UCLouvain

epl

École polytechnique de Louvain

Study of semi-supervised classification algorithms on a graph, based on convolutional neural networks and kernels on graph

Author: **Simon CALBERT**
Supervisors: **Marco SAERENS, Bertrand LEBICHOT**
Readers: **Jean-Charles DELVENNE, Sylvain COURTAIN**
Academic year 2019–2020
Master [120] in Computer Science and Engineering

Contents

List of symbols iii

1 Introduction 1

2 Preliminaries 5
2.1 Graphs and networks . . . 5
2.1.1 Definition . . . 5
2.1.2 Natural random walk on graphs . . . 7
2.2 Machine learning . . . 7
2.2.1 Types of learning . . . 7
2.2.2 Underfitting vs Overfitting . . . 8
2.3 Deep learning . . . 9
2.3.1 Feed-Forward Neural Network . . . 9
2.3.2 Convolutional Neural Network (CNN) . . . 10
2.3.3 Learning a deep neural network . . . 11
2.4 Problem statement . . . 13
2.4.1 Autocorrelation hypothesis . . . 13
2.5 Support-Vector Machine . . . 15
2.5.1 Generalisation to multi-class classification . . . 17
2.6 AutoSVM . . . 17
2.7 Diffusion Convolutional Neural Network . . . 19
2.7.1 Architecture . . . 19
2.7.2 Loss function . . . 21
2.7.3 Learning algorithm . . . 21
2.7.4 Early stopping . . . 21
2.8 Graph embedding based on kernels . . . 22
2.8.1 The bag-of-paths framework . . . 22
2.8.2 The margin-constrained bag-of-paths framework . . . 25
2.8.3 Converting a distance to a kernel . . . 27
2.8.4 Extracting features from a kernel . . . 27
2.8.5 Summary . . . 29
2.9 Ranking method and statistical tests . . . 31
2.9.1 Borda count . . . 31
2.9.2 Wilcoxon signed-rank test . . . 31

2.10 Some further related work 32

3 Investigated algorithms **35**
3.1 Trivial method 35
3.2 SVM-based methods 35
3.3 DCNN-based methods 36
3.4 Summary 39

4 Experimental methodology **41**
4.1 Datasets 41
4.1.1 Pre-processing step 42
4.1.2 Statistics 43
4.2 Performance assessment 44

5 Discussion of results **47**
5.1 SVM-based methods 47
5.2 DCNN-based methods 49
5.3 Global discussion 52
5.3.1 Execution time 53
5.3.2 Main findings 54

6 Conclusion **57**
6.1 Main findings 57
6.2 Further works 58
6.3 Limitations 59

Bibliography **61**

A Appendix **65**
A.1 SVM with RL and RCT kernels 65
A.2 DCNN-**X**-**G** 67
A.3 DCNN-**X**-**PR** 68
A.4 Reg-DCNN-**X**-**P** 69
A.5 Auto-DCNN-**X**-**P** 70

List of symbols

$G(\mathcal{V}, \mathcal{E})$	graph or network defined by the set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$
$\lvert\mathcal{S}\rvert$	cardinality of the set $\mathcal{S}$ (number of elements in the set)
x, X	scalar values
$\lvert x\rvert$	absolute value of x
$\mathbb{1}_{\text{[condition]}}$	scalar value equals to 1 if the condition is met, 0 otherwise
$\mathbf{x}$	column vector (bold)
x_i	element in position i of vector $\mathbf{x}$
$\mathbf{e}$	vector full of ones
$\mathbf{e}_i$	vector full of zeros except at the position i where it contains the value 1
$\mathbf{X}$	matrix (bold and uppercase)
$[\mathbf{X}]_{ij}, x_{ij}$	element in position (i,j) of the matrix $\mathbf{X}$
$x_{\bullet\bullet}$	sum of all values of the matrix $\mathbf{X}$
$\bar{\mathbf{X}}$	matrix $\mathbf{X}$ whose each line is divided by the sum of values on the corresponding line (if all $x_{ij} \geq 0$, $\bar{\mathbf{X}}$ is a stochastic matrix)

$\mathbf{X}^n$	n-th power of the matrix $\mathbf{X}$
$\mathbf{x} \circ \mathbf{y}, \mathbf{X} \circ \mathbf{Y}$	respectively elementwise product between vectors $\mathbf{x}$ and $\mathbf{y}$ ($[\mathbf{x} \circ \mathbf{y}]_i = x_i y_i$), and matrices $\mathbf{X}$ and $\mathbf{Y}$ ($[\mathbf{X} \circ \mathbf{Y}]_{ij} = x_{ij} y_{ij}$), also called Hadamard product
$\mathbf{x}_i, \mathbf{x}_{ij}, \mathbf{X}_i$	the indices i and ij are used to identify a vector or a matrix among several vectors and matrices, these indices do not indicate an element inside the vector or the matrix since this behavior corresponds to another notation
$\mathbf{x}^{\mathrm{T}}, \mathbf{X}^{\mathrm{T}}$	respectively the transposition of the vector $\mathbf{x}$ and of the matrix $\mathbf{X}$
$\mathbf{Diag}\left(\mathbf{x}\right)$	diagonal matrix with its diagonal equals to the vector $\mathbf{x}$
$\mathbf{Diag}\left(\mathbf{X}\right)$	diagonal matrix with its diagonal equals to the diagonal of the matrix $\mathbf{X}$
$\mathrm{vec}(\mathbf{X})$	vectorisation of the matrix $\mathbf{X}$, columns of $\mathbf{X}$ are stacked to form a single vector
$\mathbf{x} \div \mathbf{y}$	elementwise division between the vector $\mathbf{x}$ and $\mathbf{y}$ $\left(\frac{x_i}{y_i}\right)$
$\exp(\mathbf{x}), \exp(\mathbf{X})$	respectively elementwise exponential (Euler number) of the vector $\mathbf{x}$ and the matrix $\mathbf{X}$
$\log(\mathbf{X})$	natural logarithm applied elementwise on the matrix $\mathbf{X}$
$\|\|\mathbf{x}\|\|_1$	L_1-norm of the vector $\mathbf{x}$ ($\sum_i \|x_i\|$)
$\|\|\mathbf{x}\|\|_2$	L_2-norm of the vector $\mathbf{x}$ $\left(\sqrt{\sum_i x_i^2}\right)$
$\mathcal{N}(\mu, \sigma)$	normal distribution with mean μ and standard deviation σ
$U(a, b)$	uniform distribution between a and b
$\mathbb{E}$	mathematical expectation

1 Introduction

Nowadays, data are more and more available in large quantity since our society is now in a digital era facilitating their generation and their massive storage. The goal of machine learning is to learn meaningful patterns in these data in order to solve difficult tasks for humans [41]. In many applications, data is generally organised into entities with common features. Many algorithms have been developed to make predictions from these data under the assumption that the entities are independent of each other. However, it turns out that in reality it is common for these entities to be interconnected [14]. To mention just a few [39], in the World-Wide-Web (WWW), web pages are connected to each other via the hyperlinks they contain; in social networks, user profiles are linked by their friendship relationships; in citation networks, scientific papers are interconnected by the citations they contain; etc. These datasets can therefore be represented by a graph whose nodes are the entities and the edges are the links between them. In the examples cited above, it is obvious that the entities influence each other via the links that connect them. It therefore seems relevant to develop models capable of taking these interconnections into account. It often happens in such a system that a common feature is partially missing among the entities. Hence, we may wish to automatically predict this missing information from known data. When the missing values are categorical, we are in the field of semi-supervised classification of nodes on a graph.

Most algorithms exploiting the structure of the graph are built on the hypothesis of spatial autocorrelation [58]. It is based on the idea that close or strongly related entities tend to share similarities in their features. [32, 27] highlighted that taking into account the links between entities can significantly improve performance, in particular on datasets where the spatial autocorrelation hypothesis is strong.

Different types of approaches have been investigated to adapt existing models, using only the features, to be able to also exploit the structure of the graph. Here are some examples.
Graph embedding technique is a common approach. It consists in mapping the graph structure in an Euclidean space while preserving some kind of distance between nodes. Then, this new representation of the graph is added as additional features in a traditional model, such as a Support-Vector Machine model (SVM). The bag-of-paths (BoP) [15] and the margin-constrained bag-of-paths (cBoP) [20] are two

frameworks defining distances between nodes based on random walks on the graph and the minimisation of the free-energy. Classical multidimensional scaling (MDS) and radial basis functions allow to convert these distances into MDS or Gaussian kernels containing measures of global similarity between the nodes. Finally, low-dimensional representations of the graph can be extracted from these kernels and injected into the SVM model. The cBoP extends the BoP by making it possible to fix margins. These margins make it possible to give more importance to particular nodes and therefore to offer greater control over the graph embedding.
Another approach to deal with graph information, called AutoSVM, also based on a SVM model, is proposed in [27]. This model propagates autocovariates (variables correlated with the target) in a recurrent way on the graph structure by training a succession of SVM models. At the end of each training, the autocovariates are updated and injected as additional features in the next SVM model. This process is repeated until the autocovariates converge.
A last approach consists in adding a regularisation term in the objective function to force the model to predict a solution reflecting a strong spatial autocorrelation.
Many other techniques exist but are not investigated in our study.

For some time now, a sub-domain of machine learning, called deep learning, benefits from active research and great enthusiasm due to the performance of its models [41, 17]. These performances are due to the high level of abstraction that they are able to extract from data. Hence, deep learning models are made up of a large number of learnable parameters and therefore require large datasets and significant calculation resources. Different deep learning models have been specially designed to exploit the structure of the graph via convolution operators. Among them, the Diffusion Convolutional Neural Network (DCNN) [2] applies a spatial convolution based on a diffusion process on the graph. This spatial convolution operation allows to work directly on the graph. Its modeling by sharing parameters between the nodes reduces the number of parameters to be adjusted.

This master thesis revolves around the following objectives:

- Compare the quality of low-dimensional representations of the graph from the BoP and cBoP frameworks when they are used for the node classification task. Does the ability to set margins in the cBoP improves performances ? Moreover, are Gaussian kernels more interesting than MDS kernels ?

- The AutoSVM model is a model that has provided interesting results against kernel-based methods in [27]. Is it also competitive against kernels extracted from the BoP and cBoP frameworks ?

- Suggest improvements to the DCNN, inspired by those allowing the SVM model to manipulate graph information. Among them, a reweighting of the diffusion process based on global measures of similarity between the nodes coming from BoP and cBoP kernels; the addition of a regularisation term imposing a spatial autocorrelation in the solutions produced by the model; adapting the idea of autocovariates of the AutoSVM. Moreover, among these modifications, establish which ones are interesting and improve the original DCNN. All of these methods are grouped together as DCNN-based methods.

- Are DCNN-based methods competitive against SVM-based methods ? The discussion must be done based on their performance on features-driven datasets (weak spatial autocorrelation) and on graph-driven datasets (strong spatial autocorrelation). Are traditional methods (not deep learning) still relevant compared to deep learning models ?

- Identify and suggest among all the investigated methods, the most appropriate ones in terms of performance and scalability on features-driven datasets and graph-driven datasets.

The different methods will be evaluated and compared on 10 well-known datasets regularly used for the node classification task. The discussions should be carried out not only on the basis of the performances obtained but also on a global ranking on all the datasets and on a statistical test to highlight any significant difference.

This master thesis is organised as follows. The next chapter contains the theoretical background, as well as a formal description of the problem statement. The following chapter details all the investigated methods, both SVM-based and DCNN-based. Next, a chapter summarises the methodology used, including the datasets and the procedure set up to evaluate the performance of methods. Another chapter is dedicated for the discussion of results. Finally, the last chapter is dedicated to the conclusion about the previous research questions and to the suggestion of some further work allowing to enrich this experimental study.

2 Preliminaries

The purpose of this first chapter is to gradually define all the theoretical stuffs needed to understand: the task on which we focus on, the main issues related to it, the algorithms investigated in Chapter 3, the methodology that was used to assess their performance and how these results are discussed. The last section cites some further related work to solve the node classification task.

2.1 Graphs and networks

A briefly cited in the introduction, a graph or network allows to model a large variety of system in which the entities are linked via pairwise relations. The following subsections introduce the theoretical formalism used in order to manipulate such structures and the concept of random walk on graph which will be used in Section 2.8.

2.1.1 Definition

Mathematically, a graph or network [14, 39] is formalised as being a least a couple $G(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes (entities) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges (links) connecting the nodes. If $i \to j \in \mathcal{E}$, it means that it exists an edge connecting the node i to the node j.

The previous abstract definition of a graph can be encoded in a matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ called the adjacency matrix and defined as

$$[\mathbf{A}]_{ij} = \begin{cases} a_{ij} & \text{if } i \to j \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

For an unweighted graph, $a_{ij} \in \{0, 1\}$ and indicates the presence or absence of a link between two nodes. However, for a weighted graph, $a_{ij} \in \mathbb{R}^{0+}$ and measures the power of the link $i \to j$ (or the degree of affinity from i to j). If $a_{ii} \neq 0$, the link $i \leftrightarrow i$ is called a self-loop.

In an undirected graph, the matrix $\mathbf{A}$ is symmetric, meaning that each relation in $\mathcal{E}$ is symmetric. Hence, if there is a link $i \to j \in \mathcal{E}$, it also exists a link in the other direction $j \to i \in \mathcal{E}$ with the same degree of affinity ($a_{ij} = a_{ji}$). This relationship can then be represented by a single edge $i \leftrightarrow j \in \mathcal{E}$. On the other hand when $\mathbf{A}$ is not symmetric, the graph is directed. Any directed graph can be symmetrised by replacing the adjacency matrix by

$$\frac{\mathbf{A} + \mathbf{A}^{\mathrm{T}}}{2}, \tag{2.2}$$

in which the strength of the symmetrised link $i \leftrightarrow j$ is the average of the affinities in both directions $i \to j$ and $j \to i$.

In addition to the affinity a_{ij} on the edge $i \to j$, a cost $c_{ij} \in \mathbb{R}^{0+}$ can also be considered and interpreted as the cost of jumping from the node i to the node j. All these local costs are gathered in a cost matrix $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ defined by

$$[\mathbf{C}]_{ij} = \begin{cases} c_{ij} & \text{if } i \to j \in \mathcal{E} \\ \infty & \text{otherwise.} \end{cases} \tag{2.3}$$

In our study, we imposed a type of cost often encountered

$$c_{ij} = \frac{1}{a_{ij}}. \tag{2.4}$$

In such a situation, from an electrical point of view, affinities play the role of conductances and the costs are interpreted as resistances.

A path $\wp$ is defined as a sequence of adjacent nodes and can be seen as a walk on the graph. The total cost $\tilde{c}(\wp)$ of a path $\wp$ corresponds to the sum of all local costs encountered on the path which are encoded in the cost matrix $\mathbf{C}$. So, for a given path $\wp = (k_0 = i, k_1, ..., k_t = j)$ starting in node i and ending in node j,

$$\tilde{c}(\wp) = \sum_{\tau=1}^{t} \mathbf{C}_{k_{\tau-1}, k_\tau}. \tag{2.5}$$

A directed graph is said to be strongly connected (or connected for an undirected graph) if it exists at least one path $\wp$ connecting each node to each other node.

2.1.2 Natural random walk on graphs

A transition probability matrix $\mathbf{P}$ [14] can be defined from the adjacency matrix $\mathbf{A}$,

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}, \tag{2.6}$$

where $\mathbf{D} = \mathbf{Diag}(\mathbf{Ae})$ is called the outdegree matrix. $[\mathbf{P}]_{ij}$ corresponds to the probability of jumping from the node i to the node j. This probability is proportional to the degree of affinity between node i and node j. When the graph is connected, the matrix $\mathbf{P}$ is matrix stochastic since each element is positive or zero and each line sums to one. A random walker moving randomly on the graph according to this transition probability matrix, defines the natural random walk on the graph.

2.2 Machine learning

The main goal of artificial intelligence (AI) [41, 46] is to solve tasks that seem easy for people to solve but difficult for people to explicitly describe for the computer. But AI also tries to solve tasks beyond human capabilities. Machine learning (ML) [46, 1, 3, 22] is one approach of AI allowing to solve such tasks based on automated learning from large amount of data by detecting meaningful patterns. This approach benefits from a great property, the adaptability. Indeed, the machine learning model can automatically evolved according to the data provided to it. This property is a great advantage compared to solvers which are explicitly programmed by humans to solve a specific task and which have to be updated manually when the task they solve evolves.
The learning is done over a set of input data (called the training set) in order to convert it into an output (expertise). The quality of the machine learning model is its ability to provide good results on examples it has never seen, therefore its ability to generalise and produce new knowledge.

2.2.1 Types of learning

Machine learning is commonly divided in three categories according to the type of learning used.

- Supervised learning [46, 1] consists int the learning of a mapping function $h : \mathcal{X} \to \mathcal{Y}$ between a domain set $\mathcal{X}$ (inputs of the mapping) to a label set $\mathcal{Y}$ (desired outputs) based on a training set (samples) $S = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{n}$ containing examples of the desired mapping. The training set is supposed to be sampled from an unknown join distribution over the domain

set, $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \overset{\text{i.i.d}}{\sim} \mathcal{D}(\mathcal{X}, \mathcal{Y})$ where i.i.d specifies the assumption that the sample are independent and identically distributed. In general $\mathcal{X} \subseteq \mathbb{R}^d$, so that each input is described by d features encoded in a vector. When the label set $\mathcal{Y} = \{1, 2, ..., k\}$ is finite, it means that we are in the field of classification among k classes and not regression (where the mapping function is continuous).

- Unsupervised learning [46, 1] aims to extract or detect patterns in the domain set $\mathcal{X}$ only. In this situation, we speak of unsupervised learning since the corresponding outputs are not given to the model. The model must infer new knowledge by highlighting particular characteristics of the data. We used a well-known unsupervised learning method in Subsection 2.8.4 to learn a new representation of data, which is called the principal components analysis algorithm [1, 17] in order to reduce the dimensionality.

- Semi-supervised learning [58] is located between the two types of learning mentioned above and constitutes the one considered in our study. It concerns to learn the mapping function $h : \mathcal{X} \to \mathcal{Y}$ defined previously but where this time, the training set S is partially labelled. So we have input data for which we do not have the desired output. Despite these shortcomings, these data still provide significant information since they provide a better understanding of the distribution of the input space $\mathcal{D}(\mathcal{X})$. This kind of problem occurs regularly when it is difficult to obtain the labels associated with the data.
  The semi-supervised learning can be divided in two subcategories [58, 27]. The first is the inductive learning where the mapping function learned is expected to make prediction for unseen data. The second is the transductive learning in which the mapping function is designed to predict the label of only unlabelled training samples. As we will see, the problem addressed in Section 2.4 needs a transductive approach.

2.2.2 Underfitting vs Overfitting

The concepts of underfitting and overfitting [17, 46] are at the heart of any machine learning algorithm. They are directly linked to the performance of the model on both training and unseen data. Indeed, if the model is unable to sufficiently reduce the error on the training set (seen data), we speak of underfitting. On the other hand, when the model learns training data too well (by learning patterns in the training set to reduce the training error as much as possible, which are not relevant for unseen data), it is unable to generalise to unseen data (high generalisation error). So we speak about overfitting when the gap between training error and generalisation error is high.

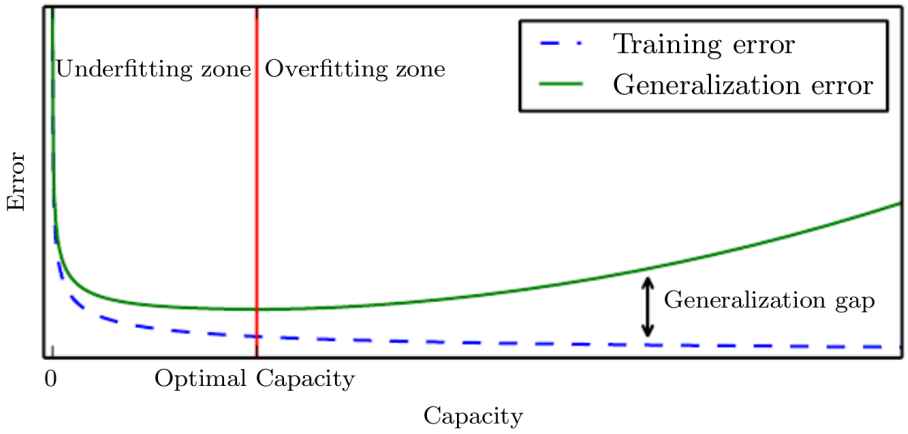

Fig. 2.1.: Highlights the concepts of underfitting and overfitting via the training and generalisation error curves as a function of the capacity of the model [17].

Figure 2.1 illustrates the behavior of the two error curves according to the capacity of the model. The capacity represents the power of the model to fit the training data. It depends on many parameters, such as the the number of adjustable parameters, the learning procedure, the feature space, the training set size, the loss function, etc. Models defined in Subsection 2.5 and 2.7 try to avoid these extrema in different ways. Subsection 4.2 explains the procedure used to have a good estimate of the generalisation error (or accuracy).

2.3 Deep learning

Deep learning [17, 38, 48, 16, 1] is a particular field of machine learning. Indeed, the idea of deep learning is to built machine learning models having particular architectures allowing to learn relevant representations of the data in order to more effectively accomplish the task that the model must solve. According to their architecture, they are cataloged in different categories. To name a few: feed-forward neural network, convolutional neural network, recurrent neural network, autoencoders, etc.

2.3.1 Feed-Forward Neural Network

We focus on the traditional structure of a deep learning model, called the feed forward neural network. This structure is introduced since it will be exploited by an investigated model in Section 2.7. A feed forward neural network [17, 48] builds a sequence of intermediate representation (called layer) of the input data. Each of these intermediate representations is built on the previous one. This incremental approach of learning representations allows the computer to learn more and more

complex concepts based on simpler concepts. We can perceive the inspiration of the human brain. The last representation is used to make the prediction. Information therefore propagates in one direction through several layers until the prediction. Such a structure can be formulated as follows

$$f(\mathbf{x};\boldsymbol{\theta}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x};\boldsymbol{\theta}_1);\boldsymbol{\theta}_2);\boldsymbol{\theta}_3), \tag{2.7}$$

where in this example, $f^{(i)}$ is the i-th layer (representation), the layers 1 and 2 are hidden layers, the layer 3 is the output layer (prediction) and $\boldsymbol{\theta}$ are the parameters to learn in order to build a mapping function with the best accuracy on unobserved data. A characteristic of deep learning models is that the number of parameters is often very high, especially when it is deep (consisting of a large number of successive layers). This gives to it a great complexity but in return requires a large number of examples (large training set) to fit the parameters. This great complexity therefore gives to it a significant chance of fitting the training set, but in exchange exposes it to the danger described in the previous section, the overfitting. Different type of regularisation are used to reduce this risk.

2.3.2 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) [3, 48, 17] are feed forward neural networks able to learn local stationary structure appearing in the data through spatially invariant properties. This is achieved by introducing a form of spatial invariance in the architecture of the network reflecting the spatial invariance in the data.
The first CNN model was built to recognise a digit in an image [29]. A spatial invariance in this situation is that if the digit undergoes a translation, it remains the same digit regardless of its position in the image. However, in the node classification task (Section 2.4), the spatial invariance appears in the fact that the node label is independent of its position (the index of the node that identifies it) in the graph. The distribution of labels is independent of any graph isomorphism (node numbering). Consequently, a CNN contains layers applying an operator behaving in the same way on each spatial position. Such a layer must therefore consist of a spatially invariant operator, in other words a convolution. This is generally implemented by sharing the same parameters over each point of the domain (each pixel for the image and each node for the graph). Parameter sharing [17, p. 247] reduces the number of adjustable parameters. However, this drop in capacity is offset by the prior knowledge about the invariance in the data introduced into the model. In addition, fewer parameters allow faster training time and require a smaller dataset. Thanks to these improvements, several convolutional layers can be stacked to extract high-level representations. It is also common to apply several convolutional operators in parallel (in the same layer) to extract different properties from the same input and

then combine them to make better predictions.
In the context of CNN, the convolutional operator [17] refers to a more general operator than the one defined traditionally in the mathematical literature since the input domain is generally not a continuous domain (e.g: non-Euclidean domain such as a graph). CNNs designed for the node classification task or in other words Convolutional Graph Neural Networks (ConvGNN) [56] are divided according to the kind of convolution used and fall into two categories

- Spatial-based ConvGNN are composed of graph convolutions aggregating information by using directly the relations between nodes. A particular spatial convolution on graph is introduced by [2] and described in Section 2.7. The latter forms the basis of this work.

- Spectral-based ConvGNN incorporates a graph convolution based on the spectral graph theory. This approach consists in considering the features and labels on the nodes as being signals defined on the irregular domain that the graph constitutes. The graph signal processing theory defined an extension of the traditional Fourier transform to a graph Fourier transform in which the notion of frequency is defined by the graph Laplacian matrix [9, 47]. Such a network consists into successive or parallel filtering of signals on graph with learnable filters.

In both cases, the convolution is generally localised (of finite support) in order to extract only local properties around the nodes. This behavior is suggested by the spatial autocorrelation hypothesis (Subsection 2.4.1) according to which a node is influenced only by its close neighborhood. Authors of [56] highlight that spatial-based ConvGNNs are generally more scalable and efficient than spectral-based ConvGNNs because in the spatial approach, the convolution is computed directly on the spatial domain (nodes) while in the spectral approach, the convolution is carried out in the frequency domain requiring the eigendecomposition of the graph Laplacian matrix.

2.3.3 Learning a deep neural network

The learning algorithm [17, 3, 46] has the task of learning from examples how to exploit the architecture of the model using the parameters $\boldsymbol{\theta}$ to obtain the desired output.
Firstly, given a training sample $(\mathbf{x}, y) \in S$, a loss (or cost) function $l(f(\mathbf{x}; \boldsymbol{\theta}), y)$ must quantify the error (deviation) committed by the prediction of the model $f(\mathbf{x}; \boldsymbol{\theta})$ with the expected result y. Since we are interested in minimising the expected error over a random sample coming from the domain set $\mathcal{X}$ according to an unknown

distribution $\mathcal{D}(\mathcal{X}, \mathcal{Y})$, we minimise the expected error over the empirical distribution $\mathcal{D}(S)$ defined by the training set S[1]. Hence, the objective function which is in practice minimised is an approximation of $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}(\mathcal{X},\mathcal{Y})}\, l(f(\mathbf{x};\boldsymbol{\theta}), y)$ and is defined as

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}(S)} l(f(\mathbf{x};\boldsymbol{\theta}), y) = \frac{1}{|S|}\sum_{i=1}^{|S|} l(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), y^{(i)}), \tag{2.8}$$

where $(\mathbf{x}^{(i)}, y^{(i)}) \in S$. The corresponding gradient is given by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}(S)} \nabla_{\boldsymbol{\theta}} l(f(\mathbf{x};\boldsymbol{\theta}), y) = \frac{1}{|S|}\sum_{i=1}^{|S|} \nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), y^{(i)}). \tag{2.9}$$

Now that the objective function is defined, the optimisation problem of a neural network lies in the minimisation of $J(\boldsymbol{\theta})$.

Since the objective function can take complex non-linear shapes due to the network architecture $f(\mathbf{x};\boldsymbol{\theta})$, its optimisation is usually done via an iterative gradient-based method[2]. Hence, it is necessary to compute the gradient according to the parameters $\boldsymbol{\theta}$ to know how to adjust them. However in the case of deep learning models, this gradient may seem a priori difficult to calculate since $f(\mathbf{x};\boldsymbol{\theta})$ can be a relatively complex function. Contrary to this idea, the back-propagation algorithm (see more details in [17, pp. 197–220]) based on the chain rule of calculus allows to compute profitably the gradients $\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$. This algorithm is thus used as a subroutine in the computation of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in (2.9). Due to the complex shape of $L(\boldsymbol{\theta})$, there is a good chance that it is non-convex, and therefore, any iterative gradient-based approach has no guarantee of converging towards a global optimum. In order to reduce the risk of falling and staying in a local minimum, it is common to use a stochastic approach [28]. The most commonly used approach is the stochastic gradient descent (SGD). The idea is that instead of applying a traditional gradient descent on the complete objective function $J(\boldsymbol{\theta})$ over the entire training set as in (2.8), the SGD method minimises different approximations of the objective function at each step of the gradient descent. Each of these approximations is done by considering a subset of the training set (called batch) in (2.8). Parameters are then updated based on an approximation of the full gradient (2.9). When the learning algorithm is passed on all the batches, this constitutes one "epoch". At the start of each new epoch, the training samples are distributed randomly to form new batches. Due to the noise introduced by the batches, the parameters are not updated in the direction of the exact gradient (2.9). This randomness allows the learning process to leave a local minimum of $L(\boldsymbol{\theta})$. Moreover, another advantage is that the computation of one of these gradient approximations is faster since it involves less gradients in (2.9).

[1] In the general case, the training set can contain multiple identical examples resulting of the sampling of the unknown distribution over $\mathcal{X}$. However in many cases duplicates are removed, which assumes a uniform data distribution.

[2] If $J(\boldsymbol{\theta})$ is not differentiable at a point, learning is still possible by using a subgradient of the function instead of the gradient [46, p. 156].

2.4 Problem statement

As mentioned in the introduction, this work focus on the semi-supervised classification on a graph. In this specific task, we not only know the features of the data encoded in the feature matrix $\mathbf{X}$ but also their relations (measures of similarities between them) encoded in a similarity matrix $\mathbf{A}$. This last matrix can then be interpreted as being the adjacency matrix (2.1) of the associated graph $G(\mathcal{V}, \mathcal{E})$. In this work, we consider only symmetrical similarities and no self-similarities. Thus, the corresponding graph is undirected and does not contain self-loops (Section 2.1). Formally the feature matrix is encoded as follows

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_{|\mathcal{V}|}^{\mathrm{T}} \end{bmatrix}, \tag{2.10}$$

where each $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector of node $v_i \in \mathcal{V}$. Hence, $\mathbf{X} \in \mathbb{R}^{|V| \times d}$. In this task, the domain set $\mathcal{X}$ corresponds to the nodes of a graph with additional information (features) on each node. This set is finite and completely known but the data are partially labeled with labels belonging to the set $\mathcal{Y} = \{1, 2, ..., |\mathcal{Y}|\}$. Accordingly, $\mathcal{X}$ is divided into the labelled data $\mathcal{X}_{\mathrm{l}}$ and the unlabelled data $\mathcal{X}_{\mathrm{u}}$, such that $\mathcal{X} = \mathcal{X}_{\mathrm{l}} \cup \mathcal{X}_{\mathrm{u}}$. The class memberships of nodes are gathered in an additional matrix $\mathbf{Y} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{Y}|}$,

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 & \mathbf{y}^2 & \dots & \mathbf{y}^{|\mathcal{Y}|} \end{bmatrix}, \tag{2.11}$$

where $\mathbf{y}^c$ is the class-membership vector of class c defined by

$$\mathbf{y}_i^c = \begin{cases} 0 & \text{if the label of node } v_i \text{ is known but does not belong to class c} \\ -1 & \text{if the label of node } v_i \text{ is not known} \\ 1 & \text{if the label of node } v_i \text{ is known and belongs to class c.} \end{cases} \tag{2.12}$$

The goal is then to build a model $h : \mathcal{X}_{\mathrm{u}} \to \mathcal{Y}$ able to predict the class label of all the unlabelled nodes based on the matrices $\mathbf{A}$, $\mathbf{X}$ and $\mathbf{Y}$. As describe in the Subsection 2.2.1, we are in the field of transductive learning.

2.4.1 Autocorrelation hypothesis

The majority of algorithms exploiting the graph information suppose a strong hypothesis on the distribution of labels on the graph. This assumption [58, p. 51][27] is generally called "local consistency" or "spatial autocorrelation" and considers that close nodes are probably likely to share the same label. This idea finds its foundations in the Tobler first law of geography stating that "everything is related to

everything else, but near things are more related than distant things" [53]. Thus algorithms built on this idea are likely to provide good results on datasets where this assumption is strongly present. Therefore this assumption constitutes one of their major weaknesses. This is why, it is possible that methods which in addition to the features, exploit the information of the graph can provide less good results.
However, two common indices coming from the spatial statistical theory [44] allow to measure the autocorrelation of a labelled graph. They allow to estimate to what extent the label of a node is correlated (influenced) by those of its neighbors (the structure of the network). Strong positive autocorrelation means that similarly labeled nodes tend to be clustered while strong negative autocorrelation indicates a trend of spatial alternation of labels. [44, 27] shows that these two indices come from a small modification of the general cross-product statistic. In addition to providing a quantification of autocorrelation, these two methods provide a statistical test [7] allowing to test the hypothesis of absence of autocorrelation (null hypothesis H_0). That is to say, a threshold beyond which, we can consider that the measure of autocorrelation is unusual in the situation where the null hypothesis was true. In such a situation, we can then reasonably consider that the null hypothesis is false and affirm the presence of autocorrelation.

- Moran's I [7] is given by

$$I(\mathbf{x}) = \frac{|\mathcal{V}|}{a_{\bullet\bullet}} \frac{\sum_{i,j=1}^{|\mathcal{V}|} a_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_{i=1}^{|\mathcal{V}|}(x_i - \bar{x})^2}, \tag{2.13}$$

  where $\bar{x}$ is the average of $\mathbf{x}$ and $a_{\bullet\bullet} = \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} a_{ij}$. The different statistical thresholds are given by

$$\begin{cases} I(x) > I_0 & \text{means positive autocorrelation} \\ I(x) = I_0 & \text{means no autocorrelation} \\ I(x) < I_0 & \text{means negative autocorrelation,} \end{cases} \tag{2.14}$$

  where $I_0 = \frac{-1}{|\mathcal{V}|-1} \approx 0$.

- Geary's c [7] is given by

$$c(\mathbf{x}) = \frac{(|\mathcal{V}|-1)}{2a_{\bullet\bullet}} \frac{\sum_{i,j=1}^{|\mathcal{V}|} a_{ij}(x_i - x_j)^2}{\sum_{i=1}^{|\mathcal{V}|}(x_i - \bar{x})^2}, \tag{2.15}$$

  and the different statistical thresholds by

$$\begin{cases} c(x) < 1 & \text{means positive autocorrelation} \\ c(x) = 1 & \text{means no autocorrelation} \\ c(x) > 1 & \text{means negative autocorrelation.} \end{cases} \tag{2.16}$$

The autocorrelation of the distribution of labels [27] is estimated by averaging the Moran or Geary index on all the class vectors $\mathbf{y}^c$ (2.12) with all labels known (no values -1 in $\mathbf{y}^c$)

$$\begin{cases} I_{\text{label}} = \dfrac{\sum_{c=1}^{|\mathcal{Y}|} I(\mathbf{y}^c)}{|\mathcal{Y}|} \\ c_{\text{label}} = \dfrac{\sum_{c=1}^{|\mathcal{Y}|} c(\mathbf{y}^c)}{|\mathcal{Y}|}. \end{cases} \tag{2.17}$$

These indices are used in Section 4.1.2 to distinguish among the investigated datasets, which benefit from a strong spatial autocorrelation and whose the graph has a better chance of providing relevant information. In the case where the spatial autocorrelation is strong, we will speak of a dataset driven by the graph (graph-driven or $\mathbf{A}$-driven) and in the opposite case, features will be more likely to be more informative and we will speak of a dataset driven by the features (features-driven or $\mathbf{X}$-driven).

2.5 Support-Vector Machine

The support-vector machine algorithm (SVM) [46, 16, 10, 1, 22] is a well known and popular machine learning model for learning linear predictors in order to classify data points between two classes. The algorithm tries to find the separating hyperplane between two classes of points while maximising the margin between him and the data points. The margin of the hyperplane is defined as the minimal distance between a point in the training set and the hyperplane. The set of support vectors $\mathcal{SV}$ is the set of points distant from the hyperplane by a distance equal to the minimum margin. These points are therefore those constraining the hyperplane. Figure 2.2 (left side) shows an example of such hyperplane in a two-dimensional space and highlights the support vectors.

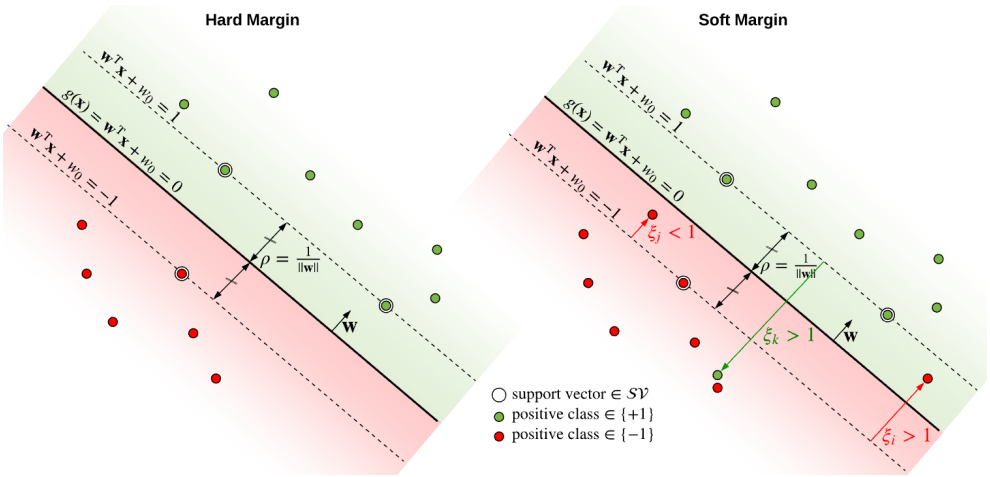

Fig. 2.2.: On the left side, a hard margin SVM is illustrated. Both classes are linearly separable. While on the right side, a soft margin SVM is used since the classes are not linearly separable. Some points will be missclassified if $\xi > 1$.

Finding the separating hyperplane of maximum margin reduces the risk of overfitting since it increases the power of generalisation by making the boundary decision more robust to data disturbances. Given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$ the class label, the separating hyperplane is defined as

$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = 0, \tag{2.18}$$

where $\mathbf{w} \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$ is the intercept. The decision rule associated to this hyperplane $f : \mathbb{R}^d \to \{-1, +1\}$ classifies a point according to the side of the hyperplane where it is located

$$f(\mathbf{x}) = \text{sign}(g(\mathbf{x})). \tag{2.19}$$

The geometrical margin ρ of the hyperplane is the distance between a support vector and the hyperplane. Hence it is given by

$$\rho = \min_{\mathbf{x}_i} \{\text{distance}\,(\mathbf{x}_i, g(\mathbf{x}) = 0)\} = \min_{\mathbf{x}_i} \left\{ \frac{|g(\mathbf{x}_i)|}{\|\mathbf{w}\|_2} \right\}. \tag{2.20}$$

Since the equation of the hyperplane is unchanged if we multiply it by a positive constant, we impose the following constraint $|g(\mathbf{x})| = 1$ if $\mathbf{x}$ is a support vector [10]. It allows the problem to be well-posed. Consequently, the margin becomes $\rho = \frac{1}{\|\mathbf{w}\|_2}$. Since we are interested in the maximum margin, the hyperplane sought, is the solution of the following problem

$$\left| \begin{array}{ll} \underset{\mathbf{w}, w_0}{\text{minimize}} & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} & y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + w_0) \geq 1 \quad \forall i = 1...n. \end{array} \right. \tag{2.21}$$

The margin constraints $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + w_0) \geq 1 \quad \forall i = 1...n$ impose that all training points are correctly classified and located at a distance from the hyperplane at least greater than the margin ρ.

However, if the training points are not linearly separable (overlapping between the two classes), there is no solution to (2.21). Hence, slack variables $\xi_i \geq 0$ are introduced to relax the margin constraints $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + w_0) \geq 1 - \xi_i$. Figure 2.2 (right side) shows that these new variables measure the discrepancy of the training samples with respect to the margin. If $\xi_i < 1$, the corresponding training sample is under the margin distance from the hyperplane but still correctly classified. On the other hand, if $\xi_i > 1$, the training sample is missclassified. This also allows the model to be less sensitive to outliers.

So, the problem (2.21) becomes a double objectives problem, maximising the margin but also minimising the margin error ($\sum_{i=1}^{n} \xi_i$),

$$\left| \begin{array}{ll} \underset{\mathbf{w},w_0}{\text{minimize}} & \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} & y_i(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i + w_0) \geq 1 - \xi_i \quad \forall i = 1...n \\ & \xi_i \geq 0 \quad \forall i = 1...n, \end{array} \right. \tag{2.22}$$

where the trade-off between these two objectives is monitored by a constant parameter C ($C \to \infty$ corresponds to the separable case (2.21)). The previous formulation has the advantage of being a convex quadratic optimisation problem. Hence, solving this problem always yields in a global optimum.

2.5.1 Generalisation to multi-class classification

We used a one-versus-all approach (or one-vs-rest) [46, 3] to be able to make multi-class predictions from the previous SVM model making only binary classification. Recall from Section 2.4 that we are interested in predicting among multiple class labels, so the SVM must learn a function $h : \mathbb{R}^d \to \mathcal{Y}$ where $\mathcal{Y} = \{1, ..., |\mathcal{Y}|\}$ is the set of classes. The one-versus-all method consists of training a binary classifier for each class, discriminating between training samples belonging to this class with the others. Each binary classifier $h_i : \mathbb{R}^d \to \{-1, +1\}$ is trained on the modified training set $S_i = \{\mathbf{x}_j, (-1)^{\mathbb{1}_{[y_j \neq i]}}\}_{j=1}^{n}$ and predicts $+1$ for a training sample belonging to its class and -1 in the contrary case. So the multi-class predictor appears to be

$$h(\mathbf{x}) = \underset{i \in \mathcal{Y}}{\arg\max}\, g_i(\mathbf{x}), \tag{2.23}$$

where $g_i(\mathbf{x})$ is the equation of the hyperplane (2.18) and plays the role of the confidence in the prediction. The further a data point is from the boundary decision on the positive side, the more it is considered likely to belong to the class of this classifier. This model can easily be applied to very large datasets since the number of parameters on which it depends is independent of the size of the training set. In addition, no matrix inversion is necessary. It is therefore said to be scalable.
The implementation of the SVM model used in this study is based on the LibLinear library in Python (see the documentation [12] for implementation details).

2.6 AutoSVM

The AutoSVM model [27] is an extension of the previous SVM model exploiting the graph information in addition to the features on the nodes. It involves explicitly

exploiting the autocorrelation hypothesis by taking into account autocovariates. These autocovariates are new variables built from the neighborhood of nodes and correlated with the target (class label). So they are very likely to improve the predictions of the SVM model when they are concatenated with the existing features. In this extension, there is one autocovariate per class and each of them is defined as the weighted average of the class-membership of the neighborhood of nodes. The autocovariate corresponding to class c is defined for all nodes as

$$\mathbf{ac}^c = \mathbf{P}\mathbf{y}^c, \tag{2.24}$$

where $\mathbf{P}$ is the transition probability matrix of the natural random walk (2.6) and $\mathbf{y}^c$ is the class-membership vector of class c. All autocovariates are gathered in a single matrix

$$\mathbf{Ac} = \begin{bmatrix} \mathbf{ac}^1 & \mathbf{ac}^2 & \dots & \mathbf{ac}^{|\mathcal{Y}|} \end{bmatrix} = \mathbf{PY}, \tag{2.25}$$

and injected by concatenation to the existing features $\mathbf{X}$ in the SVM. However, in practice, the autocovariates are not directly computable since we do not know all the class memberships $\mathbf{Y}$. A kind of expectation-maximization heuristic [8] is used to estimate both autocovariates (latent variables) and node labels. The iterative procedure is described as follows

1. At $t = 0$ (initialisation), a SVM model is trained on the features $\mathbf{X}$ only. This provides a first prediction for all nodes denoted by $\widehat{\mathbf{Y}}(0)$ (predictions of known labels are replaced by true labels).

2. The autocovariates are then estimated by computing

$$\widehat{\mathbf{Ac}}(t) = \mathbf{P}\widehat{\mathbf{Y}}(t), \tag{2.26}$$

   and are concatenated with the original features into a new SVM model. The feature matrix becomes $[\mathbf{X}, \widehat{\mathbf{Ac}}(t)]$. This model is trained and produced a new prediction of the node labels $\widehat{\mathbf{Y}}(t+1)$ (predictions of known labels are replaced by true labels).

3. Return to step 2 unless a fix-point is reached $\left(\widehat{\mathbf{Y}}(t+1) = \widehat{\mathbf{Y}}(t)\right)$ or that the maximum number of iterations is exceeded.

The implementation of this model is based on the SVM implementation described in the previous section.

2.7 Diffusion Convolutional Neural Network

The Diffusion Convolutional Neural Network (DCNN) introduced in [2] is a spatial-based ConvGNN (Subsection 2.3.2) due to its diffusion-convolution operation. As mentioned in Subsection 2.3.2, this model therefore benefits from the advantages of a spatial-based ConvGNN.

2.7.1 Architecture

The convolution defined in this model is based on a diffusion process in which the information on the nodes (features gathered in the matrix $\mathbf{X}$) is propagated by the edges more or less strongly according to the affinities (encoded in the matrix $\mathbf{A}$) between the nodes. Let the node $v_i \in \mathcal{V}$, the convolution aggregates the local information contained in its neighborhood as the expected value that a random walker starting at v_i would see if he made j hops (transitions) according to the transition probability matrix $\mathbf{P}$ defined in (2.6). The vector $\mathbf{e}_{ij} \in \mathbb{R}^d$ contains the expected value for each feature when j hops were made from the node v_i. So it is formally defined by the following weighted average

$$\mathbf{e}_{ij} = \sum_{l=1}^{|\mathcal{V}|} [\mathbf{P}^j]_{il} \mathbf{x}_l, \tag{2.27}$$

where $[\mathbf{P}^j]_{il}$ is the probability for the random walker of ending up at node v_l by making j hops and $\mathbf{x}_l \in \mathbb{R}^d$ is the feature vector associated with the node v_l. The information on the nodes close to v_i will therefore have a greater weight in the aggregation. Zero hop ($j = 0$) corresponds to no diffusion ($\mathbf{P}^0 = \mathbf{I}$ by convention) and the expected value is then simply the feature vector $\mathbf{x}_i$. The bigger is the number of hops and the more important will be the diffusion of features (the aggregation will apply to a larger neighborhood). The diffusion-convolution operation is made of $0, 1, 2, ..., H$ hops. Hence, the number of hops H is a hyperparameter to be optimised outside of the model training. The result for node v_i with all hops is stored in a matrix $\mathbf{E}_i \in \mathbb{R}^{(H+1)\times d}$,

$$\mathbf{E}_i = \begin{bmatrix} \mathbf{e}_{i0}^{\mathrm{T}} \\ \mathbf{e}_{i1}^{\mathrm{T}} \\ \vdots \\ \mathbf{e}_{iH}^{\mathrm{T}} \end{bmatrix}. \tag{2.28}$$

Once the aggregation is made, it is multiplied elementwise with a matrix $\mathbf{W}^c \in \mathbb{R}^{(H+1)\times d}$ containing learnable parameters and shared with all nodes (ensures that

the convolution operation is spatially invariant) and passed into a nonlinear function. To sum up, the node v_i is transformed by

$$\mathbf{Z}_i = \tanh\left(\mathbf{W}^c \odot \mathbf{E}_i\right), \tag{2.29}$$

to a diffusion-convolutional representation $\mathbf{Z}_i \in \mathbb{R}^{(H+1)\times d}$. The parameters contained in $\mathbf{W}^c$ allow the model to control the importance of features when they are diffused more or less strongly. The use of the hyperbolic tangent function in the model

$$\tanh(x) = 2\,\text{sigmoid}(2x) - 1 = \frac{2}{1+e^{-2x}} - 1 \tag{2.30}$$

allows to introduce non-linear capacities to the model by mapping non-linearly the values in the range $]-1,1[$. [28] indicates that the hyperbolic tangent function (centered sigmoid function) often converge faster than the sigmoid function (sigmoid(x)).
The node representation $\mathbf{Z}_i$ is then flattened and passed into a fully connected (dense) layer with a ReLu activation function to produce $|\mathcal{Y}|$ values. So, the layer can be expressed as

$$\mathbf{y}_i = \text{ReLu}(\mathbf{W}^d \text{vec}(\mathbf{Z}_i) + \mathbf{b}), \tag{2.31}$$

where $\mathbf{y}_i \in \mathbb{R}^{|\mathcal{Y}|}$, $\mathbf{W}^d \in \mathbb{R}^{|\mathcal{Y}|\times((H+1)d)}$ is a second matrix containing learnable parameters, $\mathbf{b} \in \mathbb{R}^{|\mathcal{Y}|}$ is a learnable vector containing the biases and the ReLu function refers to the Rectified Linear Unit function given by

$$\text{ReLu}(x) = \max(0, x). \tag{2.32}$$

The $|\mathcal{Y}|$ outputs taking values in the interval $[0, +\infty[$ are normalized to a probability distribution on class labels $\mathbf{o}_i \in \mathbb{R}^{|\mathcal{Y}|}$ $\left(\sum_{j=1}^{|\mathcal{Y}|} o_{ij} = 1 \text{ and } o_{ij} \geq 0\ \forall j = 1...|\mathcal{Y}|\right)$ by the softmax function

$$\mathbf{o}_i = \text{softmax}(\mathbf{y}_i), \tag{2.33}$$

where the softmax function is a vector function defined as

$$\text{softmax}(\mathbf{x}) = \frac{\exp(\mathbf{x})}{\mathbf{e}^{\mathrm{T}}\exp(\mathbf{x})}. \tag{2.34}$$

o_{ij} can be interpreted as the probability that the node v_i belongs to the class j. One of the interesting property of the DCNN is its scalability. Indeed, since the number of parameters is independent of the number of nodes, the model can be applied on very large datasets when the matrix $\mathbf{P}$ is sparse. The implementation of this model and its training is based on the Tensorflow library [33] in Python.

2.7.2 Loss function

The loss function used to compare the prediction with the real category is the multiclass hinge loss [51]. If the node v_i belongs to the class t, the discrepancy between the prediction and the true class is

$$l(\mathbf{o}_i, t) = 1 - o_{it} + \max_{j \neq t} o_{ij}. \tag{2.35}$$

Minimising this loss function consists in increasing the probability of the true class and in reducing the probability of the most violated class. The hinge loss is always positive and equals to zero when the prediction is perfect ($o_{it} = 1$).

2.7.3 Learning algorithm

A particular version of the stochastic gradient descend is used to minimise the objective function (2.8). The method is called Adam [17] and consists in an adaptive learning rate gradient descend method. It depends on the initial learning rate hyperparameter λ. Batches of data are used to approximate the gradient of the objective function (see Subsection 2.3.3 for the advantages of this approach). The size of a batch constitutes an additional hyperparameter B. Parameters contained in $\mathbf{W}^c$ are initialised according to a normal distribution with mean zero and standard deviation 0.01 ($[\mathbf{W}]^c_{ij} \sim \mathcal{N}(0, 0.01)$). On the other hand, parameters in the fully connected layer $\mathbf{W}^d$ are initialised according to a uniform distribution normalised by the number of input and output of the layer $\left([\mathbf{W}]^d_{ij} \sim U\left(-\sqrt{\frac{6}{(H+1)d+|\mathcal{Y}|}}, \sqrt{\frac{6}{(H+1)d+|\mathcal{Y}|}}\right)\right)$. Biases are set to zero. The last two initialisations are heuristics described in [17].

2.7.4 Early stopping

In order to avoid the overfitting, an early stopping procedure is used. This regularisation method [17] makes it possible to decide at which epoch, we can consider that the model has sufficiently learned the training set and generalises well to unknown data. More epochs would result in an higher precision on the training set but in a larger generalisation error (overfitting). It consists in stopping learning when the generalisation error no longer decreases and begins to increase. An additional set not used during training is used to estimate the generalisation error (called validation set). The early stopping procedure in [2] stops learning at a given epoch if the generalisation error estimated on the validation set is greater than the average of the generalisation errors on the previous few epochs. The number of previous epochs taken into consideration is an hyperparameter called "window size" W. This windowed early stopping procedure allows to temporarily degrade the precision on

the validation set to favor the exploration of the parameters space. The larger W is and the more the model can degrade its performance and explore.

2.8 Graph embedding based on kernels

The structure of the graph connecting data between them is not directly usable for traditional machine learning algorithms designed to learn from data expressed in an Euclidean space (generally, we assume the domain set $\mathcal{X} \subseteq \mathbb{R}^d$, see Section 2.2). One approach is to build a graph embedding based on the adjacency matrix $\mathbf{A}$ only. It consists in transforming the nodes and the information on edges into a new representation expressed in an Euclidean space. In order to constitute a relevant embedding allowing to improve the performances of models exploiting only the features $\mathbf{X}$, this new representation must preserve as much as possible the information concerning the structure of the graph.
This section details embedding techniques based on kernels extracted from the graph. A kernel on the graph [6, 14, 13] is a function $k : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ defining a similarity between each node. This pairwise similarity can be encoded in a single matrix (called kernel matrix) $\mathbf{K} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where $[\mathbf{K}]_{ij} = k(v_i, v_j),\ v_i, v_j \in \mathcal{V}$.
More specifically, we were interested in kernels coming from distance measures between nodes coming from the bag-of-paths framework (Subsection 2.8.1) and the margin-constrained bag-of-paths framework (Subsection 2.8.2). Both frameworks are built on the concept of random walks and the minimisation of the free-energy to handle the local and global structure of the network.
The next two subsections describe the concepts of distances between nodes defined by these two frameworks in order to understand their meaning and relevance when they are applied to semi-supervised classification tasks. Next, a subsection describes how we derived kernels from these distances. The following subsection discusses how new features expressed in a low-dimensional Euclidean space can be extracted from such kernels. Finally, the last subsection resumes all the matrices computed from the adjacency matrix $\mathbf{A}$ that are used later in this study.

2.8.1 The bag-of-paths framework

The bag-of-paths (BoP) framework [15, 14, 24, 19] is a probabilistic model defining a distance between node i and node j of a network based on the probability ($\mathrm{P}(s = i, e = j)$ called the "bag-of-paths probability") of drawing a path $\wp$ starting from the node i and ending to the node j among a bag of paths $\mathcal{P}$. In this study, we considered the bag of hitting paths $\mathcal{P}_{\mathrm{h}}$ containing all paths (even loops) whose ending node appears only once at the end of the path. In this consideration, the ending node is then made absorbing to prevent it from appearing in the path. The

bag-of-paths framework is defined for both hitting and non-hitting paths. So, for non-hitting (regular) paths, see the reference paper [15]. The model is applied on a weighted directed graph with an additional cost on each edge stored in the cost matrix $\mathbf{C}$ (2.3) and assumes that G is strongly connected. Recall from (2.4) that we selected a particular cost matrix. The model defines a reference distribution on paths $\{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)\}_{\wp\in\mathcal{P}_{\mathrm{h}}}$ based on the natural random walk (Subsection 2.1.2),

$$\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp) = \frac{\tilde{\pi}^{\mathrm{ref}}(\wp)}{\sum_{\wp'\in\mathcal{P}_{\mathrm{h}}} \tilde{\pi}^{\mathrm{ref}}(\wp')}, \tag{2.36}$$

in which

$$\tilde{\pi}^{\mathrm{ref}}(\wp) = \prod_{\tau=1}^{t} [\mathbf{P}^{\mathrm{ref}}]_{k_{\tau-1}k_\tau} \tag{2.37}$$

is the product of the transition probabilities along path $\wp$ defined by the natural random walk (2.6). So, $\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)$ represents the probability of following the path $\wp$ when moving from one node to another according to a reference transition probability proportional to the affinity between nodes given by the adjacency matrix $\mathbf{A}$.

The path probability distribution $\{\mathrm{P}(\wp)\}_{\wp\in\mathcal{P}_{\mathrm{h}}}$ of the bag-of-paths framework is then defined as the probability distribution minimising the expected total cost

$$\mathbb{E}[\tilde{c}(\wp)] = \sum_{\wp\in\mathcal{P}_{\mathrm{h}}} \mathrm{P}(\wp)\tilde{c}(\wp), \tag{2.38}$$

where $\tilde{c}(\wp)$ is the cost of the path $\wp$ defined in (2.5), under the constraint that the distribution does not differ too much from the reference distribution. The Kullback–Leibler divergence [3, 17, p. 55] is used in order to ensure this constraint,

$$D_{\mathrm{KL}}\left(\mathrm{P}(\wp)||\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)\right) = \sum_{\wp\in\mathcal{P}_{\mathrm{h}}} \mathrm{P}(\wp)\log\left(\frac{\mathrm{P}(\wp)}{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)}\right) = J_0. \tag{2.39}$$

Since the Kullback–Leibler divergence is a kind of measure of how two distributions diverge from each other, the parameter $J_0 > 0$ is used to monitor this difference. If $J_0 \to 0^+$, the path probability distribution $\{\mathrm{P}(\wp)\}_{\wp\in\mathcal{P}_{\mathrm{h}}}$ will tend towards the reference distribution (randomised) $\{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)\}_{\wp\in\mathcal{P}_{\mathrm{h}}}$, otherwise, when $J_0 \to \infty$, it will favor the probability path distribution minimising the total expected cost. The complete mathematical formulation is defined as follows,

$$\left|\begin{array}{ll} \underset{\{\mathrm{P}(\wp)\}_{\wp\in\mathcal{P}_{\mathrm{h}}}}{\text{minimize}} & \displaystyle\sum_{\wp\in\mathcal{P}_{\mathrm{h}}} \mathrm{P}(\wp)\tilde{c}(\wp) \\ \text{subject to} & \displaystyle\sum_{\wp\in\mathcal{P}_{\mathrm{h}}} \mathrm{P}(\wp)\log\left(\frac{\mathrm{P}(\wp)}{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)}\right) = J_0 \\ & \sum_{\wp\in\mathcal{P}_{\mathrm{h}}} \mathrm{P}(\wp) = 1. \end{array}\right. \tag{2.40}$$

The Kullback–Leibler divergence constraint is relaxed in the objective function with a Lagrange parameter $T > 0$. The relaxed formulation of the problem (2.40) is given by

$$\left| \begin{array}{ll} \underset{\{\mathrm{P}(\wp)\}_{\wp \in \mathcal{P}_\mathrm{h}}}{\text{minimize}} & \phi(\mathrm{P}) = \sum_{\wp \in \mathcal{P}_\mathrm{h}} \mathrm{P}(\wp)\tilde{c}(\wp) + T \sum_{\wp \in \mathcal{P}_\mathrm{h}} \mathrm{P}(\wp) \log \left(\dfrac{\mathrm{P}(\wp)}{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)} \right) \\ \text{subject to} & \sum_{\wp \in \mathcal{P}_\mathrm{h}} \mathrm{P}(\wp) = 1. \end{array} \right. \tag{2.41}$$

The new objective function $\phi(\mathrm{P})$ is called the "free-energy" and constitutes a trade-off between two objectives. On one side, the expected cost where high-cost paths have a low probability of occurring while less costly ones are more likely to occur. On the other side, the relative entropy between the path probability distribution and the reference distribution (randomised) allowing non-optimal paths in terms of costs. This trade-off is monitored by the temperature T playing the opposite role of the parameter J_0 in (2.40).
By solving the Lagrange relaxation of (2.41), we observe that the path probability distribution follows a Gibbs-Boltzmann distribution on the set of path $\mathcal{P}_\mathrm{h}$,

$$\mathrm{P}^*(\wp) = \frac{\tilde{\pi}^{\mathrm{ref}}(\wp) \exp(-\theta \tilde{c}(\wp))}{\sum_{\wp' \in \mathcal{P}_\mathrm{h}} \tilde{\pi}^{\mathrm{ref}}(\wp') \exp(-\theta \tilde{c}(\wp'))}, \tag{2.42}$$

where $\theta = \frac{1}{T}$ is defined as the inverse temperature.
From the equation (2.42), it is possible to derive $\mathrm{P}(s = i, e = j)$, the probability of drawing a path starting at node i and ending at node j. This probability is encoded at position (i,j) in the bag-of-paths probability matrix for hitting paths $\mathbf{\Pi}_\mathrm{h}$. This matrix reflects a notion of similarity between the nodes of the graph. Indeed, [15] suggests that two nodes are considered to be highly related when they are connected by many low-cost paths. This quantity therefore takes into account both a notion of local dependency (low-cost paths) and global dependency (randomness behavior introduced by the relative entropy with a random walk based on affinities). Its mathematical formulation is the following

$$[\mathbf{\Pi}_\mathrm{h}]_{ij} = \mathrm{P}(s = i, e = j) = \frac{\sum_{\wp \in \mathcal{P}_{ij}^\mathrm{h}} \mathrm{P}^*(\wp)}{\sum_{\wp' \in \mathcal{P}_\mathrm{h}} \mathrm{P}^*(\wp')}, \tag{2.43}$$

where $\mathcal{P}_{ij}^\mathrm{h}$ is the set of all hitting paths starting from i and ending in j. This matrix can be computed in closed form with the following equation (see the reference paper [15] for more details),

$$\mathbf{\Pi}_\mathrm{h} = \frac{\mathbf{Z}_\mathrm{h}}{\mathbf{e}^\mathrm{T} \mathbf{Z}_\mathrm{h} \mathbf{e}}, \text{ with } \mathbf{Z} = (\mathbf{I} - \mathbf{W})^{-1}, \mathbf{D}_\mathrm{h} = \mathbf{Diag}\,(\mathbf{Z}) \text{ and } \mathbf{Z}_\mathrm{h} = \mathbf{Z}\mathbf{D}_\mathrm{h}^{-1}, \tag{2.44}$$

where $\mathbf{Z}_\mathrm{h}$ is called the fundamental matrix for hitting paths and $\mathbf{W} = \mathbf{P}^{\mathrm{ref}} \circ \exp(-\theta \mathbf{C})$ is the weighted adjacency matrix.
From the similarity matrix (2.44), the reference paper [15] derives two distances between nodes, the surprisal distance

$$\boldsymbol{\Delta}_\mathrm{h}^{\mathrm{sur}} = \frac{-\log(\boldsymbol{\Pi}_\mathrm{h}) - \log(\boldsymbol{\Pi}_\mathrm{h}{}^{\mathrm{T}})}{2}, \tag{2.45}$$

and the free-energy distance (or potential distance)

$$\boldsymbol{\Delta}_\mathrm{h}^{\phi} = \frac{\boldsymbol{\Delta}^{\mathrm{sur}} - \log(\mathbf{e}^{\mathrm{T}}\mathbf{Z}_\mathrm{h}\mathbf{e})}{\theta}. \tag{2.46}$$

The diagonal is set to zero in both matrices. In our experiments, we decided to use the free-energy distance in the bag-of-paths framework and the surprisal distance in the margin-constrained bag-of-paths framework (Subsection 2.8.2). Both distances are very close since they only differs from an additional constant and rescaling factor.

2.8.2 The margin-constrained bag-of-paths framework

The margin-constrained bag-of-paths (cBoP) framework [20] extends the bag-of-paths framework by adding two additional constraints to the problem formulation (2.41) fixing the starting and ending nodes probability distributions of paths (margins). As in the BoP framework, we considered the bag of hitting paths $\mathcal{P}_\mathrm{h}$ and the same cost matrix (2.4). The new formulation is then given by

$$\left|\begin{array}{ll} \underset{\{\mathrm{P}(\wp)\}_{\wp\in\mathcal{P}_\mathrm{h}}}{\text{minimize}} & \phi(\mathrm{P}) = \sum_{\wp\in\mathcal{P}_\mathrm{h}} \mathrm{P}(\wp)\tilde{c}(\wp) + T\sum_{\wp\in\mathcal{P}_\mathrm{h}} \mathrm{P}(\wp)\log\left(\frac{\mathrm{P}(\wp)}{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)}\right) \\ \text{subject to} & \mathrm{P}(s=i) = \sum_{j\in\mathcal{V}}\sum_{\wp_{ij}\in\mathcal{P}_{ij}^\mathrm{h}} \mathrm{P}(\wp_{ij}) = \sigma_i^{\mathrm{in}} \qquad \forall i \in \mathcal{V} \\ & \mathrm{P}(e=j) = \sum_{i\in\mathcal{V}}\sum_{\wp_{ij}\in\mathcal{P}_{ij}^\mathrm{h}} \mathrm{P}(\wp_{ij}) = \sigma_j^{\mathrm{out}} \qquad \forall j \in \mathcal{V} \\ & \sum_{\wp\in\mathcal{P}_\mathrm{h}} \mathrm{P}(\wp) = 1, \end{array}\right. \tag{2.47}$$

where σ_i^{in} and σ_j^{out} are respectively the probability that a path begins at node i and the probability that a path ends at node j. Since these two constraints are independent, $\boldsymbol{\sigma}^{\mathrm{in}}$ and $\boldsymbol{\sigma}^{\mathrm{out}}$ must be two probability distributions on the nodes.
The reference paper [20] highlights that the solution of this new formulation corresponds to a randomised policy for the optimal transport problem on a graph. Indeed $\boldsymbol{\sigma}^{\mathrm{in}}$ can be seen as the supply and $\boldsymbol{\sigma}^{\mathrm{out}}$ as the demand, where resources must be routed to meet demand at minimum cost. The randomisation is again managed by the temperature parameter T allowing a trade-off between minimal cost when $T \to 0^+$ and entropy of paths based on the natural random walk when $T \to \infty$.
In order to be consistent with the new constraints in (2.47) when the temperature

parameter $T \to \infty$, the reference probability distribution on paths $\{\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp)\}_{\wp \in \mathcal{P}_{\mathrm{h}}}$ must also satisfy these additional constraints. However, the hitting paths benefit from the following nice property (proven in [15])

$$\sum_{\wp \in \mathcal{P}_{ij}^{\mathrm{h}}} \tilde{\pi}^{\mathrm{ref}}(\wp) = 1, \tag{2.48}$$

where $\tilde{\pi}^{\mathrm{ref}}(\wp)$ is defined in (2.37). Therefore, it follows that the reference probability distribution over paths and consistent with the two new additional constraints on the starting and ending nodes is given by

$$\tilde{\mathrm{P}}^{\mathrm{ref}}(\wp_{ij}) = \sigma_i^{\mathrm{in}} \sigma_j^{\mathrm{out}} \tilde{\pi}^{\mathrm{ref}}(\wp_{ij}). \tag{2.49}$$

By solving the Lagrangian relaxation of problem (2.47) while taking into account (2.49), it follows that the optimal path probabilities are given by

$$\mathrm{P}^*(\wp_{ij}) = \mu_i^{\mathrm{h,in}} \sigma_i^{\mathrm{in}} \mu_j^{\mathrm{h,out}} \sigma_j^{\mathrm{out}} \tilde{\pi}^{\mathrm{ref}}(\wp_{ij}) \exp(-\theta \tilde{c}(\wp_{ij})), \tag{2.50}$$

where $\theta = \frac{1}{T}$ is the inverse temperature and the vectors $\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}}$ and $\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}}$ are the Lagrange parameters. They satisfy the following system of mutually dependent equations

$$\begin{cases} \boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}} = \mathbf{e} \div \left(\mathbf{Z}_{\mathrm{h}} \left(\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}} \circ \boldsymbol{\sigma}^{\mathrm{out}}\right)\right) \\ \boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}} = \mathbf{e} \div \left(\mathbf{Z}_{\mathrm{h}}^{\mathrm{T}} \left(\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}} \circ \boldsymbol{\sigma}^{\mathrm{in}}\right)\right), \end{cases} \tag{2.51}$$

in which $\mathbf{Z}_{\mathrm{h}}$ refers to the fundamental matrix for hitting paths in (2.44). We computed them via the following iterative algorithm of type Gauss-Seidel [50]

$$\begin{cases} \boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}}(k) \leftarrow \mathbf{e} \div \left(\mathbf{Z}_{\mathrm{h}} \left(\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}}(k-1) \circ \boldsymbol{\sigma}^{\mathrm{out}}\right)\right) \\ \boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}}(k) \leftarrow \mathbf{e} \div \left(\mathbf{Z}_{\mathrm{h}}^{\mathrm{T}} \left(\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}}(k) \circ \boldsymbol{\sigma}^{\mathrm{in}}\right)\right), \end{cases} \tag{2.52}$$

starting at $\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}}(0) = \boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}}(0) = \mathbf{e}$. We iterated until the convergence criterion $\max_i |\mu_i^{\mathrm{h,out}}(k) - \mu_i^{\mathrm{h,out}}(k-1)| < 10^{-6}$ was reached or that the number of iterations has exceeded 10000.

The margin-constrained bag-of-paths probability matrix for hitting paths is defined as in (2.43) and its close form expression corresponds to

$$\boldsymbol{\Pi}_{\mathrm{h}} = \mathbf{Diag}\left(\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{in}} \circ \boldsymbol{\sigma}^{\mathrm{in}}\right) \mathbf{Z}_{\mathrm{h}} \mathbf{Diag}\left(\boldsymbol{\mu}_{\mathrm{h}}^{\mathrm{out}} \circ \boldsymbol{\sigma}^{\mathrm{out}}\right). \tag{2.53}$$

Concerning the cBoP framework, we considered only the surprisal distance derived from $\boldsymbol{\Pi}_{\mathrm{h}}$ via (2.45).

An advantage of this extended formulation is that it is possible to give more importance to certain nodes by adjusting the distributions $\boldsymbol{\sigma}^{\mathrm{in}}$ and $\boldsymbol{\sigma}^{\mathrm{out}}$. Nodes with high σ_i^{in} and σ_i^{out} will have a greater chance of being a starting or ending node. Therefore

their distances with the other nodes will decrease. We have chosen to favor the nodes having a high degree by imposing

$$\boldsymbol{\sigma}^{\text{in}} = \boldsymbol{\sigma}^{\text{out}} = \frac{\mathbf{A}\mathbf{e}}{\mathbf{e}^{\mathrm{T}}\mathbf{A}\mathbf{e}}, \tag{2.54}$$

where $\mathbf{A}$ is the graph adjacency matrix.

2.8.3 Converting a distance to a kernel

We explored two different approaches for mapping the previous distance matrices into kernel matrices containing pairwise similarities between the nodes. These two approaches have also been investigated in [15].

- Classical multidimensional scaling (MDS) [14, 13, 43, 1] allows to derive a centered kernel $\mathbf{K}$ ($\mathbf{K}\mathbf{e} = \mathbf{0}$) from a distance matrix $\boldsymbol{\Delta}$. It is given by

$$\mathbf{K} = -\frac{1}{2}\mathbf{H}\boldsymbol{\Delta}^{(2)}\mathbf{H}, \tag{2.55}$$

  where $\boldsymbol{\Delta}^{(2)}$ is the squared distance matrix (the power is taken element-wise) and $\mathbf{H} = \left(\mathbf{I} - \frac{\mathbf{e}\mathbf{e}^{\mathrm{T}}}{n}\right)$ is the centering matrix. The embedding space deduced from this kernel (in the next Subsection 2.8.4) has the particularity to preserve the distances $\boldsymbol{\Delta}$ between data ($||\mathbf{x}_i - \mathbf{x}_j||_2 = \boldsymbol{\Delta}_{ij}$).

- The Gaussian kernel (or radial basis function) [46, 3, 22] defines a similarity between nodes from a matrix of distances as follows

$$\mathbf{K} = \exp\left(-\frac{\boldsymbol{\Delta}^{(2)}}{2\sigma^2}\right), \tag{2.56}$$

  where σ is set to the variance of data. This kernel has the particularity that the inner product between two nodes expressed in the embedding space (see Subsection 2.8.4) is close to zero if the nodes are distant in $\boldsymbol{\Delta}$ and close to 1 otherwise.

2.8.4 Extracting features from a kernel

As briefly mentioned in the introduction to this section, a kernel on a graph is a function $k(v_i, v_j),\ v_i, v_j \in \mathcal{V}$ defining a similarity measure between nodes. It can be shown that if this similarity function is symmetric and semi-definite positive, it is implicitly equivalent to the inner product between two vector representations of nodes obtained from a particular mapping function $\phi : \mathcal{V} \to \mathbb{R}^{|\mathcal{V}|}$ [14, 13, 46]. This

mapping function converts each node $v_i \in \mathcal{V}$ into a vector $\phi(v_i) = \mathbf{x}_i$ called the node vector. In consequence,

$$k(v_i, v_j) = \phi(v_i)^{\mathrm{T}} \phi(v_j) = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j. \tag{2.57}$$

The matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_{|\mathcal{V}|}^{\mathrm{T}} \end{bmatrix} \tag{2.58}$$

contains all node vectors and is called the embedding matrix of the graph. To avoid confusion with the data matrix containing the features on nodes (2.61), the embedding matrix of the graph G (2.58) will be denoted as $\mathbf{X}^G$. According to (2.57), the kernel matrix is then given by

$$\mathbf{K} = \mathbf{X}^G \left(\mathbf{X}^G\right)^{\mathrm{T}}. \tag{2.59}$$

The following demonstrates this equivalence and how to deduce the embedding matrix $\mathbf{X}^G$. If the kernel matrix $\mathbf{K}$ is symmetric, linear algebra ensures that it exists a spectral decomposition of the matrix $\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}$ in which $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues of $\mathbf{K}$ sorted in decreasing order and $\mathbf{U}$ is an orthogonal matrix containing the corresponding eigenvectors of $\mathbf{K}$. Moreover, if $\mathbf{K}$ is positive semi-definite, it follows that

$$\mathbf{K} = \left(\mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}\right)\left(\mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}\right)^{\mathrm{T}}. \tag{2.60}$$

Hence, by combining (2.59) and (2.60), the embedding matrix is given by

$$\mathbf{X}^G = \mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}. \tag{2.61}$$

However, this graph embedding exists only if the kernel matrix $\mathbf{K}$ is symmetric and semi-definite positive. The notions of distances defined in the preceding subsections guarantee only the symmetric constraint. We used spectrum clip [6] on $\mathbf{K}$ to enforce the semi-definite positive constraint. The spectrum clip consists in putting all negative eigenvalues to zero. This approximation is made by computing $\mathbf{U}\widehat{\mathbf{\Lambda}}\mathbf{U}^{\mathrm{T}}$ where $\widehat{\mathbf{\Lambda}} = \max(\mathbf{\Lambda}, 0)$. It can be shown that the resulting matrix is the closest semi-definite matrix to $\mathbf{K}$ in terms of the Frobenius norm (see the proof in [14]).
Principal component analysis (PCA) [1, 3] can be used to reduce the dimensionality of the embedding space. The idea of PCA is to project the data points on several orthogonal axis maximising the variance of the projected data. However, [14, p. 399] shows that the embedding defined in (2.61) is already oriented in the directions of maximal variance. Since the variance of the projected data on an axis is proportional to the eigenvalues of $\mathbf{K}$, we kept only the r first columns of $\mathbf{X}^G$. To keep the

notations simple, the new embedding matrix is denoted by $\mathbf{X}^G \in \mathbb{R}^{|\mathcal{V}| \times r}$.
The choice of a low-dimensional embedding space is motivated by [27] in which better results are achieved when only a small number of dominant eigenvectors are kept. We fixed a general rules for all the investigated datasets, only 5% of the principal components axis are kept (corresponding to $r = \lfloor 0.05|\mathcal{V}| \rfloor$).

2.8.5 Summary

In order to avoid any confusion, Table 2.1 summarise all the matrices extracted from the adjacency matrix.

Acronym	*Param*	*Description*
$\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$	θ	Bag-of-paths probability matrix for hitting paths (2.44)
$\mathbf{\Pi}_{\text{h}}^{\text{cBoP}}$	θ	Margin-constrained bag-of-paths probability matrix for hitting paths (2.53)
$\mathbf{\Delta}_{\text{h}}^{\text{BoP}}$	θ	Free-energy distance between nodes computed with (2.46) from $\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$
$\mathbf{\Delta}_{\text{h}}^{\text{cBoP}}$	θ	Surprisal distance between nodes computed with (2.45) from $\mathbf{\Pi}_{\text{h}}^{\text{cBoP}}$
$\mathbf{K}_{\text{h,mds}}^{\text{BoP}}$	θ	MDS kernel computed with (2.55) from $\mathbf{\Delta}_{\text{h}}^{\text{BoP}}$
$\mathbf{K}_{\text{h,g}}^{\text{BoP}}$	θ	Gaussian kernel computed with (2.56) from $\mathbf{\Delta}_{\text{h}}^{\text{BoP}}$
$\mathbf{K}_{\text{h,mds}}^{\text{cBoP}}$	θ	MDS kernel computed with (2.55) from $\mathbf{\Delta}_{\text{h}}^{\text{cBoP}}$
$\mathbf{K}_{\text{h,g}}^{\text{cBoP}}$	θ	Gaussian kernel computed with (2.56) from $\mathbf{\Delta}_{\text{h}}^{\text{cBoP}}$
$\mathbf{X}_{\text{h,mds}}^{G,\text{BoP}}$	θ	Low-dimensional embedding computed with (2.61) from $\mathbf{K}_{\text{h,mds}}^{\text{BoP}}$
$\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$	θ	Low-dimensional embedding computed with (2.61) from $\mathbf{K}_{\text{h,g}}^{\text{BoP}}$
$\mathbf{X}_{\text{h,mds}}^{G,\text{cBoP}}$	θ	Low-dimensional embedding computed with (2.61) from $\mathbf{K}_{\text{h,mds}}^{\text{cBoP}}$
$\mathbf{X}_{\text{h,g}}^{G,\text{cBoP}}$	θ	Low-dimensional embedding computed with (2.61) from $\mathbf{K}_{\text{h,g}}^{\text{cBoP}}$

Tab. 2.1.: Summary of matrices extracted from the graph and used later in the study. All matrices depend on the inverse temperature parameter θ. The embedded space of the graph is made of $\lfloor 0.05|\mathcal{V}| \rfloor$ dimensions.

Figures 2.3, 2.4 and 2.5 illustrate how these embedding techniques are able to help the traditional machine learning algorithms on the node classification task when the dataset benefits from a high spatial autocorrelation. We applied these techniques on 3 graphs, two $\mathbf{A}$-driven graphs and one $\mathbf{X}$-driven graph. These graphs are presented in Section 4.1. Each figure is composed of two parts. On the left, the matrix $\mathbf{K}_{\text{h,g}}^{\text{BoP}}$ and on the right side, the mapping of nodes to the embedding space when only the two principal components (of maximum variance) of $\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$ are kept. We observe on Figures 2.3 and 2.4 that the kernel contains high similarities between nodes belonging to the same class. Such a kernel is therefore much more informative than the adjacency matrix $\mathbf{A}$ which contains only local relationships. Hence, nodes with similar labels appear close to each other and form clusters in the embedding space. These clusters help the machine learning algorithms to discriminate the classes. However, in Figure 2.5, nodes with identical labels are not particularly similar to each other. Therefore, it is no longer informative and does not help to discriminate between classes.

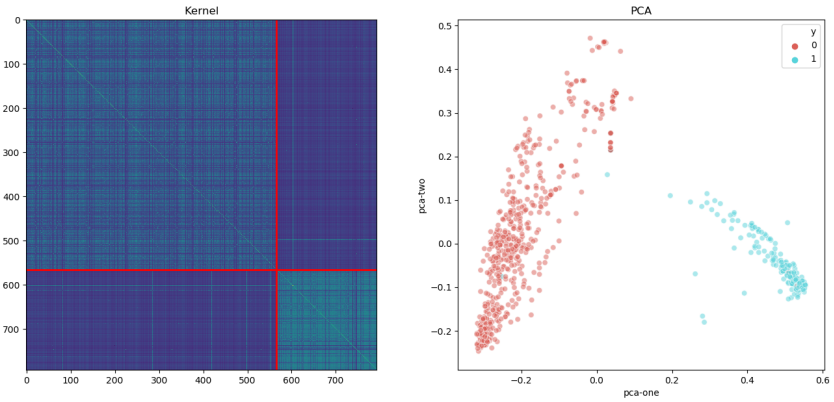

Fig. 2.3.: On the left side, the matrix $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}$ with $\theta = 0.1$ computed on *DB6* (dataset with high spatial autocorrelation). Nodes are ordered according to their label. The red lines separate the different classes. On the right side, the associated graph embedding $\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$ when only the two principal components are kept.

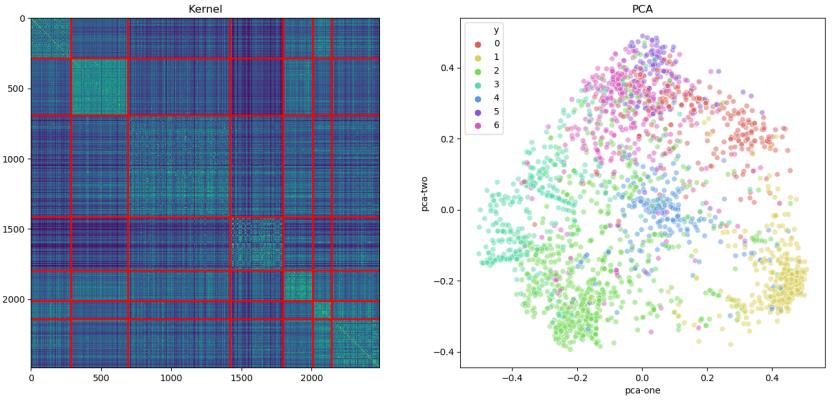

Fig. 2.4.: On the left side, the matrix $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}$ with $\theta = 0.1$ computed on *DB9* (dataset with high spatial autocorrelation). Nodes are ordered according to their label. The red lines separate the different classes. On the right side, the associated graph embedding $\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$ when only the two principal components are kept.

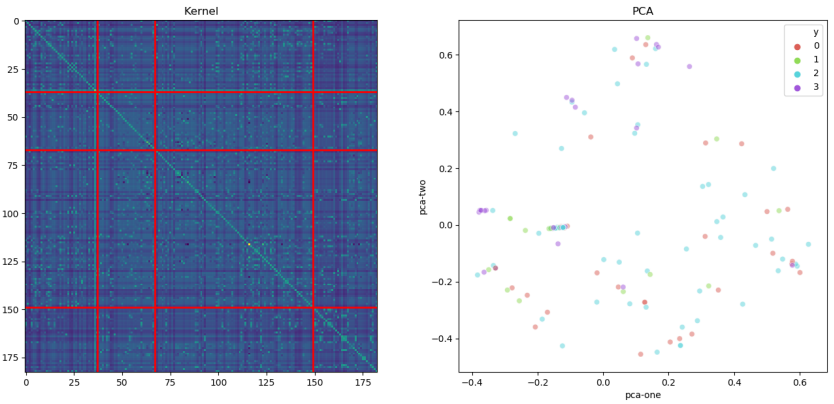

Fig. 2.5.: On the left side, the matrix $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}$ with $\theta = 0.1$ computed on *DB1* (dataset without spatial autocorrelation). Nodes are ordered according to their label. The red lines separate the different classes. On the right side, the associated graph embedding $\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$ when only the two principal components are kept.

2.9 Ranking method and statistical tests

Chapter 5 is devoted to comparing the methods presented in the next chapter. To highlight differences in performance between methods, it is necessary to evaluate them on several datasets. Chapter 4 details the methodology used to obtain the accuracy of each method on each dataset. From these accuracies, a ranking method is used to classify the investigated methods and a statistical test to confirm if significant differences are apparent.

2.9.1 Borda count

The Borda count method [11, 26] allows to rank all methods according to their performances over the different datasets based on a voting scheme. Each method can be seen as a candidate and each dataset as a voter. Therefore, the accuracy of a method on a dataset can be seen as the interest that the voter has for the candidate. Let n the number of candidates and m the number of voters. Each voter assigns points to each candidate according to the following voting scheme. The voter orders the candidates according to the interest he has in them. Then, the first candidate in this ranking receives n points, the second receives $n-1$ points, ..., and the last receives 1 point. However, in his personal ranking, a voter may have the same interest for several candidates. In this specific case, there are several possibilities to react [52]. We decided to use a fractional ranking. It consists of giving to each of the candidates concerned, the average of points that the voter should normally attribute to all of them. This avoids favoring or disadvantaging candidates and does not disturb the points obtained by candidates that are not concerned. The sum of points that a candidate received from the m voters, constitutes its global score. The final ranking of the candidates is obtained by ordering the candidates according to their global score.

2.9.2 Wilcoxon signed-rank test

The Wilcoxon signed-rank test [55] is a non-parametric statistical test allowing to compare paired data. The Wilcoxon signed-rank test will be used later between each pair of methods. Hence, the paired samples correspond to (x_i, y_i) where x_i and y_i are respectively the accuracy of the first and second method on the dataset i. It tests the null hypothesis (H_0) under which the distribution of the difference $z = x - y$ is symmetric around a median $\theta = 0$. We used the two-tailed (two-sided) version in order to avoid missing an effect in one of two directions [42].
The procedure consists in ordering the pair in the increasing order of their absolute value difference $|x_i - y_i|$. Pairs whose $|x_i - y_i| = 0$ are discarded. A rank R_i is

assigned to each of the n_r remaining pairs. A value of 1 for the first pair, 2 for the second pair, etc. Pairs having identical absolute difference are managed in the same way as in the previous Borda method with a fractional ranking. The sum of the signed ranks [54] is given by

$$S_r = \sum_{i=1}^{n_r} \text{sign}(|x_i - y_i|) R_i. \tag{2.62}$$

Under the null hypothesis, S_r follows a specific distribution of zero mean. We compute the p-value corresponding to the observed S_r. The p-value is equal to the probability of observing an even more unexpected result under the null hypothesis. The significance level is set to 5% in our experiments. So, a p-value lower or equal to 0.05 will be considered as so unexpected that we can conclude that the null hypothesis is false, and that a significant difference between the two methods seems to exist. If the p-value is greater than 5%, nothing can be established. Since the number of samples in our experiments is small (10 datasets) and below the recommended number (≈ 20), exact tables are used to find the critical values instead of the normal distribution (for more details, see the scipy.stats.wilcoxon function in the SciPy library [49] in Python).

2.10 Some further related work

There are many algorithms developed to solve the task presented in Section 2.4. These models are the result of various approaches. In this section, we cite a few of them.

The authors of [13] investigated a large number of different kernels in a simple similarity-based approach. This approach allowed to highlight which kernels enhance the class membership the most. Among all the explored kernels, the Regularised Laplacian Kernel (RL) [14] and the Regularised Commute-Time Kernel (RCT) [14] outperformed the others. Hence, we added them in our experiments. However, they provided significantly less powerful results than those associated to kernels coming from the BoP and cBoP frameworks. That is why, for clarity, we removed it from the main discussion. Their results are reported in Appendix A.1.

In our work, the SVM model is able to take into account the graph structure thanks to the addition of a graph embedding in the feature set. However, another extension of the traditional SVM model has been proposed. The LapSVM model [36] exploits the graph structure by adding a manifold regulariser in the objective function. The Laplacian regularisation term is used to impose that the distribution of labels is

smooth on the structure of the graph (a graph embedding can also be added in the feature set).

In [45], we can find a generic classification algorithm (GCA) consisting in applying iteratively to each unlabelled node, a local classifier exploiting the features and labels of nodes in the immediate neighborhood of the node (if a neighboring node does not yet have a label predicted, it is not used in the prediction). This process stops when the convergence of labels or a maximum number of iterations is reached. A simplified Gibbs Sampling (GS) procedure is also details for making label inference on the unlabelled nodes based on GCA. GCA and GS are both generic in the sense that any local classifier can be used. The paper resumes some of the most used local classifiers and states some adaptations which can lead to better accuracy. Two global objective functions to the node classification task are also described (Loopy Belief Propagation and Mean-field relaxation labeling). Both define a probability distribution of labels on the unlabelled nodes given the information on the labelled nodes and are approximations of the probability distribution associated to a pairwise Markov random field (pairwise MRF) on the graph. Pairwise means that MRF is defined on clique potential of one or two nodes (involving local consistency). These approximations make it possible to more efficiently calculate the marginal probability of predicting a given label on an unlabeled node. Pairwise MRF [25] allows to handle dependencies or independencies that Bayesian network can not since the last is limited to acyclic dependencies.

As explained in Subsection 2.3.2, spectral-based ConvGNNs suffers from high computational cost. However, a spectral-based ConvGNN which is scalable in terms of the number of nodes and computationally efficient is proposed in [23]. It avoids the eigendecomposition of the Laplacian matrix involved in the spectral convolution by approximating the learnable filter by a localised first-order Chebyshev polynomial. This linear approximation can be compensated by stacking several of these convolutional layers. Moreover, in [5], it is shown that [23] can be improved by introducing a well chosen sampling scheme in the reformulation of the loss and the gradient. However, this approach focus more on an inductive learning. We suggest the two following comprehensive surveys [57, 56] to get a better vision of what is developing in the field of deep learning.

Other deep learning approaches have been explored. In particular, the DeepWalk algorithm [40, 34]. First, DeepWalk samples paths on the graph by making short random walks. These paths are a succession of nodes and can be considered as sentences where nodes are words. Then, the well known Skip-Gram model [37] designed in the field of Natural Language Processing (NLP) is used to learn from these sentences a word (node) embedding in a continuous space. This approach is completely unsupervised. More advanced version was proposed as the node2vec

model [18] in which the path generator allows to better control the randomness in the generated paths to take into account the weight on the edges. Then, this graph embedding can be injected as new features in any classification machine learning algorithms. See [4] for a comprehensive survey of graph embedding methods and [31] for a global overview of a wide variety of neural networks proposed for learning low-dimensional graph representations.

3 Investigated algorithms

All the investigated models are precisely defined in this chapter. They result from adaptations and modifications of the models presented in Chapter 2 and grouped accordingly by the type of model. Section 3.1 is dedicated for a trivial model, Section 3.2 for the models inspired from the SVM and Section 3.3 for those based on the DCNN. Furthermore, Section 3.4 contains a summary of models with their acronym, their list of hyperparameters and a short description.

3.1 Trivial method

We implemented one of the simplest methods to predict labels. This method does not exploit the structure of the graph and the features on the nodes but uses only the proportion of classes in the training set. It predicts the label corresponding to the majority class (the one with the highest proportion in the training set) on all unlabelled nodes. By assuming that the observed proportions are representative of the real proportions, the precision on an unlabeled node will be close to the proportion of the majority class. We consider this basic method to be the simplest baseline that all of the other more sophisticated models studied must at least improve. It will be denoted by Trivial.

3.2 SVM-based methods

We tested several models based on the traditional SVM model. They are listed below.

- We included in our experiments the SVM model (Section 2.5) using only the features on the nodes as a more sophisticated baseline. It is denoted by SVM-$\mathbf{X}$. However, this model depends on an external parameter (hyperparameter) which is not optimised during the learning. This hyperparameter is the relaxation constant C monitoring the trade-off between a hard and soft margin.

- We extended the previous SVM model to take into account the graph information. We decided to make this extension by concatenating a low-dimensional embedding of the graph in Table 2.1 to the feature matrix. The feature matrix becomes $[\mathbf{X}, \mathbf{X}^G]$. This extension is denoted by SVM-$\mathbf{X}$-$\mathbf{X}^G$. We used the four embedding matrices in Table 2.1 coming from the BoP and cBoP frameworks, so $\mathbf{X}^G \in \{\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}, \mathbf{X}^{G,\text{BoP}}_{\text{h,g}}, \mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}, \mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}\}$. All these extended versions of the traditional SVM, now, include two hyperparameters instead of one. The hyperparameter C as before and in addition the inverse temperature θ associated to the graph embedding.

- The AutoSVM described in Section 2.6 builds a succession of SVMs by adding autocovariates in the feature space. It depends on the same hyperparameter as the SVM-$\mathbf{X}$. It is recognised by the acronym Auto-SVM-$\mathbf{X}$.

3.3 DCNN-based methods

We explored different versions of the DCNN detailed in Section 2.7. They are all described below.
One of the disadvantages of deep learning models is that they generally depend on a large number of hyperparameters. To overcome this problem, we reduced their number by fixing certain hyperparameters to a particular value. For example, based on a rapid study on a dataset, we imposed the window size W to be equal to 10 (wide enough to prevent learning from stopping too quickly) and defined the batch size B according to a rule common to all datasets : $B = 2^{\left\lfloor \log_2\left(\left\lfloor \frac{|S|}{10} \right\rfloor\right)\right\rfloor}$, i.e: the closest power of 2 below the tenth of the training set size.

- First, we considered the DCNN model without diffusion. So with a hop number equals to zero ($H = 0$). It will be compared to the SVM-$\mathbf{X}$ since like him, it only exploits the features (the graph is not exploited). It only depends on one hyperparameter, the initial learning rate λ used by the learning algorithm. This model is denoted by DCNN-$\mathbf{X}$.

- Then, comes the original DCNN model presented in Section 2.7 exploiting the structure of the graph via a diffusion process controlled by the number of hops H. This number therefore constitutes in addition to λ, a second hyperparameter. The model is denoted by DCNN-$\mathbf{X}$-$\mathbf{P}$.

- We decided to envisage a modification of DCNN-$\mathbf{X}$-$\mathbf{P}$ in which the diffusion is governed by a global similarity matrix $\mathbf{G}$ between the nodes instead of the local similarity matrix $\mathbf{P}$. The matrix $\mathbf{G}$ is normalised to be a stochastic

matrix as the matrix $\mathbf{P}$. The following transformation is the same as (2.6) and enforces that the lines sum to one

$$\bar{\mathbf{G}} = \mathbf{Diag}\left(\mathbf{Ge}\right)^{-1}\mathbf{G}. \tag{3.1}$$

To be a stochastic matrix, it is required that the values of $\mathbf{G}$ are all positive or zero. Due to this last constraint, we decided to select $\mathbf{G}$ among some matrices defined in Table 2.1, i.e: $\mathbf{G} \in \{\mathbf{\Pi}_{\text{h}}^{\text{BoP}}, \mathbf{\Pi}_{\text{h}}^{\text{cBoP}}, \mathbf{K}_{\text{h,g}}^{\text{BoP}}, \mathbf{K}_{\text{h,g}}^{\text{cBoP}}\}$. The MDS kernels do not guarantee that the similarities are positive or zero. As we mentioned in Subsection 2.8.1 and 2.8.2, these matrices are able to capture global relationships and are controlled by the inverse temperature θ. Since such a matrix $\mathbf{G}$ captures global relationships between the nodes, taking its powers does not make sense. This is why in this version, we considered $H = 1$. The diffusion for $j \in \{0, 1\}$ hops around node v_i defined in (2.27) becomes

$$\mathbf{e}_{ij} = \sum_{l=1}^{|\mathcal{V}|} [\bar{\mathbf{G}}^j]_{il}\mathbf{x}_l, \tag{3.2}$$

and the result for all hops (2.28) is restricted to

$$\mathbf{E}_i = \begin{bmatrix} \mathbf{e}_{i0}^{\text{T}} \\ \mathbf{e}_{i1}^{\text{T}} \end{bmatrix}. \tag{3.3}$$

This new model therefore depends on two hyperparameters, the initial learning rate λ and the inverse temperature θ. It is denoted by DCNN-$\mathbf{X}$-$\mathbf{G}$. The results of this model are not reported in the main discussion because they did not significantly improve the performance of DCNN-$\mathbf{X}$-$\mathbf{P}$. This analysis is available in Appendix A.2. Despite these results, this model constitutes the basis for the following model which tries to overcome the limit on the number of hops ($H = 1$) reducing the power of the model.

- To correct the problem described in the previous model, we had the idea of weighting the global similarity $[\mathbf{G}]_{il}$ between the nodes v_i and v_l by the probability $[\mathbf{P}^j]_{il}$, corresponding to the probability for the natural random walker to arrive at node v_l from v_i by performing j hops. The weighting is made with the following elementwise product $\mathbf{P}^j \circ \mathbf{G}$. It allows to produce a matrix containing local similarities, whose the size of the locality is controlled by the number of hops j. Then, the resulting matrix is normalised as in (3.1) to be a stochastic matrix. Once again, we chose $\mathbf{G} \in \{\mathbf{\Pi}_{\text{h}}^{\text{BoP}}, \mathbf{\Pi}_{\text{h}}^{\text{cBoP}}, \mathbf{K}_{\text{h,g}}^{\text{BoP}}, \mathbf{K}_{\text{h,g}}^{\text{cBoP}}\}$

to satisfy the constraint on $\mathbf{G}$ defined in the previous model. In other words, the diffusion of features on the nodes around the node v_i is now given by

$$\mathbf{e}_{ij} = \sum_{l=1}^{|\mathcal{V}|} [\overline{\mathbf{P}^j \circ \mathbf{G}}]_{il} \mathbf{x}_l. \tag{3.4}$$

In this formulation, the number of hops is no longer restricted. Consequently, H becomes again a hyperparameter in addition to the initial learning rate λ and the inverse temperature θ. The model is denoted by DCNN-$\mathbf{X}$-$\mathbf{PG}$.

- We also inspected other changes to the DCNN-$\mathbf{X}$-$\mathbf{P}$. In particular, we added in the objective function a regularisation term imposing that the solution provided by the model must include a kind of spatial autocorrelation. The loss function (2.35) is transformed into

$$l(\mathbf{o}_i, t) = \underbrace{1 - o_{it} + \max_{j \neq t} o_{ij}}_{\text{Multiclass hinge loss}} + \underbrace{\alpha ||\mathbf{e}_i^{\mathrm{T}} \mathbf{P} \mathbf{O}^{\text{prev}} - \mathbf{o}_i^{\mathrm{T}}||}_{\text{Regularisation}}, \tag{3.5}$$

where $\alpha \geq 0$ is a hyperparameter allowing to regulate the importance of the regularisation in the total cost function ($\alpha = 0$ corresponds to no regularisation), $||.||$ is any norm, $\mathbf{P}$ is the transition probability matrix defined by the natural random walk (2.6) and $\mathbf{O}^{\text{prev}} \in \mathbb{R}^{|V| \times |\mathcal{Y}|}$ is a matrix containing the prediction at the end of the previous epoch of class memberships of each node on each of its lines. To sum up,

$$\mathbf{O}^{\text{prev}} = \begin{bmatrix} \mathbf{o}_0^{\mathrm{T}} \\ \mathbf{o}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{o}_{|\mathcal{V}|-1}^{\mathrm{T}} \end{bmatrix}, \tag{3.6}$$

where the predictions $\mathbf{o}_i$ were calculated at the previous epoch. The vector $\mathbf{e}_i^{\mathrm{T}} \mathbf{P} \mathbf{O}^{\text{prev}}$ measures a sort of distribution of labels in the close neighborhood of the node v_i in which the contribution of nodes is weighted by their affinity with the node v_i. Thus, a node having a great affinity with v_i will have a greater weight in the distribution. Therefore, the regularisation part minimises the discrepancy between the distribution of labels on the node v_i and the distribution of labels in its neighborhood. This forces a node to resemble to its neighborhood and imposes a form of local consistency in the solutions proposed by the model. The hinge loss is still there to reduce the error on the training set. Regularisation [17] can help the model to generalise and reduce the risk of overfitting. We chosen to compare two different norms, the L_1-norm $||.||_1$ and the square of the L_2-norm $||.||_2^2$. However, during the first epoch, $\mathbf{O}^{\text{prev}}$ is not defined. We tried several models to initialise the first

prediction $\mathbf{O}^{\text{prev}}$. Namely the Trivial, SVM-$\mathbf{X}$ and Auto-SVM-$\mathbf{X}$ models. Both with $||.||_1$ and $||.||_2^2$, we found through experiments that the efficiency did not significantly change according to the initialisation used. Moreover, the square of the L_2-norm presented slightly more satisfactory results than the L_1-norm. These observations are discussed in Appendix A.4 for clarity reasons. So, for the rest of the experiments, only the regularisation with the square of the L_2-norm and an initialisation based on the Trivial model is compared to the other models.
The model is denoted by Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$ and depends on three hyperparameters, namely, the initial learning rate λ, the number of hops H and the weight of the regularisation α.

- In a continuity, we added the regularisation described in the previous model in the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$. This model is identified by Reg-DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ and depends on 4 hyperparameters, the initial learning rate λ, the number of hops H, the inverse temperature θ and the weight of the regularisation α.

- The last attempt to modify the DCNN-$\mathbf{X}$-$\mathbf{P}$ was inspired by the Auto-SVM-$\mathbf{X}$. The idea was to inject as additional features, the autocovariates defined in (2.26). These autocovariates are updated at the end of each epoch. As for the Auto-SVM-$\mathbf{X}$, an initialisation of labels is needed to compute the first autocovariates. We investigated an initialisation based on the Trivial, SVM-$\mathbf{X}$ and Auto-SVM-$\mathbf{X}$ models.
  The new model is denoted by Auto-DCNN-$\mathbf{X}$-$\mathbf{P}$ and is controlled by 2 hyperparameters, the initial learning rate λ and the number of hops H. However, the experiences in Appendix A.5 highlight that the performances obtained were significantly worse than that of DCNN-$\mathbf{X}$-$\mathbf{P}$. Hence, this model was not compared to the others.

3.4 Summary

Table 3.1 lists all the models discussed in the Chapter 5. For each of them, the table summarises its acronym, its list of hyperparameters with the values tested and a brief description. The table is divided into 3 parts, the Trivial model, the SVM-based models and the DCNN-based models.

Acronym	*Param*	*Values*	*Description*
Trivial	-	-	Trivial model predicting the majority class
SVM-$\mathbf{X}$	C	$10^{[-4,-3,2,0,2,3,4]}$	SVM on features only
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	C θ	$10^{[-4,-3,2,0,2,3,4]}$ $10^{[-3,-2,-1,0]}$	SVM with additional features $\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	C θ	$10^{[-4,-3,2,0,2,3,4]}$ $10^{[-3,-2,-1,0]}$	SVM with additional features $\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	C θ	$10^{[-4,-3,2,0,2,3,4]}$ $10^{[-3,-2,-1,0]}$	SVM with additional features $\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	C θ	$10^{[-4,-3,2,0,2,3,4]}$ $10^{[-3,-2,-1,0]}$	SVM with additional features $\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$
Auto-SVM-$\mathbf{X}$	C	$10^{[-4,-3,2,0,2,3,4]}$	SVM with autocovariates as additional features
DCNN-$\mathbf{X}$	λ	$10^{[-3,-2]}$	DCNN with 0 hops ($H=0$)
DCNN-$\mathbf{X}$-$\mathbf{P}$	λ H	$10^{[-3,-2]}$ $1,2,3$	Original DCNN
DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$	λ H θ	$10^{[-3,-2]}$ $1,2,3$ $10^{[-3,-2,-1,0]}$	DCNN with a modified diffusion based on $\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$
DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{cBoP}}_{\text{h}}$	λ H θ	$10^{[-3,-2]}$ $1,2,3$ $10^{[-3,-2,-1,0]}$	DCNN with a modified diffusion based on $\mathbf{\Pi}^{\text{cBoP}}_{\text{h}}$
DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{K}^{\text{BoP}}_{\text{h,g}}$	λ H θ	$10^{[-3,-2]}$ $1,2,3$ $10^{[-3,-2,-1,0]}$	DCNN with a modified diffusion based on $\mathbf{K}^{\text{BoP}}_{\text{h,g}}$
DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{K}^{\text{cBoP}}_{\text{h,g}}$	λ H θ	$10^{[-3,-2]}$ $1,2,3$ $10^{[-3,-2,-1,0]}$	DCNN with a modified diffusion based on $\mathbf{K}^{\text{cBoP}}_{\text{h,g}}$
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	λ H α	$10^{[-3,-2]}$ $1,2,3$ $0.1,1,2,10$	DCNN-$\mathbf{X}$-$\mathbf{P}$ with a regularisation ($\|\|.\|\|_2^2$) in the loss function enforcing a local consistency
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$	λ H θ α	$10^{[-3,-2]}$ $1,2,3$ $10^{[-3,-2,-1,0]}$ $0.1,1,2,10$	DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$ with a regularisation ($\|\|.\|\|_2^2$) in the loss function enforcing a local consistency

Tab. 3.1.: Summary of all the models discussed in Chapter 5 with their acronym, their list of hyperparameters and a short description. C controls the trade-off between the hard and soft margin in the SVM model, θ refers to the inverse temperature involved in the graph embedding coming from the BoP and cBoP frameworks, λ corresponds to the initial learning rate in the DCNN, H is the number of hops in the DCNN and α monitors the trade-off between the hinge loss and the regularisation favoring solutions with local consistency in the DCNN.

4 Experimental methodology

The previous chapter presented all the methods investigated in our study. The purpose of this chapter is to detail the complete methodology used in order to evaluate their expected accuracy on unlabelled nodes. The evaluation technique used is the same as that implemented in [15, 19, 27, 34] with some small variations and adaptations which are described below. The classifiers resumed in Table 3.1 are evaluated on several well known benchmark datasets (Section 4.1) through an evaluation process (Section 4.2) allowing both to optimise the hyperparameters and to obtain a more reliable performance estimate.

4.1 Datasets

Our experiences have focused on 10 different datasets that have already been used in a previous work [27].

- **WebKB** (*DB1-DB4*) : These four datasets contain web pages collected from computer science departments of four universities. *DB1* was collected from Cornell, *DB2* from Texas, *DB3* from Washington and *DB4* from Wisconsin. Each web page is encoded into a binary feature vector indicating the presence/absence of each word of a given dictionary in the web page. The dictionary includes 1703 words. The relations between the web pages represent citations (corresponding to asymmetric relations). If the edge $i \rightarrow j \in \mathcal{E}$, it means that the page j is cited by the page i. The associated similarity in the adjacency matrix is then established at $a_{ij} = 1$ (the absence of a citation corresponds to $a_{ij} = 0$). Each page is classified among 4 different classes : course, faculty, student and project. The category staff was merged with the category project because it contained only few pages. These datasets were also used in [15, 19, 34].

- **Ego Facebook** (*DB5-DB7*) : These datasets are three ego networks coming from users of Facebook. *DB5* corresponds to FB107, *DB6* to FB1684 and *DB7* to FB1912. The nodes of such a network are the friends of a given user called the "ego" and the edges are the friendship relationships (the ego is not included in the structure). Since a friendship relationship is symmetrical, they are modeled in the adjacency matrix as follows : if user i and j are friends,

$a_{ij} = a_{ji} = 1$ otherwise $a_{ij} = a_{ji} = 0$. Each user is characterised by a feature vector encoding his profile. While the task defined in [30] consists of inferring a given number of social circles through an unsupervised approach, we are interested here in a simplified version and adapted to the task that concerns us. A social circle is a subset of users sharing common properties with respect to the ego user, for example: family, colleagues, sports teammates, etc. The datasets have been adapted for a semi-supervised task where it only remains two social circles. The first being the largest circle and the second obtained by merging all the other circles. Therefore, the goal is to classify a user in one of these two circles.

- **CiteSeer** (*DB8*) : This dataset is like the WebKB datasets, a citation network. It consists of 3312 scientific publications encoded with a vocabulary of 3703 words. The objective is to classify each publication among six possible topics. It was also used in [45, 23].

- **Cora** (*DB9*) : This dataset is still a citation network. It is made up of 2708 scientific publications encoded with a vocabulary of 1434 words. The objective is to classify each publication among seven possible topics. It was also used in [45, 2, 23].

- **Wikipedia** (*DB10*) : This dataset consists in a citation network including 3271 Wikipedia articles that appeared in the featured list (best articles of Wikipedia) in the period Oct. 7-21, 2009. Each of them is encoded using a tf/idf-weighted feature vector based on a dictionary of 4973 words. We are once again interested in predicting in which category each page belongs to. There are a total of 14 categories characterising the topic of the article.

4.1.1 Pre-processing step

A common pre-processing step was applied to each of the datasets.

- Firstly, we kept only the biggest strongly connected component of each graph. This avoids infinite distance between two nodes in the calculation of kernels.

- Secondly, we made symmetric all the relations between the nodes by using (2.2) in order to get an undirected graph.

- Thirdly, the diagonal of the adjacency matrix was set to zero, so that any self-loops disappear.

- Fourth, we applied a feature selection by selecting the 100 most relevant features to predict the target (class label) according to a Chi-square test (χ^2) [35]. This maximum-relevance selection improves the ability of the model to generalise by reducing the risk of overfitting (Subsection 2.2.2) since less information is given to the model.
  The feature matrix (2.10) can be rewritten according to its columns

$$\mathbf{X} = \begin{bmatrix} \mathbf{f}_1 & \mathbf{f}_2 & \dots & \mathbf{f}_d \end{bmatrix}, \tag{4.1}$$

  where each vector $\mathbf{f}_i \in \mathbb{R}^{|\mathcal{V}|}$ corresponds to one feature measured on all nodes. Moreover the class membership of nodes encoded in the matrix (2.11) can be encoded in a single categorical vector $\mathbf{t} \in \{1, ..., |\mathcal{Y}|\}^{|\mathcal{V}|}$.
  Since all the datasets considered in our study are composed of categorical features, a Chi-square test was applied between each pair $(\mathbf{f}_i, \mathbf{t})$ allowing to decide if these two categorical variables are correlated. It tests the null hypothesis H_0 under which the variables are uncorrelated (absence of relation). The features have been sorted in ascending order of their p-value obtained with the statistical test. The p-value represents the probability of observing an even more extreme value than the one already observed under the null hypothesis. Hence, it provides an estimate of the confidence that we can have when we assert that the feature is correlated to the target. Therefore, we selected the first 100 features for which this confidence was the highest (smaller p-values).

4.1.2 Statistics

This subsection presents different statistics on the datasets used in our experiments. Table 4.1 gives an overview of the basic statistics of each dataset after the pre-processing steps detailed before.

Dataset	Nodes	Edges	Features	Kept Features	Classes	Training / Validation / Test
DB1	183	277	1703	100 (5.9%)	4	74 / 18 / 91
DB2	183	279	1703	100 (5.9%)	4	74 / 18 / 91
DB3	215	365	1703	100 (5.9%)	4	87 / 21 / 107
DB4	251	450	1703	100 (5.9%)	4	101 / 25 / 125
DB5	1045	27791	576	100 (17.4%)	2	419 / 104 / 522
DB6	793	14816	319	100 (31.3%)	2	318 / 79 / 396
DB7	756	30780	480	100 (20.8%)	2	303 / 75 / 378
DB8	1392	2302	3703	100 (2.7%)	6	557 / 139 / 696
DB9	2485	5069	1434	100 (7.0%)	7	995 / 248 / 1242
DB10	3254	33218	4973	100 (2.0%)	14	1302 / 325 / 1627

Tab. 4.1.: Simple statistics of the 10 investigated datasets.

On the other hand, Tables 4.2 and 4.3 contain the Moran and Geary indices of all datasets. According to the Subsection 2.4.1, these indices give us an estimation of the spatial autocorrelation of labels in the graph. We observe that both the Moran and Geary indices are in agreement and therefore allow us to clearly decide which dataset will be classified **X**-driven or **A**-driven. The **A**-driven category contains the datasets for which the graph information is likely to improve the classification accuracy of algorithms exploiting the spatial autocorrelation hypothesis.

X-Driven	*DB*1	*DB*2	*DB*3	*DB*4
Moran's I	-0.12	-0.22	-0.15	-0.06
Geary's c	1.17	1.55	1.26	1.15

Tab. 4.2.: Moran and Geary indices of **X**-driven datasets. Since $I \leq I_0$ and $c \geq 1$, these datasets are characterised by a negative spatial autocorrelation and are then classified **X**-driven. $I_0 \approx 0$ for all datasets. More details in Subsection 2.4.1.

A-Driven	*DB*5	*DB*6	*DB*7	*DB*8	*DB*9	*DB*10
Moran's I	0.66	1.09	1.27	0.53	0.77	0.13
Geary's c	0.66	0.17	0.18	0.38	0.24	0.89

Tab. 4.3.: Moran and Geary indices of **A**-driven datasets. Since $I > I_0$ and $c < 1$, these datasets are characterised by a positive spatial autocorrelation and are then classified **A**-driven. $I_0 \approx 0$ for all datasets. More details in Subsection 2.4.1.

4.2 Performance assessment

To be able to estimate the real accuracy (or generalisation error) of the model on a given dataset, it is important that this measurement is done on nodes whose the label was not used during the learning phase (in order to avoid an optimistic bias). It is why a subset of the dataset is randomly chosen while preserving the initial proportion of labels as a test set (50% of nodes). The test set contains the nodes for which the labels are hidden during the learning phase and used to estimate the expected accuracy on unlabelled nodes. The remaining subset (50%) will be used to both train the model and select the best hyperparameters. Almost all the investigated algorithms defined in chapter 3 depend on hyperparameters which are not optimised during the learning phase. These external parameters influence the behavior of the model and therefore its classification accuracy. We used a 5-folds cross-validation [17, 22] on this remaining subset to tune these hyperparameters.

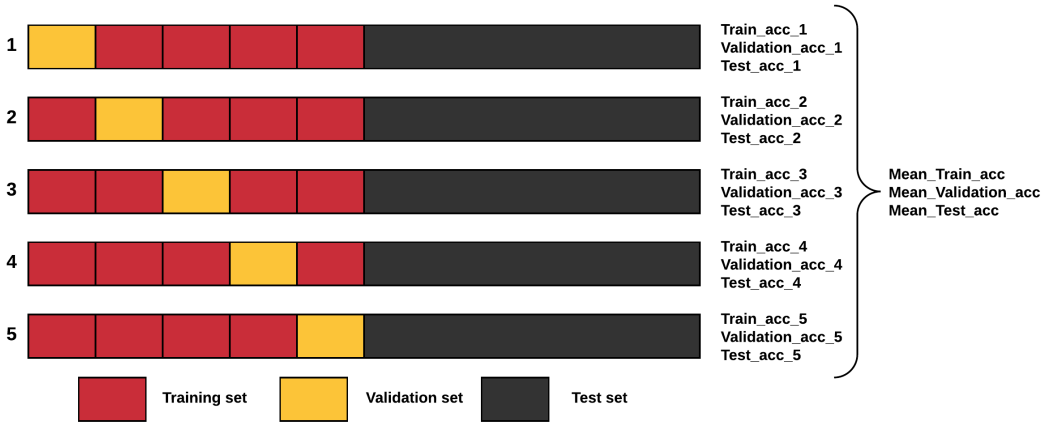

Fig. 4.1.: Illustration of the 5-folds cross-validation used in remaining subset (50% of nodes).

Figure 4.1 visually illustrates the process used in our study. This process consists in dividing the set into 5 subsets of the same size (called fold). Each fold therefore contains 10% of the total number of nodes and is built while preserving the initial proportion of labels. The model is trained on 4 folds (called training set, 40% of the total number of nodes) and evaluates on the remaining fold (called validation set, 10% of the total number of nodes). Table 4.1 provides for each dataset, an idea of the size of each set (training, validation and test) during the cross-validation phase. Each fold will take the role of the validation set. Hence, the performances on the training set, validation set and test set are averaged over these five learning. Then, the best hyperparameters are those providing the best validation accuracy (Mean_Validation_acc in Figure 4.1).
Now, we distinct two cases :

- All models except those related to the DCNN model are re-trained with the best hyperparameters on the five folds. Their accuracy then corresponds to their performance on the test set (Test_acc in Figure 4.2).

Fig. 4.2.: Illustration of the re-training over the 5 folds with the best hyperparameters.

- However, the DCNN-based models can not be re-trained on the 5 folds (50%) and tested on the test set since these models need a validation set to decide when to stop the learning (due to the early stopping procedure). To handle this exception, their accuracy corresponds to the average accuracy on the test set

(Mean_Test_acc in Figure 4.1) associated to the best hyperparameters during the cross-validation (so the model is not re-trained once again).

The accuracy obtained by this process corresponds to one run. We performed this procedure 10 times with different random distribution of nodes among the different sets. The final accuracy of the model is the average accuracy over these 10 runs. These multiple runs reduce the variance of the estimated accuracy caused by the variability of how the nodes are distributed in the different sets. Of course, to ensure equity between all the models, the 10 runs (distribution of the nodes in the sets) are the same for all models.

5 Discussion of results

This chapter is devoted to the evaluation of methods cited in Table 3.1. In this discussion, we compare the investigated models on the basis of multiple perspectives. The first one is the accuracy of each method over each dataset which is provided by the experimental procedure described previously. The second one is the Borda ranking computed from the previous values. The third one is the p-value between each method coming from the Wilcoxon signed-rank test applied on their performance across all datasets. These pairwise p-values allow to compare more precisely the relative performances between two methods.
The chapter is cut into three parts. The first section is dedicated to the comparison of the SVM-based methods. On the other hand, the second section discusses the results of the DCNN-based methods. Finally, the last section compares the two different approaches to each other.

5.1 SVM-based methods

The SVM-based methods are compared with each other as well as with the simplest baseline, the Trivial model.

From Table 5.1, Table 5.2 and Table 5.3, it is obvious that the Trivial model is overtaken by all the other models. Indeed, its performances are always clearly lower than those of the others, it is the last in the Borda ranking (where it got a score of 10 meaning that it never took control over a dataset) and finally, its p-values in Table 5.3 with the others are clearly below the fixed threshold of 5%. All the other methods are therefore able to improve the Trivial model by exploiting the features on nodes and the graph information.

An interesting result can be observed in Table 5.1 by comparing the SVM-$\mathbf{X}$ with all the other SVM-based models. We note that overall, the performance differences between the SVM-$\mathbf{X}$ with the other models are small on the first 4 datasets (*DB1*, *DB2*, *DB3*, *DB4*) and clearly increase on the next 6 datasets (*DB5*, *DB6*, *DB7*, *DB8*, *DB9*, *DB10*) on which the advantage is taken by the other models. This result is compatible with the autocorrelation measurements which are listed in Tables 4.2 and 4.3. Indeed, these last tables indicate that the first 4 datasets are $\mathbf{X}$-driven, meaning

that the graph information is not relevant for algorithms taking into account the autocorrelation hypothesis (i.e: all the algorithms using the graph in our study). This is why, on these datasets, the SVM-$\mathbf{X}$, using only the features on the nodes, is competitive with the 5 other models exploiting the structure of the graph. It even happens that the information extracted from the graph degrades the performance of the SVM. However, on the last 6 datasets, the graph information always improves performances.

The Borda ranking in Table 5.2 shows clearly that all methods exploiting the graph information in addition to the features on nodes are the best even if the SVM-$\mathbf{X}$ remains relevant on $\mathbf{X}$-driven datasets.

The scores associated with models having an acronym of the form SVM-$\mathbf{X}$-$\mathbf{G}$ and the score associated with the Auto-SVM-$\mathbf{X}$ in the Borda ranking are relatively close to each other. Moreover, the pairwise p-values coming from the Wilcoxon signed-rank test highlight the absence of a significant difference among these methods. The last two observations suggest that all methods taking the graph information into account are globally equal in terms of performance. However, according to the Table 5.3, the two graphs embedding coming from Gaussian kernels (SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$ and SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{cBoP}}$) and the Auto-SVM-$\mathbf{X}$ seem to stand out in a certain way from the others since their p-value with the SVM-$\mathbf{X}$ is below the threshold of 5%. We can also conclude that the graphs embedding extracted by the cBoP framework are of the same order of quality as those extracted from the BoP framework, at least for the node classification task.

An important notion to be taken into account is the scalability of methods since it impacts the scope of their application domains. Indeed, non-scalable methods are constrained to be able to be used only on relatively small datasets (some thousands of nodes), like the datasets used in our study. While in practice, it is likely to encounter very large problems (several million nodes). On such datasets, it is impossible in practice to use a unscalable method which involves matrix inversions or whose the number of learnable parameters is proportional to the number of nodes. Since the SVM-$\mathbf{X}$-$\mathbf{G}$ methods need a matrix inversion, they are no longer scalable methods. Hence, the Auto-SVM-$\mathbf{X}$ is the only model that is both among the scalable and best models over the $\mathbf{X}$-driven and $\mathbf{A}$-driven datasets.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
Trivial	45.05	56.04	47.66	47.2	70.5	71.46	69.05	32.61	29.23	15.61
SVM-$\mathbf{X}$	83.08	84.62	83.64	87.36	80.65	91.29	91.69	71.29	72.91	60.09
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	79.34	84.84	77.01	82.48	**84.67**	98.36	**97.3**	74.68	84.14	66.67
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	82.75	**87.58**	**83.93**	87.2	84.64	97.95	96.69	74.94	**86.72**	60.71
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	79.45	83.63	80.65	84.16	84.33	**98.59**	97.25	76.31	86.66	**66.99**
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	**83.96**	86.15	83.83	87.2	84.39	98.03	96.48	74.77	85.98	60.14
Auto-SVM-$\mathbf{X}$	82.64	85.38	**83.93**	**89.92**	83.24	98.46	96.32	**76.36**	85.73	60.82

Tab. 5.1.: Table containing the performances of the SVM-based methods on all datasets. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	52
Auto-SVM-$\mathbf{X}$	50
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	47
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	47
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	41
SVM-$\mathbf{X}$	31
Trivial	10

Tab. 5.2.: Borda ranking of the SVM-based methods computed on Table 5.1.

Method	Id	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Trivial	(1)	1	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**
SVM-$\mathbf{X}$	(2)	**0.005**	1	0.241	**0.017**	0.074	**0.009**	**0.009**
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	(3)	**0.005**	0.241	1	0.285	0.114	0.333	0.203
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	(4)	**0.005**	**0.017**	0.285	1	0.575	0.11	0.906
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	(5)	**0.005**	0.074	0.114	0.575	1	0.721	0.721
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	(6)	**0.005**	**0.009**	0.333	0.11	0.721	1	0.878
Auto-SVM-$\mathbf{X}$	(7)	**0.005**	**0.009**	0.203	0.906	0.721	0.878	1

Tab. 5.3.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between all the SVM-based methods. The p-values are computed on Table 5.1 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

5.2 DCNN-based methods

The DCNN-based methods are compared with each other as well as with the simplest baseline, the Trivial model.

As for the SVM-based methods, the same conclusions can be made about the Trivial model. This therefore demonstrates the interest for these DCNN-based methods.

By comparing in Table 5.4 the DCNN-$\mathbf{X}$ taking only the features on nodes into account with the DCNN-$\mathbf{X}$-$\mathbf{P}$ also exploiting the structure of the graph, we note that once again the difference in performance is increased on $\mathbf{A}$-driven datasets. On the other hand, DCNNs using a kernel containing global similarities between nodes (DCNN-$\mathbf{X}$-$\mathbf{PG}$) seem to strongly improve the DCNN-$\mathbf{X}$ not only on $\mathbf{A}$-driven datasets but also on $\mathbf{X}$-driven datasets. Table 5.5 and Table 5.6 show that the DCNN-$\mathbf{X}$ is significantly less efficient than all the other DCNN-based models.

Table 5.5 and 5.6 highlight that the DCNN-$\mathbf{X}$-$\mathbf{PG}$ with $\mathbf{G} \in \{\mathbf{\Pi}_h^{\text{BoP}}, \mathbf{\Pi}_h^{\text{cBoP}}, \mathbf{K}_{h,g}^{\text{BoP}}, \mathbf{K}_{h,g}^{\text{cBoP}}\}$ constitutes a real improvement of the DCNN-$\mathbf{X}$-$\mathbf{P}$, especially on $\mathbf{X}$-driven datasets. Indeed, all versions of DCNN-$\mathbf{X}$-$\mathbf{PG}$ benefit from a higher score in the Borda ranking and are all significantly different from the DCNN-$\mathbf{X}$-$\mathbf{P}$ according to the Wilcoxon signed-rank test. Moreover to show that the use of such kernel for $\mathbf{G}$ is relevant, that is to say that the interesting performance of the DCNN-$\mathbf{X}$-$\mathbf{PG}$ observed previously are not only due to the structure of the model but also to the kernel $\mathbf{G}$ used, we compared them to a DCNN-$\mathbf{X}$-$\mathbf{PG}$ where $\mathbf{G}$ is a random matrix whose values are between 0 and 1. This comparison is discussed in Section A.3 and demonstrates their relevance especially on $\mathbf{X}$-driven datasets.
The investigated DCNN-$\mathbf{X}$-$\mathbf{PG}$ models can be divided into two sets, one containing the matrices $\mathbf{\Pi}_h^{(c)\text{BoP}}$ and the other the Gaussian kernels $\mathbf{K}_{h,g}^{(c)\text{BoP}}$. According to the Borda ranking, the first set seems to stand out positively from the other set. However, the Wilcoxon signed-rank test indicates that there is not really a significant difference.

Regarding the regularisation, the Borda ranking shows that the two models benefiting from the regularisation (Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$ and Reg-DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_h^{\text{BoP}}$) improve the corresponding models without regularisation (DCNN-$\mathbf{X}$-$\mathbf{P}$ and DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_h^{\text{BoP}}$). From the Wilcoxon signed-rank test, we observe that the p-values are not below the significant level. However, the p-value between DCNN-$\mathbf{X}$-$\mathbf{P}$ and Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$ (0.059) is very close to the significant level. Despite this, the regularisation still seems to slightly improve the models, especially the DCNN-$\mathbf{X}$-$\mathbf{P}$.

As mentioned in the previous section, the scalability of methods is important when we are looking at large datasets. As cited in Section 2.7, the DCNN-$\mathbf{X}$-$\mathbf{P}$ is scalable. Hence, the DCNN-$\mathbf{X}$ is also scalable since it is a special case of DCNN-$\mathbf{X}$-$\mathbf{P}$ where the number of hops is set to zero.
Adding a kernel to DCNN-$\mathbf{X}$-$\mathbf{P}$ automatically makes the DCNN-$\mathbf{X}$-$\mathbf{PG}$ model unscalable due to the matrix inversion involved in the kernel computation. It therefore constitutes a significant improvement of DCNN-$\mathbf{X}$-$\mathbf{P}$ on small datasets like those investigated in our study.
The addition of the regularisation does not influence the scalability of methods. Con-

sequently, the Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$ could be an interesting improvement of DCNN-$\mathbf{X}$-$\mathbf{P}$ on large datasets.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
Trivial	45.05	56.04	47.66	47.2	70.5	71.46	69.05	32.61	29.23	15.61
DCNN-$\mathbf{X}$	77.49	77.32	75.7	85.1	78.95	91.71	91.48	69.31	70.77	56.09
DCNN-$\mathbf{X}$-$\mathbf{P}$	75.12	78.75	75.31	85.01	81.6	96.99	94.85	77.32	83.42	61.36
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	78.02	86.13	80.88	87.54	81.82	98.36	97.24	76.93	85.31	64.42
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	77.05	85.89	81.25	**87.71**	82.53	98.14	97.06	76.73	85.24	**64.62**
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{BoP}}$	75.54	85.67	80.67	87.25	81.78	97.75	97.2	77.39	85.2	62.77
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{cBoP}}$	76.88	85.54	80.5	87.55	82.13	97.71	97.07	**77.49**	85.31	62.75
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	74.46	82.68	78.52	82.82	82.37	98.2	96.94	77.4	**85.67**	62.83
Reg-DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	77.87	**86.26**	**81.27**	86.75	**82.81**	**98.37**	97.14	76.86	85.5	64.21

Tab. 5.4.: Table containing performances of the DCNN-based methods on all datasets. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
Reg-DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	75
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	72
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	66
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{cBoP}}$	58
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{BoP}}$	55
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	53
DCNN-$\mathbf{X}$-$\mathbf{P}$	32
DCNN-$\mathbf{X}$	28
Trivial	10

Tab. 5.5.: Borda ranking of the DCNN-based methods computed on Table 5.4.

Method	Id	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Trivial	(1)	1	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**
DCNN-$\mathbf{X}$	(2)	**0.005**	1	**0.037**	**0.005**	**0.007**	**0.007**	**0.007**	**0.017**	**0.005**
DCNN-$\mathbf{X}$-$\mathbf{P}$	(3)	**0.005**	**0.037**	1	**0.009**	**0.007**	**0.005**	**0.005**	0.059	**0.007**
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	(4)	**0.005**	**0.005**	**0.009**	1	0.683	**0.037**	0.11	0.114	0.878
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	(5)	**0.005**	**0.007**	**0.007**	0.683	1	0.059	0.114	0.074	0.333
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{BoP}}$	(6)	**0.005**	**0.007**	**0.005**	**0.037**	0.059	1	0.507	0.333	**0.047**
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{cBoP}}$	(7)	**0.005**	**0.007**	**0.005**	0.11	0.114	0.507	1	0.241	0.093
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	(8)	**0.005**	**0.017**	0.059	0.114	0.074	0.333	0.241	1	**0.032**
Reg-DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	(9)	**0.005**	**0.005**	**0.007**	0.878	0.333	**0.047**	0.093	**0.032**	1

Tab. 5.6.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between all the DCNN-based methods. The p-values are computed on Table 5.4 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

5.3 Global discussion

This section is devoted to the comparison between SVM-based and DCNN-based methods. Only the most efficient method of each type of model is kept for discussion. The Reg-DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ is not discussed since it did not significantly improve the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$.

By comparing directly the SVM-$\mathbf{X}$ and the DCNN-$\mathbf{X}$ in Table 5.7, it is obvious that the DCNN-$\mathbf{X}$ is much less efficient than its direct competitor among the SVM-based methods. Tables 5.8 and 5.9 confirm this observation.

Based on Table 5.7, the DCNN-$\mathbf{X}$-$\mathbf{P}$ remains clearly less efficient on $\mathbf{X}$-driven datasets than the simple SVM-$\mathbf{X}$. However, on $\mathbf{A}$-driven datasets, it reverses the positions. This shows that the DCNN is capable of exploiting the structure of the graph well but has weaknesses to capture the relevant information among the features on the nodes. However, based on all datasets, the Wilcoxon signed-rank test indicates that there is no significant difference.

The addition of a kernel (in this case the matrix $\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$) in the DCNN-$\mathbf{X}$-$\mathbf{P}$ to form the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ allows to compensate the weakness of the DCNN-$\mathbf{X}$-$\mathbf{P}$ on the $\mathbf{X}$-driven datasets. Indeed, the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ is competitive on the $\mathbf{X}$-driven datasets compared to the SVM-$\mathbf{X}$ and slightly improves the DCNN-$\mathbf{X}$-$\mathbf{P}$ on $\mathbf{A}$-driven datasets. This improvement is confirmed by the Borda ranking and the Wilcoxon signed-rank test in which the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ is almost significantly different from the SVM-$\mathbf{X}$. In addition, the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ achieves similar performance to that of the SVM-$\mathbf{X}$-$\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$ and Auto-SVM-$\mathbf{X}$, which both compensate for the weakness of SVM-$\mathbf{X}$ on $\mathbf{A}$-driven datasets. Indeed, the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$, the SVM-$\mathbf{X}$-$\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$ and the Auto-SVM-$\mathbf{X}$ have similar scores in the Borda ranking.

The regularisation added in the DCNN-$\mathbf{X}$-$\mathbf{P}$ improved it so that it got a better ranking in the Borda ranking than the SVM-$\mathbf{X}$. However, it remains less efficient than SVM-$\mathbf{X}$ on $\mathbf{X}$-driven datasets.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
SVM-$\mathbf{X}$	**83.08**	84.62	83.64	87.36	80.65	91.29	91.69	71.29	72.91	60.09
SVM-$\mathbf{X}$-$\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$	82.75	**87.58**	**83.93**	87.2	**84.64**	97.95	96.69	74.94	**86.72**	60.71
Auto-SVM-$\mathbf{X}$	82.64	85.38	**83.93**	**89.92**	83.24	**98.46**	96.32	76.36	85.73	60.82
DCNN-$\mathbf{X}$	77.49	77.32	75.7	85.1	78.95	91.71	91.48	69.31	70.77	56.09
DCNN-$\mathbf{X}$-$\mathbf{P}$	75.12	78.75	75.31	85.01	81.6	96.99	94.85	77.32	83.42	61.36
DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$	78.02	86.13	80.88	87.54	81.82	98.36	**97.24**	76.93	85.31	**64.42**
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	74.46	82.68	78.52	82.82	82.37	98.2	96.94	**77.4**	85.67	62.83

Tab. 5.7.: Table containing performances of some SVM-based and DCNN-based methods on all datasets. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
Auto-SVM-$\mathbf{X}$	54
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	53
SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$	52
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	42
SVM-$\mathbf{X}$	32
DCNN-$\mathbf{X}$-$\mathbf{P}$	30
DCNN-$\mathbf{X}$	16

Tab. 5.8.: Borda ranking of some SVM-based and DCNN-based methods computed on Table 5.7.

Method	Id	(1)	(2)	(3)	(4)	(5)	(6)	(7)
SVM-$\mathbf{X}$	(1)	1	**0.017**	**0.009**	**0.009**	0.878	0.074	0.445
SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$	(2)	**0.017**	1	0.906	**0.005**	**0.028**	0.508	0.139
Auto-SVM-$\mathbf{X}$	(3)	**0.009**	0.906	1	**0.005**	**0.013**	0.508	0.169
DCNN-$\mathbf{X}$	(4)	**0.009**	**0.005**	**0.005**	1	**0.037**	**0.005**	**0.017**
DCNN-$\mathbf{X}$-$\mathbf{P}$	(5)	0.878	**0.028**	**0.013**	**0.037**	1	**0.009**	0.059
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	(6)	0.074	0.508	0.508	**0.005**	**0.009**	1	0.114
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	(7)	0.445	0.139	0.169	**0.017**	0.059	0.114	1

Tab. 5.9.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between some of the SVM-based and DCNN-based methods. The p-values are computed on Table 5.7 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

5.3.1 Execution time

Table 5.10 groups together the execution times (training times) of the previous models on all the datasets. First, in general, we observe that the training time is strongly correlated to the size of the datasets. Thus, the largest datasets (*DB5*,*DB8*,*DB9*,*DB10*) require more time. Next, among the SVM-based models, we observe that the training time of the SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$ remains relatively similar to that of the SVM-$\mathbf{X}$, except on the largest datasets where it increases. On all datasets, the Auto-SVM-$\mathbf{X}$ needs a bigger training time since it involves several SVM models. Regarding the DCNN-based models, they have approximately similar execution times on small datasets but differences appear on the bigger ones. The most important observation is that SVM-based methods are clearly faster than DCNN-based methods on all datasets, except the Auto-SVM-$\mathbf{X}$ which obtains a time close to that of DCNN-$\mathbf{X}$-$\mathbf{P}$ on *DB10*.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
SVM-$\mathbf{X}$	0.02	0.01	0.01	0.01	0.02	0	0.01	0.06	0.17	0.62
SVM-$\mathbf{X}$-$\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$	0.01	0.03	0.02	0.01	0.03	0.01	0.01	0.29	0.51	2.22
Auto-SVM-$\mathbf{X}$	0.05	0.12	0.08	0.22	0.6	0.15	0.09	1.58	5.94	40.87
DCNN-$\mathbf{X}$	4.07	4.07	2.74	4.07	7.23	7.45	6.35	9.41	16.3	20.87
DCNN-$\mathbf{X}$-$\mathbf{P}$	5	4.45	3.68	4.13	10.67	9.9	7.19	18.76	34.6	42.43
DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$	5.23	5.27	3.66	4.46	9.09	9.17	7.42	18.63	47.38	63.01
Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$	5.48	5.68	4.34	4.64	9.3	10.46	10.21	28.82	86.27	98.14

Tab. 5.10.: Table containing the estimated execution time (seconds) of some of the SVM-based and DCNN-based methods on each dataset. The execution time is averaged over the 10 runs with the best hyperparameters.

5.3.2 Main findings

Comparison between the BoP and cBoP, Gaussian and MDS kernels

On the node classification task, we observed that the graphs embedding extracted from the Gaussian kernels ($\mathbf{X}_{\text{h,g}}^{G,\text{BoP}}$ and $\mathbf{X}_{\text{h,g}}^{G,\text{cBoP}}$) coming from the BoP and cBoP frameworks seem to perform better than those from MDS kernels ($\mathbf{X}_{\text{h,mds}}^{G,\text{BoP}}$,$\mathbf{X}_{\text{h,mds}}^{G,\text{cBoP}}$). Moreover, we do not see a significant difference between the two frameworks, at least for the margin imposed in our experiences (2.54).

Auto-SVM model vs kernel-based SVM models

The Auto-SVM-$\mathbf{X}$ taking autocovariates as additional features reaches similar performances than the SVM models using a low-dimensional graph embedding coming from the BoP and cBoP frameworks (SVM-$\mathbf{X}$-$\mathbf{X}^G$). No significant difference was observed among them.

Suggested improvements for the DCNN (DCNN-based models)

With regard to DCNN-based methods, the extension inspired from the Auto-SVM-$\mathbf{X}$ consisting in adding autocovariates as additional features and updating them at each epoch, denoted by Auto-DCNN-$\mathbf{X}$-$\mathbf{P}$, produced disappointing results since it degraded the performance of the original DCNN model (DCNN-$\mathbf{X}$-$\mathbf{P}$). On the other hand, the extension adding a regularisation term to enforce local consistency in the predicted solution, denoted by Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$, slightly improves the DCNN-$\mathbf{X}$-$\mathbf{P}$ while maintaining its scalability, but not enough to be a significant improvement. We showed that the initialisation did not matter and that the square of the L_2-norm works better than the L_1-norm. The DCNN-$\mathbf{X}$-$\mathbf{PG}$ using a kernel with positive values improves significantly the DCNN-$\mathbf{X}$-$\mathbf{P}$ on $\mathbf{X}$-driven datasets. Among them, the matrices $\mathbf{\Pi}_{\text{h}}^{\text{BoP}}$ and $\mathbf{\Pi}_{\text{h}}^{\text{cBoP}}$ produced slightly higher performance than the Gaussian

kernels ($\mathbf{K}^{\text{BoP}}_{\text{h,g}}$ and $\mathbf{K}^{\text{cBoP}}_{\text{h,g}}$). Again, we did not find a difference between the BoP and the cBoP. However, the use of such kernels makes this model unscalable.

SVM-based models vs DCNN-based models

SVM-$\mathbf{X}$ clearly outperforms its direct concurrent (DCNN-$\mathbf{X}$), both in terms of performance and execution time. Two trends appeared in results. On the one hand, the SVM-$\mathbf{X}$ is efficient on $\mathbf{X}$-driven datasets while on the other hand, the DCNN-$\mathbf{X}$-$\mathbf{P}$ is efficient on $\mathbf{A}$-driven datasets. However, it is possible to compensate the weakness of SVM-$\mathbf{X}$ on $\mathbf{A}$-driven datasets by the addition of a low-dimensional graph embedding (e.g: SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$) or of autocovariates (Auto-SVM-$\mathbf{X}$). On the other hand, we shown that the addition of a kernel in the DCNN-$\mathbf{X}$-$\mathbf{P}$ (e.g: DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$) allows it to compensate its weakness on $\mathbf{X}$-driven datasets and thus become competitive against the SVM-$\mathbf{X}$-$\mathbf{X}^{G}$ and the Auto-SVM-$\mathbf{X}$.

Identification of efficient methods in terms of performance and scalability on $\mathbf{X}$-driven and $\mathbf{A}$-driven datasets

Based on the previous observations, the most efficient models on $\mathbf{X}$-driven datasets are the SVM-$\mathbf{X}$, the SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$, the Auto-SVM-$\mathbf{X}$ and finally the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$. However, the SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$ and the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$ are not scalable since they both involve the computation of a kernel on the graph. Hence, they can not be applied on large datasets. In addition, the training time and the number of hyperparameters to optimise in the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$ are much higher than in the other models, which makes it more difficult to use. From these findings, we recommend the use of the SVM-$\mathbf{X}$ or the Auto-SVM-$\mathbf{X}$.

On the $\mathbf{A}$-driven datasets, the best performing methods are the SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$, the Auto-SVM-$\mathbf{X}$, the DCNN-$\mathbf{X}$-$\mathbf{P}$ and the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$. For the same reasons as before, we recommend favoring the use of the Auto-SVM-$\mathbf{X}$. Indeed, SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$ and DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$ are not scalable and the DCNN-$\mathbf{X}$-$\mathbf{P}$ depends on many more hyperparameters than the Auto-SVM-$\mathbf{X}$.

However, in practice, it is difficult to evaluate in advance if a dataset is $\mathbf{X}$-driven or $\mathbf{A}$-driven. It is not possible to test the autocorrelation hypothesis when we only partially know the labels. Unless, a priori on the basis of prior knowledge, we know whether or not the dataset benefits from an autocorrelation. If we do not have prior knowledge on this subject, it is important to use a method which works well both on $\mathbf{X}$-driven and $\mathbf{A}$-driven datasets. From the previous discussions, such models are the SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$, the Auto-SVM-$\mathbf{X}$ and the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}^{\text{BoP}}_{\text{h}}$. The Auto-SVM-$\mathbf{X}$ therefore seems to be the model to be favored in all circumstances since it only depends on a single hyperparameter and is completely scalable.

6 Conclusion

During this study, we broadly compared two different families of algorithms for the problem of semi-supervised classification on a graph. Namely, the SVM-based methods which are more traditional machine learning algorithms and the DCNN-based methods which are coming from deep learning. The two families of algorithms mentioned above are the result of the respective extension or modification of a linear SVM model on one side and the DCNN on the other side. These extensions and modifications consisted in the use of kernels (from BoP and cBoP frameworks) on graph, the use of autocovariates or the addition of a regularisation imposing a local consistency on the solution produced by the model.

A procedure has been put in place to be able to best assess the real performance of these models (the expected accuracy on an unseen example) on a dozen well-known datasets. The aim of this procedure was therefore to avoid optimistic bias and to reduce the randomness involved both in the initialisation of the learnable parameters and in the distribution of data in the training set, validation set and test set. However, it should be noted that the performances obtained are not exact since the optimisation of the hyperparameters was partial. Indeed, for reasons of timing, we tested only a few values considered relevant for each of the hyperparameters. Then, the comparison of the models was carried out on the basis of these performances, a global ranking (Borda ranking) and a statistical test to more precisely measure the relative performances between two methods (Wilcoxon signed-rank test).

6.1 Main findings

We found that the kernels and graphs embedding from the BoP and the cBoP frameworks are of similar quality on the node classification task, at least for the margins that we used in the cBoP. Among all kernels, it seems that the Gaussian kernels provide slightly better results than MDS kernels.
The Auto-SVM-$\mathbf{X}$ based on autocovariates proved to be competitive on all datasets (similar performance) with the SVM models using a low-dimensional graph embedding as additional features (SVM-$\mathbf{X}$-$\mathbf{X}^G$).
The models obtained by the addition of a low-dimensional graph embedding and autocovariates in the SVM-$\mathbf{X}$ have significantly improved the SVM-$\mathbf{X}$ exploiting only

the features on nodes on the $\mathbf{A}$-driven datasets.
Among all the investigated versions of the DCNN-$\mathbf{X}$-$\mathbf{P}$, only the addition of a kernel in the diffusion process (e.g: DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$) made it possible to improve the model significantly. This modification compensates the weakness of the DCNN-$\mathbf{X}$-$\mathbf{P}$ on $\mathbf{X}$-driven datasets and allows it to be competitive (same performance) with the SVM-based methods (SVM-$\mathbf{X}$-$\mathbf{X}^G$ and Auto-SVM-$\mathbf{X}$). In another way, the addition of the regularisation in the loss function used by the DCNN-$\mathbf{X}$-$\mathbf{P}$ slightly improves its performance but not significantly. On the other hand, the adaption of the DCNN-$\mathbf{X}$-$\mathbf{P}$ to handle the same autocovariates as in the Auto-SVM-$\mathbf{X}$ have degraded its performances and therefore is not relevant.
On $\mathbf{X}$-driven datasets, the best models are the SVM-$\mathbf{X}$, the SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$, the Auto-SVM-$\mathbf{X}$ and the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$. While on $\mathbf{A}$-driven datasets, the best models are the SVM-$\mathbf{X}$-$\mathbf{X}_{\mathrm{h,g}}^{G,\mathrm{BoP}}$, the Auto-SVM-$\mathbf{X}$, the DCNN-$\mathbf{X}$-$\mathbf{P}$ and the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$. The SVM-$\mathbf{X}$-$\mathbf{X}$ and the DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$ are not scalable models because they use a kernel. Moreover, DCNN-based models are more difficult to train than the SVM-based models since they depend on a larger number of hyperparameters and need more training time. Hence, the Auto-SVM-$\mathbf{X}$ seems to be the model to be favored because it is effective on both $\mathbf{X}$-driven and $\mathbf{A}$-driven datasets, scalable and easy to train since it only depends on a single hyperparameter.

In conclusion, these experiments made it possible to highlight that on datasets of a few thousand nodes, more traditional methods inspired by the linear SVM remain preferable to a deep learner like the DCNN.

6.2 Further works

From this conclusion, it could be interesting to make additional experiments among all the best scalable methods presented in our study (Auto-SVM-$\mathbf{X}$, DCNN-$\mathbf{X}$-$\mathbf{P}$ and Reg-DCNN-$\mathbf{X}$-$\mathbf{P}$) on large datasets. Such a dataset could be the US Patent Citations network [21] containing approximately almost four millions nodes. Perhaps due to the high availability of training samples on this kind of datasets, the two DCNN-based methods would become more competitive than the SVM-based method and the DCNN with the regularisation significantly more efficient than the DCNN-$\mathbf{X}$-$\mathbf{P}$.
Since in our study, the SVM-$\mathbf{X}$-$\mathbf{X}^G$ (based on kernels) did not produce significantly better results than the Auto-SVM-$\mathbf{X}$ (which does not use a kernel), it could also be interesting to investigated if it exists other kernels that are significantly more efficient than the Auto-SVM-$\mathbf{X}$, at least to justify their use on small datasets.

6.3 Limitations

In a general way, in our study, the SVM approach seems to win against the deep learning approach. The most likely reason for these results is due to the small size of the datasets used. On this type of datasets, SVM-based methods have an advantage over DCNN-based methods. Indeed, the number of learnable parameters is much lower in a SVM model than in a DCNN model. Hence, a DCNN model need a bigger training set. In addition, the deep learners used in our experiments use an early stopping procedure that requires sacrificing some of the data available for training to stop learning before it begins to overfit.

Bibliography

[1]Ethem Alpaydin. *Introduction to machine learning*. MIT Press, 2004 (cit. on pp. 7–9, 15, 27, 28).

[2]James Atwood and Don Towsley. “Diffusion-convolutional neural networks”. In: *Advances in neural information processing systems*. 2016, pp. 1993–2001 (cit. on pp. 2, 11, 19, 21, 42).

[3]Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006 (cit. on pp. 7, 10, 11, 17, 23, 27, 28).

[4]Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. “A comprehensive survey of graph embedding: Problems, techniques, and applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 30.9 (2018), pp. 1616–1637 (cit. on p. 34).

[5]Jie Chen, Tengfei Ma, and Cao Xiao. “Fastgcn: fast learning with graph convolutional networks via importance sampling”. In: *arXiv preprint arXiv:1801.10247* (2018) (cit. on p. 33).

[6]Yihua Chen, Eric K Garcia, Maya R Gupta, Ali Rahimi, and Luca Cazzanti. “Similarity-based classification: Concepts and algorithms”. In: *Journal of Machine Learning Research* 10.Mar (2009), pp. 747–776 (cit. on pp. 22, 28).

[7]P De Jong, C Sprenger, and F Van Veen. “On extreme values of Moran’s I and Geary’s c”. In: *Geographical Analysis* 16.1 (1984), pp. 17–24 (cit. on p. 14).

[8]Chuong B Do and Serafim Batzoglou. “What is the expectation maximization algorithm?” In: *Nature biotechnology* 26.8 (2008), pp. 897–899 (cit. on p. 18).

[9]Xiaowen Dong. *Guest lecture :* "*Graph signal processing Concepts, tools and applications*". Department of Engineering Science, University of Oxford (cit. on p. 11).

[10]Pierre Dupont. *Lecture notes :* "*LINGI2262: Support Vector Machines*". Université Catholique de Louvain. 2018 (cit. on pp. 15, 16).

[11]Peter Emerson. “The original Borda count and partial voting”. In: *Social Choice and Welfare* 40.2 (2013), pp. 353–358 (cit. on p. 31).

[12]Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. “LIBLINEAR: A library for large linear classification”. In: *Journal of machine learning research* 9.Aug (2008), pp. 1871–1874 (cit. on p. 17).

[13] François Fouss, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens. "An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification". In: *Neural networks* 31 (2012), pp. 53–72 (cit. on pp. 22, 27, 32, 65).

[14] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and models for network data and link analysis*. Cambridge University Press, 2016 (cit. on pp. 1, 5, 7, 22, 27, 28, 32, 65).

[15] Kevin Françoisse, Ilkka Kivimäki, Amin Mantrach, Fabrice Rossi, and Marco Saerens. "A bag-of-paths framework for network data analysis". In: *Neural Networks* 90 (2017), pp. 90–111 (cit. on pp. 1, 22–27, 41).

[16] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001 (cit. on pp. 9, 15).

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 2, 8–12, 21, 23, 38, 44).

[18] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864 (cit. on p. 34).

[19] Guillaume Guex, Sylvain Courtain, and Marco Saerens. "Covariance and correlation kernels on a graph in the generalized bag-of-paths formalism". In: *arXiv preprint arXiv:1902.03002* (2019) (cit. on pp. 22, 41).

[20] Guillaume Guex, Ilkka Kivimäki, and Marco Saerens. "Randomized optimal transport on a graph: framework and new distance measures". In: *Network Science* 7.1 (2019), pp. 88–122 (cit. on pp. 1, 25).

[21] Bronwyn Hall, Adam Jaffe, and Manuel Trajtenberg. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*. NBER Working Papers 8498. National Bureau of Economic Research, Inc, 2001 (cit. on p. 58).

[22] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013 (cit. on pp. 7, 15, 27, 44).

[23] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016) (cit. on pp. 33, 42).

[24] Ilkka Kivimäki, Masashi Shimbo, and Marco Saerens. "Developments in the theory of randomized shortest paths with a comparison of graph node distances". In: *Physica A: Statistical Mechanics and its Applications* 393 (2014), pp. 600–616 (cit. on p. 22).

[25] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009 (cit. on p. 33).

[26] Zachary F Lansdowne and Beverly S Woodward. "Applying the borda ranking method". In: *Air Force Journal of Logistics* 20.2 (1996), pp. 27–29 (cit. on p. 31).

[27] Bertrand Lebichot and Marco Saerens. "An experimental study of graph-based semi-supervised classification with additional node information". In: *arXiv preprint arXiv:1705.08716* (2017) (cit. on pp. 1, 2, 8, 13–15, 17, 29, 41).

[28] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on pp. 12, 20).

[29]Yann LeCun, LD Jackel, Léon Bottou, et al. “Learning algorithms for classification: A comparison on handwritten digit recognition”. In: *Neural networks: the statistical mechanics perspective* 261 (1995), p. 276 (cit. on p. 10).

[30]Jure Leskovec and Julian J Mcauley. “Learning to discover social circles in ego networks”. In: *Advances in neural information processing systems*. 2012, pp. 539–547 (cit. on p. 42).

[31]Bentian Li and Dechang Pi. “Network representation learning: a systematic literature review”. In: *Neural Computing and Applications* (2020), pp. 1–33 (cit. on p. 34).

[32]Sofus A Macskassy and Foster Provost. “Classification in networked data: A toolkit and a univariate case study”. In: *Journal of machine learning research* 8.May (2007), pp. 935–983 (cit. on p. 1).

[33]Martın Abadi, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on p. 20).

[34]Safi Mbungu and Marco Saerens. “Graph embedding with application to semi-supervised classification, visualization, reconstruction and neighborhood preservation tasks: an experimental comparison”. In: () (cit. on pp. 33, 41).

[35]Mary L McHugh. “The chi-square test of independence”. In: *Biochemia medica: Biochemia medica* 23.2 (2013), pp. 143–149 (cit. on p. 43).

[36]Stefano Melacci and Mikhail Belkin. “Laplacian support vector machines trained in the primal”. In: *Journal of Machine Learning Research* 12.Mar (2011), pp. 1149–1184 (cit. on p. 32).

[37]Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119 (cit. on p. 33).

[38]Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Vol. 7700. springer, 2012 (cit. on p. 9).

[39]M. E. J. Newman. *Networks: An introduction*. Oxford University Press, 2010 (cit. on pp. 1, 5).

[40]Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710 (cit. on p. 33).

[41]Stuart Russell and Peter Norvig. *Artificial intelligence: A modern approach*. 2nd. Prentice Hall, 2003 (cit. on pp. 1, 2, 7).

[42]Graeme D Ruxton and Markus Neuhäuser. “When should we use one-tailed hypothesis testing?” In: *Methods in Ecology and Evolution* 1.2 (2010), pp. 114–117 (cit. on p. 31).

[43]Marco Saerens and Christine Decaestecker. *Lecture notes : "LSINF2275: Feature selection and extraction"*. Université Catholique de Louvain (cit. on p. 27).

[44]Sawada, Michael. *Global Spatial Autocorrelation Indices - Moran’s I, Geary’s C and the General Cross-Product Statistic*. Department of Geography, University of Ottawa, `http://www.lpc.uottawa.ca/publications/moransi/moran.htm`. [Online; accessed 24-March-2020]. 2009 (cit. on p. 14).

[45] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, et al. “Collective classification in network data”. In: *AI magazine* 29.3 (2008), pp. 93–93 (cit. on pp. 33, 42).

[46] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014 (cit. on pp. 7, 8, 11, 12, 15, 17, 27).

[47] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”. In: *IEEE signal processing magazine* 30.3 (2013), pp. 83–98 (cit. on p. 11).

[48] Sandro Skansi. *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018 (cit. on pp. 9, 10).

[49] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272 (cit. on p. 32).

[50] Wikipedia contributors. *Gauss–Seidel method — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Gauss_Seidel_method&oldid=956135376. [Online; accessed 18-May-2020]. 2020 (cit. on p. 26).

[51] Wikipedia contributors. *Hinge loss — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Hinge_loss&oldid=949912210. [Online; accessed 24-May-2020]. 2020 (cit. on p. 21).

[52] Wikipedia contributors. *Ranking — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Ranking&oldid=956080472. [Online; accessed 25-May-2020]. 2020 (cit. on p. 31).

[53] Wikipedia contributors. *Tobler’s first law of geography — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Tobler’s_first_law_of_geography&oldid=929891532. [Online; accessed 24-March-2020]. 2019 (cit. on p. 14).

[54] Wikipedia contributors. *Wilcoxon signed-rank test — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Wilcoxon_signed-rank_test&oldid=955167703. [Online; accessed 25-May-2020]. 2020 (cit. on p. 32).

[55] RF Woolson. “Wilcoxon signed-rank test”. In: *Wiley encyclopedia of clinical trials* (2007), pp. 1–3 (cit. on p. 31).

[56] Zonghan Wu, Shirui Pan, Fengwen Chen, et al. “A comprehensive survey on graph neural networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020) (cit. on pp. 11, 33).

[57] Jie Zhou, Ganqu Cui, Zhengyan Zhang, et al. “Graph neural networks: A review of methods and applications”. In: *arXiv preprint arXiv:1812.08434* (2018) (cit. on p. 33).

[58] Xiaojin Zhu and Andrew B Goldberg. “Introduction to semi-supervised learning”. In: *Synthesis lectures on artificial intelligence and machine learning* 3.1 (2009), pp. 1–130 (cit. on pp. 1, 8, 13).

Appendix

A

The appendix contains all the experimental results of models which are not discussed in Chapter 5 for reasons of clarity.

A.1 SVM with RL and RCT kernels

Since the Regularised Laplacian Kernel (RL) and Regularised Commute-Time Kernel (RCT) [14] outperformed a large amount of other kernels in a simple similarity-based approach in [13], we decided to add them in our experiments. We extracted a graph embedding from them with the process described in Subsection 2.8.4. These low-dimensional graphs embedding are respectively denoted by $\mathbf{X}^G_{\text{RL}}$ and $\mathbf{X}^G_{\text{RCT}}$. They were then added as additional features in a SVM model.
However, both kernels depend on a parameter denoted by α, but playing a different role in the RL and RCT kernels. For the RL, the following condition must be satisfied $0 < \alpha < \rho(\mathbf{L})^{-1}$ where the upper bound is the inverse of the spectral radius of the Laplacian matrix ($\mathbf{L} = \mathbf{D} - \mathbf{A}$ with $\mathbf{D} = \mathbf{Diag}\,(\mathbf{Ae})$ and $\mathbf{A}$ is the adjacency matrix) of the graph. We tested four α values that are uniformly distributed in the interval mentioned above. On the other hand, for the RCT, the condition to be satisfied being $0 < \alpha < 1$, we chose four α values distributed uniformly in this interval. Table A.1 recaps those two new versions of SVM-$\mathbf{X}$-$\mathbf{X}^G$.

Acronym	*Param*	*Values*	*Description*
SVM-$\mathbf{X}$-$\mathbf{X}^G_{\text{RL}}$	C	$10^{[-4,-3,2,0,2,3,4]}$	SVM with additional features extracted from the RL kernel
	α	$[0.2, 0.4, 0.6, 0.8]\rho(\mathbf{L})^{-1}$	
SVM-$\mathbf{X}$-$\mathbf{X}^G_{\text{RCT}}$	C	$10^{[-4,-3,2,0,2,3,4]}$	SVM with additional features extracted from the RCT kernel
	α	$0.2, 0.4, 0.6, 0.8$	

Tab. A.1.: Summary of two additional SVM-$\mathbf{X}$-$\mathbf{X}^G$ models with their acronym, their list of hyperparameters and a short description. C controls the trade-off between the hard and soft margin in the SVM model. As mentioned above, α plays a different role in the RL and RCT kernels.

From the Borda ranking in Table A.3, it is clear that the graphs embedding from the BoP and cBoP frameworks are more efficient. This difference is even more obvious in the Table A.2. Indeed, the two new graphs embedding have much weaker performances on the datasets *DB5*,*DB6*,*DB7*,*DB8* and *DB9* which are $\mathbf{A}$-driven. They

have difficulties to improve the SVM-$\mathbf{X}$ which only uses the features on the nodes. The Wilcoxon signed-rank test in Table A.4 highlights two elements. First, the two Gaussian kernels are significantly better than the RL and RCT kernels. Secondly, the RL is significantly more efficient than the RCT.
Since overall, these two kernels did not provide very interesting results, they were removed from the main discussion.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
SVM-$\mathbf{X}$	83.08	84.62	83.64	87.36	80.65	91.29	91.69	71.29	72.91	60.09
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	79.34	84.84	77.01	82.48	**84.67**	98.36	**97.3**	74.68	84.14	66.67
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	82.75	**87.58**	**83.93**	87.2	84.64	97.95	96.69	74.94	**86.72**	60.71
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	79.45	83.63	80.65	84.16	84.33	**98.59**	97.25	**76.31**	86.66	**66.99**
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	**83.96**	86.15	83.83	87.2	84.39	98.03	96.48	74.77	85.98	60.14
SVM-$\mathbf{X}$-$\mathbf{X}^{G}_{\text{RL}}$	82.75	84.62	**83.93**	**87.6**	81	95.98	91.59	71.84	82.52	60.09
SVM-$\mathbf{X}$-$\mathbf{X}^{G}_{\text{RCT}}$	82.75	84.51	**83.93**	87.36	80.36	91.26	91.72	71.67	77.76	60.09

Tab. A.2.: Table containing the performances on all datasets of some SVM-based methods. Especially, embeddings coming from the RL and RCT. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	53
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	48
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	44
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	42
SVM-$\mathbf{X}$-$\mathbf{X}^{G}_{\text{RL}}$	35
SVM-$\mathbf{X}$-$\mathbf{X}^{G}_{\text{RCT}}$	28
SVM-$\mathbf{X}$	27

Tab. A.3.: Borda ranking of some SVM-based methods computed on Table A.2.

Method	Id	(1)	(2)	(3)	(4)	(5)	(6)	(7)
SVM-$\mathbf{X}$	(1)	1	0.241	**0.017**	0.074	**0.009**	0.069	0.726
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,mds}}$	(2)	0.241	1	0.285	0.114	0.333	0.575	0.285
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{BoP}}_{\text{h,g}}$	(3)	**0.017**	0.285	1	0.575	0.11	**0.017**	**0.017**
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,mds}}$	(4)	0.074	0.114	0.575	1	0.721	0.169	0.074
SVM-$\mathbf{X}$-$\mathbf{X}^{G,\text{cBoP}}_{\text{h,g}}$	(5)	**0.009**	0.333	0.11	0.721	1	**0.022**	**0.022**
SVM-$\mathbf{X}$-$\mathbf{X}^{G}_{\text{RL}}$	(6)	0.069	0.575	**0.017**	0.169	**0.022**	1	**0.043**
SVM-$\mathbf{X}$-$\mathbf{X}^{G}_{\text{RCT}}$	(7)	0.726	0.285	**0.017**	0.074	**0.022**	**0.043**	1

Tab. A.4.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between some SVM-based methods. The p-values are computed on Table A.2 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

A.2 DCNN-**X**-**G**

In this section, we assess the DCNN-$\mathbf{X}$-$\mathbf{G}$ with $G \in \{\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}, \mathbf{\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}, \mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}, \mathbf{K}_{\mathrm{h,g}}^{\mathrm{cBoP}}\}$ by comparing it to the DCNN-$\mathbf{X}$-$\mathbf{P}$ and the improved version DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$.
Table A.5 summarises these new models with the range of hyperparameters tested.

*A*cronym	*P*aram	*V*alues	*D*escription
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	λ θ	$10^{[-3,-2]}$ $10^{[-3,-2,-1,0]}$	1-hop DCNN with a modified diffusion based on $\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	λ θ	$10^{[-3,-2]}$ $10^{[-3,-2,-1,0]}$	1-hop DCNN with a modified diffusion based on $\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$
DCNN-$\mathbf{X}$-$\mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}$	λ θ	$10^{[-3,-2]}$ $10^{[-3,-2,-1,0]}$	1-hop DCNN with a modified diffusion based on $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}$
DCNN-$\mathbf{X}$-$\mathbf{K}_{\mathrm{h,g}}^{\mathrm{cBoP}}$	λ θ	$10^{[-3,-2]}$ $10^{[-3,-2,-1,0]}$	1-hop DCNN with a modified diffusion based on $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{cBoP}}$

Tab. A.5.: Summary of the DCNN-$\mathbf{X}$-$\mathbf{G}$ models with their acronym, their list of hyperparameters and a short description. λ is the initial learning rate and θ is the inverse temperature.

Overall, from the Borda ranking in Table A.7, we observe that all versions of DCNN-$\mathbf{X}$-$\mathbf{G}$ are less efficient than the DCNN-$\mathbf{X}$-$\mathbf{P}$. One reason that may explain this low efficiency is the reduction in the number of degrees of freedom. Indeed, this model has less learnable parameters and cannot give more or less weight for features aggregated by performing 1 hop, 2 hops, etc. It is only able to give more or less weight for features aggregated by performing one hop based on the global similarities between the nodes.
Only the DCNN-$\mathbf{X}$-$\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$ reaches the same score as the original DCNN. Moreover, Table A.8 confirms that the model DCNN-$\mathbf{X}$-$\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$ is significantly different than those with $\mathbf{G} \in \{\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}, \mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}, \mathbf{K}_{\mathrm{h,g}}^{\mathrm{cBoP}}\}$ but not significantly different than the original DCNN.
In addition, by combining Tables A.7 and A.8, we see that the two kernels $\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$ and $\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$ work significantly better than the two Gaussian kernels $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{BoP}}$ and $\mathbf{K}_{\mathrm{h,g}}^{\mathrm{cBoP}}$.
As expected, the improved version DCNN-$\mathbf{X}$-$\mathbf{P}\mathbf{\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$ significantly beats all methods.
This discussion about the DCNN-$\mathbf{X}$-$\mathbf{G}$ has been placed in Appendix since it provides poor results face to the DCNN-$\mathbf{X}$-$\mathbf{P}$. Despite this, it served as a building block for the DCNN-$\mathbf{X}$-$\mathbf{PG}$.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
DCNN-$\mathbf{X}$-$\mathbf{P}$	75.12	78.75	75.31	85.01	81.6	96.99	94.85	**77.32**	83.42	61.36
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_h^{BoP}$	77.08	82.7	78.92	82.75	80.17	96.37	95.93	76.19	77.91	61.27
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_h^{cBoP}$	74.48	80.31	78.22	83.28	79.57	96.08	95.27	76.25	77.18	60.37
DCNN-$\mathbf{X}$-$\mathbf{K}_{h,g}^{BoP}$	71.96	78.35	78.04	84.21	78.3	95.1	93.57	69.32	62.48	50.37
DCNN-$\mathbf{X}$-$\mathbf{K}_{h,g}^{cBoP}$	72.13	82.33	77.68	81.78	77.69	92.44	92.09	69.29	62.02	51.31
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_h^{BoP}$	**78.02**	**86.13**	**80.88**	**87.54**	**81.82**	**98.36**	**97.24**	76.93	**85.31**	**64.42**

Tab. A.6.: Table containing performances of some DCNN-based methods on all datasets. Especially, the DCNN-$\mathbf{X}$-$\mathbf{G}$. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_h^{BoP}$	59
DCNN-$\mathbf{X}$-$\mathbf{P}$	41
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_h^{BoP}$	41
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_h^{cBoP}$	33
DCNN-$\mathbf{X}$-$\mathbf{K}_{h,g}^{BoP}$	20
DCNN-$\mathbf{X}$-$\mathbf{K}_{h,g}^{cBoP}$	16

Tab. A.7.: Borda ranking of some DCNN-based methods computed on Table A.6.

Method	Id	(1)	(2)	(3)	(4)	(5)	(6)
DCNN-$\mathbf{X}$-$\mathbf{P}$	(1)	1	0.878	0.241	**0.022**	**0.028**	**0.009**
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_h^{BoP}$	(2)	0.878	1	**0.017**	**0.013**	**0.005**	**0.005**
DCNN-$\mathbf{X}$-$\mathbf{\Pi}_h^{cBoP}$	(3)	0.241	**0.017**	1	**0.009**	**0.017**	**0.005**
DCNN-$\mathbf{X}$-$\mathbf{K}_{h,g}^{BoP}$	(4)	**0.022**	**0.013**	**0.009**	1	0.333	**0.005**
DCNN-$\mathbf{X}$-$\mathbf{K}_{h,g}^{cBoP}$	(5)	**0.028**	**0.005**	**0.017**	0.333	1	**0.005**
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_h^{BoP}$	(6)	**0.009**	**0.005**	**0.005**	**0.005**	**0.005**	1

Tab. A.8.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between some DCNN-based methods. The p-values are computed on Table A.6 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

A.3 DCNN-**X**-**PR**

The goal of this section is to check if the use of a kernel for the matrix $\mathbf{G}$ in the DCNN-$\mathbf{X}$-$\mathbf{PG}$ is relevant. We compare the models using a kernel with a new model denoted by DCNN-$\mathbf{X}$-$\mathbf{PR}$ where the matrix $\mathbf{G}$ is a randomly generated matrix and contains values between 0 and 1.

Table A.9 indicates that the DCNN-$\mathbf{X}$-$\mathbf{PR}$ has performances close to the other models on the $\mathbf{A}$-driven datasets but not on $\mathbf{X}$-driven datasets. Table A.10 and Table A.11 confirm that it is significantly less effective than the others. Hence, the use of a kernel is therefore relevant, especially on $\mathbf{X}$-driven datasets.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	**78.02**	**86.13**	80.88	87.54	81.82	**98.36**	**97.24**	76.93	**85.31**	64.42
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	77.05	85.89	**81.25**	**87.71**	**82.53**	98.14	97.06	76.73	85.24	**64.62**
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{BoP}}$	75.54	85.67	80.67	87.25	81.78	97.75	97.2	77.39	85.2	62.77
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{cBoP}}$	76.88	85.54	80.5	87.55	82.13	97.71	97.07	**77.49**	**85.31**	62.75
DCNN-$\mathbf{X}$-$\mathbf{PR}$	72	80.2	75.36	85.79	80.61	96.45	95.98	77.2	84.06	61.32

Tab. A.9.: Table containing performances of DCNN-$\mathbf{X}$-$\mathbf{PG}$ methods on all datasets. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	40
DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	38
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{cBoP}}$	31
DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{BoP}}$	28
DCNN-$\mathbf{X}$-$\mathbf{PR}$	12

Tab. A.10.: Borda ranking of the DCNN-$\mathbf{X}$-$\mathbf{PG}$ methods computed on Table A.9

Method	DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{BoP}}$	DCNN-$\mathbf{X}$-$\mathbf{P\Pi}_{\mathrm{h}}^{\mathrm{cBoP}}$	DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{BoP}}$	DCNN-$\mathbf{X}$-$\mathbf{PK}_{\mathrm{h,g}}^{\mathrm{cBoP}}$
DCNN-$\mathbf{X}$-$\mathbf{PR}$	**0.007**	**0.007**	**0.005**	**0.005**

Tab. A.11.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between the DCNN-$\mathbf{X}$-$\mathbf{PR}$ and the other DCNN-$\mathbf{X}$-$\mathbf{PG}$ methods. The p-values are computed on Table A.9. p-values below the significance level (5%) are highlighted in bold.

A.4 Reg-DCNN-**X**-**P**

In this section, we analyse in details the extension of the DCNN-$\mathbf{X}$-$\mathbf{P}$ when a regularisation term is added. More precisely, we observe how the initialisation and the norm used impact the model. The lower index of the prefix "Reg" indicates the initialisation used while the upper index the norm used (1 for the L_1-norm and 2 for the square of L_2-norm).

From the Borda ranking in Table A.13, we can clearly see that the initialisation does not matter since the scores are similar. This is why for simplicity, it is advisable to use the simplest model, namely the Trivial model. This can be explained by the fact that during the first epoch, the model is still very poorly adjusted and that at the end of this epoch, its prediction is still relatively random. Whatever the initialisation, the model is in a similar situation to start the following epochs.

Moreover, the square of the L_2-norm seems to perform a little better than the L_1-norm. However, the Wilcoxon signed-rank test in Table A.14 does not contain a

glaring significant difference between the two norms, only a slight trend for the L_2-norm. The norm used in this model therefore seems to constitute a more important factor than the initialisation on the behavior of the model.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
$\text{Reg}^1_{\text{Trivial}}$-DCNN-**X**-**P**	72.02	81.14	77.94	84.05	**82.37**	98.23	97.25	77.21	84.02	61.94
$\text{Reg}^1_{\text{SVM}}$-DCNN-**X**-**P**	71.45	81.49	**78.8**	82.7	82.18	98.12	97.25	77.23	84.41	61.36
$\text{Reg}^1_{\text{AutoSVM}}$-DCNN-**X**-**P**	72.7	80.53	75.96	83.87	82.02	98.18	**97.26**	77.29	84.65	62.29
$\text{Reg}^2_{\text{Trivial}}$-DCNN-**X**-**P**	**74.46**	82.68	78.52	82.82	**82.37**	98.2	96.94	77.4	85.67	**62.83**
$\text{Reg}^2_{\text{SVM}}$-DCNN-**X**-**P**	73.38	**82.9**	77.66	84.45	82.03	**98.33**	97.19	77.21	**85.68**	62.34
$\text{Reg}^2_{\text{AutoSVM}}$-DCNN-**X**-**P**	72.73	85.34	78.04	**85.01**	82.02	98.15	**97.26**	**77.41**	85.04	62.78

Tab. A.12.: Table containing performances of Reg-DCNN-**X**-**P** methods on all datasets. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
$\text{Reg}^2_{\text{Trivial}}$-DCNN-**X**-**P**	43
$\text{Reg}^2_{\text{AutoSVM}}$-DCNN-**X**-**P**	43
$\text{Reg}^2_{\text{SVM}}$-DCNN-**X**-**P**	39
$\text{Reg}^1_{\text{Trivial}}$-DCNN-**X**-**P**	28
$\text{Reg}^1_{\text{AutoSVM}}$-DCNN-**X**-**P**	27
$\text{Reg}^1_{\text{SVM}}$-DCNN-**X**-**P**	25

Tab. A.13.: Borda ranking of the Reg-DCNN-**X**-**P** methods computed on Table A.12

Method	Id	(1)	(2)	(3)	(4)	(5)	(6)
$\text{Reg}^1_{\text{Trivial}}$-DCNN-**X**-**P**	(1)	1	0.594	0.959	0.139	0.086	**0.037**
$\text{Reg}^1_{\text{SVM}}$-DCNN-**X**-**P**	(2)	0.594	1	0.507	0.093	0.114	0.059
$\text{Reg}^1_{\text{AutoSVM}}$-DCNN-**X**-**P**	(3)	0.959	0.507	1	0.074	**0.037**	**0.021**
$\text{Reg}^2_{\text{Trivial}}$-DCNN-**X**-**P**	(4)	0.139	0.093	0.074	1	0.575	0.721
$\text{Reg}^2_{\text{SVM}}$-DCNN-**X**-**P**	(5)	0.086	0.114	**0.037**	0.575	1	0.508
$\text{Reg}^2_{\text{AutoSVM}}$-DCNN-**X**-**P**	(6)	**0.037**	0.059	**0.021**	0.721	0.508	1

Tab. A.14.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between the Reg-DCNN-**X**-**P** methods. The p-values are computed on Table A.12 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

A.5 Auto-DCNN-**X**-**P**

The Auto-DCNN-**X**-**P** is trained on the same hyperparameters than the DCNN-**X**-**P** in Table 3.1. Different initialisations have been tested. They are indicated in subscript

below the prefix "Auto".

The Borda ranking in Table A.16 shows that no Auto-DCNN-**X**-**P** is able to beat the original model and that initialisation does not seem important since they have similar scores. Table A.17 suggests that despite these observations, no significant difference is present.

The Auto-SVM-**X** builds a new SVM each time the autocovariates are updated. Conversely, in this new version of DCNN, the autocovariates are modified during the learning of the same model. One possible reason for this small performance degradation is that the learnable parameters in the network are no longer adapted to the autocovariates of the following epoch. The results are therefore not reported in the main discussion. However, completely imitating the Auto-SVM-**X** by training a new DCNN-**X**-**P** each time the autocovariates are updated is possible and could, like for the SVM, increase its performances, but the computational cost would be very high.

Method	*DB1*	*DB2*	*DB3*	*DB4*	*DB5*	*DB6*	*DB7*	*DB8*	*DB9*	*DB10*
DCNN-**X**-**P**	**75.12**	78.75	75.31	**85.01**	81.6	**96.99**	**94.85**	**77.32**	**83.42**	61.36
$\text{Auto}_{\text{Trivial}}$-DCNN-**X**-**P**	73.65	**83.16**	78.07	84.99	81.04	94.5	94.21	76.32	83.04	63.62
Auto_{SVM}-DCNN-**X**-**P**	72.33	78.59	**80.79**	84.72	**82.33**	92.95	92.59	76.56	81.87	**64.26**
$\text{Auto}_{\text{AutoSVM}}$-DCNN-**X**-**P**	71.03	81.71	80.77	83.41	81.9	95.73	94.53	76.82	82.69	63.59

Tab. A.15.: Table containing performances of Auto-DCNN-**X**-**P** on all datasets. Each performance corresponds to the accuracy obtained by a method on a dataset by following the process described in Section 4.2. For each dataset, the best accuracy is highlighted in bold.

Method	Score
DCNN-**X**-**P**	30
$\text{Auto}_{\text{Trivial}}$-DCNN-**X**-**P**	24
$\text{Auto}_{\text{AutoSVM}}$-DCNN-**X**-**P**	24
Auto_{SVM}-DCNN-**X**-**P**	22

Tab. A.16.: Borda ranking of Auto-DCNN-**X**-**P** methods computed on Table A.15.

Method	Id	(1)	(2)	(3)	(4)
DCNN-**X**-**P**	(1)	1	0.878	0.508	0.878
$\text{Auto}_{\text{Trivial}}$-DCNN-**X**-**P**	(2)	0.878	1	0.333	0.959
Auto_{SVM}-DCNN-**X**-**P**	(3)	0.508	0.333	1	0.508
$\text{Auto}_{\text{AutoSVM}}$-DCNN-**X**-**P**	(4)	0.878	0.959	0.508	1

Tab. A.17.: Table containing the p-values coming from a pairwise Wilcoxon signed-rank test between some Auto-DCNN-**X**-**P** methods. The p-values are computed on Table A.15 and allow to more precisely measure the relative performances between two methods. p-values below the significance level (5%) are highlighted in bold.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**