Manifold Learning and Graph Kernels

Third Assignment of the course in Artficial Intelligence held by Prof. Torsello

Bernardi Riccardo - 864018

Index:

1. Problem Statement

Read this article presenting a way to improve the disciminative power of graph kernels.

Choose one graph kernel among

- Shortest-path Kernel
- Graphlet Kernel
- Random Walk Kernel
- Weisfeiler-Lehman Kernel

Choose one manifold learning technique among

- Isomap
- Diffusion Maps
- Laplacian Eigenmaps
- Local Linear Embedding

Compare the performance of an SVM trained on the given kernel, with or without the manifold learning step, on the following datasets:

- PPI
- Shock

Note: the datasets are contained in Matlab files. The variable G contains a vector of cells, one per graph. The entry am of each cell is the adjacency matrix of the graph. The variable labels, contains the class-labels of each graph.

NEW I have added zip files with csv versions of the adjacecy matrices of the graphs and of the lavels. the files graphxxx.csv contain the adjaccency matrices, one per file, while the file labels.csv contais all the labels

- PPI
- Shock

2. Introduction

We are going to explain in this paper the experiments we run over the two datasets provided, they are called PPI and SHOCK. The PPI dataset deals with the Protein Protein Interaction, It consists of 86 graphs that repesent proteins and between them we would like to discover interesting similarities. In the former dataset ew would like to classify between 2 classes "Acidovorax" and "Acidobacteria". The second dataset contains 150 graphs and we would like as before to find a way to efficiently compute similarities between them. In the chapter 3 we are going to introduce the kernels and the graph kernels, also we are going to propose a library that provides them. In this second dataset there are 10 classes used to classify the graphs. In the chapter 4 we are going to look inside the possible manifold learning techniques to reduce the visited space of our algorithm and also to visualize our result in 2Dimensions. The 5th chapter talks about the Comparisons that we made and we will discuss about possible improvements. In the last chapter that is the 6th we will draw the conclusions. The other chapters at the very end of this paper are the bibliography and the appendix.

Disclaimer:

This assignment was done only by Riccardo Bernardi(864018@stud.unive.it) , both the code, the report and the experiments. Material taken from internet or books is cited in the bibliography.

During this assignment was also created the Dominant Set Graph Kernel, this was both invented and implemented by Riccardo Bernardi(864018@stud.unive.it).

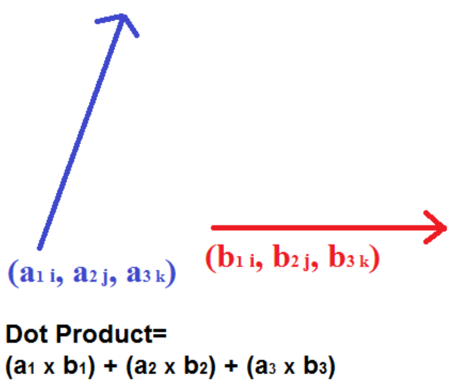
3. The Graph Kernel

We are going here to answer these questions:

- what is a kernel and how to create one ?
- what is a graph kernel ?
- which kernels are available and where ?

3.1 What is a kernel

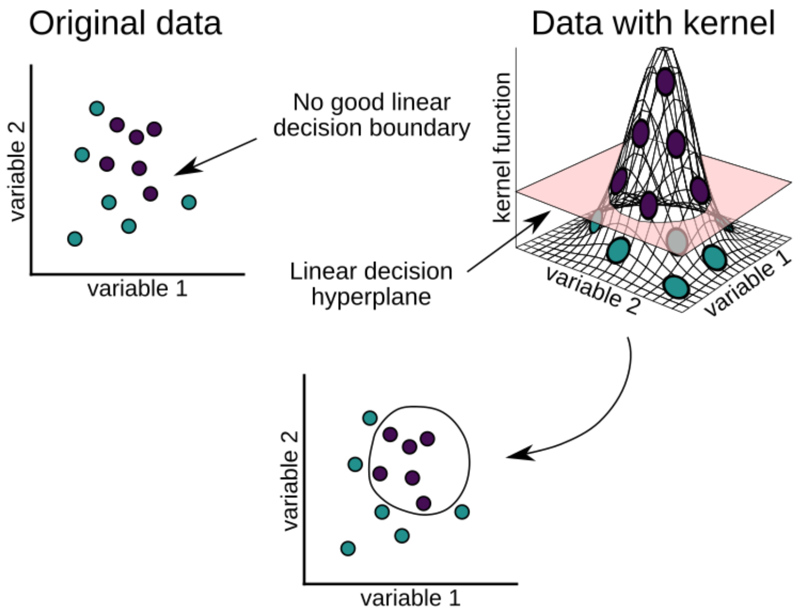
Kernel is a way of computing the dot product of two vectors $\mathbf{x}$ and $\mathbf{y}$ in some (possibly very high dimensional) feature space, which is why kernel functions are sometimes called "generalized dot product". Suppose we have a mapping $\varphi:\mathbb{R}n\to\mathbb{R}m$ that brings our vectors in $\mathbb{R}^n$ to some feature space $\mathbb{R}^m$. Then the dot product of $\mathbf{x}$ and $\mathbf{y}$ in this space is $\varphi(x)^T\varphi(y)$. A kernel is a function k that corresponds to this dot product, i.e. $k(x,y)=\varphi(x)^T\varphi(y)$. So Kernels give a way to compute dot products in some feature space without even knowing what this space is and what is φ. here the dot product visually :



For example, consider a simple polynomial kernel $k(x,y)=(1+x^Ty)^2$ with $x,y\in R^2$. This doesn't seem to correspond to any mapping function φ, it's just a function that returns a real number. Assuming that x=(x1,x2) and y=(y1,y2), let's expand this expression:

$$k(x,y) = (1 + x^T y)^2 = (1 + x_1 y_1 + x_2 y_2)^2 == 1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2$$

Note that this is nothing else but a dot product between two vectors $(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$ and $(1, y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2)$. So the kernel $k(x,y) = (1 + x^T y)^2 = \varphi(x)^T \varphi(y)$ computes a dot product in 6-dimensional space without explicitly visiting this space. In the image below we can see the idea of the kernel, it finds a non linear dividing hyperplane on a higher-space:



Another example is Gaussian kernel $k(x,y) = exp(-\gamma \|x - y\|^2)$. If we Taylor-expand this function, we'll see that it corresponds to an infinite-dimensional codomain of ϕ.

This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

The kernel trick avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary.

The key restriction is that the dot product must be a proper inner product. An inner field holds when it has:

- conjugate symmetry
- linearity
- Positive definiteness

On the other hand, an explicit representation for ϕ is not necessary, as long as V is an inner product space. The alternative follows from Mercer's theorem: an implicitly defined function ϕ exists whenever the space X can be equipped with a suitable measure ensuring the function k satisfies Mercer's condition.

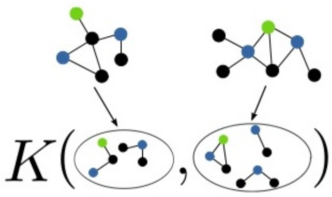
Mercer's theorem: K is said to be non-negative definite (or positive semidefinite) if and only if:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i, x_j) c_i c_j \geq 0$$

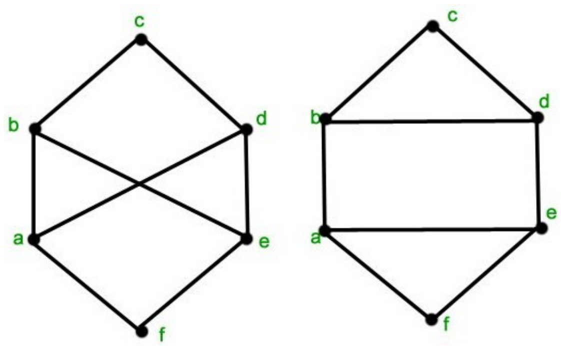
Theoretically, a Gram matrix $K \in Rn \times n$ with respect to $\{x1,\ldots,xn\}$ (sometimes also called a "kernel matrix"[3]), where $Kij=k(xi,xj)$, must be positive semi-definite (PSD). Empirically, for machine learning heuristics, choices of a function k that do not satisfy Mercer's condition may still perform reasonably if k at least approximates the intuitive idea of similarity. Regardless of whether k is a Mercer kernel, k may still be referred to as a "kernel".

3.2 What is a Graph kernel

A graph kernel is a kernel function that computes an inner product on graphs. Graph kernels can be intuitively understood as functions measuring the similarity of pairs of graphs. They allow kernelized learning algorithms such as support vector machines to work directly on graphs, without having to do feature extraction to transform them to fixed-length, real-valued feature vectors. Here a graphical way to see the kernel graph isomorphism problem:



All starts with Graph isomorphism: Find a mapping f of the vertices of G1 to the vertices of G2 such that G1 and G2 are identical; Here an example of an isomorphism problem:



i.e. (x,y) is an edge of G1 iff (f(x),f(y)) is an edge of G2. Then f is an isomorphism, and G1 and G2 are called isomorphic. No polynomial-time algorithm is known for graph isomorphism. Neither is it known to be NP-complete.

We can move to Subgraph isomorphism. Subgraph isomorphism asks if there is a subset of edges and vertices of G1 that is isomorphic to a smaller graph G2. Subgraph isomorphism is NP-complete.

Pros

- Subgraph isomorphism is simpler
- more approximation ease the problem
- Solving the problem

Drawbacks:

- Excessive runtime in worst case
- Runtime may grow exponentially with the number of nodes

- For larger graphs with many nodes and for large datasets of graphs, this is an enormous problem

The more common way to proceed though is to create a kernel function that should perform a reasonable approximation of the graph isomorphism problem and can tell at the end of the process how much two graphs are similar to each other. The way it works is extracting some patterns that we believe are really important and can characterize well the graph such that they can be something like a fingerprint and such that it can be compared.

3.3 The available kernels

The kernels we used come from GraKel [125]. It is a library that provides implementations of several well-established graph kernels. The library unifies these kernels into a common framework. Furthermore, it provides implementations of some frameworks that work on top of graph kernels. Specifically, GraKeL contains 15 kernels and 2 frameworks.

Also we introduced a brand new kernel called Dominant-Set Graph Kernel. This kernel is crafted, implemented and invented by the author of this report.

Random Walk

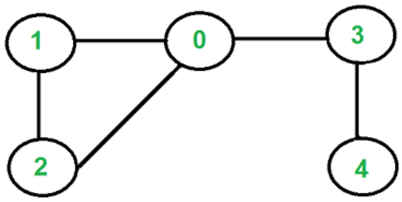
The principle is to count common walks in two input graphs G and G', walks are sequences of nodes that allow repetitions of nodes. The Pros are that walks of length k can be computed by looking at the k-th power of the adjacency matrix, easy. Some Disadvantages are Runtime, Tottering and Halting. Some potential solutions are presented in [68][79][81]. So the direct computation takes $O(n^6)$. The solution is to cast computation of random walk kernel as Sylvester Equation, these can be solved in $O(n^3)$. The equation:

$$AX + XB = C.$$

The Vec-Operator flattens an n x n matrix A into an n^2x1 vector vec(A). It stacks the columns of the matrix on top of each other, from left to right. The Kronecker Product is the product of two matrices A and B in which each element of A is multiplied with the full matrix B. An example here:

$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1\cdot 0 & 1\cdot 3 & 2\cdot 0 & 2\cdot 3 \\ 1\cdot 2 & 1\cdot 1 & 2\cdot 2 & 2\cdot 1 \\ 3\cdot 0 & 3\cdot 3 & 1\cdot 0 & 1\cdot 3 \\ 3\cdot 2 & 3\cdot 1 & 1\cdot 2 & 1\cdot 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 6 \\ 2 & 1 & 4 & 2 \\ 0 & 9 & 0 & 3 \\ 6 & 3 & 2 & 1 \end{bmatrix}$$

The phenomenon of tottering occurs when walk allow for repetitions of nodes. A heavy problem can consist in a walk that can visit the same cycle of nodes all over again. Here in the image it can be that the partition visited remains only the one comprised in the cycle. Here an example of a graph that can coduct to tottering because standing that it is un directed it has a cycle that can be walked indefinitely:



Another problem lies on the fact that a kernel measures similarity in terms of common walks. Hence a small structural similarity can cause a huge kernel value.

Random walk graph kernel has been used as an important tool for various data mining tasks including classification and similarity computation. Despite its usefulness, however, it suffers from the expensive computational cost which is at least $O(n^3)$ or $O(m^2)$ for graphs with n nodes and m edges. A more efficient way to compute it is its variant called Ark that exploits the low rank structure to quickly compute random walk graph kernels in $O(n^2)$ or $O(m)$ time.

Computing Random Walk Graph Kernel can be done with these methods:

- Naive Method. The naive algorithm is to com- pute the Equation (2.1) by inverting the n2 × n2 matrix W. Since inverting a matrix takes time proportional to the cube of the number of rows/columns, the running time is $O(n^6)$.
- Sylvester Method. If the weight matrix can be decomposed into one or two sums of Kronecker products, Sylvester method solves the Equation in $O(n^3)$ time. However, there are two drawbacks in Sylvester method. First, the method requires the two graphs to have the same number of nodes, which is often not true. Second, the theoretical running time of Sylvester method on the weight matrix composed of more than two Kronecker products is unknown.
- Spectral Decomposition Method. For unlabeled and unnormalized matrices, spectral decomposition method runs in $O(n^3)$ time. The problem of spectral decomposition method is that it can't run on the labeled graph or normalized matrix.
- Conjugate Gradient Method. Conjugate gradient (CG) method is used to solve linear systems efficiently. To use CG for computing random walk graph kernel,we first solve (I−cW)x=p for x using CG, and compute qT x. Each iteration of CG takes $O(m^2)$ since the most expensive operation is the matrix-vector multiplication. Thus CG takes $O(m^{2iF})$ time where iF denote the number of iterations. A problem of the CG method is its high memory requirement: it requires $O(m^2)$ memory.
- Iteration method first solves (I − cW)x = p for x by iterative matrix-vector multiplications, and then computes qT x to compute the kernel. Note that the fixed point iteration method converges only when the decay factor c is smaller than |ξ1|−1 where ξ1 is the largest magnitude eigenvalue of W . Similar to CG, the fixed point iteration method takes $O(m^{2iF})$ time for iF iterations, and has the same problems of requiring $O(m^2)$ memory.

Graphlet Kernel

Graphlets are small connected non-isomorphic induced subgraphs of a large network. An induced subgraph must contain all edges between its nodes that are present in the large network, while a partial subgraph may contain only some of these edges.

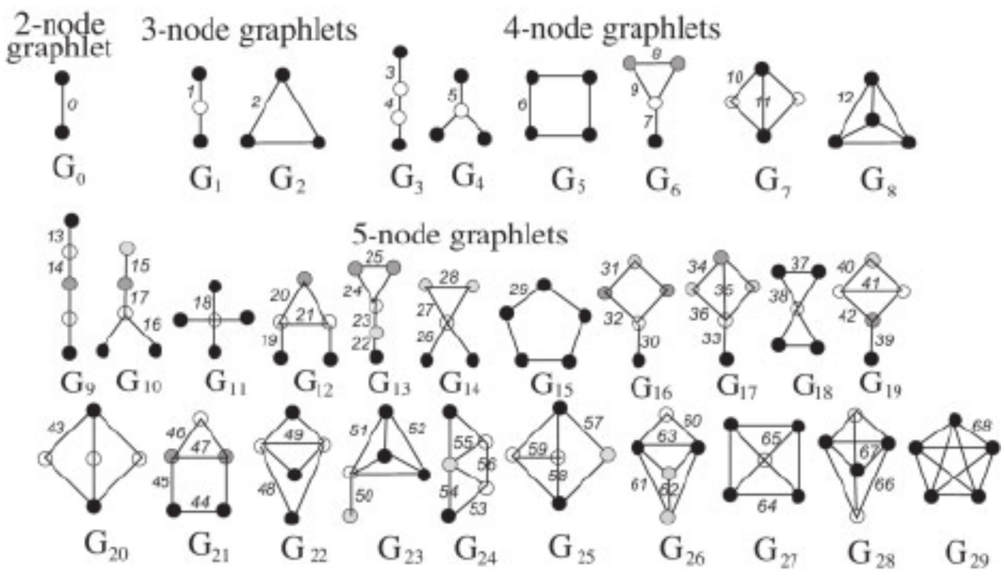
The principle is to count subgraphs of limited size k in G and $G‘$, these subgraphs are referred to as graphlets and then define a graph kernel that counts isomorphic graphlets in two graphs. More formally we let be $G = \{graphlet(1), \ldots, graphlet(N_k)\}$ be the set of size-k graphlets and G be a graph of size n. Define a vector f_G of length N_k whose i-th component corresponds to the frequency of occurrence of $graphlet(i)$ in G, #(graphlet(i) ⊑ G). We will call f_G the k- spectrum of G. This statistic is the foundation of our novel graph kernel. Given two graphs G and G' of size $n \geq k$, the graphlet kernel kg is defined as $kg(G, G') := f_G \cdot f_G'$.

As our goal is to develop scalable graph kernels, we study graphlet kernels based on the 3, 4 and 5 spectra of graphs here. In order to account for differences in the sizes of the graphs, which can greatly skew the frequency counts f_G, we normalize the counts to probability vectors:

Clearly, if $G \sim= G'$, then $f_G = f_G'$. But is the reverse true? It has been shown that when n = k+1 and n ≤ 11, equality of k-spectra implies isomorphism. For n > 11, it is still a conjecture whether a graph of size n can be reconstructed from its subgraphs of size n − 1.

The runtime problems are that the pairwise test of isomorphism is expensive and another one is that the number of graphlets scales as $O(n^k)$. Two solutions on unlabeled graphs are to precompute isomorphisms and to extract sample graphlets. One disadvantage is that the same solutions not feasible on labeled graphs.

Here an example of the most used graphlets:



Weisfeiler-Lehman Kernel

the kernel comes directly from the Weisfeiler-Lehman Isomorphism test that is explained here below.

Here is the algorithm for the Weisfeiler-Lehman Isomorphism Test. It produces for each graph a canonical form. If the canonical forms of two graphs are not equivalent, then the graphs are definitively not isomorphic. However, it is possible for two non-isomorphic graphs to share a canonical form, so this test alone cannot provide conclusive evidence that two graphs are isomorphic. Here the algorithm:

Algorithm 1 One iteration of the 1-dim. Weisfeiler-Lehman test of graph isomorphism

1: Multiset-label determination
- For $i = 0$, set $M_i(v) := l_0(v) = \ell(v)$. [2]
- For $i > 0$, assign a multiset-label $M_i(v)$ to each node v in G and G' which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.

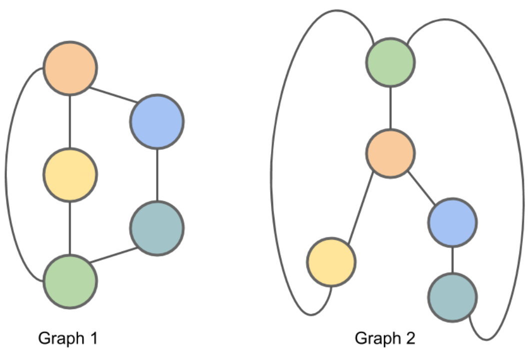
3: Label compression
- Sort all of the strings $s_i(v)$ for all v from G and G' in ascending order.
- Map each string $s_i(v)$ to a new compressed label, using a function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G and G'.

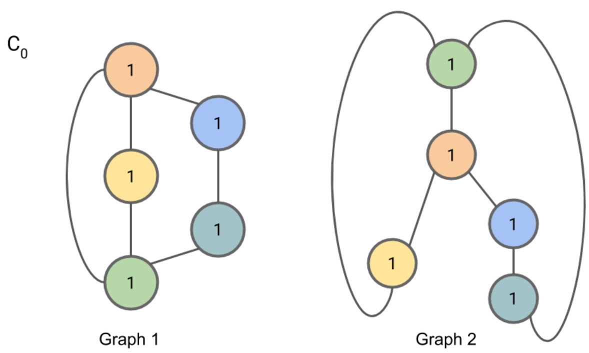
When using this method to determine graph isomorphism, it may be applied in parallel to the two graphs. The algorithm may be terminated early after iteration i if the sizes of partitions of nodes partitioned by compressed labels diverges between the two graphs; if this is the case, the graphs are not isomorphic.

Example of the Weisfeiler-Lehman Isomorphism Test:

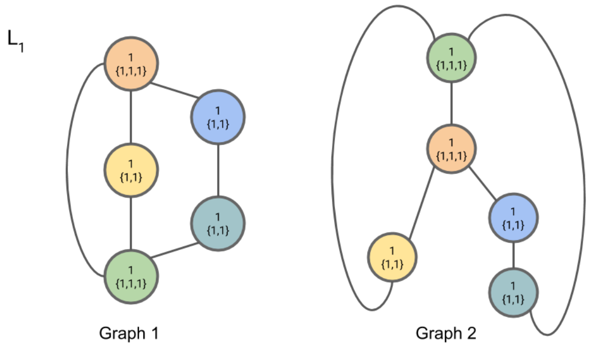
We demonstrate here the Weisfeiler-Lehman isomorphism test using the example graphs from above. The graphs are shown again here for completeness. In the figure here below Graph 1 and Graph 2 are isomorphic. We will apply the Weisfeiler-Lehman isomorphism test to these graphs as a means of illustrating the test:



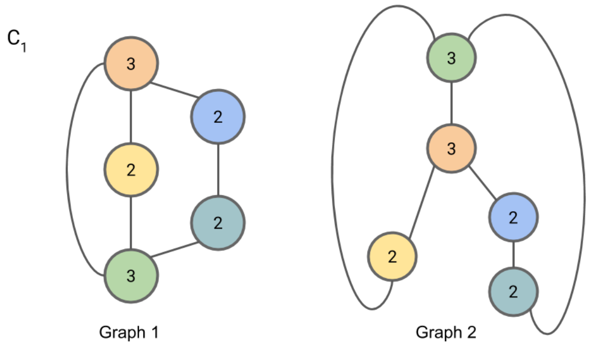
(Step 1) To initialize the algorithm, we set C0,n=1 for all nodes n.



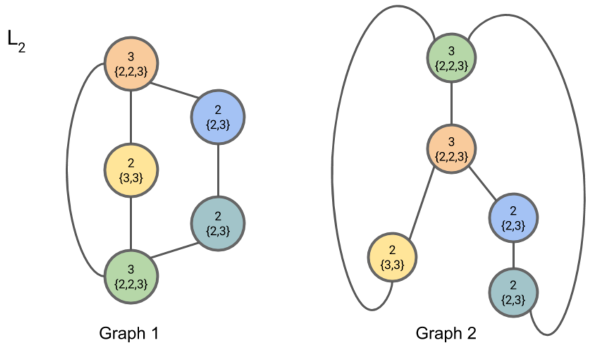
(Step 2) For **(Iteration 1)**, we compute L1. The first part of a node's L is the node's old compressed label; the second part of a node's L is the multiset of the neighboring nodes' compressed labels.



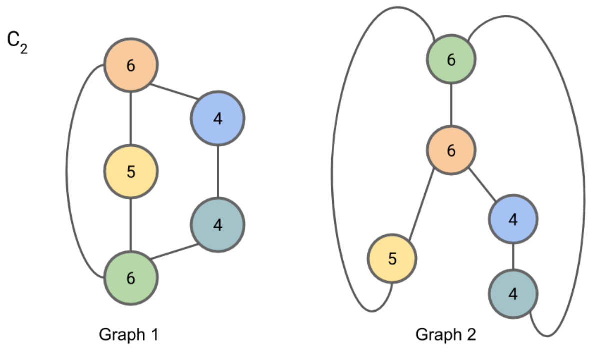
(Step 3) For **(Iteration 1)**, we introduce "compressed" labels C1 for the nodes:



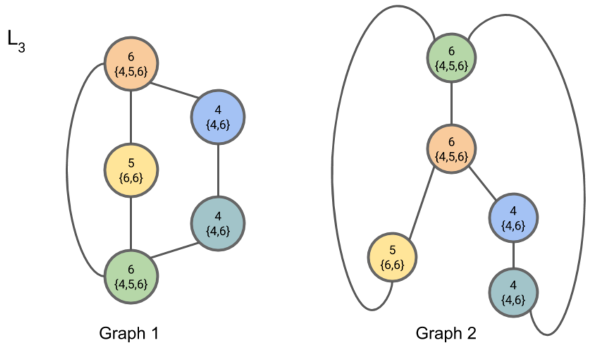
(Step 2) We now begin **(Iteration 2)**. We compute L2:



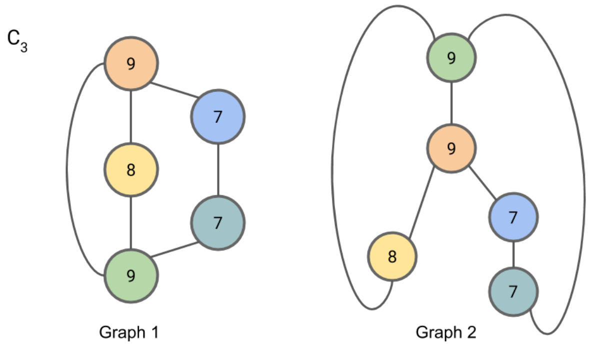
(Step 3) In **(Iteration 2)**, We compute C2:



(Step 2) In **(Iteration 3)**, We compute L3:



(Step 3) In **(Iteration 3)**, we compute C3:



Since the partition of nodes by compressed label has not changed from C2 to C3, we may terminate the algorithm here.

Concretely, the partition of nodes by compressed label may be represented as the number of nodes with each compressed label. That is: **"2 7s, 1 8, and 2 9s"**. This is the canonical form of our graph. Since both Graph 1 and Graph 2 have this same canonical form, we cannot rule out the possibility that they are isomorphic (they *are* in fact isomorphic, but the algorithm doesn't allow us to conclude this definitively.)

DSGK - Dominant Set Graph Kernel

The Dominant set graph kernel came out spontaneously while trying to improve the performances of the graph kernels, in particular it seemed in some cases that the weisfeiler-lehman kernel could have been the best kernel with respect to the others in the classification. The successive hypothesis was that an improvement could have been done if the algorithm could have worked on the **dominant** graphs discarding the ones that were only noise. Stating this hypothesis we used an algorithm to compute the dominant sets on an adjacency matrix through the replicator dynamic technique and after that we computed the weisfeiler-lehman kernel on the dominant set. Intuitively this improves the generality of the prediction because you are going to compute the kernel not on all the points that can be also noisy but on a set that you are guaranteed to be a robust cluster. The experiments are below and show that this algorithms works well in practice being the best one over all the other algorithms.

The algorithm computes only the triangular superior matrix of similarity, in this way it is faster. There are some auxiliary functions such as the "from_set_to_adj" and the "from_adj_to_set". These are auxiliary functions needed to be compliant with the SVM, GraKel and networkx libraries. For sure these operations can be improved in a future version of the kernel.

```
class DomSetGraKer():
  def __init__(self):
    self.train_graphs = None

  def similarity(self,g1adj,g2adj):
    # launch similarity measure Weisfeiler-lehman kernel
    # the inputs are 2 adj matrices of the 2 graphs to be compared
    # the output is the value of similarity

  def fit_transform(self, graphs):
    # return the kernel matrix given the adj matrix of the given set

  def transform(self, graphs):
    # return the kernel matrix given the training set and the test set
```

4. The Manifold Technique

We are going here to answer these questions:

- what is a manifold technique and how to use one ?
- which manifold techniques are available and where ?

4.1 What is a Manifold Technique

It is also called Nonlinear dimensionality reduction. High-dimensional data, meaning data that requires more than two or three dimensions to represent, can be difficult to interpret. One approach to simplification is to assume that the data of interest lie on an embedded non-linear manifold within the higher-dimensional space. If the manifold is of low enough dimension, the data can be visualised in the low-dimensional space.

Consider a dataset represented as a matrix (or a database table), such that each row represents a set of attributes (or features or dimensions) that describe a particular instance of something. If the number of attributes is large, then the space of unique possible rows is exponentially large. Thus, the larger the dimensionality, the more difficult it becomes to sample the space. This causes many problems. Algorithms that operate on high-dimensional data tend to have a very high time complexity. Many machine learning algorithms, for example, struggle with high-dimensional data. This has become known as the curse of dimensionality. Reducing data into fewer dimensions often makes analysis algorithms more efficient, and can help machine learning algorithms make more accurate predictions. Humans often have difficulty comprehending data in many dimensions. Thus, reducing data to a small number of dimensions is useful for visualization purposes.

The reduced-dimensional representations of data are often referred to as "intrinsic variables". This description implies that these are the values from which the data was produced. For example, consider a dataset that contains images of a letter 'A', which has been scaled and rotated by varying amounts. Each image has 32x32 pixels. Each image can be represented as a vector of 1024 pixel values. Each row is a sample on a two-dimensional manifold in 1024-dimensional space (a Hamming space). The intrinsic dimensionality is two, because two variables (rotation and scale) were varied in order to produce the data. Information about the shape or look of a letter 'A' is not part of the intrinsic variables because it is the same in every instance. Nonlinear dimensionality reduction will discard the correlated information (the letter 'A') and recover only the varying information (rotation and scale).

By comparison, if Principal component analysis, which is a linear dimensionality reduction algorithm, is used to reduce this same dataset into two dimensions, the resulting values are not so well organized. This demonstrates that the high-dimensional vectors (each representing a letter 'A') that sample this manifold vary in a non-linear manner.

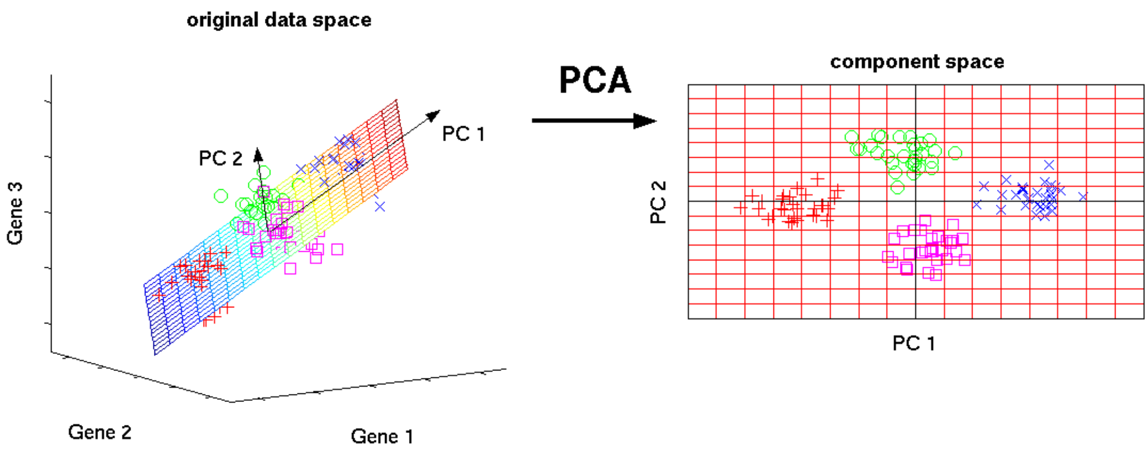
4.2 The available Manifold Techniques

PCA

Principal component analysis (PCA) is a technique that is useful for the compression, visualisation, summarisation and classification of data. The purpose is to reduce the dimensionality of a data set (sample) by finding a new set of variables, smaller than the original set of variables, that nonetheless retains most of the sample's information.

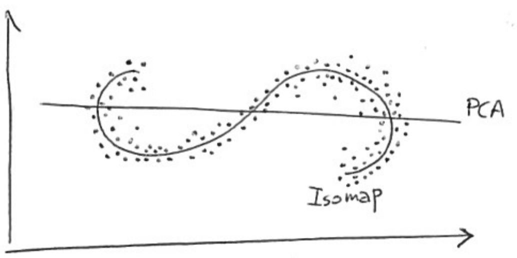
By information we mean the variation present in the sample, given by the correlations between the original variables. The new variables, called principal components (PCs), are uncorrelated, and are ordered by the fraction of the total information each retains.

More formally: given a sample of *n* observations on a vector of *p* variables. Define the first principal component of the sample by the linear transformation $z_1 = a_1^T x$ where the vector $a_1 = (a_{11}, \ldots, ap1)$ is chosen such that is maximum the $Var[z_1]$. You continue adding other dimensions but constraining the successive dimension being orthogonal to the previous one so having zero correlation. Another constraint is the fact that $a_k^T a_k = 1$. In the image below we can se that through the PCA we are moving from a 3D space to a 2D space.



Isomap

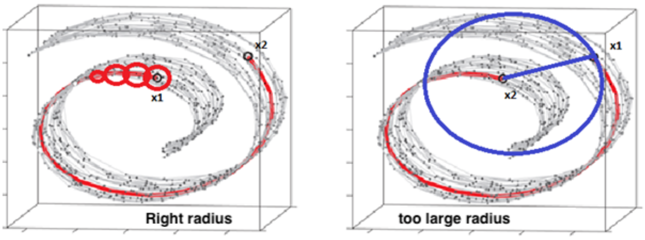
Isomap is a combination of the Floyd–Warshall algorithm with classic Multidimensional Scaling. Classic Multidimensional Scaling (MDS) takes a matrix of pair-wise distances between all points and computes a position for each point. Isomap assumes that the pair-wise distances are only known between neighboring points, and uses the Floyd–Warshall algorithm to compute the pair-wise distances between all other points. This effectively estimates the full matrix of pair-wise geodesic distances between all of the points. Isomap then uses classic MDS to compute the reduced-dimensional positions of all the points. In the image below we can see that while the reduction provided by the PCA is linear w.r.t. the input, the Isomap is non-linear and can capture a greater variance:



Isomap works in 3 phases:

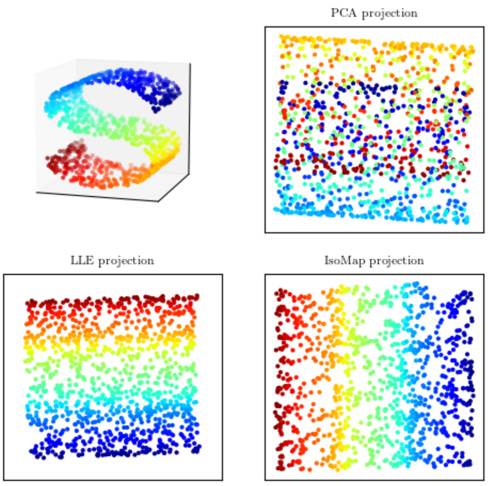
- First we want to determine the umber of nearest neighbors on the manifold using the Euclidean distance, the number of neighbors is selected within a fixed circumference or with a fixed number, for example it can be used the KNN
- In the second step we are going to estimate the shortest path between each pair of points, to do so we can use the dijkstra algorithm
- The third step is about the application of the multi dimensional scaling on the distance matrix of the geodesics to move it to an euclidean space with the same geometrical structure. The vectors of the euclidean space are selected to minimize the cost function $E = \|\tau(D_G) - \tau(D_Y)\|_{L^2}$ where Dy is the matrix of the euclidean distances, D_G is the geodesic matrix, L2 is the classic norm-2 and the τ is a conversion function.

Choosing the right radius is also important:



Locally-linear embedding

Locally-Linear Embedding (LLE) has several advantages over Isomap, including faster optimization when implemented to take advantage of sparse matrix algorithms, and better results with many problems. LLE also begins by finding a set of the nearest neighbors of each point. It then computes a set of weights for each point that best describes the point as a linear combination of its neighbors. Finally, it uses an eigenvector-based optimization technique to find the low-dimensional embedding of points, such that each point is still described with the same linear combination of its neighbors. LLE tends to handle non-uniform sample densities poorly because there is no fixed unit to prevent the weights from drifting as various regions differ in sample densities. As we can see here in the image LLE and Isomap are not so different in the result but LLE can perform better on sparse matrices:

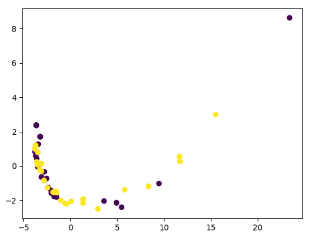
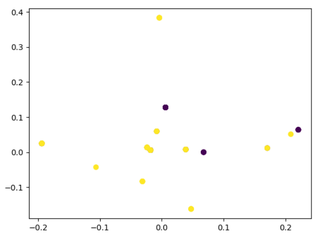
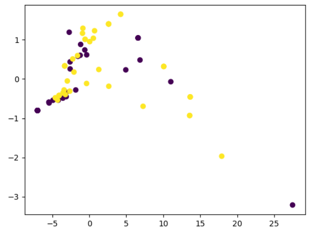


Also the computation of LLE as the ISOMAP happens in 3 main steps:

- Compute the neighbors of each data point.
- Compute the weights that best reconstruct each data point from its neighbors, minimizing the cost in $\mathcal{E}(W) = \sum_i \left|\vec{X}_i - \sum_j W_{ij}\vec{X}_j\right|^2$ by constrained linear fits. This represents the cost of reconstruction error of a point using the neighbours.
- Compute the vectors best reconstructed by the weights ,minimizing the quadratic form in $\Phi(Y) = \sum_i \left|\vec{Y}_i - \sum_j W_{ij}\vec{Y}_j\right|^2$ by its bottom nonzero eigenvectors. This represents the cost for a vector X_i to be represented with lower dimensional reconstruction called Y_i.

Example of the Manifold Reduction

In this example we can see below the various manifold reduction that we experimented with, these tests are all conducted on the Dominant Set Kernel composed with the linear kernel.

PCA	LLE	Isomap
		

5. Experiments and Analysis

To start the experiments we have first chose the kernels below to be tested:

- SPK -> Shortest path
- WLK -> Weisfeiler-lehman
- STK -> Subtree
- DSGK -> Dominant-Set

We have chosen these kernels because they were the most important ones but also they were the faster kernels! For example the Random walk kernel resulted too slow at a first glance so it was discarded. The Dominant set kernels as seen before is a brand-new kernel invented during these experiments to prove the hypothesis that "working on a dominant set is better because improves generality of the results".

The kernels above are tested in their "Vanilla"(no modifications) option.

Each kernel is tested composing the input kernel with other kernels(linear, rbf) or with no composition(precomputed). In the case of reduced matrices it is not possible to use the "precomputed" option because it is needed to have a square matrix as input to the SVM but it is not going to be so with the reduction.

The kernels above are also tested using dimensionality reduction to improve the results(higher generality) and to permit the visualisation. The manifold techniques are listed here:

- no-RED -> no reduction is applied
- ISO -> isometric reduction
- LLE -> linear local embedding reduction
- tSNE -> t-distributed Stochastic Neighbour Embedding applied

In the first column we have a progressive number so the reader can go to the code and find out the specific test in which she is interested in. The second column is a composition of the words we have just seen before, in particular you can read for example "SPK-precomputed-no-RED" in this way: "Shortest path kernel composed with precomputed kernel(so no composition) and no reduction". In the third column we have the scores for the PPI dataset. In the last column we have the SHOCK dataset scores.

4.2 General Results

Results of Manifold Techniques

	method	PPI_score	SHOCK_score
0	SPK-linear-PCA	Acc: min 0.44 - avg 0.73 - max 1.0 - std 0.180	Acc: min 0.2 - avg 0.36 - max 0.5 - std 0.097
1	SPK-rbf-PCA	Acc: min 0.44 - avg 0.72 - max 1.0 - std 0.176	Acc: min 0.2 - avg 0.37 - max 0.45 - std 0.067
2	WLK-linear-PCA	Acc: min 0.44 - avg 0.68 - max 0.87 - std 0.118	Acc: min 0.1 - avg 0.36 - max 0.5 - std 0.107
3	WLK-rbf-PCA	Acc: min 0.22 - avg 0.59 - max 0.88 - std 0.219	Acc: min 0.1 - avg 0.36 - max 0.5 - std 0.105
4	STK-linear-PCA	Acc: min 0.62 - avg 0.76 - max 1.0 - std 0.105	Acc: min 0.2 - avg 0.41 - max 0.6 - std 0.106
5	STK-rbf-PCA	Acc: min 0.44 - avg 0.71 - max 0.88 - std 0.154	Acc: min 0.15 - avg 0.27 - max 0.45 - std 0.100
6	DSGK-linear-PCA	Acc: min 0.55 - avg 0.76 - max 0.88 - std 0.096	Acc: min 0.1 - avg 0.33 - max 0.5 - std 0.130
7	DSGK-rbf-PCA	Acc: min 0.55 - avg 0.76 - max 0.88 - std 0.085	Acc: min 0.1 - avg 0.27 - max 0.4 - std 0.09
8	SPK-precomputed-no-RED	Acc: min 0.33 - avg 0.47 - max 0.77 - std 0.143	Acc: min 0.0 - avg 0.0 - max 0.0 - std 0.0
9	SPK-linear-no-RED	Acc: min 0.55 - avg 0.77 - max 0.87 - std 0.105	Acc: min 0.25 - avg 0.40 - max 0.7 - std 0.142
10	SPK-rbf-no-RED	Acc: min 0.11 - avg 0.62 - max 0.88 - std 0.198	Acc: min 0.2 - avg 0.35 - max 0.6 - std 0.122
11	WLK-precomputed-no-RED	Acc: min 0.11 - avg 0.41 - max 0.77 - std 0.166	Acc: min 0.0 - avg 0.03 - max 0.1 - std 0.045
12	WLK-linear-no-RED	Acc: min 0.22 - avg 0.69 - max 0.88 - std 0.197	Acc: min 0.1 - avg 0.35 - max 0.55 - std 0.112
13	WLK-rbf-no-RED	Acc: min 0.37 - avg 0.47 - max	Acc: min 0.1 - avg 0.28 - max

		0.55 - std 0.053	0.4 - std 0.09
14	STK-precomputed-no-RED	Acc: min 0.11 - avg 0.43 - max 0.77 - std 0.191	Acc: min 0.0 - avg 0.04 - max 0.2 - std 0.061
15	STK-linear-no-RED	Acc: min 0.33 - avg 0.73 - max 1.0 - std 0.208	Acc: min 0.1 - avg 0.37 - max 0.55 - std 0.155
16	STK-rbf-no-RED	Acc: min 0.5 - avg 0.73 - max 1.0 - std 0.151	Acc: min 0.2 - avg 0.37 - max 0.5 - std 0.095
17	DSGK-precomputed-no-RED	Acc: min 0.12 - avg 0.36 - max 0.55 - std 0.146	Acc: min 0.0 - avg 0.02 - max 0.1 - std 0.040
18	DSGK-linear-no-RED	Acc: min 0.55 - avg 0.79 - max 0.88 - std 0.106	Acc: min 0.2 - avg 0.41 - max 0.6 - std 0.128
19	DSGK-rbf-no-RED	Acc: min 0.37 - avg 0.68 - max 0.88 - std 0.163	Acc: min 0.2 - avg 0.24 - max 0.3 - std 0.043
20	SPK-linear-ISO	Acc: min 0.22 - avg 0.62 - max 0.87 - std 0.180	Acc: min 0.1 - avg 0.32 - max 0.45 - std 0.105
21	SPK-rbf-ISO	Acc: min 0.55 - avg 0.75 - max 0.87 - std 0.088	Acc: min 0.2 - avg 0.34 - max 0.45 - std 0.085
22	WLK-linear-ISO	Acc: min 0.33 - avg 0.53 - max 0.66 - std 0.112	Acc: min 0.0 - avg 0.22 - max 0.4 - std 0.110
23	WLK-rbf-ISO	Acc: min 0.37 - avg 0.64 - max 1.0 - std 0.181	Acc: min 0.0 - avg 0.32 - max 0.45 - std 0.128
24	STK-linear-ISO	Acc: min 0.25 - avg 0.56 - max 0.87 - std 0.157	Acc: min 0.05 - avg 0.16 - max 0.3 - std 0.081
25	STK-rbf-ISO	Acc: min 0.22 - avg 0.57 - max 0.77 - std 0.168	Acc: min 0.2 - avg 0.29 - max 0.4 - std 0.072
26	DSGK-linear-ISO	Acc: min 0.37 - avg 0.66 - max 1.0 - std 0.192	Acc: min 0.1 - avg 0.22 - max 0.4 - std 0.116
27	DSGK-rbf-ISO	Acc: min 0.55 - avg 0.73 - max 0.88 - std 0.116	Acc: min 0.2 - avg 0.32 - max 0.45 - std 0.087
28	SPK-linear-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.0 - avg 0.18 - max 0.3 - std 0.084
29	SPK-rbf-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.0 - avg 0.18 - max 0.3 - std 0.084
		Acc: min 0.5 - avg 0.53 - max	Acc: min 0.0 - avg 0.19 - max

30	WLK-linear-LLE	0.55 - std 0.027	0.3 - std 0.105
31	WLK-rbf-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.0 - avg 0.19 - max 0.3 - std 0.105
32	STK-linear-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.0 - avg 0.11 - max 0.2 - std 0.069
33	STK-rbf-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.0 - avg 0.12 - max 0.2 - std 0.075
34	DSGK-linear-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.1 - avg 0.22 - max 0.4 - std 0.095
35	DSGK-rbf-LLE	Acc: min 0.5 - avg 0.53 - max 0.55 - std 0.027	Acc: min 0.1 - avg 0.22 - max 0.4 - std 0.095

6. Conclusions

At the end we would like to mention that the trials to build up this analysis were much more, in particular we tried also to run the graphlet algorithm but due to its high complexity on these huge datasets the time to complete was too long so we dropped. Another dropped kernel was the random walk one, for the same reasons of the graphlet one it was excluded. The Dominant set kernel was improved to perform well than the mere rude version ptting attention on the implementation of the cycles in particular. Another important thing to be mentioned is that the calculus of the Dominant set itself is pretty expensive but we have found a library called "numexpr" that takes numpy arrays and drastically improves the computations on them. The benchmarks are present on the website[129].

It's astonishing but the brand new kernel called DSGK invented during the development of this project is the best performing in the overall ranking so probably it has to be tested more and hopefully it can worths a publication. We hope that this can end up with an advancement in this field. Another conclusion to be drawn is that the reduction reduced the time of the computation but severely decreased the accuracy of the result in the majority of the cases.

Bibliography

1. Manifold Learning and Dimensionality Reduction for Data Visualization... - Stefan Kühn - https://www.youtube.com/watch?v=j8080l9Pvic
2. Unsupervised Learning Explained (+ Clustering, Manifold Learning, ...) - https://www.youtube.com/watch?v=-OEgiMH5aok
3. Unfolding Kernel Embeddings of Graphs - https://www.dsi.unive.it/~atorsell/AI/graph/Unfolding.pdf
4. Graph Kernels - https://www.dsi.unive.it/~atorsell/AI/graph/kernels.pdf

5. Manifold Learning - https://scikit-learn.org/stable/modules/manifold.html
6. In-Depth: Manifold Learning - https://jakevdp.github.io/PythonDataScienceHandbook/05.10-manifold-learning.html
7. What Is Manifold Learning? - https://prateekvjoshi.com/2014/06/21/what-is-manifold-learning/
8. Manifold Learning: The Theory Behind It - https://towardsdatascience.com/manifold-learning-the-theory-behind-it-c34299748fec
9. Introduction to manifold learning - https://onlinelibrary.wiley.com/doi/pdf/10.1002/wics.1222
10. Is Manifold Learning for Toy Data only? - https://www.stat.washington.edu/mmp/Talks/mani-MMDS16.pdf
11. Proximity graphs for clustering and manifold learning - https://papers.nips.cc/paper/2681-proximity-graphs-for-clustering-and-manifold-learning.pdf
12. Manifold Learning - https://indico.in2p3.fr/event/6040/attachments/29587/36427/Manifold_learning.pdf
13. GRAPH CONSTRUCTION FOR MANIFOLD DISCOVERY - https://people.cs.umass.edu/~ccarey/pubs/thesis.pdf
14. Machine Learning on Graphs: A Model and Comprehensive Taxonomy - https://arxiv.org/pdf/2005.03675.pdf
15. Manifold Learning and Spectral Methods - http://mlss2018.net.ar/slides/Pfau-1.pdf
16. Manifold Learning in the Age of Big Data - https://www.stat.washington.edu/mmp/Talks/mani-sppexa19.pdf
17. manifold learning with applications to object recognition - https://people.eecs.berkeley.edu/~efros/courses/AP06/presentations/ThompsonDimensionalityReduction.pdf
18. Representation Learning on Graphs: Methods and Applications - https://www-cs.stanford.edu/people/jure/pubs/graphrepresentation-ieee17.pdf
19. Data Analysis and Manifold Learning (DAML) - http://perception.inrialpes.fr/people/Horaud/Courses/DAML_2011.html
20. Spectral Methods for Dimensionality Reduction - http://cseweb.ucsd.edu/~saul/papers/smdr_ssl05.pdf
21. Robust Principal Component Analysis for Computer Vision - http://files.is.tue.mpg.de/black/papers/rpca.pdf
22. K-means Clustering via Principal Component Analysis - http://ranger.uta.edu/~chqding/papers/KmeansPCA1.pdf
23. K-means Clustering & PCA - https://www.inf.ed.ac.uk/teaching/courses/inf2b/labs/learn-lab3.pdf
24. Charting a Manifold - https://papers.nips.cc/paper/2165-charting-a-manifold.pdf
25. Learning High Dimensional Correspondences from Low Dimensional Manifolds - https://repository.upenn.edu/cgi/viewcontent.cgi?article=1131&context=ese_papers
26. Is manifold learning for toy data only?, Marina Meila - https://www.youtube.com/watch?v=ddhbjCLIjho
27. Locally Linear Embedding - https://www.youtube.com/watch?v=scMntW3s-Wk&list=PL_AYx6iB_DjTXmIN126hH2wZc1aGWb0u9
28. A Global Geometric Framework for Nonlinear Dimensionality Reduction - http://www.robots.ox.ac.uk/~az/lectures/ml/tenenbaum-isomap-Science2000.pdf
29. Laplacian Eigenmaps for dimensionality reduction and data representation - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.5888&rep=rep1&type=pdf

30. Non-linear dimension reduction - http://statweb.stanford.edu/~tibs/sta306bfiles/isomap.pdf
31. Isomap - Isometric feature mapping - https://www.cise.ufl.edu/class/cap6617fa17/ISOMAP.pptx.pdf
32. Pattern Search Multidimensional Scaling - https://arxiv.org/pdf/1806.00416.pdf
33. An Introduction to Locally Linear Embedding - https://cs.nyu.edu/~roweis/lle/papers/lleintro.pdf
34. Nonlinear Dimensionality Reduction by Locally Linear Embedding - http://www.robots.ox.ac.uk/~az/lectures/ml/lle.pdf
35. Manifold Learning: The Price of Normalization - http://www.jmlr.org/papers/volume9/goldberg08a/goldberg08a.pdf
36. Dimensionality Estimation, Manifold Learning and Function Approximation using Tensor Voting - http://www.jmlr.org/papers/volume11/mordohai10a/mordohai10a.pdf
37. Riemannian Manifolds - An Introduction to Curvature - https://www.maths.ed.ac.uk/~v1ranick/papers/leeriemm.pdf
38. Adaptive Neighboring Selection Algorithm Based on Curvature Prediction in Manifold Learning - https://arxiv.org/pdf/1704.04050.pdf
39. Nonlinear Manifold Learning Part One: Background, LLE, IsoMap - http://web.mit.edu/6.454/www/www_fall_2003/ihler/slides.pdf
40. Sparse Manifold Clustering and Embedding - http://cis.jhu.edu/~ehsan/Downloads/SMCE-NIPS11-Ehsan.pdf
41. Sampling Methods for the Nystrom Method - http://www.jmlr.org/papers/volume13/kumar12a/kumar12a.pdf
42. On the Nystro¨m Method for Approximating a Gram Matrix for Improved Kernel-Based Learning - http://www.jmlr.org/papers/volume6/drineas05a/drineas05a.pdf
43. Ensemble Nystro¨m Method - https://papers.nips.cc/paper/3850-ensemble-nystrom-method.pdf
44. Revisiting the Nystr¨om method for improved large-scale machine learning - http://proceedings.mlr.press/v28/gittens13.pdf
45. Spectral Grouping Using the Nystro¨m Method - https://people.eecs.berkeley.edu/~malik/papers/FBCM-nystrom.pdf
46. LAURENS VAN DER MAATEN
47. http://lvdmaaten.github.io/drtoolbox/
48. Dimensionality Reduction: A Comparative Review. - http://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf
49. Visualizing Data using t-SNE - http://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf
50. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation - https://www2.imm.dtu.dk/projects/manifold/Papers/Laplacian.pdf
51. Laplacian eigenmaps and spectral techniquesfor embedding and clustering - http://web.cse.ohio-state.edu/~belkin.8/papers/LEM_NIPS_01.pdf
52. Diffusion Maps: Analysis and Applications - https://core.ac.uk/download/pdf/1568327.pdf
53. Computing and Processing Correspondences with Functional Maps - http://www.lix.polytechnique.fr/~maks/fmaps_SIG17_course/notes/siggraph17_course_notes.pdf
54. Vector Diffusion Maps and the Connection Laplacian - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.435.8939&rep=rep1&type=pdf
55. Nonlinear Dimensionality Reduction II: Diffusion Maps - https://www.stat.cmu.edu/~cshalizi/350/lectures/15/lecture-15.pdf
56. Understanding the geometry of transport: diffusion maps for Lagrangian trajectory data

unravel coherent sets - https://arxiv.org/pdf/1603.04709.pdf
57. Diffusion Maps, Spectral Clustering and Eigenfunctions of Fokker-Planck Operators - https://papers.nips.cc/paper/2942-diffusion-maps-spectral-clustering-and-eigenfunctions-of-fokker-planck-operators.pdf
58. Diffusion Maps for Signal Processing - http://www.eng.biu.ac.il/~gannot/articles/Diffusion%20Magazine.pdf
59. Applications of Diffusion Wavelets - https://core.ac.uk/reader/1145976
60. Diffusion Wavelets and Applications - http://helper.ipam.ucla.edu/publications/mgaws5/mgaws5_5164.pdf
61. Value Function Approximation with Diffusion Wavelets and Laplacian Eigenfunctions - https://people.cs.umass.edu/~mahadeva/papers/nips-paper1-v5.pdf
62. Wavelet methods in statistics: Some recent developments and their applications - https://arxiv.org/pdf/0712.0283.pdf
63. StatQuest: t-SNE, Clearly Explained - https://www.youtube.com/watch?v=NEaUSP4YerM
64. Shortest-path kernels on graphs - https://www.dbs.ifi.lmu.de/~borgward/papers/BorKri05.pdf
65. Generalized Shortest Path Kernel on Graphs - https://arxiv.org/pdf/1510.06492.pdf
66. Shortest-Path Graph Kernels for Document Similarity - https://www.aclweb.org/anthology/D17-1202.pdf
67. An Introduction to Graph Kernels - https://ethz.ch/content/dam/ethz/special-interest/bsse/borgwardt-lab/documents/slides/CA10_GraphKernels_intro.pdf
68. Fast shortest-path kernel computations using approximate methods - http://publications.lib.chalmers.se/records/fulltext/215958/215958.pdf
69. Shortest-path kernels on graphs - https://www.dbs.ifi.lmu.de/Publikationen/Papers/borgwardt.pdf
70. Graphlet Kernels - https://ethz.ch/content/dam/ethz/special-interest/bsse/borgwardt-lab/documents/slides/BNA09_3_4.pdf
71. Efficient graphlet kernels for large graph comparison - http://proceedings.mlr.press/v5/shervashidze09a/shervashidze09a.pdf
72. The Graphlet Spectrum - http://members.cbio.mines-paristech.fr/~nshervashidze/publications/KonSheBor09.pdf
73. Efficient graphlet kernels for large graph comparison - https://people.mpi-inf.mpg.de/~mehlhorn/ftp/AISTATS09.pdf
74. Generalized graphlet kernels for probabilistic inference in sparse graphs - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.720.557&rep=rep1&type=pdf
75. Graphlet Decomposition: Framework, Algorithms, and Applications - https://nickduffield.net/download/papers/KAIS-D-15-00611R2.pdf
76. Efficient Graphlet Counting for Large Networks - https://www.cs.purdue.edu/homes/neville/papers/ahmed-et-al-icdm2015.pdf
77. Halting in Random Walk Kernels - https://papers.nips.cc/paper/5688-halting-in-random-walk-kernels.pdf
78. Graph Kernels - http://www.jmlr.org/papers/volume11/vishwanathan10a/vishwanathan10a.pdf
79. Fast Random Walk Graph Kernel - http://www.cs.cmu.edu/~ukang/papers/fast_rwgk.pdf
80. GRAPH KERNELS - https://sites.cs.ucsb.edu/~xyan/tutorial/GraphKernels.pdf
81. Fast Computation of Graph Kernels - https://pdfs.semanticscholar.org/4459/336b270333c3666310a332acfb2641b27c0d.pdf

82. Weisfeiler-Lehman Graph Kernels - http://www.jmlr.org/papers/volume12/shervashidze11a/shervashidze11a.pdf
83. The Weisfeiler-Lehman Kernel - https://ethz.ch/content/dam/ethz/special-interest/bsse/borgwardt-lab/documents/slides/CA10_WeisfeilerLehman.pdf
84. Wasserstein Weisfeiler-Lehman Graph Kernels - https://papers.nips.cc/paper/8872-wasserstein-weisfeiler-lehman-graph-kernels.pdf
85. Global Weisfeiler-Lehman Kernels - https://arxiv.org/pdf/1703.02379.pdf
86. A Persistent Weisfeiler–Lehman Procedure for Graph Classification - http://proceedings.mlr.press/v97/rieck19a/rieck19a.pdf
87. A Fast Approximation of the Weisfeiler-Lehman Graph Kernel for RDF Data - https://work.delaat.net/awards/2013-09-23-paper.pdf
88. RDRToolbox A package for nonlinear dimension reduction with Isomap and LLE. - https://www.bioconductor.org/packages/release/bioc/vignettes/RDRToolbox/inst/doc/vignette.pdf
89. An Introduction to Diffusion Maps - https://inside.mines.edu/~whereman/talks/delaPorte-Herbst-Hereman-vanderWalt-DiffusionMaps-PRASA2008.pdf
90. Convergence of Laplacian Eigenmaps - http://papers.neurips.cc/paper/2989-convergence-of-laplacian-eigenmaps.pdf
91. Laplacian Eigenmap for Image Retrieval - http://people.cs.uchicago.edu/~niyogi/papersps/paper.pdf
92. Quantum Laplacian Eigenmap - https://arxiv.org/pdf/1611.00760.pdf
93. Laplacian Eigenmaps from Sparse, Noisy Similarity Measurements - https://arxiv.org/pdf/1603.03972.pdf
94. Laplacian eigenmaps for multimodal groupwise image registration - https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjIwqKVvqvqAhVU6qYKHeM7AA4QFjAKegQIChAB&url=https%3A%2F%2Frepub.eur.nl%2Fpub%2F100364%2FRepub_100364_O-A.pdf&usg=AOvVaw2xVygozmBAE735xOQ8m_rQ
95. Nonlinear Dimensionality Reduction I: Local Linear Embedding - https://www.stat.cmu.edu/~cshalizi/350/lectures/14/lecture-14.pdf
96. A NOTE ON THE LOCALLY LINEAR EMBEDDING ALGORITHM - https://core.ac.uk/reader/21747186
97. LOCALL Y LINEAR EMBEDDING ALGORI THM - http://jultika.oulu.fi/files/isbn9514280415.pdf
98. Truly Incremental Locally Linear Embedding - http://ai.stanford.edu/~schuon/learning/inclle.pdf
99. Supervised locally linear embedding - http://rduin.nl/papers/icann_03_lle.pdf
100. Me gusta en YouTube: On Graph Kernels - https://www.youtube.com/watch?v=xwVOarJGD7Q
101. Embedding & Manifold Learning - https://moodle.unive.it/mod/resource/view.php?id=176673
102. Wednesday 29/4/2020 - https://drive.google.com/file/d/1jQfGEqw9CYOHYAiIIdxmG7zc-7GKh5sY/view
103. Monday 27/4/2020 - https://drive.google.com/file/d/1IM9csbR7s-ec2_I_ck1GzOoZeaRAaFGx/view
104. Wednesday 22/4/2020 - https://drive.google.com/file/d/1wzkmQJ344orELbQKoVL1P-yrAMofi3v8/view
105. On Graph Kernels - https://www.youtube.com/watch?v=xwVOarJGD7Q
106. Weisfeiler-Lehman Neural Machine for Link Prediction - https://www.youtube.com/watch?v=QYhgLVt56z8

107. Deep Graph Kernels - https://www.youtube.com/watch?v=hqbMbTlTpXU
108. Graph Theory FAQs: 03. Isomorphism Using Adjacency Matrix - https://www.youtube.com/watch?v=UCle3Smvh1s
109. Deep Graph Kernels - https://dl.acm.org/doi/pdf/10.1145/2783258.2783417
110. GRAPHLET COUNTING - http://evlm.stuba.sk/APLIMAT2018/proceedings/Papers/0442_Hocevar.pdf
111. Graphlet based network analysis - https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=2207&context=open_access_dissertations
112. Graphlet Counting for Topological Data Analysis - https://webthesis.biblio.polito.it/7641/1/tesi.pdf
113. Estimating Graphlet Statistics via Lifting - https://arxiv.org/pdf/1802.08736.pdf
114. GEM - https://github.com/palash1992/GEM
115. awesome-graph-classification - https://github.com/benedekrozemberczki/aweso/me-graph-classification
116. node2vec: Embeddings for Graph Data - https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef
117. Graph Embedding - Graph Analysis and Graph Learning - https://maelfabien.github.io/machinelearning/graph_5/#
118. DeepWalk: Implementing Graph Embeddings in Neo4j - https://neo4j.com/blog/deepwalk-implementing-graph-embeddings-in-neo4j/
119. Inference on Graphs with Support Vector Machines - http://members.cbio.mines-paristech.fr/~jvert/talks/040206insead/insead.pdf
120. sklearn.manifold.SpectralEmbedding - https://scikit-learn.org/stable/modules/classes.html#module-sklearn.manifold)
121. sklearn.manifold.LocallyLinearEmbedding - https://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html#sklearn-manifold-locallylinearembedding
122. Graph Classification - https://www.csc2.ncsu.edu/faculty/nfsamato/practical-graph-mining-with-R/slides/pdf/Classification.pdf
123. SVMS and kernel methods for graphs - https://courses.cs.ut.ee/2011/graphmining/Main/KernelMethodsForGraphs
124. Graph Representation Learning and Graph Classification - https://www.cs.uoregon.edu/Reports/AREA-201706-Riazi.pdf
125. GraKeL: A Graph Kernel Library in Python - https://github.com/ysig/GraKeL
126. Fast Subtree kernels on graphs - https://papers.nips.cc/paper/3813-fast-subtree-kernels-on-graphs.pdf
127. weisfeiler lehman isomorphism test - https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/
128. A tutorial on Principal Components Analysis - http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
129. Numexpr - We show how to significantly speed up your mathematical calculations in Numpy and Pandas using a small library - https://www.kdnuggets.com/2020/07/speed-up-numpy-pandas-numexpr-package.html
130. An Introduction to Locally Linear Embedding - https://cs.nyu.edu/~roweis/lle/papers/lleintro.pdf

Appendix

Appendix 1

NP-completeness

A decision problem C is NP-complete iff

- C is in NP
- C is NP-hard, i.e. every other problem in NP is reducible to it.