Zoo Clustering

Evaluating unsupervised learning algorithms over a simple categorical dataset

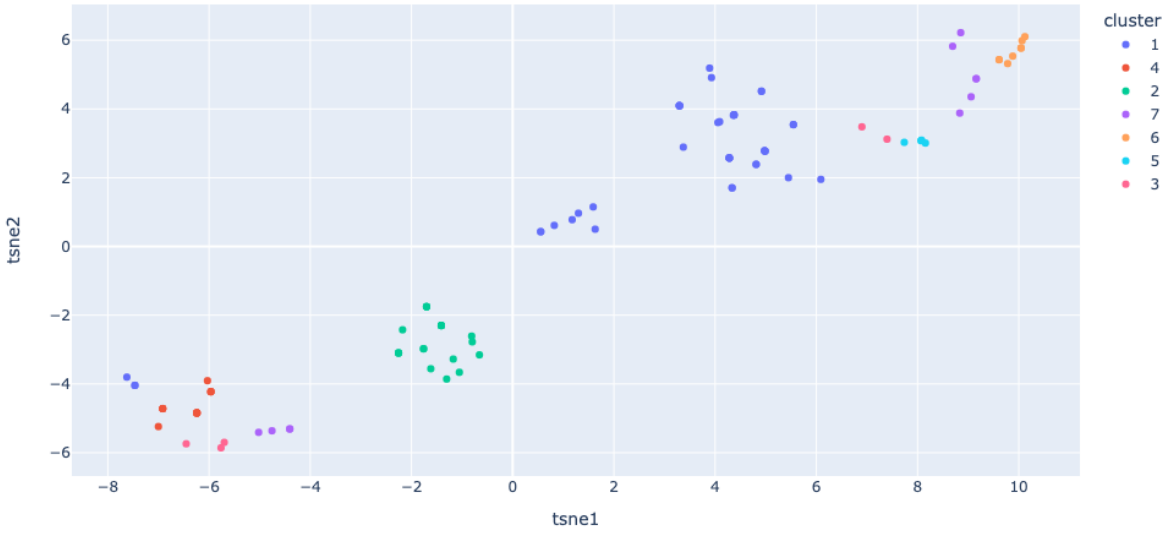
The project aims at assessing some of the most known clustering algorithms by applying them to a simple categorical dataset. The dataset can be downloaded at https://www.kaggle.com/datasets/uciml/zoo-animal-classification. We'll use "class.csv" as well as "zoo.csv", which is the main dataset.
The dataset contains 101 samples and 18 features. With the exception of "animal_name", all features can be considered to be categorical features ("legs" included, even if it's not a binary feature). Our target feature is "class_type". We can associate class types with meaningful labels by looking at "class.csv". For example, class_type 1 refers to mammals.
We carried out an assessment of the clustering algorithm with two visualization methods and a scikit-learn metric (adjusted Rand index).

The project core consists of three classes: Manager, Algorithm and Visualizer.
Manager objects are provided with methods to manage the dataset:

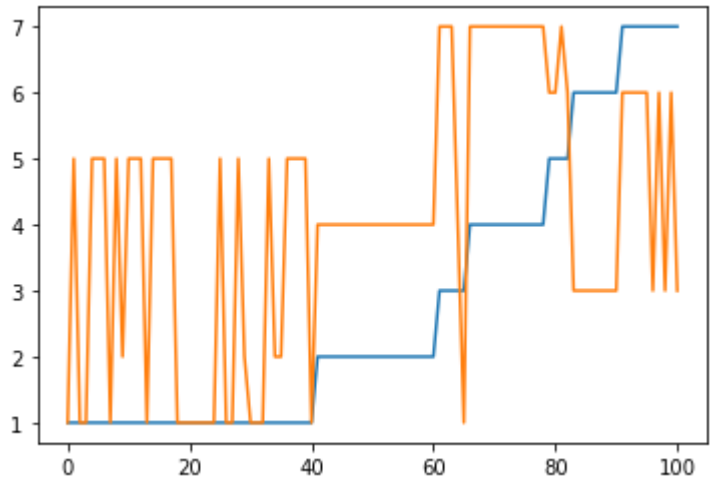
- new creates a pandas dataframe from a .csv file
- truth isolates the target feature and puts it in a numpy array
- drop drops columns from the dataframe
- standardize standardizes the whole dataframe
- np converts a dataframe into a numpy array

The **Algorithm** class contains a method for each of the algorithms we're using. Each method gives an array with the ordered cluster memberships as output.
Visualizer objects are used to compare the ground truth with the models' output, and they do so with four methods: stair and lineplot use matplotlib to plot the target and output arrays for comparison; tsne and update_tsne use TSNE dimensionality reduction and plotly.express to create and update a scatterplot with info about how an algorithm classified the samples and what's their true cluster membership. We'll provide graphical examples of these two visualization methods in this pdf.

Once our classes have been defined, we can start the actual analysis. A Manager object ("m") is used to create and clean the dataframe. The df is then plotted using a Visualizer ("v")'s tsne method, using the target array to assign clusters. Here is the result we obtain with the random_state parameter set to 0.

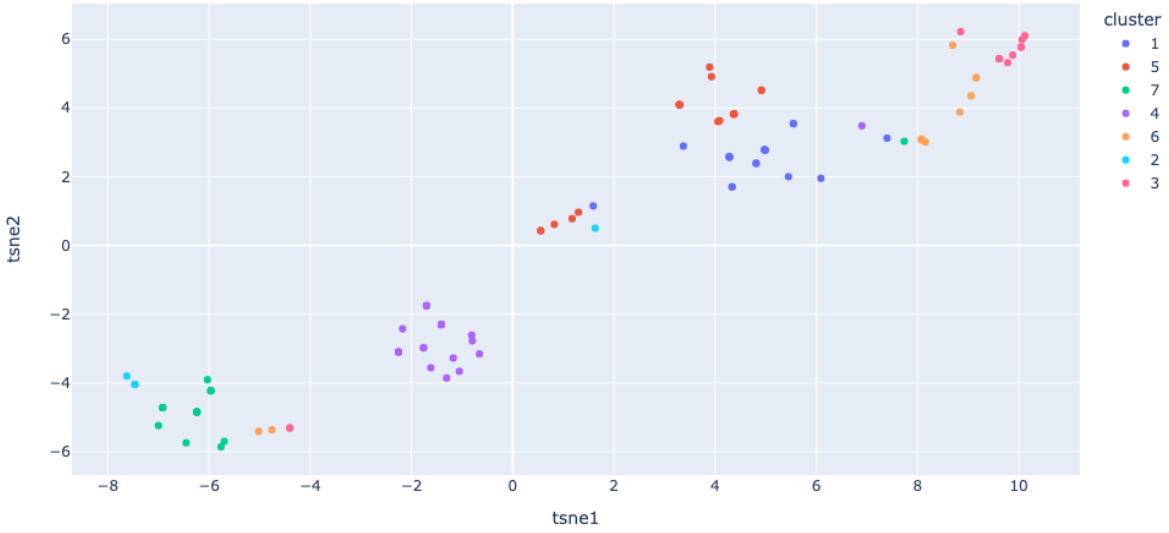


Now we can start using an Algorithm object (“a”). Since the dataset we’re working with is categorical, our first guess is to avoid the standard algorithms and use KModes instead. KModes is specifically designed to cluster categorical variables. We run the algorithm with a and visualize its outputs with v. Here’s a plot of the result in a matplotlib line plot:



How to read this plot: The blue line plots an array containing all the true class memberships of the 101 samples, ordered by class. The orange line plots the kmodes_output array, but it’s rearranged so that the samples follow the order of the blue line indexes. This means that if sample 40 of the “blue” array is “elephant”, sample 40 of the “orange” array is “elephant” too. The perfect clustering would then look like a pair of stairs made of completely parallel steps (the steps wouldn’t necessarily overlap because the clustering algorithm assigns numbers to their clusters without a particular order). For instance, knowing that step 2 corresponds to birds, this plot shows us that KModes can perfectly cluster all the birds (because the orange step 2 is completely parallel to the blue one and no other sample is assigned to class 4 in the “orange” array).

After the lineplot, we can use v to update our tsne plot so that the colors of the dots follow kmodes_output.



The last step will be to assess how well KModes performed using the adjusted Rand index (mathematical formulation available at https://scikit-learn.org/stable/modules/clustering.html#mathematical-formulation).

All the other clustering algorithms are not specifically designed for categorical variables, so we'll use the standardize method to change the dataset before moving on.

All the other clustering algorithms (KMeans, AffinityPropagation, MeanShift, SpectralClustering, HierarchicalClustering, DBSCAN, OPTICS, and Birch) are part of Scikit-learn and will be assessed as shown above.

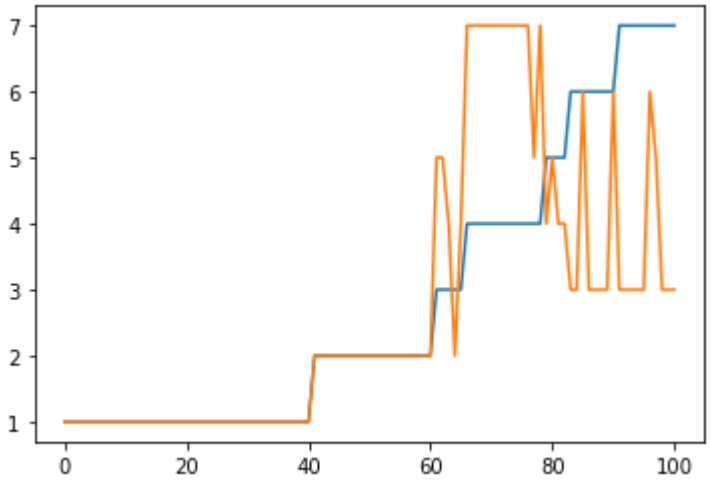
Notes:

- KMeans turns out to be more efficient than KModes, in our case
- AffinityPropagation and MeanShift don't allow us to set the number of clusters hyperparameter and they're particularly bad at solving our clustering problem. DBSCAN and OPTICS, though they have the same limitation, create exactly 7 clusters.
- AffinityPropagation doesn't always converge with our dataset
- Hierarchical Clustering and DBSCAN always perform the same clustering
- Birch is the most efficient clustering algorithm for our dataset

Here's a score table from an execution of our program.

	algorithm	adjusted Rand score
0	KModes	0.588353
1	KMeans	0.735990
2	AffinityPropagation	0.480357
3	MeanShift	0.034061
4	SpectralClustering	0.866478
5	HierarchicalClustering	0.910951
6	DBSCAN	0.910951
7	OPTICS	0.319007
8	Birch	0.924848

Note: a score of 0.92 doesn't mean that 92% of the samples were classified in the right way, because this information is simply not possible to obtain. A score of 1.0, though, means that the algorithm converges to a flawless clustering. This is how the Birch execution with adjusted_rand_score of 0.924848 is plotted with our Visualizer.



Final observation: There are two main problems with our dataset, resulting in an imperfect clustering. The first one is the limited number of samples. The second problem lies in the ground truth itself. Since the animal taxonomy is defined by humans, it is possible that it doesn't always correspond to the animal features in a way that an algorithm could predict. In any case, our algorithms already did a great job from this point of view: the Birch algorith plotted above perfectly clustered dolphins, whales and platypuses as mammals!

Bomba Monica, Mosconi Mattia