Università
Ca'Foscari
Venezia

ARTIFICIAL INTELLIGENCE: KNOWLEDGE REPRESENTATION AND PLANNING
CM 0472-1

Assignment 3

Student:

Giosuè Zannini 873810

Academic year 2021/2022

Contents

Chapter 1

Introduction

1.1 Problem

In this assignment I have to choose one graph kernel among:

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

And one manifold learning technique among:

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

After making this choice I had to compare the performance of an SVM trained on the given kernel, with or without the manifold learning step, on the following datasets:

- PPI
- SHOCK

1.2 Idea of implementation

The first point in this assignment is to compare the performance obtained from a learning algorithm, in this specific case I had to use SVM, and the one obtained following an approach of non linear dimensionality reduction using the same algorithm. The main idea of the manifold learning technique is that probably an high-dimensional space can introduce some noise, instead a low-dimensional space can represent better the data points. The second point that is dealt in this assignment is that data are not expressed with the vector data model, but they are expressed using the graph model and for this reason I had to use the graph kernels to can apply a supervised learning algorithm. The datasets that I had to treat they have the following characteristics:

- PPI: this dataset consists of protein-protein interaction (PPIs) networks related to histidine kinase. Histidine kinase is a key protein in the development of signal transduction. If two proteins have direct (physical) or indirect (functional) association, they are connected by an edge. Here I consider the PPIs from two different groups: 40 PPIs from Acidovorax avenae and 46 PPIs from Acidobacteria.

- SHOCK: this dataset consists of graphs from a database of 2D shapes. Each graph is a skeletal-based representation of the differential structure of the boundary of a 2D shape. There are 150 graphs divided into 10 classes, each containing 15 graphs.

For this assignment I chose as manifold Isomap method and as graph kernel Shortest path kernel.

Chapter 2

Graph Kernel

A different way from feature vector to represent entities is to use graphs structure.
A graph G consists of an ordered set of n vertices $V = \{V_1, V_2, \ldots, V_n\}$, and a set of directed or undirected edges $E \subseteq V \times V$. When G is weighted, it means that there are weights on the edges, it can be represented by a $n \times n$ matrix A, with $A_{i,j} = w_{i,j}$ if $(i, j) \in E$. When G is unweighted, it means no weights on the edges, it can be represented by a $n \times n$ matrix A, with $A_{i,j} = 1$ if $(i, j) \in E$. In the specific case of this assignment I was dealing with unweighted and undirected graphs, which means that the matrix A is symmetric. Graph model has the advantage of improving the expressiveness and versatility of models, but this introduce a problem, since I know that machine learning algorithms are based on the usage of vectorial form. Graphs have an arbitrary structure, they are collections of nodes and edges without a location in space, since there are different ways to represent the same graph. Due to the difficulty to convert a graph into a vector, it is necessary to solve the problem giving a method that allows to represent graphs into a space. The solution that I adopted is based on the usage of the kernel trick, which allows to represent graphs not in vectorial representation but using similarity representation. Thanks to kernel trick I can apply SVM to the datasets using a defined similarity measure, it can be expressed like a function $s : G \times G \to \mathbb{R}$, such that $s(G, G')$ quantifies the similarity (or dissimilarity) of G and G'. Generally spoken, graph kernels are based on

the comparison of graph-substructures via kernels, walks, subtrees and cyclic patterns have been considered for this purpose. However, kernels on these substructures are either computationally expensive, sometimes even NP-hard to determine, or limited in their expressiveness. These disadvantages of existing kernel methods are due to competing requirements in graph kernel design: first, the kernel should be a good measure of similarity for graphs; second, its computation should be possible in polynomial time; third, the kernel must still be positive definite; fourth, ideally, it should be applicable to all graphs, not just a small subset of graphs. Existing graph kernels have difficulties with reaching at least one of these four goals.

2.1 Shortest-Path Kernel

The kernel that I chose measure similarity based on shortest paths in graphs, that are computable in polynomial time, that are positive definite and that are applicable to a wide range of graphs. While determining all paths is NP-hard, finding special subsets of paths is not necessarily. Determining longest paths in a graph is again NP-hard, as it would allow to decide whether a graph contains a Hamilton path or not. Computing shortest paths in a graph, however, is a problem solvable in polynomial time. Prominent algorithms such as Dijkstra (for shortest paths from one source node) or Floyd-Warshall (for all pairs of nodes) allow to determine shortest distances in $O(m + n \log n)$ and $O(n^3)$ time, respectively. The essential first step in the shortest-path kernel is to transform the original graphs into shortest-paths graphs. A shortest-paths graph S contains the same set of nodes as the input graph G. Unlike in the input graph, there exists an edge between all nodes in S which are connected by a walk in G. Every edge in S between nodes v_i and v_j is labeled by the shortest distance between these two nodes. Any algorithm which solves the all-pairs-shortest-paths problem can be applied to determine all shortest distance edge labels in

S. I used Floyd's algorithm, because this algorithm is applicable to graphs with negative edge weights, but must not contain negative-weighted cycles. The Shortest-path graph kernel is defined as: Let G_1 and G_2 be two graphs that are Floyd-transformed into $S1$ and S_2. I can then define the shortest-path graph kernel on $S_1 = (V_1, E_1)$ and $S_2 = (V_2, E_2)$ as:

$$K(S_1, S_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k_{\text{length}}(e_1, e_2)$$

where k_{length} is a kernel that compares the lengths of two shortest paths as:

$$k_{\text{length}}(e_i, e_j) = e_i * e_j$$

The advantages of this kernel are:

- No tottering, better accuracy on classification benchmarks.
- Runtime is in $O(n^4)$.
- Empirically faster than random walk kernels

The disadvantages are:

- $O(n^4)$ too slow for large graphs.
- Dense matrix representation for connected graphs, may lead to memory problems on large graphs.

Chapter 3

Manifold Learning

Real world data usually has a high dimensionality, so each instance of a dataset is composed by a large set of features. When analyzing huge sets of data, problems often occur when the data are high-dimensional. This problem was firstly discussed by Richard E. Bellman who gives to it the name of curse of dimensionality, which refers to the problem of finding structure in data embedded in a highly dimensional space. In theory there are different dimensionality reduction techniques that are used for the analyzing and visualize complex sets of data. These techniques can be classified in linear or non linear. Most common method of linear dimension reduction is principle component analysis (PCA), which is basically a linear technique based on preserving the maximum explained variability. PCA finds the directions along which the data has maximum variance in addition to the relative importance of these directions. The main assumption of PCA is that the principle components are a linear combination of the original features. If this is not true, PCA could not provide accurate results. In real life the non linear structures are rather common so for this it is necessary to introduce more sophisticated methods like manifold learning. These algorithms are based on the idea that the dimensionality of many data sets is only artificially high. Indeed the data points can be described as a function of only a few parameters. Manifold learning algorithms attempt to extract a low-dimensional representation that represent as better as possible high-dimensional data.

In this assignment I applied a manifold learning technique to the distance matrix to increase if possible the decision boundary of each decision area. The distance matrix D is computed starting from the kernel matrix K obtained from the Shortest-Path Kernel over a set of n graphs. Specifically D is computed in the following way:

$$d_{i,j} = \sqrt{k_{i,i} + k_{j,j} - 2k_{i,j}}$$

After this I used a SVM with a linear kernel to perform the task.

3.1 Isomap

Isomap stands for isometric mapping has been one of the first algorithms introduced for manifold learning and has gained significant popularity due to its conceptual simplicity and efficient implementation. It is a non-linear dimensionality reduction method based on the spectral theory which tries to preserve the geodesic distances in the lower dimension. Isomap starts by creating a neighborhood network. After that, it uses graph distance to the approximate geodesic distance between all pairs of points. And then, through eigenvalue decomposition of the geodesic distance matrix, it finds the low dimensional embedding of the dataset. In non linear manifolds, the Euclidean metric for distance holds good if and only if neighborhood structure can be approximated as linear. If neighborhood contains holes, then Euclidean distances can be highly misleading. In contrast to this, if I measure the distance between two points by following the manifold, I will have a better approximation of how far or near two points are.

Chapter 4

Results and Comparison

In this section I will treat the performance regard all models. As metric to choose the best hyper parameters for each model I decided to take the best result about the mean between accuracy score and F1 score since in this way I try to optimize whole metrics that I used in this assignment.

4.1 PPI Dataset

4.1.1 Without Isomap

As first I want to show the result about SVM without manifold technique with the best hyper parameter C:

Metrics	Mean	Std deviation
Accuracy	0.7	0.238
Precision	0.667	0.325
Recall	0.633	0.339
F1_score	0.627	0.313

The following graph shows the variation in performance as the C parameter changes:

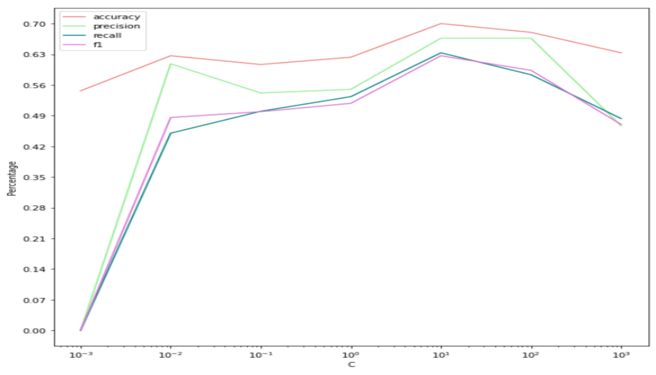

FIGURE 4.1: SVM with Shortest-path Kernel without using Isomap

From this graph I can notice that the best hyper parameter for C is 10. The following confusion matrix shows the performance at test time with the best configuration of model:

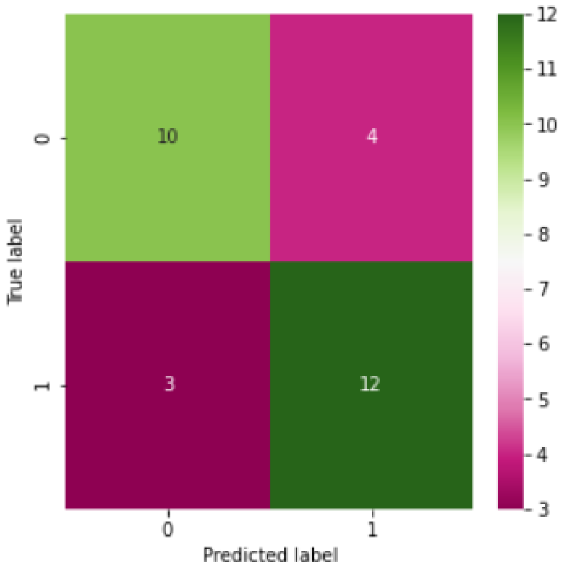

FIGURE 4.2: Confusion matrix associated with the previous results

4.1.2 With Isomap

As before I want to show the result with manifold technique with the best hyper parameter C regards SVM and the best best pair of hyper parameters neighbors and dimension about Isomap:

Metrics	Mean	Std deviation
Accuracy	0.753	0.208
Precision	0.717	0.325
Recall	0.633	0.339
F1_score	0.657	0.323

From the following graphs I can notice the best hyper parameters:

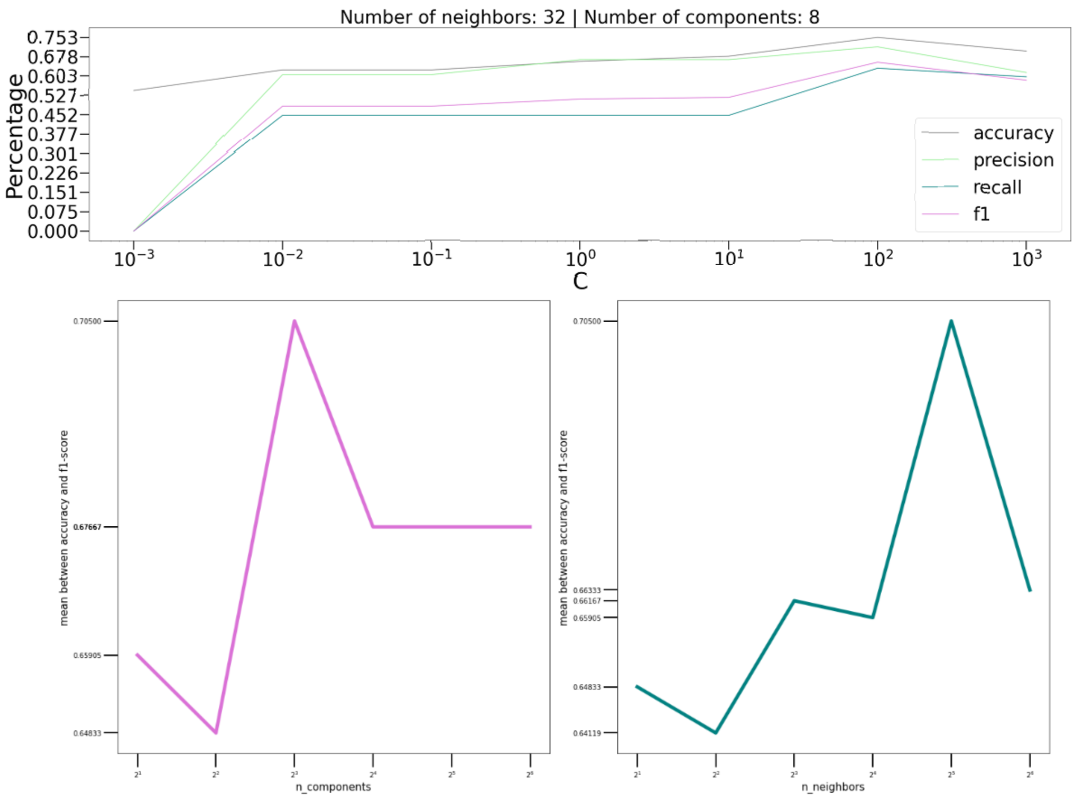

FIGURE 4.3: SVM with Shortest-path Kernel with using Isomap

And they are: C equal 100, n° components equal 8 and n° neighbors equal 32. The following confusion matrix shows the performance at test time with the best configuration of model:

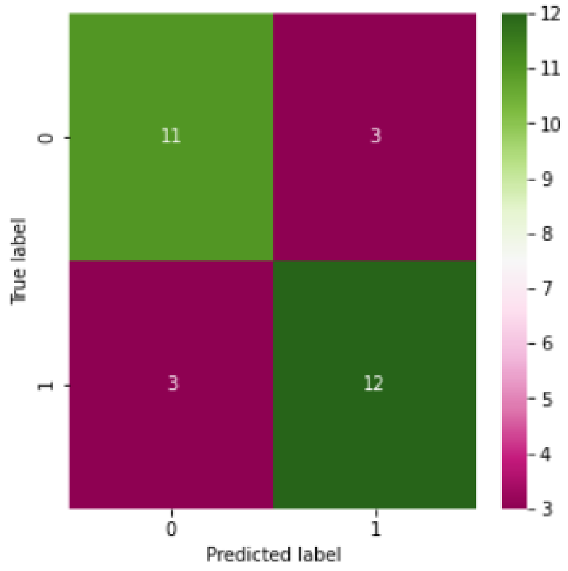

FIGURE 4.4: Confusion matrix associated with the previous results

4.2 SHOCK Dataset

Unlike the previous dataset that was a binary classification problem, this is a multi classification problem because it is composed by 10 classes. SVM can solve this type of task in two different strategies: one vs the rest and one vs one. For this assignment I chose to use one vs the rest approach in this way SVM will create K classifiers each of which solves a two class problem of separating points in a particular class from points not in that class. This is the typical approach, and the class will be the one that maximize the following problem:

$$\max_k f_k(x)$$

where f_k is a discriminate function.

4.2.1 Without Isomap

As first I want to show the result about SVM without manifold technique with the best hyper parameter C:

Metrics	Mean	Std deviation
Accuracy	0.34	0.02
Precision	0.267	0.055
Recall	0.33	0.029
F1_score	0.271	0.034

The following graph shows the variation in performance as the C parameter changes:

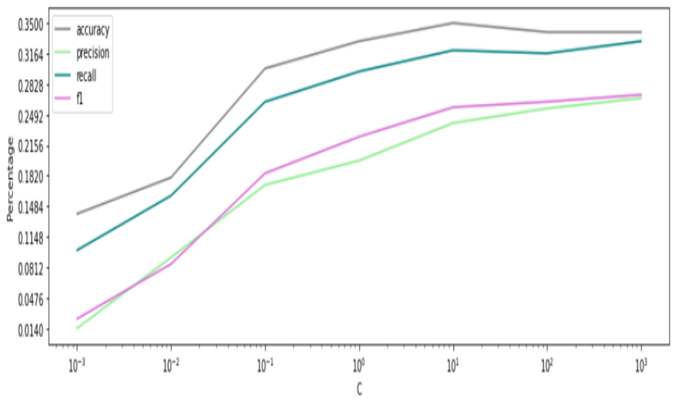

FIGURE 4.5: SVM with Shortest-path Kernel without using Isomap

From this graph I can notice that the best hyper parameter for C is 1000.

The following confusion matrix shows the performance at test time with the best configuration of model:

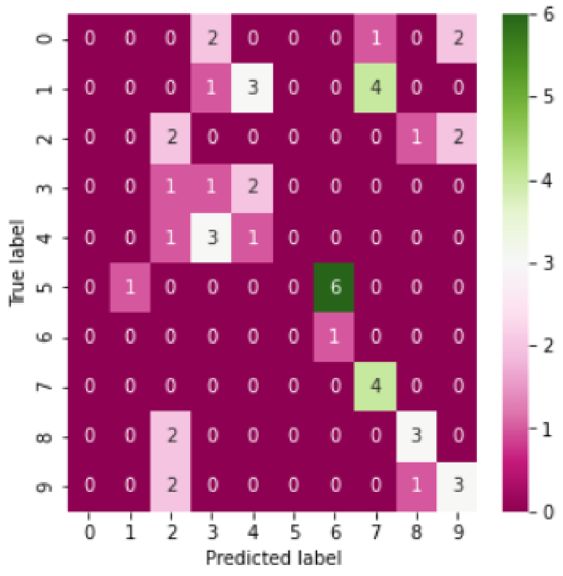

FIGURE 4.6: Confusion matrix associated with the previous results

4.2.2 With Isomap

As before I want to show the result with manifold technique with the best hyper parameter C regards SVM and the best best pair of hyper parameters neighbors and dimension about Isomap:

Metrics	Mean	Std deviation
Accuracy	0.46	0.073
Precision	0.427	0.104
Recall	0.447	0.054
F1_score	0.399	0.078

From the following graphs I can notice the best hyper parameters:

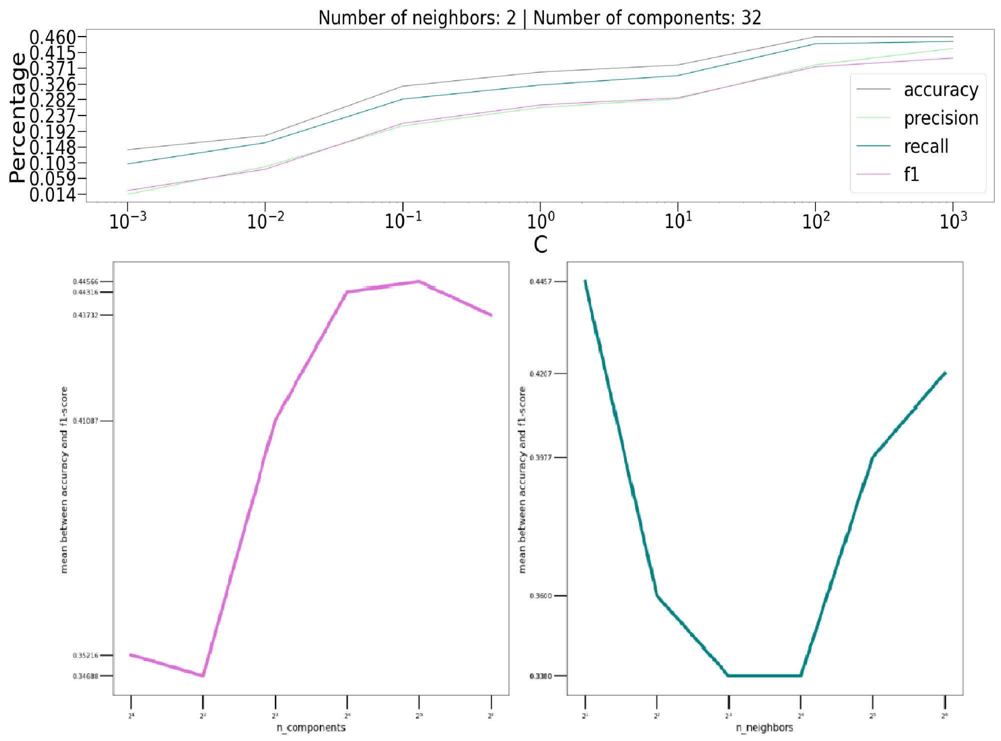

FIGURE 4.7: SVM with Shortest-path Kernel with using Isomap

And they are: C equal 1000, n° components equal 32 and n° neighbors equal 2.

The following confusion matrix shows the performance at test time with the best configuration of model:

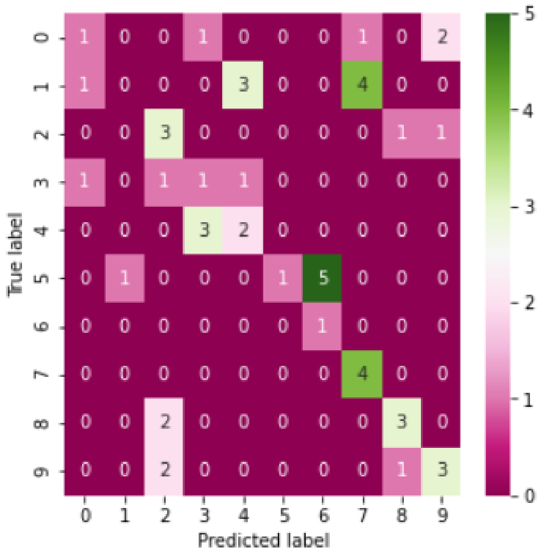

FIGURE 4.8: Confusion matrix associated with the previous results

4.3 Consideration

From these data I can notice that with manifold technique both dataset increase their performance, and this can be seen from the previous tables and graphs. So I can tell that the manifold technique, and in particular case the Isomap helps to increase the decision boundary to well separate the data points of each class. This aspect is more accentuated in SHOCK dataset since the increment of performance is greater.

Chapter 5

Conclusions

In this assignment, I had convert graph structures into vectorial form with the purpose of using the classical machine learning technique SVM over them. I noticed that there is not a specific definition to convert a graph into a vector, but it is possible to solve this problem using the kernel trick, which consists on computing the similarity between the graphs in a specific space. Moreover I had to use manifold learning algorithms, commonly useful for dimensionality reduction. The main aim was to evaluate if Isomap method could enhance the classes separability. Analyzing the obtained results with the two datasets, I have seen that the application of the manifold learning algorithm brought to a slight improvement of the obtained scores, meaning that they have enhanced the class separability a little bit.
Another important aspect is that with any manifold learning technique I am able to embed the original data space into a new one which is composed by less components, and this bring to reduce the computational cost, time and space. Another aspect is that manifold learning techniques are more applicable unlike PCA components since the data in difficult way follow a linear relation among them. As final think I want to notice that Isomap works considering two fundamental hyper parameters: number of neighbors and number of components. I seen that with a small number of components I don't have enough degrees of freedom and the number of neighbors explain how many local information each point considers.

At the end I want to remark that in most of the cases the separation of the data could be increased after applying manifold learning on the original kernels by choosing the right hyper parameters and this suggest that it is generally a good idea to apply this approach when using a graphlet kernel.

Bibliography

[1] S.V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, K. M. Borgwardt, Graph Kernels, Journal of Machine Learning Research 11 (2010).

[2] L. Rossi, A. Torsello, E. R. Hancock, Unfolding Kernel Embeddings of Graphs: Enhancing Class Separation through Manifold Learning.

[3] K. M. Borgwardt, H. P. Kriegel, Shortest-path kernels on graphs.