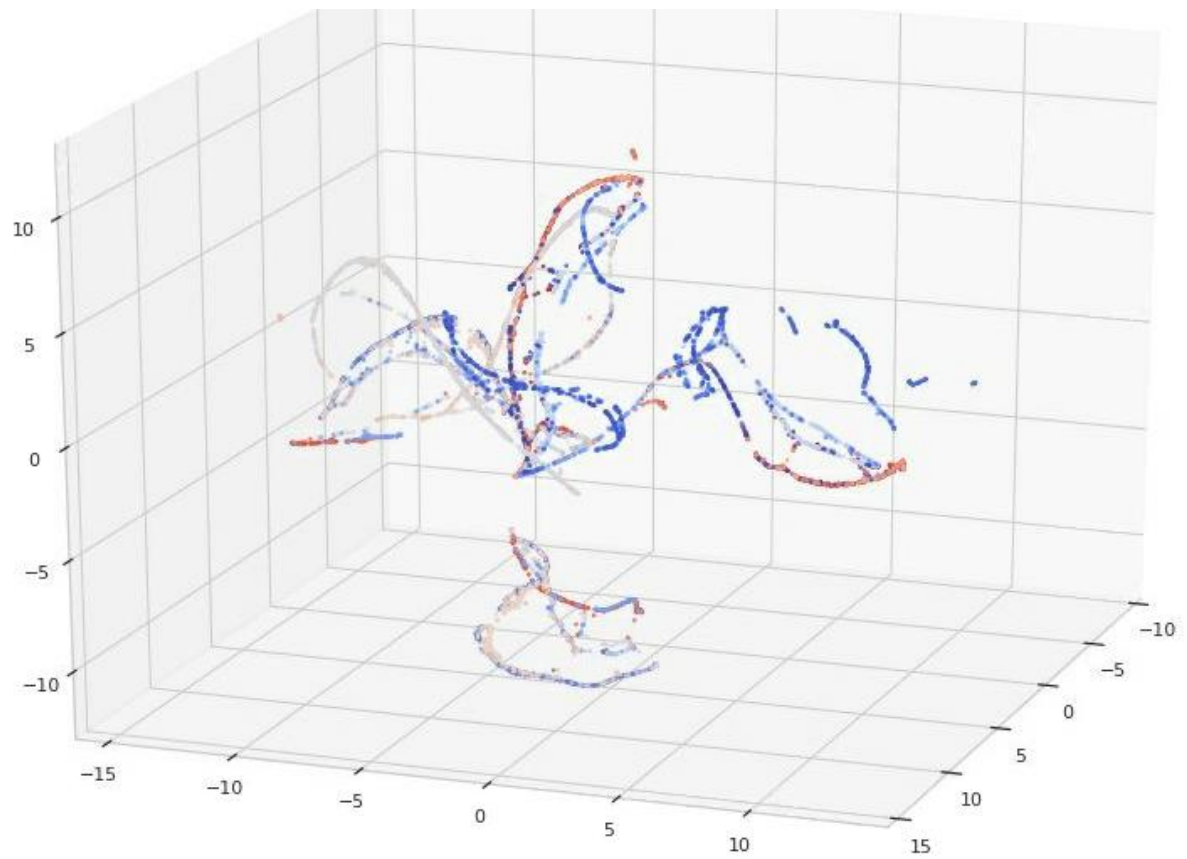


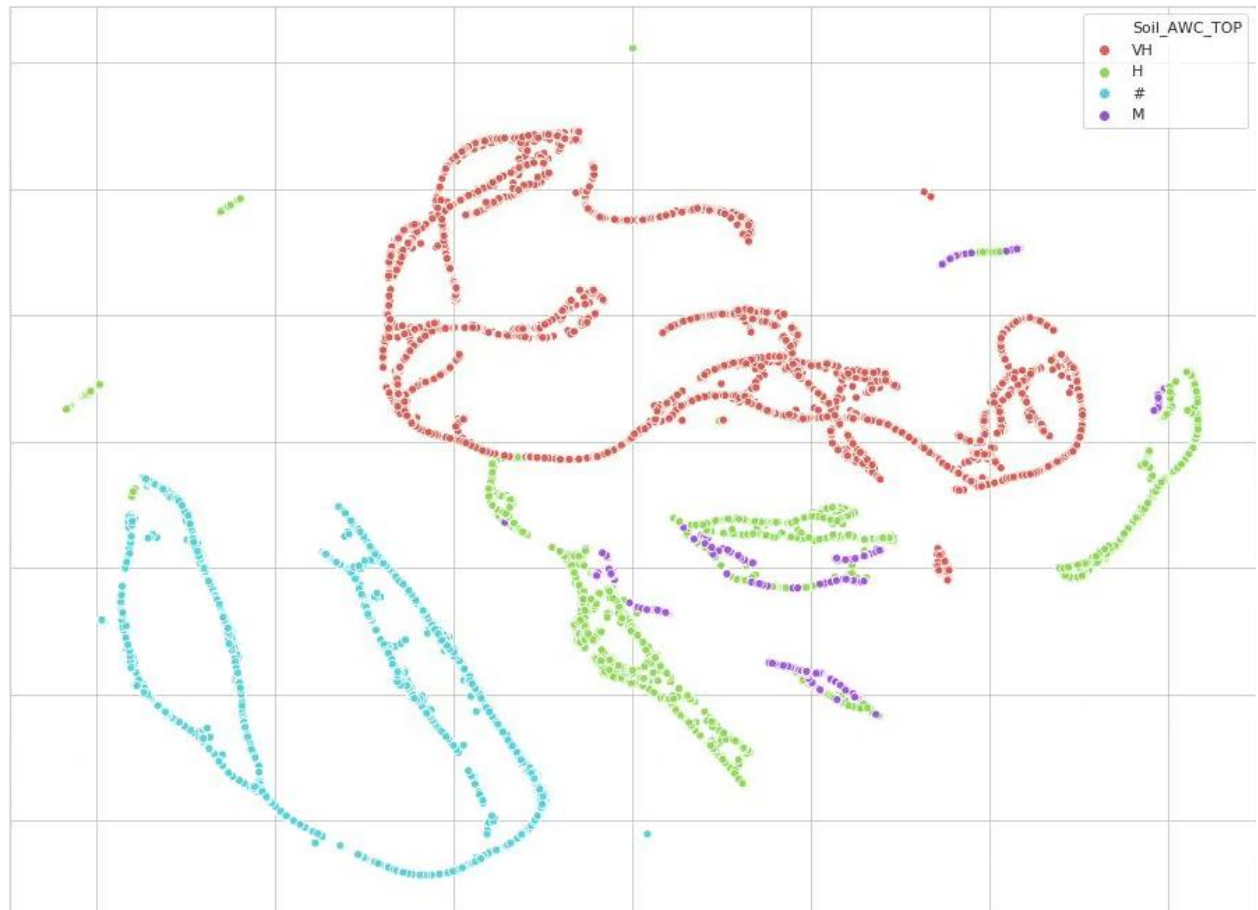
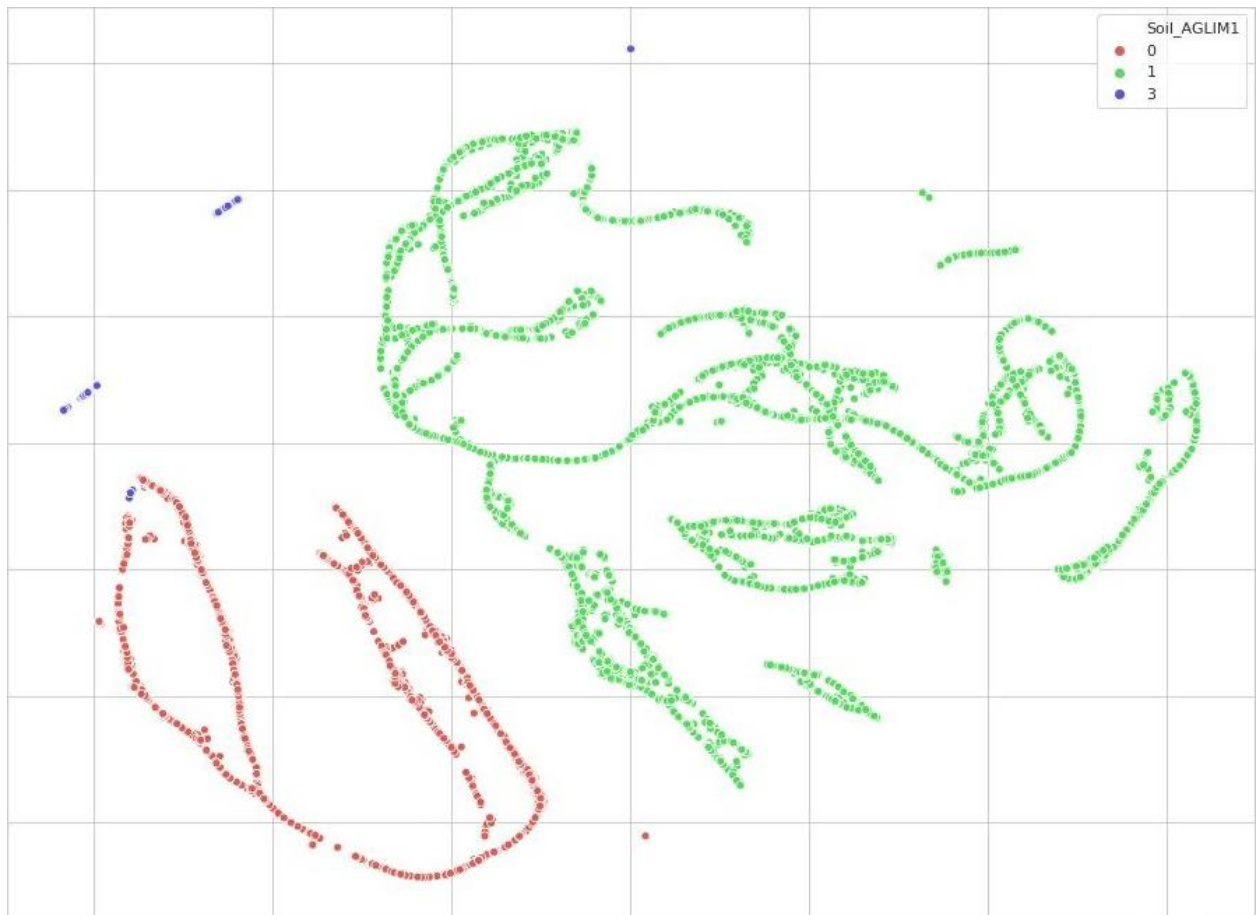
4. Merge (concat) PLIF dataset with clusters

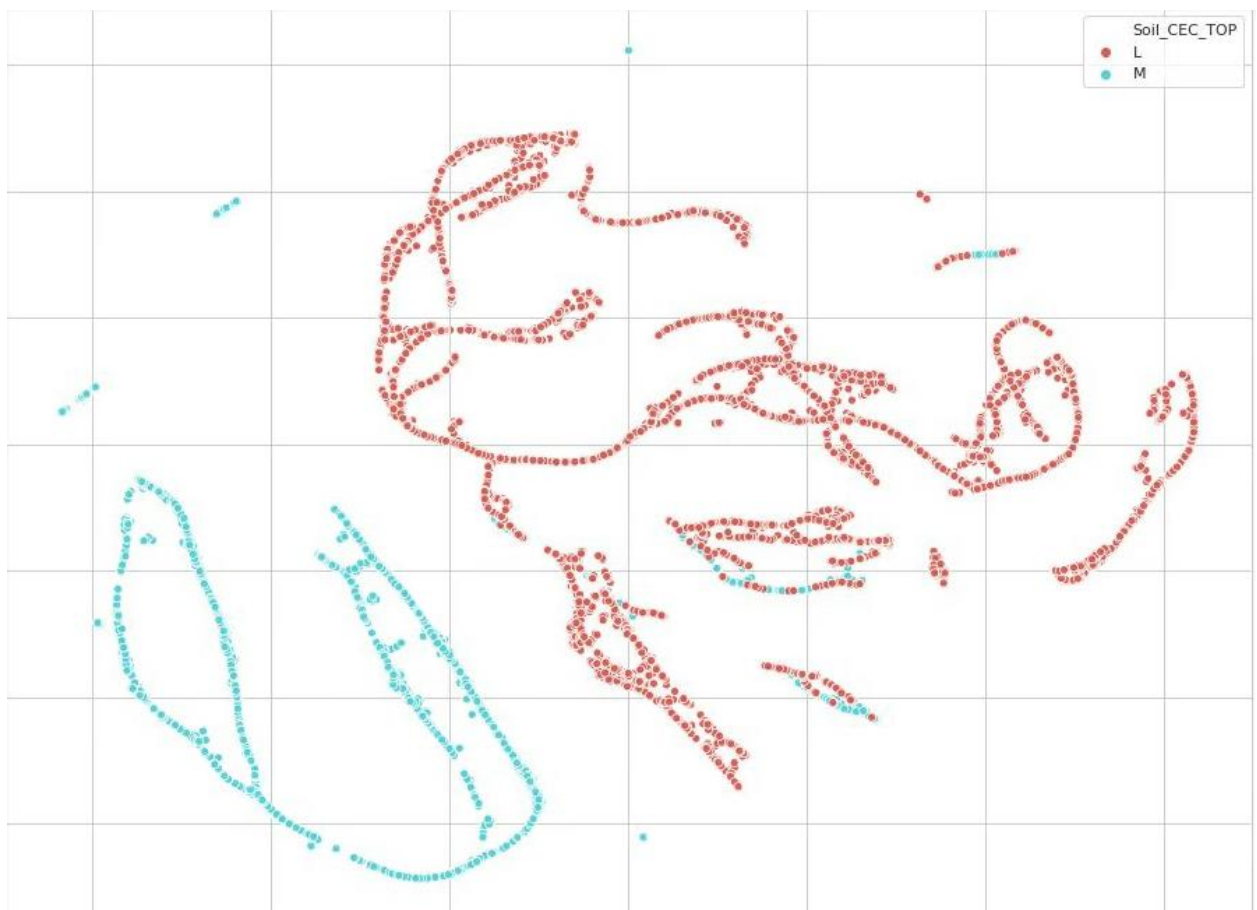
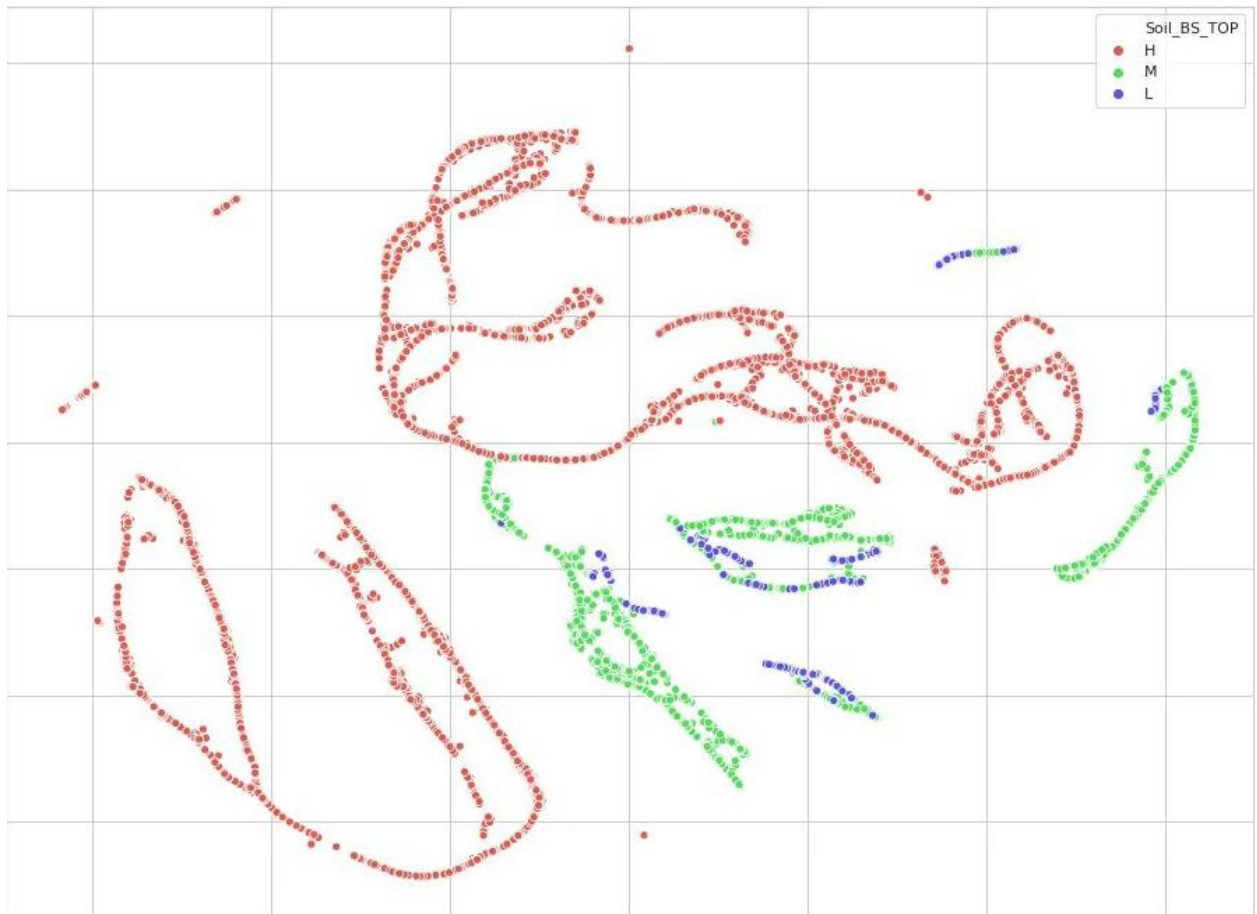
3.2.3.3.2 ZONE2

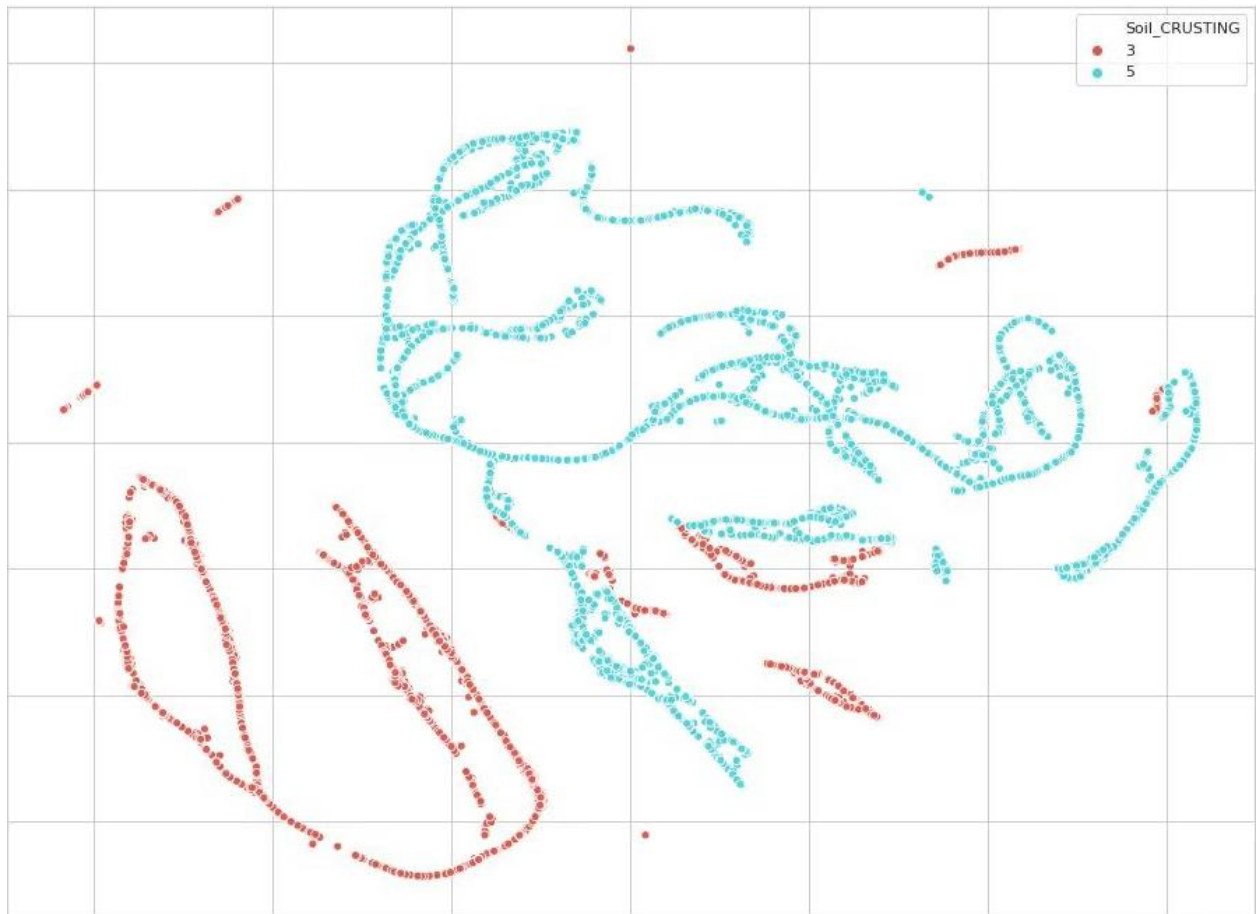
Standard UMAP

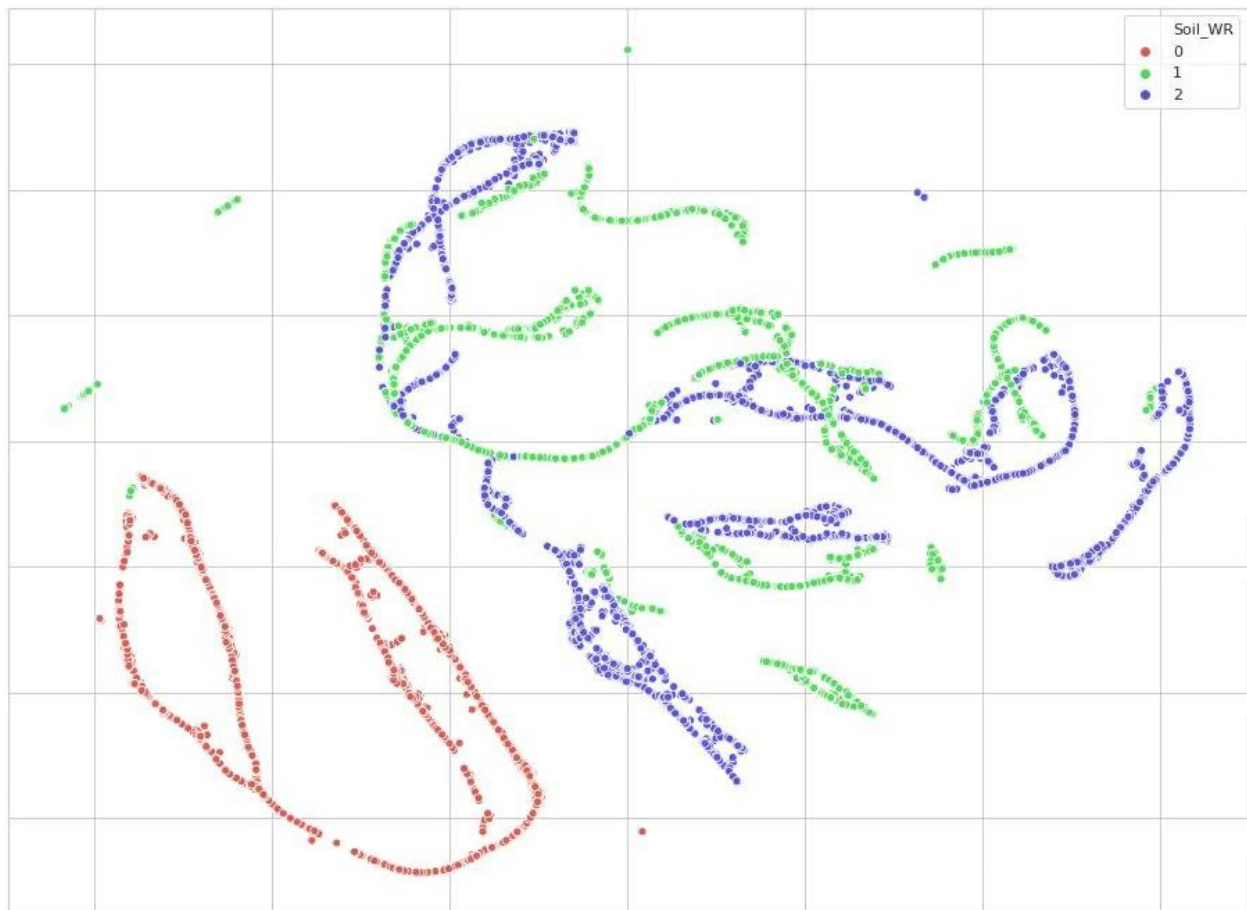












UMAP+HDBSCAN

```
%%time
clusterable_embedding = umap.UMAP(
    n_neighbors=100,
    min_dist=0.0,
    # n_components=2,
    n_components=20,
    random_state=42,
).fit_transform(zone1.enc[col_list[2:]])
```

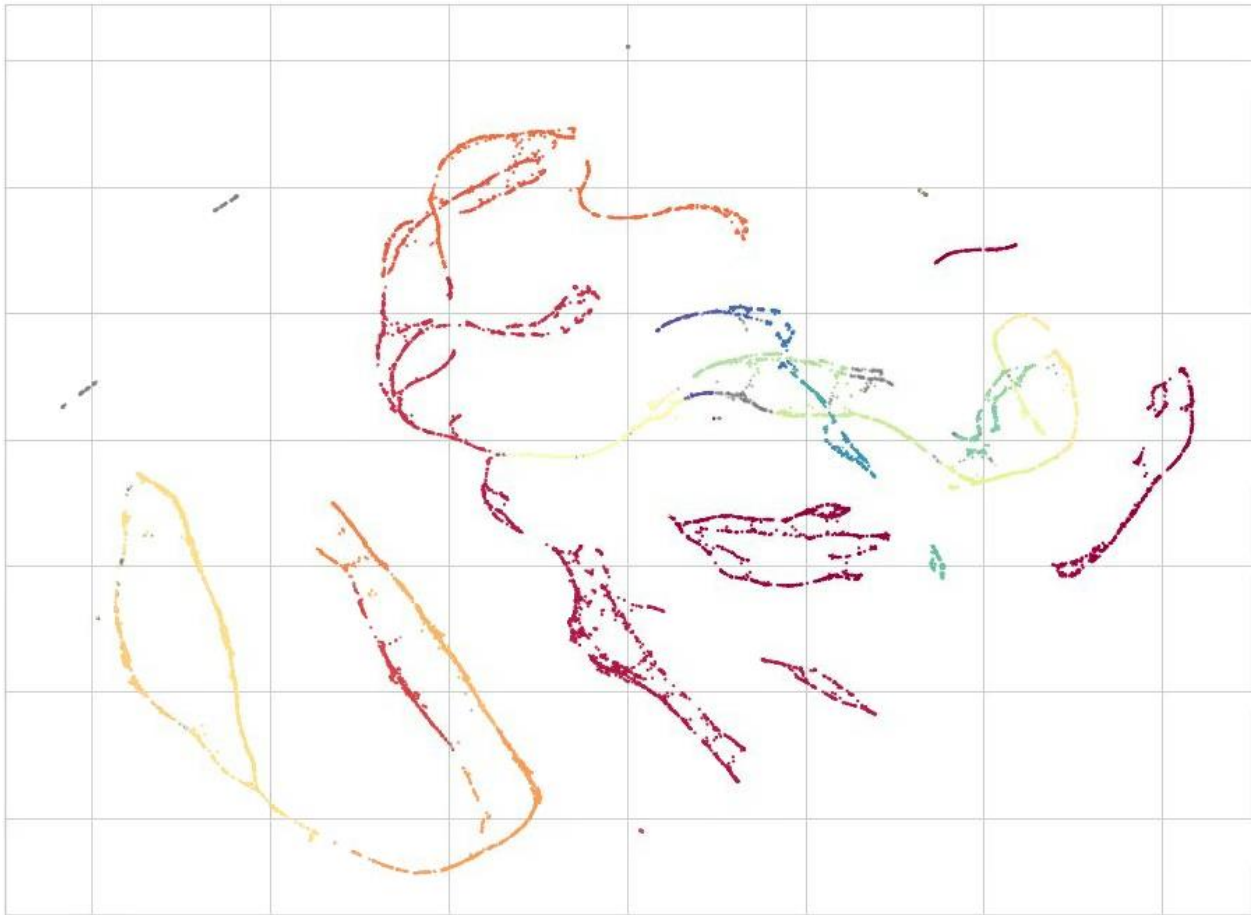
CPU times: user 1min 10s, sys: 6.48 s, total: 1min 17s
Wall time: 1min 8s

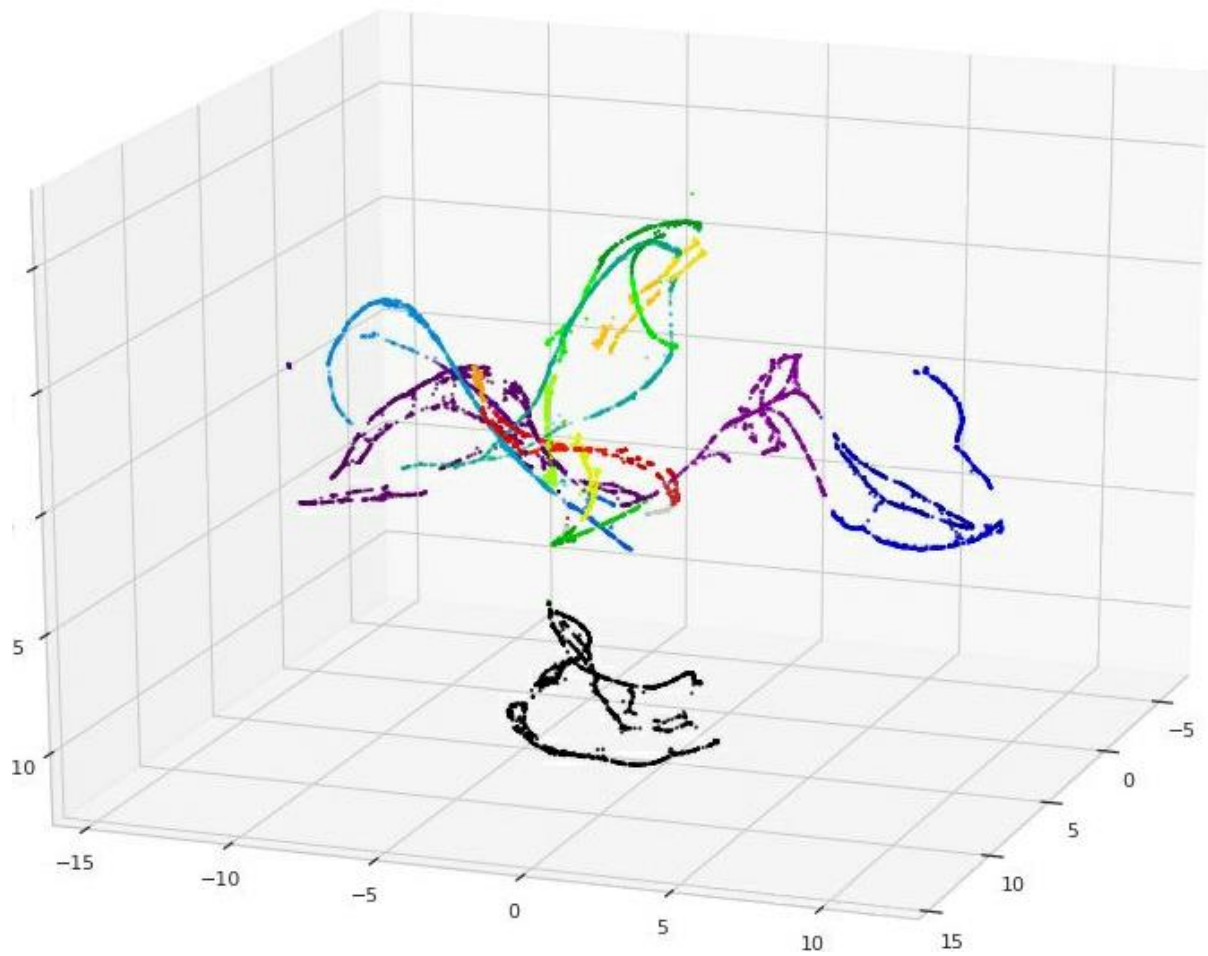
```
clusterable_embedding.shape
```

```
(12497, 20)
```

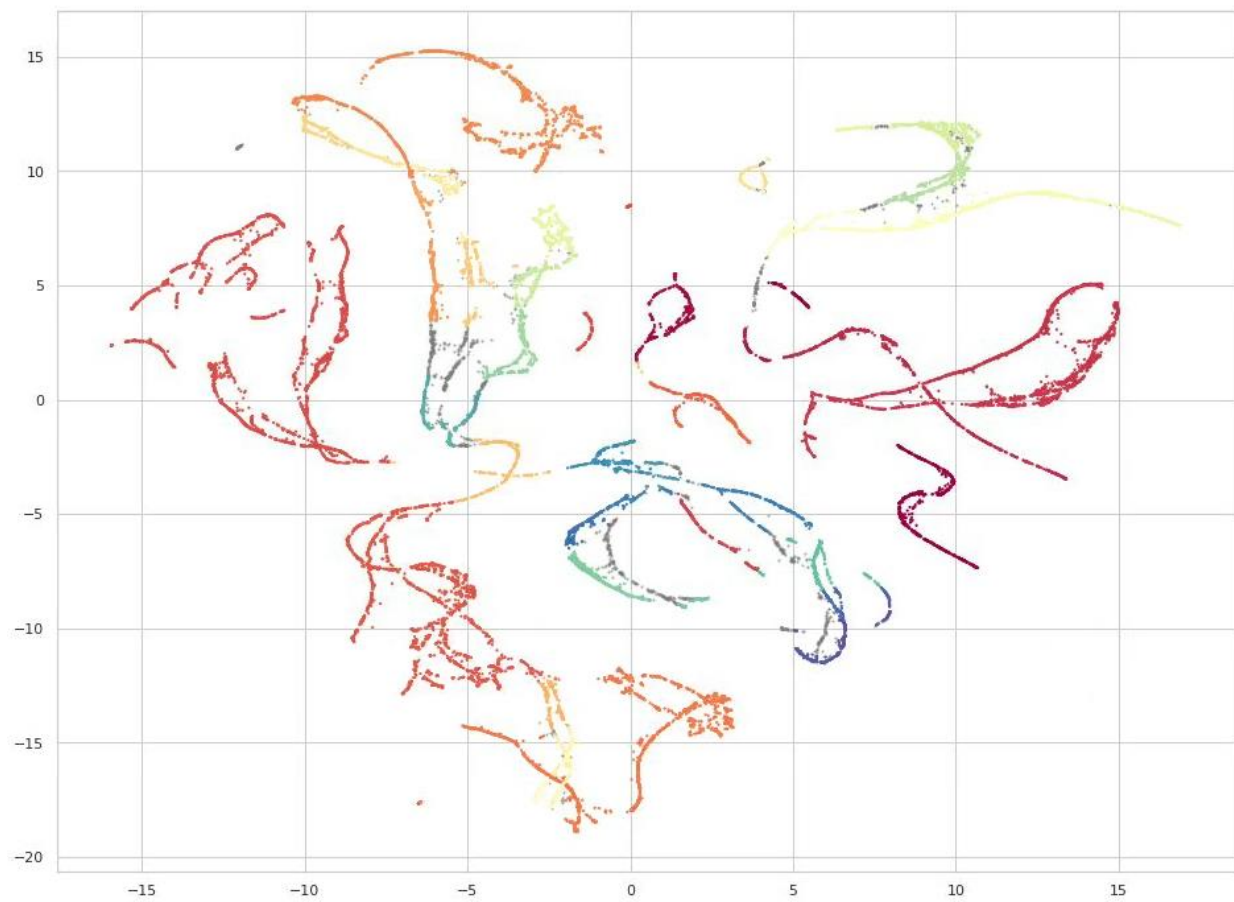
The next step is to cluster this data. We'll use HDBSCAN again, with the same

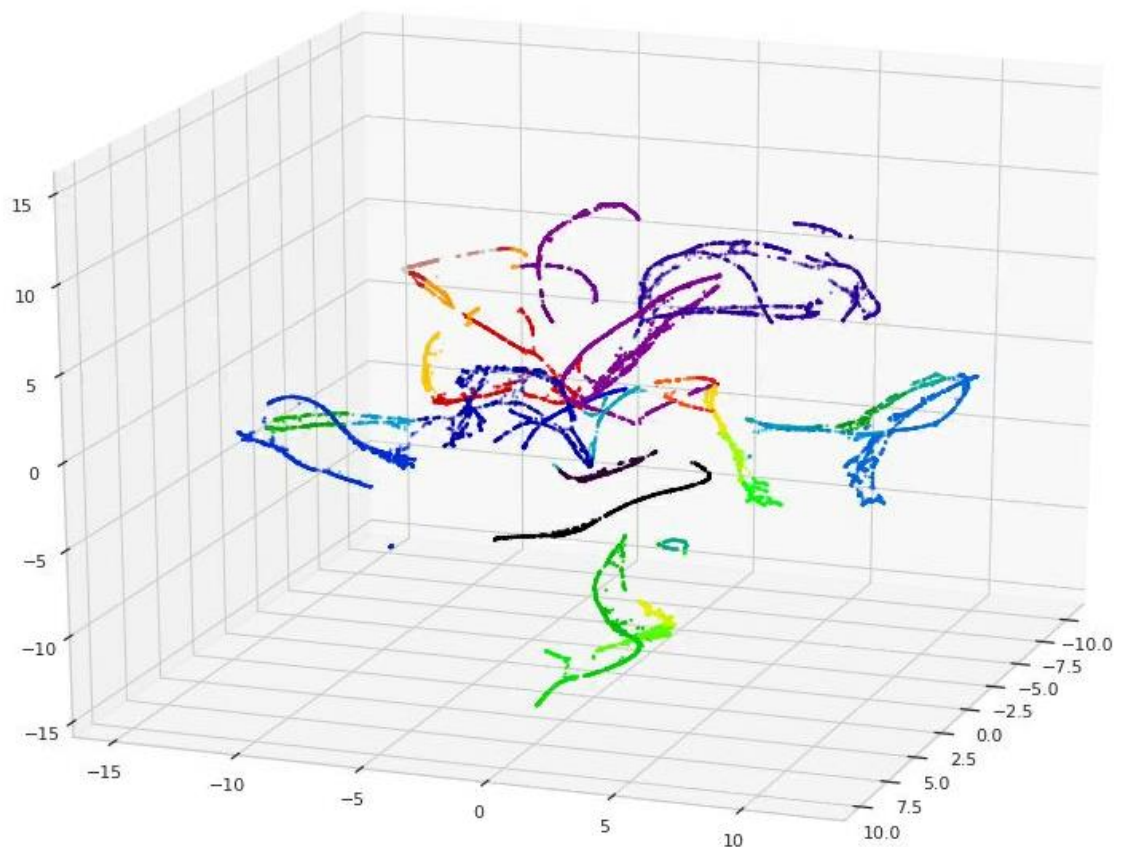
```
%%time
labels = hdbscan.HDBSCAN(
    min_samples=15,
    min_cluster_size=100,).fit_predict(clusterable_embedding)
```





3.2.3.3.3 ZONE1+2





```
%%time
clusterable_embedding = umap.UMAP(
    n_neighbors=100,
    min_dist=0.0,
    # n_components=2,
    n_components=30,
    random_state=42,
).fit_transform(zone1.enc[col_list[2:]])
```

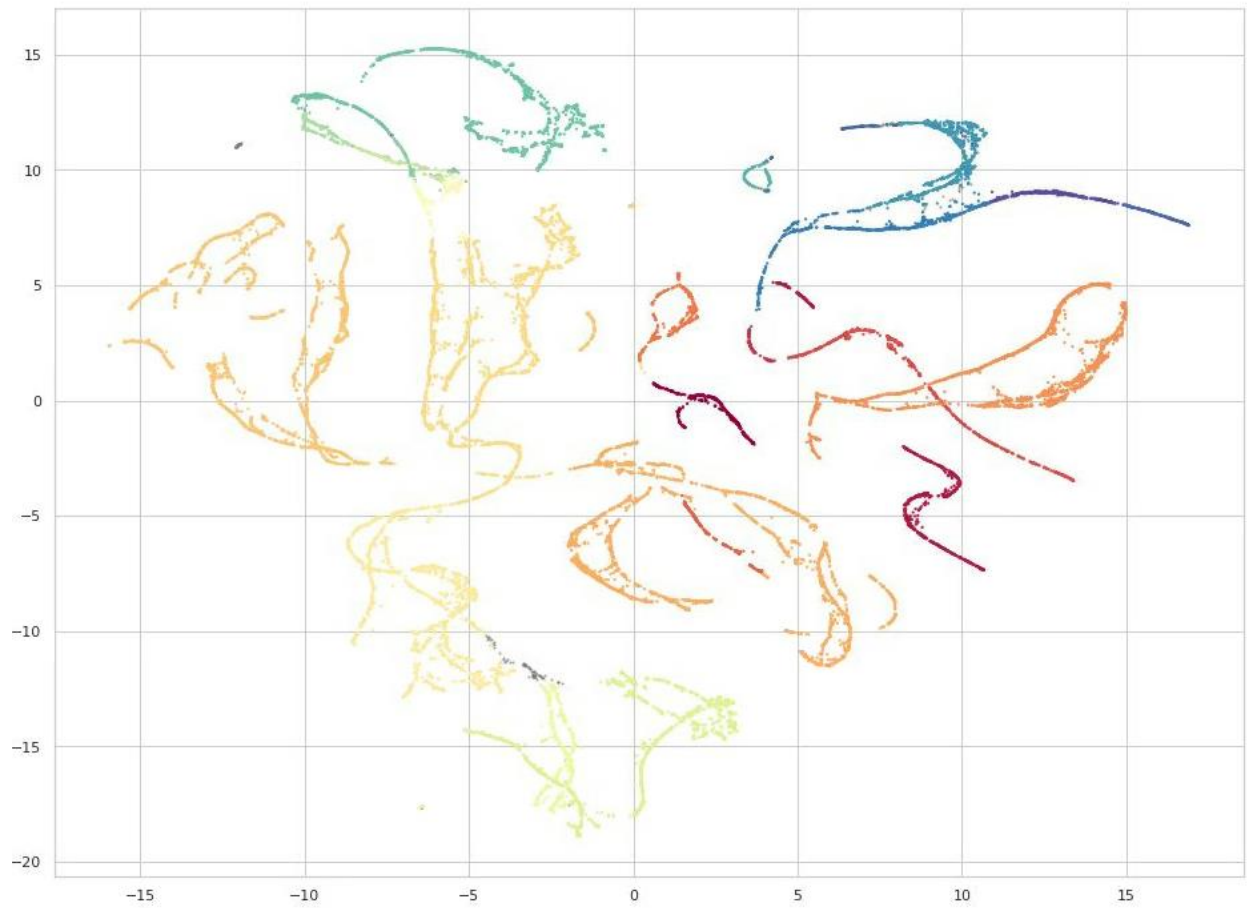
CPU times: user 3min 2s, sys: 17.2 s, total: 3min 19s
Wall time: 2min 54s

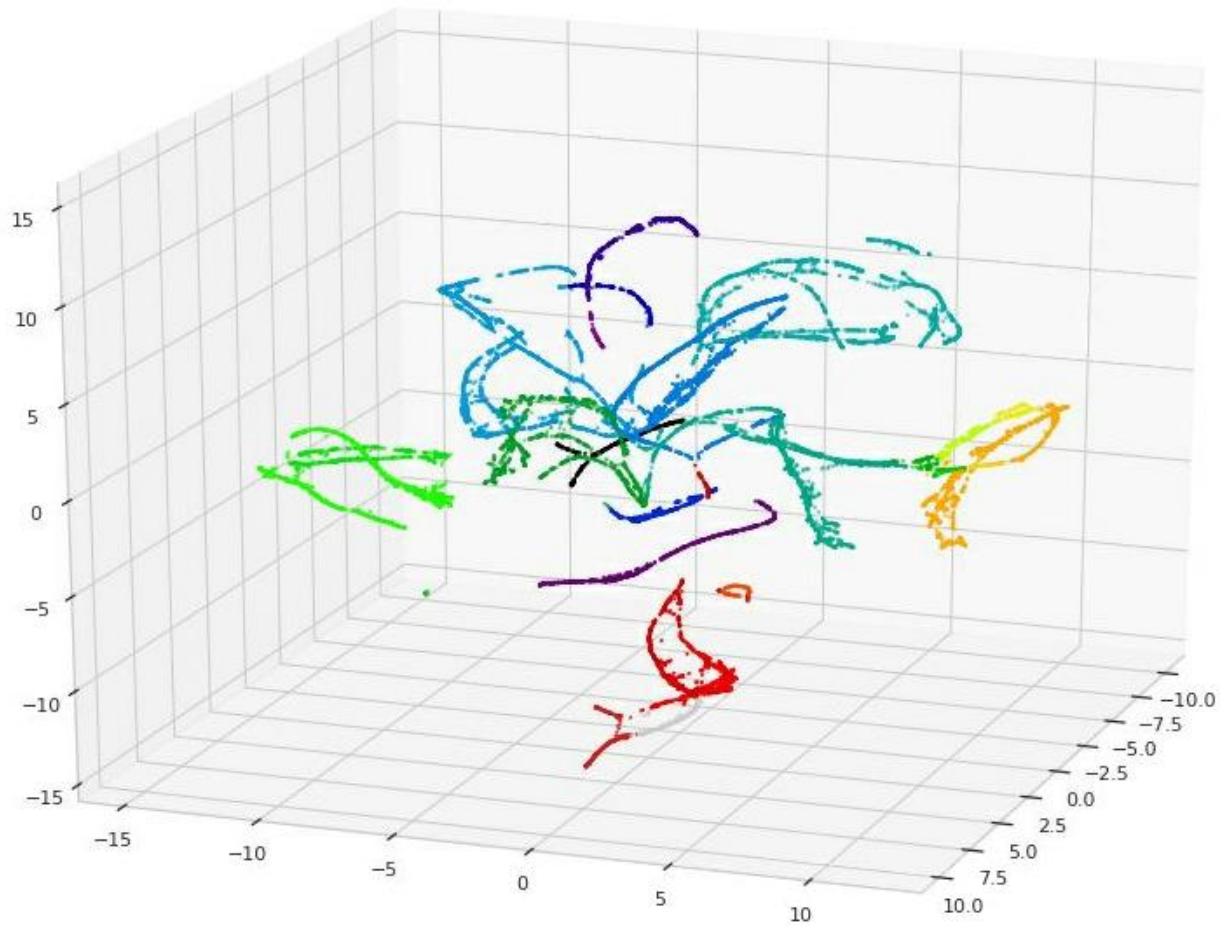
```
clusterable_embedding.shape
```

```
(24838, 30)
```

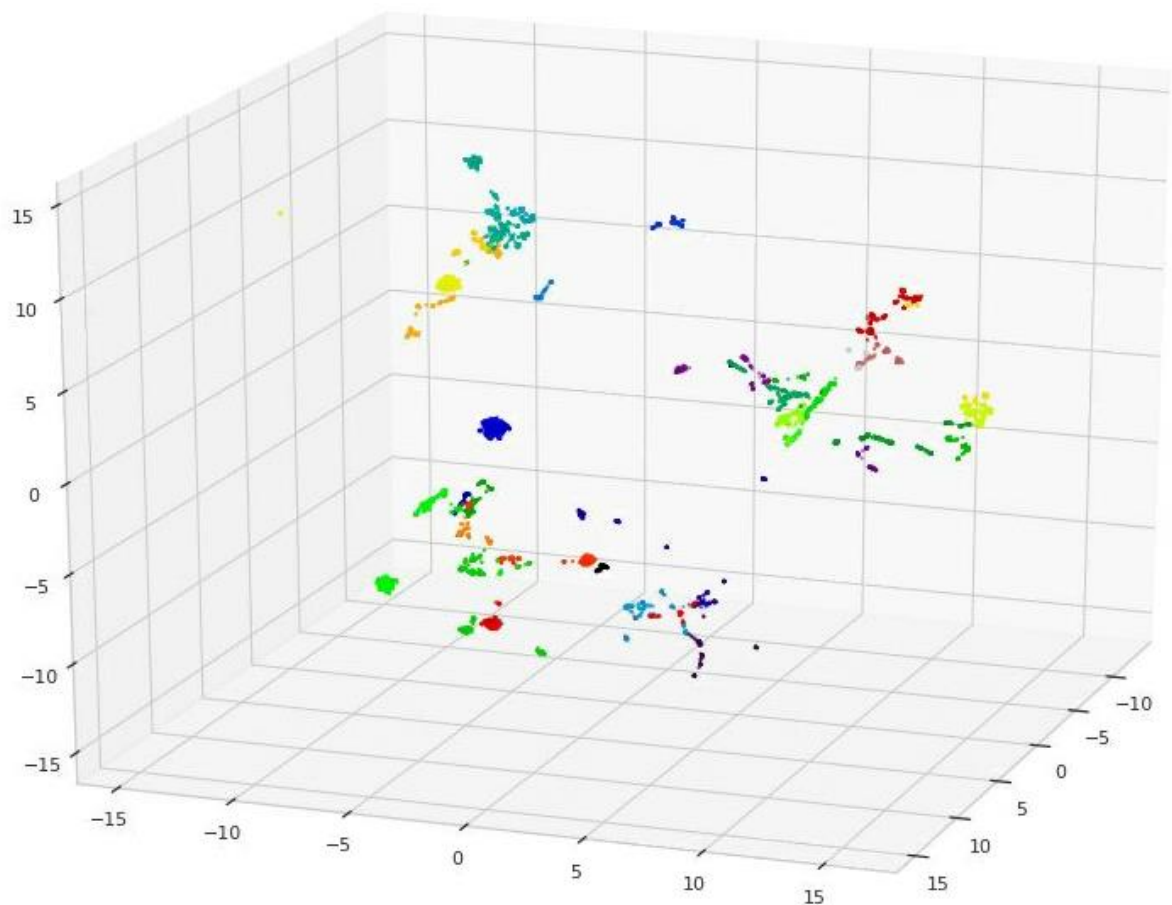
The next step is to cluster this data. We'll use HDBSCAN again, with the same

```
%%time
labels = hdbscan.HDBSCAN(
    min_samples=15,
    min_cluster_size=100,).fit_predict(clusterable_embedding)
```





Testing with binary metrics like hamming (only acting on binary columns resulting from one-hot encoding)



38 clusters on ZONE2

```
%%time
clusterable_embedding = umap.UMAP(
    n_neighbors=100,
    min_dist=0.0,
    # n_components=2,
    n_components=20,
    metric='hamming',
    random_state=42,
).fit_transform(zone1.enc[col_list[5:]])

/anaconda/envs/py35/lib/python3.5/site-packages/umap/spectral.
meta-embedding (experimental)
n_components

clusterable_embedding.shape
```

The next step is to cluster this data. We'll use HDBSCAN again, with the same

```
%%time
labels = hdbscan.HDBSCAN(
    min_samples=15,
    min_cluster_size=100,).fit_predict(clusterable_embedding)
```

<https://docs.scipy.org/doc/scipy/reference/spatial.distance.html>

