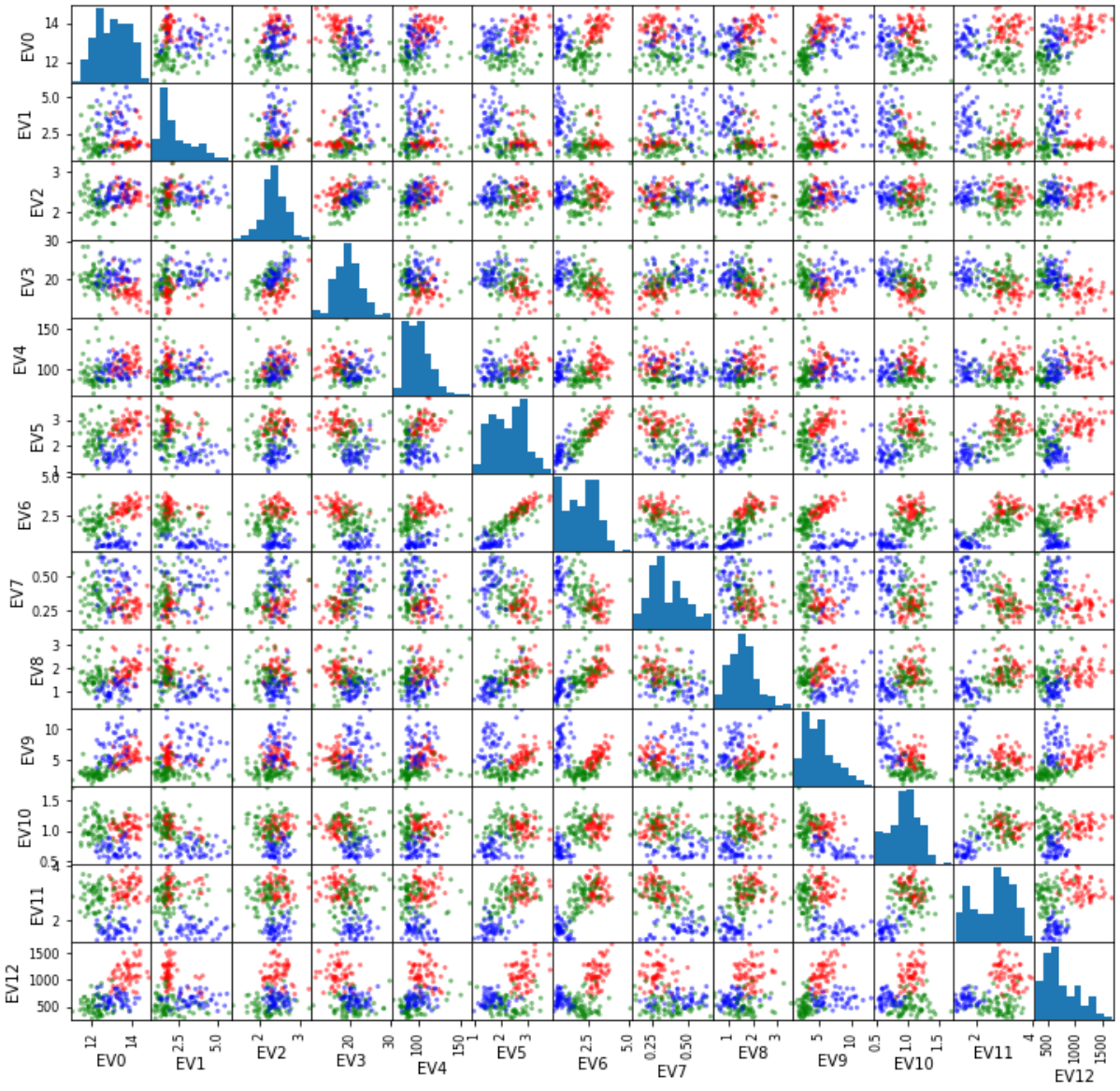
ML-PW-13

by Aurélien Héritier and Jean Nanchen

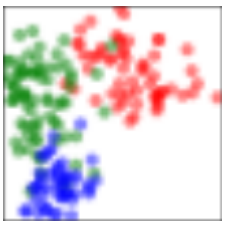
PCA

1. Provide the scatter matrix and select the pair of variables (by visual inspection of the scatter matrix) that appears to allow the recognition of the three classes of wine. Explain.



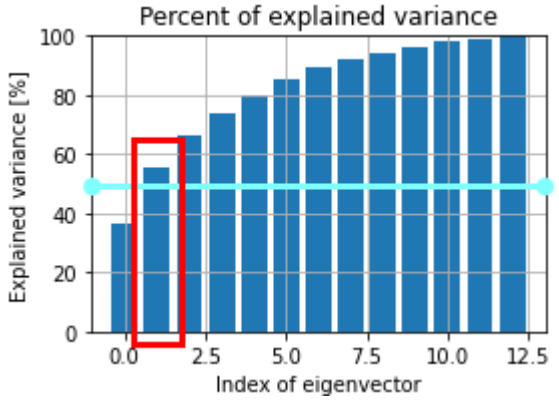
The scatter plot matrix show, for every classes' combination, the 2D equivalent plot for only the 2 classes. It serves to identify the best 2D variables' combination. In the middle diagonal, it's only the distribution of the data int the class.

The best case to allow the better recognition with our visual inspection is the EV12/EV11 plot. We can see below that the 3 classes are well distinct.



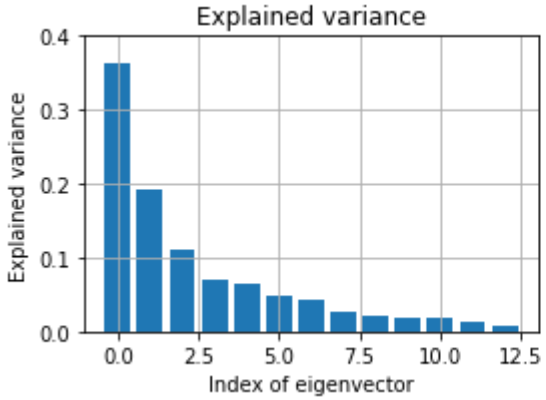
EV12/EV11 scatter plot

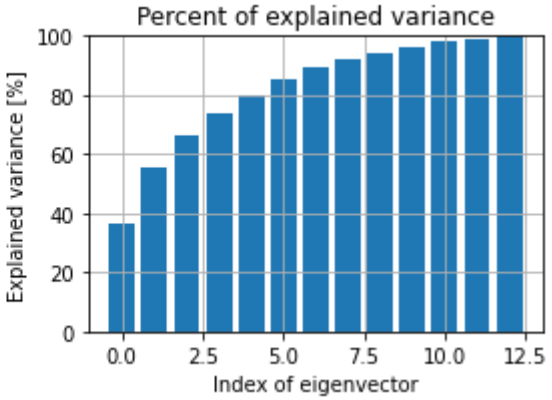
2. Find the smallest set of components capable of explaining at least 50% of the variance of the data.



We need at least 2 eigenvectors to keep a minimum of 50% of data variance. We see this in the plot for the explained variance in function of the number of eigenvectors used.

3. What is the percentage of the variance of the data explained by each one of the first 3 principal components ?





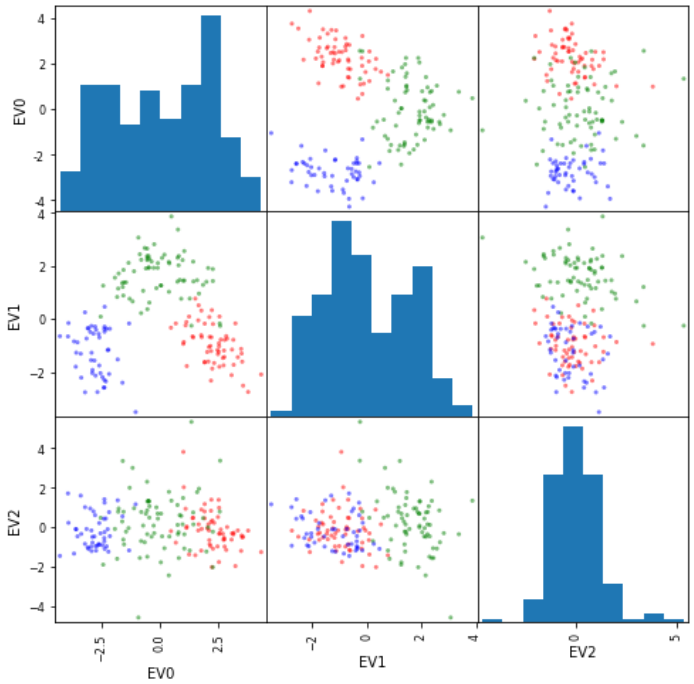
Index	Explained variance in %
0	36.19
1	19.2
2	11.1

We can see in the left plot, the variance explained by every component from the bigger to the smaller. The 3 first components explained ⅔ of the variance.

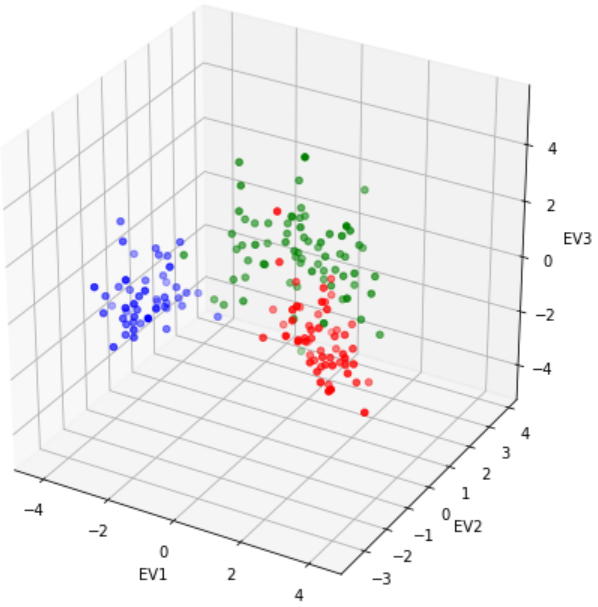
4. Find the smallest set of components capable of explaining at least 60% of the variance of the data. How do you print the resulting eigenvectors ? print them and if only 3 components are required, provide the 3D projection of the original dataset.

For a minimum of 60% of the variance of the data, we need at least 3 vectors (so 3 vectors explain 66.5% of the variance of the data).

We cam print it like a scatter matrix :

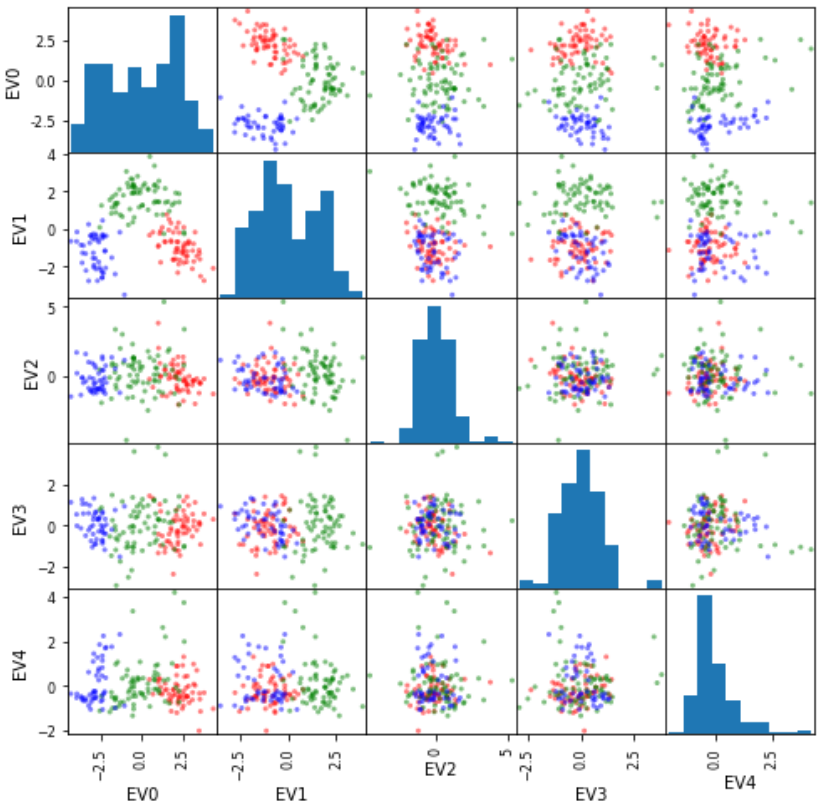


Or we can print it in a 3D space :



5. Find the smallest set of components capable of explaining at least 80% of the variance of the data.

```
The projection will try to keep 80.0 % of the variance
5 eigenvectors are needed
Keeping 80.16229275554788 % of the variance
```

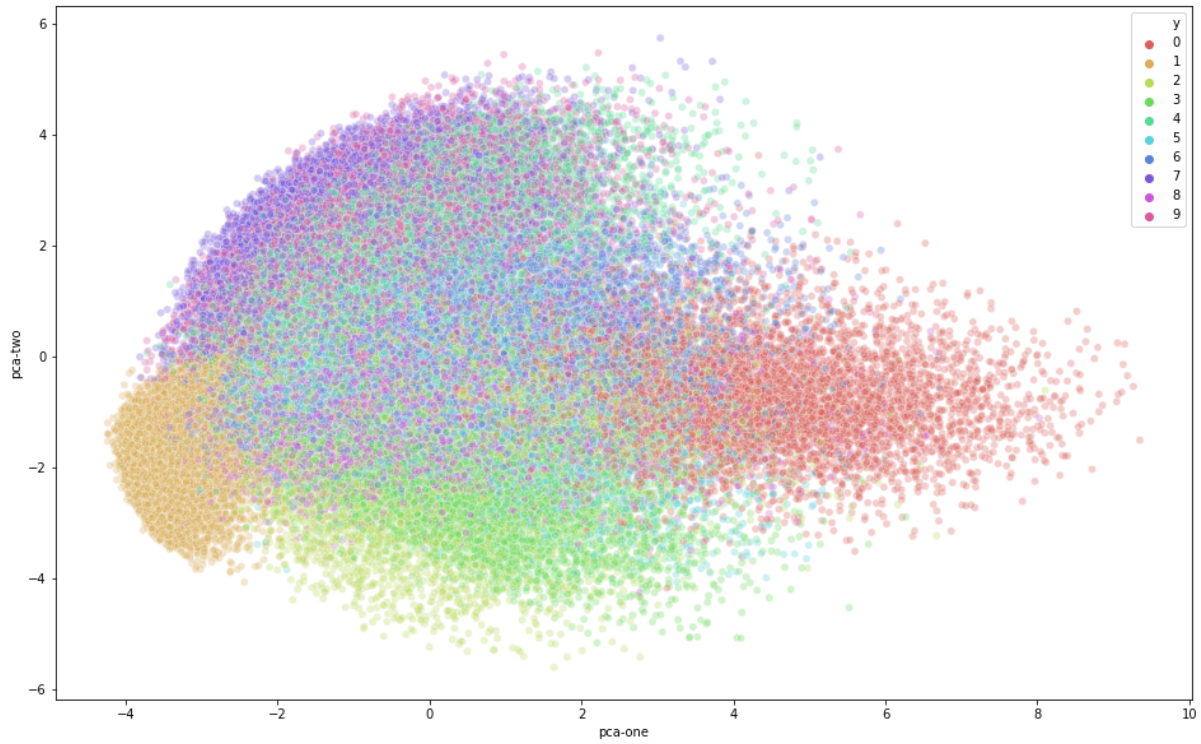


For 80% of the variance of the data, 5 eigenvectors are required.

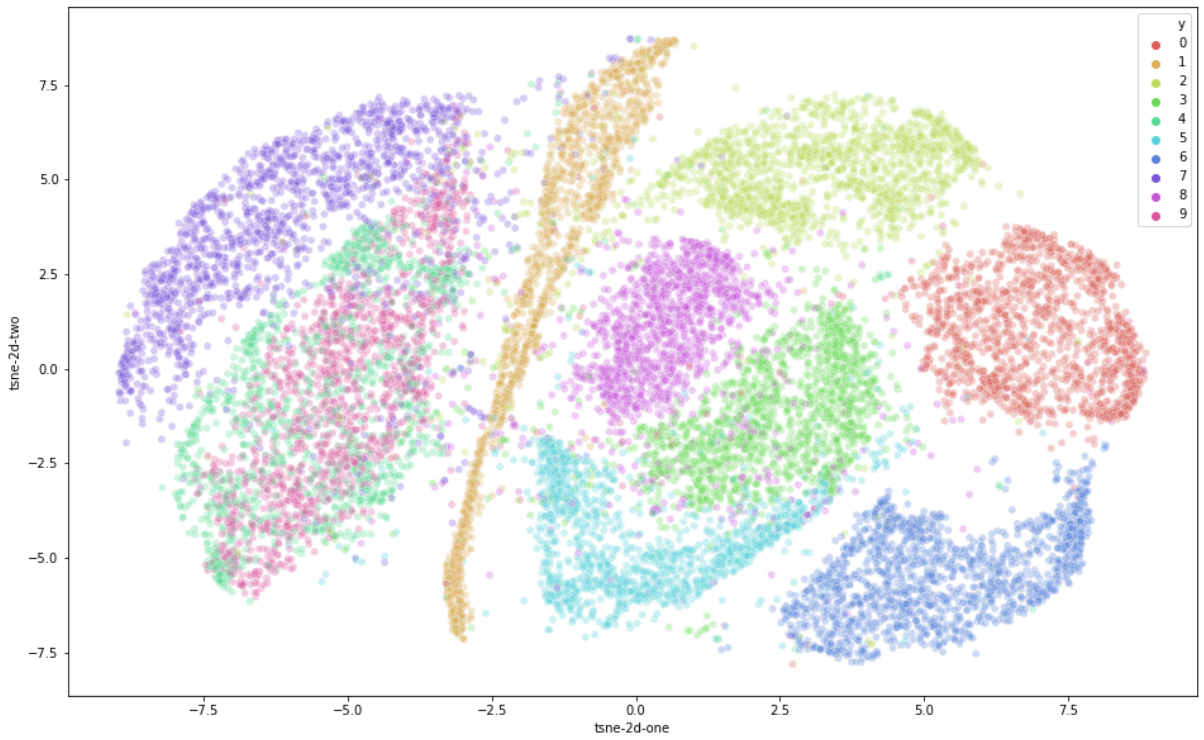
t-SNE

6. Run the notebook and observe the resulting 2D visualization of the dataset. Are there ten classes clearly separated ? provide that visualization.

Clearly no for the PCA !, the same classes are in the same region, but not clearly separated.



Clearly better than PCA for the t-SNE, nearly all the classes are well separated. But we can see that the 4 and the 9 are not well separated :



But there are some few “mistakes”, but the result is by far better.

7. What is the dimensionality of the input data ? what is the final dimensionality at the output of t-SNE ? What is the dimensionality of the input data being fed to t-SNE ?

784 inputs data (each pixel, 28x28) + y (the result in int) and label (the result in string)
The dimensionality at the output of the t-sne is 2. **t-sne-2d-one** and **t-sne-2d-two**.
At the input of the t-sne, there are 784 inputs.

8. Identify the values of the parameters having been used to obtain those results: perplexity, learning rate, momentum, number of iterations.

```
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
```

perplexity : 40
learning rate : default=200.0
momentum : the default method is’barnes_hut for the gradient calculation algorithm, so there is no momentum.
number of iterations : 300

9. What is the formula being used to compute the error given every 10 iterations ?

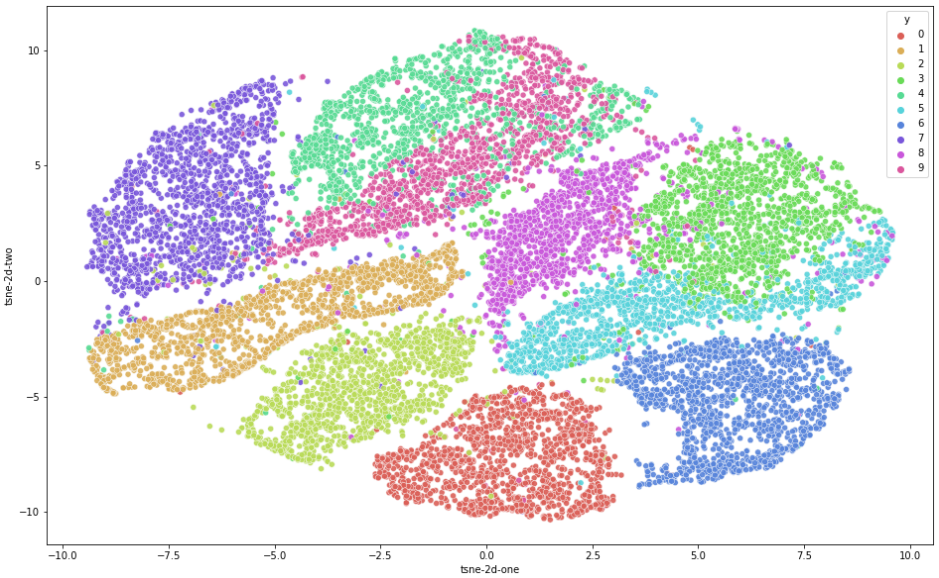
Barnes-Hut approximation

10.What happens if you feed the original dataset to the t-SNE algorithm directly ?

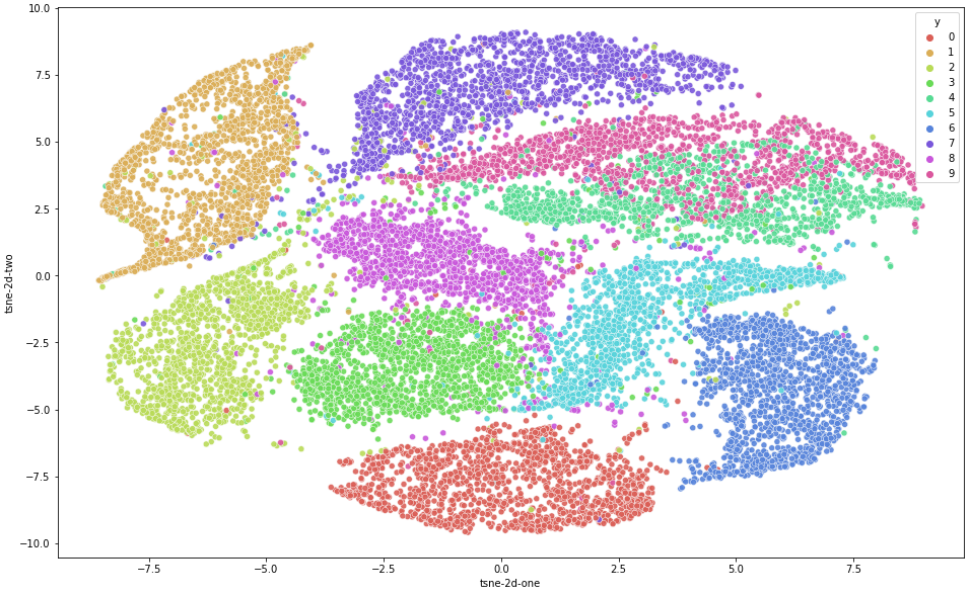
It takes too much time ! (65s for 15000)

It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples.
(https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html)

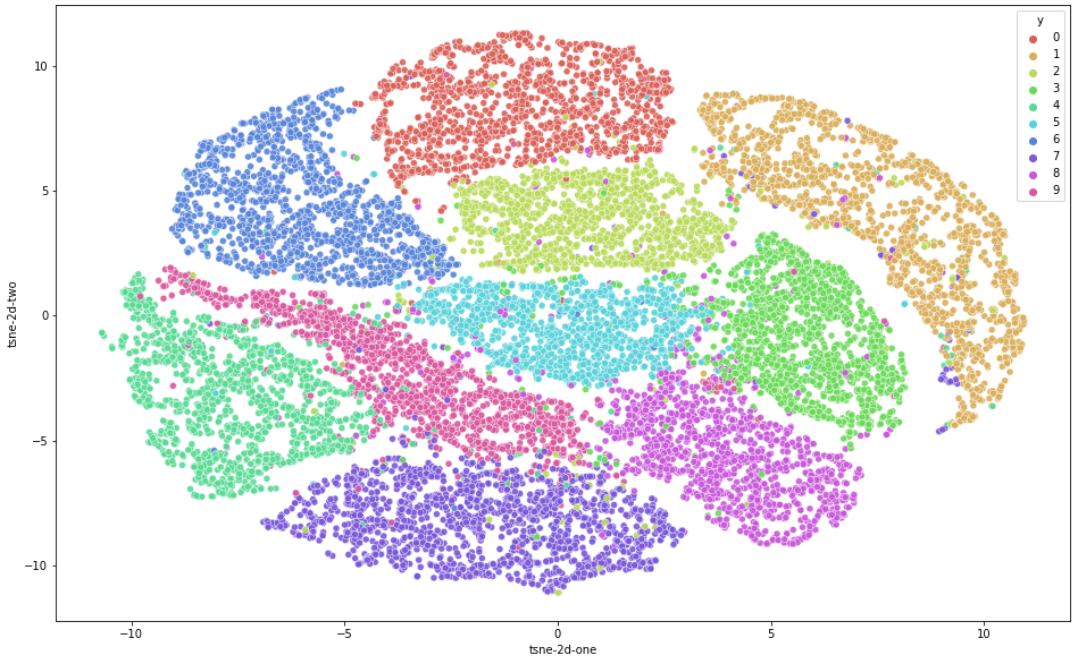
11.Are you happy with the 2D visualization of the ten classes of datapoint ? if not, modify the parameters to get a better one. Provide the resulting visualization accompanied with the selected parameters.



perplexity = 10



perplexity = 20



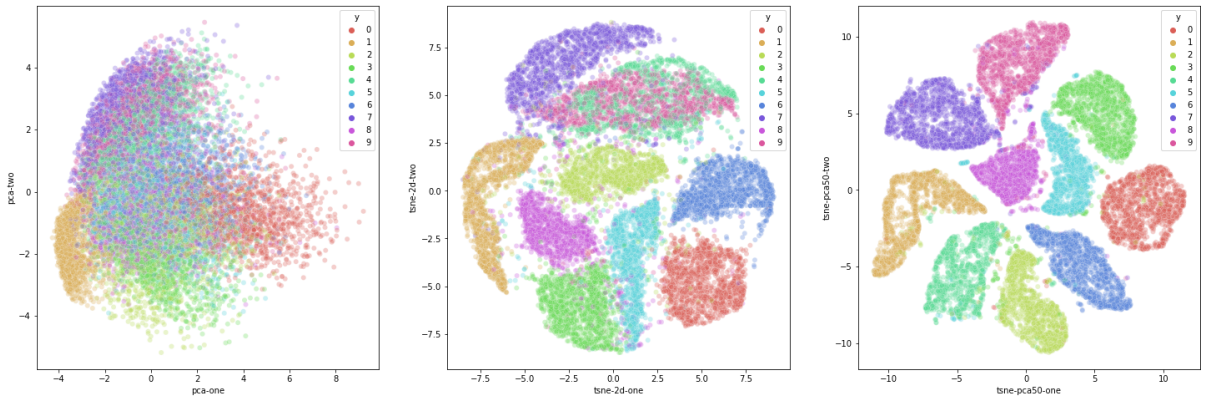
perplexity = 5

We can observe that with a 5 perplexity, the dataset is well split. It's the better perplexity choice, but some noises are still there.

12.Check the computational time required to perform PCA and then t-SNE or t-SNE on the raw data and compare the results.

For 15000 samples :

2.11s for PCA
65s for the t-SNE
t-SNE with PCA : 2.11s for PCA and 50.16s for t-SNE, so 52.27s



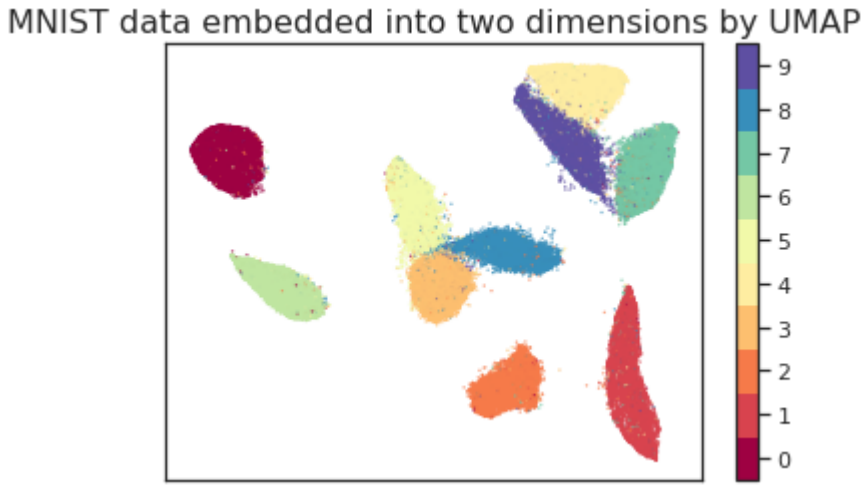
PCA t-SNE PCA + t-SNE

PCA is clearly faster, but not very efficient. t-SNE takes longer but is more accurate. But we can combine the two methods to reduce the input for t-SNE to only 50 inputs. So we save 12s and the result is better than only t-SNE (less noisy).

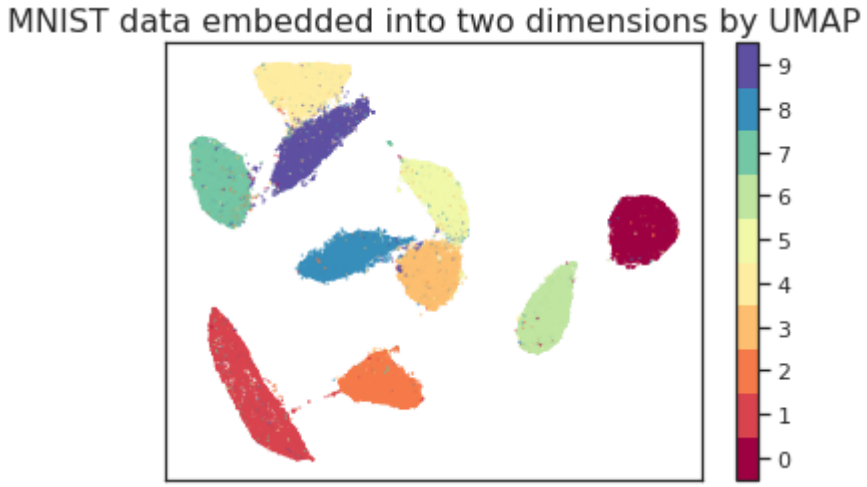
UMAP

13. Try different values for the n_neighbors and min_dist parameters and present 4 results obtained, and comment those results.

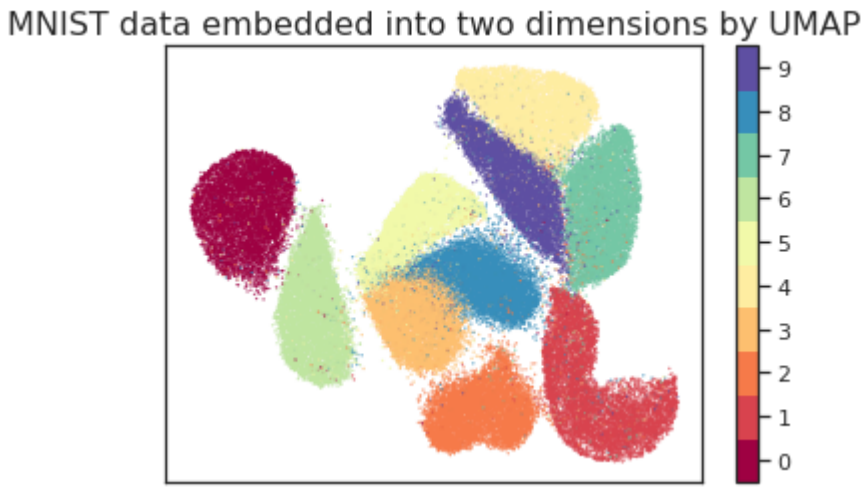
n_neighbors = 50, min_dist = 0.1 :



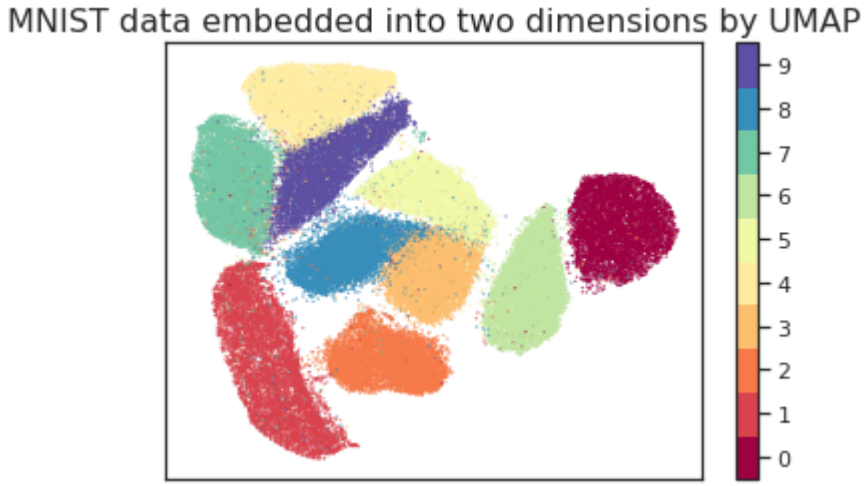
n_neighbors = 10, min_dist = 0.1 :



n_neighbors = 50, min_dist = 0.5 :



n_neighbors = 10, min_dist = 0.5 :



Comments :

When the min_dist is low, the cluster are scattered. When the min_dist is high, clusters are more compact. ⇒ Low values of min_dist will result in clumpier embeddings.

Low value of neighbors, push UMAP to focus on local structure. High value of neighbors pushes UMAP towards representing the big-picture structure while losing fine detail.

14. Compare the resulting projections obtained with UMAP and t-SNE and provide your comments.

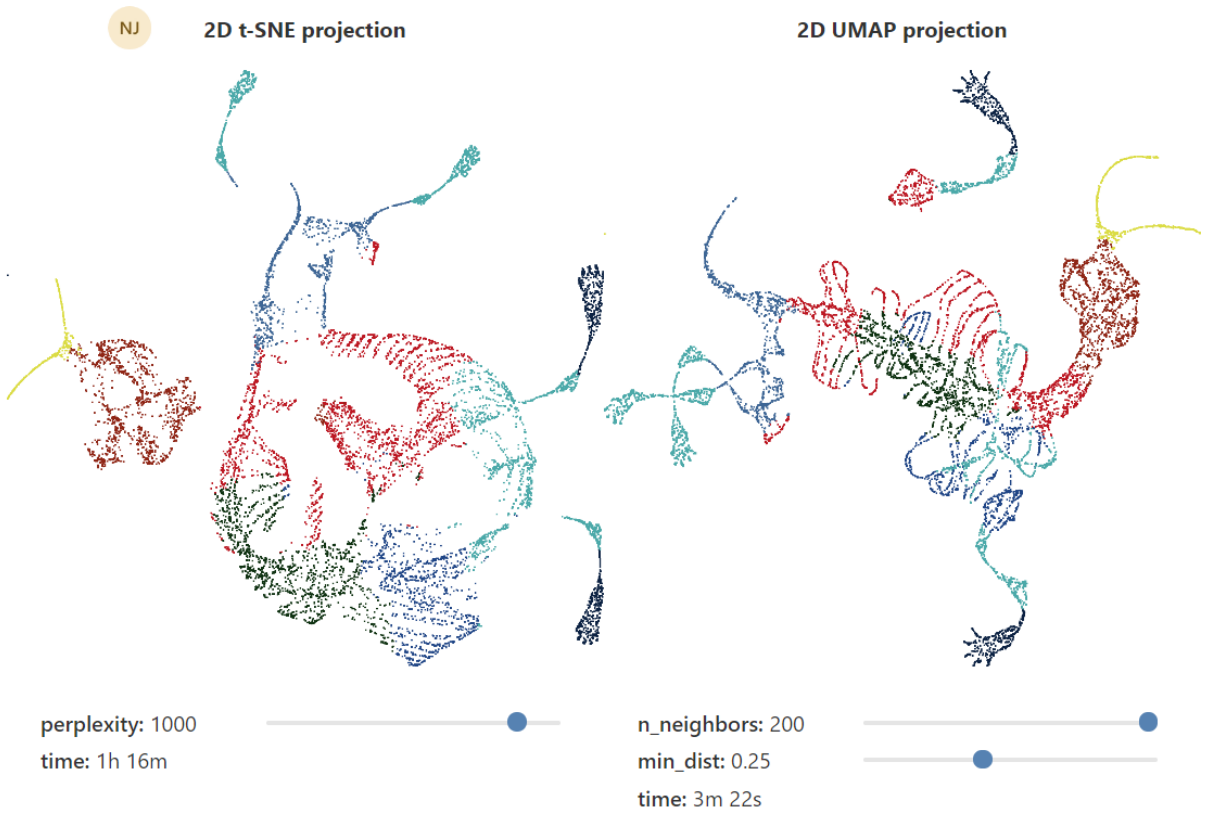


Figure 6: A comparison between UMAP and t-SNE projections of a 3D woolly mammoth skeleton (50,000 points) into 2 dimensions, with various settings for parameters. Notice how much more global structure is preserved with UMAP, particularly with larger values of `n_neighbors`.

image from : https://pair-code.github.io/understanding-umap/

For a result where the cluster are well separated, the time spend for a t-SNE projection is significantly higher than UMAP. (circa x25) But for some datasets, the t-SNE projection give far better results than UMAP.

Clustering of low-dimensional projections of data

15. Apply a clustering algorithm like K-Means on the embeddings obtained by the UMAP algorithm and compute the homogeneity of the resulting clusters. Compare it with the homogeneity of the clusters found by K-Means on the raw data. Provide these results as well as the homogeneity of the clusters found by a different clustering algorithm like HDBSCAN. Comment your results.

Compute Homogeneity, Completeness and V score for HDBSCAN :

```
def cluster_mnist_hdbscan(data):
    hdbscan_labels = hdbscan.HDBSCAN(min_samples=4, min_cluster_size=600).fit_predict(data)
    homogeneity = metrics.homogeneity_score(mnist.target, hdbscan_labels)
    completeness = metrics.completeness_score(mnist.target, hdbscan_labels)
    v_score = metrics.v_measure_score(mnist.target, hdbscan_labels)
    print(f"homogeneity | completness | v_score")
    print(f"{homogeneity:>11.3f} | {completeness:>11.3f} | {v_score:>7.3f}")
```

```
cluster_mnist_hdbscan(embedding)
```

Result for KMeans with raw data :

```
[16] cluster_mnist(mnist.data)

homogeneity | completness | v_score
      0.496 |        0.503 |   0.500
```

Result for KMeans with embeddings :

```
[17] cluster_mnist(embedding)

homogeneity | completness | v_score
      0.827 |        0.845 |   0.836
```

Result for KMeans with embeddings3D :

```
[18] cluster_mnist(embedding_3d)

homogeneity | completness | v_score
      0.836 |        0.855 |   0.846
```

Result for HDBSCAN with raw data :
it takes too much time to compute (more than 2 hours …)

Result for HDBSCAN with embeddings :

```
[24] cluster_mnist_hdbscan(embedding)

homogeneity | completness | v_score
      0.744 |        0.919 |   0.822
```

Result for HDBSCAN with embeddings3D :

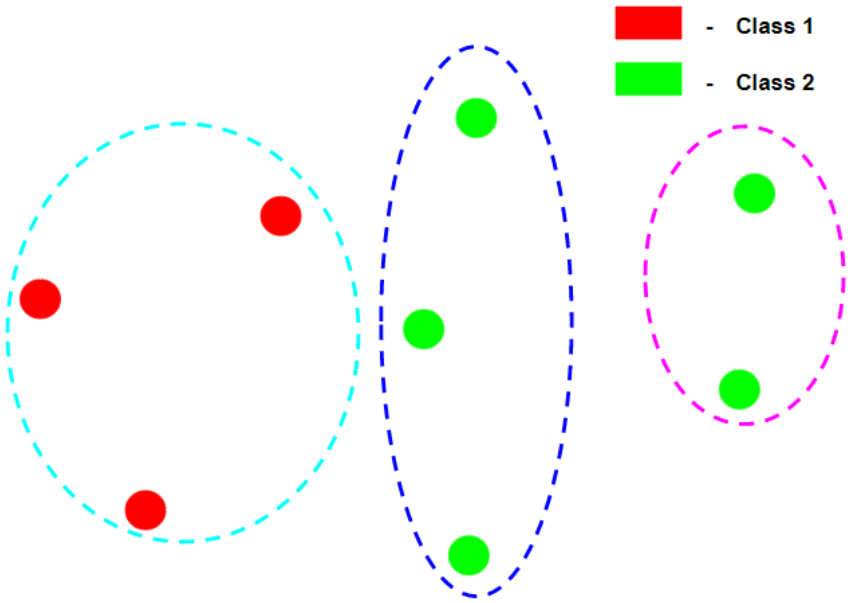
```
[25] cluster_mnist_hdbscan(embedding_3d)

homogeneity | completness | v_score
      0.747 |        0.926 |   0.827
```

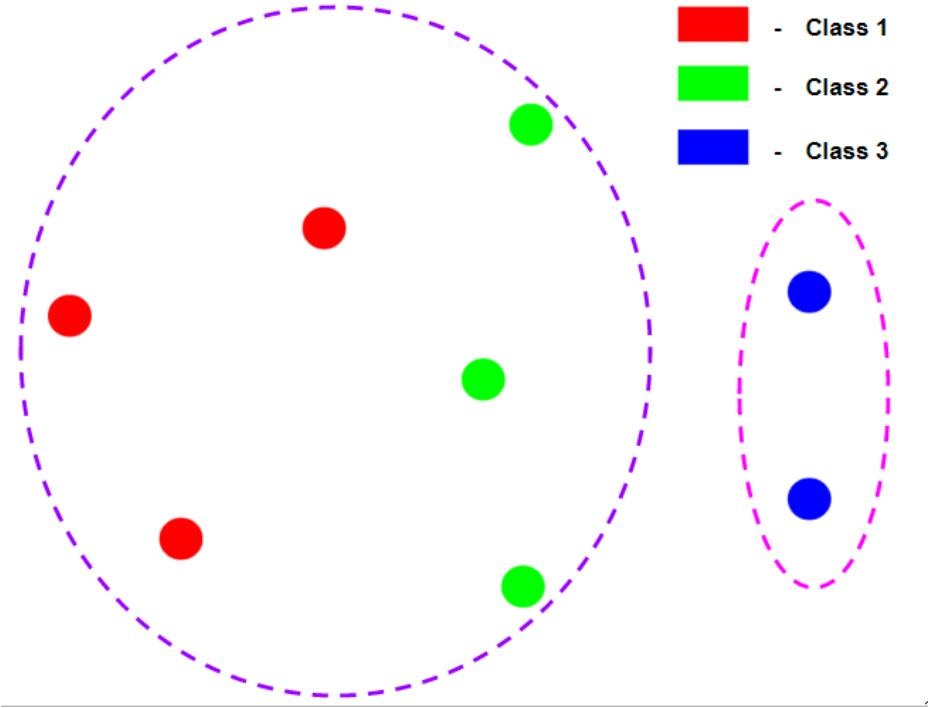
For the two techniques (KMeans and HDBSCAN), using the raw data is not good. The UMAP allows you to prepare the data much better before giving them in the clustering algorithm.

We can achieve a better v_score with the KMeans method. It's important to note that HDBSCAN can refuse data and consider it as noise. So the completeness is better with HDBSCAN, but homogeneity is worse :(

The goal is to have high homogeneity and high completeness, so a high v_score !



In the picture above, we can see that the homogeneity is good, but the completeness is low because there are two clusters with the same class.



In the picture above, we can observe a high completeness (none cluster with the same class) and a low homogeneity, because there are two classes in the same cluster.