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

Protein Protein Interaction dataset

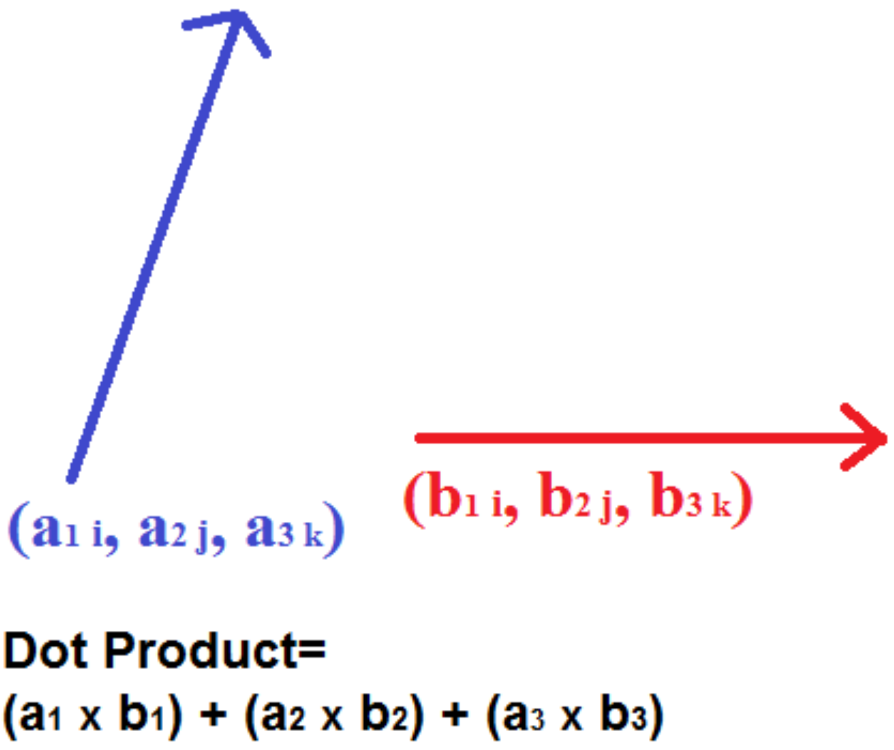
3. The Graph Kernel

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

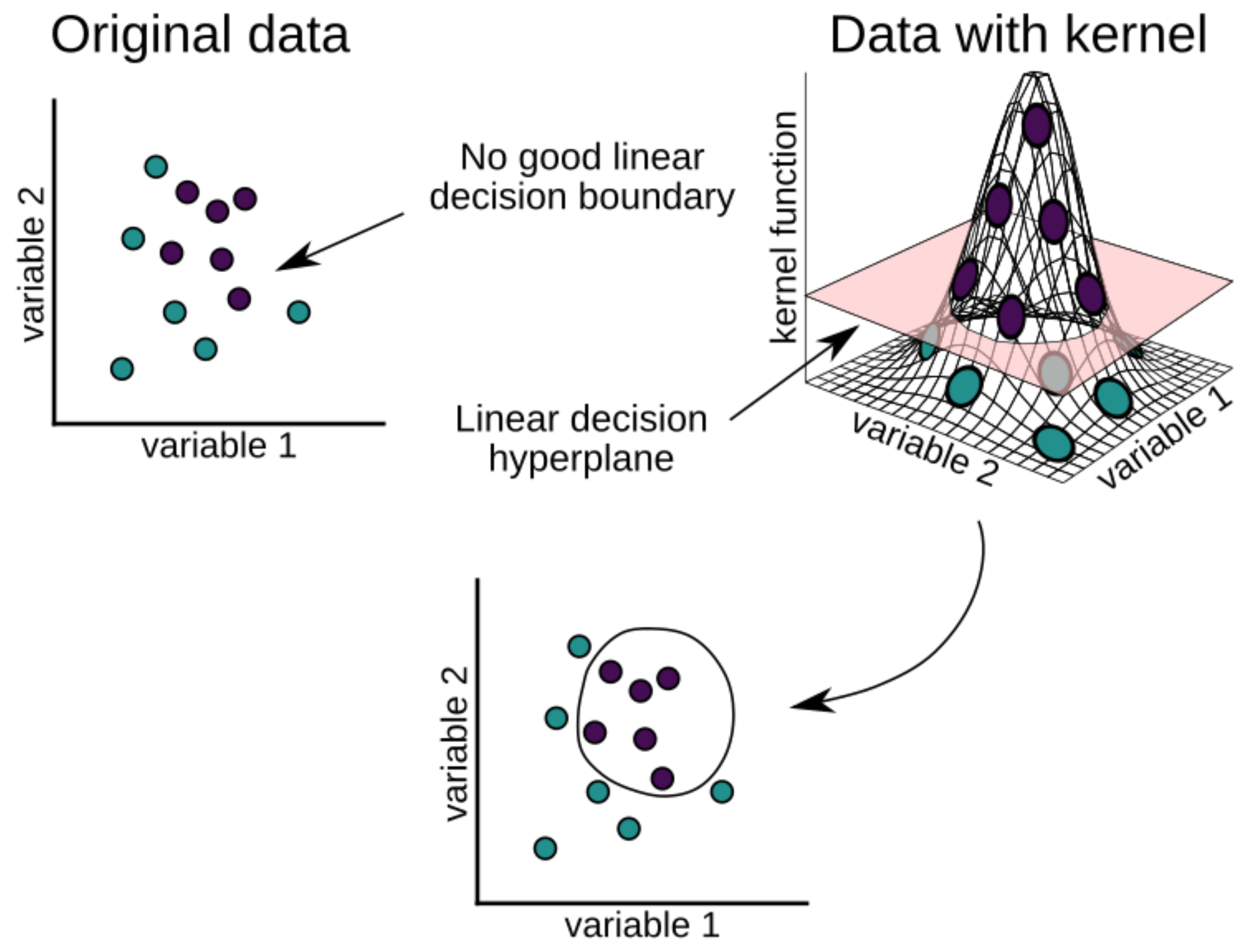
Kernel is a way of computing the dot product of two vectors $\mathbf{x}$ and $\mathbf{y}$ in some (possibly very high dimensional) feature space, which is why kernel functions are sometimes called "generalized dot product". Suppose we have a mapping $\varphi:\mathbb{R}n\rightarrow\mathbb{R}m$ that brings 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.



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:

$$k(x,y)=(1+x^Ty)^2=(1+x_1y_1+x_2y_2)^2==1+x_1^2y_1^2+x_2^2y_2^2+2x_1y_1+2x_2y_2+2x_1x_2y_1y_2$$

Note that this is nothing else but a dot product between two vectors $(1,x_1^2,x_2^2,\sqrt{2}x_1,\sqrt{2}x_2,\sqrt{2}x_1x_2)$ and $(1,y_1^2,y_2^2,\sqrt{2}y_1,\sqrt{2}y_2,\sqrt{2}y_1y_2)$. So the kernel $k(x,y)=(1+x^Ty)^2=\varphi(x)^T\varphi(y)$ computes a dot product in 6-dimensional space without explicitly visiting this 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 φϕ.

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. 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.

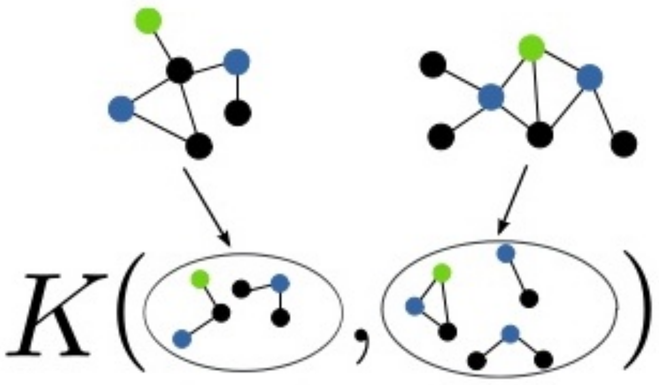
K is said to be non-negative definite (or positive semidefinite) if and only if:

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

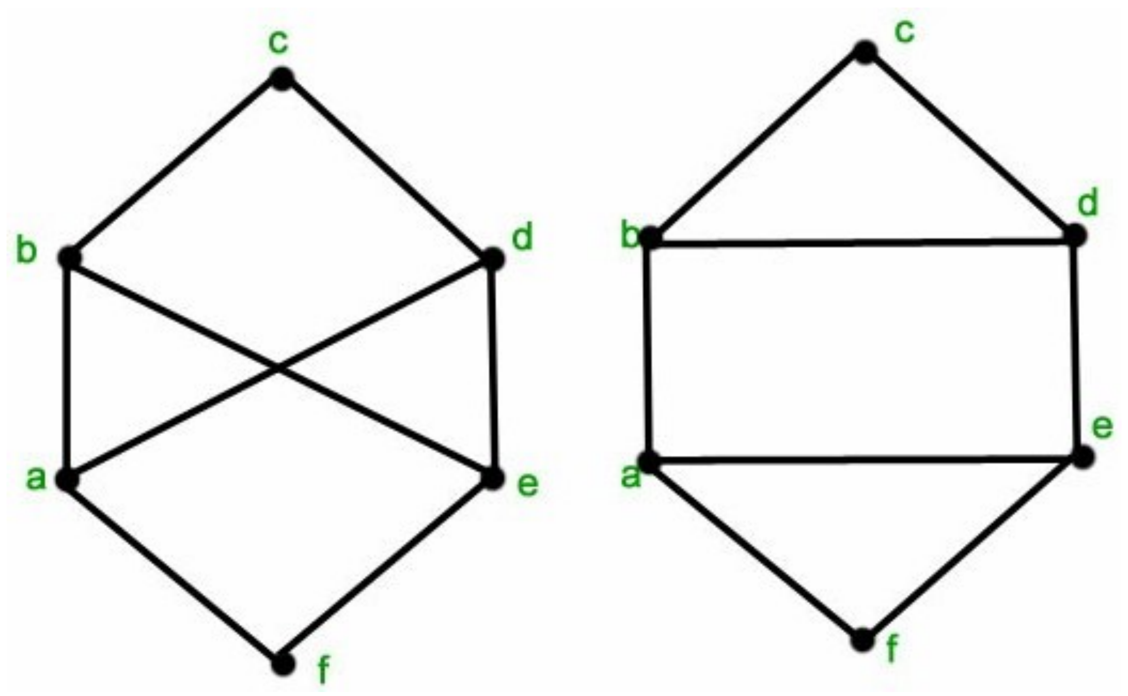
Theoretically, a Gram matrix K∈Rn×n with respect to {x1,...,xn} (sometimes also called a "kernel matrix"[3]), where Kij=k(xi,xj), must be positive semi-definite (PSD).[4] Empirically, for machine learning heuristics, choices of a function k that do not satisfy Mercer's condition may still perform reasonably if k at least approximates the intuitive idea of similarity.[5] 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.



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;



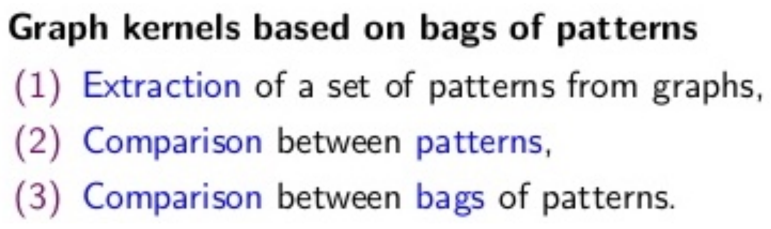
i.e. (x,y) is an edge of G1 iff (f(x),f(y)) is an edge of G2. Then f is an isomorphism, and G1 and G2 are called isomorphic. No polynomial-time algorithm is known for graph isomorphism. Neither is it known to be NP-complete.

We can move to Subgraph isomorphism(easier?). 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.

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 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. This approach is called graph kernel through bag of patterns. The Pros are that we can control the precision and the computational cost moving from easier of more complex extractions of patterns.



Graph kernels based on bags of patterns

(1) Extraction of a set of patterns from graphs,

(2) Comparison between patterns,

(3) Comparison between bags of patterns.

3.3 The available kernels

The kernels we used come from GraKel. Explain...

Random Walk

Principle:

- Count common walks in two input graphs G and G’
- Walks are sequences of nodes that allow repetitions of nodes

Pros:

- Elegant computation
- Walks of length k can be computed by looking at the k-th power of the adjacency matrix ! Construct direct product graph of G and G'
- Count walks in this product graph Gx=(Vx,Ex)
- Each walk in the product graph corresponds to one walk in G and G'

Disadvantages:

- Runtime problems
- Tottering
- Halting

Potential solutions:

- Fast computation of random walk graph kernels (Vishwanathan et al., NIPS 2006) ! Preventing tottering and label enrichment (Mahe et al., ICML 2004)
- Graph kernels based on shortest paths (B. and Kriegel, ICDM 2005)

Direct computation: O(n**6)**

Solution: Cast computation of random walk kernel as Sylvester Equation ! These can be solved in O(n3)

Vec-Operator flattens an n x n matrix A into an n2 x1 vector vec(A).

- It stacks the columns of the matrix on top of each other, from left to right.

Vec-Operator and Kronecker Products

Kronecker Product

- Product of two matrices A and B
- Each element of A is multiplied with the full matrix B:

Phenomenon of tottering:

- Walks allow for repetitions of nodes
- A walk can visit the same cycle of nodes all over again
- Kernel measures similarity in terms of common walks
- Hence a small structural similarity can cause a huge kernel value

Subtree Kernel (Ramon and Gaertner, 2004)

Principle:

- Compare subtree-like patterns in two graphs
- Subtree-like pattern is a subtree that allows for repetitions of nodes and edges (similar to walk versus path)

For all pairs of nodes v from G and u fromG‘:

- Compare u and v via a kernel function
- Recursively compare all sets of neighbours of u and v via a kernel function

Advantages:

- Richer representation of graph structure than walk-based approach

Disadvantages:

- Runtime grows exponentially with the recursion depth of the subtree-like patterns

Graphlet Kernel (B., Petri, et al., MLG 2007)

Principle:

- Count subgraphs of limited size k in G and G‘
- These subgraphs are referred to as graphlets (Przulj, Bioinformatics 2007)
- Define graph kernel that counts isomorphic graphlets in two graphs

Runtime problems

- Pairwise test of isomorphism is expensive ! Number of graphlets scales as O(nk)

Two solutions on unlabeled graphs

- Precompute isomorphisms
- Sample graphlets

Disadvantage:

- Same solutions not feasible on labeled graphs

Weisfeiler-Lehman Kernel

Our graph kernels use concepts from the Weisfeiler-Lehman test of isomorphism (Weisfeiler and Lehman, 1968), more specifically its 1-dimensional variant, also known as “naive vertex refinement”. Assume we are given two graphs *G* and *G*' and we would like to test whether they are isomorphic. The 1-dimensional Weisfeiler-Lehman test proceeds in iterations, which we index by *i* and which comprise the steps given in Algorithm 1.

The key idea of the algorithm is to augment the node labels by the sorted set of node labels of neighbouring nodes, and compress these augmented labels into new, short labels. These steps are then repeated until the node label sets of *G* and *G*' differ, or the number of iterations reaches *n*. See Figure 2, a-d, for an illustration of these steps (note however, that the two graphs in the figure would directly be identified as non-isomorphic by the Weisfeiler-Lehman test, as their label sets are already different in the beginning).

Sorting the set of multisets allows for a straightforward definition and implementation of *f* for the compression of labels in step 4: one keeps a counter variable for *f* that records the number of distinct strings that *f* has compressed before. *f* assigns the current value of this counter to a string if an identical string has been compressed before, but when one encounters a new string, one increments the counter by one and *f* assigns its value to the new string. The sorted order of the set of multisets guarantees that all identical strings are mapped to the same number, because they occur in a consecutive block. However, note that the sorting of the set of multisets is not required for defining *f* . Any other injective mapping will give equivalent results. The alphabet Σ has to be sufficiently large for *f* to be injective. For two graphs, |Σ| = 2*n* suffices.

The Weisfeiler-Lehman algorithm terminates after step 4 of iteration *i* if {l**i(v)|v ∈ V } ≠ {l**i(v')| '
''

v ∈ *V* }, that is, if the sets of newly created labels are not identical in *G* and *G* . The graphs are then not isomorphic. If the sets are identical after *n* iterations, it means that either *G* and *G*' are isomorphic, or the algorithm has not been able to determine that they are not isomorphic (see Cai et al., 1992, for examples of graphs that cannot be distinguished by this algorithm or its higher- dimensional variants). As a side note, we mention that the 1-dimensional Weisfeiler-Lehman algorithm has been shown to be a valid isomorphism test for almost all graphs (Babai and Kucera, 1979).

Note that in Algorithm 1 we used the same node labeling functions l,*l*0,...,*l***h* for both *G* and *G*' in order not to overload the notation. We will continue using this notation throughout the paper and assume without loss of generality that the domain of these functions l,*l*0,...,*l***h* is the set of all nodes in our data set of graphs, which corresponds to *V* ∪ *V* ' in the case of Algorithm 1.

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

1: Multiset-labeldetermination

• For*i*=0,set*M*(*v*):=*l* (*v*)=l(*v*). 2 *i*0

• 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* ∈ *N* (*v*)}.

2: Sortingeachmultiset • 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: Labelcompression • 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* : Σ∗ → Σ 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* .

DSGK - Dominant Set Graph Kernel

Explain

3.3 Examples

examples of some kernels

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

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.

Plot of the two-dimensional points that results from using a NLDR algorithm. In this case, Manifold Sculpting used to reduce the data into just two dimensions (rotation and scale).

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). The image to the right shows sample images from this dataset (to save space, not all input images are shown), and a plot of the two-dimensional points that results from using a NLDR algorithm (in this case, Manifold Sculpting was used) to reduce the data into just two dimensions.

PCA (a linear dimensionality reduction algorithm) is used to reduce this same dataset into two dimensions, the resulting values are not so well organized.

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.

It should be apparent, therefore, that NLDR has several applications in the field of computer-vision. For example, consider a robot that uses a camera to navigate in a closed static environment. The images obtained by that camera can be considered to be samples on a manifold in high-dimensional space, and the intrinsic variables of that manifold will represent the robot's position and orientation. This utility is not limited to robots. Dynamical systems, a more general class of systems, which includes robots, are defined in terms of a manifold. Active research in NLDR seeks to unfold the observation manifolds associated with dynamical systems to develop techniques for modeling such systems and enable them to operate autonomously.[3]

4.2 The available Manifold Techniques

Isomap

Isomap[5] 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. Landmark-Isomap is a variant of this algorithm that uses landmarks to increase speed, at the cost of some accuracy.

Locally-linear embedding

Locally-Linear Embedding (LLE) was presented at approximately the same time as Isomap. It 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. LLE has no internal model.

LLE computes the barycentric coordinates of a point X**i based on its neighbors X**j. The original point is reconstructed by a linear combination, given by the weight matrix W**ij, of its neighbors. The reconstruction error is given by the cost function E(W).

The weights W**ij refer to the amount of contribution the point X**j has while reconstructing the point X**i. The cost function is minimized under two constraints: (a) Each data point X**i is reconstructed only from its neighbors, thus enforcing W**ij to be zero if point X**j is not a neighbor of the point X**i and (b) The sum of every row of the weight matrix equals 1.

The original data points are collected in a D dimensional space and the goal of the algorithm is to reduce the dimensionality to d such that D >> d. The same weights W**ij that reconstructs the ith data point in the D dimensional space will be used to reconstruct the same point in the lower d dimensional space. A neighborhood preserving map is created based on this idea. Each point Xi in the D dimensional space is mapped onto a point Yi in the d dimensional space by minimizing the cost function

In this cost function, unlike the previous one, the weights Wij are kept fixed and the minimization is done on the points Yi to optimize the coordinates. This minimization problem can be solved by solving a sparse N X N eigen value problem (N being the number of data points), whose bottom d nonzero eigen vectors provide an orthogonal set of coordinates. Generally the data points are reconstructed from K nearest neighbors, as measured by Euclidean distance. For such an implementation the algorithm has only one free parameter K, which can be chosen by cross validation.

Laplacian eigenmaps

See also: Manifold regularization

Laplacian Eigenmaps[7] uses spectral techniques to perform dimensionality reduction. This technique relies on the basic assumption that the data lies in a low-dimensional manifold in a high-dimensional space.[8] This algorithm cannot embed out-of-sample points, but techniques based on Reproducing kernel Hilbert space regularization exist for adding this capability.[9] Such techniques can be applied to other nonlinear dimensionality reduction algorithms as well.

Traditional techniques like principal component analysis do not consider the intrinsic geometry of the data. Laplacian eigenmaps builds a graph from neighborhood information of the data set. Each data point serves as a node on the graph and connectivity between nodes is governed by the proximity of neighboring points (using e.g. the k-nearest neighbor algorithm). The graph thus generated can be considered as a discrete approximation of the low-dimensional manifold in the high-dimensional space. Minimization of a cost function based on the graph ensures that points close to each other on the manifold are mapped close to each other in the low-dimensional space, preserving local distances. The eigenfunctions of the Laplace–Beltrami operator on the manifold serve as the embedding dimensions, since under mild conditions this operator has a countable spectrum that is a basis for square integrable functions on the manifold (compare to Fourier serieson the unit circle manifold). Attempts to place Laplacian eigenmaps on solid theoretical ground have met with some success, as under certain nonrestrictive assumptions, the graph Laplacian matrix has been shown to converge to the Laplace–Beltrami operator as the number of points goes to infinity.[10] Matlab code for Laplacian Eigenmaps can be found in algorithms[11] and the PhD thesis of Belkin can be found at the Ohio State University.[12]

In classification applications, low dimension manifolds can be used to model data classes which can be defined from sets of observed instances. Each observed instance can be described by two independent factors termed 'content' and 'style', where 'content' is the invariant factor related to the essence of the class and 'style' expresses variations in that class between instances.[13]Unfortunately, Laplacian Eigenmaps may fail to produce a coherent representation of a class of interest when training data consist of instances varying significantly in terms of style.[14] In the case of classes which are represented by multivariate sequences, Structural Laplacian Eigenmaps has been proposed to overcome this issue by adding additional constraints within the Laplacian Eigenmaps neighborhood information graph to better reflect the intrinsic structure of the class.[15] More specifically, the graph is used to encode both the sequential structure of the multivariate sequences and, to minimise stylistic variations, proximity between data points of different sequences or even within a sequence, if it contains repetitions. Using dynamic time warping, proximity is detected by finding correspondences between and within sections of the multivariate sequences that exhibit high similarity. Experiments conducted on vision-based activity recognition, object orientation classification and human 3D pose recovery applications have demonstrate the added value of Structural Laplacian Eigenmaps when dealing with multivariate sequence data.[15] An extension of Structural Laplacian Eigenmaps, Generalized Laplacian Eigenmaps led to the generation of manifolds where one of the dimensions specifically represents variations in style. This has proved particularly valuable in applications such as tracking of the human articulated body and silhouette extraction.[16]

Kernel principal component analysis

Perhaps the most widely used algorithm for manifold learning is kernel PCA.[31] It is a combination of Principal component analysis and the kernel trick. PCA begins by computing the covariance matrix of the m×n matrix X

It then projects the data onto the first k eigenvectors of that matrix. By comparison, KPCA begins by computing the covariance matrix of the data after being transformed into a higher-dimensional space,

It then projects the transformed data onto the first k eigenvectors of that matrix, just like PCA. It uses the kernel trick to factor away much of the computation, such that the entire process can be performed without actually computing $\Phi(x)$. Of course Φ must be chosen such that it has a known corresponding kernel. Unfortunately, it is not trivial to find a good kernel for a given problem, so KPCA does not yield good results with some problems when using standard kernels. For example, it is known to perform poorly with these kernels on the Swiss roll manifold. However, one can view certain other methods that perform well in such settings (e.g., Laplacian Eigenmaps, LLE) as special cases of kernel PCA by constructing a data-dependent kernel matrix.[32]

KPCA has an internal model, so it can be used to map points onto its embedding that were not available at training time.

Diffusion maps

Diffusion maps leverages the relationship between heat diffusion and a random walk (Markov Chain); an analogy is drawn between the diffusion operator on a manifold and a Markov transition matrix operating on functions defined on the graph whose nodes were sampled from the manifold.[34]](https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction#cite_note-34) In particular, let a data set be represented by X=[x1,x2,…,xn]∈Ω⊂RD![\mathbf {X} = [x{1},x{2},\ldots ,x_{n}]\in \Omega \subset \mathbf {R^{D}} . The underlying assumption of diffusion map is that the high-dimensional data lies on a low-dimensional manifold of dimension d. Let X represent the data set and μ represent the distribution of the data points on X. Further, define a kernel which represents some notion of affinity of the points in X. The kernel k has the following properties[35]

k is symmetric

k is positivity preserving

Thus one can think of the individual data points as the nodes of a graph and the kernel k as defining some sort of affinity on that graph. The graph is symmetric by construction since the kernel is symmetric. It is easy to see here that from the tuple (X,k) one can construct a reversible Markov Chain. This technique is common to a variety of fields and is known as the graph Laplacian.

For example, the graph K = (X,E) can be constructed using a Gaussian kernel.

In the above equation, xi∼xj denotes that xi is a nearest neighbor of xj. Properly, Geodesic distance should be used to actually measure distances on the manifold. Since the exact structure of the manifold is not available, for the nearest neighbors the geodesic distance is approximated by euclidean distance. The choice σ modulates our notion of proximity in the sense that if ‖xi−xj‖2≫σ then Kij=0 and if ‖xi−xj‖2≪σ then Kij=1. The former means that very little diffusion has taken place while the latter implies that the diffusion process is nearly complete. Different strategies to choose σ can be found in.[36]

In order to faithfully represent a Markov matrix, K must be normalized by the corresponding degree matrix D:

P now represents a Markov chain. P(xi,xj) is the probability of transitioning from xi to xj in one time step. Similarly the probability of transitioning from xi to xj in t time steps is given by Pt(xi,xj). Here Pt is the matrix P multiplied by itself t times.

The Markov matrix P constitutes some notion of local geometry of the data set X. The major difference between diffusion maps and principal component analysis is that only local features of the data are considered in diffusion maps as opposed to taking correlations of the entire data set.

K defines a random walk on the data set which means that the kernel captures some local geometry of data set. The Markov chain defines fast and slow directions of propagation through the kernel values. As the walk propagates forward in time, the local geometry information aggregates in the same way as local transitions (defined by differential equations) of the dynamical system.[35]The metaphor of diffusion arises from the definition of a family diffusion distance {Dt}t∈N

For fixed t, Dt defines a distance between any two points of the data set based on path connectivity: the value of Dt(x,y) will be smaller the more paths that connect x to y and vice versa. Because the quantity Dt(x,y) involves a sum over of all paths of length t, Dt is much more robust to noise in the data than geodesic distance. Dt takes into account all the relation between points x and y while calculating the distance and serves as a better notion of proximity than just Euclidean distance or even geodesic distance.

Hessian Locally-Linear Embedding (Hessian LLE)

Like LLE, Hessian LLE[37] is also based on sparse matrix techniques. It tends to yield results of a much higher quality than LLE. Unfortunately, it has a very costly computational complexity, so it is not well-suited for heavily sampled manifolds. It has no internal model.

Modified Locally-Linear Embedding (MLLE)

Modified LLE (MLLE)[38] is another LLE variant which uses multiple weights in each neighborhood to address the local weight matrix conditioning problem which leads to distortions in LLE maps. MLLE produces robust projections similar to Hessian LLE, but without the significant additional computational cost.

4.3 Examples

examples of some Manifold Techniques

5. Comparison

Train an SVM with the kernel chosen applying the manifold technique or not and show the difference

	method	PPI_score	SHOCK_score
0	SPK-precomputed-no-RED	Acc: 49.17%	Acc: 0.0%

1	SPK-linear-no-RED	Acc: 75.14%	Acc: 43.0%
2	SPK-rbf-no-RED	Acc: 62.22%	Acc: 31.5%
3	WLK-precomputed-no-RED	Acc: 42.64%	Acc: 3.0%
4	WLK-linear-no-RED	Acc: 75.56%	Acc: 38.5%
5	WLK-rbf-no-RED	Acc: 47.78%	Acc: 26.0%
6	STK-precomputed-no-RED	Acc: 41.94%	Acc: 1.5%
7	STK-linear-no-RED	Acc: 73.47%	Acc: 42.5%
8	STK-rbf-no-RED	Acc: 67.36%	Acc: 39.5%
9	DSGK-precomputed-no-RED	Acc: 36.25%	Acc: 3.5%
10	DSGK-linear-no-RED	Acc: 79.17%	Acc: 42.0%
11	DSGK-rbf-no-RED	Acc: 67.5%	Acc: 30.0%
12	SPK-linear-ISO	Acc: 62.92%	Acc: 32.0%
13	SPK-rbf-ISO	Acc: 75.83%	Acc: 34.5%
14	WLK-linear-ISO	Acc: 53.47%	Acc: 22.5%
15	WLK-rbf-ISO	Acc: 64.03%	Acc: 33.0%
16	STK-linear-ISO	Acc: 56.81%	Acc: 17.0%
17	STK-rbf-ISO	Acc: 57.22%	Acc: 29.5%
18	DSGK-linear-ISO	Acc: 66.25%	Acc: 23.0%
19	DSGK-rbf-ISO	Acc: 73.19%	Acc: 33.0%
20	SPK-linear-LLE	Acc: 53.33%	Acc: 18.0%
21	SPK-rbf-LLE	Acc: 53.33%	Acc: 18.0%
22	WLK-linear-LLE	Acc: 53.33%	Acc: 19.5%
23	WLK-rbf-LLE	Acc: 53.33%	Acc: 19.5%
24	STK-linear-LLE	Acc: 53.33%	Acc: 11.0%
25	STK-rbf-LLE	Acc: 53.33%	Acc: 12.5%
26	DSGK-linear-LLE	Acc: 53.33%	Acc: 22.0%
27	DSGK-rbf-LLE	Acc: 53.33%	Acc: 22.0%
28	SPK-linear-TSNE	Acc: 56.67%	Acc: 39.5%

29	SPK-rbf-TSNE	Acc: 57.08%	Acc: 41.5%
30	WLK-linear-TSNE	Acc: 70.83%	Acc: 27.5%
31	WLK-rbf-TSNE	Acc: 70.83%	Acc: 41.5%
32	STK-linear-TSNE	Acc: 48.75%	Acc: 15.0%
33	STK-rbf-TSNE	Acc: 53.33%	Acc: 36.5%
34	DSGK-linear-TSNE	Acc: 61.67%	Acc: 25.5%
35	DSGK-rbf-TSNE	Acc: 74.31%	Acc: 33.5%

4.1 Training without Manifold

Training without Manifold

4.2 Training with Manifold

Training with Manifold

4.2 Results

Results of Manifold Techniques

6. Conclusions

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%2

0Magazine.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-wasserst

ein-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_Hocev

ar.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

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.