University of Paris

Unsupervised learning project

Wacim Belahcel and Imad Oualid Kacimi

28/11/2020

Table des matières

Introduction

This report is the result of a project we worked on for the Master 2 "Machine Learning for Data Science" of the University of Paris, we were asked, during the course "Unsupervised Learning", to study a data set using different unsupervised learning methods seen in class.

The dataset concerns the human activity recognition database that was built from the recordings of 30 study participants performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. Each person performed six activities (WALKING, WALKING*UPSTAIRS, WALKING*DOWNSTAIRS, SITTING, STANDING, LAYING).

Data Exploration:

The dataset is 7352 * 561, Each sample corresponds to a measurement described by 561 variables.

Activity label distribution:

From the Fig1, we can see that although there are fluctuations in the label counts, the labels are equally distributed.

Knowing that participants had to walk up and down the same number of stairs, we are supposed to have the same number of samples for both types of walks, but we can see that the results for the upward walk are slightly more frequent than for the downward walk, which may be due to the fact that the participant walks faster at the bottom.

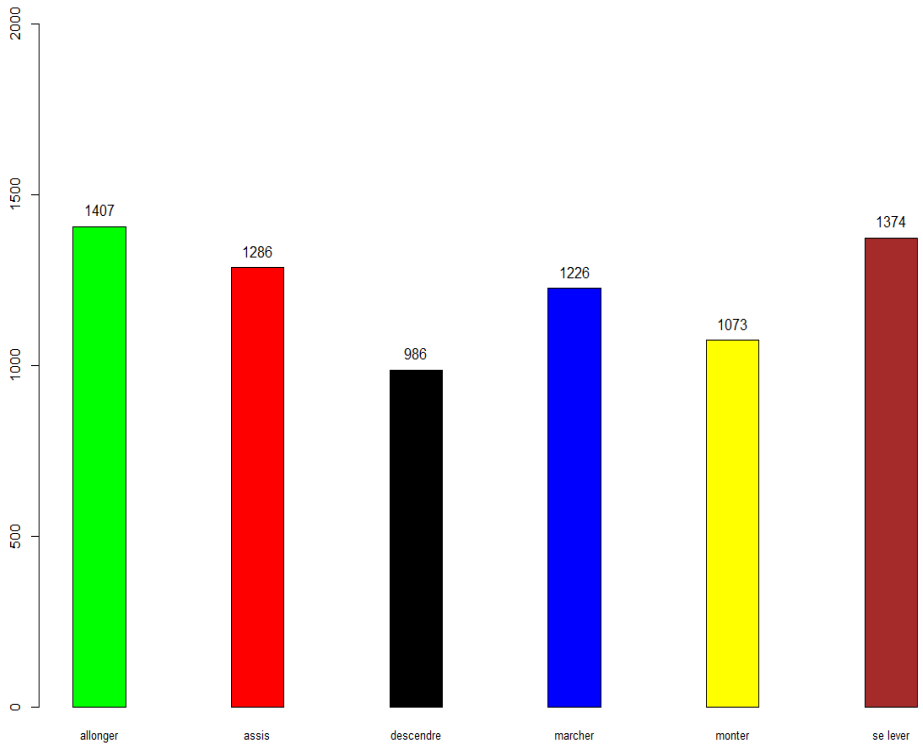

Figure 1: Activity label distribution

Having 561 variables, we tried to use PCA as a dimensionality reduction method to try to reduce the number of dimensions and visualize our data, we obtained the results presented in **Figure 2**.

Figure 2: PCA visualization

However, the PCA reduction seems to not separate each class correctly, this result was expected due to the fact that the two vectors only explain 67% of the total variance from the initial data as shown in the fig 3

```
> summary(pca_result)
Importance of first k=2 (out of 561) components:
                            PC1     PC2
Standard deviation     5.9012 1.65380
Proportion of Variance 0.6255 0.04913
Cumulative Proportion  0.6255 0.67467
```

Figure 3: pca result summary

Knowing that pca is based on the linear relation between variables, we tried visualizing our data using the tsne dimensionality reduction which based on **nonlinear** relation between features.

We obtained the result shown in **fig4,** we can clearly see that the classes are in this case separated correctly, except for the two cases “stand up” and “sitting” this can be due to the fact that sitting or having to stand up does not affect the acceleration of the participant.

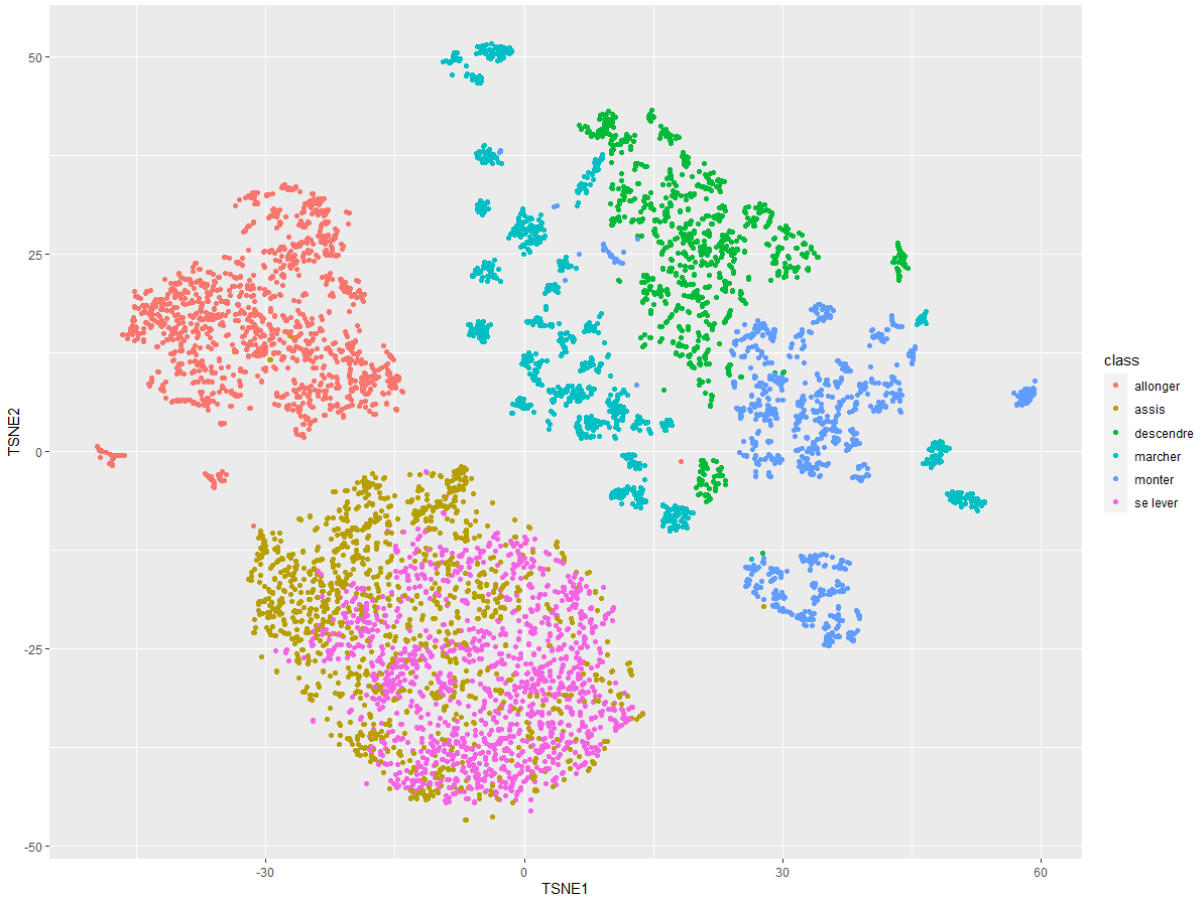

Figure 4: Tsne visualization

Experiments:

As a way to try and improve our algorithms results, we tried different scenario, in this section we will present some of them and their results using kmeans.

Without feature selection

Our first experiment was by using kmean with k=6 without any feature selection using the whole dataset, with such parameters we only got to 51% accuracy, below, we can see the confusion table resulting from this experiment.

	1	2	3	4	5	6
1	595	810	135	1	0	10
2	0	0	0	306	413	163
3	84	49	223	0	0	0
4	0	0	0	930	961	0
5	547	214	628	0	0	0
6	0	0	0	49	0	1234

Figure 5: kmean without feature selection results

We can clearly see that in such configuration the model only seems to differentiate between 2 cluster groups, that is between 1 to 3 and 4 to 5, but he doesn't seem to differentiate between clusters of those groups

With linear correlation-based feature selection

Dependent variables

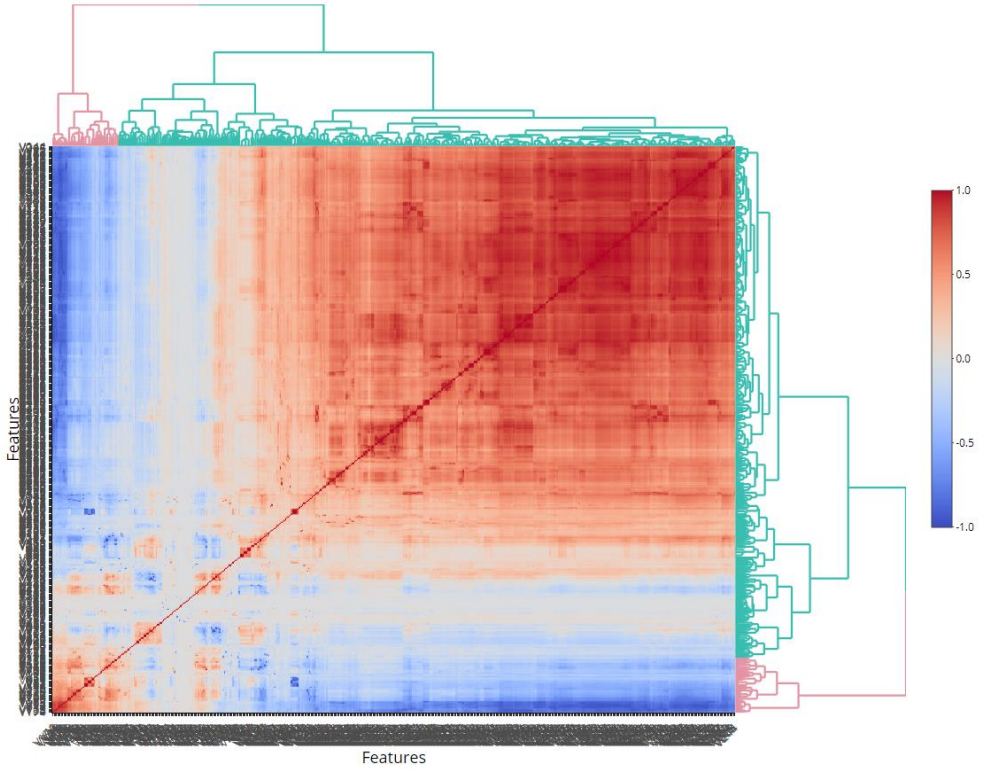

Figure 6 correlation matrix heatmap

From our data exploration, we found that as you can see in the correlation heatmap above, a lot of variable in the dataset are strongly correlated, our second guess was then to try to decrease the dimension of our data to improve our clustering results (curse of dimensionality) by removing some strongly correlated variables.

We found 347 correlated variables (with correlation > 0.9), after removing correlated variable, we kept a total of 214, after training kmean we got an accuracy of 51.8%, which mean we only improved the accuracy to 0.8% from our previous experiment.

Our first conclusion from this experiment was that there probably exist a strong nonlinear correlation between our variables that cannot be captured using feature selection based on a linear correlation matrix or PCA.

PCA

To confirm that the linear correlation between our variable isn't enough to have a decent clustering result, we then tried to apply our algorithms using PCA feature vectors, as mentioned before knowing that PCA failed to separate our clusters correctly, it also gave a poor accuracy of 51.2% with 16 feature vectors which had a cumulative proportion of variance of 84%.

With non-linear correlation-based feature selection

From those results we concluded that linear correlation can't capture the complex link between our variables, we then tried to reduce our feature using nonlinear dimensionality reduction methods such as TSNE.

TSNE

T-distributed stochastic neighborhood embedding is a non-linear dimensionality reduction technique that try to embed points while preserving the neighborhood local structure. it works by trying to convert the high Euclidian distance between two point into a conditional probability that represents the similarity between points, I then try to preserve those calculated probability in the lower space by moving points closer or further from each other's.

As mentioned above, it is a good method for visualization, in our case we used the T-SNE algorithm to reduce our data into a 3-dimensional point.

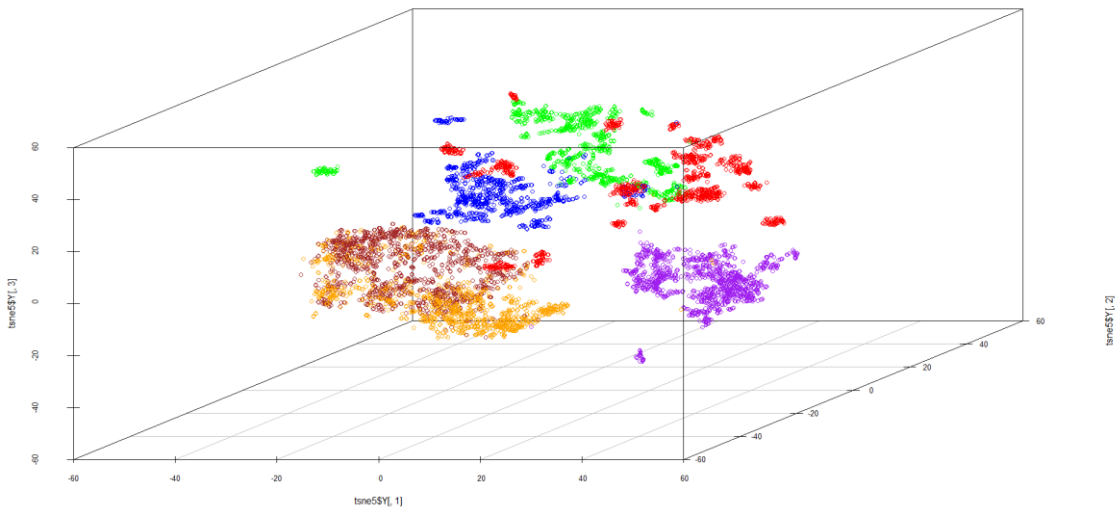

Figure 7 TSNE 3D visualization

We tried different hyperparameter configuration (number of dimensions, perplexity; training step), different configuration gave better results depending on the algorithm.

in the next sections, we will present our results using T-SNE feature extraction, which gave us the best results among all other methods, 8 different configurations were tested:

Configuration name	Training step	Perplexity
TSNE 1	1500	30
TSNE 2	2500	30
TSNE 3	1500	50
TSNE 4	2500	50
TSNE 5	1500	30
TSNE 6	2500	30
TSNE 7	1500	50
TSNE 8	2500	50

Table 1 TSNE tested configurations

Models

CAH

CAH is hierarchical method that is built step by step from the finest level where each individual starts in its own group to the most aggregated level where all individuals are in the same group. In each step the distance between each group is calculated and the 2 closest groups are merged into the same group.

In our analyses we applied CAH by initially getting the distance matrix based on the Euclidean distance, then applying the CAH algorithm with the WARD method.

As mentioned before and as shown in the table below**,** we tried several tsne configurations:

TSNE1	TSNE2	TSNE3	TSNE4	TSNE5	TSNE6	TSNE7	TSNE8
0.709	0.658	0.679	0.716	0.536	0.616	0.682	0.635

Tableau 2: All CAH Accuracy

The best results was obtained by using the TSNE4, and the figure below shows the confusion matrix we go from comparing it to the true labels.

```
> print(comparing.Partitions(vec_pred4,vec_lab4))
[1] 0.7157836
>
> table(vec_pred4,my_data$class)

vec_pred4 allonger assis descendre marcher monter se lever
        1        0   546         0       0      0     1194
        2        1   733         0       0      0      180
        3     1406     6         0       0      0        0
        4        0     0       912     437     30        0
        5        0     1        20     149   1039        0
        6        0     0        54     640      4        0
>
```

Figure 8: CAH Tsne4 confusion table

We can see that the model was quite good with an accuracy of 71%. After examining the confusion matrix, we can see that it was able to predict almost all the “laying” class with only 6 data points classified as “sitting”. In addition, it was hard for him to distinguish between the two classes “sitting” and “stand up” which was expected because we saw early through tsne visualization that the two classes are better suited to one group rather than two separated clusters. It was also difficult for him to separate properly the classes “walk down” “walk”

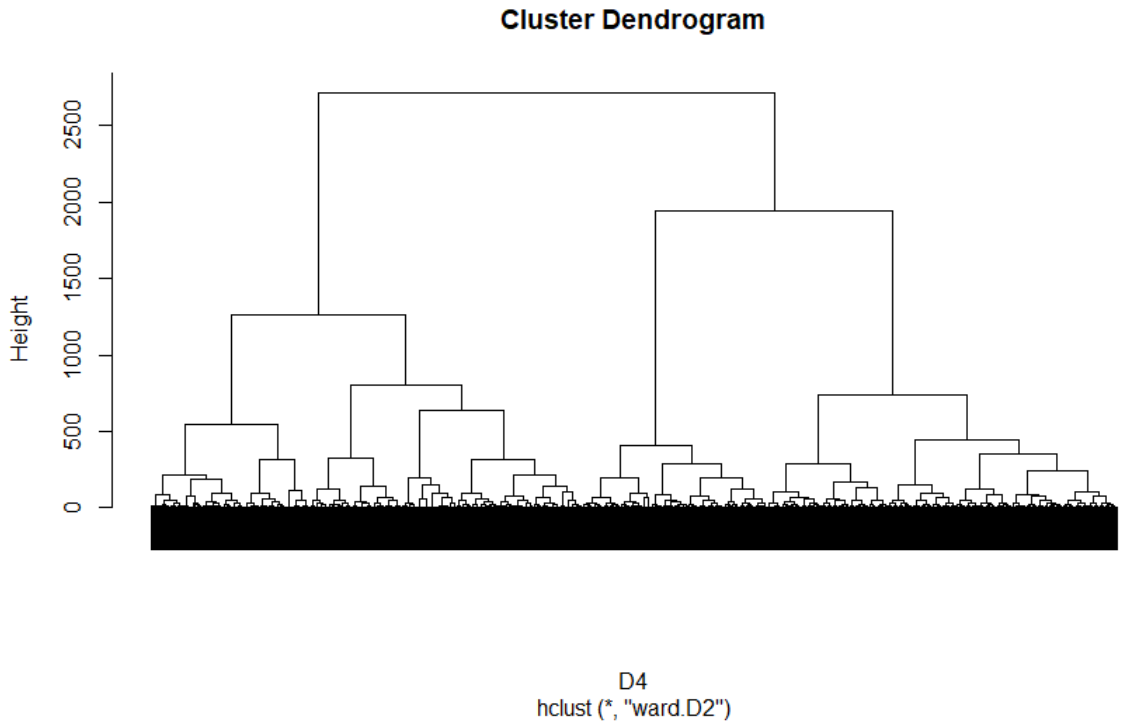

Figure 9 TSNE4 CAH dendogram

K-means

K-means is an unsupervised non-hierarchical clustering algorithm that allows to group *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean. We can achieve this by minimizing the sum of the distances between each individual and the centroid.

In our analyses we applied **K-means** with 50 iterations, with 6 clusters as we know that we have 6 classes and with 50 different starts.

As mentioned before and as shown in the table below**,** we tried several tsne model:

TSNE1	TSNE2	TSNE3	TSNE4	TSNE5	TSNE6	TSNE7	TSNE8
0.712	0.676	0.733	0.765	0.615	0.664	0.728	0.662

Tableau 2: All Kmeans results

The best results were also obtained by using the TSNE4, we can report such results using the confuing matrix shown below:

```
> print(comparing.Partitions(km4$cluster,as.vector(as.matrix(my_data$class))))
[1] 0.7652509
> table(km4$cluster,my_data$class)

    allonger assis descendre marcher monter se lever
  1        0     0       923      50    145        0
  2        0   362         0       0      0      985
  3        0     1        63     361    920        0
  4        0    11         0     815      8       10
  5     1406     6         0       0      0        0
  6        1   906         0       0      0      379
```

Figure 10: Kmean TSNE4 confusion matrix

We can see that **k-means** performed better than CAH with a 76% accuracy. After examining the confusion matrix, we can see that **k-means** was better in predicting the two classes “sitting” and “stand up” and also made less errors while separating the “walk down” and “walk” classes. He also had similar results compared to **CAH** predicting the “laying” class. However, the algorithm seems to have troubles trying to distinguish between the “walk up” and “walk”.

Dbscan

Density-based spatial clustering application with noise is a useful algorithm that have some benefit over k-means and works better in some cases, two parameters are to take into accounts while working with dbscan, that is epsilon (maximum radius of the neighborhood) and minpoints (minimum number of points in the epsilon neighborhood of a point).

The goal is to identify dense regions, which can be measured by the number of objects close to a given point.

Here below we can see results using dbscan with different configurations:

Configuration	Best epsilon	accuracy
TSNE 1	9	60.2%
TSNE 2	9.8	61.90%
TSNE 3	7.1	61.8%
TSNE 4	7.7	61.4%
TSNE 5	11.1	61.5%
TSNE 6	13.2	60%
TSNE 7	9.3	60.7%
TSNE 8	**10.5**	**61.94%**

Table 3 DBSCAN results

As we can see, in our case dbscan gave the worst results among the three methods proposed, we can easily identify why by looking at the graph with true labels:

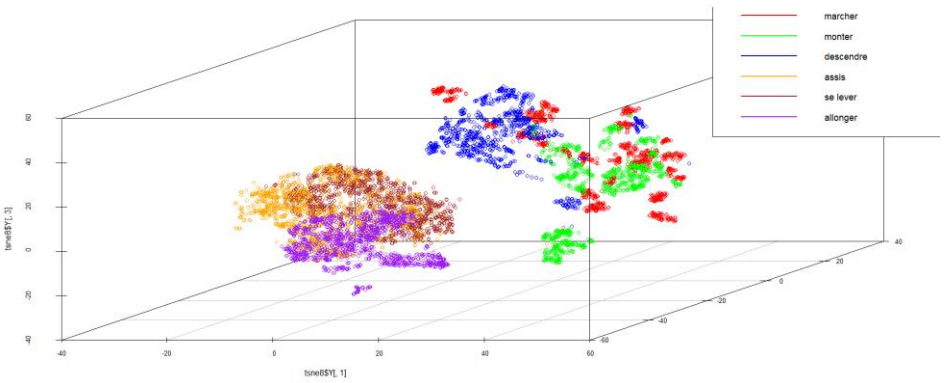

Figure 11 True labels visualization with TSNE8

Before even looking at the Dbscan prediction graph, we can see where dbscan will probably get it wrong, that is “sitting, standing, laying” clusters that are completely mixed due to the fact that those are static movement, which mean there aren’t much difference between them.

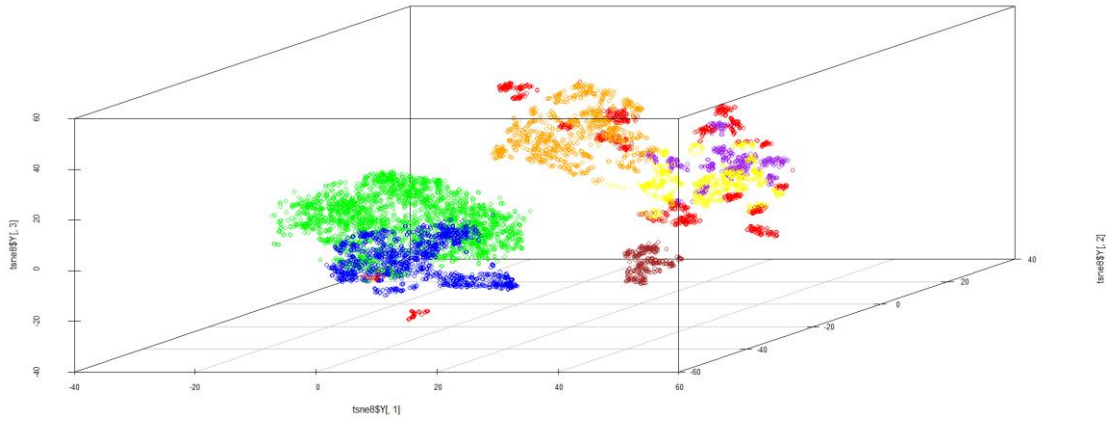

Figure 12 Dbscan clusters with TNSE8

As we can see in the graph above, the cluster “sitting” is completely omitted, this is due to the way dbscan works, the two cluster “sitting” and “standing” are too mixed up, and we can’t differentiate the two clusters which both get mixed up when calculating the neighborhood distances.

This results show the limitation of dbscan with closely mixed clusters, but due to the fact that the there is a varying cluster density between our clusters, we cannot afford to reduce the the epsilon distance (which will make us fail to detect other clusters that requires a bigger epsilon to be correctly detected).

The only possible solution will then be to find a way to better separate our two “difficult” clusters, or to use dbscan on the green cluster alone as a second step.

Results comparison

Model	Configuration	Accuracy
CAH	TSNE4 (2d)	71.6%
Kmeans	**TSNE4 (2d)**	**76.5%**
Dbscan	TSNE8 (3d)	61.94%

Table 4 models best results comparison

From the table above we can clearly compare our results, strength, and weaknesses of each method, first of all, dbscan is clearly inadapted for this dataset for two reasons. first, the clusters “sitting” and “standing” are too mixed up, which makes it impossible for dbscan to differentiate with a high epsilon value, which brings us to the seconds problematic faced by dbscan, the algorithm does not work very well for sparse datasets or datasets with varying density which is our case.

In the case of kmeans and CAH, they both have pretty good results, especially for a 6 class clustering problem, they both seem to have a harder time differentiating with static movement class, this is due to the “crowding” problem of clustering, once we reduce our data, some class could get mixed up and be harder to differentiate, and even though TSNE try its best to reduce this issue, this can still happen.

On the other hand, we can’t keep the data as is without reducing it, which will makes us fall in the curse of dimensionality and makes the problematic impossible to solve for algorithm like Kmeans and CAH, and since in this case there is a strong non-linear dependence, we cannot afford using a linear dimensionality reduction method, we could maybe improve those results by combining different

dimensionality reduction methods though.

Further experiments could bring better results by trying or combining other dimensionality reduction and features selections methods.

Conclusion

During this project, we had the opportunity to experiment different clustering algorithm on a dataset with varying cluster variance and overlap, which made the problematic interesting and made us able to think about clever ways to improve our results.

From this project we were able to learn what are the strength and weaknesses of each methodology and compare them, some being more efficient in specific configurations. Moreover, we learned to implement many different dimensionality reduction methods and compare their capabilities as a visualization and feature extraction tool.