Université catholique de Louvain

LINMA2472 : Algorithms in data science

Homework 2 - part I : Graph Kernel

Students : Vincent Cammarano - 5391 21 00
Louis Parys - 7256 17 00
Mattias Van Eetvelt - 1660 18 00

Group : 6

Professor : Jean-Charles Delvenne
Gautier Krings
Estelle Massart
Rémi Delogne
Bastien Massion
Brieuc Pinon

November 2022

1 Introduction

For the first part of this second project, we were ask to familiarise ourself with the Weisfeiler-Lehman subtree kernel of three different labeled graph origined from chemistry/bio-chemistry. Then, since the data suffer from high-dimension we will visualize them with non linear dimensionality reduction technique such as the kernel-PCA and tSNE. In the end, we will also show how the kernel computed above is doing when fitted to a support vector machine classifier.

2 Computing the kernels

2.1 Weisfeiler-Lehman subtree complexity

An asymptotic computational complexity is the usage of asymptotic analysis for the estimation of computational complexity of algorithms. Algorithmic complexities are classified according to the type of function appearing in the big O notation. For example, an algorithm with time complexity O(n) is a linear time algotithm.

As written in the "Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeilerlehman graph kernels. Journal of Machine Learning Research, 12(9)" research paper at page 2544 [1] : "The runtime complexity of the 1-dimensional Weisfeiler-Lehman algorithm with h iterations is O(hm)."

2.2 Compute the kernels

We first initialized a Weisfeiler-Lehman subtree kernel with the WeisfeilerLehman function from the grakel library with a arbitrary number of iteration (parameter H) of 10. Then we used the fit_transform() method on the graph kernel. That method is used to scale the data and learn the scaling parameters of those.

2.3 Explicit features

The graph embedding methods are formally categorized as implicit graph embedding or explicit graph embedding.

The implicit graph embedding methods are based on graph kernels. A graph kernel is a function that can be thought of as a dot product in some implicitly existing vector space. Instead of mapping graphs from graph space to vector space and then computing their dot product, the value of the kernel function is evaluated in graph space. Such an implicit embedding satisfies all properties of a dot product. Since it does not explicitly map a graph to a point in vector space, a strict limitation of implicit graph embedding is that it does not permit all operations that could be defined on vector spaces.

On the other hand, explicit graph embedding methods explicitly embed an input graph into a feature vector and thus enable the use of all the methodologies and techniques devised for vector spaces. The vectors obtained by an explicit graph embedding method can also be employed in a standard dot product for defining an implicit graph embedding function between two graphs.

2.4 Explicit embedding versus kernel

The ranks of the Weisfeiler-Lehman subtree kernel matrix for the three datasets with H = 10 iterations are respectively 175, 595 and 4002 for the MUTAG, ENZYMES and NCI1 datasets. Unfortunately and despite our researches, we couldn't find any link between the dimension of the implicit embedding space and the rank of the matrix.

1. https://www.jmlr.org/papers/volume12/shervashidze11a/shervashidze11a.pdf

3 Visualization

3.1 Kernel centralization

Before applying Kernel-PCA on the three different kernel, we need them to be centered. To do that in the embedding space we need to use this formula :

$$\tilde{K} = K - 1^N @ K - K @ 1^N + 1^N @ K @ 1^N \tag{1}$$

The 1^N represent a NxN matrix with all values set 1/N, and the @ represent a dot product.
We implemented this formula into a python function in order to center the pairwise kernel of the three dataset computed before, now the data is ready for the Kernel-PCA.

3.2 kernel-PCA implementation

Like it was presented in the slides, the **kernel PCA** is a solution to the **PCA** being worthless for non-linearity problem. It is all about computing the eingenvalues and eigenvectors of the centered kernel computed above. We can do it in Python with one line of code thanks to *scipy.linalg.eig* which is a function from the scipy package that return the eigenvalues, eigenvectors.

3.3 kernel-PCA visualization

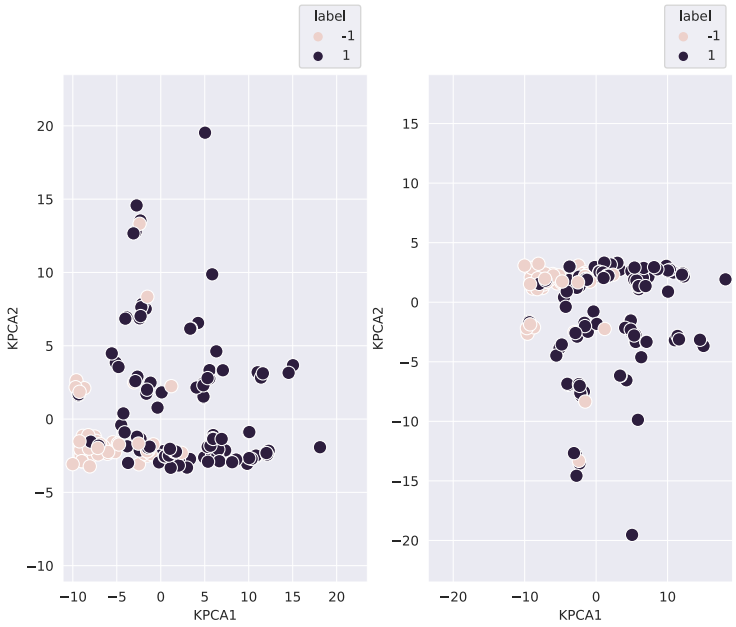


FIGURE 1 – Comparison between the implementation of kernel-PCA of sklearn and our implementation on the centered Mutag

As we can notice with the comparison above, our implementation of the kernel-PCA doesn't match exactly the one of sklearn. But after all, the goal of this algorithm is to get a visualization of the data in a lower dimension, and even if the sign of our second component is not right we get the same idea of the data as if we look at the algorithm of sklearn. If we want our implementation to match the one of sklearn, we should use their function *svd_flip*[2] on the eigenvalues/eigenvectors to obtain the same result.

2. https://github.com/scikit-learn/scikit-learn/blob/7ec1bfc2ab7425bcd984d631b5bfeb9082ce11bf/sklearn/utils/extmath.py#L760

And if we focus our attention on the graph, we can see that there is one big agglomeration of the representation of the *-1* label and some farthest points, while for the *1* label there is much more noise in the data.

3.4 Distance

The sklearn.metrics.pairwise.pairwise_distances method (using "euclidean" as metric parameter) and the homemade function using the function $\sqrt{k(G_1, G_1) + k(G_2, G_2) - 2k(G_1, G_2)}$ proposed in the guidelines are giving the same result.

The euclidean distance in a n-dimensional space between G_1 and G_2 can be written as followed : $d(G_1, G_2) = \sqrt{\sum_{i=1}^{n} k(G_{1i} - G_{2i})^2}$ if n=1 and we develop the square of the parenthesis, we fall back on the function given in the guidelines.

3.5 tSNE

t-distributed stochastic neighbor embedding (t-SNE) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. It is based on Stochastic Neighbor Embedding originally developed by Sam Roweis and Geoffrey Hinton, where Laurens van der Maaten proposed the t-distributed variant.It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. [3]

As we can see in the citation above, tSNE is a method that has the same purpose as the Kernel-PCA. It is designed to lower the dimension of data and being able to visualize it in two or three dimensions, this algorithm frequently finds clusters even if it is non-clustered data. So we need to be cautious on how to interpret the results.

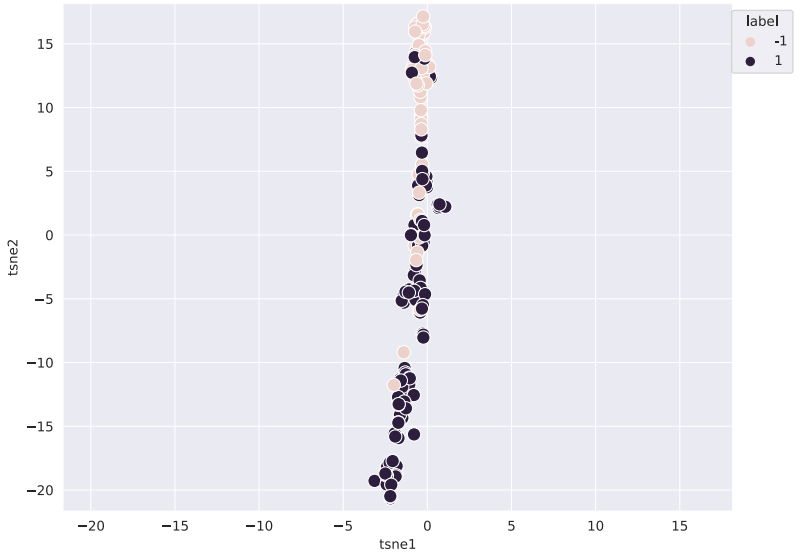


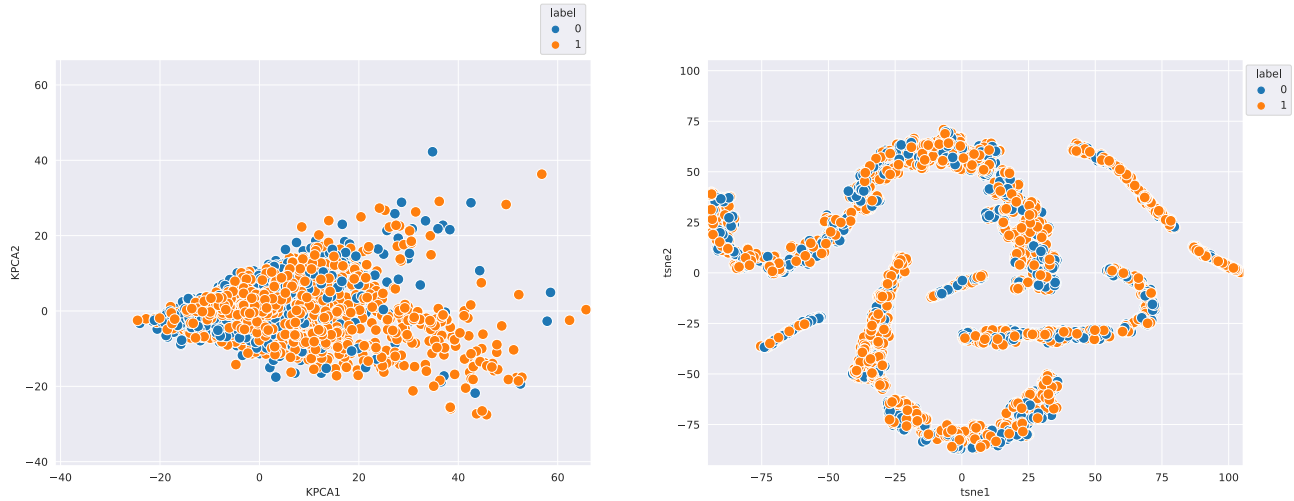
FIGURE 2 – Result of the tSNE implementation of sklearn on the Mutag pairwise distances

3. https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

On the graph above we can see better the clustering of the data, even though, we still see the noise in this representation.

3.6 Comparison

Here is a direct comparison between the kernel-PCA algorithm and the tSNE :



(a) k-PCA on the centered kernel matrix of NCI1

(b) tSNE on the pairwise distances of NCI1

FIGURE 3

If we take a look at both graphs, the first observation we can make is that we can directly see we get more informations on the visualization of the data by looking at the tSNE algorithm. Indeed, with the tSNE graph we can notice some clusters while on the k-PCA graph features dispersed points. The second thing we need to keep in mind is that the tSNE algorithm showed here is applied on the pairwise distances and not on the centered kernel as k-PCA. So, what we actually see on the graph is the relation between the distances of the projected label and not directly on the centered data. And, even if the data look clustered on the tSNE algorithm, this is not the case since the label are mixed.

4 Classification

4.1 A simple baseline

A constant classifier (or model) always output the same target variable by defining it as a strategy, we use this model to get an idea on how the data can be classified in the easiest possible way. For this purpose, we used the sklearn module DummyClassifier. It is a model that can take a strategy parameter with the keyword "constant" we can then include the parameter constant in the model, with our constant strategy. In order to find the best strategy, we loop on the different possible target variable and put them as the strategy, the one with the best accuracy of classification score is the best constant model. Here is the result on the three dataset :

	best constant model	best strategy
Mutag	0.76	1
Enzimes	0.21	3
NCI1	0.50	1

As we can see on the table above, the mutag best constant model score is high because there are only 2 variables and the variable 1 is the most represented. The dataset Enzimes has a low score, because there

are 6 variables and the highest represented one is 3. Lastly, we can say that the NCI1 is almost uniformly distributed since is best score tend to 1/2 and there is only two variables.

4.2 Support Vector Machines (SVM)

As advised in the project description, we used the sklearn implementation of SVM. But there are multiples implementations of support vector machine on sklearn. However, only one is fit to the kernel we have, it is the *SVC*, because the other one are for linear kernels or regressions and the *NuSCV* doesn't have the regularization parameter (c).

Now that we choose the model, we can initialise the classifier with the parameters *kernel='precomputed'* and *C=1e2*. We set the parameters kernel as *precomputed* because we don't ask sklearn to compute the kernel for us since we are going to compute it with the Weisfeiler-Lehman framework implemented in grakel. So, we initialise our Weisfeiler-Lehman framework with grakel and set its parameters as *n_iter = 10* and *base_graph_kernel =VertexHistogram* which is a default graph base kernel. Then, we split the graph to *80%* training set and *20%* test set. Now, we need to transform them into pairwise kernel, we *fit_transform* the training set with our Weisfeiler-Lehman framework and after that, we transform the test set. Now the data is ready to be fitted to our *SVC*, so we fit it with the training set, and then we asses the accuracy by predicting the test set and compare it to the true test. Here is the result of repeating the above operation on the three datasets :

	SVM model accuracy score
Mutag	92.11%
Enzimes	55.0%
NCI1	82.0%

We can see that the results were good for the classification on the Mutag and NCI1 data, and that we got bad results for the Enzimes dataset. We think this is because there are more target variables. There are six on the Enzimes, while it is binary on the two others.

4.3 Select hyperparameters

The goal of this section, is to find the best regularization parameter C of the *SVC* and the best H which is the *n_iter* parameter of the Weisfeiler-Lehman framework. In order to obtain a meaningfull result, we search them on a 10 fold cross validation, we used the sklearn *cross_val_score* (this is the implementation of cross validation from sklearn) function with *cv=10*. We make a double loop on the list of parameters we want to tune so that we can find the best model parameters. For this purpose, we repeat the operation to score the SVM but **only** on the training test previously made. Therefor, instead of calculating a score, we obtain a list of scores from *cross_val_score*. We take the mean of this list score and check if it is better or not than the best precision found so far. In the end, we obtain these values for the hyper parameters :

	best score	best C	best H
Mutag	87.3%	1e3	1
Enzimes	50.2%	1e-1	3
NCI1	85.7%	1e-1	7

As previously said, this *cross_validation* was made on the training set so these are the mean results got from a little part of the training set. We have displayed below the results got from the validation set with the best parameters found :

	validation score
Mutag	94.74%
Enzimes	55.0%
NCI1	85.28%

What can be seen with this table of results is that the scores are higher than in the section 4.2. That means our hyperparameters tuning did well and we have a stronger model since we did a *cross_validation* which means the model is robust toward overfitting.

5 Observations

In conclusion, with what we observed from the visualization part and with the different accuracy scores we obtained, we can deduce a few thing. Firstly, dataset with binary target variable tends to have better accuracy which seems logical since we are using support vector machine as classifier and it is looking for the best margin classifier which is easier to find with two variable. Secondly, for the Mutag data we can see a bit of a cluster of the labels which can explain the very good accuracy we have for it. Finally, we can explain the low accuracy on the Enzimes dataset by the fact that it has six variables, and when we visualize the data with both algorithms we see that everything is mixed up.