2015

Approximation contexts in addressing graph data structures

Nguyen Van Tuc
University of Wollongong

APPROXIMATION CONTEXTS IN ADDRESSING GRAPH DATA STRUCTURES

A Dissertation Submitted in Fulfilment of
the Requirements for the Award of the Degree of

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Nguyen Van Tuc

School of Computing and Information Technology
Faculty of Engineering and Information Sciences

2015

CERTIFICATION

I, **Nguyen Van Tuc**, declare that this thesis, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

(Signature Required)

Nguyen Van Tuc
30 July 2015

Dedicated to

My family

Table of Contents

List of Tables

List of Figures

ABSTRACT

While the application of machine learning algorithms to practical problems has been expanded from fixed sized input data to sequences, trees or graphs input data, the composition of learning system has developed from a single model to integrated ones. Recent advances in graph based learning algorithms include: the SOMSD (Self Organizing Map for Structured Data), PMGraphSOM (Probability Measure Graph Self Organizing Map), GNN (Graph Neural Network) and GLSVM (Graph Laplacian Support Vector Machine). A main motivation of this thesis is to investigate if such algorithms, whether by themselves individually or modified, or in various combinations, would provide better performance over the more traditional artificial neural networks or kernel machine methods on some practical challenging problems. More succinctly, this thesis seeks to answer the main research question: when or under what conditions/contexts could graph based models be adjusted and tailored to be most efficacious in terms of predictive or classification performance on some challenging practical problems? There emerges a range of sub-questions including: how do we craft an effective neural learning system which can be an integration of several graph and non-graph based models? Integration of various graph based and non graph based kernel machine algorithms; enhancing the capability of the integrated model in working with challenging problems; tackling the problem of long term dependency issues which aggravate the performance of layer-wise graph based neural systems. This thesis will answer these questions.

Recent research on multiple staged learning models has demonstrated the efficacy of multiple layers of alternating unsupervised and supervised learning approaches. This underlies the very successful front-end feature extraction techniques in deep neural networks. However much exploration is still possible with the investigation of the number of layers required, and the types of unsupervised or supervised learning models which should be used. Such issues have not been considered so far, when the underlying input data structure is in the form of a graph. We will explore empirically the capabilities of models of increasing complexities, the combination of the unsupervised learning algorithms, SOM, or PMGraphSOM, with or without a cascade connection with a multilayer perceptron, and with or without being followed by multiple layers of GNN. Such studies explore the effects of including or ignoring context. A parallel study involving kernel machines with or without graph inputs has also been conducted empirically.

Moreover, some graph learning tasks sometimes contain difficult aspects including high dimensional inputs, imbalanced class distribution of output labels and path dependencies. It is common practice that only one such aspect will be addressed at any one time. This thesis introduces an integrated learning framework containing three functions for solving these three problems by assuming these three effects are largely independent of one another. In particular, a Lasso-type regularization is used for feature selection to handle the high dimensional input situation, a non-uniform sampling method is designed to handle the label imbalance issue, and a deep learning strategy is used to handle the path dependency issue. This thesis evaluates the proposed ideas on several challenging real world problems, including the UK 2006 and UK2007 web spam detection datasets and a large scale XML document classification problem, the INEX 2008 dataset. It is shown that the proposed approaches obtain results equal to or better than the state-of-the-art performances obtained by other techniques in the literature.

The aforementioned integrated models are also applied to another real world problem, namely predicting the activity type of preschool and school children from wearable accelerometer measurements attached to various parts of their bodies. Our research is focused on addressing the long term dependency problem, as various recurrent neural networks are applied to solve this problem. The results obtained permit some comparisons among the methods deployed, as there is no ground truth information available. The conclusion which can be made is the influence of age on the consumption level of energy.

The contributions of this thesis cover a range of machine learning algorithms. These include: insight into the integration of machine learning methods for robust graph based models, and dealing with the long term dependency issue of the layer-wise neural network model and kernel machines. Extensive experiments, either on benchmark datasets or real world problems are conducted, and comparisons are made with other results where they exist to make concrete statements.

KEYWORDS: Graph, Neural networks, Kernel machines, Long term dependency, Deep learning, Hierarchical or layer architectures.

List of Publications

- N. V. Tuc, A. C. Tsoi, and M. Hagenbuchner, "Cost-sensitive cascade Graph Neural Networks". *In Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013, pp. 527-532. (This publication forms the part of Chapter 7)

- M. Hagenbuchner, D. P. Cliff, S. G. Trost, N. V. Tuc, and G. E. Peoples, "Prediction of Activity Type in Preschool Children using Machine Learning Techniques". *Journal of Science and Medicine in Sport*, vol. 18, no. 4, pp. 426-431, 2015. (This publication forms the part of chapter 8)

- N. V. Tuc, M. Hagenbuchner, A. C. Tsoi, and S. Zhang, "On the effects of incorporating input graph topology in neural networks". Submitting to *IEEE Transactions on Neural network and learning systems*. (From materials contained in Chapter 5)

- N. V. Tuc, M. Hagenbuchner, and A. C. Tsoi, "On the effects of encoding structural topology in kernel machines". Submitting to *IEEE Transactions on Neural network and learning systems*. (From materials contained in Chapter 6)

- N. V. Tuc, M. Hagenbuchner, and A. C. Tsoi, "A hierarchical neural network model for graph learning". Submitting to *Machine learning*. (From materials contained in Chapter 7)

Acknowledgements

I would firstly like to express my deepest gratitude to A. Prof Markus Hagenbuchner for being my supervisor during my PhD study. His consistent encouragement, guidance and direction is incomparable. It has been a great honour for me to study under Markus's supervision. He always provides very detailed and constructive instructions on my work. Without his continuous, meticulous directions and the uncountable-hours of work, I would not have been able to conquer different hurdles during my study. I have been deeply impressed by Markus's friendliness, politeness and endurance from my initial research program. And I am looking forward to working with Markus in the future.

I would also express my special appreciation to my co-supervisor Prof Ah Chung Tsoi for his massive help during my research. Ah Chung Tsoi selflessly shared his strong and unique research vision with me and helped me with many brilliant research ideas. His inspiration and encouragement will never be forgotten. This thesis would never be the same without the tireless working and insightful contributions from Ah Chung. In addition, I appreciate to have been on the receiving side of helpful suggestions and comments from other researchers. They are Prof. Franco Scarselli, who is very friendly and supportive, and Dr. ShuJia Zhang who provided me with such informative advice at the beginning stage of my study, Dr. Milly Kc, and many other researchers I had the pleasure to collaborate with. I also would like to take this opportunity to thank Dr. Zhiquan Zhou who was the supervisor of my summer project that encouraged me to follow this research topic.

Thirdly, I acknowledge the financial support received in form of a scholarship for my studis from the ARC discovery project grant (DP110103376) that was awarded to A. Prof Markus Hagenbuchner and Prof. Ah Chung Tsoi. I especially appreciate the technical support from the Information Technology Services staff at the University of Wollongong, allowing me to access the high performance computer clusters which were essential for my research experimental needs. They were always willingly available to help with any problems with the internet connection and printing resources.

Last but not least, I likewise owe an eternal debt to my family for their unconditional love. I would like to thank my parents for their continuous encouragement both financially and emotionally during the hardness of my research. And certainly, life seems meaningless without the whole-hearted support of my wife Trang Thi Nguyen and without my adorable little daughter Anh Thu Nguyen. They endlessly fill me up with laughter and joy which is as invaluable as aspirin in helping through the more stressful moments. Finally, my gratitude also extends to my friends who have been assisting me in time of needs.

List of Abbreviations

AEE	Activity Energy Expenditure
AMB	Absolute Mean Bias
(A)NNs	(Artificial) Neural Networks
AUC	Area Under Curve
B-GNN	Boosting Graph Neural Network
BPTS	Back Propagation Through Structure
BPTT	Back Propagation Through Time
C	Content based features
CSGNN	Cost Sensitive Graph Neural Network
DAG	Directed acyclic graphs
EE	Energy Expenditure
GLSVM	Graph Laplacian Support Vector Machine
GNN	Graph Neural Network
GNN2	Graph Neural Network for graph of graphs learning
GoG	Graph of Graphs
HG	Host Graph
HNBD	Hybrid Negative Binomial Distribution
INEX	INitiative for the Evaluation of XML Retrieval
KKM	Kernel K-Means
L-GNN	Layered Graph Neural Network
LSTM	Long Short Term Memory

MB	Mean Bias
MET	Metabolic Equivalent
MLPs	Multilayer Perceptrons
MSGNN	Multiple Staged Graph Neural Network
MSVM	Multi-kernel Support Vector Machine
NARX	Nonlinear Autoregressive network with exogenous inputs
NBD	Negative Binomial Distribution
PA	Physical Activity
PMGraphSOM	Probability Mapping Graph Self Organizing Map
RL	Raw Links based features
RMSE	Root Mean Squared Error
SKC	Spectral Kernel Clustering
SOM	Self Organizing Map
SOMSD	Self Organizing Map for Structured Data
SSE	Sum Square Error
SSNN	Stable State Neural Network
SVM	Support Vector Machine

Notation

In this thesis, the mathematic representation is uniformly presented as follows: Lowercase script letters like n are used to indicate scalars and constants. Parameters of a learning model are shown by lowercase Greek letters such as γ. Sets and matrices are indicated by upper case letters, e.g., M. Calligraphic letters like $\mathcal{G}$, $\mathcal{N}$ and $\mathcal{E}$ are respectively used to represent graphs, a set of nodes, and a set of edges. Letters used in combination with brackets such as $h(x, y)$ denote functions. Typical examples are given below:

$x(t)$	The parameter x depends on time t.
$F_w(x, y)$	The function F takes a vector x and y as its arguments, and depends on the variable w.
$M = KL$	The multiplication of the two matrices, or the dot product.
$n = \|d\|$	n is denoted the cardinality of vector d.
$n = \|\|m\|\|$	Variable n takes the positive value of m.
$x = (x_1, x_2, ..., x_n)$	x is a vector containing n elements.
$n \in \{10, 15, 20, 24\}$	A number n can take a value from a set of four elements.

Software Usage

This thesis was completed using LaTeX for Linux version 3.14159265 ©1999 by D.E. Knuth with the Kile user-friendly editor. Some of the images were created in the format of EPS using LibreOffice 4.1 version ©2007 from the Free Software Foundation. Some other EPS format files were produced with gnuplot v4.6.5 ©2004 by Thomas Williams and Colin Kelley, or by xfig version 3.2 patch-level 2 ©1989-1998 by Brian V. Smith, and 1991 by Paul King.

Hardware Environment

The work presented in this thesis includes results from a wide range of experiments on a number of neural networks as well as kernel methods. Hardware resources which were utilized for the experiments are as follows:

No	Hardware		OS	Number of cores	Core speed	Usage years
	Category	Type				
1	Workstation	Intel	Linux	2	2.1	3
2	Workstation	Intel	Linux	2	2.4	1.5
3	Workstation	Intel	Linux	4	3.5	2
4	Supercomputer	SGI	Linux	9	2.1	3
5	Cluster	AMD	Scientific Linux	240	1.5	3
6	Workstation	Intel	Linux	7	2.1	3

Core speed is an approximate value relative to a 1GHz single-core Intel Pentium. The core speed is approximate since the actual speed of a machine dependents on the amount and speed of RAM, the speeds of permanent storage devices such as hard-discs, the number of running tasks, and other factors.

Chapter 1

Introduction

A common assumption of algorithms in artificial neural networks and kernel machines is that the input is available in vectorial form. These algorithms thus assume that the input data is a set of mutually independent vectors. However, there are many real world learning problems such as in image processing, molecular chemistry, World Wide Web, document classification, logo recognition, video processing, and many more, in which the inputs are more appropriately modelled as a graph [1].

Working with graphs rather than with vectors is more challenging as graph inputs relate the feature vector at each node with those other nodes in its neighborhood, while working with vectors, they are assumed to be mutually independent of their neighborhood. Hence, machine learning algorithms for input graph processing are more complex [1, 2, 3, 4, 5]. These models share the common characteristics that they can model dependencies of data and hence, avoid the possible loss of information that can occur when squashing data structures into a vectorial form during pre-processing. However for practical processing, it is far

[1] In this thesis we are using a number of terms interchangeably: graph input structure, input graph structure, graph processing. By and large, these refer to the same concept: the inputs to machine learning models are assumed to be in the form of a graph, consisting of nodes and links; each node has associated feature vectors, which are sometimes called labels. The links could also be endowed with feature vectors, although in this thesis invariably we will only consider the simpler situation, where the links may be weighted with a connection weight.

more convenient to maintain a fixed sized vector for input to the more traditional vectorial input machine learning algorithms.

In the vectorial input case, there are many neural network and kernel machine models. The behaviours of some of these models are well understood [6]. For example, in the case of unsupervised learning, we have learning vector quantization, self organizing map [7], neural auto-associator [6], spectral clustering [8], manifold embedding, linear support vector machines, and simple kernel machines [6, 9]. It is known that these models do not have universal approximator properties, implying that they cannot approximate an input to a cluster mapping arbitrarily closely. This is expected, as without target information, the data somehow organizes itself into clusters, or groups, in which the intra-group distances are smaller than the inter-group distances. But the issue of the number of clusters is notoriously difficult, as it is not known how the data could be grouped together so that some distance criterion can be satisfied. Hence, as expected, the clusters cannot approximate the underlying clusters of the data arbitrarily closely, as such underlying clusters are ill-posed. On the other hand, for the supervised learning paradigm, the most popular example is the multiple hidden layered feedforward neural network, alternatively known as mulitlayer perceptron (MLP). In such a situation, it is known that the MLP can approximate an underlying nonlinear input output mapping arbitrarily closely [6]. The case of kernel machines is less clear. It is known that kernel machines can be formed with various kernels e.g. radial basis function kernel, polynomial kernel, in which some of these common kernels are universal approximators [6].

For input graphs, there exist some neural network and kernel machine models which can handle the input graph structures. In unsupervised learning paradigm, there are various extensions of the self organizing map to handle graphs of various complexity, e.g., trees, graphs without self loops [2, 10]. The most recent generation PMGraphSOM can handle graphs with self loops [3]. It is also not expected that these various extensions of SOM in the graph input cases would have universal approximation properties. On the other hand,

there are extensions of the multiple layer perceptron to handle graph inputs, called a graph neural networks [4, 11, 12, 13, 14, 15, 16]. These models possess the universal approximation property, in that they are capable of approximating nonlinear input graphs to vectorial outputs to an arbitrary close value.

Recently, there has been an emerging trend in combining several neural network models so as to improve prediction results. The most common situation is the composition of unsupervised and supervised neural network models as introduced in [17, 18]. In [18], it was shown that combining models does provide better performance than if only one neural network model is used. However, these combined neural network models appear to be *ad hoc*. There have not been any systematic studies devoted to investigate the effects of combining these neural network models, or in general, machine learning modules (which include kernel machines). Another situation is observed in the deep neural network where composite or multiple layers of one or different neural models are stacked together [19, 20]. In this case, the engineering of a suitable architecture like the number of layers, the size of each layer, for a particular problem appear to be somewhat of a "black art" [19, 20, 21, 22, 23, 24, 25]. One just brings several neural network modules together [2], as a result, the performance of the combined model is better than that obtained using a single neural network model.

We desire to understand this issue, namely what rationale does one use to combine neural network modules?, this would be a great motivation to pursue in this thesis. Indeed this thesis investigates the following issues:

- What guidelines does one use in combining different machine learning modules together? Here by machine learning modules, we mean unsupervised and supervised learning modules for vectorial inputs in both neural networks (e.g., self organizing map, multilayer perceptron), or kernel machines (linear support vector machines, ker-

[2] This is no disrespect to those who work in this area. If one considers the way that multiple layers are being positioned, with different sizes in deep neural networks, there does not appear to be any in-principle fashion in which these are put together.

nel machines with different kernels); and for graph inputs (PMGraphSOM in unsupervised learning, GNN in supervised learning, graph Laplacian for kernel machines) [3].

- As a corollary, this thesis strives to provide an answer to the following question: given that one has a combined learning model using various modules, what would be the effect of adding one module on the performance of the combined network in some practical problems?

- As a corollary to the above questions, another question arises: could one improve the performance of these combined models, if the practical problem comes with some of the following characteristics: high dimensional feature vectors associated with each node, imbalanced output class labels, long term dependency problem (which is known to plague supervised learning of neural networks with multiple hidden layers)?

- In a practical problem which is particularly prone to long term dependency issues, the prediction of activity types of preschool children and school children, based on measurements made by accelerometers wearable at various parts of their bodies, particular effective ways of overcoming the long term dependency issue will be investigated.

In this thesis, one would notice a very general principle: a layerwise approach to deal with different problems. The layerwise principle is inspired by the deep neural network approach, though it is not the same concept. In a layerwise approach, we merely deal with a particular aspect of the practical problem, e.g., the need for unsupervised learning of graph inputs, the need to resolve the recursiveness which might exist in the training data, the need to deal with the prediction of the output labels (where the output labels only extend to *some* of the outputs), etc. In the deep neural network technique, it is assumed that there is a frontend, used to extract features from the raw inputs, or very little preprocessed inputs (e.g.,

[3] Theoretically one could combine both neural network modules and kernel machine modules together, though this is not done in this thesis.

speech signals). These features once extracted implicitly would be passed to a neural network classifier [22]. This would be different from the layerwise approach, which is proposed here. Moreover the phrase "layerwise" has also been used in the training of multiple hidden layer feedforward neural networks previously [19, 20]. However, its usage here in this thesis would be following the lead of its usage in this sense in [18]. It appears that its usage in this sense is for the first time extensively explored in this thesis.

From the parallel investigation of the layerwise approach to combine neural network models, and kernel machine models, one could gain considerable insight into the working of these models, in particular how they relate to two particular practical datasets, viz., the UK2006 and UK2007 web spam detection datasets. The layerwise approach is deployed for both the neural network and kernel machine models to the same datasets. Then their results will be compared, and interpretation sought of the results provided thereafter.

The investigation on how such integrated models can handle commonly encountered issues in practical problems, like high dimensionality of the feature vectors, the imbalance of the output label distributions, and the long term dependency issues, follow rather more traditional lines in handling such problems, e.g., using some kind of regularization technique to reduce the dimension of the feature vectors, the deployment of a non-uniform sampling technique to sample the output class labels and using them in the training and testing procedures, and the use of techniques specifically designed to overcome long term dependency issues in neural network models. The handling of the long term dependency issue in neural networks deserves some special mention, as it is commonly encountered in training multiple layer feedforward neural networks. The underlying idea of long short term memory [26] technique to overcome the long term dependency issue is investigated in this thesis.

As implicitly obvious in the discussions here, this thesis investigates these problems using empirical means, i.e., by applying the ideas to practical challenging problems. There is some theoretical development where necessary to "glue" the techniques together. The

strength of this thesis lies in the deployment of an in-principle manner: a layerwise approach to combine machine learning modules into an integrated model, and the interpretation of the results obtained.

The remaining sections of this chapter are organized as follows: Section 1.1 presents the contributions of this thesis. The benefit of this research is summarized in Section 1.2. Finally, the thesis outline is given in Section 1.3.

1.1 Contributions of the thesis

The contributions of this thesis include:

- Investigation of a layerwise approach to combine machine learning modules into an integrated model. This can be divided into two separate parallel though related studies:
  - Combination of the neural network modules into an integrated model. This is denoted as Model A.
  - Combination of the kernel machine modules into an integrated model. This is denoted as Model B.

  The relationships between the classification accuracies and the architectural complexities in both Model A and Model B are obtained. This is based on applying both models to the same two datasets: UK2006 and UK 2007 web spam detection datasets.

  The interpretation of the results obtained leads to a conclusion: in the UK 2006 web spam dataset, it appears that there is more evidence of link farms being used as a spamming device by the spamsters. On the other hand, in the UK2007 web spam dataset, it appears that these link farms were replaced by more sophisticated spamming techniques, e.g., content based spamming.

- Dealing with the following commonly encountered issues in practical learning problems: high dimensionality of the input feature vectors, imbalance of the output class label distributions, and long term dependency issues. In each case, it is shown that if the issue is yaken into consideration then improved results can be obtained.

- In the case of long term dependency, this thesis investigates such occurrence in a practical prediction problem: predicting the activity types of preschool children and school children from measurements obtained from wearable accelerometers attached to various parts of their bodies. This is a particularly nasty problem because there are many stages for the backprop errors to go through beforethey reach the input end, and hence the backprop errors could become vanishingly small after passing through a number of hidden layers. It is shown that using methods specifically designed to overcome long term dependency issues can assist in overcoming these, thereby obtaining quite reasonable results.

1.2 Research benefits

Many real-world problems could be easily modeled by either sequences, trees or graphs in general. The richer information representation in practice would help the prediction task to achieve much better performance. However, graph representation is only the first stage over the learning process. An appropriate learning model being designed could bring significant benefits. A well designed prediction model should not only express the adaptability of the graph type of input, but can also exploit effectively the topological relations between graph elements. Several main benefits would be attained from this research and they are enumerated as follows:

1. A practical demonstration would be shown that graph based modeling methods would perform at least equivalent to traditional learning approaches. This is interesting since

numerous real-life applications could be effectively represented as sequences, trees or graphs. Typical examples include modeling body movements for physical activity prediction, modeling an image/video as graphs or a sequence of graphs, document categorization, modeling molecules, DNA, genes in the fields of Chemistry and Biology, and many others.

2. The context of integrated architectures would play important roles in prediction outcome. In particular, the research clarifies the specific conditions under which the performance of model integration could be promoted. It is found that placing an unsupervised pretrained model prior to one or a cascade of supervised learning models is quantitatively beneficial. Such a context is consistent with the underlying idea of the deep learning algorithm. Another situation is that layering a cascade of the same learning modules, or applying the long short term memory idea may help to alleviate the detrimental effects of the long term dependency problem in terms of neural network processing, and enhance the learning performance accordingly. Layerwise applications in kernel learning are likewise proven to be a fruitful area of research.

3. Data-driven processes such as feature selection and imbalance class treatment are not at all trivial since these may significantly influence model prediction accuracy. Proper understanding of the effects of the curse of dimensionality and the imbalanced data problem may help one to better prepare for each application and to develop an effective model.

1.3 Thesis structure

The thesis is organized as follows:

Chapter 1: Overview of the research, which include the underlying ideas of the research topic, the thesis contributions, the benefit of the research, and the outline of the thesis.

Chapter 2: Background on the traditional machine learning models and several approaches to improve the model's learning performance.

Chapter 3: Literature review on graph based learning, with recent solutions to long term dependencies in the graph domain.

Chapter 4: A detailed description of several real-world problems and benchmark datasets related to graph data structures and evaluation approaches.

Chapter 5: A comparison of the performances of the traditional neural networks and graph based neural network models. Various integrated models are introduced to seek the conditions under which the learning system can be promoted.

Chapter 6: Addressing similar questions arising in Chapter 5, but with kernel machines instead of neural networks.

Chapter 7: A proposed hierarchical machine learning technique to deal with difficult problems encompassing high input dimensionality, highly imbalanced class distribution and the remote path dependency problem.

Chapter 8: Research in alleviating the long term dependency in the real world physical activity classification problem.

Chapter 9: Summary of the findings of the research, the limitations, and some suggestions for future work.

Chapter 2

Traditional machine learning algorithms

2.1 Introduction

Artificial neural networks (ANNs) have a successful history in solving machine learning problems [6, 7]. ANNs are algorithms which simulate the ability of a biological brain to learn from sensory inputs. The enormous interest in developing suitable algorithms has resulted in a remarkable number of distinct learning algorithms that have found widespread applications in the real world. But since these types of algorithms are designed to mimic their biological counterparts and hence, the consequence is that ANNs can be implemented as a massive parallel system that takes advantage of modern multi-core CPU and multi-core GPU technology. In the past, however, due to the limitation of available computer processing technologies, ANNs were most commonly implemented as a non-parallel single process. Thus, despite the success of such systems, the computation time for training ANNs on large scale datasets was relatively long, rendering them impractical for general use. ANNs fell out of favor with researchers with the introduction of faster methods such as support vector machines (SVM) or kernel methods (KM) [9, 27]. SVMs and KMs quickly became the preferred choice in the field of machine learning. This trend has been reversed in recent years due to the development of multi-core CPUs and massively parallel GPU systems and

because of recent developments in deep learning [20, 24]. Such computing systems allow the implementation of ANNs as massive parallel systems which greatly speed up computations during the learning phase of an ANN. ANNs have since become more scalable than SVMs and KMs, and have been applied to very large learning problems (i.e. in image and video recognition problems, the WWW and social network based learning problems)[20, 24, 28].

The rest of this Chapter aims at providing some background on conventional, non-graph based machine learning methods such as the Self-organising map, Multilayer perceptron, Elman recurrent neural net, as well as the classic non-graph based kernel machine learning such as kernel K-means and SVM. Section 2.2 and Section 2.3 respectively explain these two learning algorithms. Several common or important adaptations of the learning methods are then described in Section 2.4. The learning methods presented in this chapter will serve as a basis for comparisons with more recent graph-based models which will be presented in Chapter 3.

2.2 Classic Artificial neural networks

2.2.1 Self-organising map

Teuvo Kohonen proposed the Self-organising (feature) map (SOM) in the 1980s [7]. The SOM is a type of artificial neural network that is trained by an unsupervised learning mechanism. The SOM is widely used for the purpose of clustering or for the projection and visualization of high dimensional signal spaces on low dimensional display spaces. The display space can be of any dimension though two dimensional display spaces are most common. The following assumes for simplicity that the display space is two dimensional. The display space is thus parametrized by a two dimensional grid. At the intersection of the grid points (denoted as nodes or neurons), it is assumed that there is a vector of weights (codebook vector). The dimension of the codebook vectors must match the dimension of

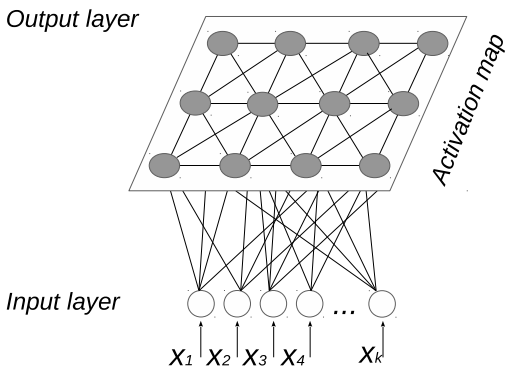

Figure 2.1: Common architecture of a SOM, the neighborhood relationship between the nodes on the map is shown. There are fully connected links between the k-dimensional inputs and the nodes. There are codebook vectors defined for each node. The output of network regarding each input value is the associated coordinates where the input is mapped on.

the input vectors. The main purpose is to enable these weights to approximate the training input, such that the vectors which are nearby each other in the high dimensional feature space will remain close when projected on the low dimensional display space. The SOM architecture is illustrated in Figure 2.1, which contains an input layer and an output layer or projection (activation) map.

The SOM training algorithm can be described as follows: Let x be one of the k dimensional input vector in a set of input training vectors. Each of the codebook vectors is defined as $c_i = (c_{i1}, c_{i2}, ..., c_{ik})^T \in \mathcal{R}^k$, $i = 1, 2, \ldots, n \times m$, where n and m are respectively the two dimensions of the grid, and T denotes the transpose of a vector. There is one codebook associated with each node in the feature map. c_i is usually initialized using random values. In practice, however a probability density function $p(x)$ of the input data is often used to control the value and range of the initial values for c_i. The number of clusters, in most of the cases, is defined by the user. Unlike other (supervised) learning algorithms, the SOM does not normally require the normalization of input vectors. For example, when clustering a set of bag-of-words vectors then the normalization process might be even harmful, because that may reduce the significance of some word or set of words.

The learning process consists of two major steps, the competitive step and the cooperative step:

Competitive step: The best matching codebook is identified. The best-matching unit (BMU) r is required to be the minimum value in the matching criterion when compared with the other codebooks in the SOM.

$$r = \arg\min_i \{d(x, c_i)\} \tag{2.1}$$

Here, r is denotes the winning neuron or BMU. The matching criteria is most commonly based on Euclidean distance $d = \|x - c_i\|^2$.

Cooperative step: The elements in a neighborhood set $\mathcal{N}_i$ of the BMU are modified by the following quantity:

$$\Delta c_i = \alpha(t) h(\Delta_{ri})(x - c_i), \tag{2.2}$$

where $\alpha(t)$ is a scalar learning rate factor which decreases with time t. Δ_{ri} defines the topological distance between c_r and c_i, and the smooth Gaussian kernel function is widely used as the neighborhood function $h(.)$. The neighborhood function defined as i.e. in Equation 2.3 controls the amount by which the codebooks in the neighbourhood are updated.

$$h(\Delta_{ri}) = \exp\Big(- \frac{\|l_r - l_i\|^2}{2\sigma^2(t)}\Big), \tag{2.3}$$

where l_r and l_i are the location vectors of winning neuron r and the i-th neuron in the lattice respectively. The parameter $\sigma(t)$ represents the kernel size. Both $\alpha(t)$ and $\sigma(t)$ are monotonically decreasing functions of time t.

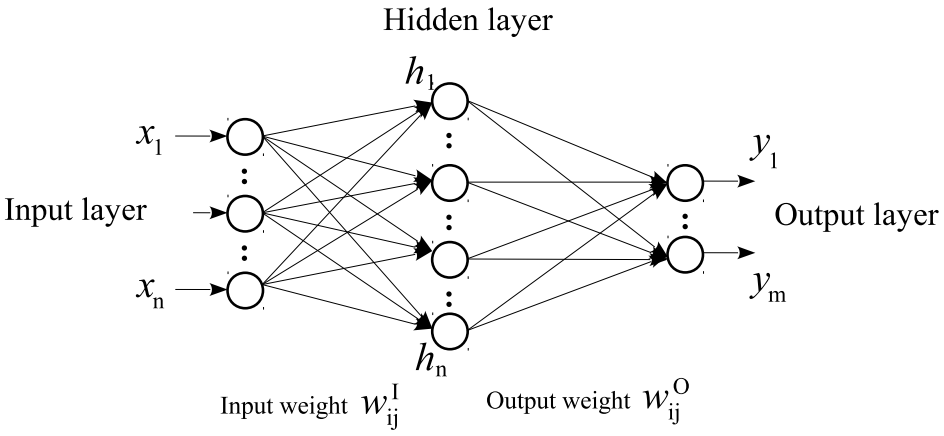

Figure 2.2: Common architecture of multilayer layer perceptron with three layers. n input neurons, u hidden neurons and m output neurons.

2.2.2 Multilayer perceptron (MLP)

The multilayer perceptron (MLP) is a supervised artificial neural network [6]. The MLP consists of a set of neurons which are organized in layers. The most common MLP architecture contains one input layer (n sensory inputs), one hidden layer (u neurons) and one output layer (m outputs) as demonstrated in Figure 2.2. The neurons in each layer are fully connected with neurons in adjacent layers. Each connection is weighted by an adjustable *weight* value. Hence, in practise, these connections are simply referred to as *weights*. Let W^I denote a $n \times u$ dimensional weight matrix containing the weights between the input and the hidden layers, and let W^O denote a $u \times m$ matrix containing the weights between the hidden and the output layer. The number of neurons in input and output layers are defined by the learning problem at hand, while the number of hidden neurons can be adjusted freely. The activation function (sometime called as the cost or error function) can be either linear or non-linear depending on whether the problem is linearly separable or not. Common non-linear functions include the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, or the hyperbolic function $tanh(x)$, where x is denoted the input vector.

The learning algorithm consists of two main stages called the *feedforward* stage and the

backward or error backpropagation stage. The algorithm can be used for both classification and regression tasks.

Feedforward stage: Given the training input x, the output of hidden unit is calculated as follow $h_j = \sigma(\sum_{i=1}^{n} w_{ij}^I x_i)$. Similarly, the (actual) output of the output layer is $y_k = \sigma(\sum_{j=1}^{u} w_{jk}^O h_j)$. If d is denoted the desired output (or the target value), the cost/error function then is defined based on the least mean squared error $E = \frac{1}{2}\sum_{k=1}^{m}(y_k - d_k)^2$.

Backpropagation stage: The error is propagated back through the architecture and the weights are updated into the negative direction of the gradient. The purpose of this step is to adjust the weights such that the network output becomes closer to the target value. The gradient with respect to each of the weights needs to be calculated accordingly. Starting with the weights in the output layer the gradient is computed as follows [1]:

$$\frac{\partial E}{\partial w_{jk}^O} = (d_k - y_k)y_k(1 - y_k)h_j \tag{2.4}$$

The gradients with respect to weights connecting the input and hidden neurons are computed based on the sum of all gradients of the outputs

$$\frac{\partial E}{\partial w_{ij}^I} = \sum_{k=1}^{m}(d_k - y_k)y_k(1 - y_k)\Big(\sum_{j=1}^{u} w_{jk}^O \frac{\partial h_j}{\partial w_{ij}^I}\Big) \tag{2.5}$$

where $\frac{\partial h_j}{\partial w_{ij}^I} = h_j(1 - h_j)x_i$ if the sigmoid activation is used. All connection weights are updated with the amount of change being $\Delta w = -\gamma\frac{\partial E}{\partial w}$ where $\gamma \in [0, 1]$ is the learning rate.

The two training steps are repeated for a number of iterations, and the stopping criterion could be defined based on a certain number of training iterations or the error threshold.

[1] Assuming that the linear activation function is used at the output neurons.

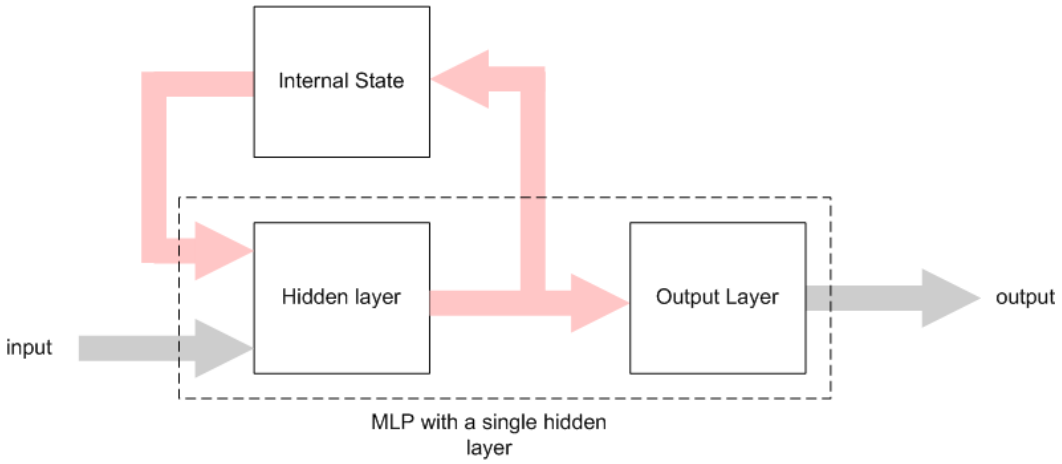

Figure 2.3: Time series Elman neural network model

2.2.3 Elman recurrent neural network

Traditionally, the Elman net [29] is one of the first recurrent neural network models introduced to address time series prediction. Since the learning problems contain time series data, Elman neural network model can be applied in order to improve the prediction performance. The Elman network is a supervised algorithm which can take as targets: values associated with a given input vector (i.e. for classification tasks), or subsequent values of a given data sequence (i.e. for regression, prediction tasks). The network is trained by a gradient descent method which differs from the MLP algorithm in that the gradient is being propagated back though the recurrent structure of the network. The recurrence depends on the length of the input sequences. Figure 2.3 illustrates the Elman network which extends the MLP architecture with the internal state layer (or context layer) that stores the last hidden state at every iteration. In practice, Elman networks, as well as most other recurrent NNs suffers from the long sequence learning problem, because the error signal becomes vanishingly small while being back-propagated through the sequence. However, the Elman net is better than traditional MLP in that it can learn the sequence data.

2.3 Kernel learning

This section will describe some basic kernel methods that have long gained significant interest in the machine learning community.

2.3.1 Kernel K-means

The KKM is known as a simple and effective unsupervised learning model. It has been adapted to large-scaled clustering problems [27]. The model aims to cluster a set of given data into k subsets $S_1, ..., S_k$ such that the within-cluster sum of squared distances are minimized. $\arg\min_{\mu} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|$, where μ_i is the mean of points or the center of cluster S_i. Generally, the algorithm contains two main steps. The first one is denoted as Assignment phase where each sample is assigned to a cluster considering the central locations of clusters. The second one is the Update phase, in which the centers μ are updated to be the centroids of those new clusters. The kernel K-means model is a later generation of the original K-means where the input data points are mapped non-linearly into a high-dimensional feature space via a pre-defined kernel function.

2.3.2 Support vector machine

In machine learning, the SVM [9] is one of the most well-known kernel methods. It is a supervised learning model. The underlying idea is that the SVM construct a hyper-plane to separate the data in high-dimensional space into binary categories if given a set of input feature vectors and associated class labels. The formulation of SVM is defined as a non-probabilistic linear classifier. It is however, efficiently adaptable to a non-linear classification by the application of kernel functions. The commonly used kernel function is the radial basis function $k(x_i, x_j) = exp\Big(-\frac{\|x_i - x_j\|}{2\sigma^2}\Big)$, where σ denotes the kernel function parameter. x_i and x_j are two arbitrary input samples. A common learning algorithm can be presented

briefly as follows:

Given a set of training examples and corresponding class labels, the data is defined as $\mathcal{D} = \{(x_i, y_i)|x_i \in R^m, y_i \in \{-1, 1\}\}_{i=1}^n$. The output of the SVM is calculated as $y = \sum_{i=1}^n \alpha_i y_i K(x, x_i) + b$, where $K(.)$ denotes a kernel function, b is offset value. (x_i, y_i) is the $i-th$ training sample and corresponding class label in n training inputs. If an unseen sample x is present, the output y of the SVM is computed accordingly. The model parameters $\alpha = \{\alpha_i\}_{i=1}^n$ are learned by solving the optimization problem raised in the dual form as shown in Equation 2.6.

$$\min_{\alpha_i}\Big(\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i y_i \alpha_j y_j K(x_i, x_j)\Big), \tag{2.6}$$

satisfying the constrains $\sum_{i=1}^n \alpha_i y_i = 0, 0 \leqslant \alpha_i \leqslant C, i = 1, ..., n$, where C denotes an upper bound for the soft margin of the optimal hyper-plane.

2.3.3 Multiple kernels support vector machine

A common approach in kernel machines is to use multikernel learning which is developed by combining several linear and non-linear kernel functions applied to the input space. This method has been proven to enhance the prediction accuracy [30, 31]. The approach can be adapted to solve large scale problems [32]. The multiple kernels can be solved by using the Sequential Minimal Optimization (SMO) approach. The multikernel formulation can be described as follows:

Given the training data $\{(x_i, y_i)\}$, ones can define a set of m base kernels $\{K_k\}_{k=1}^m$ which may include linear function, radial basis function, sigmoid and polynomial kernel functions. The corresponding feature map is denoted as $\{\Phi_k\}$ such that $K_k(x_i, x_j) = \Phi_k(x_i).\Phi_k(x_j)$. The multiple kernel SVM aims to learn a linear combination of kernels $K = \sum_k d_k K_k$, with

$d_k \geqslant 0$. The primal problem can be formulated as follow.

$$\max_{w,b,\xi\geqslant 0,d\geqslant 0} \frac{1}{2}\sum_k w_k^t w_k + C\sum_i \xi_i + \frac{\lambda}{2}(\sum_k d_k^p)^{\frac{2}{p}} \tag{2.7}$$

subject to $y_i(\sum_k \sqrt{d_k} w_k^t \Phi_k(x_i) + b) \geqslant 1 - \xi_i$, where the normal vector is denoted as w_k, slack variable ξ_i and a constant λ. The common use of p-norm here is $p = 2$.

2.4 Approaches to improve standard learning algorithms

The following presents some of the approaches that have been developed as part of an effort to improve the effectiveness or speed of standard learning algorithms. Gradient based learning algorithms in particular are known to have deficiencies such as a slow convergence rate (i.e. they are known as slow learners), and problems with training deep network architectures or learning long time sequences (known as the long term dependency problem). The following methods are among the more successful approaches to address such shortcomings.

2.4.1 Rprop algorithm

The term Rprop stands for the Resilient Propagation. The method is designed to remove the harmful effect of the size of the gradient when training MLP networks. Rprop changes weights based on the sign of the gradient and assigns a learning rate that allows acceleration through a momentum to each of the network weights [33]. The learning rule is as follows:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial E^{(t-1)}}{\partial w_{ij}} * \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial E^{(t-1)}}{\partial w_{ij}} * \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{else} \end{cases} \tag{2.8}$$

where $0 < \eta^- < 1 < \eta^+$. $\Delta_{ij}^{(t)}$ is the amount by which the weight connecting the j-th neuron with the i-th neuron is changed at time (or iteration) t. The amount of weight change becomes smaller if the sign of the gradient has changed since the previous iteration otherwise, if the sign of the gradient remained unchanged then the weight change value increases (i.e. it builds a momentum). The learning rule now becomes:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ \Delta_{ij}^{(t)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0, & \text{else} \end{cases} \tag{2.9}$$

There is one exception that if the derivative changes sign and the previous updating step was large, the target minimum was missed, then the updating rule is given as follows:

$$\Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t-1)}, \text{ if } \frac{\partial E^{(t-1)}}{\partial w_{ij}} * \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \tag{2.10}$$

2.4.2 Pseudo-Newton Optimization based methods

The Pseudo-Newton method is proposed in [34]. It takes advantage of the second order derivatives of the cost function (Hessian matrix) in MLPs. For an input x and the weight value w_i, the corresponding cost/error function is $E(x)$. The updating of the weight follows the online learning rule:

$$\Delta w_i(x) = -\frac{\lambda}{\| \frac{\partial^2 E(x)}{\partial^2 w_i} \| + \mu} * \frac{\partial E(x)}{\partial w_i} \tag{2.11}$$

where, λ and μ are small positive constants. $\frac{\partial^2 E(x)}{\partial^2 w_i}$ denotes the second order derivative of the error function. The first term on the right hand side of Equation 2.11 denotes the learning rate. In this case, the learning rate is locally modified for each weight value. When $\| \frac{\partial^2 E(x)}{\partial^2 w_i} \|$ is small, a larger learning rate would be obtained. The μ value is to prevent the $\Delta w_i(x)$ from

becoming very large.

Another method based on the pseudo-Newton optimization adapts to a longer span of input/output dependencies, and is called weighted time pseudo-Newton optimization [35]. The method updates the weight by the sum of all $\Delta w_{it}(x)$ at all time instances t. The μ value here is updated online to limit $\Delta w_i(x)$ to lower values than a predefined upper bound.

2.4.3 Genetic algorithm or Particle swarm optimization

Genetic algorithms (GA) have been introduced to avoid the need for gradient computation altogether [36]. Hence, a GA should not be affected by long term dependencies. The authors of [36] proposed the use of a cellular genetic algorithm and additionally, two learning techniques named Lamarckian and Baldwinian for training parametric systems. The Lamarckian method guarantees that offspring genotypes inherit good experiences from their parents. On the other hand Baldwinian learns better fitness values. A *chromosome* will survive in the next generation if its learned fitness is better. The base artificial neural network used is the recurrent neural network.

Related to GA is the Particle swarm optimization (PSO) algorithm [37]. The PSO is inspired by the collective behavior exhibited in swarms of social insects. Each particle represents a potential solution based on its own position and flight velocity, which is being adjusted during the optimization process. The PSO is better than the GA in terms of convergence speed and the ability to escape from a local optima.

2.4.4 Deep learning underlying concept

A large number of deep learning models have been proposed in recent years. The earliest exploration of deep leaning originated from a multilayer neural networks (MLP). Hinton and Bengio introduced Deep Belief Networks (DBN) by using the Restricted Boltzmann Machine (RBM) [22] as the unsupervised pre-training layer, followed by a fully connected

neural network for the supervised learning stage. General knowledge and analysis of deep learning architectures is provided in [21]. The underlying idea of deep learning could efficiently be applied in both neural processing and kernel machine cases, in which the unsupervised and supervised learning algorithms are properly integrated together. Subsequent chapters in this thesis provide a deeper exposure to methods for training deep network architectures.

2.5 Conclusion

This chapter presented a number of widely known machine learning algorithms. They have in common that they are vector based rather than graph based models. While the conventional artificial neural networks have a longer application history, the simpler kernel machine models have gained interest in the machine learning community in more recent years. This is largely due to the learning speed of these algorithms, efficiency in learning, and testing a real world large scale problem. However, more recent developments in deep, hierarchical neural networks has brought back the center of attention to the parametric type of algorithms. Nevertheless, as will be shown in the subsequent chapter several powerful models in kernel machines are being proposed. Some of these kernel methods are inspired by the deep learning concept in neural processing, for example the ones introduced in [38, 39]. In practice however their performance has not improved as dramatically as in the case of deep or hierarchical neural network learning.

Chapter 3

Machine Learning Methods for Structural Data

3.1 Introduction

This chapter provides an overview of graph based machine learning techniques. Both non-parametrized and parametrized models will be reviewed. Some explanation to structural data representations will be offered in Section 3.2. The formal definition of a graph based learning problem is given in Section 3.3. Section 3.4 provides an overview of graph based artificial neural network models. Unsupervised learning models are considered first, then supervised models are presented. Section 3.5 presents graph kernel machines. Special attention will be given to models which will be utilized later in this thesis. Existing methods addressing long term dependency (or the vanishing gradient problem) in the graph domain are presented in Section 3.6. Finally, Section 3.7 concludes this chapter.

3.2 Data structure representation

This thesis will refer to any data that is represented by scalars or fixed sized vectors as *unstructured data*, whereas sequences, trees, and graphs will be referred to as *structured data*. The following will discuss several typical types of structural data representation.

The simplest type of structured input representation is a temporal sequence. A sequence conveys a temporal order of its elements. It has one beginning and one ending point. This data representation technique differs from conventional fixed-sized vectorial data in that its elements can not be arranged randomly. The underlying temporal order can be exploited by appropriate methods such as the SOMSD, Elman network and LSTM learning systems. Temporal sequences are commonly used to represent time series problems. Some real-world examples of problems which can be represented by temporal sequences include physical activity classification, energy expenditure prediction, stock market prediction, handwritten letter/digit/word prediction, speech recognition, and forecasting in general. There are two main types of sequence learning problems, namely prediction (forecasting) and classification of sequences. The former is often seen as a time series problem where a learning model learns the time variation up to the current point, and then predicts what will happen next in the immediate future. The latter problem types usually appear when each sequence represents an object, e.g. the DNA sequence, and each sequence can be categorized into a specific class, like active or inactive class. All such data are modeled based on the time order of appearance elements.

A second type of structural data representation are spacial temporal or contextual temporal sequences. This type of data considers both the temporal order of sequence elements and the spacial/contextual relationship among features within a single time step. The two cases of this type of problems are illustrated in Figure 3.1. Case 1 of this figure shows that a physical activity can be modeled by a temporal sequence, and can be classified into a typical type of physical activity. A commonly used sensor for physical activity recognition learning

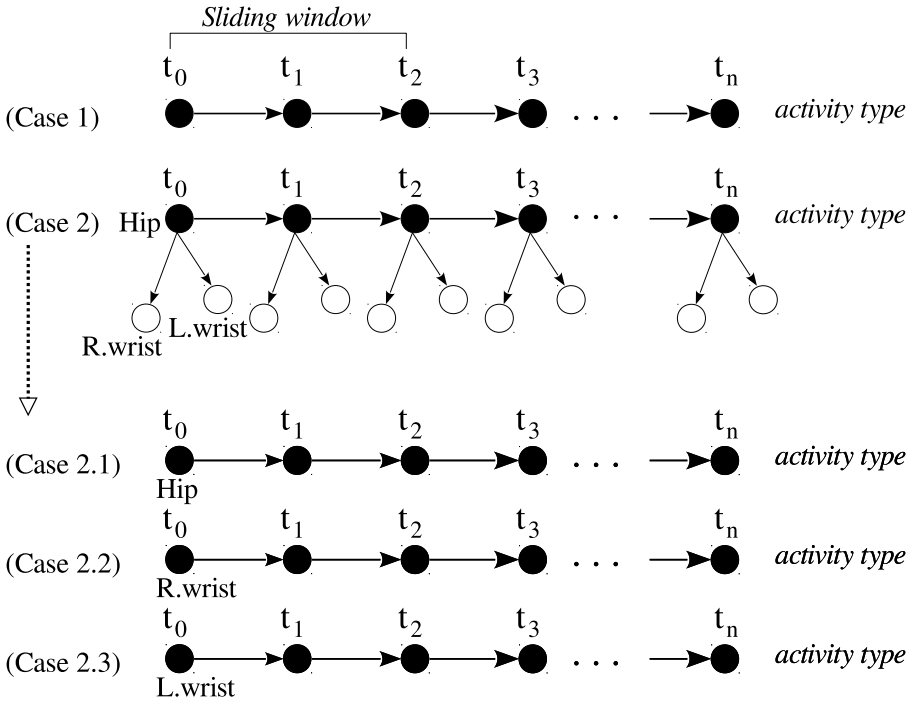

Figure 3.1: Contextual/temporal sequence type of problems

problems measures g-forces (using accelerometers) at regular time intervals (i.e. at 100Hz). The task is to model the patterns in the data sequence in order to predict activity type or energy expenditure. Such accelerometry is commonly taken from either hip, left or right wrist. When training a learning system, a sliding window and step size can be provided for every sequence resulting in a fixed dimensional input. Learning is accomplished by sliding the window to the end of the sequence where the target label is found. The input to the learning model at each iteration is the feature data within the window. This type of learning has been used extensively in the literature i.e. by recurrent neural networks [26, 29, 40]. In the second case, Figure 3.1 shows that the conceptual relation between movements of hip, left and right wrists can be modelled. In this case, at each time step, the information might include the spatial relation between different positions of a human's body. Thus, the three sequences are linked by the same time frame.

A third type of structural data are tree data structures. A tree is a simple graph in which

pairs of nodes are connected by a single path. Cycles are not permitted in a tree. A non-empty tree consists of a *root node* (or super-source) and potentially many levels of additional nodes. The root node is located at the top level and has no parent nodes, while the *leaf nodes* are those without children, and are located at the lowest level of the tree. The ordered links in a tree differ from unordered links in that those links are arranged in a predefined order. A value (could be a vector) attached to a node will define its characteristic, which is denoted as a *label* or *feature vector* of that node. Data trees are very common in computer science. A typical example of this type is obtained by parsing semi-structured documents such as source code or hypertext documents. Such trees are referred to as *parsing trees*. XML documents make a good example. In XML documents, contents are located in different blocks which are called tags. The entire document is represented by a complete XML tree. The tree consists of nodes which represent the XML tags (e.g $< p >$ and $< br >$), and links which represent the nesting of the tags. The process to construct a tree structure from an XML document is named a parsing stage. Since a tree requires an existing root node, the top level tag in the XML document is parsed to be the root node, and each tag belongs to one level of the tree. The label attached to the nodes may be the identifier of the unique tag. The drawback of this data structure representation is that the related content of the XML document is normally not involved in the XML tag tree. This may reduce the information richness of the XML tag tree.

A more generic type of structural data representation is graphs. A graph may contain any kinds of links (self links, undirected, directed, ordered, unordered links) and may contain cycles. Graphs are supposed to sufficiently represent various complicated real world problems. An example would be seen in the WWW situation. Webpages and internet documents are typical representatives of hyperlink based structural data. Those objects are connected via hyperlinks. A webpage can have an arbitrary number of hyperlinks to other documents on the WWW. Those connections have a characteristic that they may be multi-fold refer-

encing or self-referencing. The graph built from hyperlink structure of documents on the Web is referred to a directed cyclic graph. In this case, the content of documents/webpages will form the feature label vectors of the nodes. In a webpage classification problem, for instance, a single webpage could be designated normal if it is not known to be a spam one. In practice, a directed cyclic graph could be utilized for modeling the spam/non-spam classification problem. In particular, each node in the graph is present to either belong to the spam or normal class, while the graph connections are viewed as hyperlinks on the internet. The intentional removal of hyperlink information would lead to a loss of useful information regarding data representation.

One of the data structures that extends the representational power of graphs are called *hypergraphs*. Hypergraphs are a generalization of a graph in that the edges are no longer limited to link just pairs of nodes. The links in a hypergraph can connect any number of nodes. The connections in the hypergraph are called hyperedges. Applications for which hypergraph representations are useful can be found in telecommunication, parallel computing, and game theory.

Another type of data structure that extends the representational power of graphs are called graph-of-graphs (GoG) [16]. A GoG is a graph whose nodes are labeled by other graphs. This is useful for application domains which need to be represented by more than one type of graph. Figure 3.2 uses the World Wide Web as an example. The WWW consists of interlinked documents and hence this is suitably represented as a Web graph. The Web graph consists of nodes that represent web pages and (directed) links that represent the hyperlink structure of the Web. Since each node in the Web graph represents a web page and since most documents in the WWW are described by some markup language (XML and HTML are most common) which describe the structure of the document and hence, each document is most suitably represented by the corresponding XML (or HTML) tree. Thus, each node in the Web graph is suitably described (or labeled) by an XML tree. The

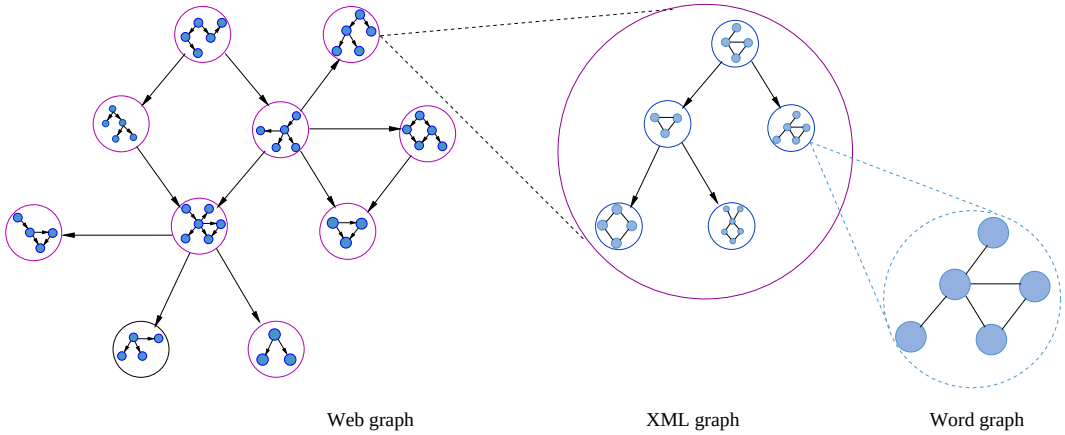

Figure 3.2: Web GoG illustration.

procedure of labeling nodes in one graph with other graphs can continue up to any suitable depth. For example, Figure 3.2 shows that the nodes of the XML tree can be suitably described by a concept graph or word graph. The Web graph in this example is referred to as the level 1 graph or root graph, the XML tree is referred to as level 2 graph, and consequently the word graph is the level 3 graph. The GoG depicted in Figure 3.2 is thus said to be a GoG of depth 3. Another example of a GoG is a time series of graphs. This refers to a set of graphs for which an ordering is defined. This, for example, is used to represent videos in video classification problems. A video consists of a temporal series of still images (called frames) where each frame is suitably represented by a region adjacency graph.

It is interesting to note that GoGs are the most general type of data structure presented in this thesis. A GoG contains hypergraphs, graphs, trees, sequences, vectors, and scalars as special cases. Hence, any vector, sequence, graph, etc. is a special case of a GoG. Generally, one is able to select a wide range of graph representation approaches, whichever is most suited to the problems at hand. This thesis focuses on graph learning problems, since the modelling of a graph is a universal method for most (if not all) real world learning problems.

3.3 Formal definition of graph learning problems

The problem domain involving graphs as studied in this thesis can be stated as follows: A classification problem is denoted to belong to the graph domain if it could physically be modelled as graphs. If the whole problem is represented as a single graph, then each node commonly will account for a training sample. In this case, the nodes become the subjects of interest. The sample's feature vector is practically the means to label the nodes. Each node can have a unique desired target/class label. On the other hand, if an input sample can be modeled as a graph containing a root node, there may be one supervised root node in the graph. In practice, if a root node is not clearly specified, the first node in the graph would be assigned to be the root node. In this case, the graphs become the subjects of interest. Generally, a graph learning model can create a mapping for each graph in a non-empty set of graphs that is said to be a *graph focused* (*multiple graphs* or *many graphs*) application. It can also map each node in a non-empty set of graphs that is called a *node focused* (*single graph* or *one graph*) application. Such multiple graphs technically can be seen as a single graph without connections between sub-graphs. The detailed description of these graph problems is explained in [4].

The term graph data structure (or simply graph) is a generic representation of a data structure that includes temporal sequences, trees, and directed/undirected cyclic/acyclic graphs. Formally, a graph is represented as follows, $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ where $\mathcal{N} = \{1, 2, \ldots, |\mathcal{N}|\}$ is the set of nodes, while $\mathcal{E}$ is the set of edges $\mathcal{E} \subseteq \{(u, v)|u, v \in \mathcal{N}\}$. The indegree of a node v is the number of incoming edges to v, whereas the outdegree of v is the number of outgoing edges from v. In the case of a tree structure, an edge is viewed to be directed if a tuple $(u, v) \in \mathcal{E}$ is an ordered pair where the natural direction is presented from the vertex u to vertex v of the edge. Every undirected edge can be represented as being directed by adding to every tuple (u, v) the reverse connection (v, u).

Each node or link may be labeled by a feature vector $\mathbf{x}_i \in \mathcal{R}^L$. We assume that these

features are numeric and can be of any fixed dimension (including zero-dimensional). Some of the nodes may have targets. Without loss of generality, we assume that the problem contains examples of two classes, $+1$ and -1. Let the set of nodes with $+1$ targets and -1 be denoted by $\mathcal{N}^+ \in \mathcal{N}$, with $|\mathcal{N}^+|$ nodes and $\mathcal{N}^- \in \mathcal{N}$ with $|\mathcal{N}^-|$ nodes respectively. The problem is that given the set of target nodes, $\mathcal{N}^+ \oplus \mathcal{N}^-$, together with the feature vectors in each node $n \in \mathcal{N}$, can one infer the targets of the nodes in the set $\mathcal{N} \cap \{\mathcal{N}^+ \oplus \mathcal{N}^-\}$?

In reality, a graph learning problem may encapsulate three difficulties:

1. Curse of dimensionality

2. Imbalanced class distribution

3. Remote path dependency or long term dependency

The curse of dimensionality refers to the particular problem that the dimension of input feature vectors is significantly large ($L \gg 0$). When the dimension of the input space increases, the available data would become sparse which can become problematic for some machine learning approaches. The curse of dimensionality also refers to a conjunction problem of both the input data space and the applied learning algorithm. This arises since the learning model does not scale well to high dimensional data, typically due to the high demand in computation time or memory storage that exponentially increases with the dimension of data.

The imbalanced class distribution is another common problem regardless of whether binary or multiple class cases are used. The problem occurs when the number of samples that belong to one class is significantly different to the number of samples in any of the other classes. For example, a binary classification problem in the domain of graphs in which nodes are labeled by target values, where a class imbalance problem can occur when there is an overwhelmingly large number of negative samples compared with the number of positive samples. Formally this can be shown as $|\mathcal{N}^+| \ll |\mathcal{N}^-|$. Similarly, the case

of $|\mathcal{N}^+| + |\mathcal{N}^-| \ll |\mathcal{N}|$ also falls into the category of imbalanced class distribution. The latter indicates that not all nodes are supervised and this is best understood by assuming that all unlabeled nodes belong to the imaginary class "unlabeled". The imbalance issue can hinder the standard prediction models to achieve high accuracy on the minority class, since the overwhelming major class examples can alter the learning model's objective. This is particularly the case for learning methods that are robust to noise. Such models often dismiss the minority class as noise.

The remote path dependency problem can occur in graphs in which the shortest path between a pair of nodes becomes very long. Two nodes n_{d1} and n_{d2} are said to be *remote path dependent* if there exists a physical path $\mathcal{P}$ starting from node n_{d1}, going through edge e_i and different nodes n_i, then ending at node n_{d2} or vice versa. One can define $\mathcal{P} = (n_{d1}, e_1, n_1, ..., n_\tau, e_\tau, n_{d2})$, where τ is a positive number and $n_{d1}, n_{d2} \neq n_i, 1 \leqslant i \leqslant \tau$. Nodes on the path $\mathcal{P}$ do not necessarily belong to a single class. A robust classifier would be able to categorize correctly those two far apart nodes n_{d1} and n_{d2}. A related problem is the more traditional problem of the so-called vanishing gradient or *long term dependency* problem which originated in learning a recurrent/recursive neural network. The learning algorithm used for that network is usually based on computing the gradient of an objective function with respect to the network weights. In the backward phase of the back-propagation mechanism, the error gradient is backward in time (i.e along the temporal sequence backward to the beginning point/time) and recursively re-computes the required gradients. While the desired output at time t_c depends very much on the input information presented at a very earlier time $t_e \ll t_c$ in the sequence, this results in the gradient gradually vanishing during the back-propagation pass. This makes learning with long term dependency very difficult [35]. In the graph learning domain, it turns out that the remote path dependency is very closely related to the long temporal sequence in time series problems, though the former seems a little more complicated. Hence, one could theoretically define a long term depen-

dency problem in graph based neural network models, similar to which occurs in recurrent neural networks. In fact, the learning mechanisms of recurrent NNs and graph based NNs are both in recursive forward and backward intervals. Later this thesis will interchangeably use the terms remote path dependency, long temporal sequence, vanishing gradient and long term dependency to refer to a similar learning problem in recurrent or recursive NNs.

3.4 Artificial Neural Networks for structural data

Table 3.1 lists some of the milestones in the development of ANNs in chronological order, considering both unsupervised learning and the supervised learning paradigms. The evolution of neural networks shown in Table 3.1 with respect to data types than can be processed by these methods. It is observed that the development of neural networks for modelling increasingly complex data types has accelerated in recent years. Neural networks were limited to processing fixed sized vectors for nearly 30 years. Then new methods were introduced which were able to process sequences, then data trees, then general graphs, and most recently, graph-of-graph data structures. The table also shows that supervised learning methods were generally leading the way, with unsupervised methods trailing by several years in their ability to process complex data structures.

3.4.1 Unsupervised neural networks

The SOM architecture and training algorithm has been generalized in recent years to enable the processing of structural data. Approaches that have been introduced include the Self-Organizing Map for Structured Domains (SOMSD) [2, 49, 50], supervised training algorithm for SOM [51], GraphSOM [52], Probability measure graph self organizing map (PMGraphSOM) [3] and most recently the CompactSOM [41]. In addition, non-sparse kernels based on SOMSD activation maps have been proposed very recently [53]. The im-

Table 3.1: The evolution of NNs with respect to the capability of processing different types of input data.

Year	Supervised learning	Unsupervised learning
2011		Compact SOM [41]
2010	GoG neural network [16]	
2009		Probability mapping SOM [42]
2008		Graph Self organizing Map [10]
2004	Graph neural network [43]	
2002		EditSOM [44]
2001		SOM for structure data [2, 45]
1997	Recursive neural network [5]	
1996	Backpropagation throught structure [46]	
1995		Self organizing Map [7]
1993	Labeling RAAM [47]	
1990	Elman recurrent neural network [29]	
.		
.		
1961	Multilayer Perceptron [48]	

proved SOMs present robust clustering properties and are capable of processing various types of input graph data, such as labeled, undirected, sparse, and cyclic graphs.

3.4.1.1 The SOM for structured data (SOMSD)

The SOMSD is an extension of the traditional SOM with the capability of processing labeled directed acyclic graphs (DAGs) [2]. A constraint of DAGs is that there must exist at least one super-source node. In a DAG D, nodes with no outgoing edges are called $leaf$ nodes, nodes without incoming edges are *root* nodes. The remaining nodes are defined to be $intermediate$ nodes. Each node j in a graph is represented by a hybrid vector x_j which contains any data label that may be attached to the node as well as information about the mappings (coordinates) of the nodes' offsprings. The vector is defined as $x_j = (l_j, v)^T$, where l_j denotes a p-dimensional data label, and v is an o-dimensional coordinate vector, with o being the maximum outdegree of any node in the training set of graphs. Vector v is padded if the outdegree of a node is smaller than the maximum outdegree of the graph. T

denotes the transpose of the vector.

Unlike the traditional SOM that merely considers data labels in vector form as its input samples, the SOMSD utilizes both the information of labels [1] as well as the relationship among the data input. The relationship of the data is presented through vector v, which stores the information about the node's offspring. The mechanism is to maintain the process of passing the information about children nodes to the parent node. Because the SOMSD processes nodes in a recursive manner, information about any (connected) node is eventually passed to the root node of the graph. Information about a graphs' topology is finally "condensed" at the root node. The nature of the SOMSD input requires that the nodes in a graph are processed in inverse topological order (bottom-up, or from the leaf nodes to the root node). The training algorithm is an extension of the training algorithm of the standard SOM.

Training algorithm: Vector v contains the coordinates of the winning neurons of a nodes' children. Because a node can have several parents, the coordinates of the winning neuron are stored so as to be available when processing the parent nodes. This avoids computational expense in that the winning neuron needs to be computed only once for each node and within a training iteration. The training algorithm can be given as follows:

Step 1: Select a graph in the input space, then store all vertices of the graph in an inverted order (leaves first, then intermediates and finally the root node).

Step 2: For each vertex j in the list, form an input vector x_j; the winning neuron r is calculated as in Equation 3.1.

$$r = \arg\min_i \|\Lambda(x_j, c_i)\| \tag{3.1}$$

The Euclidean distance between input vector x_j and codebook vector c_i of the i-th

[1] Note that this thesis will use the "label" term in order to indicate the feature vector attached on the nodes of the graph. We will use the term "target" to denote the class label.

neuron on the map is weighted by the new matrix Λ of $(p+2o) \times (p+2o)$-dimension. The weight matrix Λ is a diagonal matrix, consisting of p diagonal elements which have the value μ. All other diagonal elements are $1 - \mu$. This parameter is used to control the contribution of the label data or the spatial location information with respect to a graph node in the training algorithm.

Step 3: After the BMU r is found, the winning codebook vector and its neighboring ones are updated by the following amount.

$$\Delta c_i = \alpha(t)h(\Delta_{ri})(x_j - c_i) \tag{3.2}$$

where $\alpha(t)$ is a learning rate. $h(.)$ denotes the neighbourhood function that is dependent on Δ_{ri} defining the distance between neuron r and neuron i.

Step 4: The coordinates of the current winning nodes are passed to their parent nodes so that their coordinate vector v is defined.

These steps are repeated until all nodes in a training set have been processed and for a number of training iterations until a stopping criterion is met. Common stopping criteria are: a maximum number of training iterations, or a threshold in the mapping performance is reached.

The SOMSD training algorithm works in an unsupervised manner. However, a supervised SOMSD learning framework does exist [51]. In fact, Kohonen introduced an approach to training SOMs in a supervised fashion [7]. However, Kohonens' approach has been found to have some drawbacks, namely, (1) the input vectors attached with class information lead to an imbalance in error measurement, especially in the case when the SOM is trained with very large dimensional target labels; (2) the computational complexity of the learning algorithm increases as a class label is attached to the input vector; (3) the algorithm is limited to a vectorial input situation. These problems have been overcome by the supervised SOMSD

algorithm introduced in [51]. In practice, the supervised SOMSD is useful for partially supervised learning problems, and supervised learning problems requiring visualizations. The supervised SOMSD will not be considered in this thesis since it is not suited to incorporation in the integrated model.

3.4.1.2 Contextual self organizing map for structured data (CSOMSD)

The CSOMSD was proposed in [49]. The purpose of CSOMSD is to overcome a limitation of the SOMSD when encoding contextual differences between identical subgraphs. A SOMSD cannot differentiate between identical subgraphs that appear in a different context within a graph. Furthermore, CSOMSD is able to address cyclic graphs [50] by using a pre-processor to replace cycles by special nodes. The major property of CSOMSD is that it considers the mappings of not only offspring nodes but also parent nodes when forming the input vectors. In particular, an input vector of the SOMSD regarding a node j is defined as $x_j = (l_j, y_{ch[j]})$, where l_j is the node label, $y_{ch[j]}$ denotes the concatenated list of coordinates of the winning neurons with respect to each node's offspring, which is also called a *state*. The input vectors for the CSOMSD are extended through concatenation of the mappings of the parent vectors as in $x_j = (l_j, y_{pa[j]}, y_{ch[j]})$, where $y_{pa[j]}$ and $y_{ch[j]}$ are states of parent and offspring nodes, respectively. This input representation has been reported to improve the mapping quality [50] but at the cost of requiring larger mapping space and consequently and increase in computational time requirements.

The CSOMSD is limited to processing acyclic and bounded graphs. When handling unbounded graphs, the model experiences decreased mapping quality and significantly increased computational time. The SOMSD and CSOMSD cannot process graphs whose maximum outdegree is not known. For example, when modelling the Web graph then the maximum number of hyperlinks that a web page can have is not known. The problem can be overcome by assuming a maximum outdegree and then pruning the state vector of any nodes for which the outdegree exceeds the maximum assumed value. This, however, can

result in decreased mapping quality and a significantly increased computational time [2]. The introduction of Graph self organizing map is a effective solution to unbounded graph learning.

3.4.1.3 Graph self organizing map (GraphSOM)

The GraphSOM as proposed in [52] is designed to overcome the problem with unbounded and cyclic graphs. A key difference between the (C)SOMSD and the GraphSOM is the way by which the state vector is represented. Instead of listing the mappings of each offspring (and parent), the GraphSOM uses consolidated activation of the mapping space. Given an activation map M and $k = |M|$ the number of neurons on the map M, the GraphSOM defines the input vector as $x_j = (l_j, M_{ne[j]})$, where l_j of dimension p is a data (feature) label vector, and $M_{ne[j]}$ is a k-dimensional vector containing the activation of each neuron on the map M with respect to the neighboring nodes of node j. One may define M_{ab} as the state of the neuron at the coordinate $[a, b]$. The state of a neuron is assigned to the number of neighbors that were mapped at that neuron's location. For example, if one wishes to train a simple undirected graph with 5 nodes as shown in Figure 3.3, one could use an activation map of size $2 \times 4 = 8$ neurons. For simplicity, it is assumed that no data labels are associated with each node. The figure shows that node $id = 0$ has three direct neighbors which are identified as 1,2, and 3. The example assumes that the mapping of these nodes (as obtained by a pervious training iteration) occurred at locations (1,1), (1,1), and (2,3) respectively. Thus, the state vector of node 0 would be $M_{ne[0]} = (2, 0, 0, 0, 0, 0, 1, 0)$. Since the example assumed that there is no feature vector associated with it, hence the corresponding input vector would be $x_0 = M_{ne[0]}$. Note that the elements in the state vector list the consolidation information about the mappings of node's neighbors. Since two neighbors were mapped to the same location, hence the corresponding element on the activation map is activated twice. This is indicated by the value 2 in the corresponding position of the state vector. All

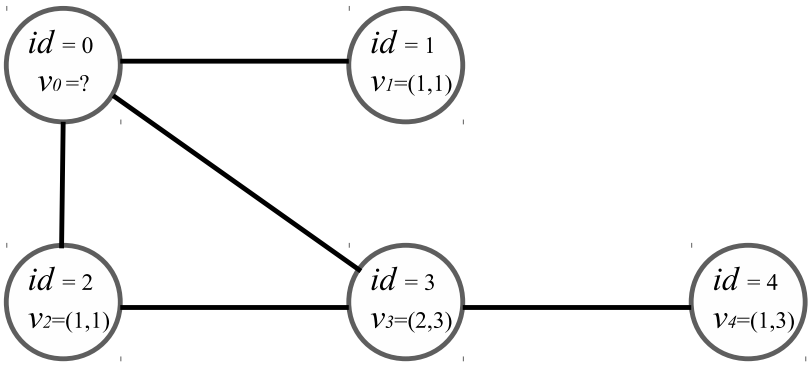

Figure 3.3: Illustration undirected graph with 5 nodes, the node id, and corresponding coordinate v of the best matching codebook vector are shown.

remaining ones are zero since no mappings occured at the corresponding locations. In other words, the node $id = 0$ has two neighbor nodes $id = 1$ and node $id = 2$, both being mapped to the same coordinate $[1, 1]$, and one neighbor node $id = 3$ mapped to coordinate $[2, 3]$. Given that the dimension of M is fixed, hence the GraphSOM is effective in dealing with learning problems for which the maximum degree of a node is unknown or unbounded.

The training algorithm of GraphSOM is similar to the one used for SOMSD. Since both methods form a hybrid input vector consisting of data label and state information, hence the diagonal matrix Λ is again used to control the influence of the data label, and the state component to the Euclidean distance. In GraphSOM the matrix is of dimension $(p + k) \times (p + k)$. The first p diagonal elements are set to μ, and all remaining diagonal elements are set to $1 - \mu$ as before. Note that if $1 - \mu = 0$ then the algorithm reduces to the basic SOM training algorithm. The GraphSOM training algorithm also consists of two main stages, the competitive phase that is present in Equation 3.1, and the cooperative phase shown in Equation 3.2.

The GraphSOM improves the abilities of the SOMSD and CSOMSD in that (1) construction of states is independent to the order of the neighbors. Hence, the approach can process unordered or non-positional graphs. However, the algorithm can also consider the order of the neighbors by concatenating the states to the input vector in node order. This leads to the fact that GraphSOM contains CSOMSD and SOMSD as special cases. (2) The

dimensions of input and codebook vector are independent of the maximum outdegree of all nodes. Thus, GraphSOM is suitable for graphs with a large outdegree. (3) Redundancies in the mappings of nodes are reduced.

However, the GraphSOM exhibits a problem in that the magnitude of changes in the mapping of nodes is not captured appropriatly by the state vector. For instance, assuming that the new mapping coordinate of node $id = 2$ is shifted to $[1, 2]$ and all other nodes are mapped to their previous locations, then the new input vector of node $id = 0$ would become $x_0 = (1, 1, 0, 0, 0, 0, 1, 0)$. If, however the mapping of node 2 had shifted more substantially to say $[1, 3]$, then the input vector would become $x_0 = (1, 0, 1, 0, 0, 0, 1, 0)$. Note that the Euclidan distance between $(2, 0, 0, 0, 0, 0, 1, 0)$ and $(1, 1, 0, 0, 0, 0, 1, 0)$ is the same as the Euclidan distance between $(2, 0, 0, 0, 0, 0, 1, 0)$ and $(1, 0, 1, 0, 0, 0, 1, 0)$. Hence, this way of representing new states does not provide a good indication of topographic similarities of the mapping of nodes, and the GraphSOM has consequently limited abilities to serve learning problems that require the topology preserving clustering of data. The problem is overcome by the Probability Mapping GraphSOM.

3.4.1.4 Probability mapping GraphSOM

The PMGraphSOM was designed to overcome the aforementioned weakness of the GraphSOM [3, 42]. The problem with GraphSOM is that it *hard codes* mappings in which the mappings of nodes would take either 1 if there exists a mapping at a given location, or 0 if there is not. The PMGraphSOM uses a *soft code* representation of the mappings by using a probability of a mapping at any location on the map. This allows the Euclidean to capture the relative distance of a neighbor's mapped location toward the location of the winning neuron. The probability mapping of an element in M is computed as follows

$$M_i = \frac{exp(-\frac{\|r_c - r_i\|^2}{2\sigma(t)^2})}{\sqrt{2\pi}\sigma(t)}, \tag{3.3}$$

where, r_c and r_i are the coordinates of the winning neuron and the i-th neuron in the lattice respectively. $\frac{1}{\sqrt{2\pi}\sigma(t)}$ is for normalization purposes, so that $\sum M_i \approx 1$. The cumulation is calculated for all the i-th nodes neighbors. $\sigma(t)$ decreases towards zero with time t. This leads to the early learning process creating a significant change in mapping, whereas in the late stage, as $\sigma(t) \to 0$, the probability mapping will become close to the hard code method with states.

3.4.2 Other unsupervised approaches for graph data structures

Kernel method aims to project the data space into higher dimensional space so that the input space can be separated better by a hyperplane in the kernel space. The new kernel function proposed in [53] and [54] (called KernelSOMSD) is based on the SOMSD in order to deal with graph data structures.

An extension of recursive neural networks to the unsupervised learning approach for structural data is proposed [55]. The authors present a fixed-length vector representation for DAGs, and define the method of Euclidean measuring the similarity of two graphs using a maximum entropy approach.

Besides that, there exists an approach that uses a graph similarity meassure called graph edit distance in order to map graphs onto a display space [56]. The authors defined a series of graph edit operations which represent their frequency of occurrence. Thus, proper edit operations are required to address various graphs which may feature errors and distortions. The task is to calculate the edit distance between two DAGs and find the minimum cost so that the best matching neuron is able to be found. However, the disadvantage of this method is that for a given dataset, there is no automatic mechanism available to derive edit cost operations. Instead, this is determined experimentally. Moreover, the computation of the graph edit distance does not scale with the size of graphs, hence the method is only useful for small learning problems.

Another approach to graph domain is the EditSOM [44]. The author applied the edit cost operations in order to measure the similarity of two graphs, then used the same set of operations for the cooperative stage in which the distance between input tree and the winning neuron is shortened. This requires the number of clusters to be known a priori as it is used as a parameter in the algorithm. The edit cost distance method does not need numeric presentation of a graph, only the graph relations is required. But as before, the EditSOM is limited to small scale learning problems due to the reason that is associated with the computation of the graph edit distance.

Another approach to the development of a SOM for learning problems involving tree structured data is the Multilayer Self-Organizing Map (MLSOM) as introduced in [57]. Here, the nodes at each level of a tree are processed by a different layer of the MLSOM. This approach enables MLSOM to be comparable with SOMSD performance in clustering, utilization and classification for tree structured data.

In summary: A wide range of unsupervised models for structured data have been introduced in the literature in recent years. Each model is designed to address a particular graphical data type. However, the SOMSD and its later generations such as CSOMSD, GraphSOM and PMGraphSOM appear to be the most complete series of learning models that one can choose to deal with the greatest variety of types of graph data structures. This is the main reason why this set of models is selected for applications in this thesis.

3.4.3 Supervised neural network models

One representative model for time series prediction problems is known as the Long Short-Term Memory (LSTM) [26]. More recently, several novel approaches dealing with structured data have been proposed, such as the back-propagation through structure [46], and recursive cascade-correlation [58]. A more generic model for data structures is proposed in [5]. However, those models are restricted in processing acyclic and directed graphs. Some

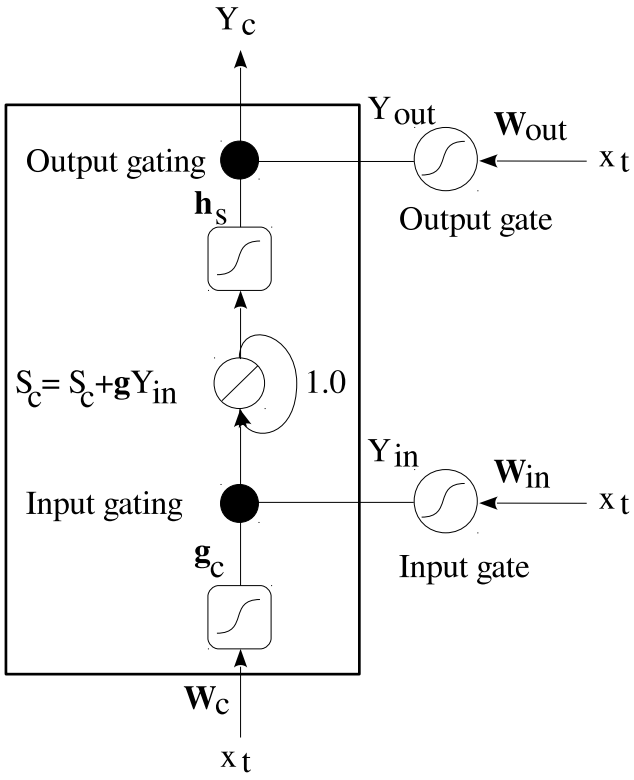

Figure 3.4: LSTM cell structure of the LSTM neural network model

other extensions in addressing the cyclic and labeled-link graphs were introduced in [59]. The graph neural network, a recent generation of recursive neural network can handle more general types of graphs such as cyclic, directed and undirected graphs [4, 43, 60]. One recent approach tackling graph-of-graphs problems has been examined [16, 61]. The following will explain some of the more well-known supervised neural processing prediction models in more detail.

3.4.3.1 LSTM recurrent neural network

This recurrent NN model has proved its effectiveness in solving long term dependency problems [26]. The LSTM architecture contains special memory blocks located at the hidden layer. Each memory block may include one memory cell or more. The memory is built with a fixed self-connection. The model is learned by seeking an appropriate way to open and

shut the input and output gates. For instance, the gate remains closed if the model accesses the input information as not useful and vice versa.

Figure 3.4 illustrates the single block memory with a single cell. The input x_t at time step t is given to each input, output gates and the memory cell. The corresponding weights are W_{in}, W_{out}, W_c. The squashing function used in the input gate and output gate are sigmoidal $f(x) = \frac{1}{1+e^{-x}}$. The squashing function at input of memory cell is the logistic sigmoidal function $g(x) = \frac{4}{1+e^{-x}-2}$, and at the output of memory cell is centered sigmoid $g(x) = \frac{2}{1+e^{-x}-1}$. Hence, we denote Y_{in}, Y_{out}, Y_c to be respectively the outcome of input, output gates and the memory cell. One has:

$$
\begin{aligned}
Y_{in} &= f(\sum W_{in} \times x_t) \\
Y_{out} &= f(\sum W_{out} \times x_t) \\
S_c &= S_c + Y_{in} \times g(\sum W_c x_t) \\
Y_c &= Y_{out} \times h(S_c)
\end{aligned}
$$

A major problem with gradient descent for standard recurrent NNs is that error gradients vanish exponentially quickly with the size of the time lag between important events. With the LSTM memory blocks, however, when error values are back-propagated from the output, the error becomes trapped in the memory portion of the block. This is referred to as an “error carousel”, which continuously feeds errors back to each of the gates until they become trained to cut off the value. Thus, regular back-propagation become more effective by training LSTM blocks to remember values for very long durations.

3.4.3.2 Recursive neural network (RMLP)

The RMLP utilizes the back-propagation through structure (BPTS) algorithm introduced in [62]. The model sometimes is called BPTS for simplicity. An application based on this neural network is suggested in [63]. A more generalization representation of the RMLP is

named as the recursive neural network [5, 64]. The main feature of RMLP is the recursive mechanism in learning a cyclic type of graph. Generally, RMLP can be referred to as an extended recurrent cascade correlation network, extended real time recurrent network and neural tree [5]. The RMLP will be described in detail here.

For a graph G, considering the vertex x with label l, the vertex $out_X(x, j)$ is the vertex pointed by the j-th pointer leaving from x. Now, considering the current node c and its children nodes $ch[c]$, then x_c denotes the state of the current node in the given graph, $x_{ch[c]}$ is the states of children of x_c. Let l_c be the label of c, and N be the dimension of the input node label corresponding to N label input neurons (see Figure 3.5). H is the number of hidden neurons, also the number of recurrent state neurons and moreover is the dimension of the state vectors. The output o corresponding to each node at time step t (or the state of a node at time t) is calculated as follows:

$$o_c(t) = f(l_c, x_{ch[c]}) \tag{3.4}$$

Or, to be more specific, one can include the weight values as follows:

$$\begin{aligned} o_c(t) &= f\Big(W^I l_c + \hat{W}^I x_{ch[c]}(t)\Big) \text{ or} \\ o(t+1) &= F\Big(W^I l_c + \hat{W}^I o(t)\Big) \end{aligned} \tag{3.5}$$

It is worth noting that the RMLP utilizes the same weight value for the same order of each child's node to connect to the hidden layer. Specifically, any node c has an i-th child which corresponds to the weight $\hat{W_i^I}$. That is the reason why RMLP is limited to deal with ordered or positional graphs. In the output layer, y is computed as follows:

$$y(t) = g\Big(W^O x_s(t)\Big) \tag{3.6}$$

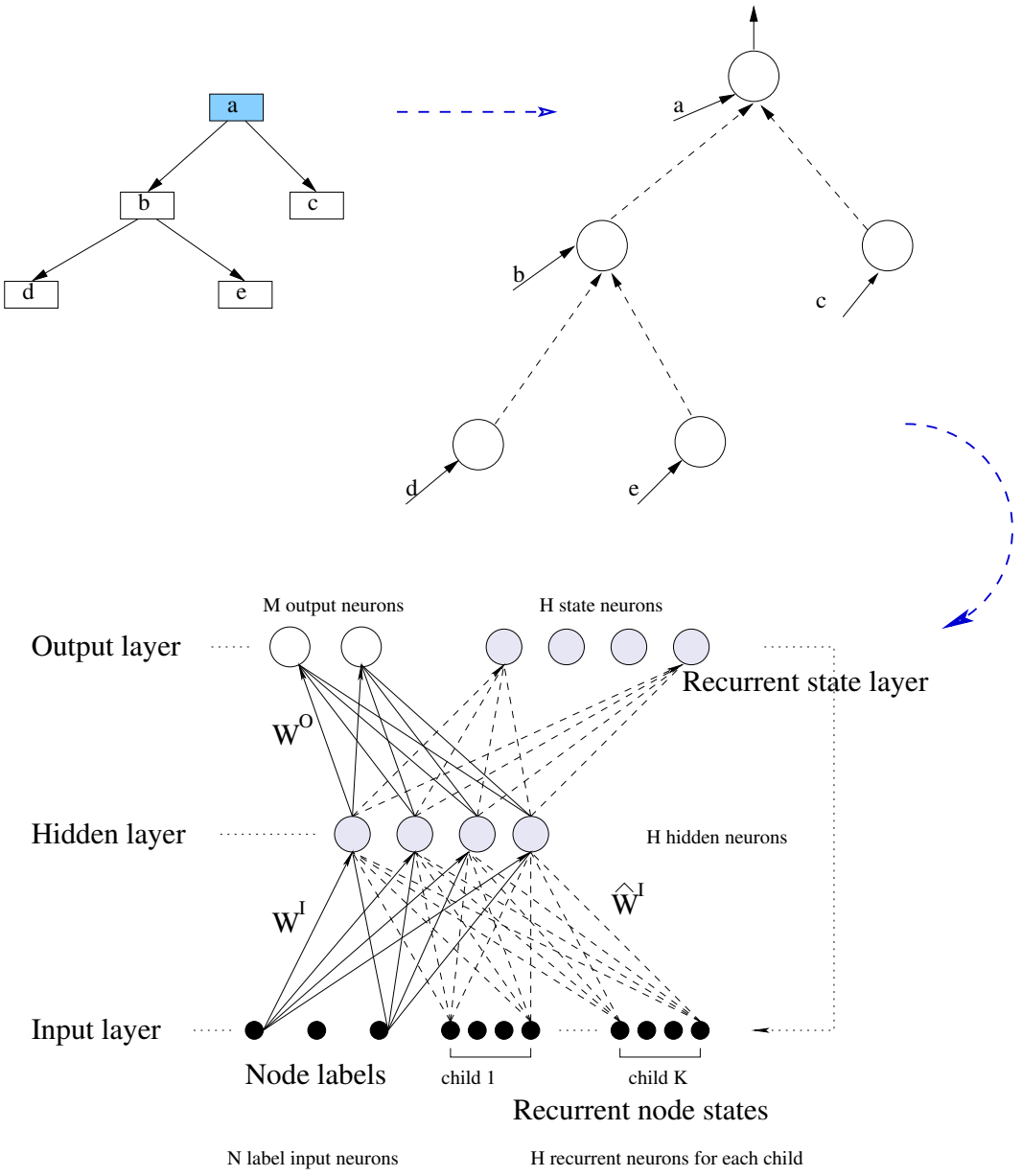

Figure 3.5: RNN model: The graph on the top left, the unfolding network on the top right and the network architecture shown on the bottom.

Here, f and g are transition and output functions, respectively.

3.4.3.3 Graph neural network (GNN)

The Graph neural network was first introduced in [43]. It has been applied to a number of practical applications, namely, for XML document and sentence classification [11, 15], for web page ranking and processing [14, 65], and for an image recognition application [13]. A comprehensive explanation of the GNN learning model and computational complexity is presented in [4, 60]. The GNN is considered a generic model which can accept various types of graph input, such as directed/undirected, ordered/unordered, edge labeled and cyclic graphs.

In the encoding network, consider the current node c and its neighboring nodes ne. Then x_c denotes the state of current node in the given graph, and x_{ne} is the state of neighbors of x_c. Let l_c be the label of c, and l_{ne} be the labels of ne. Linked-edge labels between c and a node u of ne is $l_{(c,u)}$. s is the dimension of the nodes' state. For non-positional GNN, the current node's state and the output o corresponding to each node at time step t are calculated as follows:

$$\begin{aligned} x_c(t) &= \sum_{u \in ne} h_w(x_u(t), l_u, l_c, l_{(c,u)}) \\ o_c(t) &= g_w(x_c(t), l_c) \end{aligned} \tag{3.7}$$

where h_w and g_w are local transition and output functions, respectively. Function h_w is introduced in order to make the GNN to be applicable to un-ordered graphs. For simplicity sake, one reduces the representation of Equation 3.7 as follows:

$$\begin{aligned} x &= F_w(x, l) \\ o &= G_w(x, l_c) = G_w(F_w(x, l), l_c) \end{aligned} \tag{3.8}$$

Here F_w and G_w are global transition and output functions, respectively. l is stacked by all labels or the current node, edge and neighbor node labels. However, note that x in

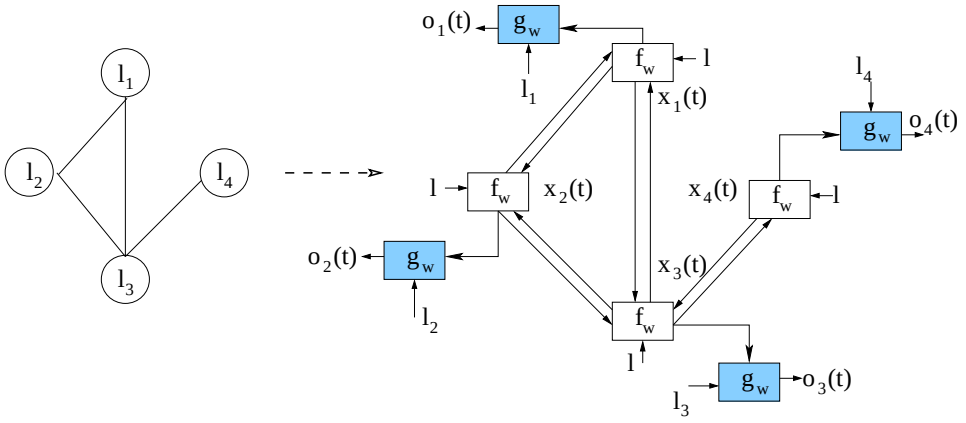

Figure 3.6: The graph neural network is present where a given graph on the left, unfolding network on the right.

Equation 3.8 on the left hand side is not the same as x in the right hand side. At time step t, the current state x_c of a node on the left of Equation 3.7 is computed, then in the next time step $t+1$, that value of x_c would become x_{ne} on the right hand side, if at this time step we consider the activation of the c neighboring node. Because of the cyclic nature of the graph being processed, the state value x_c can be iteratively calculated in order to achieve a stable solution at which the state of each individual node is almost unchanged. Figure 3.6 gives an example of a graph with 4 nodes (on the left), and demonstrates the corresponding unfolding network of the GNN model (on the right). It shows how the transition and output functions at each particular node receives the input and provides the output.

The GNN^2 (GNN squared): The GNN^2 has been designed for modelling GoGs. The curious nomenclature arises out of the fact that the GNN^2 consists of a number of GNNs that have been stacked on top of each other. Thus, this approach to address GoGs is an extension to the GNN algorithm [63]. The network architecture for each level in the GoGs has inherited some features from the GNN model [43]. The outcome of the states of each deeper-level graph will be used to re-label the corresponding node of its parent graph. This process takes place from the innermost graph out to the outermost one. The backpropagation stage is then applied for the purpose of weight updating using gradient descent. Formally, the state of a

node at level 0 is computed as follows:

$$x_i^0 = F_i^0(l_i^0, o_{out[i]}^1, x_{ne[i]}^0) \tag{3.9}$$

$$o_i^0 = G_i^0(l_i^0, x_i^0) \tag{3.10}$$

where, i is a given node ID. $o_{out[i]}^1$ and o_i^0 denote the resulting state of a child graph as a whole, featuring node i and the resulting state of node i, respectively. $ne[i]$ are the neighbors of node i, and l represents the label of node i. Finally, F_i^0 and G_i^0 are transition and output functions that can be realized as linear, non-linear or hyperbolic functions. In an iterative manner, the resulting state of a child graph at level 1 is given by:

$$x_i^1 = F_i^1(l_i^1, o_{out[i]}^2, x_{ne[i]}^1) \tag{3.11}$$

$$o_i^1 = G_i^1(l_i^1, x_i^1) \tag{3.12}$$

and so on. thus, the general form of the GNN can be stated as follows

$$x_i^k = F_i^k(l_i^k, o_{out[i]}^{k+1}, x_{ne[i]}^k) \tag{3.13}$$

$$o_i^k = G_i^k(l_i^k, x_i^k) \tag{3.14}$$

At the top-most level K of a GoG, the vector o becomes a null vector such that the output is given by:

$$x_i^K = F_i^K(l_i^K, x_{ne[i]}^K) \tag{3.15}$$

$$o_i^K = G_i^K(l_i^K, x_i^K) \tag{3.16}$$

The parameters in a GNN are updated by a gradient descent method similar to the one used for MLPs and RNNs.

3.5 Kernel machines for structural data

Kernel methods have recently emerged as an interesting development of machine learning algorithms. Various graph kernels have been proposed, such as the diffusion kernel based on graph spectral analysis of the input graph [66], graph regularization [67], or graph Laplacian support vector machine [68, 69, 70]. Regardless of supervised or unsupervised algorithms, the kernel matrix is computed according to a graph matrix (such as graph Laplacian) that is derived from the adjacency matrix expressing the neighborhood relations between samples. The following sections will provide some details on two well-known kernel methods, namely Spectral kernel clustering and Graph Laplacian support vector machine.

3.5.0.4 Spectral kernel clustering

The underlying concept of SKC follows the spectral graph theory [8]. The idea is to utilize eigenvectors of the similarity graph to perform input dimensionality reduction/ transformation, then using the KKM algorithm for the rest. The learning algorithm is presented briefly as follows. Given the input data in the form of a similarity graph, the task is to seek the most efficient way to partition the graph into a specific number of node groups with the constrain that the links between nodes of different groups are most weakly connected. The nodes of a group would be most similar to one other, or the connections within a group's nodes are strong.

Given that the number of clusters is user pre-defined (say k clusters), the SKC algorithm consists of three main phases. In the first phase, the graph Laplacian matrix L is constructed via the adjacency matrix. By applying eigen decomposition in the second phase, the first k eigenvalues and the corresponding k eigenvectors $(u_1, ..., u_k)$ of L are taken into account. Let U be the matrix whose columns are composed of $(u_1, ..., u_k)$. Rows of U would be associated with n original input samples. Finally, the KKM is applied on the row-based samples of U, to cluster them into k clusters $C_1, ..., C_k$.

3.5.0.5 Graph Laplacian support vector machine

While the SVM and MKSVM are vectorial based learning models, the GLSVM is able to incorporate data structures in its learning process. The GLSVM is a manifold learning mechanism which has been extensively researched [1, 68, 69]. The model encapsulates several concepts, namely spectral graph theory [8], manifold assumption and regularization Reproducing Kernel Hilbert Spaces [1]. The introduction of GLSVM was inspired by the classical SVM dual formulation. In practice, it has achieved promising performance in semi-supervised learning [1]. The model can be solved via the primal form together with the usage of a conjugate gradient regime [71].

The learning algorithm can be presented as follows. In the input space, given the dataset $S = \mathcal{D} \cup \mathcal{U}$, with labeled data $\mathcal{D} = \{(x_i, y_i), i = 1, ..., l\}$ and unlabeled data $\mathcal{U} = \{(x_i, i = l+1, ..., n\}$. If the kernel matrix K and the graph Laplacian matrix L are available, the dual form of GLSVM is as follows:

$$\min_{\alpha \in \mathcal{R}^n, \xi \in \mathcal{R}^l} \sum_{i=1}^{l} \xi_i + \gamma_A \alpha^T K \alpha + \gamma_I \alpha^T K L K \alpha \tag{3.17}$$

subject to $y_i(\sum_{j=1}^{n} \alpha_i k(x_i, x_j) + b) \geqslant 1 - \xi_i, \xi_i \geqslant 0, i = 1, ..., l$. Also, the primal form of GLSVM is given here.

$$\min_{\alpha \in \mathcal{R}^n, b \in \mathcal{R}} \sum_{i=1}^{l} V(x_i, y_i, k^T \alpha + b) + \gamma_A \alpha^T K \alpha + \gamma_I (\alpha^T K + 1^T b) L (K\alpha + 1b). \tag{3.18}$$

where, $V(.)$ is a squared hinge loss function or L_2 loss. 1 indicates the vector of n elements equal to 1. γ_A is a co-efficient controlling the complexity of ambient space, which is another name for the parameter C in the cases of SVM and MKSVM. One may use γ_A instead of C for every kernel machine model. Finally, parameter γ_I controls the complexity of the intrinsic structural penalty function.

3.6 Learning long term dependency problem

This section will review recent approaches to addressing the long term dependency problem, with special focus on problems in learning graph data structures.

3.6.1 Optimizing the free parameters of BPTS algorithm:

Recently, an improved algorithm was proposed to solve the long term dependency problem in RNNs [72]. The mechanism is created to optimize the free weight parameters $\theta = (A, B, C, D)$ that exist in the BPTS model.

$$\begin{aligned} x &= F(Aq^{-1}o + Bu) \\ o &= G(Cx + Du) \end{aligned} \tag{3.19}$$

Because the gradient tends to vanish in the training algorithm with respect to deep structures, a penalty term is introduced in the learning rule. The learning rule for θ at time t becomes:

$$\theta(t) = \theta(t-1) + \alpha\Big(\theta^* - \theta(t-1)\Big) + \beta\Phi \tag{3.20}$$

where θ^* represents the suboptimal state that is the result of applying the least square method through several iterations. Φ can be called the penalty term. α is the learning rate and β decides the weight of Φ. In fact, β is set to increase gradually by a small proportion to escape from the suboptimal state. This method is experimentally proved to be robust both in learning speed and network performance.

3.6.2 Leaky integrator based method

An earlier and more simple method is to bring past information to the current learning point. However, this approach is unable to handle the significance of each time step information

in the past. The NARX method introduced in [73] is inspired by this mechanism. A recent method makes use of leaky integrator factors in each NN learning unit while keeping the forward and the backward procedures pass efficient [74]. The cost function is computed by the cross-entropy method: $E(t,o) = -\sum_i t^i log(o^i)$. Here, t and o are the desired and the actual outputs, respectively. The output of the hidden units at time step t in the following compact form:

$$y_t = f\Big(\sum_{i=1}^{t}\Big(\lambda^{i-1}Wy_{t-i} + \hat{\lambda}^{i-1}\hat{W}x_{t-i}\Big)\Big) \tag{3.21}$$

The leaky integrator is defined as: $S_t^y = \sum_{i=1}^{t}\lambda^{i-1}y_{t-i}$ and $S_t^x = \sum_{i=1}^{t}\hat{\lambda}^{i-1}x_{t-i}$. To represent the leaky integrator in an iterative fashion, one can write $S_t^y = y_{t-1} + \lambda S_{t-1}^y$ and $S_t^x = x_{t-1} + \hat{\lambda}S_{t-1}^x$. The λ and $\hat{\lambda}$ are constrained between $(0,1)$ by applying the sigmoid function, assigning them to $1/(1+exp(-\ell))$, and learning the unconstrained ℓ accordingly.

3.6.3 Long short term memory

This method has gained significant success in solving the long term dependency problems [26]. This method was very successful in solving a range of pattern recognition, like speech signal recognition, handwriting recognition and time-series problems. The LSTM can handle learning problems with considerable long term dependencies by utilizing special memory units located at the hidden layer. Each memory cell is built with a fixed self-connection. The error signal is trapped in the cell and cannot be changed. The output gate of the memory cell has to learn which error to trap by properly scaling them. Meanwhile, the input gate learns when to release the error, again by a scaling method. Then the error is truncated once it is allowed to leave to the memory cell. The design of such memory units allows the gradient of the error function to freely back-propagate through the network with possibly arbitary duration.

3.6.4 Genetic algorithm and Particle swarm optimization

The GA for estimating the network's parameters in RNNs has been attempted in [75]. The authors indicated that the GA is more robust than BPTS when dealing with a flower image classification problem. It is interesting to note that the authors applied two different GA methods: The first one concatenates all the weights into a single chromosome (termed whole-in-one); the second one uses four GAs for four set of weights used in RNN (termed 4-parallel). The latter performs better than the former. Different objective/fitness functions are used in these two methods. It is also reported that PSO may be applied as an effective replacement for GA in graph based learning models such as RNN and GNN. The advantage of PSO is that it helps to improve the computational time efficiency over the conventional GA model.

3.6.5 Hierarchical learning

Hierarchical architectures for modelling graph data structures were first proposed in [17]. The authors made use of a combination of unsupervised PMGraphSOM and supervised GNN, in which the PMGraphSOM is used as a pre-processor. The results obtained from the PMGraphSOM are then concatenated to the feature vector of the corresponding node, and this is then used to train the GNN. It was shown that this approach reduces (but not eliminates) the long term dependency problem [17].

Another approach that uses a hierarchical architecture for the modelling of graphs was proposed in [76, 77]. The approach takes advantages of a series of GNNs. The output of one GNN is used to re-label the associated nodes in the dataset. The re-labelled graph is then used to train another GNN. The procedure is repeated until no further improvement in classification accuracy is observed. It was empirically shown that such a layered approach produces better results than the baseline method that only utilizes one GNN for the learning task, and that two to three layers of GNNs should be sufficient in most cases. This finding

provides evidence that a layered learning architecture can reduce the long term dependency problem in the domain of graphs.

3.7 Conclusion

This chapter has given an overview of graph based machine learning algorithms. The history, definitions and key concepts were reviewed. This chapter first provided background on graph data representation and the formal definition of a graph learning application. Several neural network models, namely some generations of SOMs for structural data and graph based neural networks were explained. Existing approaches addressing the problem of recurrent/recursive architectures were reviewed. Furthermore, this chapter has provided insights into two well known graph kernel machine algorithms. The kernel machines will later be used to compare learning and prediction performance with corresponding neural network models.

Chapter 4

Problem descriptions and evaluation methods

4.1 Introduction

This chapter offers a description of datasets that will be used in this thesis for the evaluation of proposed methods and for comparison purposes. The datasets used are:

UK2006 and UK2007: These two datasets are widely used for benchmarking learning systems in their ability to classify web pages into spam or non-spam sites. This is a heavily unbalanced binary classification problem consisting of one very large graph for each of the data sets.

Mutag: A dataset for activity prediction in chemical molecules. This defines a regression type of learning problem consisting of a set of relatively small graphs.

INEX2008 A dataset pertaining to XML documents. This is a multi-class classification problem consisting of a set of medium sized graphs.

Physical activity prediction: Two datasets on physical activity prediction in young chil-

dren and adolescents respectively. This is a very recent dataset involving multi-class classification on a limited number of temporal sequences (of graphs).

This chapter is organized as follows: Section 4.2 presents an overview the UK 2006 and the UK 2007 web spam detection datasets. These are some of the most well-known and challenging benchmark problems in the domain of graphs. The relatively small Mutag dataset will be described in Section 4.3. The large text document categorization problem as posed by the INEX2008 dataset is presented in Section 4.4. The two datasets for the physical activity prediction in young children and adolescents will be described in Section 4.5 and Section 4.6 respectively. Section 4.7 will present several evaluation metrics that will be used in this thesis, and Section 4.8 summarizes this chapter.

4.2 The UK2006 and UK2007 web spam detection datasets

The UK2006 and UK2007 datasets are widely used as benchmark problems for the evaluation of prediction models which can explore information provided in the form of a web graph, each node in the graph is described by a high dimensional feature vector [18, 67, 78, 79, 80, 81]. They consist of large collections of hosts retrieved from the .UK top-level domains. The web pages are grouped together according to the location of their host. The hyperlinks between pages/nodes define the topology (known as host graph or HG) of the resulting directed graph of the web pages [78, 79].

There is a feature vector associated with each node in the graph. The feature vector contains 96 elements which describe the content of the associated host such as average word length, and number of words in the title (these are denoted as content based features or C in short). The vector also consists of a 41-dimensional description of the connectivity of the associated node such as PageRank, number of neighbors, TrustRank (these are named as link based features or RL). In addition, a third set of features is 138 in dimension, being the

transformation of link-based features, such as normalization, logarithm and so on (this set is denoted as transformed link-based features or TL). The feature vector associated with each node is a concatenation of these three sets: C, RL and TL, a total of 275 dimensions. Note that not all of the dimensions are independent of one another. In other words, the node can be described by a surface of lower dimensions; this insight gives rise to the regularization studies conducted in this thesis later.

Some of the nodes come with a manually labelled class label: spam or normal (some other nodes labelled as ”unknown” will not be recognized as a separate class since it is not clear if the nodes belong to normal or spam ones). By the nature of the problem, the number of nodes labelled as normal would be much higher than the number of nodes which are labelled abnormal or spam. This gives rise to the issue of imbalance output class distribution which will be investigated later in this thesis. Since the labels are obtained manually, by a group of human volunteers, hence the number of labelled nodes is small relative to the size of the web.

These two datasets are ideal for the purpose of graph based learning investigations, since they contain both relational and feature data. A traditional learning model can only learn on the feature data, while a graph based model can exploit both topological and feature data.

The challenge is to classify the unlabelled nodes in the graph into either *spam*, or *non-spam* ones. For a detailed discussion on what constitutes spam on the Web the interested reader is referred to detailed descriptions readily available in [78, 79, 82]. For the purpose of this thesis, it suffices to indicate that spam pages are web documents whose content or hyperlinks are designed to inflate their rank on Web search engines. Hence, web pages may be classified as content-based spam, link-based spam, or both. The two given datasets labels a document simply as spam if it falls into any one of the three spam categories. What makes this dataset of interest for this thesis is that the learning problem requires from a learning system the ability to model content as well as the topology of the dataset. Relevant properties

Table 4.1: Relevant details of the UK webspam datasets.

Dataset		**UK2006**	**UK2007**
Number of hosts		11,402	114,529
Number of pages		77M	100M
Host graph (HG) Topology	Number of graphs	1	1
	Number of nodes	11,402	114,529
	Number of links	730,774	1,885,820
	ϕ number of links per node	64	16
	Max. out-degree	5,994	51,692
Feature sets	Content-based features (C)	96	96
	Raw link-based features (RL)	41	41
	Transformed link-based features (TL)	138	138
Train set	size	9,551	4,275
	spam	767	222
	non-spam	7,472	3,776
	unknown	1,312	277
Test set	size	1,851	2,204
	spam	1,250	122
	non-spam	601	1,933
	unknown	0	149

of both datasets are given in Table 4.1. From Table 4.1 it can be observed that the UK2007 dataset is about 10 times the size of the UK2006 dataset while containing fewer training samples, much fewer links between nodes, and much larger deviations of the number of outgoing links. As a consequence the UK2007 graph is more sparse than the UK2006 one. This implies that the average length of a path between two labelled nodes is longer for the UK2007 dataset. A main challenge with the UK2007 dataset is to overcome possible path dependencies between (labelled) nodes in the graph.

For the UK2006 dataset, there are 8,239 training hosts in which the number of spam hosts accounts for a small portion of $\approx 9.3\%$. The testing dataset contains 1,851 hosts with more than 67.5% of spam ones. Interestingly, the proportion between spam and non-spam hosts in the training and testing set is significantly different. In particular, while the number of spam hosts is two fold that of the non-spam hosts in the testing set, the number of spam hosts in the training set is just under one ten that of non-spam hosts. This raises a challenge for the prediction models that the training dataset might not cover all possible behaviours of the underlying system, which the testing dataset may require.

Regarding the UK2007 dataset, there are approximately 10 times as many hosts as in the

UK2006 dataset. However, the number of labeled hosts is small (6,479). The percentage of spam hosts involved in both the training and testing datasets is around 5%. This imbalanced nature of UK2007 is much more severe than that of the UK2006 dataset.

4.3 Mutagenesis dataset

Mutag is a relatively small benchmark dataset [83]. The purpose of this dataset is to investigate whether it is possible to predict the mutagenicity of chemical componds/molecules. The term mutagenesis refers to a biological process that drives the genetic constitutions of an organism. This usually leads to a mutation in genes. Mutagenesis can take place spontaneously in nature or is physically or chemically activated by a mutagen. Mutagenesis can lead to cancer or some other disease. Hence, there is considerable interest in researching methods that can predict whether a medical compound is mutagenic or non-mutagenic. The mutagenesis dataset can be referred to as a classification or regression problem [83]. Each molecule is given a real valued number representing the capability if it is mutagenic (denotes mutagenicity) or not. We will use the dataset as a binary classification problem for consistency with previous researches. This also relates to common practice in that harmful substances are classified as dangerous (i.e prohibited) and not-so dangerous (permitted), by using a threshold as defined by Safe Work Australia. Hence, a medical compound is either categorized as active (mutagenicity $\geqslant 1$) or inactive (mutagenicity < 1). They are correspondingly assigned to the class labels +1 and -1. The properties of this dataset are sumarized in Table 4.2.

The table shows that for the 230 medical molecules three different datasets are derived as three separated benchmark problems. Specifically, these are the regression friendly part, regression unfriendly part, and the whole data. The regression friendly part contains 188 molecules, while the remaining molecules are attributed to the regression-unfriendly part. The whole data refer to all 230 medical compounds. A molecule can have descriptive fea-

Table 4.2: General properties of the Mutagenesis datasets.

Dataset		**Whole data**	**Friendly part**	**Unfriendly part**
Number of molecules		230	188	42
Number of active mutag		138	125	13
Number of inactive mutag		92	63	29
Atom bond Topology	Number of graphs	230	188	42
	Number of nodes	5894	4893	1001
	Number of links	6309	5243	1066
	ϕ number of nodes per graph	25	26	23
	ϕ number of links per graph	27	27	25
	ϕ number of links per node	1	1	1
	Max. out-degree	4	4	3
Feature sets	Atom type and charge (AB)	10	10	10
	Chemical measurements (C)	2	2	2
	Precoded structural attributes (PS)	2	2	2
Evaluation method		10-fold	10-fold	L-o-o

tures at the atomic level or at the molecular level. In particular, a single atom's properties include atom type and its charge. There are 9 different types of atom, namely C, H, O, N, F, Cl, Br, I and S. A one-hot encoding technique is used to represent an atom type in binary form. Thus, the encoding technique produces a 10-dimensional vector consisting of a nine-dimensional binary vector representing the atom type and a one-dimensional element representing its charge value. In the following, this 10-dimensional vector will be referred to as the AB feature vector. At the compound level, there are two chemical measurement features named as lowest unoccupied molecule orbital and water/octanol partition coefficient (denotes C features), and other two features named as precoded structural attributes (denotes PS features). In general, there are three sets of features AB, C and PS what will be used in our experiments.

Finally, the atom bond provides the conceptual material to construct the graph topology of a compound. One molecule is modelled by a graph whose nodes represent atoms and links are the bond. This creates multiple graph problems. The evaluation methods used for those datasets are indicated as 10 fold cross validation for the regression friendly part and whole datasets, and leave-one-out (l-o-o) for the regression unfriendly part.

Table 4.3: The INEX 2008 dataset brief information

	Total	Train	Test
Size of corpus	≈751 Mb	≈75 Mb	≈676 Mb
Number of documents	114,366	11,437	102,929
Number of distinct words	166,619	115,002	
Graph topology	636,187 directed links, average 5.5 links/node		

Table 4.4: The INEX 2008 categories: Number of documents

ID	Category names	Total	Train	Test
471	United states	29,980	2,945	27,035
49	Reference	14,905	1,474	13,431
339	Sports	9,435	915	8,520
252	Social institutions	8,199	866	7,333
1530	Politics by region	7,749	789	6,960
1542	Urban geography	7,121	696	6,425
10049	Human behavior	6,933	679	6,254
380	Fiction	6,262	639	5,623
897	Categories by nationality	6,166	637	5,529
4347	Americas	6,088	592	5,496
9430	Demographics	3,948	405	3,543
1310	Tourism	2,880	294	2,586
5266	Art genres	2,544	264	2,280
323	Sociology	1,165	128	1,037
1131	Europe	991	114	877
Total		114,366	11,437	102,929

4.4 The INEX 2008 dataset

This is a large set of text documents in XML format. The dataset has been used widely as a benchmark categorization problem since 2008 [84]. The corpus is a subset of the Wikipedia XML Corpus [85]. 114,336 documents have been extracted from the original data cohort. The links between those documents have also been derived for the purpose of document graph construction. These links either correspond to the links created by the authors of the Wikipedia articles or automatically generated by Wikipedia. Table 4.3 gives some statistics about the documents. The documents allocated to the training set are much fewer than the test set. This would raise difficulty in generalization prediction since the learning feature

space may not cover that of unknown samples. The dataset features 636,187 directed links among available documents. Each document is connected by 5.5 links (in-link) on average, and provides 5.5 links to other documents (out-link) on average. Table 4.4 gives information about the 15 different categories and the corresponding number of documents existing in the training and test sets. In addition to the much larger testing set, the class distribution is severely imbalanced in this dataset. The largest class, "United states" is approximately 31 times the size of the smallest one "Europe". This unbalance in class sizes can also pose a challenge for prediction models.

For the experiments, this thesis will use the following data representation method. Each document is represented by a feature vector. Each element in the feature vector is computed based on Term frequency (TF) or with Inverse document frequency ($TF.IDF$) values. One can define a list of terms $T = t_1, t_2, ..., t_{|T|}$ extracted from the list document $D = d_1, d_2, ..., d_{|D|}$. The term frequency TF is the relative frequency of term t_j in a document d_i:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_{l=1}^{|T|} n_{i,l}}, \tag{4.1}$$

where $n_{i,j}$ is the number of occurrences of term t_i in document d_j, normalized by the total number of terms in d_j. The more frequent the term t_i, the higher the $tf_{i,j}$.

On the other hand, the IDF measures the discriminatory power of a term t_j:

$$idf_j = log\frac{|D|}{|\{d_i : t_j \in d_i\}|}, \tag{4.2}$$

where $|D|$ is the cardinality of documents in the training corpus and $|\{d_i : t_j \in d_i\}|$ is the number of documents containing term t_j. The less frequent the term t_j in the dataset, the higher the idf_j. One can define the weight $TF.IDF$ of a term within a document as follows:

$$tf.idf_{i,j} = tf_{i,j} * idf_j \tag{4.3}$$

The $TF.IDF$ means that the more frequent the term t_j in d_i and less in other documents, the higher is the weight $tf.idf_{i,j}$ Therefore, a feature vector can be represented by the concatenated TF or $TF.IDF$ values of every existing term in that given document.

4.5 Preschool children physical activity cohort

The dataset consists of physical activity data collected from 11 pre-school children within the age range of 3-6 years. The dataset was collected in the laboratory environment in 2012. Participants were requested to complete 12 protocol activity trials over two laboratory visits scheduled within a 3 week period. The activities in visit 1 included watching TV (TV), sitting on floor being read to (reading), standing making a collage on a wall (art), walking (walking), playing an active game against an instructor (active game), and completing an obstacle course (obstacle course). Six more activities were preformed at visit 2: Sitting on a chair playing a computer tablet game (tablet), sitting on floor playing quietly with toys (quiet play), treasure hunt (treasure hunt), cleaning up toys (clean up), bicycle riding (bicycle), and running (running). Each trial lasted approximately 4-5 min. A summary of these 12 activities and corresponding activity type classes is shown in Table 4.5. The main purpose of the grouping is that each PA activity class is more or less equivalent in the amount of energy expended, while running and walking are the two most popular actions which are hence treated as separate classes.

The participants were wearing an ActiGraph GT3X+ sensor (an accelerometer) on three body positions hip, left wrist and right wrist. The acceleration information is recorded at 100 Hz. The sensors measured and stored triaxial acceleration of those body's parts. As a result, there are three 3-dimensional datasets extracted from each of the three accelerometers, denoted as *Hip* data, *Lwr* data and *Rwr* data.

Table 4.5: All physical activity (PA) types

Name	Description	Activity type class
TV	Watching TV	Sedentary
ST	Story time	
iP	Playing iPad	
QP	Quiet play	
CO	Collage	Light lifestyle activities
TH	Treasure hunt	
CU	Cleaning up	
BR	Using bicycle/tricycles	Moderate-to-vigorous activities
OC	Obstacle course	
BB	Bean bags	
WA	Walk	Walking
JG	Running	Running

4.6 School-age children and Adolescence cohort data

The second data cohort in the field of physical health is the school children and adolescents PA problem denoted as *SCA data*. This is a relatively large dataset consisting of 100 participants in the age group 5 to 15 years. The data was collected at the Queensland University of Technology in 2010. They also used accelerometers but sampled the information at 30Hz which were positioned at the waist of the participants using flexible elastic belts. Each participant also performed 12 activity trials: lying down, handwriting, laundry task, throw and catch, comfortable overground walk, aerobic dance, computer game, floor sweeping, brisk overground walk, basketball, overground run/jog, and brisk treadmill walk. All activity trials lasted approximately 5 minutes, except for the lying down trial, which was completed in 10 minutes. Based on the movement pattern and the amount of EE, these activities are categorized into 5 classes, similar to the case of preschool children data. Those classes are sedentary (lying down and handwriting computer game), light household (HH) activities or games (floor sweep, laundry task, and throw and catch), moderate-to-vigorous games or sports (aerobic dance and basketball), walking (comfortable overground walk, brisk over ground walk, and brisk treadmill walk), and finally running (overground run/jog) [86]. Both

Table 4.6: General confusion matrix.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

the preschool children dataset and the adolescence dataset can be modelled as a time series or temporal sequence classification problem, since each activity is composed of a series of time-step based acceleration information. The information at the current time-step may more or less be influenced by information that happened in the past.

4.7 Evaluation approaches

Various evaluation metrics will be applied in our experiments. For the classification problems, Accuracy (ACC), (macro/micro) Recall, F1, and Area under the ROC curve (AUC) indicators will be utilized. The Root mean square error (RMSE) and (absolute) Mean bias are the evaluation metrics for the regression problems.

Accuracy (ACC): ACC represents the percentage of correctly predicted examples over the dataset size. On the basis of the confusion matrix given in Table 4.6, the accuracy is calculated as follows. $ACC = \frac{TP+TN}{TP+FN+TN+FP}$. Despite its popularity, the ACC performance meassure is limited in expressing the true performance of a classifier on unbalanced learning problems.

Recall: Recall is defined as the proportion of target documents returned. There two conventional methods of calculating the performance of a text categorization system based on recall, namely micro-averaging and macro-averaging. Micro-averaged values are calculated by constructing a global contingency table and then calculating recall using these sums. In contrast, macro-averaged scores are calculated by first calculating

precision and recall for each category and then taking the average of these. The difference between these is that micro-averaging gives equal weight to every document while macro-averaging gives equal weight to every category.

$$R_{micro} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FN_i} \tag{4.4}$$

$$R_{macro} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{TP_i}{TP_i + FN_i} \tag{4.5}$$

F-measure (F1): F1 can reflect more accurately the generalization performance of a classifier in an imbalanced dataset. The larger the F-measure value the better the performance on the positive class. Its calculation is a balance between precision $Pr = \frac{TP}{TP+FP}$ and recall $Re = \frac{TP}{TP+FN}$ in that the F-measure is $F_1 = \frac{2*Pr*Re}{Pr+Re}$

Area under the curve (AUC): AUC refers to the probability that a learning model ranks a randomly chosen positive sample higher than a randomly chosen negative one. In fact, if a model classifies the negative examples correctly, then a poor performance in predicting the positive examples would be reflected by a low AUC value.

RMSE and (absolute) Mean bias: For a regression problem, the evaluation metrics used are (absolute) mean bias and RMSE. Absolute mean bias is more indicative for assessing the overall performance of a prediction model than the mean bias, since the mean bias is sometimes very small if the positive and negative values are compensated together. The (absolute) mean bias is computed by taking the mean of (absolute) distance between predicted and measured output for all testing samples. The RMSE is taking the square root of the sum of all squared distances between predicted and measured output for every testing sample.

4.8 Conclusion

Benchmark problems that will be used for the experiments in this thesis have been described. The data representation of these sets of data is available in both topological form and (feature) vectorial form. Four of the benchmark problems express properties with some similarities. For example, each data sample can be viewed as a node in a graph, which is constructed via the relational information between nodes. The problems derived from the Web spam detection and INEX 2008 data are single graph node-focused learning problems. It is intuitive that the nodes will be classified into different classes. On the other hand, the Mutag dataset is as multiple graph classification problem where each graph represents a single molecule, the task of which is to correctly classify each graph to the corresponding class. In contrast, the two physical activity datasets contain temporal sequences. Each input sample is represented by a single time series or temporal sequence. These problems could be generalized to multiple graph (sequence) problems. The two datasets will be used in Chapter 8 for both classification and regression experiments in this thesis.

Chapter 5

Incorporating Input Graph Topology in Neural Networks

5.1 Introduction

This chapter presents a systematic study on a set of benchmark problems to investigate whether and when the modeling of input graph structures has an advantage when compared to corresponding classic machine learning models. The benchmark dataset includes two Web spam detection problems UK2006 and UK2007, and three medical compound Mutag regression problems, as described in Chapter 4. Research questions that will be answered in this Chapter are (1) whether modelling the inputs as graphs confers any advantages over those of classic machine learning algorithms, (2) if so, under which conditions do these materialize and by how much? (3) Is the incorporation of the inputs as graphs worth the effort in terms of the results obtained?

A careful step-by-step approach is taken by considering the simpler learning methods first, then by applying models of increasing complexity. The aim is to obtain baseline results to which more advanced methods can be compared with. This should allow for conclusions on how graph-based models compare with non-graph based models when modelling a struc-

tured domain in practice. In order to highlight the role of incorporating input graph topology in the neural network architecture, the experimental results are presented in a non-traditional manner. In particular, we will show results on datasets using different neural network architectures, first without using any input graph information, and then we will progressively repeat the same experiments by incorporating a graph topology. Thus, we will be able to show the effectiveness of incorporating the input graph topology in neural network architectures, trained using an unsupervised learning technique, or supervised learning technique, or a combination of both. Along with the experiments, several integrated learning models are proposed, being motivated by the hierarchical and deep learning regime.

This chapter is organized as follows: The experimental setting will be described in Section 5.2. Section 5.3 presents the results of experiments when deploying classic unsupervised learning architectures, and supervised learning architectures without assuming any input graph topology, i.e., the set of links is assumed to be zero. Section 5.4 studies the case in which the set of links is non-zero. A comparison and discussion is given in Section 5.5. Finally, some conclusions are drawn in Section 5.6.

5.2 Evaluations and Experimental Setting

This section will give the evaluation methods being used and the order of significance of these, and will present the procedure for the experiments.

5.2.1 Evaluation consideration

Three different evaluation metrics are applied including AUC, ACC and F1. These three metrics are not always uniformly behaved. Hence, it is important to first understand the expressive power of each of these metrics and then to define an order of importance in any given situation.

For example, when learning from a significantly imbalanced dataset, a simple learner may predict all samples to belong to the majority class. Hence, this might result in a high prediction accuracy. The UK2007 dataset is the one typical example that a weak learner would obtain a relatively good accuracy of 88% by simply assigning every webpage to the non-spam class. On the other hand, if the learning model classified all spam pages correctly while classifying all other pages in a random fashion then the accuracy would be reduced greatly to 53%. The reason is that the total number of correctly predicted hosts can decline when the dataset in imbalanced, thus negatively affecting the ACC performance. Hence, by relying on the ACC evaluation, one may wrongly underestimate a learner that correctly classifies all spam hosts (because truly predicting spam is the main purpose of Web spam detection problems). The ACC is a good performance indicator when the dataset is well balanced.

If the class distribution is not severely imbalanced as, for example, with the Mutag dataset, then the ACC could be a more indicative metric. In the literature, both AUC and ACC have been primarily used to evaluate models which were trained on the Web spam detection and Mutagenesis problems [78, 79, 87, 88]. In practice, the AUC exhibits similar properties to the F1 method since both measure the accuracy balance between minority and majority class, although the AUC appreciates the minority class samples a little more. Hence, we use three evaluation metrics and rank the significance of those metrics in descending order as follows: AUC $\rightarrow$ F1 $\rightarrow$ ACC for Web spam detection problems and ACC $\rightarrow$ AUC $\rightarrow$ F1 for the Mutagenesis problem. The reader is reminded to take this ordering of metrics into consideration when experimental results are compared later in this chapter.

5.2.2 Experimental Procedures

Unsupervised and supervised learning systems are considered in this chapter. The different nature of these learning methods require a different treatment when selecting training mod-

els. The SOM and PMGraphSOM were trained for at least 500 iterations, then the SOMs which produce the best ACC performance on the training set selected. Traditionally, an unsupervised model assumes that a target vector is not available and hence, a probability or distance index such as the Davies-Bouldin method [89] would be applied. However, in our experiments, the target vectors are available for all problems. As a result, ACC will be used for model selection, since it is better for demonstration purposes.

When training supervised models, a validation set consisting of 10% randomly selected data from the training set is created. Various model architectures are tried and the best one is selected via the validation set performance over 1000 training iterations. We quantitatively observed that after that number of training epoches, the network's error does not decrease significantly. Due to the variation of evaluation methods, the key indicator for model selection is AUC in cases of the Web spam detection problems and ACC for the Mutag dataset. Other learning parameters like SOMs' map sizes or the size and number of hidden layers will be shown in the first column of the results tables. The hybrid models simply use the parameters from the best component learning modules. For instance, the SOM+MLP and PMGraphSOM+MLP models are constructed from the best previously reported components SOM/PMGraphSOM and MLP respectively.

Each experiment is repeated 10 times subject to random initial conditions for Web spam problems. For the Mutagenesis learning, the experiments are repeated 3 times. 10-fold cross validation evaluation is applied for the regression friendly and the whole Mutag dataset while leave-one-out is used for the regression unfriendly part (due to its small size). This is referred to as 10-fold and l-o-o in the subsequent tables.

The experimental results indicate the average performance and corresponding standard deviation in brackets over 10 and 3 runs for the Web spam and Mutag problems, respectively. The small value of standard deviation reflects the representative and stable performance results. Because the performance on the Mutag datasets is calculated through cross validation

approaches, it will be shown that the standard deviations resulting from Mutag experiments are normally higher than that of Web spam classification.

In the following, the column **T** in the results tables indicates whether or not topological information in the input data was modelled by corresponding models. For example, the SOM and MLP cannot model the graph topology resulting in the associated entries in this column being left blank. Additionally, AUC is not possible for the SOM and PMGraphSOM because their output is only coordinates on the activation map.

5.3 Classic neural network architectures

The SOMs and MLPs are trained on the vectorial input data. The results shown in this section will form a baseline to compare with the graph neural network models.

5.3.1 Self organising feature map

The empirical results of SOM for the UK2006, UK2007 and for the Mutag datasets are given in Table 5.1. The map sizes of SOM training on the Web spam problems were selected within 101x70, 80x66, 73x55 and 64x43. The size of maps are not too large to avoid the unnecessary computational cost. For the Mutag dataset, the maps were selected within 59x44, 40x30, 35x25 and 30x22. The reason for setting the map sizes for the mutag dataset to be smaller than the Web spam ones is that their set of input vectors is much smaller compared to the Web spam datasets. We define a compression ratio c_r as the number of samples divided by the number of neurons. The larger the c_r values, the higher compression rate or higher density the samples would be projected on the map. With respect to different SOMs' maps, c_r for the UK2006 dataset ranges between the highest magnitudes $[1.29, 3.44]$. It is hence expected that a high ACC performance of SOMs on the UK2006 problem would hardly be achieved. However, the SOM's ACC performance practically likewise depends

Table 5.1: SOM results for the UK2006, UK2007, and the Mutag dataset.

SOM			F1		ACC	
Map	**Features**	**T**	**Train**	**Test**	**Train**	**Test**
UK 2006 dataset						
64x43	C		0.458 [0.093]	0.358 [0.069]	0.921 [0.012]	0.419 [0.068]
64x43	RL		0.597 [0.025]	0.586 [0.141]	0.938 [0.008]	0.535 [0.056]
64x43	C+RL		0.602 [0.102]	0.634 [0.043]	0.901 [0.048]	0.572 [0.034]
UK 2007 dataset						
73x55	C		0.346 [0.016]	0.072 [0.010]	0.950 [0.026]	0.732 [0.035]
73x55	RL		0.514 [0.020]	0.099 [0.006]	0.949 [0.018]	0.697 [0.079]
73x55	C+RL		0.460 [0.018]	0.088 [0.015]	0.951 [0.028]	0.750 [0.025]
Mutag: Whole dataset						
40x30	AB		0.849 [0.006]	0.756 [0.055]	0.808 [0.009]	0.684 [0.071]
40x30	AB+C		0.838 [0.012]	0.780 [0.096]	0.800 [0.013]	0.735 [0.105]
40x30	AB+C+PS		0.823 [0.011]	0.769 [0.093]	0.787 [0.009]	0.726 [0.094]
Mutag: Regression friendly part						
40x30	AB		0.893 [0.003]	0.825 [0.043]	0.850 [0.004]	0.761 [0.057]
40x30	AB+C		0.876 [0.012]	0.835 [0.121]	0.834 [0.015]	0.791 [0.134]
40x30	AB+C+PS		0.978 [0.004]	0.879 [0.080]	0.971 [0.006]	0.844 [0.098]
Mutag: Regression unfriendly part						
30x22	AB		0.870 [0.018]	0.500 [0.304]	0.913 [0.017]	0.682 [0.162]
30x22	AB+C		0.774 [0.031]	0.433 [0.335]	0.881 [0.018]	0.715 [0.224]
30x22	AB+C+PS		0.773 [0.032]	0.440 [0.343]	0.881 [0.018]	0.740 [0.241]

on the portion of input samples which are linearly separable. Similarly, the small c_r values (ranging between $[0.02, 0.06]$) regarding Mutag regression unfriendly data, does not guarantee a high SOM's ACC performance. The selection of map sizes here is thus relative for each datasets. The other learning parameters for SOM were selected as follows $\alpha(0) \in \{0.01, 0.1, 0.3, 0.5, 0.8\}$ and $\sigma(0) \in \{5, 10, 17, 29, 40\}$. Via trial-and-error, we empirically found that the value $\alpha(0) = 0.1$ is best applied for all datasets, while $\sigma(0) = 10$ and $\sigma(0) = 29$ are suitable values for Mutag and Web spam problems, respectively.

The SOM works best on learning problems that consist of linearly separable groups of data. This however is not the case for many real world applications. It is hence expected that the generalization performance of the SOM would be quite poor for the spam detection datasets. This can be observed in Table 5.1. The high ACC on the training sets could be observed for all datasets. On the other hand, as expected, the generalization ACC performance for the UK2006 dataset is poorest when compared with others. The extreme case here is the considerably low F1 testing performance for the UK2007 dataset. This can be explained

Table 5.2: MLP performances on the UK2006, UK2007 and Mutagenesis dataset.

MLP			AUC		F1		ACC	
Hid. units	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 Dataset								
25	C		0.800 [0.011]	0.805 [0.021]	0.491 [0.007]	0.653 [0.034]	0.885 [0.009]	0.637 [0.024]
37	RL		0.931 [0.005]	0.815 [0.020]	0.652 [0.005]	0.728 [0.030]	0.911 [0.008]	0.694 [0.024]
37	C+RL		0.946 [0.005]	0.865 [0.017]	0.704 [0.009]	0.791 [0.021]	0.925 [0.008]	0.752 [0.019]
UK 2007 Dataset								
25	C		0.637 [0.028]	0.659 [0.023]	0.364 [0.011]	0.298 [0.019]	0.947 [0.005]	0.933 [0.005]
25	RL		0.670 [0.010]	0.635 [0.017]	0.221 [0.015]	0.148 [0.009]	0.894 [0.011]	0.872 [0.016]
40	C+RL		0.660 [0.041]	0.673 [0.024]	0.424 [0.020]	0.324 [0.034]	0.949 [0.007]	0.930 [0.017]
Mutag: Whole dataset								
4	AB		0.493 [0.019]	0.484 [0.135]	0.754 [0.006]	0.751 [0.059]	0.615 [0.018]	0.612 [0.088]
16	AB+C		0.835 [0.007]	0.802 [0.068]	0.835 [0.009]	0.822 [0.066]	0.797 [0.013]	0.783 [0.079]
13	AB+C+PS		0.876 [0.007]	0.840 [0.061]	0.870 [0.007]	0.852 [0.052]	0.843 [0.008]	0.825 [0.050]
Regression friendly part								
10	AB		0.500 [0.027]	0.462 [0.127]	0.813 [0.016]	0.805 [0.068]	0.703 [0.032]	0.694 [0.095]
13	AB+C		0.906 [0.013]	0.837 [0.108]	0.896 [0.009]	0.876 [0.086]	0.861 [0.011]	0.836 [0.114]
16	AB+C+PS		0.939 [0.011]	0.851 [0.096]	0.930 [0.008]	0.912 [0.082]	0.908 [0.011]	0.885 [0.105]
Regression unfriendly part								
4	AB		0.583 [0.146]	0.500 [0.237]	0.564 [0.056]	0.385 [0.259]	0.619 [0.206]	0.503 [0.249]
10	AB+C		0.777 [0.043]	0.517 [0.242]	0.722 [0.022]	0.506 [0.347]	0.844 [0.036]	0.738 [0.273]
7	AB+C+PS		0.751 [0.049]	0.517 [0.242]	0.705 [0.025]	0.529 [0.334]	0.848 [0.034]	0.815 [0.175]

via a detrimental effect on SOM caused by the severely imbalanced class distribution of this dataset. In addition, the high ACC results for this dataset is also a good implication that most data is assigned to the overwhelmingly larger class (containing normal hosts).

The testing performance of SOM, on the other hand is seen better for the mutag datasets. This implies that a major proportion of the input patterns may be linearly separable. The standard deviation regarding the regression unfriendly data is larger than the regression friendly and the whole Mutagenesis problem due to the leave-one-out evaluation procedure for the regression unfriendly data. The standard deviation is computed over 42 outputs, which is a much larger range than the case of 10 fold cross validation.

5.3.2 Multilayered neural networks with a single hidden layer

In this experiment, each MLP is configured with only a single hidden layer. The number of hidden units are selected from $\{17, 25, 31, 37, 40\}$ for the Web spam datasets, and from $\{4, 7, 10, 13, 16\}$ for the Mutagenesis dataset. An adaptive learning rate mechanism is used during the learning process. All the input data are normalized. The empirical result is shown in Table 5.2.

As can be seen in Table 5.2, the single-hidden-layered MLP generally produces better results in terms of classification and generalization performance, compared to the SOM. Both F1 and ACC experience a significant improvement. In particular, the testing performance improves in the range from 15% to 23%, whereas the training ACC remains high for both the UK2006 and UK2007 datasets. Interestingly, both training and testing AUC performance for UK2006 are much higher than for the UK2007 dataset.

The enhancement in testing performance is much less conspicuous for the Mutag data. ACC witnessed an increase by about 4% to 8% compared to the SOM results. AUC performance for the regression friendly part and whole dataset is relatively high, which is not observed for the regression unfriendly part. In addition, the standard deviations associated with the MLP results are noticeably smaller. This indicates that the generalization ability of MLP is much more representative than that of the SOM.

5.3.3 MLP network with multiple hidden layers

This set of experiments are made inspired by a deep MLP model which contains three hidden layers, and was investigated by Lecun et al. [19]. We will present the results of MLPs being configured with two or three hidden layers. For short, the three hidden layer MLPs is denoted as Lecun5 MLP. For the two hidden layered MLP, the hidden neuron number in each hidden layer is selected from $\{7, 9, 13, 16, 20\}$ for the Web spam datasets and from $\{3, 5, 7, 10, 12\}$ for the Mutagenesis dataset. In the Lecun5 MLP experiments, the number of hidden neurons is chosen from $\{4, 8, 10, 16, 20\}$ for the Web spam datasets and from $\{2, 4, 6, 8, 10\}$ for the Mutagenesis dataset. The training procedure and learning parameters are established the same as a one hidden layer MLP. Table 5.3 and Table 5.4 summarize the results of two and three hidden layer MLPs, respectively.

As is evident from the results shown in the tables, the MLPs with additional hidden layers produce a modest performance improvement for the Web spam data, and result in

Table 5.3: Two hidden layer MLP performances on the UK2006, UK2007 and Mutag data

2 hidden layer MLP			AUC		F1		ACC	
Hid. units	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 Dataset								
16-9	C		0.812 [0.004]	0.799 [0.004]	0.513 [0.003]	0.669 [0.023]	0.889 [0.004]	0.649 [0.016]
20-13	RL		0.930 [0.001]	0.812 [0.004]	0.652 [0.005]	0.740 [0.030]	0.910 [0.007]	0.695 [0.022]
16-13	C+RL		0.949 [0.004]	0.865 [0.009]	0.740 [0.009]	0.767 [0.013]	0.938 [0.003]	0.731 [0.011]
UK 2007 Dataset								
16-13	C		0.636 [0.038]	0.670 [0.035]	0.326 [0.035]	0.293 [0.013]	0.937 [0.014]	0.930 [0.010]
9-13	RL		0.632 [0.004]	0.603 [0.005]	0.191 [0.005]	0.141 [0.008]	0.876 [0.007]	0.845 [0.010]
20-9	C+RL		0.654 [0.015]	0.682 [0.017]	0.371 [0.038]	0.286 [0.027]	0.943 [0.009]	0.932 [0.009]
Mutag: Whole dataset								
10-7	AB		0.493 [0.029]	0.465 [0.119]	0.750 [0.005]	0.740 [0.049]	0.609 [0.014]	0.594 [0.064]
12-5	AB+C		0.835 [0.008]	0.804 [0.069]	0.835 [0.008]	0.815 [0.064]	0.798 [0.012]	0.775 [0.076]
10-7	AB+C+PS		0.877 [0.008]	0.843 [0.067]	0.873 [0.007]	0.851 [0.063]	0.847 [0.008]	0.828 [0.061]
Mutag: Regression friendly part								
7-8	AB		0.507 [0.026]	0.529 [0.137]	0.807 [0.016]	0.789 [0.070]	0.692 [0.031]	0.672 [0.098]
8-5	AB+C		0.900 [0.014]	0.834 [0.106]	0.896 [0.009]	0.871 [0.090]	0.858 [0.013]	0.831 [0.115]
12-8	AB+C+PS		0.930 [0.014]	0.854 [0.095]	0.930 [0.008]	0.903 [0.091]	0.908 [0.010]	0.877 [0.111]
Mutag: Regression unfriendly part								
8-3	AB		0.353 [0.041]	0.339 [0.150]	0.479 [0.051]	0.443 [0.261]	0.354 [0.058]	0.372 [0.197]
12-5	AB+C		0.674 [0.026]	0.517 [0.242]	0.701 [0.028]	0.562 [0.363]	0.857 [0.017]	0.815 [0.209]
12-5	AB+C+PS		0.676 [0.023]	0.517 [0.242]	0.699 [0.026]	0.562 [0.363]	0.856 [0.016]	0.840 [0.182]

largely unchanged performance for the Mutag data. It is known that multiple hidden layer MLPs are better suited to significantly non-linear problems. This improvement is offset by the detrimental effects of the long-term dependency problem which become more obvious with the deeper network architectures. The situation gets even worse for the Lecun5 MLP when compared to the two-hidden layer MLP, since a slight reduction is observed in the Lecun5 MLP learning performance. Nevertheless, the degradation is not very significant. This might reflect the fact that the long-term dependency is not a significant problem for non-recursive neural processing. This statement will be further clarified in the following sections.

5.3.4 Multi-staged Multilayered feedforward neural network

This section examines the effect of changing the learning architecture from a multiple hidden layer MLP into several single hidden layer MLPs, one for each hidden layer. These MLPs are stacked together in a hierarchical manner. The output of one MLP forms part of the input to the next MLPs. All MLPs are trained separately using the class labels from respective learning sets. The approach helps to add a relaxation between different hidden layers, which

Table 5.4: Lecun5 MLP performances on the UK2006, UK2007 and Mutag datasets

Lecun5 MLP			AUC		F1		ACC	
Hid. units	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 Dataset								
20-8-16	C		0.807 [0.008]	0.800 [0.002]	0.503 [0.006]	0.679 [0.022]	0.884 [0.005]	0.655 [0.016]
10-16-8	RL		0.930 [0.002]	0.806 [0.005]	0.646 [0.008]	0.735 [0.033]	0.909 [0.007]	0.689 [0.025]
16-10-4	C+RL		0.946 [0.003]	0.863 [0.006]	0.733 [0.007]	0.750 [0.027]	0.938 [0.003]	0.718 [0.022]
UK 2007 Dataset								
16-10-4	C		0.644 [0.013]	0.672 [0.013]	0.257 [0.040]	0.253 [0.024]	0.884 [0.034]	0.882 [0.032]
16-8-4	RL		0.623 [0.006]	0.590 [0.007]	0.182 [0.008]	0.129 [0.009]	0.864 [0.014]	0.833 [0.018]
20-10-4	C+RL		0.676 [0.024]	0.689 [0.021]	0.291 [0.067]	0.256 [0.034]	0.901 [0.034]	0.894 [0.033]
Mutag: Whole dataset								
10-6-4	AB		0.497 [0.044]	0.440 [0.114]	0.751 [0.009]	0.738 [0.050]	0.612 [0.018]	0.593 [0.067]
6-8-2	AB+C		0.829 [0.017]	0.796 [0.076]	0.835 [0.008]	0.810 [0.079]	0.799 [0.012]	0.770 [0.089]
6-8-2	AB+C+PS		0.872 [0.007]	0.843 [0.068]	0.874 [0.008]	0.859 [0.060]	0.847 [0.009]	0.835 [0.060]
Mutag: Regression friendly part								
8-4-6	AB		0.508 [0.044]	0.462 [0.111]	0.736 [0.205]	0.730 [0.189]	0.650 [0.104]	0.631 [0.146]
6-8-2	AB+C		0.870 [0.138]	0.804 [0.185]	0.890 [0.021]	0.865 [0.103]	0.848 [0.039]	0.820 [0.127]
10-8-2	AB+C+PS		0.926 [0.012]	0.852 [0.100]	0.929 [0.007]	0.906 [0.097]	0.905 [0.010]	0.884 [0.111]
Mutag: Regression unfriendly part								
6-8-2	AB		0.520 [0.125]	0.462 [0.201]	0.522 [0.043]	0.402 [0.211]	0.481 [0.167]	0.422 [0.259]
10-6-4	AB+C		0.676 [0.022]	0.517 [0.242]	0.712 [0.035]	0.545 [0.353]	0.854 [0.023]	0.803 [0.214]
8-4-2	AB+C+PS		0.676 [0.023]	0.517 [0.242]	0.700 [0.030]	0.579 [0.363]	0.849 [0.031]	0.842 [0.164]

might in practice eliminate the adverse effects of the long term dependency problem.

As can be seen, the cascade staged MLP, on average, produces better results than both the two hidden layer MLP and the Lecun5 MLP. In particular, the improvements are more apparent for the larger Web spam learning problems. The results obtained here may suggest that the Lecun5 MLP performance was degraded, possibly being attributed to the long term dependency problem.

5.3.5 Multi-staged deep learning architectures

The later sections will show that a graph neural network model likewise embraces in its learning mechanism the long term dependency problem. Such problems can be effectively solved via a multiple staged learning approach. The proposed approach can either be a cascade of several one-hidden-layer MLPs, or multiple stages of unsupervised neural networks and MLPs. The latter is inspired by the deep learning concept. In this section, a baseline deep learning architecture will be presented, which trains two or more neural networks, however the difference is that the integrated model includes both unsupervised and supervised network modules. An unsupervised network is first trained and when done, its outputs form

Table 5.5: Layered MLP performances on the UK2006, UK2007 and Mutag datasets

Layered MLP			AUC		F1		ACC	
Hid. units	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2007 Dataset								
25	C		0.815 [0.003]	0.798 [0.004]	0.512 [0.005]	0.652 [0.034]	0.893 [0.008]	0.636 [0.024]
37	RL		0.932 [0.001]	0.835 [0.005]	0.658 [0.007]	0.775 [0.026]	0.910 [0.005]	0.729 [0.023]
37	C+RL		0.945 [0.003]	0.870 [0.007]	0.708 [0.009]	0.764 [0.015]	0.929 [0.003]	0.729 [0.014]
UK 2007 Dataset								
25	C		0.691 [0.054]	0.708 [0.044]	0.394 [0.010]	0.322 [0.012]	0.945 [0.003]	0.936 [0.004]
25	RL		0.655 [0.004]	0.624 [0.003]	0.200 [0.004]	0.129 [0.018]	0.882 [0.016]	0.850 [0.017]
40	C+RL		0.701 [0.038]	0.715 [0.033]	0.457 [0.022]	0.314 [0.031]	0.952 [0.006]	0.932 [0.011]
Mutag: Whole dataset								
4	AB		0.493 [0.023]	0.460 [0.119]	0.752 [0.008]	0.741 [0.054]	0.611 [0.020]	0.596 [0.071]
16	AB+C		0.843 [0.011]	0.803 [0.063]	0.835 [0.010]	0.819 [0.066]	0.798 [0.014]	0.778 [0.079]
13	AB+C+PS		0.882 [0.008]	0.840 [0.069]	0.871 [0.007]	0.851 [0.045]	0.844 [0.007]	0.825 [0.043]
Mutag: Regression friendly part								
10	AB		0.499 [0.025]	0.489 [0.120]	0.812 [0.015]	0.800 [0.075]	0.700 [0.031]	0.685 [0.108]
13	AB+C		0.904 [0.015]	0.838 [0.107]	0.897 [0.009]	0.870 [0.098]	0.862 [0.011]	0.831 [0.121]
16	AB+C+PS		0.942 [0.008]	0.849 [0.093]	0.932 [0.008]	0.904 [0.082]	0.912 [0.011]	0.877 [0.104]
Mutag: Regression unfriendly part								
4	AB		0.624 [0.171]	0.486 [0.229]	0.595 [0.056]	0.363 [0.268]	0.713 [0.182]	0.650 [0.260]
10	AB+C		0.816 [0.057]	0.517 [0.242]	0.775 [0.043]	0.512 [0.337]	0.872 [0.024]	0.753 [0.254]
7	AB+C+PS		0.812 [0.051]	0.517 [0.242]	0.735 [0.030]	0.562 [0.363]	0.841 [0.036]	0.840 [0.182]

Table 5.6: SOM + MLP deep learning performances on all datasets

SOM+MLP			AUC		F1		ACC	
Datasets	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam Datasets								
UK 2006	C+RL		0.974 [0.006]	0.930 [0.018]	0.888 [0.038]	0.694 [0.015]	0.972 [0.011]	0.680 [0.012]
UK 2007	C+RL		0.888 [0.059]	0.831 [0.042]	0.521 [0.050]	0.327 [0.036]	0.942 [0.013]	0.911 [0.017]
Metagenetic Datasets								
Whole Dataset	AB+C+PS		0.898 [0.007]	0.845 [0.057]	0.888 [0.007]	0.854 [0.063]	0.865 [0.009]	0.823 [0.071]
Friendly part	AB+C+PS		0.944 [0.009]	0.861 [0.080]	0.938 [0.008]	0.907 [0.084]	0.918 [0.010]	0.879 [0.105]
Unfriendly part	AB+C+PS		0.754 [0.042]	0.517 [0.242]	0.704 [0.026]	0.561 [0.372]	0.854 [0.023]	0.845 [0.169]

an additional input to the supervised model. The approach has been proven very effective for difficult problems which might be related to the long term dependency problem [21, 90].

In experiments, we applied the proposed method by first training a SOM then using the mapping coordinates of the SOM as an additional input to training the MLPs. Empirically, we took the best results of SOMs obtained in Section 5.3.1, and then re-trained MLPs with the new set of input data. For short, we denote the model as SOM+MLPs. Table 5.6 presents the results of the SOM+MLPs model.

It can be seen that the deep learning inspired model significantly improved the classification and performance generalization. The increase in AUC indicator ranges between 7% and 15% when compared with the best results obtained so far on the UK2006 and UK2007 datasets, respectively. This impressive improvement is interestingly not reflected by the

other F1 and ACC performance. Nevertheless, for the Web spam problems, the AUC is the most important indicator to evaluate the highly imbalanced class distribution of the datasets.

The efficiency of the deep learning regime has been meticulously studied in [90]. The effectiveness of the deep learning architecture is confirmed here through this set of experiments, especially for the challenging Web spam detection problems. In fact, the deep learning system consists of a large number of learning parameters. This may require much by way of computational resources and time, however this also allows an effective approximation of complex problems. Another advantage of the deep learning regime is that the over-fitting problem could be circumvented by training the various separate learning modules.

5.4 Graph based Neural network architectures

We have so far ignored topological relationships between samples in the datasets. Models capable of encoding such dependencies are examined in this section. We will deploy graph based methods to model the graph data structures of the given learning problems. Accordingly, the "Host graph" and the atom bond relation are utilized as contextual information of the data for the Web spam problems and Mutag dataset, respectively.

5.4.1 The application of PMGraphSOM

The PMGraphSOM will be trained by using the same parameters as were used for the SOM. The PMGraphSOM requires the setting of the weight parameter μ that controls the bias between features and topological information. Its weight μ is selected from within the set $\{0.001, 0.01, 0.1, 0.3, 0.6, 0.9\}$. In addition, the map size of the PMGraphSOM is selected to either be 101×70, 80×66, or 73×55 for the Web spam, and either be 59×44, 40×30, or 35×25 for the Mutag dataset. The size of the PMGraphSOM is chosen slightly larger when compared to the previously trained SOM because the input dimension (and amount of

Table 5.7: PMGraphSOM performances on the UK2006, UK2007 and Mutag dataset.

PMGraphSOM			**F1**		**ACC**	
Map	**Fts.**	**T**	**Train**	**Test**	**Train**	**Test**
UK 2006 dataset						
80x66		√	0.621 [0.008]	0.519 [0.011]	0.937 [0.002]	0.506 [0.009]
80x66	C	√	0.723 [0.008]	0.622 [0.004]	0.951 [0.001]	0.557 [0.004]
80x66	RL	√	0.611 [0.004]	0.671 [0.120]	0.942 [0.007]	0.582 [0.057]
80x66	C+RL	√	0.849 [0.019]	0.757 [0.027]	0.973 [0.004]	0.648 [0.024]
UK 2006 dataset						
73x55		√	0.601 [0.028]	0.096 [0.013]	0.962 [0.001]	0.656 [0.027]
73x55	C	√	0.524 [0.075]	0.100 [0.019]	0.958 [0.004]	0.711 [0.053]
73x55	RL	√	0.558 [0.014]	0.099 [0.008]	0.959 [0.001]	0.684 [0.006]
73x55	C+RL	√	0.551 [0.038]	0.102 [0.006]	0.960 [0.003]	0.712 [0.031]
Mutag: Whole dataset						
59x40		√	0.909 [0.055]	0.800 [0.066]	0.877 [0.096]	0.740 [0.087]
59x40	AB	√	0.918 [0.004]	0.823 [0.084]	0.901 [0.005]	0.796 [0.092]
59x40	AB+C	√	0.922 [0.005]	0.790 [0.101]	0.907 [0.006]	0.748 [0.106]
59x40	AB+C+PS	√	0.925 [0.006]	0.800 [0.104]	0.908 [0.007]	0.757 [0.113]
Mutag: Regression friendly part						
40x30		√	0.927 [0.061]	0.864 [0.056]	0.894 [0.101]	0.805 [0.070]
40x30	AB	√	0.959 [0.006]	0.880 [0.089]	0.943 [0.009]	0.835 [0.120]
40x30	AB+C	√	0.917 [0.007]	0.841 [0.103]	0.888 [0.009]	0.803 [0.100]
40x30	AB+C+PS	√	0.955 [0.006]	0.876 [0.070]	0.939 [0.008]	0.835 [0.086]
Mutag: Regression unfriendly part						
35x25		√	0.842 [0.043]	0.322 [0.364]	0.876 [0.097]	0.685 [0.199]
35x25	AB	√	0.929 [0.028]	0.443 [0.402]	0.955 [0.018]	0.740 [0.241]
35x25	AB+C	√	0.928 [0.021]	0.423 [0.409]	0.955 [0.012]	0.645 [0.303]
35x25	AB+C+PS	√	0.919 [0.021]	0.450 [0.416]	0.952 [0.011]	0.790 [0.170]

information provided) to the PMGraphSOM is increased as a result of adding topological information. In other words, given the same map size, the compression ratio associated to PMGraphSOM is larger than that associated with the SOM.

The PMGraphSOM results are summarized in Table 5.7. Similar to the SOM, it is observed that the test results are improved with the number of features used. With respect to the UK2006 and UK2007 datasets, it is observed that the combination of the C+RL features produces better results than when using either C or RL, or when using no features other than the graph topology. Similarly, the feature set AB+C+PS produces better results than any other combination of features. An interesting observation is that the PMGraphSOM can produce a reasonably good accuracy even if no features other than the topology is used for training. This is a somewhat expected result given that link based spam can be captured by

the graphs' topology. A similar observation is made for the mutag datasets. This implies that the structure of the molecule contributes to its mutagenecity.

Moreover, the incorporation of relational information usually leads to a generalisation enhancement of the network when compared to the standard SOM shown in Table 5.1. The testing ACC for the UK2006 and UK2007 datasets is enhanced by about 8.6% and 2.1%, respectively. Similarly, an approximate improvement of 2% in ACC can be observed for the Mutag problems. At first sight there appear to be rather modest improvements in results, and hence this raises the question as to whether the improvement in results is worth the effort. The realization that even a minor improvement, of say 1%, is very significant when considering the scale of the Web dataset, and it is a significant improvement if the residual error approaches zero.

5.4.2 Graph neural network

The GNN models were trained using different numbers of hidden units selected from within $\{14, 25, 31, 37, 40\}$, and the number of state neurons selected from within $\{2, 5, 8, 10, 12\}$ for the Web spam data. In the case of Mutagenesis dataset, the numbers of hidden and state neurons are chosen from within $\{2, 5, 8, 10, 12\}$. All other experimental settings for GNNs are the same as for the MLPs in the previous section.

The experimental results are presented in Table 5.8. It is observed that the GNN generalization ability is significantly better than that of the MLPs (refer to Table 5.2). This is an interesting observation because the improvement in the training performance is much less significant. Moreover, a simple comparison between the two graph based models, GNN and PMGraphSOM, indicates an obvious improvement in excess of 10% in ACC testing performance. The variances of MLP and PMGraphSOM results are generally larger when compared with the GNN results, which implies the two former models' performance are much more sensitive to the initialization conditions.

Table 5.8: GNN performances on the UK2006, UK2007, and the Mutagenesis datasets

GNN				AUC		F1		ACC	
Hidden	**State**	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 Dataset									
14	9		√	0.743 [0.076]	0.589 [0.097]	0.363 [0.100]	0.749 [0.079]	0.621 [0.176]	0.659 [0.044]
37	21	C	√	0.863 [0.013]	0.855 [0.011]	0.624 [0.006]	0.774 [0.006]	0.914 [0.002]	0.739 [0.006]
31	15	RL	√	0.912 [0.001]	0.788 [0.002]	0.601 [0.005]	0.790 [0.005]	0.883 [0.003]	0.730 [0.004]
40	21	C+RL	√	0.944 [0.004]	0.902 [0.009]	0.714 [0.012]	0.815 [0.009]	0.934 [0.002]	0.781 [0.009]
UK 2007 Dataset									
25	15		√	0.528 [0.006]	0.554 [0.006]	0.112 [0.001]	0.122 [0.003]	0.369 [0.034]	0.370 [0.036]
25	21	C	√	0.772 [0.020]	0.755 [0.027]	0.456 [0.010]	0.329 [0.011]	0.955 [0.002]	0.938 [0.003]
14	9	RL	√	0.678 [0.015]	0.628 [0.018]	0.269 [0.017]	0.153 [0.037]	0.919 [0.018]	0.898 [0.017]
37	15	C+RL	√	0.788 [0.017]	0.781 [0.010]	0.451 [0.011]	0.317 [0.008]	0.953 [0.004]	0.937 [0.004]
Mutag: Whole dataset									
5	2		√	0.300 [0.245]	0.300 [0.245]	0.756 [0.022]	0.752 [0.072]	0.600 [0.100]	0.600 [0.074]
5	8	AB	√	0.550 [0.023]	0.573 [0.109]	0.767 [0.012]	0.736 [0.035]	0.600 [0.011]	0.621 [0.077]
5	8	AB+C	√	0.841 [0.019]	0.823 [0.093]	0.844 [0.012]	0.821 [0.072]	0.798 [0.017]	0.830 [0.078]
10	8	AB+C+PS	√	0.837 [0.047]	0.884 [0.020]	0.868 [0.009]	0.852 [0.074]	0.813 [0.048]	0.852 [0.023]
Mutag: Regression friendly part									
2	5		√	0.400 [0.200]	0.400 [0.200]	0.799 [0.005]	0.801 [0.047]	0.665 [0.014]	0.658 [0.134]
10	5	AB	√	0.523 [0.048]	0.539 [0.163]	0.815 [0.009]	0.797 [0.060]	0.665 [0.011]	0.676 [0.086]
5	2	AB+C	√	0.907 [0.014]	0.869 [0.056]	0.906 [0.007]	0.845 [0.075]	0.851 [0.012]	0.904 [0.071]
10	2	AB+C+PS	√	0.867 [0.063]	0.891 [0.019]	0.935 [0.005]	0.911 [0.054]	0.894 [0.074]	0.936 [0.026]
Mutag: Regression unfriendly part									
8	2		√	0.321 [0.240]	0.403 [0.183]	0.473 [0.013]	0.817 [0.010]	0.310 [0.011]	0.309 [0.462]
12	5	AB	√	0.670 [0.043]	0.474 [0.080]	0.588 [0.030]	0.800 [0.005]	0.690 [0.011]	0.690 [0.462]
5	2	AB+C	√	0.696 [0.017]	0.675 [0.066]	0.924 [0.060]	0.918 [0.033]	0.856 [0.012]	0.809 [0.392]
10	2	AB+C+PS	√	0.706 [0.031]	0.678 [0.020]	0.867 [0.019]	0.857 [0.043]	0.851 [0.021]	0.833 [0.373]

When looking at the AUC results related to Mutag datasets, the generalization performance of the GNN on the regression unfriendly part is poorer than the training performance. This is however not observed on the regression friendly part or when using the whole Mutag datasets. This is possibly because of the non-linear properties of regression unfriendly data samples, which is known as the hard-to-solve regression task [91, 92].

5.4.3 Multi-stage Graph neural networks (MSGNN)

This section considers a cascaded system of GNNs as explained in Section 3.4.3.3. A number of GNNs are trained where the output of one GNN is taken as an additional input (concatenated to the other features) for the training of a second GNN; the output of which is added to to train a third GNN, and so on, up to a pre-defined depth. It was quantitatively proven that such a hierarchical model consisting of two GNNs is the most effective architecture [17, 93]. Hence, this Section will present the results of a trained cascade that consists of two GNNs.

Table 5.9: MSGNN performances on the UK2006, UK2007 and Mutagenesis datasets.

MSGNN				AUC		F1		ACC	
Hidden	**State**	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 Dataset									
14	9		√	0.840 [0.006]	0.733 [0.001]	0.511 [0.001]	0.608 [0.003]	0.895 [0.001]	0.594 [0.002]
37	21	C	√	0.890 [0.030]	0.869 [0.004]	0.633 [0.020]	0.774 [0.020]	0.915 [0.005]	0.737 [0.020]
31	15	RL	√	0.915 [0.008]	0.791 [0.019]	0.608 [0.031]	0.770 [0.041]	0.900 [0.014]	0.713 [0.037]
40	21	C+RL	√	0.949 [0.008]	0.914 [0.020]	0.727 [0.015]	0.839 [0.028]	0.935 [0.002]	0.804 [0.030]
UK 2007 Dataset									
25	15		√	0.530 [0.007]	0.561 [0.005]	0.114 [0.003]	0.112 [0.002]	0.421 [0.097]	0.425 [0.091]
25	21	C	√	0.786 [0.018]	0.771 [0.015]	0.396 [0.010]	0.277 [0.016]	0.921 [0.010]	0.918 [0.014]
14	9	RL	√	0.688 [0.008]	0.649 [0.013]	0.221 [0.013]	0.186 [0.014]	0.856 [0.034]	0.851 [0.037]
37	15	C+RL	√	0.809 [0.015]	0.796 [0.016]	0.447 [0.011]	0.332 [0.014]	0.950 [0.015]	0.933 [0.014]
Mutag: Whole dataset									
5	2		√	0.400 [0.200]	0.350 [0.229]	0.750 [0.008]	0.745 [0.078]	0.600 [0.011]	0.600 [0.072]
5	8	AB	√	0.535 [0.038]	0.607 [0.119]	0.772 [0.010]	0.756 [0.047]	0.600 [0.012]	0.626 [0.085]
5	8	AB+C	√	0.843 [0.016]	0.835 [0.085]	0.844 [0.012]	0.826 [0.070]	0.795 [0.012]	0.844 [0.075]
10	8	AB+C+PS	√	0.844 [0.054]	0.884 [0.017]	0.872 [0.009]	0.852 [0.074]	0.826 [0.055]	0.874 [0.028]
Mutag: Regression friendly part									
2	5		√	0.450 [0.150]	0.450 [0.150]	0.799 [0.005]	0.801 [0.047]	0.665 [0.070]	0.672 [0.117]
10	5	AB	√	0.516 [0.054]	0.570 [0.098]	0.812 [0.008]	0.797 [0.051]	0.665 [0.011]	0.693 [0.091]
5	2	AB+C	√	0.910 [0.014]	0.873 [0.053]	0.908 [0.005]	0.859 [0.068]	0.850 [0.012]	0.899 [0.066]
10	2	AB+C+PS	√	0.869 [0.042]	0.897 [0.018]	0.938 [0.005]	0.897 [0.043]	0.872 [0.080]	0.957 [0.030]
Mutag: Regression unfriendly part									
8	2		√	0.298 [0.245]	0.486 [0.117]	0.473 [0.013]	0.817 [0.010]	0.310 [0.011]	0.310 [0.460]
12	5	AB	√	0.682 [0.034]	0.529 [0.178]	0.596 [0.023]	0.817 [0.010]	0.690 [0.011]	0.690 [0.462]
5	2	AB+C	√	0.861 [0.041]	0.732 [0.095]	0.935 [0.060]	0.881 [0.056]	0.851 [0.020]	0.833 [0.373]
10	2	AB+C+PS	√	0.708 [0.030]	0.678 [0.071]	0.862 [0.020]	0.857 [0.043]	0.858 [0.019]	0.857 [0.350]

Table 5.9 summarizes the experimental results of the MSGNN model. In general, the MSGNN outperforms the GNN in the key evaluation method, AUC, by about 1% to 2% for Web spam problems. Similarly, the MSGNN improves the results on the key evaluation indicator for Mutag problems, ACC, by 1% to 3% when compared with the single GNN model's results. The results do not express a consistent improvement with regard to the less important evaluation metrics, i.e. the F1 and ACC for Web spam and AUC and F1 for Mutag problems. This may be observable in some cases, since the model is pushed to perform best on a single evaluation method, rendering it less focused on the others.

From an architectural perspective, the multiple stages of GNNs are similar to the case of having cascaded layers of MLPs instead of having a single GNN or MLP. The multiple stage models are hence effective in the sense that they help to relax the long term dependency problem, which may occur when configuring the model internally with multiple hidden or state layers.

Table 5.10: PMGraphSOM+MLPs performances on all datasets

PMGraphSOM+MLP			AUC		F1		ACC	
Datasets	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam Datasets								
UK 2006	C+RL	$\surd$	0.975 [0.008]	0.931 [0.019]	0.882 [0.057]	0.700 [0.025]	0.970 [0.016]	0.685 [0.020]
UK 2007	C+RL	$\surd$	0.893 [0.042]	0.835 [0.024]	0.540 [0.047]	0.336 [0.024]	0.944 [0.014]	0.915 [0.011]
Metagenetic Datasets								
Whole Dataset	AB+C+PS	$\surd$	0.897 [0.007]	0.845 [0.049]	0.893 [0.008]	0.873 [0.044]	0.872 [0.010]	0.848 [0.049]
Friendly part	AB+C+PS	$\surd$	0.939 [0.015]	0.855 [0.095]	0.936 [0.008]	0.915 [0.085]	0.916 [0.011]	0.890 [0.107]
Unfriendly part	AB+C+PS	$\surd$	0.691 [0.024]	0.517 [0.241]	0.701 [0.027]	0.561 [0.361]	0.857 [0.017]	0.840 [0.181]

5.4.4 Graph Self organising map with multilayered feedforward neural network

The MSGNNs would not normally be referred to as a deep learning model, since unsupervised components are not engaged in the learning architecture. A deep learning inspired architecture engaging the PMGraphSOM as an unsupervised component is considered in this section. The PMGraphSOM is deployed as a pre-training stage, followed by the training of a MLP. The PMGraphSOM is used instead of the SOM since it is capable of modelling structural data and since it preforms a projection of graphs onto a fixed dimensional display space. Hence, the PMGraphSOM reduces a set of graphs to a set of vectors which can then be used to train a MLP. For simplicity, we will refer to this architecture as PMGraphSOM+MLP. This system will be trained by using the same parameters as when training the PMGraphSOM and MLP in the previous sections.

Table 5.10 displays the PMGraphSOM+MLP's experimental results. The ACC performance of the PMGraphSOM+MLP is poorer when compared with the results of the GNN and MSGNN for Mutag problems, however its AUC indicator outperforms the two models for the case of Web spam datasets. Due to the nature of PMGraphSOM learning algorithm, the projection map output does not allow as good generalization classification results as the MLP model. The PMGraphSOM at least provides a helpful grouping of similar input samples, either in feature-based or structural space. The integration between PMGraphSOM and MLP would bring more or less relational aspects of input data to the classification output of MLP. However, it depends on the information richness of topological data of learning prob-

lems. In our cases, the experimental results reflect that the host graphs available in Web spam problems may be more informative than the bond structures provided in the Mutag dataset. In addition, the PMGraphSOM+MLP model expresses similar behavior to the SOM+MLP model, except that the former incorporates relational data in its learning process. Hence, some improvement is evidently observed in the results of the PMGraphSOM+MLP.

5.4.5 Graph Self organising map with multilayered feedforward neural network for GNN filtering

Even though the PMGraphSOM+MLP model expresses such impressive results, there is a good reason to add the GNN model to the end of the PMGraphSOM+MLP learning system. The GNN might incorporate the topological information associated in the input data space in its learning process, i.e the supervised graph based learning. Hence, one of the most promising approaches is to integrate modules such that the relational data could be most productively exploited. We eventually come up with the complex model that is integrated from three different neural networks, namely PMGraphSOM, MLP and GNN, which is denoted as the PMGraphSOM+MLP+GNN model. In this model, the output of PMGraphSOM+MLP will form the additional input to the MSGNN, since the MSGNN can learn on the output of other neural networks. In addition, the placement of MSGNN could not only help to exploit topological relation effectively, but circumvent the long term dependency problem as well. The MSGNN here serves as a classification filter for the whole learning task. The learning capability of PMGraphSOM+MLP+GNN can be relatively derived through the learning capabilities of two models, i.e PMGraphSOM+MLP and MSGNN.

Table 5.11 presents the best results for our learning problems. As can be observed, the PMGraphSOM+MLP+GNN architecture produces outstanding performance on all the studied datasets. The AUC and ACC performance gained an increase of at least 1% to 3% for all Web spam and Mutag problems when compared with the best results attained by any

Table 5.11: PMGraphSOM+MLP+GNN performances on all datasets

PMGraphSOM+MLP+GNN			AUC		F1		ACC	
Datasets	**Features**	**Topo**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam Datasets								
UK 2006	C+RL	√	0.972 [0.010]	0.958 [0.014]	0.820 [0.011]	0.894 [0.013]	0.955 [0.005]	0.865 [0.016]
UK 2007	C+RL	√	0.909 [0.016]	0.854 [0.034]	0.614 [0.022]	0.355 [0.010]	0.964 [0.022]	0.938 [0.011]
Metagenetic Datasets								
Whole Dataset	AB+C+PS	√	0.854 [0.090]	0.892 [0.039]	0.872 [0.009]	0.851 [0.073]	0.843 [0.078]	0.887 [0.058]
Friendly part	AB+C+PS	√	0.882 [0.058]	0.901 [0.019]	0.936 [0.006]	0.920 [0.047]	0.894 [0.071]	0.963 [0.034]
Unfriendly part	AB+C+PS	√	0.708 [0.078]	0.678 [0.071]	0.858 [0.020]	0.857 [0.043]	0.858 [0.019]	0.857 [0.350]

Table 5.12: Different learning models divided into two categories and a complex model

N	Classic Neural Nets	Graph-based Neural Nets
1	SOM	PMGraphSOM
2	MLP	GNN
3	MSMLP	MSGNN
4	SOM+MLP	PMGraphSOM+MLP
	PMGraphSOM+MLP+GNNs	

method tried so far. The following section will take the results presented here to compare with other existing methods applied to the same benchmark problems.

5.5 Comparison and Discussion

In this Section, we will show a relative comparison between classic and graph based neural architectures. Each non-graph learning model is paired and compared with a relatively defined counterpart appearing in graph based learning group. Table 5.12 presents different learning models divided into pairs (indexed as **N**). The pairs' empirical results are compared in Table 5.13. This section will also compare the PMGraphSOM+MLP+GNN performance with those produced by other studies in the literature.

It is noticed that making a comparison between two models is difficult since their architectures and algorithms are not the same. The pairing of models is made so as to correspondingly belong to two sets of learning systems, i.e non-graph and graph based models. The purpose of this comparison is to expose the strength and weakness of each model. It will be clarified by how much an improvement is made when incorporating the topological information in the learning models.

Table 5.13: Generalization performance comparison of Neural network models with and without topology incorporated. Abbreviations used: S-supervised, U-unsupervised, D-deep learning. AUC is not available for unsupervised models.

Learning Models				**UK 2006**		**UK 2007**		**Whole Mutag**		**Friendly part**		**Unfriendly part**	
No	S	U	D	Classic	Graph	Classic	Graph	Classic	Graph	Classic	Graph	Classic	Graph
AUC performance													
2	√			0.865	**0.902**	0.673	**0.781**	0.840	**0.884**	0.851	**0.891**	0.517	**0.678**
3	√		√	0.870	**0.914**	0.715	**0.796**	0.840	**0.884**	0.849	**0.897**	0.517	**0.678**
4	√	√	√	0.930	**0.931**	0.831	**0.835**	0.845	0.845	**0.861**	0.855	0.517	0.517
F1 performance													
1		√		0.634	**0.757**	0.088	**0.102**	0.769	**0.800**	**0.879**	0.876	0.440	**0.450**
2	√			0.791	**0.815**	**0.324**	0.317	0.843	**0.852**	**0.912**	0.911	0.529	**0.857**
3	√		√	0.764	**0.839**	0.314	**0.332**	0.851	**0.852**	**0.904**	0.897	0.562	**0.857**
4	√	√	√	0.694	**0.700**	0.327	**0.336**	0.854	**0.873**	0.907	**0.915**	0.561	0.561
ACC performance													
1		√		0.572	**0.648**	**0.750**	0.712	0.726	**0.757**	**0.844**	0.835	0.740	**0.790**
2	√			0.752	**0.781**	0.930	**0.937**	0.825	**0.852**	0.885	**0.936**	0.815	**0.833**
3	√		√	0.729	**0.804**	0.932	**0.933**	0.825	**0.874**	0.877	**0.957**	0.840	**0.857**
4	√	√	√	0.680	**0.685**	0.911	**0.915**	0.823	**0.848**	0.879	**0.890**	**0.845**	0.840

For ease of comparison, Table 5.13 brings together the results of the various learning models. Bold face is used to highlight the better of the two results. It is observed in Table 5.13 that the graph neural models' performance dominates vector based learning models. Statistically, the graph based models could bring 100% possibility of attaining better testing AUC (up to 9% improvement) for the Web spam problems. Since AUC is the primary evaluation method used for Web spam problems, the models were pushed in order to achieve as good AUC results as possible. This may in some cases result in a degradation of F1 or ACC performance, i.e for the models GNN and PMGraphSOM when learning the challenging UK2007 dataset. For the whole mutag dataset, graph based models are always better when compared with the classical ones. It is noticed that the ACC is the key evaluation metric for Mutag problems. There are some results associated with the regression friendly and unfriendly parts, by which not 100% out-performance of ACC or other evaluation methods is achieved by the graph learning models. The reason is that rather than attempt to attain higher ACC resulting in a decline in F1 and AUC performance, the small number of input samples available in each Mutag data part may hinder the learning mechanism of graph based models (which have to learn with larger input dimension, including both feature and relational information, and with the same small number of input samples when compared with the classic

Table 5.14: A performance comparison of different techniques on Web spam detection problems.

Learning approaches	**Features**	**Cite**	**AUC**
UK2006 Dataset			
Graph Regularization	C+RL+HG+other	[67, 94]	0.963
PMGraphSOM+MLP+GNNs	C+RL+HG		**0.958**
Stacked Learning	C+other	[95]	0.956
Stacked Learning	C+other	Benczúr[78]	0.931
Layered Learning	C+RL+HG	[17]	0.930
Bayesian	C+RL+TL+other	Filoche[78]	0.929
ERUS with C4.5	C+TL+other	Geng[78]	0.927
Language Model Analysis	C+RL+TL	[96]	0.860
Spam Score Propagation	RL+HG	Abou[78]	0.803
UK2007 Dataset			
PMGraphSOM+MLP+GNNs	C+RL+HG		**0.854**
Linked LDA	other	[80]	0.854
ERUS with C4.5	C+HG+other	Geng[79]	0.848
Random forest	C+RL+TL	Tang[79]	0.824
Graph Regularization	C+RL+HG+other	Abernethy[79]	0.809
LDA	other	Siklosi[79]	0.796
Boosted SVM	C+HG+other	Bauman[79]	0.783
Language Model Analysis	C+RL+TL	[96]	0.750
Decision Tree	C+RL+HG	Skvortsov[79]	0.731

models). In addition, the models indexed 4 (i.e SOM+MLP and PMGraphSOM+MLP) produce poorer results than the MSGNN, even though more properties are turned on, such as unsupervised, supervised and deep learning properties. The rationale is that they do not have a graph-based supervised learning property.

Placing our final results with other existing approaches in the Table 5.14 and Table 5.15, it is difficult to make a fair comparison since the learning feature sets are different for each model. For the UK2006 and UK2007 problems, some authors intentionally created the input features themselves based on the raw content of hosts, i.e ”+other” features shown in Table 5.14. For simplicity, the input features are all ignored. The comparison is focused only on the best obtained results. We strictly obey the rule that the number of testing hosts and the evaluation methods must to be the same for all models. Approaches not presented here are those that do not follow the above specification.

Our results stand at the first places for the UK2007 and regression friendly part, and at

Table 5.15: A performance comparison of different techniques on Mutag problem.

Learning approaches	**Features**	**Cite**	**ACC**
Whole dataset			
Improved Bayesian	AB+C+PS	[97]	93.91
PMGraphSOM+MLP+GNNs	AB+C+PS		**88.69**
1nn(dm)	AB+C	[91]	88.00
RelNN	AB+C+PS	[88]	83.04
TILDE	AB+C	[87]	82.00
RDBC	AB+C	[92]	82.00
Regression-friendly part			
PMGraphSOM+MLP+GNNs	AB+C+PS		**96.26**
RS	AB+C+PS+FG	[98]	95.80
MFLOG	AB+C	[99]	95.70
Bayesian	AB+C+PS	[97]	95.22
RSD	AB+C+FG	[100]	92.60
RelNN	AB+C+PS	[88]	91.49
1nn(dm)	AB+C	[91]	91.00
Regression-unfriendly part			
PMGraphSOM+MLP+GNNs	AB+C+PS		**85.71**
TILDE	AB+C	[87]	85.00
RDBC	AB+C	[92]	79.00
1nn(dm)	AB+C	[91]	72.00

the second places for the UK2006, regression unfriendly part and the whole Mutag dataset. This is an indicative comparison, since it reflects that the proposed learning model is especially effective in dealing with a wide range of real-world challenging problems.

5.6 Conclusions

This chapter studied the incorporation of unsupervised and supervised neural network architectures in different ways, with the main focus of identifying the architectural conditions, under which the integration model expresses its best effectiveness by taking advantages of each individual learning component. We have found that the learning system with an unsupervised PMGraphSOM model and a supervised MLP architecture, and followed by a number of stages of GNN as filters, is the most robust model that can effectively solve classification and regression applications.

Though the long term dependency problem was not meticulously studied in this chapter,

we have quantitatively proven that layer/hierarchy based learning systems can deal with the problem of long term dependencies much better than many layers integrated internally in a single prediction model.

Chapter 6

Encoding Structural Data with Kernel machines

6.1 Introduction

Kernel methods such as SVM, multi-kernel SVM, GLSVM, kernel K-means and spectral kernel clustering have become popular in machine learning because of their efficiency and simplicity of learning algorithm. Similar to neural algorithms, the more recent research activities in kernel methods focused on introducing graph based kernel learning. Both supervised and unsupervised kernel based algorithms rely on a graph/node similarity measure to compute an adjacency matrix which represents the neighborhood relations between input samples. Then a kernel matrix is defined by, for example, following a graph Laplacian method.

Even though there exist a good number of graph based kernel machine methods, there is no systematic study in assessing the effectiveness of graph-based over the classical non-graph based kernel methods nor has there been a sufficiently deep study on the integration of graph-kernel learning modules as a way of enhancing the capabilities of a learning system. A number of research questions will be addressed in this chapter: (1) Do graph based learning

models have a performance advantage over non-graph based models? An expected answer would be that graph-based kernel methods should be able to at least match the computational capabilities of their non-graph based counterparts. (2) Is it possible to construct hierarchical or deep kernel architectures (similar to those introduced in ANNs in Chapter 5), and would such architectures enhance the general performance? (3) How does the learning capability of graph kernel methods compare to learning capability of graph neural networks? This chapter presents a comprehensive study on the effect of modelling structural topology of data by building kernel machine models. This study is conducted analogous to the study on Neural Networks that was presented in Chapter 5. To facilitate a comparison, the models are trained and applied to the same sets of data viz. the Web spam detection datasets and the Mutag datasets.

The organization of this Chapter is as follows. Section 6.2 presents the graph matrix selection. The general experimental setting is presented in Section 6.3. In Section 6.4, experimental results of non-graph based unsupervised and supervised learning architectures will be given, while in Section 6.5, a graph-based model will be presented. Comparisons and a discussion will be offered in Section 6.6. Conclusions are drawn in Section 6.7.

6.2 Graph matrix selection

Any learning problem consisting of a set of graphs can be transformed into an equivalent single graph learning problem or into an equivalent node-focused learning problem. Details of the transformation process will be presented in Section 6.2.1. The construction of the similarity graph, adjacency matrix and graph Laplacian matrix will be shown in Section 6.2.2 and Section 6.2.3 respectively.

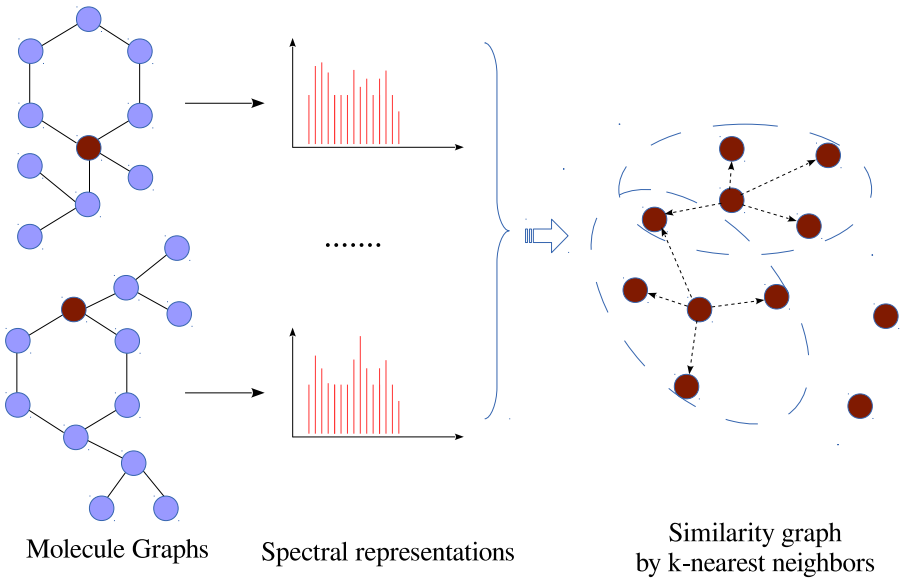

Figure 6.1: Steps to construct a similarity graph on an example: Two molecules (graphs on the left) are transformed into spectral representation (middle) which in turn is used to compose the similarity graph (right) using k-nearest neighbor method.

6.2.1 Spectral transformation

A widely accepted approach to applying the graph Laplacian kernel method is to use spectral transformation. Spectral graph theory provides an approach to addressing the problem of multiple graphs [101]. This approach is oriented by the mathematical foundation that exposes the characteristics of the structural information of graphs using the eigenvectors and eigenvalues of the adjacency matrix. In particular, each graph structure is transformed into a vectorial representation. The following steps are executed: (1) The adjacency matrix A associated with a single graph is computed. (2) The eigenvalues of matrix V are obtained by the eigen decomposition approach $A = U^T V U$. (3) A vector of positive eigenvalues is used as the additional input to each input sample. Each graph is provided with a spectral representation or a spectral vector. This vector is then added to the input features of each learning sample as additional relational information. Figure 6.1 demonstrates the transformation from graphs to spectral feature values. In the next step, a similarity graph on the whole input feature space (including both original feature vector and spectral feature vector) will be composed.

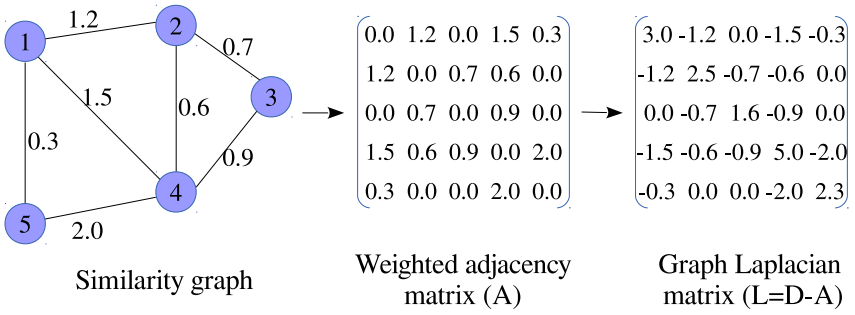

Figure 6.2: Constructing the Graph Laplacian matrix on an example: Given the labeled edge similarity graph (left), the corresponding weighted adjacency matrix is constructed (middle), and finally the weighted graph Laplacian is computed (right)

6.2.2 Similarity graphs

Figure 6.1, on the right, illustrates the process of creating a similarity graph for the whole input space from a multiple graph learning problem. The most popular way to build a similarity graph is the k-nearest neighbor approach. This figure particularly shows the construction of a similarity graph by using the 4-nearest neighbor approach. Nodes in the graph represent input samples. Every node is connected to 4 other nodes which are closest to it in terms of Euclidean distance with respect to the node's feature vectors. In practice, a similarity graph can also be derived from the structural topology of the data input, thus avoiding the need to compute a spectral representation. For instance, in the WWW domain one can use the hyperlink connections to build the similarity graph. The weight (scalars to labels on edges) of each edge can be calculated using the squared distance among the nodes' feature vectors.

6.2.3 Building the Graph Laplacian Matrix

A graph matrix is one that represents the neighborhood relations between nodes of a graph in a condensed numerical form. The Graph Laplacian matrix is one of the most well-known approaches of this type. The resulting matrix is suited as input for the graph-based kernel learning methods.

The Graph Laplacian is constructed as follows. Assuming that A is an adjacency matrix with its elements $a_{i,j}$ equal to 1 if there is a link between node v_i and v_j, and 0 otherwise. The Graph Laplacian matrix will be $L = D - A$, where D is a diagonal degree matrix. $d_{ii} = \sum_{j=1}^{N} a_{i,j}$, and N is the number of nodes in a graph (the dimension of A). If the links in a graph are weighted then the Graph Laplacian can be constructed based on a weighted adjacency matrix, as depicted in Figure 6.2. The figure shows that the elements a_{ij} of the adjacency matrix A would then be assigned to the weight of the edge connecting v_i and v_j. The weighted L can be computed accordingly. Finally, L is normalized by calculating $L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$.

6.3 Experimental procedures

A common approach when adopting GLSVM is that the similarity graph is computed via the sample feature vectors [70]. For the Web spam problem, the similarity graph is derived from the topology of the host graph. Due to the restriction of GLSVM in solving the multiple graph problem, we adopt spectral transformation for Mutag datasets. In composing the similarity graph, a maximum number of nearest neighbors is required. The number of nearest neighbors is tuned within the range [4,10] for regression unfriendly, regression friendly and the whole Mutag datasets. We respectively select the maximum number of nearest neighbors to be 9, 5 and 6 for those three datasets through a number of experiments.

In the unsupervised learning cases, i.e KKM and SKC, the best number of clusters will be shown in the results tables in a column named **NoC** (short for "Number of Clusters"). To compute the training performance, clusters are assigned to class labels (-1 or +1) using majority voting. For example, a cluster belongs to class +1 if the predominant samples in that cluster belong to class +1. This will allow the performance computation for evaluation and testing purposes using ACC and F1. The column **T** in the resulting tables indicates whether or not topological information is modelled by the corresponding learning method.

A validation set is used by randomly selecting 10% of the training samples. The models are trained on the training set, the best parameters then selected on the basis of the validation performance. The AUC and ACC indicators are respectively used as the performance indicators for validation purposes for the Web spam detection and Mutagenesis problems. Each experiment is repeated 10 times (Web spam detection task), and 3 times (Mutagenesis problems). The mean of the training and testing performances and the associated standard deviation will be reported. 10-fold cross validation is applied for the Regression friendly and the whole Mutagenesis dataset, and leave-one-out applied for the Regression unfriendly data. The selected models' parameters are shown in the first columns in the result tables. For any supervised model, three metrics AUC, F1 and ACC are used for experimental evaluation. The significant order of these regarding each learning problem is set the same as was shown in the Chapter 5.

6.4 Applications of classic Kernel Machines

This section will apply KKM, SVM, and an integrated method consisting of both KKM and SVM. These models do not incorporate the relational aspects of input data in their learning process. The results shown here will form the baselines for the comparisons with the graph based kernel methods.

6.4.1 Kernel K-means

KKM is one of the simplest kernel clustering methods. The number of clusters is tuned respectively within the range of [10,20] for the Mutagenesis datasets and within [20,40] for the Web spam detection problems. The number of clusters was derived empirically under the the reasonable assumption that more clusters are expected for the much larger problems UK2006 and UK2007 when compared with the Mutagenesis dataset. The KKM's

Table 6.1: KKM performances on Web spam and Mutagenesis problems

KKM			F1		ACC	
NoC	**Features**	**T**	**Train**	**Test**	**Train**	**Test**
UK 2006 dataset						
37	C		0.333 [0.017]	0.689 [0.048]	0.745 [0.051]	0.636 [0.031]
37	RL		0.332 [0.010]	0.784 [0.019]	0.605 [0.057]	0.690 [0.014]
37	C+RL		0.329 [0.009]	0.762 [0.009]	0.642 [0.029]	0.675 [0.007]
UK 2007 dataset						
25	C		0.164 [0.009]	0.160 [0.008]	0.636 [0.047]	0.622 [0.045]
25	RL		0.138 [0.015]	0.130 [0.005]	0.594 [0.126]	0.570 [0.120]
25	C+RL		0.145 [0.018]	0.136 [0.014]	0.646 [0.118]	0.632 [0.120]
Whole Mutag dataset						
20	AB		0.748 [0.005]	0.747 [0.039]	0.604 [0.006]	0.601 [0.051]
20	AB+C		0.802 [0.012]	0.795 [0.096]	0.764 [0.022]	0.762 [0.093]
20	AB+C+PS		0.822 [0.012]	0.815 [0.097]	0.795 [0.015]	0.791 [0.089]
Regression Friendly part						
16	AB		0.799 [0.006]	0.796 [0.058]	0.671 [0.009]	0.668 [0.076]
16	AB+C		0.857 [0.019]	0.849 [0.105]	0.809 [0.018]	0.808 [0.115]
16	AB+C+PS		0.870 [0.033]	0.861 [0.121]	0.829 [0.038]	0.828 [0.134]
Regression Unfriendly part						
10	AB		0.632 [0.032]	0.464 [0.395]	0.826 [0.024]	0.828 [0.167]
10	AB+C		0.686 [0.066]	0.556 [0.405]	0.798 [0.058]	0.800 [0.211]
10	AB+C+PS		0.681 [0.061]	0.558 [0.408]	0.805 [0.048]	0.807 [0.216]

experiment results are summarized in Table 6.1. It can be seen that the KKM performance varies significantly with the initialization condition. This is most obvious in the case of ACC performance on the UK2007 data. The standard deviation of results for the Mutag datasets is also relatively high. However the reason here is that its calculation is based on the cross validation approach. An interesting observation is that while the ACC performance is quite reasonable for all the datasets, the F1 indicator is very poor for UK2007 when compared with UK2006 and the other datasets. This observation can be attributed to the imbalanced nature of the Web spam problems.

6.4.2 Support Vector learning

Two main parameters that need to be set when training a SVM are the kernel's parameters, and the soft margin parameter γ_A (or C). These parameters are selected via a grid search strategy with exponentially growing sequences of σ and γ_A. More specifically, γ_A is selected from within the range $\{2^{-3}, 2^{-2}, ..., 2^{9}\}$ while σ is selected from within $\{2^{-11}, 2^{-10}, ..., 2^{2}\}$.

Table 6.2: SVM performances on Web spam and Mutagenesis problems

SVM				AUC		F1		ACC	
σ	γ_A	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 dataset									
2^2	2^5	C		0.931 [0.012]	0.764 [0.006]	0.783 [0.079]	0.457 [0.042]	0.967 [0.010]	0.520 [0.022]
2^0	2^5	RL		0.965 [0.012]	0.822 [0.022]	0.647 [0.138]	0.539 [0.063]	0.948 [0.016]	0.566 [0.036]
2^{-2}	2^6	C+RL		0.977 [0.014]	0.882 [0.014]	0.749 [0.109]	0.656 [0.050]	0.961 [0.014]	0.650 [0.034]
UK 2007 dataset									
2^{-2}	2^6	C		0.901 [0.023]	0.734 [0.018]	0.502 [0.123]	0.282 [0.078]	0.963 [0.006]	0.946 [0.002]
2^{-3}	2^7	RL		0.820 [0.035]	0.607 [0.012]	0.156 [0.084]	0.023 [0.008]	0.949 [0.003]	0.938 [0.002]
2^{-2}	2^5	C+RL		0.963 [0.025]	0.726 [0.011]	0.723 [0.179]	0.284 [0.068]	0.977 [0.012]	0.937 [0.005]
Whole Mutag dataset									
2^{-9}	2^4	AB		0.524 [0.004]	0.480 [0.095]	0.750 [0.001]	0.743 [0.013]	0.602 [0.001]	0.591 [0.039]
2^{-5}	2^5	AB+C		0.869 [0.009]	0.845 [0.025]	0.848 [0.012]	0.832 [0.029]	0.813 [0.014]	0.794 [0.030]
2^{-2}	2^2	AB+C+PS		0.896 [0.010]	0.868 [0.010]	0.866 [0.008]	0.843 [0.027]	0.839 [0.009]	0.812 [0.031]
Regression Friendly part									
2^{-11}	2^2	AB		0.496 [0.009]	0.508 [0.038]	0.799 [0.001]	0.792 [0.006]	0.665 [0.001]	0.662 [0.030]
2^{-4}	2^3	AB+C		0.941 [0.004]	0.927 [0.043]	0.896 [0.008]	0.886 [0.036]	0.859 [0.012]	0.846 [0.048]
2^{-5}	2^4	AB+C+PS		0.969 [0.012]	0.948 [0.027]	0.939 [0.019]	0.913 [0.031]	0.921 [0.025]	0.888 [0.036]
Regression Unfriendly part									
2^{-4}	2^5	AB		0.735 [0.032]	0.745 [0.045]	0.631 [0.001]	0.599 [0.041]	0.833 [0.001]	0.772 [0.176]
2^{-5}	2^5	AB+C		0.937 [0.043]	0.839 [0.027]	0.869 [0.145]	0.704 [0.168]	0.933 [0.066]	0.815 [0.167]
2^{-5}	2^4	AB+C+PS		0.967 [0.046]	0.853 [0.044]	0.860 [0.111]	0.703 [0.094]	0.926 [0.054]	0.813 [0.139]

A popular Gaussian radial basis function is applied as the kernel function.

The experimental results are presented in Table 6.2. It can be observed that the classic kernel machine SVM generally outperforms KKM in both classification and generalization. The most obvious improvement is observed for UK2007 on both ACC and F1 indicators. The generalization results are improved by 15% in F1 and 30% in ACC when compared with the KKM results. A similar observation is made with the Mutag datasets. The ACC performance increases by 1% to 6% when compared with the KKM results. Moreover, it is evident that the model's performance is improved when more features are utilized for learning. With respect to the UK2006 data, the AUC performance is generally better than those from the UK2007 dataset, due to the severely imbalanced nature of the UK2007 dataset. Any standard learning models would find it non-trivial to achieve a good ACC result (by simply assigning most samples to the majority class) in this type of dataset, rendering it difficult to obtain a high AUC (which may bias correct classification of minority class samples). It is also confirmed here that the ACC performance regarding the UK2007 dataset is always seen to be best for both training and generalization performance when compared with other datasets.

6.4.3 Multi-kernel Support Vector machine

We used $p = 2$ in the experiments as it was shown to be the most effective p-norm regularization value [102]. The number of kernels can be arbitrary, however a large number of kernels can increase the computational time requirement unnecessarily. We found that 10 kernels is suited to the given learning problems. All other kernel and learning parameters are similar to the SVM model.

The experimental results of the MKSVM are summarized in Table 6.3. It is generally observed that the utilization of linearly combined kernel functions improved the performance on all evaluation indicators. The training performance is close to 100% for all of the learning problems. This result can be attributed to the complementary nature of the kernel functions so that the strength of each kernel is taken into account. Particularly for generalization performance, the UK2006 dataset experienced the greatest improvement in results on all of the evaluation metrics but especially for ACC and F1, which improved by 25% and 20% improvement, respectively. The UK2007 dataset, however only gained about a 3% increase in the AUC generalization performance when compared to using a single kernel function. For the Mutag datasets, the MKSVM generally produces a better ACC performance and is more stable (smaller deviation) than the SVM case.

6.4.4 Deep learning using kernel machines

The work presented in this section is inspired by the recent successes of deep learning in the field of neural networks [21, 103] as well as in the kernel machine domain [38, 104, 105]. The fundamental idea in deep kernel methods is based on the layered architecture consisting of several kernel methods within each of the layers. This architecture often contains just two layer of SVMs [39, 106, 107, 108]. The SVMs in the first layer are trained in parallel, taking a subset of the training data as input by sampling data from the original input space. The outputs of the first layer are combined and then form the input for the second layer of SVMs.

Table 6.3: MKSVM performances on Web spam and Mutagenesis problems

MKSVM				AUC		F1		ACC	
σ	γ_A	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 dataset									
2^{-5}	2^{2}	C		0.913 [0.044]	0.768 [0.008]	0.698 [0.155]	0.623 [0.065]	0.949 [0.039]	0.622 [0.042]
2^{-4}	2^{3}	RL		0.975 [0.010]	0.803 [0.010]	0.830 [0.044]	0.816 [0.012]	0.964 [0.010]	0.748 [0.008]
2^{-3}	2^{3}	C+RL		0.998 [0.002]	0.887 [0.008]	0.950 [0.043]	0.866 [0.004]	0.990 [0.009]	0.808 [0.006]
UK 2007 dataset									
2^{-6}	2^{3}	C		0.970 [0.014]	0.736 [0.006]	0.942 [0.003]	0.354 [0.006]	0.994 [0.000]	0.908 [0.004]
2^{-6}	2^{5}	RL		0.936 [0.014]	0.580 [0.006]	0.854 [0.037]	0.125 [0.030]	0.986 [0.003]	0.846 [0.040]
2^{-4}	2^{3}	C+RL		0.996 [0.003]	0.760 [0.011]	0.990 [0.005]	0.279 [0.015]	0.999 [0.001]	0.832 [0.014]
Whole Mutag dataset									
2^{-2}	2^{7}	AB		0.559 [0.056]	0.547 [0.026]	0.753 [0.015]	0.748 [0.007]	0.614 [0.011]	0.605 [0.011]
2^{0}	2^{2}	AB+C		0.991 [0.005]	0.779 [0.024]	0.980 [0.015]	0.800 [0.010]	0.976 [0.018]	0.761 [0.011]
2^{-3}	2^{2}	AB+C+PS		0.994 [0.003]	0.870 [0.017]	0.987 [0.007]	0.846 [0.006]	0.984 [0.009]	0.820 [0.007]
Regression Friendly part									
2^{-5}	2^{4}	AB		0.475 [0.027]	0.506 [0.037]	0.806 [0.003]	0.800 [0.004]	0.681 [0.007]	0.673 [0.006]
2^{0}	2^{2}	AB+C		0.990 [0.007]	0.807 [0.026]	0.977 [0.020]	0.867 [0.006]	0.970 [0.026]	0.824 [0.009]
2^{-1}	2^{3}	AB+C+PS		0.996 [0.005]	0.923 [0.024]	0.987 [0.007]	0.912 [0.006]	0.983 [0.009]	0.883 [0.008]
Regression Unfriendly part									
2^{-6}	2^{5}	AB		0.755 [0.016]	0.694 [0.078]	0.631 [0.000]	0.611 [0.015]	0.833 [0.000]	0.818 [0.112]
2^{-4}	2^{5}	AB+C		1.000 [0.000]	0.873 [0.026]	0.999 [0.006]	0.694 [0.063]	0.999 [0.004]	0.838 [0.111]
2^{-6}	2^{6}	AB+C+PS		1.000 [0.000]	0.882 [0.021]	1.000 [0.000]	0.671 [0.038]	1.000 [0.000]	0.849 [0.113]

A more strict interpretation of the term deep learning requires that the system comprises of an unsupervised model as a pre-training component. A corresponding deep learning kernel machine was proposed in [38] where an unsupervised Principal component analysis (PCA) kernel and the supervised SVM algorithm are integrated in a hierarchical architecture.

In this chapter, several adaptive deep learning architectures will be presented. The deep model described in this section is structured by integrating KKM and SVM models. For simplicity, we denote the model as KKM+SVM. Figure 6.4.4 illustrates the proposed deep learning model. Firstly, the KKM is trained unsupervised on the original input data. The clustering result is then used as additional input to the 2-layer SVMs. A number of SVMs in the first SVM layer was tried between 20 and 120. We empirically found that 100 SVMs is a reasonable number for the tasks. All SVMs are trained in parallel. The input subset of each SVM component is sampled from the whole training data. The minority class samples are all present in each subset, while the number of samples in the majority class are randomly sampled to ensure that the size of the two classes is equal. This sampling mechanism should help the SVM to learn better. The concatenation of SVM outputs from the first layer will form the input to the second SVM layer which contains only one SVM. As for any ensem-

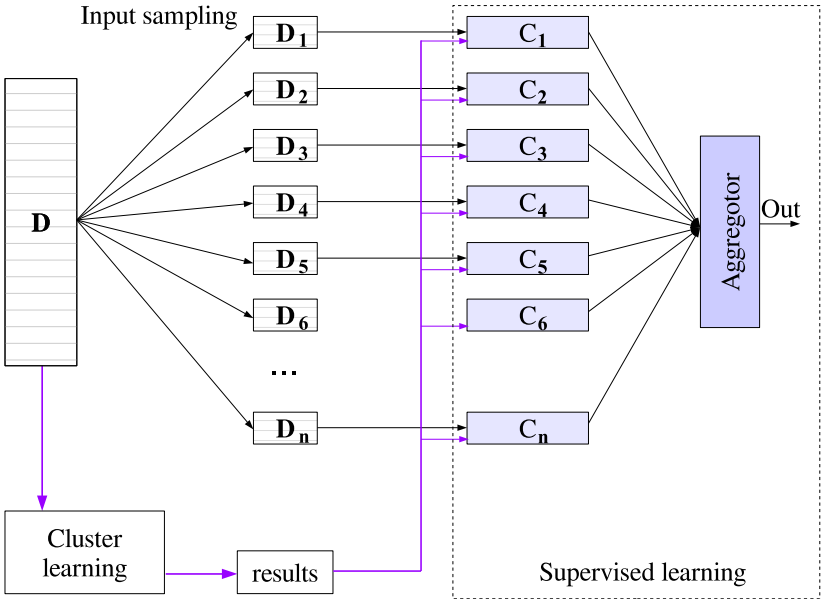

Figure 6.3: Deep learning using KKM clustering and SVM learners. D represents the input data while D_i is a sampled data from D. The parallel layers include classic SVMs (denoted C_i). The Aggregator in this case is another SVM learning the output of previous layer.

Table 6.4: KKM+SVM performance on the Web spam and Mutagenesis learning problems.

KKM+SVM			AUC		F1		ACC	
Datasets	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam datasets								
UK 2006	C+RL		0.962 [0.008]	0.887 [0.009]	0.622 [0.065]	0.593 [0.048]	0.944 [0.007]	0.607 [0.032]
UK 2007	C+RL		0.963 [0.024]	0.771 [0.006]	0.700 [0.173]	0.285 [0.022]	0.976 [0.013]	0.945 [0.005]
Metagenetic datasets								
Whole Dataset	AB+C+PS		0.898 [0.013]	0.863 [0.006]	0.875 [0.009]	0.855 [0.011]	0.850 [0.010]	0.826 [0.013]
Friendly part	AB+C+PS		0.955 [0.017]	0.943 [0.017]	0.926 [0.009]	0.910 [0.006]	0.904 [0.012]	0.884 [0.018]
Unfriendly part	AB+C+PS		0.925 [0.034]	0.821 [0.034]	0.705 [0.034]	0.676 [0.021]	0.853 [0.015]	0.838 [0.110]

ble model the selection of learning parameters is done by using the best parameters of the individual learning components. The experimental results of the KKM+SVM are summarized in Table 6.4. The deep learning model generally produces better results than the single kernel SVM, and is comparable with the MKSVM, i.e the AUC testing performance. The ACC indicator gains 1% to 2% increase compared with that of the SVM model. The advantages of KKM+SVM over the MKSVM are that it is simply implemented and reduces the computational cost of executing multiple kernels internally in the support vector learning algorithm since it allows parallel computations.

Table 6.5: SKC performances on Web spam and Mutagenesis problems

SKC			F1		ACC	
NoC	**Features**	**T**	**Train**	**Test**	**Train**	**Test**
UK 2006 dataset						
39	C	√	0.339 [0.007]	0.717 [0.028]	0.743 [0.021]	0.662 [0.022]
39	RL	√	0.352 [0.009]	0.742 [0.016]	0.695 [0.034]	0.657 [0.014]
39	C+RL	√	0.347 [0.009]	0.764 [0.023]	0.653 [0.029]	0.682 [0.019]
UK 2007 dataset						
25	C	√	0.170 [0.012]	0.165 [0.014]	0.647 [0.063]	0.638 [0.064]
25	RL	√	0.143 [0.007]	0.129 [0.004]	0.596 [0.069]	0.574 [0.061]
25	C+RL	√	0.154 [0.009]	0.149 [0.015]	0.642 [0.055]	0.637 [0.058]
Whole Mutag dataset						
20	AB	√	0.740 [0.038]	0.751 [0.084]	0.674 [0.020]	0.686 [0.097]
20	AB+C	√	0.780 [0.029]	0.805 [0.062]	0.741 [0.030]	0.770 [0.067]
20	AB+C+PS	√	0.833 [0.014]	0.858 [0.048]	0.796 [0.023]	0.807 [0.056]
Regression Friendly part						
17	AB	√	0.791 [0.015]	0.789 [0.071]	0.707 [0.015]	0.707 [0.092]
17	AB+C	√	0.858 [0.027]	0.849 [0.127]	0.804 [0.038]	0.803 [0.145]
17	AB+C+PS	√	0.892 [0.027]	0.885 [0.113]	0.856 [0.037]	0.855 [0.127]
Regression Unfriendly part						
12	AB	√	0.609 [0.047]	0.472 [0.365]	0.769 [0.041]	0.767 [0.178]
12	AB+C	√	0.727 [0.053]	0.598 [0.425]	0.827 [0.029]	0.815 [0.192]
12	AB+C+PS	√	0.720 [0.083]	0.565 [0.423]	0.812 [0.065]	0.811 [0.181]

6.5 Applications of Graph-based Kernel Machines

This section will present several graph based kernel learning models, in which the relational aspects of data input will be encoded.

6.5.1 Spectral kernel clustering

The SKC has many similar properties to KKM. The only difference is that the SKC at first exploits the topological information given by the similarity graph. The KKM then plays a modular role in the SKC model. The KKM is applied on the resulting matrix whose columns are eigenvectors received from eigen decomposition of the graph Laplacian matrix. The SKC's experimental results are given in Table 6.5. Note that column **T** is ticked implying the usage of topological structures. It can be observed that both SKC and KKM performance are poor in terms of the F1 indicator for the UK2007 dataset. This can be explained by the severely imbalanced nature of this dataset. Since the majority of class sam-

ples are largely predominant in the resulting clusters, the ACC performance is accordingly high. Similar observations can be seen for the F1 testing performance for all datasets. Nevertheless, the learning performance of SKC is generally better than that of KKM, since it takes the relational aspects of the data into consideration. All the SKC's training parameters and the learning procedures were the same as those in the KKM case. This clearly shows that spectral kernels benefit from modelling relational data.

6.5.2 Graph Laplacian Support Vector Machine

The GLSVM is capable of incorporating structural topology in its learning process. In addition to the kernel parameter σ and the ambient space parameter γ_A as in the SVM model, the GLSVM's variable γ_I decides the influence of the structural information. A grid search strategy is used to identify the best choice of these three parameters. They are selected from within the following ranges, $\sigma \in \{2^{-5}, 2^{-4}, ..., 2^{1}\}$, $\gamma_A \in \{2^{-3}, 2^{-2}, ..., 2^{3}\}$ and $\gamma_I \in \{2^{-3}, 2^{-2}, ..., 2^{3}\}$.

Results are shown in Table 6.6. It is observed that the GLSVM's generalization ability is much better than that of the non-graph based SVM model given in Table 6.2. In particular, the AUC and ACC indicators are both improved by around 1% - 3% for the Web spam detection and Mutagenesis problems. Interestingly, the training performance of GLSVM remains very similar to that of the SVM and MKSVM models. This reflects that incorporation of the graph data structure in the GLSVM encourages generalization ability of the model. If compared with the unsupervised SKC model, a significant improvement in ACC generalization performance can be observed, especially for UK2006 and UK2007. The effectiveness of modelling graph data in supervised kernel learning is likewise evident when comparing results with the deep learning KKM+SVM model. The GLSVM results are better than the KKM+SVM results. This, too, indicates that the encoding of topological information in a supervised kernel machine algorithm can bring a measurable benefit.

Table 6.6: GLSVM-A performances on Web spam and Mutagenesis problems

GLSVM					AUC		F1		ACC	
σ	γ_A	γ_I	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
UK 2006 Dataset										
2^{-1}	2^{-2}	2^{-3}	C	√	0.845 [0.010]	0.806 [0.006]	0.568 [0.017]	0.714 [0.015]	0.900 [0.007]	0.683 [0.010]
2^{0}	2^{-1}	2^{0}	RL	√	0.961 [0.019]	0.843 [0.005]	0.728 [0.021]	0.775 [0.016]	0.928 [0.007]	0.731 [0.013]
2^{1}	2^{-5}	2^{0}	C+RL	√	0.964 [0.006]	0.895 [0.009]	0.732 [0.022]	0.796 [0.031]	0.934 [0.005]	0.760 [0.029]
UK 2007 Dataset										
2^{-1}	2^{-2}	2^{2}	C	√	0.800 [0.004]	0.756 [0.006]	0.420 [0.002]	0.343 [0.002]	0.942 [0.002]	0.931 [0.006]
2^{1}	2^{1}	2^{-3}	RL	√	0.694 [0.010]	0.631 [0.009]	0.208 [0.002]	0.142 [0.006]	0.870 [0.017]	0.844 [0.020]
2^{2}	2^{1}	2^{1}	C+RL	√	0.766 [0.024]	0.760 [0.007]	0.284 [0.023]	0.241 [0.004]	0.863 [0.036]	0.920 [0.012]
Whole Mutag dataset										
2^{-1}	2^{-4}	2^{-2}	AB	√	0.542 [0.000]	0.364 [0.086]	0.777 [0.001]	0.745 [0.016]	0.674 [0.001]	0.604 [0.041]
2^{1}	2^{-1}	2^{-3}	AB+C	√	0.875 [0.012]	0.807 [0.072]	0.849 [0.004]	0.822 [0.019]	0.809 [0.004]	0.788 [0.026]
2^{1}	2^{-2}	2^{-2}	AB+C+PS	√	0.856 [0.006]	0.815 [0.010]	0.853 [0.004]	0.848 [0.034]	0.824 [0.002]	0.817 [0.017]
Regression Friendly part										
2^{0}	2^{-5}	2^{-3}	AB	√	0.507 [0.000]	0.408 [0.023]	0.827 [0.001]	0.800 [0.002]	0.729 [0.000]	0.670 [0.060]
2^{1}	2^{-1}	2^{-3}	AB+C	√	0.931 [0.001]	0.912 [0.026]	0.900 [0.001]	0.895 [0.018]	0.864 [0.002]	0.854 [0.027]
2^{-1}	2^{-3}	2^{-3}	AB+C+PS	√	0.934 [0.007]	0.919 [0.024]	0.926 [0.004]	0.923 [0.021]	0.902 [0.004]	0.899 [0.024]
Regression Unfriendly part										
2^{1}	2^{-2}	2^{-1}	AB	√	0.781 [0.026]	0.500 [0.030]	0.662 [0.013]	0.593 [0.047]	0.797 [0.014]	0.781 [0.134]
2^{-1}	2^{1}	2^{-2}	AB+C	√	0.966 [0.003]	0.813 [0.032]	0.960 [0.001]	0.704 [0.048]	0.976 [0.000]	0.833 [0.140]
2^{-1}	2^{-3}	2^{-3}	AB+C+PS	√	0.981 [0.002]	0.835 [0.028]	0.984 [0.018]	0.739 [0.052]	0.990 [0.012]	0.860 [0.122]

6.5.3 Learning hyperlink and feature based similarity graphs

In the Web spam problems, either hyperlink-based or feature-based similarity graphs could be utilized in our experiments. Hence, it is worth clarifying which method is more advantageous. Two models will correspondingly learn on the two types of graph inputs. These will be denoted simply as GLSVM-A and GLSVM-B. The two models are identical but the input is different. The GLSVM-A results are shown in Table 6.6. We select the number of nearest neighbors from 3 to 9 in constructing the feature-based similarity graphs for GLSVM-B. We found that the best number of nearest neighbors is 4 for both the UK2006 and UK2007 datasets. If we use the same C+RL feature set for training the GLSVM-A and GLSVM-B models. Comparison results between two models GLSVM-A and GLSVM-B are shown in Table 6.7. The former always outperforms the latter in all cases. In particular, the GLSVM-B performance is about 1% AUC and 2% F1 and ACC for UK2006, 0.5% AUC, 1% F1 and 5% ACC for UK2007, poorer than that of GLSVM-A.

Figure 6.4 and Figure 6.5 compare the disagreed prediction results (displaying samples which might be correctly classified by GLSVM-A, but not GLSVM-B, and vice versus) be-

Table 6.7: GLSVM performances on Web spam detection problems

Compare					AUC		F1		ACC	
σ	γ_A	γ_I	Models	T	Train	Test	Train	Test	Train	Test
UK 2006 dataset										
2^1	2^{-5}	2^0	GLSVM-A	$\surd$	0.964 [0.006]	0.895 [0.009]	0.732 [0.022]	0.796 [0.031]	0.934 [0.005]	0.760 [0.029]
2^1	2^{-4}	2^{-2}	GLSVM-B	$\surd$	0.944 [0.010]	0.886 [0.005]	0.658 [0.023]	0.774 [0.020]	0.918 [0.008]	0.738 [0.017]
UK 2007 dataset										
2^2	2^1	2^1	GLSVM-A	$\surd$	0.766 [0.024]	0.760 [0.007]	0.284 [0.023]	0.241 [0.004]	0.863 [0.036]	0.920 [0.012]
2^0	2^{-5}	2^2	GLSVM-B	$\surd$	0.824 [0.061]	0.754 [0.002]	0.344 [0.076]	0.235 [0.024]	0.884 [0.020]	0.854 [0.004]

tween the GLSVM-A and GLSVM-B models. The disagreed data points are shown (left y-axis versus x-axis) in relations with the confident levels of correct or incorrect classification (because the classification output is obtained within a range $[0, 1]$, this is referred to as probability of correct or incorrect classification). The disagreed data points are also shown with the corresponding number of in-coming links/in-degree and out-going links/out-degree (right y-axis versus x-axis). A high positive confidence level of a data point implies that the learning model can provide a correct prediction of that data point with a high probability. Conversely, a high negative confidence level (close to 0) indicates that the learning model provides a wrong prediction of that data point with a low probability. The confidence levels are sorted. The out-degree and in-degree of each node/data point are shown respectively above and below the zero axis. The correct data points classified by the GLSVM-A and GLSVM-B models are separated by a solid vertical line. For simplicity, the disagreed data points are listed ascendingly. As can be seen, within dissent points, spam nodes (illustrated as triangles) cover a large area compared with normal ones (shown as circles) for the UK2006 data, and the converse situation is observed for the Uk2007 dataset. In both cases, the data points correctly classified by GLSVM-A always dominates that of GLSVM-B model. The GLSVM-B can best exploit the feature-based similarity graphs with relatively sparse connection for both UK2006 and UK2007 datasets. Those graphs statistically are more sparse than the hyperlink-based similarity graphs used by the GLSVM-A model. In particular, the average number of in-coming links to nodes that are correctly predicted by the GLSVM-B model is equivalent to 65.87% and 58.46% that of the GLSVM-A model for the UK2006 and UK2007 data, respectively. Similarity, the average number of out-going links

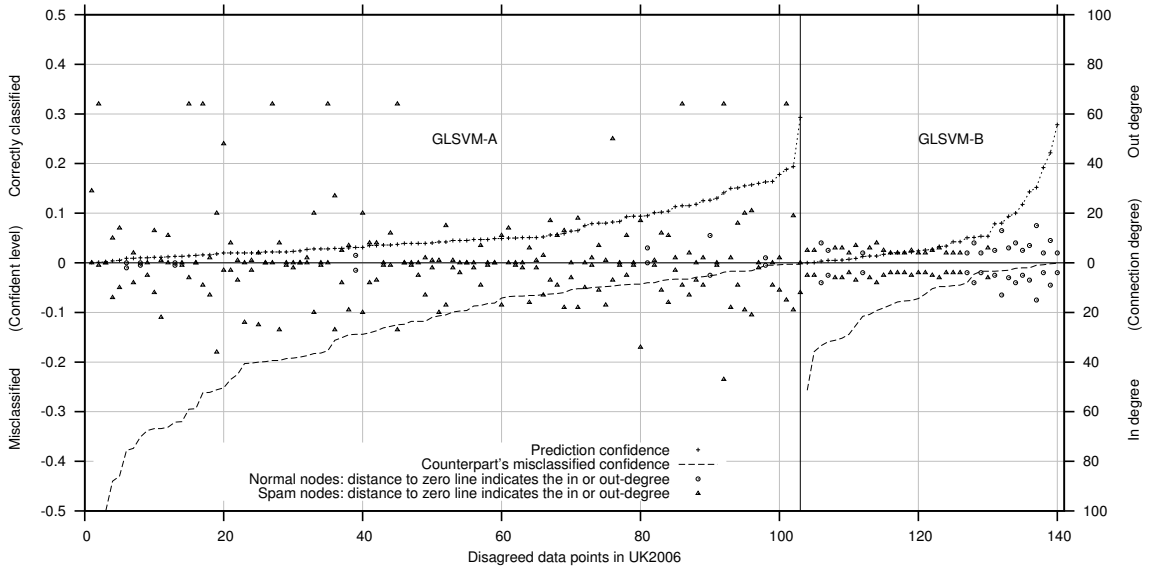

Figure 6.4: The disagreements among the testing data points when learning GLSVM with hyperlink-based similarity graph (GLSVM-A) and feature-based similarity graph (GLSVM-B) for UK2006 dataset

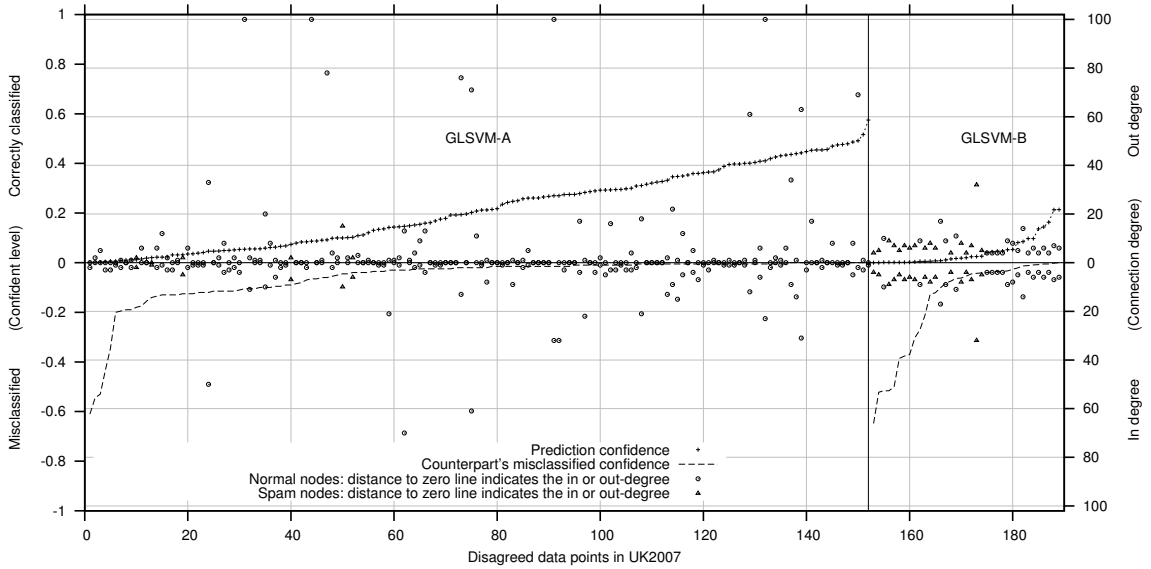

Figure 6.5: The disagreements among the testing data points when GLSVM with hyperlink-based similarity graph (GLSVM-A) and feature-based similarity graph (GLSVM-B) for UK2007 dataset

to nodes that are correctly classified by the GLSVM-B model is approximately 72.92% and 92.06% that of the GLSVM-A model for the UK2006 and UK2007 data, respectively. This clearly indicates that the topological information from the hyperlink graph is more useful than that of the feature-based graph.

Table 6.8: SKC+SVM performances on Web spam and Mutagenesis problems

SKC+SVM			AUC		F1		ACC	
Datasets	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam Datasets								
UK 2006	C+RL	$\surd$	0.965 [0.009]	0.888 [0.014]	0.653 [0.063]	0.611 [0.041]	0.948 [0.008]	0.619 [0.029]
UK 2007	C+RL	$\surd$	0.965 [0.026]	0.771 [0.020]	0.728 [0.186]	0.287 [0.024]	0.978 [0.014]	0.945 [0.020]
Metagenetic Datasets								
Whole Dataset	AB+C+PS	$\surd$	0.901 [0.010]	0.869 [0.006]	0.878 [0.004]	0.863 [0.006]	0.853 [0.005]	0.834 [0.018]
Friendly part	AB+C+PS	$\surd$	0.960 [0.012]	0.945 [0.011]	0.933 [0.014]	0.917 [0.006]	0.913 [0.018]	0.893 [0.018]
Unfriendly part	AB+C+PS	$\surd$	0.951 [0.051]	0.814 [0.023]	0.810 [0.036]	0.687 [0.030]	0.903 [0.020]	0.839 [0.116]

6.5.4 Spectral kernel clustering and Support Vector learning

This section investigates an integrated model consisting of SKC and SVM and which will be denoted simply as SKC+SVM. This model is similar to KKM+SVM as discussed previously. In the SKC+SVM deep learning, we incorporate the topological information in the (unsupervised) SKC model, followed by a layered SVM learning. The difference between the KKM+SVM and SKC+SVM models is the first layer. The results are summarized in Table 6.8. Both deep learning inspired models share similar behaviour on classification and generalization performance, although the SKC+SVM results are consistently improved by about 1% to 3% (i.e. as can be seen on the ACC generalization performance).

6.5.5 Support Vector Machines with Graph Laplacian Support Vector Machines

Two supervised kernel methods, the classic SVM and the graph based model GLSVM are integrated to form a two-layered learning architecture (denoted as SVM+GLSVM for simplicity). The first layer consists of 100 SVMs (same as the case of KKM+SVM) which is arranged in a parallel fashion, and is configured similar to the first SVM layer in the KKM+SVM and SKC+SVM models. In the second layer, the same number of GLSVMs are trained taking the outputs of the SVMs in the first layer as their additional inputs. Finally, all outputs of the GLSVMs are merged to produce the final results. The experimental results of SVM+GLSVM are summarized in Table 6.9. As can be observed, the model improves the

Table 6.9: SVM+GLSVM performances on Web spam and Mutagenesis problems

SVM+GLSVM			AUC		F1		ACC	
Datasets	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam Datasets								
UK 2006	C+RL	√	0.969 [0.002]	0.927 [0.007]	0.794 [0.010]	0.899 [0.003]	0.948 [0.003]	0.857 [0.006]
UK 2007	C+RL	√	0.871 [0.016]	0.839 [0.014]	0.463 [0.060]	0.368 [0.022]	0.911 [0.030]	0.892 [0.012]
Metagenetic Datasets								
Whole Dataset	AB+C+PS	√	0.834 [0.011]	0.848 [0.012]	0.853 [0.003]	0.847 [0.004]	0.824 [0.002]	0.826 [0.005]
Friendly part	AB+C+PS	√	0.940 [0.013]	0.920 [0.003]	0.928 [0.005]	0.921 [0.006]	0.907 [0.007]	0.904 [0.017]
Unfriendly part	AB+C+PS	√	0.981 [0.002]	0.867 [0.120]	0.984 [0.019]	0.783 [0.098]	0.990 [0.012]	0.881 [0.102]

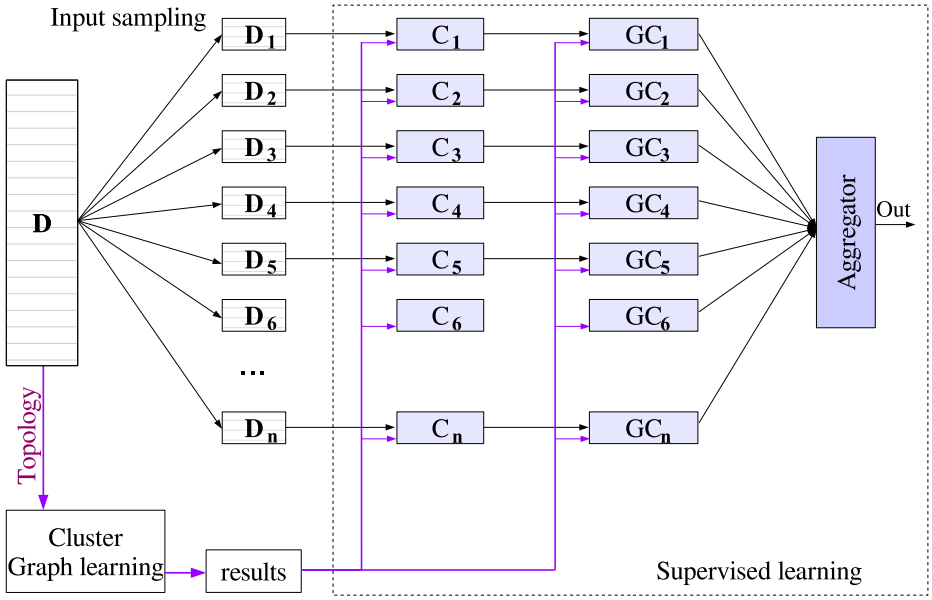

Figure 6.6: Deep learning with SKC clustering based graph topology learning, paralleled layer 1 with classic learner SVMs (denoted C_i) and paralleled layer 2 with graph learning GLSVMs (denoted GC_i). D stands for the input data while D_i is a sampled data from D. The A (stands for Aggregator) is operated by averaging over all the results obtained by individual GLSVMs.

generalization performance by 4% to 6% on the AUC indicator for the UK2006 and UK2007 datasets compared with other layered architectures presented earlier, i.e. KKM+SVM and SKC+SVM. A similar improvement can also be observed regarding ACC testing performance for the Mutagenesis problem.

6.5.6 A complex model using kernel learning and clustering

A complex graph-based kernel machine is proposed and applied here. It is developed on the basis of the SVM+GLSVM model, by incorporating an unsupervised SKC compo-

nent for each learning layer. We simply denote the model as SKC+SVM+GLSVM. The model is illustrated in Figure 6.6. The main difference between the SVM+GLSVM and SKC+SVM+GLSVM architectures is the integration of SKC for pre-training purposes in the latter. The clustering results of SKC are used as additional input to each learning component in the SVM and GLSVM layers. Placing a SKC clustering model before SVM+GLSVM provides a gist or overall information of the input space to the supervised layered learning. This mechanism theoretically obeys the deep learning model in the discipline of neural processing, which was proved advantageous in Chapter 5. Moreover, the layered architecture SVM+GLSVM has been shown to be beneficial when compared with single model learning, most especially, the GLSVM layer at the final stage helps the entire system to exploit efficiently both feature and relational information from the input. Hence, most of the inspiration to create SKC+SVM+GLSVM is from the deep learning idea and layered architecture in the graph learning domain. The final model can be seen as a companion with the complex model previously introduced, PMGraphSOM+MLP+GNN.

The best results achieved by the SKC+SVM+GLSVM learning system are summarized in Table 6.10. Around 1% improvement on AUC testing performance can be seen for the Web spam detection, and on ACC generalization performance for Mutagenesis problem when compared with the best performance achieved so far using kernel methods, given that AUC and ACC are the key evaluation metrics for two problems, respectively. A massive improvement can be observed if we compare the final results with the results obtained by any single learning model, for instance, up to 12% better than SVM performance, and up to 8% better than GLSVM performance given the key evaluation metric over all ranges of problem. It is noticed that for challenging and large scale problems like Web spam detection, even a 1% improvement in accuracy could result in hundreds of hosts to be correctly categorized.

Table 6.10: SKC+SVM+GLSVM performances on Web spam and Mutagenesis problems

SKC+SVM+GLSVM			AUC		F1		ACC	
Datasets	**Features**	**T**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Web spam Datasets								
UK 2006	C+RL	√	0.970 [0.005]	0.933 [0.009]	0.794 [0.010]	0.905 [0.004]	0.948 [0.004]	0.868 [0.009]
UK 2007	C+RL	√	0.890 [0.015]	0.848 [0.013]	0.516 [0.044]	0.382 [0.023]	0.936 [0.016]	0.914 [0.013]
Metagenetic Datasets								
Whole Dataset	AB+C+PS	√	0.898 [0.020]	0.872 [0.013]	0.867 [0.024]	0.860 [0.020]	0.842 [0.026]	0.843 [0.021]
Friendly part	AB+C+PS	√	0.930 [0.016]	0.929 [0.014]	0.921 [0.012]	0.932 [0.014]	0.896 [0.014]	0.910 [0.019]
Unfriendly part	AB+C+PS	√	0.951 [0.014]	0.836 [0.064]	0.890 [0.014]	0.833 [0.090]	0.934 [0.008]	0.904 [0.096]

6.6 Comparison and Discussion

6.6.1 Comparison between Kernel machines, and between Kernel machine and Neural network models

In this section, various models are compared in pairs. Comparisons can be made between kernel machine methods as listed in the upper part of Table 6.11. The table defines a reference symbol for each pair. For example, the symbol 'a' refers to the pair KKM and SKC. Similarly, comparisons between kernel machines and neural network methods can be made as defined in the lower part of Table 6.11. Again, the table defines a reference symbol for each of the pairs. These symbols are used for ease of reference in the following result tables. The comparative results between kernel machines are shown in Table 6.12. Comparative results between kernel machine and neural networks are given in Table 6.13.

Making a fair comparison is always difficult since models have different internal parameters as well as architectural designs. Table 6.12 presents the experimental results of relative pairs between a conventional and a graph based model. As can be observed, the graph based models outperform the classic ones in most cases for all datasets. The better performance degree is indicated by the blue and cyan level of the corresponding cells' background color in the table, where the higher color intensity of the cells means the better they perform compared with their associated counterparts. Statistically, the graph based models produce better results than the classic models in all cases for the UK2006 dataset. This is however not observed for other problems, either the large scale UK2007 dataset or small problems

Table 6.11: Different learning models divided into two categories

Paired models within kernel machines		
No	**Classic kernel machines**	**Graph kernel machines**
a	KKM	SKC
b	SVM	GLSVM
c	MKSVM	SVM+GLSVM
d	KKM+SVM	SKC+SVM
Paired models between neural networks and kernel machines		
No	**Neural networks**	**Kernel machines**
1	SOM	KKM
2	MLP	SVM
3	Lecun 5 MLP	MKSVM
4	SOM+MLP	KKM+SVM
5	PMGraphSOM	SKM
6	GNN	GLSVM
7	MSGNN	SVM+GLSVM
8	PMGraphSOM+MLP	SKC+SVM
9	PMGraphSOM+MLP+GNN	SKM+SVM+GLSVM

Table 6.12: Generalization performance comparison between **Classic** versus **Graph** kernel machines. Abbreviations are: S-supervised, U-unsupervised, D-deep learning.

Learning Models				**UK 2006**		**UK 2007**		**Whole Mutag**		**Friendly part**		**Unfriendly part**	
No	S	U	D	Classic	Graph	Classic	Graph	Classic	Graph	Classic	Graph	Classic	Graph
AUC performance													
b	√			0.882	0.895	0.726	0.760	0.868	0.815	0.948	0.919	0.853	0.835
c	√		√	0.887	0.927	0.760	0.839	0.870	0.848	0.923	0.920	0.882	0.867
d	√	√	√	0.887	0.888	0.771	0.771	0.863	0.869	0.943	0.945	0.821	0.814
F1 performance													
a		√		0.762	0.764	0.136	0.149	0.815	0.858	0.861	0.885	0.558	0.565
b	√			0.656	0.796	0.284	0.241	0.843	0.848	0.913	0.923	0.703	0.739
c	√		√	0.866	0.899	0.279	0.368	0.846	0.847	0.912	0.921	0.671	0.783
d	√	√	√	0.593	0.611	0.285	0.287	0.855	0.863	0.910	0.917	0.676	0.687
ACC performance													
a		√		0.675	0.682	0.632	0.637	0.791	0.807	0.828	0.855	0.807	0.811
b	√			0.650	0.760	0.937	0.920	0.812	0.817	0.888	0.899	0.813	0.860
c	√		√	0.808	0.857	0.832	0.892	0.820	0.826	0.883	0.904	0.849	0.881
d	√	√	√	0.607	0.619	0.945	0.945	0.826	0.834	0.884	0.893	0.838	0.839

like regression unfriendly data. This may imply that the topological data available in the UK2006 dataset is considerably more beneficial for graph based learning algorithms. This behavior was previously viewed in the connectionist models. Nevertheless, if only the key evaluation metric is considered for each dataset, the graph models always outperform the classical ones.

A comparison between neural network and kernel machine model performance is presented in Table 6.13. While the testing results are shown in the upper part, the corresponding

Table 6.13: Performance comparison of neural networks (**NN**) and kernel machines (**KM**). AUC and ACC are evaluation metrics shown for Web spam and Mutag problems respectively. Abbreviations are: S-supervised, U-unsupervised, T-topology and D-deep learning.

Learning Models					**UK 2006**		**UK 2007**		**Whole Mutag**		**Friendly part**		**Unfriendly part**	
No	S	U	T	D	NN	KM	NN	KM	NN	KM	NN	KM	NN	KM
Comparison of testing performance														
1		√			0.572	0.675	0.750	0.632	0.726	0.791	0.844	0.828	0.740	0.807
2	√				0.873	0.882	0.673	0.726	0.816	0.812	0.882	0.888	0.815	0.813
3	√				0.863	0.887	0.689	0.760	0.835	0.820	0.884	0.883	0.842	0.849
4	√	√		√	0.930	0.887	0.831	0.771	0.823	0.826	0.879	0.884	0.819	0.838
5		√	√		0.648	0.682	0.712	0.637	0.757	0.807	0.835	0.855	0.790	0.811
6	√		√		0.902	0.895	0.781	0.760	0.852	0.817	0.936	0.899	0.833	0.860
7	√		√	√	0.914	0.927	0.796	0.839	0.874	0.826	0.957	0.904	0.857	0.881
8	√	√	√	√	0.931	0.888	0.835	0.771	0.848	0.834	0.890	0.893	0.840	0.839
9	√	√	√	√	0.958	0.933	0.854	0.848	0.887	0.843	0.963	0.910	0.857	0.904
Comparison of corresponding standard deviation														
1		√			0.034	0.007	0.025	0.120	0.094	0.089	0.098	0.134	0.241	0.216
2	√				0.017	0.014	0.024	0.011	0.062	0.031	0.111	0.036	0.207	0.139
3	√				0.006	0.008	0.021	0.011	0.060	0.007	0.111	0.008	0.164	0.113
4	√	√		√	0.018	0.009	0.042	0.006	0.071	0.013	0.105	0.018	0.218	0.110
5		√	√		0.024	0.019	0.031	0.058	0.113	0.056	0.086	0.127	0.170	0.181
6	√		√		0.019	0.014	0.024	0.020	0.049	0.018	0.107	0.018	0.181	0.116
7	√		√	√	0.009	0.009	0.010	0.007	0.023	0.017	0.026	0.024	0.373	0.122
8	√	√	√	√	0.020	0.007	0.016	0.014	0.028	0.005	0.030	0.017	0.350	0.102
9	√	√	√	√	0.014	0.009	0.034	0.013	0.058	0.021	0.034	0.019	0.350	0.096

standard deviations are given in the lower part of the table. The better results between any two models are highlighted by some intensity levels of blue and cyan scale. As can be observed, kernel machines outperform their associated counterparts for the regression unfriendly part, since this dataset has a small number of input samples compared relative to the input dimension. The parametric models would find disadvantage when learning with insufficient data input.

For other datasets, it is difficult to conclude which models are predominant, though there are more blue cells the more that properties of the learning models are turned on. In particular, the better performance regarding each dataset (except for the regression unfriendly part) is all marked on the neural network models (shown intuitively at the bottom line No 9 of the upper part of this table). The NN models outperform their KM counterparts by 2.5% for UK2006, 0.6% for UK2007, 4.4% for the whole mutag and 5.1% for the regression friendly dataset. The predominant results of NN based models implies that the deep learning regime is better fitted, and the topology exploit-ability is more obvious in parametric

rather non-parametric models. The following will further explain the suitability of NNs for the tasks. In the graph learning domain, a remote path dependency problem between nodes might exist. The neural network models are found to be capable of handling deep learning problems or problems containing remote path dependencies, since the models are formed by deep neuron layers internally by themselves. Such model architectures could allow learning of a deep representation of data. The kernel machine architectures, on the other hand, seem not to allow a robust mechanism to effectively learn the deep representation of data or the remote path dependency problem in graph data structures. Hence, in terms of neural processing, the presence of a classifier after a clustering model in a hierarchical/deep learning model is absolutely accountable. Adding relaxation between neural layers to reduce the adverse effect of long term dependency would generally enhance network performance. Therefore, the model integration techniques are well suited to the neural network models, and the neural networks generally are more superior compared with graph-based deep learning models (with all the properties of learning models being used) when compared with the kernel machine ones.

Nevertheless, one obvious advantage of KM models is that the results obtained are seen to be more stable than with neural networks. The lower part of Table 6.13 clearly shows that the standard deviations of the results provided by kernel machines are generally lower than those of the neural networks. This implies that the neural network models are very much dependent on the initialization conditions, which is not the case for kernel machine learning.

6.6.2 Projection of final experimental results

In order to examine the coverage and differences between the training and testing sets in Web spam detection, the UK2006 and UK2007 datasets, Figure 6.7 and Figure 6.8 present the projection of input samples/hosts on two dimensional maps. This is done by running a PMGraphSOM on the final results of the complex kernel machine architecture, i,e input to

the PMGraphSOM the probability output of the SKM+SVM+GLSVM model. Because the PMGraphSOM is capable of preserving the structural topology of the input space, similar nodes in the graph are projected at nearby locations on the map.

As can be observed, in the UK2007 dataset, the training samples cover quite well the testing samples, while this seems not to be the case in the UK2006 dataset. For instance, considering the upper left corners of the training and testing mappings in Figure 6.7, while not many incorrect samples in the training set can be seen, many misclassified samples in the testing set are observed. The reason might be due to the unbalanced nature of the the two datasets. Whereas the spam nodes are dominant in the testing set, the major samples existing in the training set are non-spam. This is particularly seen in the upper left corners. Hence, the underlying feature information utilized for training seems not to be sufficient to allow the model to generalize well in the testing period. Another observation is that the misclassification often occurs when the spam and normal nodes are overlapped on the activation map.

6.6.3 Comparison between the GLSVM and GNN models

In this section, the experimental results of two representative graph based classification models, GLSVM and GNN are further examined. They are trained on the same hyperlink-based graph and use the same set of feature vector C+RL. Figure 6.9 and Figure 6.10 show the disagreed data points/results/nodes provided by the GLSVM and GNN models (i.e. the former predicts a node to be spam, but the latter predicts it to be normal and vice versa), and nodes' related properties. Hence, a disagreed node is either correctly identified by the GLSVM or by the GNN model, for both normal or spam node. In general, the nodes correctly predicted by the GNN usually dominate those predicted by the GLSVM model. Each figure contains three parts. The upper parts are similar to that in Figure 6.4 and Figure 6.5. The middle parts present the average in-degree and out-degree of disagreed nodes. Finally, the bottom

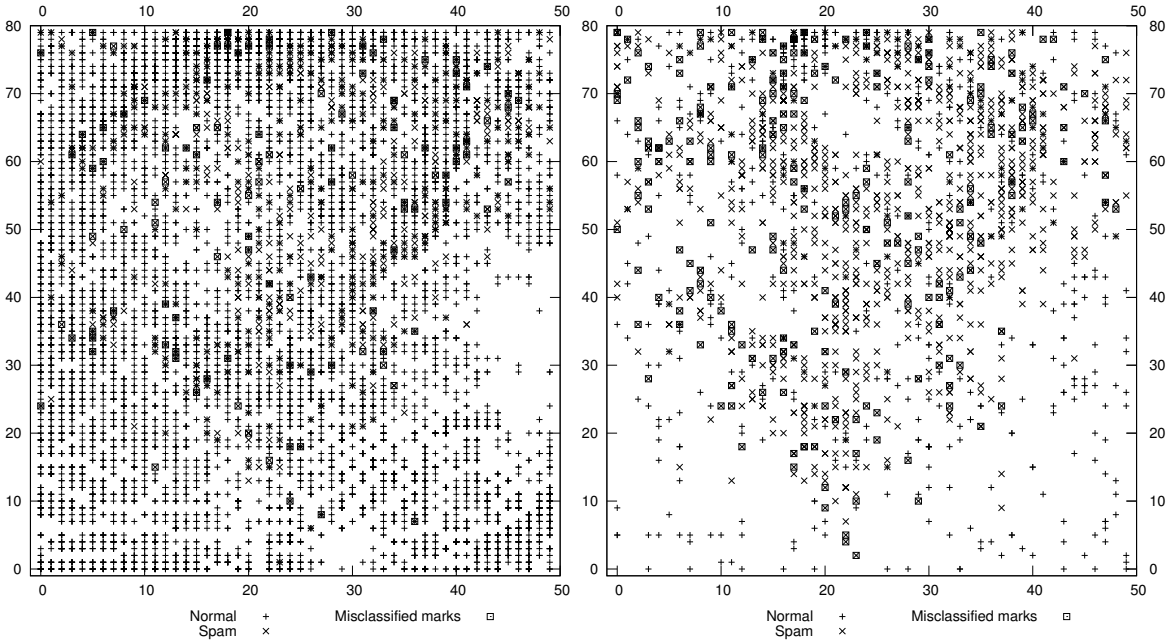

Figure 6.7: The illustration of the PMGraphSOM mapping (map size 80x60) for the UK2006 training (left) and testing set (right). The cross and plus shapes are respectively representing the spam and normal nodes. The squared shape indicates the misclassified samples for both training and testing set.

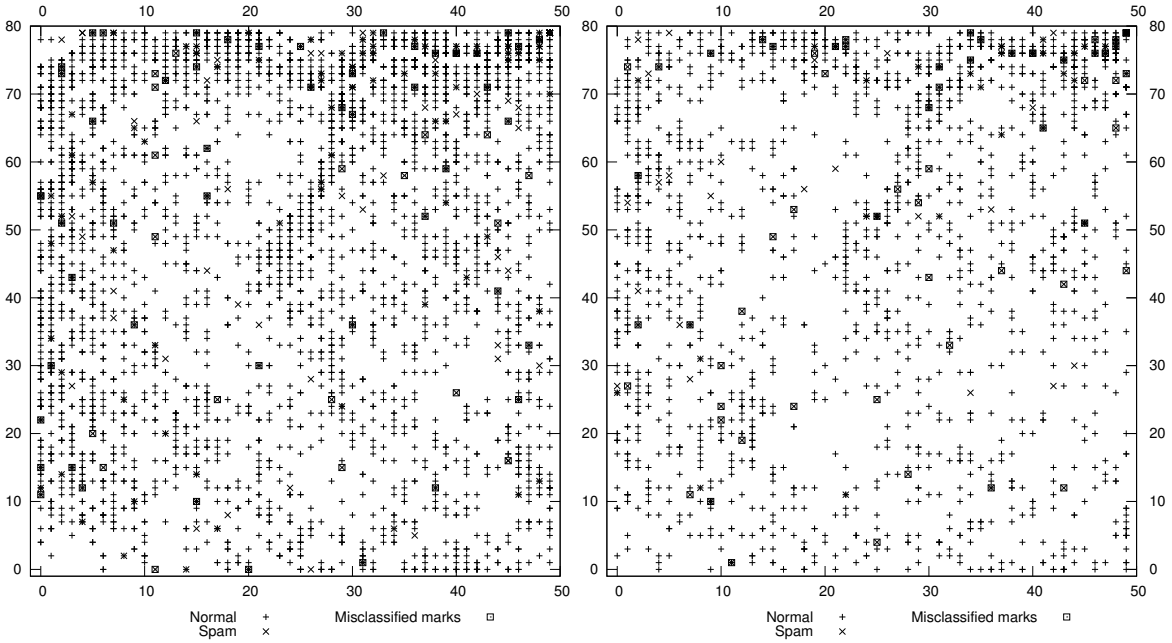

Figure 6.8: The illustration of the PMGraphSOM mapping (map size 80x60) for the UK2007 training (left) and testing set (right). The cross and plus shapes are respectively representing the spam and normal nodes. The squared shape indicates the misclassified samples for both training and testing set.

parts show the ratios regarding three typical content-based features [1].

Link-farm and link-based spam: It is observed in the UK2006 data that a link-farm might be present with a high possibility. Considering four spam data points/nodes at the bottom right corner of Figure 6.9 as examples, these all have a relatively large number of in-coming links. In practice, the large number of out-going connections is not strange since a website sometimes aims to cite as many other related pages as possible. If a spam site is linked by many other spam ones, this would account for the link-farm case. There are 94.9% and 96.6% in-coming links from other spam nodes to the first two indicated nodes (shown with 93 and 79 in-degree connections in Figure 6.9, respectively). Also, there are 100% spam links pointing to the other two indicated nodes (shown with 88 and 64 in-degree connection in Figure 6.9, respectively). Thus, it can be stated that the UK2006 dataset is featured by the link-based spam method. The middle part of Figure 6.9 further clarifies this statement. The in-degree of spam nodes, on average are about two times higher than that of normal ones. Theoretically, the GNN model can regulate the graph connectivity better than the GLSVM model because while the topological information only forms a component in the GLSVM's objective function, it becomes a structural part of the GNN model since the GNN algorithm iteratively learns patterns on its topological basis. Finally, the bottom part of Figure 6.9 appears not to support the statement that a content-based spam is the key method in the UK2006 case, since the three features are not provided with clearly differentiable values between nodes.

Content-based spam: As can be observed, a link-farm seems not to be present in the UK2007 data. More specifically, the spam node indicated with in-degree 38 (or 38 in-coming links) shown in the bottom right corner of Figure 6.10, for example, contains no connections from other spam nodes. This is again confirmed through the middle part of Figure 6.10, namely that the averaged in-degree of spam nodes is always significantly lower

[1] By looking into the three key content-based features: (1) Number of words of webpage (2) The independent LP of webpage and (3) The top 1000 corpus precision, we can particularly respectively examine (1) how long the page is (2) the popularity of the words used and (3) how the words are unrelated with each other.

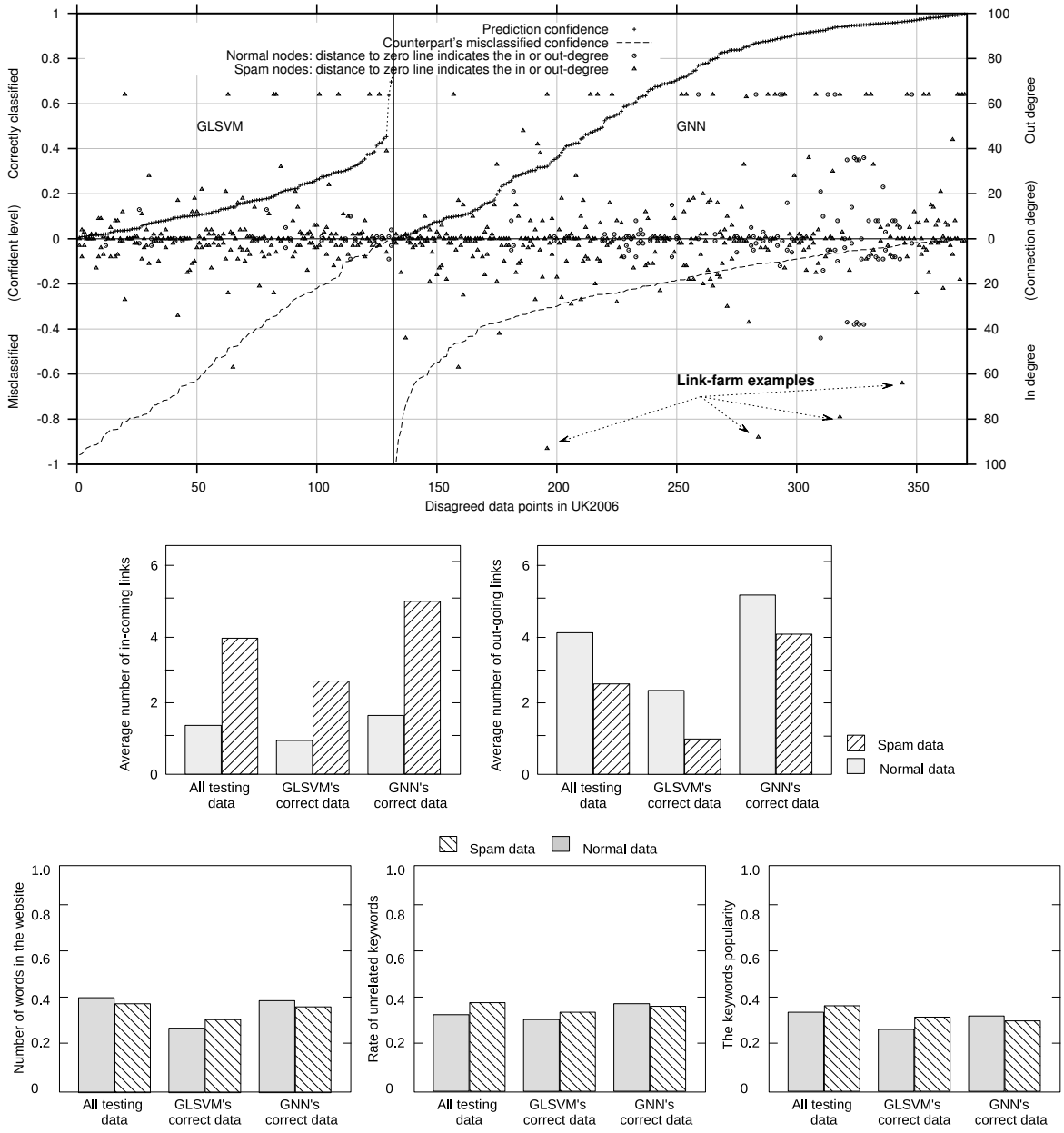

Figure 6.9: Upper: the disagreed data points when learning GLSVM and GNN with hyperlink-based similarity graph - Middle: the average number of in-degree and out-degree links - Bottom: The normalized averages of three content-based features in UK2006.

than that of normal nodes. However, the bottom part of this figure shows that the spam pages on average contain more words, popular words and uncorrelated words than normal ones. The interesting aspect shown in the bottom part is that the GNN seems to be more conservative than its counterpart in identifying spam nodes, since it can predict the spams with a very large number of unrelated words. This proves that the UK2007 data is characterized by

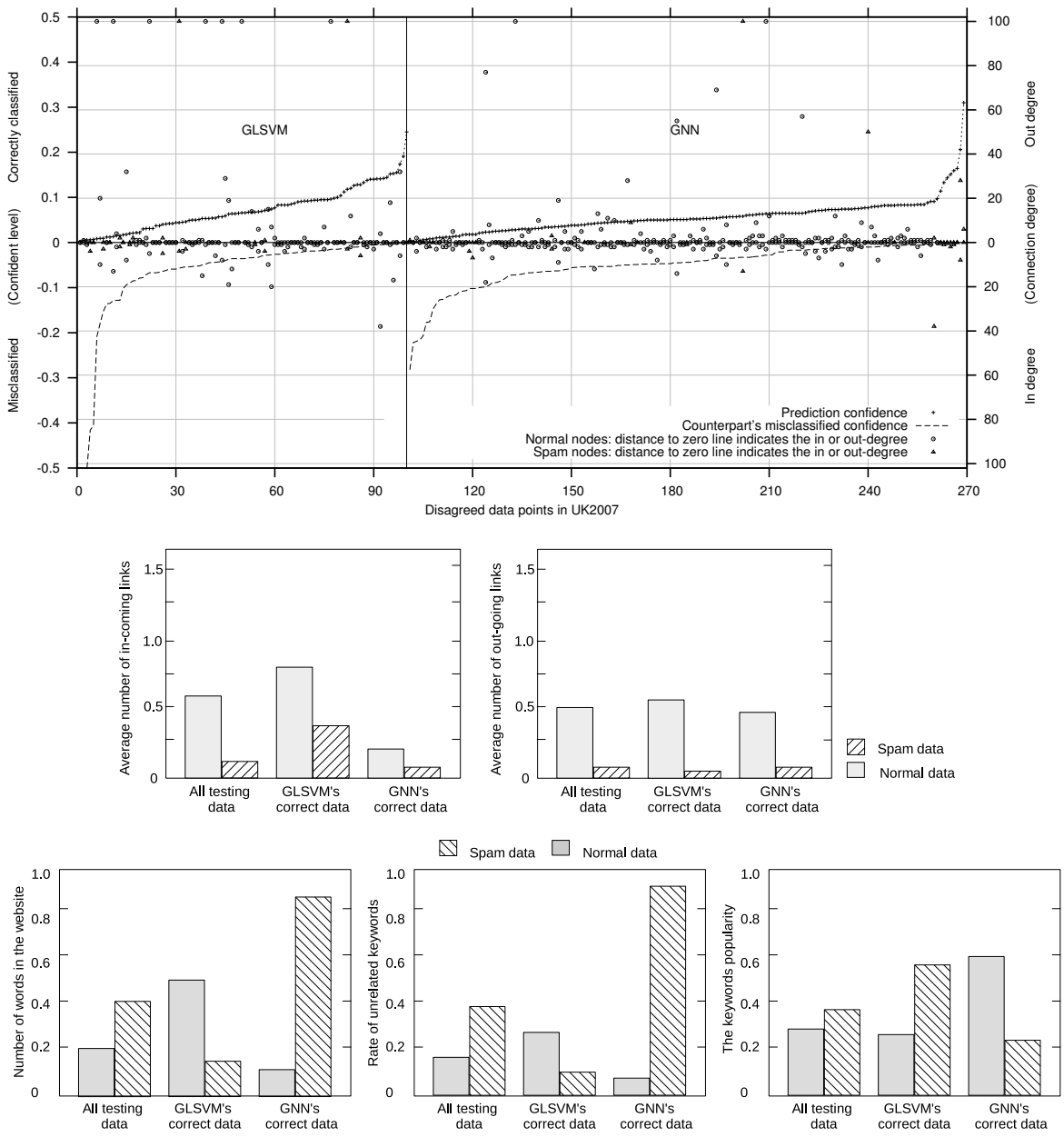

Figure 6.10: Upper: The disagreed data points when learning GLSVM and GNN with hyperlink-based similarity graph - Middle: the average number of in-degree and out-degree links - Bottom: The normalized averages of three content-based features in UK2007.

some content-based spam techniques. In fact, many spams were created by using either (1) keyword stuffing or (2) article spinning and duplication which are numerically represented by the three content-based features shown in Figure 6.10.

6.7 Conclusion

This chapter presented a systematic study on the effect of encoding the relational topology in the learning process of several kernel methods. It has been shown that the integrated model which takes the most advantage of individual learning components is the SKC+SVMs+GLSVM hierarchical architecture. It is quantitatively shown that the graph based kernel machines predominantly perform better than the more conventional kernel based predictors. We have also shown that the results obtained by kernel methods are generally more stable than the case with neural networks.

It can be derived that the layer architecture method is beneficial for both neural and kernel machine learning. Nevertheless, the neural network based models possibly gain much more advantage in hierarchical learning than the kernel based algorithms. In other words, the deep learning inspired models are practically realistic and effective in learning graph data structures. The context in which individual learning modules are arranged, is considered important in promoting the overall network performance. It is observed that long term dependencies are not likely to appear in kernel machines learning since the learning algorithms are not based on the gradient descent approach. However, this *is* the case concerning with the neural network models. Further investigation and solutions to long term dependencies will be shown later in Chapter 7.

Chapter 7

A Hierarchical neural network for graph learning

7.1 Introduction

This chapter introduces an integrated learning system containing three functions in order to tackle three associated problems arising in graph-based practical applications, by assuming that the effects of each of these three would be largely independent of one another. In the literature, each of these problems is treated individually. In the proposed integrated model, the first function is based on the deep learning strategy which is incorporated to deal with possible remote path dependency issues of a graph learning problem. The second function is a L_1 based regularization for significant feature selection from the large feature set. Thirdly, a non-uniform sampling function is used to solve possible imbalance class distribution in given datasets. As we assume the effect of each of these on one another is largely indeoendent, we can put them into the the same framework, and solve them one after the other. Note that we do not propose a method in which the effects of all these are considered simultaneously, and that the parameters involved are optimized simultaneously. This chapter will show that the proposed model is effective when addressing challenging real world

problems. This is shown through applications to two well-known benchmark datasets, viz., the UK 2006 and UK2007 web spam detection datasets, and a large scale XML document classification problem, the INEX 2008 dataset. It is found that the proposed approaches help to obtain state-of-the-art results on those datasets. Thus, through such empirical investigations, we have shown that th assumption of the independency of the effects of these three functions is valid, at least in these practical problems. There could be other problems for which such an assumption might not be valid, though we have not found them yet in our experience.

First, in a graph based classification problem, the learning performance of a learner would be exasperated if the distribution of class labels among the input is severely imbalanced. That is, for instance, there are many more negative examples than positive examples in a binary class problem. When dealing with severely imbalanced training datasets, it is common to use a sampling method on the inputs. In general, the aim of sampling methods is to match the number of negative examples with positive examples in a particular manner [109]. For example, a rough balance between two class examples based on Negative Binomial Distribution (NBD) was proposed in [110]. An alternative approach is to re-weight examples which are misclassified in this learning round, so as to better classify them in the next learning round [111, 112]. However, to the best of our knowledge, such a technique has not yet been applied to the graph structured domain. Indeed, it is intuitively clear how such techniques could be helpful within the context of graph domain.

Secondly, the feature vector of an input could be very high dimensional which can cause problems known as the *curse of dimensionality*. A number of approaches have been proposed when dealing with high dimensional input feature vectors. Many have engaged some forms of pruning by computing the maximum information gain on the vector elements [113, 114, 115, 116]. Some have applied regularization approaches [117, 118, 119]. This chapter will employ the L_1 regularization technique which allows the selection of important

features in the input space [118]. We will specifically apply L_1 regularization to the graph based problem learning.

Thirdly, a common deep learning approach consists of multiple stages of an unsupervised learning unit, followed by a supervised learning module [21], e.g., as those explored in previous chapters. In practice, however deep learning has so far only been applied to feature vectors which are assumed to be independent. To the best of our knowledge, apart from the work presented in [17, 93], there have not been any other attempts in developing a deep learning strategy on graph structured inputs. The main reason is that it is not clear how one may take into account the topology of a graph in the input of an unsupervised learning model.

In general, by assuming that the effects of these three on one another are largely independent, this chapter proposes an integrated model which combines three sub-solvers, namely a remote path dependency solver, an imbalanced data solver, and a high input dimension solver, and solve the problem sequentially one after the other. This is a non-trivial task as it is not intuitively clear how to combine these techniques such that the positive effects of the individual technique are consolidated. We found that by making such an assumption, and by solving each of the sub-problems sequentially, we achieved the best results on three different challenging datasets. This is an interesting finding since there are not many techniques which perform as robustly and well on those datasets. Moreover, by applying such a method to the practical problems, the assumption is validated *post hoc*, at least for these practical probelms evaluated, as it showed an improvement on the performance when compared with other techniques, which might not have taken such effects into account.

The rest of this chapter is organized as follows: The general hierarchical deep learning scheme is presented in Section 7.2. Section 7.3 describes the experimental procedures. Section 7.4 provides the experimental results, comparisons and discussions. Finally, some conclusions are drawn in Section 7.5.

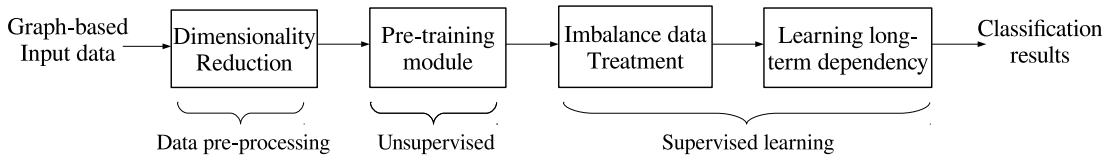

Figure 7.1: The multiple staged learning model: compact diagram.

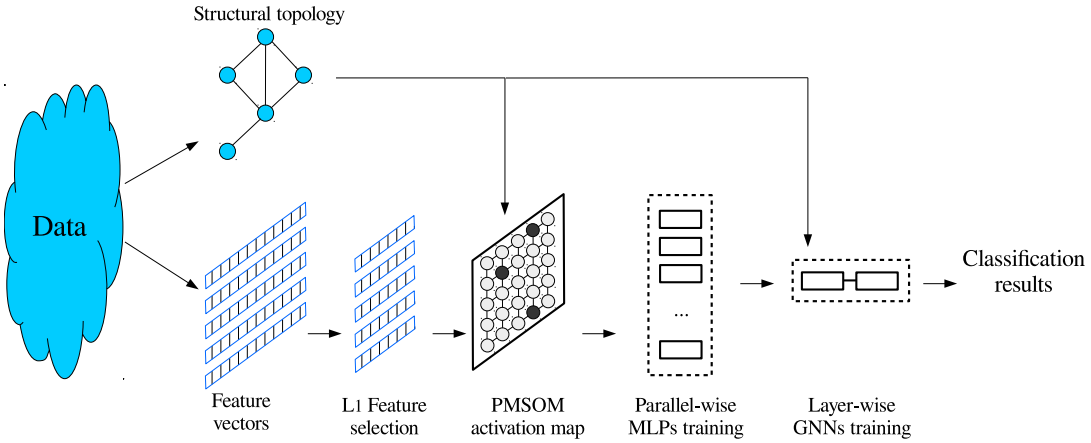

Figure 7.2: The multiple staged learning model: detailed diagram.

7.2 Graph-based hierarchical neural network: the learning system

This section presents the work flow or the order of components within the hierarchical learning system. A compact diagram of the model is illustrated in Figure 7.1, while a more specific model design can be seen in Figure 7.2. Intuitively, the proposed model consists of four main components [1], namely dimensionality reduction which is denoted as a pre-processing stage, unsupervised pre-training, an imbalanced data solver and a long-term dependency solver. The two latter solvers are based on independent supervised learning models. It can be seen that the input data is constituted by two separable pieces of information: structural topology and feature vectors. Because of the possible high dimensionality of the feature vectors, it is reasonable that the dimensionality is reduced by using a L_1 regularization approach which effectively selects the significant features from the original set. Regarding

[1] The model's components are applied interchangeably with the following: a model's units, modules, learning stages or levels.

the dimensionality reduction in Section 7.2.1, the L_1 approach to the binary classification problems will be presented first, and then the information gain approach for text document categorization problem will be shown.

The reduced feature vector set obtained after the dimensionality reduction approach, together with the graph structural topology, will form the input to the pre-training unit. In practice, almost all unsupervised learning models do not assume that the feature vectors are related through an underlying graph structure. The only known unsupervised learning technique which could handle general graph structured inputs is called PMGraphSOM [42]. The PMGraphSOM considers the proper ratio between topology and feature information in its learning process so as to achieve the best performance where the relational context and nodes' feature are exploited. Once fully trained, the PMGraphSOM will produce clusters of input samples, in which feature vectors that are similar in feature space as well as similar in the relational context in which they occur within the graph, will be clustered together in the display space of the PMGraphSOM. The coordinates of these mappings along with the feature set derived from the dimensionality reduction approach, will form an augmented reduction input (ARI) for a subsequent stage.

The ARI obtained so far is then utilized for the imbalanced treatment stage. Note that the ARI is now a set of independent vectors rather than a graph. The rationale is that the mappings produced by the PMGraphSOM are in fact the projection of the nodes in the graph. Hence, the coordinates of these mappings are the encoding of features as well as of the context within which the features occur in the graph. As a consequence, the ARI contains an abstract description of the graph topology in vectorial form. This enables us to deploy a standard MLP model at this stage. The MLP is seen as a weak learner in terms of a parallel based sampling approach. Since the MLP is a supervised method which relies on the node labels, and since assuming that the training dataset may be severely imbalanced and hence, a hybrid sampling algorithm (HNBD) is introduced with the base learner MLP and

the ARI as inputs. Details of the HNBD sampling method will be described in Section 7.2.2.

Once the training of the imbalanced solver is completed, its output will be the predicted classes of the nodes that can be associated with a given input. This prediction together with the graph topology information, will then form the input to a layered GNN module. More information about the layered GNNs architecture will be found in Section 7.2.3. The rationale for using the GNN is that in some situations, a single hidden layered MLP might have some difficulties in obtaining a good mapping between the input and the output. Hence a multiple hidden layer MLP could be deployed in obtaining an accurate input output mapping. Hence, in the graph input situation, it is surmised that a multiple GNN would allow a much better depiction between the graph inputs and the outputs. The accuracy of a single hidden layer MLP is limited by the accuracy of the mappings produced by the PMGraphSOM. The PMGraphSOM performs mappings to a discrete display space, hence these mappings are to be considered an approximation. These mappings form part of the input for the MLP so that the output of the MLP is an approximated prediction of the pattern class. By augmenting the original graph structure with these predictions, this will enable the GNN to learn and reduce the residual classification error based on information about a graph's topology. Finally, the output of the GNN is the resulting classification of the nodes.

One will find that the results can be improved through the cascaded stages of the GNNs. Here, the prediction outputs of a GNN are used to label the associated nodes in the graph and then to train another layer of GNN on this input. This strategy is repeated until no further improvements in the classification of the nodes in the training set are observed. The observation that a GNN can improve the results of the previous and fully trained GNN, was an unexpected finding given that the optimal capabilities of the GNN have been proven formally [60]. Thus, from a formal perspective, there should be no reason as to why subsequent GNNs should improve on the results of a single GNN. We suspect that the reason behind our observation can be attributed to the gradient descent based learning method in

the GNN. Such methods are known to converge to local minima which may be far away from the global minimum. By cascading several GNNs, each GNN will be trained to reduce residual errors. This is possible since each of the GNNs is initialized differently (i.e. randomly), and hence enables the system to find a better local minimum with every step in the cascade. It will be demonstrated that the cascade of GNNs with a boosting mechanism would be beneficial in learning challenging graph-based problems. Another reason might be that the optimality proof is based on the assumption that the number of hidden units (and hence the number of adjustable parameters) is unbounded, whereas in practice, the number of hidden units is always limited. Thus, a second GNN can improve the results of the first GNN if the number of hidden units is less than optimal for a given learning problem.

Generally speaking, the underlying idea of the integrated learning system is that it extends the approach in [17] as it incorporates a dimension reduction of the input feature space. It likewise allows a sampling technique to be incorporated in order to deal with possibly severely imbalanced class distribution datasets. These two components were missing in [17], rendering it less effective when compared with the proposed model. Deep learning occurs in the integration stage between the PMGraphSOM and the MLP. This deep learning differs from the standard deep learning approach [21], in that we take into account the underlying graph topology representing the relationships among the feature vectors. By imposing an unsupervised learning model, as a front-end in the processing of the data, the long term dependency effect will be reduced. It is also surmised that the contextual relation is more effectively exploited by using the cascaded section of GNNs [93] at the end of the learning system.

7.2.1 Dimensionality reduction

7.2.1.1 L_1 Regularization for feature selection

In the domain of feature selection, a relatively large number of approaches have been proposed to address the *curse of dimensionality* [120]. The intuitive benefits of these feature selection methods are: (1) The reduction of computational cost as the size of approximator becomes small along with the input dimension; (2) The enhancement of the generalization prediction accuracy by reducing overfitting of the training set [121].

Many approaches engage *wrapper* methods that use a predictive model to score feature subsets. *LASSO* or L_1 regularization method penalizes the regression coefficients, shrinking many of them to zero. Features with non-zero regression coefficients are selected by the LASSO algorithm. Similarly, a large number of regularization approaches has been introduced. In particular, *Least angle regressions* or LARS [117] is an efficient algorithm of L_1. *Ridge regression* or L_2 [118] and $L_{1/2}$ [119] are different variants of regularization-based feature selection techniques.

Other approaches are categorized by the *filter* selection type. Filter methods select features by ranking them using different criteria such as *mutual information* or *correlation* coefficients [113]. Peng et al. proposed mutual information based feature selection [115]. They select a condensed set of features following the maximal statistical dependency criterion defined on the mutual information between the joint distribution of the selected features and the class labels. Other methods based on information theory include the likelihood maximization approach [113], and correlation based feature selection [116]. The correlation measure evaluates subsets of features following the criteria that useful feature sets should encapsulate those highly correlated with the target labels, yet uncorrelated to each other. In addition to filter and wrapper, *Embedded* methods integrate feature selection in the model learning process [120].

In the proposed model, we apply a well-known approach to the selection of feature vec-

tors using L_1 regularization formulation [122]. Assume that there are N training samples, $\mathbf{x}_i$, $i = 1, 2, \ldots, N$, $\mathbf{x}_i$ is a M dimensional vector, and y_i, $i = 1, 2, \ldots, N$ the corresponding target values. Then it is possible to model the targets as follows: $\mathbf{y} = \beta_0 \infty + A\beta + \epsilon$, where β_0 is a constant, ∞ is a N-dimensional vector with all elements 1, A is $N \times M$ matrix, the i-th row being the vector $\mathbf{x}_i$, β is a M vector, with elements β_j, $j = 1, 2, \ldots, M$, ϵ is a noise vector. The j-th feature is selected if $\beta_j \neq 0$. So the feature selection problem can be formulated as follows: $\min_{\beta_j} J = \frac{1}{2}\epsilon^T\epsilon$, subject to the constraint $\sum_{j=1}^{M} \|\beta\|_p^p \leqslant t$, where $\|\cdot\|_p^p$ is a L_p norm, and p is an integer. If $p = 2$ this is the usual squared norm, while if $p = 1$ this will be the usual L_1 norm. The constraint will force some of the β_j to 0.

Without loss of generality, one can assume $p = 1$. Thus, the constraint is given as $\sum_{j=1}^{M} |\beta_j| \leqslant t$. This optimization problem can be solved using the Lagrange multiplier technique by augmenting the cost function as follows: $\overline{J} = \frac{1}{2}\epsilon^T\epsilon + \lambda \sum_{j=1}^{M} \beta_j$. We propose to use the alternating directions multiplier method (ADMM) to solve this problem [123]. This is because the ADMM is an efficient method for solving L_1 regularization problems with linearly separable cost functions, and linearly separable constraints. Our problem of feature selection fulfills these conditions and hence the ADMM is deployed to solve the regularization problem. The concept behind ADMM is trivial. It evaluates the gradients of the augmented cost function, and then the unknowns are updated using the Gauss-Seidel updating method and using Newton's method for solving the algebraic equations (each variable once found will be used in subsequent updates, instead of updating all the variables at the same time).

7.2.1.2 Information gain method for text categorization problem

In information theory, the mutual information of two discrete variables X and Y is

$$I(X, Y) = \sum_{x \in X} \sum_{y \in Y} P(x, y) log_2 \left(\frac{P(x, y)}{P(x)P(y)} \right), \tag{7.1}$$

where P denotes probability. In text mining, the information gain is calculated as the mutual information between terms and topical categories.

Let $C = c_i, i = 1, 2, ..., n$ be a set of categories in the input space. One can define $T = t, \bar{t}$ in which t and $\bar{t}$ refer to an occurrence and non-occurrence of term t, respectively. Then the information gain between C and T is

$$I(T, C) = \sum_{t \in T} \sum_{c \in C} P(t, c) log_2 \left(\frac{P(t, c)}{P(t)P(c)} \right) \tag{7.2}$$

Or for each term t one would have

$$I(t) = \sum_{i=1}^{n} P(t \wedge c_i) log_2 \left(\frac{P(t \wedge c_i)}{P(t)P(c_i)} \right) + \sum_{i=1}^{n} P(\bar{t} \wedge c_i) log_2 \left(\frac{P(\bar{t} \wedge c_i)}{P(\bar{t})P(c_i)} \right) \tag{7.3}$$

The information gain (IG = I) calculated above will be the selection criteria for the dimensionality reduction strategy. Indeed, one can select a number of terms that are associated with the highest IG values. Then a given text document, which is known as a set of terms, can be compressed into a series of those selected terms. As shown in Chapter 4, each term is represented by a concatenated $TF.IDF$ value. Hence, one can finally assign a feature vector containing $TF.IDF$ values of selected terms to a given document.

7.2.2 HNBD sampling for imbalanced data issue

A large number of studies have been conducted to reduce the effect of the imbalanced class distribution on predictive models [124]. Fundamentally, the approaches are divided into three main approaches, namely *Bagging* or sampling, the *Boosting* algorithm [125], and *Cost-sensitive* learning [126, 127].

The Bagging aggregating or sampling idea was pioneered by Breiman [109]. The concept refers to a method of generating multiple versions of a predictor trained individually on boostrap replicates of the learning set. The actual output of the prediction model will be de-

cided by voting on the individual predictor's outputs [109]. More recently, many sampling approaches have been developed based on a bagging-based ensemble to address class imbalanced problems [128, 129, 130] such as *undersampling* [129], the *oversampling* method [130], and the *synthetic sampling* technique [128]. A *roughly balanced bagging* approach was proposed in [110]. In a boostrap replica, all minority examples are selected while the number of majority examples are chosen following the negative binomial distribution (NBD) with a fixed probability of success of 0.5. Thus, the average number of positive and negative examples through all the sampling runs are almost equal [110].

While sampling is related to the data level approach, boosting ensembles are seen as algorithm level methods [125]. Schapire first proposed the boosting idea, in that the learning mechanism would turn weak classifiers into a strong learner [111]. The first and well-known approach based on a boosting ensemble was Adaboost [131]. Adaboost increases the weights of misclassified instances after each training round. The incorrectly classified examples in the current iteration will be given more focus in the next iteration. Each individual classifier is also assigned another weight for the purpose of assessing the final output of the learning system [131]. Recently, similar methods have been proposed, including SMOTE-Boost [132] and DataBoost.IM [133]. In addition, other methods apply boosting directly in learning algorithms, such as the boosting neural network [134] and the boosting support vector machine [135].

Finally, cost-sensitive combines both data and algorithm level approaches to engage different misclassification weights for each class during learning process [125]. Cost-sensitive is applied in a neural network by incorporating the high weights of misclassification instances into the objective function [136]. Metacost [126] is a bagging-like ensemble. It relabels the training examples with their estimated minimal cost classes. Fan et al. introduced Adacost [127], which is a variant of Adaboost [131]. Adacost uses the cost of incorrectly classified examples to update the training distribution after each iteration [127].

Costing is presented in [137], and applies the cost proportionate rejection sampling method. Another cost-sensitive method applied in a kernel machine can be found in [112].

In this section, we propose a new sampling approach. Its underlying idea is to combine under-sampling [129] and roughly balanced bagging [110]. The former uses a sampling ratio over the number of majority examples, while the minority instances are kept fixed. The latter relies on NBD to sample the number of majority examples. The probability of success remains unchanged at 0.5, such that the average number of majority class examples are roughly equivalent to that of small class ones. We will demonstrate that in severely imbalanced datasets the probability of success should not be fixed in order to achieve the best results. We propose a hybrid sampling approach (HNBD) that takes advantage of the NBD based sampling while the size of majority-class sampled sets is flexible and will be validated in the learning phase.

In our experiments, the optimal value of q is searched and validated during the training phase of the prediction model. Another justification is that when the training subsets are down-sampled from the majority class, the idea of under-sampling [129] is still satisfied. The average size of the negative class is not expected to be smaller than the positive one in a sampled subset. In practice, we found that when $q < 0.5$, the prediction performance is improved for severely imbalanced datasets. In general, the HNBD still inherits the dynamic sampling of majority examples via the NBD method.

Specifically, NBD is defined as a probability distribution of the number of success m^+ in Bernoulli trials before the number of failures m^- appear. This is computed by the probability function shown in Equation 7.4.

$$p(m^-|m^+) = \begin{pmatrix} m^+ + m^- - 1 \\ m^+ \end{pmatrix} q^{m^+}(1-q)^{m^-} \tag{7.4}$$

Therefore, in order to obtain the distribution of m^-, the value of m^+ and q must be presented.

Algorithm 1 The HNBD algorithm inputs the full training set $\mathcal{S}$. Base classifier L. Number of base classifiers K. Probability of success q

procedure HNBD-TRAIN($\mathcal{S}, L, K, q$)
  for $i = 1$ **to** K **do**
    Sampling the full training set to achieve subset S_i, $S_i \in \mathcal{S}$. S_i = $\mathcal{C}^+$ + c^-, $c^- \in \mathcal{C}^-$. c^- are randomly chosen in $\mathcal{C}^-$ with no duplicates. Let n^- be the size of c^- then n^- following NBD with q.
    Running classifier L_i using S_i as training set, to obtain the trained model M_i.
procedure HNBD-PREDICT(M_i,$\mathcal{U}$)
  For each sample $x \in \mathcal{U}$ present to M_i, we derive two-dimension output P_i^+ and P_i^- referring to probabilities that the given sample belongs to +1 and -1 class, respectively.
  Aggregate all the models' results to get the final probability output P_{out} of x.
  $P_{out}(x) = \frac{1}{K} \sum_{i=1}^{K} \frac{P_i^+}{P_i^+ + P_i^-}$

It can be referred to a sampling method that m^- means the -1 class samples while m^+ implies +1 class ones. In particular, one can draw a number of -1 class samples, n^- from $\mathcal{N}^-$ samples based on NBD, if the number of +1 class samples, $\mathcal{N}^+$ and probability q are given in Equation 7.4.

The proposed HNBD sampling method is presented in Algorithm 1. Let $\mathcal{S}$ be the training set, K be the number of base learners (L denotes a learner), and q be a pre-defined probability of success. These are given as algorithm's inputs. Hence, it allows us to judge on different q values. After training a base classifier L_i, a trained model M_i (for example in MLPs, a trained model is a MLP with its parameters being tuned after training) will be derived. In the prediction stage, the testing/unknown set $\mathcal{U}$ is presented. An aggregation of all individual trained models is applied to obtain the final probability output for each sample. In this stage, the prediction performance can be derived for both training and testing sets. Referring to web spam detection problems, the output probability P_{out} (Equation 7.5) becomes the $spamicity$ and is calculated as follows:

$$spamicity = \frac{\sum_{i=1}^{K} \frac{Pspam_i}{Pspam_i + Pnormal_i}}{K}, \tag{7.5}$$

where $Pspam_i$ and $Pnormal_i$ respectively correspond to P^+ and P^-.

7.2.3 Learning the remote path dependencies

In practice, a recurrent or recursive neural network commonly suffers from the long term dependency problem, in which the network's error vanishes while being fed back via the back-propagation algorithm. In the graph learning domain, the error is back-propagated through the deep graphical structure, the gradient contribution gradually decreases along with the depth of the graph. Possible approaches to this long term dependency problem are numerous, including the Leaky integrator learning algorithm [74], Long short term memory [26] and Genetic algorithm [36, 75]. The hierarchical methods proposed in [76, 77] initially deal with the long term dependency in graph learning problems, however are unable to address the imbalanced issue and the curse of dimensionality of the data input. These missing properties cause the approaches to be less effective in learning challenging problems.

One of our early attempts in addressing the issue of long term dependency in the graph domain was using GA or PSO as an optimization function in the GNN learning algorithm. The main purpose of this method is that we use the GA/PSO module to produce good initialization for GNN network parameters, such that the long term dependencies are not durably affected in the GNN learning algorithm. We seek a set of parameters through a series of chromosome generations (for the GA case), or through random solutions and updating searches in the pre-defined population (for the PSO case), in order to minimize the objective function. When applying the GA/PSO optimization algorithm to the GNN, the corresponding model is fairly slow compared with the GNN learning by itself. The computational requirement for both algorithms is very much dependent on the population size and the number of generations. It was roughly estimated that the applied methods are about 200 times slower than the original GNN learning. The PSO applied to GNN can provide more promising results than the GA and the GNN model by self, since the PSO based model required less learning iterations to achieve the equivalent accuracy results. However, the final accuracy performance is about the same for those learning models given that the experiments

Algorithm 2 Boosting-GNN algorithm

procedure B-GNN($w_i, t_i, pr, l, Pc, Nc, K$)
  repeat
    Build a new graph $\boldsymbol{G}_k$ with nodes labeled by pr
    Compute threshold δ that maximize F-measure on the training set.
    for $i = 1$ **to** q **do**
      if $(t_i = 1)$ and $(pr_{n_i} < \delta)$ **then**
        $w_i^{new} = w_i^{old} + \|t_i - pr_{n_i}\| * Pc$
      if $(t_i = 0)$ and $(pr_{n_i} > \delta)$ **then**
        $w_i^{new} = w_i^{old} + \|t_i - pr_{n_i}\| * Nc$
      Otherwise $w_i^{new} = w_i^{old}$
    Train GNN_k with the cost function
    $\mathcal{E} = \sum_{i=1}^{q}[(t_i - pr_{n_i}) * w_i^{new}]^2$
    Update prediction values pr_{n_i}
    $k = k + 1$
  until $k = K$, where K is the pre-defined maximum number of training cycles.

were conducted on the Web spam detection problems.

In this section, a novel layer based boosting graph neural network (B-GNN) is introduced. The Pseudo code of the algorithm is given in Algorithm 2. Without loss of generality, assuming that the given learning problem is binary, the input of the B-GNN model and several variables are explained as follows.

(1) $k = 1$, GNN_k denotes the k-th GNN trained on the $\boldsymbol{G}_k$-th graph.

(2) Initialize weights of nodes $w_i = 1, i = 1, 2...q, q = |\boldsymbol{N}|$.

(3) Training target of node n_i: $t_i = 1$ if $n_i \in$ +1 class, $t_i = 0$ if $n_i \in$ -1 class.

(4) First GNN is trained to obtain current prediction values $pr_{n_i} = \boldsymbol{\Omega}(\boldsymbol{G}, n_i)|pr_i \in \mathbb{R}, i = 1, 2...N$

(5) Pc is a constant weighting the positive class.

(6) Nc is a constant weighting the negative class.

In the algorithm, we optimize the threshold of the F_1 measure to balance the precision (P) and recall (R) values of the retrieval performance. The F_1 measure is defined as the harmonic means of P and R. In our implementation, we used an initial condition of weighing the ratio

of the weights associated with the positive samples and the negative samples by the inverse ratio of the number of positive samples N^+ to the negative samples N^-, and then used a greedy algorithm to find the approximate threshold T which maximizes the F_1 measure. The B-GNN algorithm uses a graph $\mathcal{G}_0$ as input. A GNN_0 is trained on the $\mathcal{G}_0$ and the output of GNN_0 is obtained. That output is used to compute a weight matrix of all the nodes and to re-label the nodes (the output is added to the original feature vector attached to the node) to obtain a modified graph $\mathcal{G}_1$, which is then used to train a GNN_1, and so on. Thus, the weight matrix is re-computed after each GNN layer. The node corresponding to a high weight value is associated with a high prediction error at the output layer of the GNN. The approach forces the GNNs to gradually improve on the residual classification errors. The algorithm is terminated when no further improvement can be obtained. Hence, the approach optimizes the classification rate which results in a general improvement of the classification performance. The advantages of B-GNN is that it remembers the historical weights of all nodes in the graph and cumulatively updates the weights through the layers of the GNN.

7.3 Experimental procedures

Firstly, for evaluation purposes, three metrics AUC, F1 and ACC are applied. The evaluation method that used for binary problems, the UK2006 and UK2007 datasets, is AUC. In practice, however we will use all three listed metrics for a comprehensive comparison. Because it is observed that web spam detection problems are of severely imbalanced class distribution, the most suitable evaluation method for this would be AUC. Hence, the performance exhibited by AUC would be seen to be the most important. In addition, for the text categorization field, the suitable methods include macro average recall (R_{macro}), micro average recall (R_{micro}) and F1, all of which are utilized in the INEX document classification problem.

The features provided with the UK2006 and UK2007 datasets consist of three groups:

96 *content based* features, 41 *link based* features, and 138 *transformed link based* features, resulting in a total of 275 features. For these two datasets, the L_1 regularization method solved by the ADMM algorithm is used for feature selection purposes. The best number of features, regardless of belonging to any feature group, will be selected as per the following procedure. We apply a K-folds cross validation method by dividing the training dataset into K subsets. This is done by selecting $K-1$ subsets for training and leaving one set out each round for validation purposes. For each sub training set, we solve a L_1 optimization problem. Finally, the performance on the training datasets and the validation datasets are averaged through all K learning rounds. We then can find the number of features (F_{best}) which on average is related to the best performance on the validation set. In practice, we set $K = 5$ for the feature selection experiment.

At each learning stage of the hierarchical learning system, experimental settings applied for Web spam detection problems and the text categorization problem may differ. For the common procedures, the learning parameters are set as follows. In the unsupervised PMGraphSOM pre-training stage, the radius is selected within $\{5, 10, 20, 30\}$ while the learning rate is tuned within $\{0.2, 0.4, 0.8, 1\}$. Different map sizes have been tried including 48x30, 54x42, 64x40 and 80x70. The AUC evaluation is not derivable in the unsupervised learning algorithm, hence it is expected not to be seen in the PMGraphSOM results. In the HNBD sampling stage, 100 MLPs are trained in parallel, and the final results are merged. The number of hidden units is chosen in $\{7, 9, 13, 16, 20\}$. The learning rate is set to be adaptive. Both PMGraphSOM and MLP training are stopped at 2000 iterations. In B-GNN learning, each GNN has a number of hidden and state units which are tuned within $\{14, 25, 31, 37, 40\}$ and $\{2, 5, 8, 10, 12\}$, respectively. The training of each GNN layer is terminated at 1500 training iterations. Each model is tested in at least 5 runs with different initialization conditions. The best model is selected based on the best training performance.

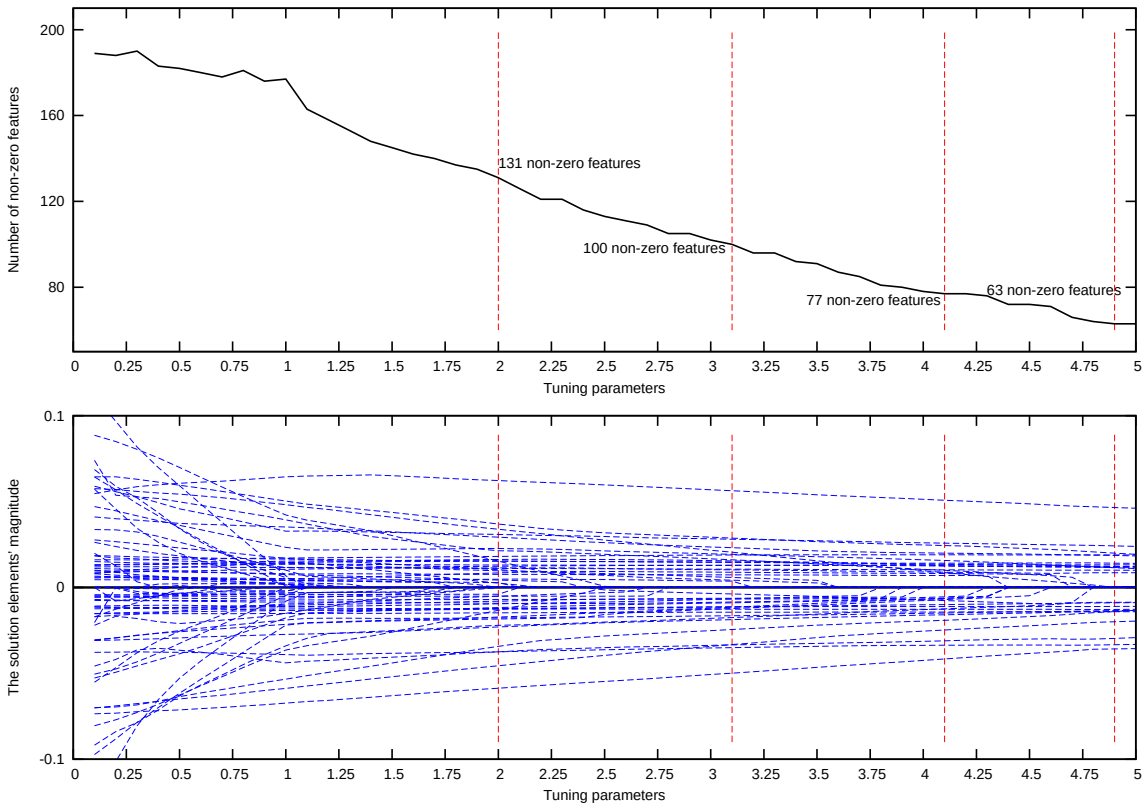

Figure 7.3: L_1 Lasso solution paths based on different tuning parameters λ shown on lower plot and corresponding number of selected features on the upper plot

7.4 Experimental Results

This section is structured in the order of incremental model complexity. The generalization performance of each learning unit and integrated modules will be presented.

7.4.1 Dimensionality reduction

7.4.1.1 L_1 Feature selection for Web spam detection problems

The tuning parameter λ in L_1 regularization controls the number of features being selected. Hence, the first step in the experiment is to determine a suitable value for λ. Figure 7.3 illustrates the application of the L_1 regularization method to the UK2006 dataset. The L_1

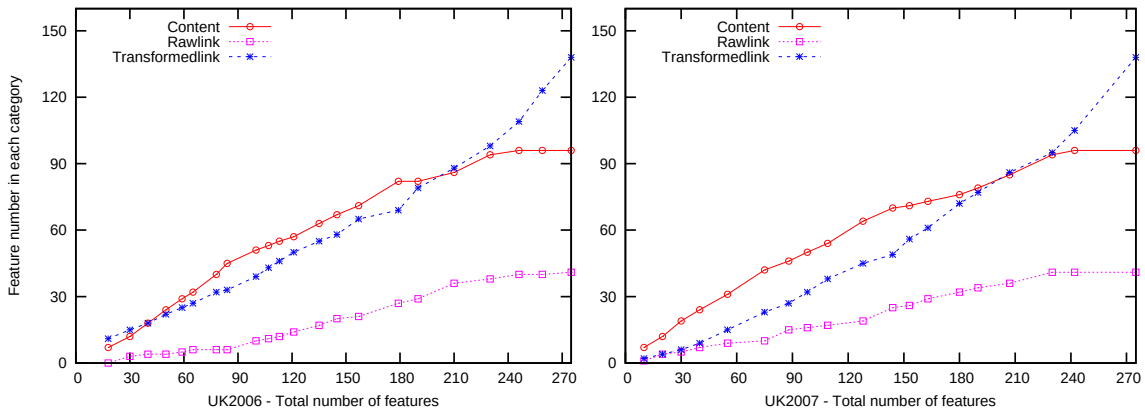

Figure 7.4: The curve showing the number of selected features for the UK2006 and UK2007 using the L_1 feature selection

solution path is shown in the lower part of Figure 7.3. It can be seen that the magnitudes of all solution elements decreases to zero along with increasing size of λ. The number of features at each point corresponds to non-zero solution elements. Shown in the upper part of Figure 7.3 is the number of features and some examples which have 131, 100, 77 and 63 non-zero solution elements. Note that the λ value is bounded in the range [0.1, 5.0] where reasonable results can be obtained.

It is interesting to investigate the effect of L_1 regularization on three individual set of features, content, raw link and transformed link -based feature groups. This is shown in Figure 7.4. The total number of L_1 features is given on the x-axis, while the numbers of features in each of the three groups is associated with the y-axis. It is observed that for both datasets, as the tuning parameter increases, the L_1 regularization removes transformed link and raw link based features first. This implies that the transformed link and raw link based features are less valuable than the content based feature to the learning task. For example, when the total number of features is reduced to 100, about two thirds of the raw link and transformed link -based features are already removed, while up to two thirds of the content-based features are kept. When the total number of remaining features shrinks to 18, there are no raw link features left in the UK2006 case. Thus, it can be stated that content-based

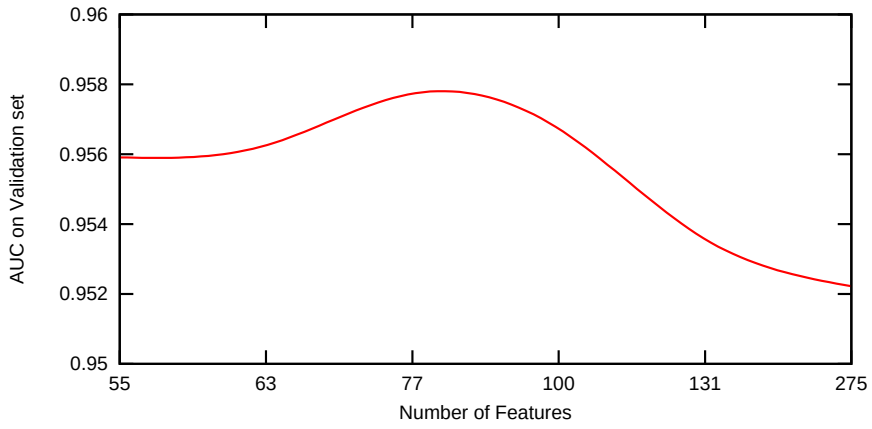

Figure 7.5: Average AUC performances on validation sets with respect to different number of features regarding the UK2006 dataset

information contributes more important features than the other two feature categories.

In order to draw a final best set of features, regardless of any groups, L_1 regularization is applied to the mean of 5-fold cross validation. This was done as follows: Presuming that the number of features is bounded in the range 55 to 275, for each of those numbers, we run a 5-fold cross validation, and the number corresponding to the highest average AUC performance on the validation set will be selected. The AUC performance with respect to different number of features are shown in Figure 7.5 for the UK2006 dataset. Finally, we find the best number of features, that is $F_{best} = 85$ for the UK2006 case and $F_{best} = 98$ for the UK2007 dataset. Within 98 features of the UK2007 dataset, there are 51 content based, 10 raw link based, and 37 transformed link based features. Several feature examples are the number of words in the title of a page, the fraction of visible text (hp), the compression rate of the hp, the top 500 corpus precision (hp) content-based features, just to mention a few.

The experimental results shown in Table 7.1 proves that the application of L_1 regularization method for feature selection is really helpful in the cases of Web spam detection problems. The table presents the AUC training and generalization performance for both datasets. More particularly, different feature combinations have been conducted. It can be observed that the more features involved in learning, the better the AUC that is obtained. It

Table 7.1: MLP's AUC performance on the UK2006 and UK2007 datasets using different feature sets.

Feature sets	**Reference**	**UK2006**			**UK2007**		
		Feat. size	Train	Test	Feat. size	Train	Test
C	[95]	96	0.8797	0.8615	96	0.7398	0.6632
RL	[78]	41	0.9193	0.8174	41	0.6745	0.6192
C+RL	[17]	137	0.9201	0.8755	137	0.7537	0.7428
C+RL+TL	[96]	275	0.9278	0.8719	275	0.7643	0.7356
L_1 reduced feature set		85	0.9450	0.8902	98	0.7890	0.7681

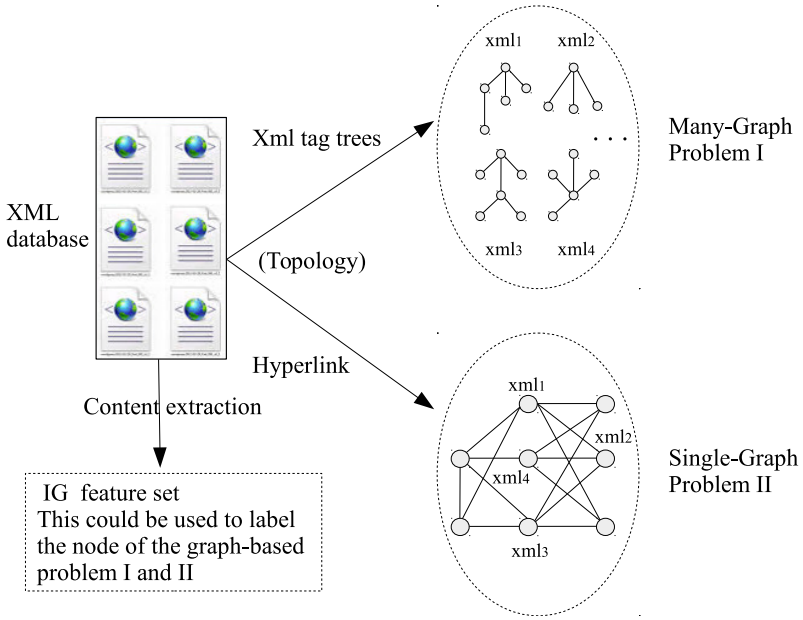

Figure 7.6: Feature and topology extraction in the INEX 2008 XML document categorization problem.

is however not the case when the number of features becomes large, say up to 275, where the MLP may be overfitting and results in poor generalization performance. In general, the reduced feature sets help to improve around 2%-3% for both training and testing results in the UK2006 and UK2007 datasets.

7.4.1.2 Dimensional reduction for INEX 2008 dataset

Since there are not any existing feature sets, a practical method is to obtain a list of important terms for each document based on the IG approach. Due to the large number of terms

available in documents of the INEX 2008 cohort, the following lexing steps are conducted to ensure that a short list of terms would be derived:

(1) Lower case all words.

(2) Toss words from the stop-list (including a, the, and so on).

(3) Remove words containing non-alphabet characters.

(4) Apply a stemming function and remove the last characters (s, e, es) of words.

(5) Remove words containing only digits.

After lexing, the number of terms is reduced from 115,002 to 66,934. The IG of each word is then computed. Also, each term can be represented by a $TF.IDF$ value. We then select 40,000 terms with the highest IG values. By doing so, each document can be represented by a feature vector of 40,000 dimension. A linear SVM filter is then used to reduce the feature space to 15, which is equal to the number of class labels. These resulting features are denoted as content based features (CON), which will be used as the feature vector of each document.

Figure 7.6 gives an overview of the pre-processing stage applied in the INEX2008 problem. In addition to the CON feature set, two types of graph can be derived. The first one is only a single graph that is constructed based on the hyperlinks connecting different xml documents. It is denoted as a hyperlink graph (HYP). The second one is a set of xml tag based graphs. Each document is related to one tag graph (TAG). In order to compose TAG graphs, the xml structure is read, and a xml tag becomes a node in the graph. Finally, we achieve the following for TAG trees:

(1) The number of unique xml tags is 639

(2) The number of TAG trees is 114,366

(3) The total number of node is 4,850,964

(4) The maximum number of nodes on a TAG tree is 1,119

(5) The maximum out-degree is 193

The nodes of TAG trees are attached either with the corresponding tag index (ID) or with the CON feature vector. The HYP graph nodes are only attached with the CON feature vectors. The number of HYP graph nodes is exactly the number of documents (or the number of TAG trees).

(1) The number of nodes is 114,366

(2) The maximum out-degree is 101

For the later experiments, three main types of graph can be created, namely the TAG graph, TAG+CON graph or HYP+CON graph. For MLP training, only the content-base feature set is utilized. However, with PMGraphSOM, all these graph types can be its input.

7.4.2 Unsupervised learning

The output of the PMGraphSOM is represented in the form of 2D coordinates on its projection map. This feature information is brought forward in our hierarchical learning regime in such a way that it is added to the reduced feature set that has been obtained so far. These final set of features is denoted as CO-FEAT.

7.4.2.1 The PMGraphSOM for Web spam detection problems

The input of PMGraphSOM consists of both feature and topological information that allows the PMGraphSOM to be able to condense all data into a contraction form on its activation map. Several experimental results of PMGraphSOM training with different parameters are shown in Table 7.2. This table provides the training results with two evaluation metrics, ACC and F1. It can be observed that the training performance of PMGraphSOM is relatively high with the best at 96.77% and 96.29% in accuracy for the UK2006 and UK2007 datasets, respectively. The F1 performance varies significantly due to the changes in the network's architectures. This is reflective of the selection of the PMGraphSOM parameters significantly influencing the learning outcome for these web spam problems.

Table 7.2: PMGraphSOM training ACC and F1 performance for two web spam detection datasets.

Map	Map size	μ	UK2006		UK2007	
			ACC	F1	ACC	F1
1	64x40	0.01	**0.9766**	**0.8683**	0.9562	0.5125
2	64x40	0.25	0.9691	0.8299	0.9537	0.4410
3	64x40	0.50	0.9518	0.7315	0.9587	0.5528
4	48x30	0.01	0.9504	0.6266	0.9544	0.4972
5	48x30	0.25	0.9492	0.7152	0.9514	0.4327
6	48x30	0.50	0.9180	0.3588	0.9539	0.4802
7	54x42	0.01	0.9276	0.5120	**0.9629**	**0.6205**
8	54x42	0.25	0.9293	0.5275	0.9582	0.5498
9	54x42	0.50	0.9537	0.6776	0.9592	0.5434

7.4.2.2 PMGraphSOM learning results for the INEX 2008 dataset

As shown, the CON features are received from the IG based dimensionality reduction method. The TAG graphs and HYP graph are extracted from the XML tag structure and the hyperlink information, and they are typical examples of many-graph/graph-focused and one-graph/node-focused problems, respectively. This section will show the training of PMGraphSOM on three different datasets, including TAG, TAG+CON and HYP+CON. For many-graph problems, the Compact PMGraphSOM has been tried since it is more suitable for learning very deep tree structures. It is observed that the required training duration is reduced by about 10 times compared with the PMGraphSOM case. The learning speed of Compact PMGraphSOM is similar to that of SOMSD.

Table 7.3 shows the results of PMGraphSOMs for the three datasets. It can be said that the incorporated CON to TAG graph helps to improve the results very little compared with the PMGraphSOM trained on TAG only. The best performance is seen regarding HYP+CON data learning. The results show very obviously the large difference between HYP+CON based learning and the other two based on TAG graphs. This clearly indicates that PMGraphSOM can provide supportive output features for the later learning units in the system.

Table 7.3: The PMGraphSOM's R_{micro} and R_{macro} training results on different augmented datasets: TAG; TAG+CON; HYP+CON

Classs ID	TAG	TAG+CON	HYP+CON
1. United states	0.803	0.11	0.995
2. Reference	0.573	0.811	0.97
3. Sports	0.114	0.625	0.991
4. Social institutions	0.043	0.396	0.905
5. Politics by region	0.057	0.05	0.962
6. Urban geography	0.037	0.085	0.974
7. Human behavior	0.074	0.009	0.956
8. Fiction	0.033	0.021	0.989
9. Nationalities	0.008	0.013	0.991
10. Americas	0.022	0.006	0.968
11. Demographics	0.072	0.03	0.926
12. Tourism	0	0.052	0.973
13. Art genres	0	0.01	0.996
14. Sociology	0	0	0
15. Europe	0	0	0.772
R_{micro}	0.309	0.337	**0.961**
R_{macro}	0.134	0.16	**0.896**

7.4.3 One-staged deep learning PMGraphSOM+MLP

Supervised learning models will be presented from this section. The one-staged hierarchical learning model does not form an actual component in our proposed learning system. It is however, informative in the sense that the unsupervised pre-training step will prove very helpful for the task. Here, a single MLP will be trained on the feature set CO-FEAT which contains both the output of the PMGraphSOM and the L_1 based reduced feature set. In the later sections, the PMGraphSOM is not recalled anyway because its influence is conveyed in its feature output.

The following will compare the MLP performance when trained without or with an unsupervised pre-trained stage. In other words, one can compare the results of the MLP when training with only the L_1 based reduced feature set and training with the CO-FEAT feature set, respectively. Firstly, for the web spam detection problems, it is observed that the generalization AUC results are improved from 0.8902 to 0.9003 for UK2006, and from 0.7681 to 0.7905 for UK2007. It is also seen to improve by 0.005 and 0.018 in the training performance for UK2006 and UK2007, respectively.

Table 7.4: The MLP training with and without the trained PMGraphSOM outputs.

Categories	No pre-trained		Pre-trained	
	Train	Test	Train	Test
R_{micro}	0.9493	0.7859	0.9510	**0.7864**
R_{macro}	0.9385	0.7183	0.9428	**0.7245**
F-measure	0.9432	0.7290	0.9457	**0.7304**

Secondly, Table 7.4 shows the results of training MLP with regard to the INEX 2008 dataset. The MLP with the PMGraphSOM pre-trained can improve both the training and testing performance. In this case, we only used the output of PMGraphSOM training with HYP+CON input, due to the relatively poor performance on the cases of TAG and TAG+CON. Since the learning problem is relatively large, small improvement shown here is due to the fairly large number of documents being correctly classified.

7.4.4 Imbalance data treatment

In this section, HNBD sampling using the base learner MLP is presented. The input of the MLPs is the CO-FEAT feature set. For the first step, input data normalization is applied. We utilize standard 3-layer MLPs for the experiment. The HNBD will be present in detail for the web spam problems. The method is then similarly applied to the INEX 2008 dataset.

7.4.4.1 The HNBD sampling results for Web spam problems

For each probability of success q, Algorithm 1 is implemented once. We conducted 100 MLP experiments for each q. The values of q are randomly selected from the range $q \in [0.15, 0.6]$. An example of NBD in which the number of non-spam hosts can be drawn is shown in Figure 7.7. If the number of spam hosts ($n = 222$) are used in the training set, the number of non-spam hosts m are sampled according to the NBD. As a consequence, the number of non-spam hosts m would likely fall to between 700 and 1100. By varying q values, the size of the majority class will be changed in the sampled training subsets, while all the minority class instances are selected. This mechanism allows us to validate the performance of the base learner to seek the best q for a particular imbalanced problem.

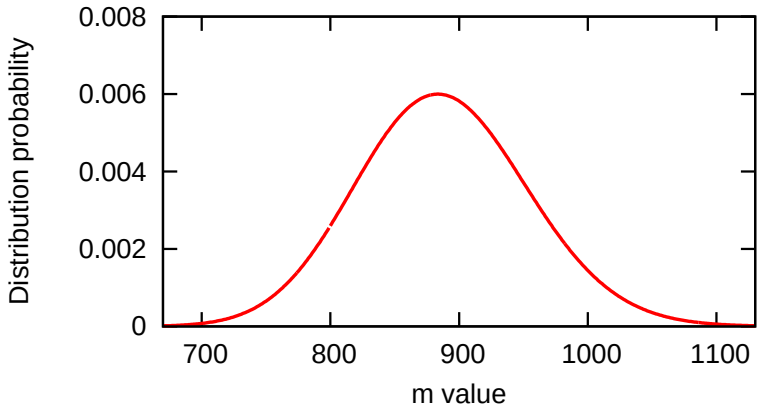

Figure 7.7: Negative Binomial Distribution when n = 222, and q = 0.2 in UK2007. The number of non-spam hosts would be drawn based on value m of the distribution.

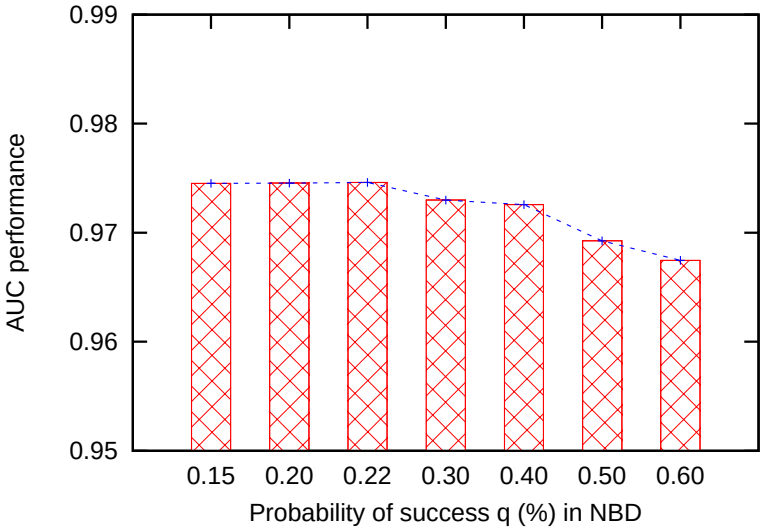

Figure 7.8: AUC training performance while applying HNBD to the UK2006 dataset.

Figure 7.8 and Figure 7.9 present the AUC training performance for the UK2006 and UK2007 datasets, respectively. It is observed that the best AUC performance can be obtained when setting $q = 0.22$ for UK2006 and $q = 0.20$ for the UK2007 dataset. In general, it is found that a variation of the q-probability can lead to significant changes in the system's performance when learning with imbalanced datasets. In fact, the optimal q-probability value is obtained by the proposed automated process rather than a value that needs to be determined by trial-and-error.

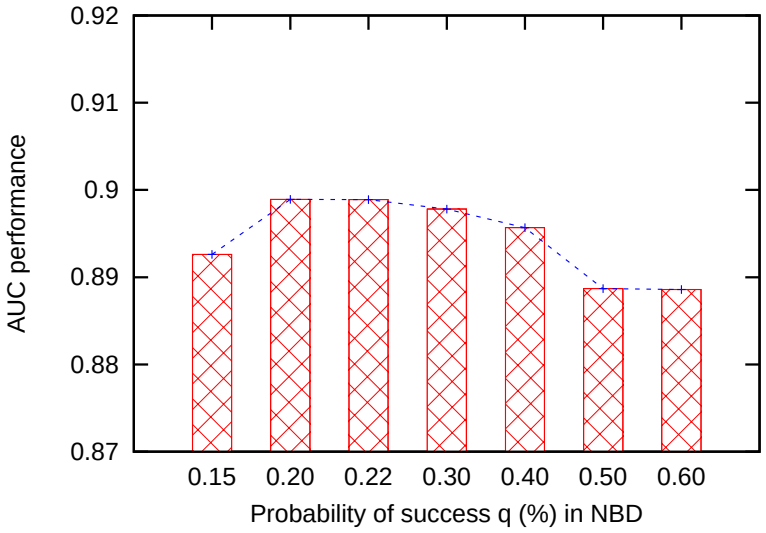

Figure 7.9: AUC training performance while applying HNBD to the UK2007 dataset.

The effectiveness of HNBD is summarized in Table 7.5 for the UK2006 and UK2007 datasets. The method denoted as MLPs in the table refers to the MLPs learning without a sampling method, which forms the comparison base line. For comparisons, we implemented approaches introduced in [136] for the cost-sensitive neural network by setting the weight on the error cost function, the approach in [138] for undersampling, and the approach in [110] for roughly balance bagging. All the MLPs settings for those methods are made the same as for the HNBD. The comparison is fair since those methods all take the MLPs as base learners. It is evident that the proposed HNBD method outperforms all others given both imbalanced datasets, UK2006 and UK2007. The poorest result is associated with the MLPs without embedding any sampling techniques. The cost sensitive method seems not to perform as well as the under-sampling and the RB methods. The RB sampling approach is seen to be most close to our proposed HNBD method. It seems to perform quite well in the case of less imbalanced data like the UK2006 dataset, while it suffers from severely skewed class distribution such as UK2007. The HNBD method achieves a fairly moderate improvement when compared with the others in the case of K2006, while the improvement is more significant for the UK2007 dataset. This is reflective of the fact that the HNBD is

Table 7.5: Advancement of HNBD sampling approach compared with others regarding the AUC performance.

Methods	**Reference**	**UK2006**		**UK2007**	
		Train	Test	Train	Test
MLPs		0.9490	0.9003	0.8167	0.7905
Cost-Sensitive with MLPs	[136]	0.9512	0.9024	0.8320	0.8073
Under-sampling based	[138]	0.9648	0.9196	0.8887	0.8183
Roughly balance bagging (RB)	[110]	0.9692	0.9225	0.8545	0.8215
Proposed HNBD sampling		0.9743	0.9283	0.9006	0.8375

Table 7.6: The MLPs training with and without sampling approach engaged.

Classs ID	**No sampling**		**With sampling**	
	Train	Test	Train	Test
R_{micro}	0.9487	0.7851	0.9511	0.7858
R_{macro}	0.9373	0.7190	0.9432	0.7247
F-measure	0.9424	0.7290	0.9464	0.7313

more robust when dealing with severely imbalanced data.

7.4.4.2 Sampling results for the INEX 2008 problem

Due to the fact that the INEX 2008 dataset is a multiple class problem, application of HNBD is required to adjust to some extent. In particular, we set the NBD parameter p as being fixed at 0.5, and apply HNBD for all the classes' samples. This approach in practice results in an improvement of R_{macro} value by 0.011, however both R_{micro} and F1 performances decline. We then apply a simple approach as follows: Firstly, we train the full training set and achieve result A, then we train the HNBD based sampling training set and obtain result B. Finally, the two results A and B are merged to form the ultimate output. Table 7.6 presents the results of this boot-trap sampling approach. A possible explanation for the improvement is that it may take advantage of the high R_{micro} result on the full training set, and the high R_{macro} on the sampling training set.

7.4.5 Experimental results on the top level of the leaning system

The final experimental results of the proposed learning system are presented in this section. One can denote the model as the three-staged hierarchical learning or in short PMGraph-SOM + HNBD(MLPs) + B-GNN model. In this level, the output of the HNBD(MLPs) together with the graph topology will form the graph-based input for the B-GNN model. Note that the topological information of the INEX 2008 dataset is HYP graph, while host graph is the one for web spam detection problems.

7.4.5.1 For the web spam detection problems

In this section, the B-GNN will be trained with the input taken from the output of the previous layer and the graph topological information. A number of different models trained on the same input as B-GNN are also conducted for comparison purposes. They consist of kernel methods (SVM and GLSVM) and parametric models (MLP and original GNN). The models are either able to incorporate the input topology such as B-GNN, GNN and GLSVM, or are unable to encode relational information like SVM and MLP. By placing different types of learning models on the top of a parametric based deep learning architecture, we are able to examine the integrative suitability of those models in the proposed hierarchical regime. It can also be seen which models will perform best given the same feature and topological input.

For this experiment, we configure the particular setting for kernel based methods' parameters as follows. The kernel used is the Gaussian radial basis function. The best combinations of kernel parameter σ, and soft margin parameter γ are selected by a grid search with exponentially growing sequences of σ and γ. Here, γ is tuned within $\{2^{-3}, 2^{-2}, ...2^{9}\}$ while σ is selected within $\{2^{-11}, 2^{-10}, ...2^{2}\}$. The GLSVM learns relational data by using the host graph to compute the adjacency matrix which will be applied in its learning process. In addition to kernel parameter σ and the ambient space parameter γ_A, another

Table 7.7: Comparison of the five models when trained based on the top level of the proposed learning system.

Models		**AUC**		**F1**		**ACC**	
Name	Topology	Train	Test	Train	Test	Train	Test
Web Spam UK2006 Dataset							
+B-GNN	√	**0.9851**	**0.9685**	0.6391	0.3593	0.9712	0.8843
+GNN	√	0.9720	0.9582	0.8203	0.8939	0.9551	0.8650
+GSVM	√	0.9738	0.9481	0.8385	0.8904	0.9601	0.8603
+MLP		0.9533	0.8770	0.7616	0.7514	0.9436	0.7190
+SVM		0.9745	0.9483	0.8306	0.8841	0.9592	0.8543
Web Spam UK2007 Dataset							
+B-GNN	√	**0.9154**	**0.8572**	0.6395	0.3591	0.9655	0.9360
+GNN	√	0.9097	0.8534	0.6145	0.3553	0.9647	0.9382
+GSVM	√	0.9010	0.8455	0.5740	0.3644	0.9512	0.9251
+MLP		0.7433	0.7344	0.5139	0.3278	0.9569	0.9345
+SVM		0.8393	0.6730	0.4891	0.3372	0.9440	0.9424

variable γ_I of GLSVM decides the influence of structural information. Similar to the SVM, the grid search of three parameters (σ, γ_A, γ_I) is conducted, i.e. $\sigma \in \{2^{-5}, 2^{-4}, , ...2^{1}\}$, $\gamma_A \in \{2^{-3}, 2^{-2}, ...2^{3}\}$ and $\gamma_I \in \{2^{-3}, 2^{-2}, ...2^{3}\}$.

The experimental results are shown in Table 7.7. The "Topology" column indicates whether or not the "on-top" prediction models (illustrated by a + before a model name) utilize relational information in their learning process.

It can be derived from the table that if one wishes to achieve very good AUC performance on a problem featured with imbalanced class nature and topological data structures, then GNN based models would be the most suitable. Nevertheless, the results obtained by GLSVM are competitive, and especially SVM provides an excellent performance on the ACC basic [2] for the UK2007 dataset. The models learning without topological relation are not as robust as the ones taking advantage of relational data. It is also shown that the SVM and GLSVM are not integrated well [3] in the parametrized based hierarchical models.

It is worth mentioning the computational requirements of those models for a closer ex-

[2]The Kernel machine can provide very good accuracy, as the nature of learning a support vector is to absolutely separate input samples into positive or negative class. Therefore, the ACC performance is boosted when trained with a kernel machine approach.

[3]The output of the parametric model is definite values which might better support the AUC performance, while the probabilistic output of kernel methods is not as strong since it may not provide sufficient information.

amination about the aspects of real-time processing advantages. In practice, kernel methods usually hold the initiative in that they can learn a problem within a relatively short duration. In particular, for the case of web spam detection problems, the SVM requires between 10.16 to 15.75 seconds for the training phase, while only 1.63 to 2.33 seconds are needed for testing. Similarly, even though GLSVM incorporates both feature and topology in its learning algorithm, its computation cost is approximately 4 times more than that of SVM model. On the other hand, parametric models usually take more time for the training process. More specifically, training a MLP with the web spam detection problems cost from 420 to 510 seconds (using the single thread implementation. The implementation as a massive parallel system was not available at the time of the experiment. When implemented as the parallel one, the speed up by a factor of 40x can be expected [139]). It is however interesting that the testing computation time is only 0.5 to 2.0 seconds. The highest computational load is for the GNN learning algorithm. Its training process requires around 200 times longer than the MLP, which is about 24 to 48 hours depending on the parameters configured and the status of the computer processor. The testing time requirement for the GNN model is again not very long. From 40 to 70 seconds are needed for calculating the test results of all test set samples in the UK2006 and UK2007 datasets. In other words, for one unknown sample, it requires less than 0.05 second for the prediction time. What is evident here is that the GNN based learning model to some extent delivers better learning performance. It is in fact practical since commonly the model is not required to be re-trained during the prediction stage, and since the testing time for the GNNs model is still reasonable for a real-world learning problem.

If our final experimental results are placed along with other methods that have been published in the literature, our results come at the first place for both UK2006 and UK2007 datasets. In particular, in the case of UK2006 dataset, the best AUC result obtained so far was by Abernethy et al [67, 94] with AUC = 0.963. The authors applied the graph

Table 7.8: The GNN training performance.

Cls	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Sum	R_{micro}
0	**2937**	2	0	1	0	0	0	0	0	0	5	0	0	0	0	**2,945**	**0.9973**
1	2	**1463**	0	2	3	0	2	0	1	0	0	0	1	0	0	**1,474**	**0.9925**
2	1	1	**912**	0	0	1	0	0	0	0	0	0	0	0	0	**9,15**	**0.9967**
3	1	0	2	**857**	5	0	1	0	0	0	0	0	0	0	0	**866**	**0.9896**
4	3	3	0	5	**777**	0	1	0	0	0	0	0	0	0	0	**789**	**0.9848**
5	5	0	3	0	1	**687**	0	0	0	0	0	0	0	0	0	**696**	**0.9871**
6	0	1	0	3	0	0	**674**	1	0	0	0	0	0	0	0	**679**	**0.9926**
7	1	1	0	0	0	0	1	**636**	0	0	0	0	0	0	0	**639**	**0.9953**
8	0	1	0	1	0	0	0	0	**635**	0	0	0	0	0	0	**637**	**0.9969**
9	1	0	0	1	1	0	1	0	0	**588**	0	0	0	0	0	**592**	**0.9932**
10	16	3	0	0	0	2	0	0	1	0	**383**	0	0	0	0	**405**	**0.9457**
11	2	0	0	0	0	3	0	0	0	0	0	**289**	0	0	0	**294**	**0.9830**
12	0	0	0	0	0	0	0	0	1	0	0	0	**263**	0	0	**264**	**0.9962**
13	0	0	0	0	0	0	0	0	0	0	0	0	0	**128**	0	**128**	**1.0000**
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**114**	**114**	**1.0000**

Regularization approach with usage of the C+RL+HG feature, topology and many other features which they computed from the webpage content. Our model provides a little better result for AUC = 0.969 (0.6% improvement). In terms of the UK2007 dataset, the model named linked LDA introduced in [80] gives the result with AUC = 0.854. Our proposed model produces AUC = 0.8572 which is about 0.3% better. All comparisons are suppose that the size of the testing set is the same for all the models. These results confirm that the proposed hierarchical learning system can perform consistently well on the two challenging datasets.

7.4.5.2 For the INEX 2008 document categorization problem

Due to the large scale of INEX 2008 dataset, training a GNN would take almost 2 weeks! We intentionally train a single GNN model at this final stage and then directly compare the result with the other approaches available in the literature. The best final training and the corresponding testing performance are shown in the two following Table 7.8 and Table 7.9. It can be observed that the training result is almost perfect at R_{micro}=0.9918 and F1=0.9919. The results imply that it is difficult to achieve better results for not only the training but the testing set as well.

Table 7.9: The GNN testing performance.

Cls	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Sum	R_{micro}
0	**25529**	365	128	123	108	77	50	29	179	188	108	86	33	9	23	**27,035**	**0.9443**
1	292	**10305**	174	414	505	162	349	455	355	144	53	23	68	64	68	**13,431**	**0.7673**
2	144	122	**7796**	92	19	69	80	37	65	45	19	11	11	7	3	**8,520**	**0.9150**
3	87	621	116	**5147**	545	45	200	206	145	68	28	10	39	65	11	**7,333**	**0.7019**
4	237	817	16	582	**4978**	49	85	24	83	28	38	9	3	9	2	**6,960**	**0.7152**
5	159	355	183	102	62	**4789**	58	33	327	113	176	41	12	4	11	**6,425**	**0.7454**
6	71	631	86	286	91	61	**4491**	245	116	66	16	26	29	29	10	**6,254**	**0.7181**
7	47	512	32	130	20	39	205	**4487**	55	30	10	5	30	15	6	**5,623**	**0.7980**
8	179	630	59	221	79	240	85	92	**3571**	82	55	55	71	7	103	**5,529**	**0.6459**
9	346	219	172	110	62	53	56	46	100	**4211**	36	42	14	2	27	**5,496**	**0.7662**
10	398	166	163	53	35	181	18	16	123	45	**2304**	27	7	4	3	**3,543**	**0.6503**
11	240	39	30	73	14	120	22	24	128	69	10	**1809**	6	0	2	**2,586**	**0.6995**
12	38	249	37	68	15	36	74	87	239	18	13	18	**1366**	8	14	**2,280**	**0.5991**
13	24	208	6	65	23	15	109	25	37	12	5	4	3	**496**	5	**1,037**	**0.4783**
14	4	112	18	17	20	13	5	7	61	3	3	1	12	1	**600**	**877**	**0.6842**

Table 7.9 shows the corresponding generalization performance (in a confusion matrix) for the INEX 2008 dataset. We achieved the final results of R_{micro}=0.7955 and F1=0.7407. Since the size of the testing set is much larger than that of the training set for this data, it was viewed as a non-trivial problem, and is very difficult to improve the R_{micro} significantly. In fact, for each 0.01 increase in the R_{micro} indicator, nearly 1200 additional documents will be correctly classified. The generalization performance here indicates that the larger the classes, the better the R_{micro} performance is likely to be achieved. For example, the largest document class consists of 27,035 documents, which is associated with a R_{micro} performance of 0.9443. The poorest one corresponding to R_{micro} = 0.4783 belongs to the class number 13 with only 1,037 documents. This again emphasizes the fact that the INEX 2008 problem is hard to obtain very good testing performance even though the training performance can be easily enhanced to almost 100% accuracy.

Table 7.10 gives the generalization performance of several learning models applied to the INEX 2008 dataset. The best R_{micro} performance of 0.799 is retrieved by the SLVM learning model which exploits the closed frequent subtrees [140]. It is obvious that the generalization performance is only marginally improved even though this model takes many

Table 7.10: INEX 2008 results.

Methods	Features	References	R_{micro}
SLVM closed frequent subtree	HYP+CON+other	[140]	0.799
Our proposed approach	HYP+CON		0.7955
Entropy based method	CON	[141]	0.7879
SLVM origin	CON	[114]	0.7876
Link frequencies	HYP+CON	[84]	0.7849
Label propagation	HYP+CON+TAG+other	[142]	0.776
Naive Bayes extensions	HYP+CON	[143]	0.6980
Naive Bayes	HYP+CON	[144]	0.6813

additional features (other than the CON+HYP ones) as its input. Our model falls in second place with R_{micro} = 0.796, which is almost 1% better than the third place performance of R_{micro} = 0.788 [141]. Most of the other results have been cited from the INEX text document categorization competition which took place in Australia in 2008 [84].

7.5 Conclusion

This chapter presented a hierarchical learning system in an integrative and comprehensive manner. It has been shown that the proposed model is able to handle three different limitations of a common/graph prediction model, including imbalanced class distribution, high input dimensionality and the remote path dependency problem. We have shown experimental results in a staged based approach. The results are presented step by step from the first to the last learning layer. The comparisons have presented us with a better understanding of the model's capability. It is found that the proposed prediction model is capable of dealing with various difficult real-world applications and in obtaining state-of-the-art learning performance.

The long term dependency has also been addressed in this chapter. In this case, deep learning or hierarchical model was shown to be very effective in overcoming the problems. In the next chapter, the problems of long term dependency will be further analyzed.

Chapter 8

Learning long-term dependency for temporal classification problems

8.1 Introduction

Long term dependency in the graph learning domain has been addressed to some extent in the previous two chapters of this thesis. This chapter will consider different aspects in learning temporal time sequences, the dependency of one piece of data with the other are mainly their correlation in time, i.e., a much more restricted form of dependency than had they been in the graph domain. The question which we seek to answer is: in this restricted form of dependency, are there better ways to handle the dependency than if they been in the more general graph domain. Intuitively the answer would be affirmative, but it is the burden of this chapter to demonstrate the qualitative difference in the performance of the techniques used, had such an effect been taken into account.

Long term dependency issues have been originally observed in learning a parametric model for a temporal sequence [1]. When learning a parametrized model, the gradient of the

[1] Long term dependency issues do not occur in non parametric models, like support vector machines, kernel machines.

backprop error could be vanishingly small in some of the stages close to the input end of the neural network architecture. This is due to the fact that the sigmoid nonlinearity used as activation function of the hidden layer neurons, could wander deep into the nonlinear region and hence its derivatives at such regions become very small. If this is propagated back a number of stages in time in a time-unfolding of the recurrent neural network, then the error could become smaller and smaller. This can cause a non-effective updating of the parameters as the update depends on the derivative of the sigmoid function. Thus, long term dependency is a special feature in learning parametric models using sigmoidal activation functions in the hidden layer of the model. In recent years, alternative activation functions were proposed for such parametric models, viz., rectilinear functions [145] or maxout functions [146]. Essentially, the idea is to synthesize sigmoid functions from linear or the maximum of a number of outputs. Such activation functions do not suffer from long term dependency. However, the deployments of such synthesized models often would involve large number of parameters which consequently would require a large amount of training data (which we do not have in the PA activity data), and thus, in this thesis, we will not be considering this activation function synthesis approach.

In this chapter, we will deploy several machine learning approaches, with particular focus on both recursive and recurrent neural networks [2] and the integration of these, to predict some functions related to physical activity (PA) data in preschool and school aged children. As will be shown later in the chapter, PA type of data is particularly prone to long term dependency issues if one wishes to use a parametric model to model its behaviours. Hence, the PA type data would serve as an excellent example to study the long term dependency of time sequences.

To date, several studies have employed MLPs to predict physical activity type in children aged from 5 years old [86, 147, 148]. Due to age-based behavioral factors, models applied to older children might not generalize well to predict the behaviour of younger children, since

[2]The difference between recursive and recurrent will be explained later in this chapter.

the prediction task might be difficult for younger aged children who are less disciplined than older ones. To the best of our knowledge, machine learning based accelerometry data analysis has not been evaluated on data of very young children such as preschoolers.

This chapter aims to examine and compare the accuracy of various prediction models, some of them integrated ones from components for predicting PA type in children aged between 3 and 15.

The rest of this chapter is organized as follows: The data measurement and feature extraction methods will be presented in Section 8.2 and Section 8.3, respectively. Section 8.4 describes the prediction models and integration approaches. The experimental setting is given in Section 8.5. The illustrations for long term dependency problem are shown in Section 8.6. Section 8.7 and Section 8.8 provide the experimental results for the preschool children data and the SCA data, respectively. Some conclusions are drawn in Section 8.9.

8.2 Data measured

The following measurements were made, or inferred from the primary measurements:

- Metabolic equivalents (MET) is a physiological measure expressing the energy cost of physical activities and is defined as the ratio of metabolic rate (the rate of energy consumption) during a specific physical activity to a reference metabolic rate, set by convention to 3.5 ml O_2 kg^{-1}/min or equivalently:

$$1MET \equiv 1\frac{kcal}{kg * h} \equiv 4.184\frac{kJ}{kg * h}$$

  where $kcal$ is the energy expenditure in 1,000 calorie units, kJ is the energy expenditure in 1,000 joule units, kg is the weight in kilograms, and h is the time spent in doing the physical activity.

- Activity energy expenditure (AEE) in kcal/kg/min.

- Measured time in minutes

The activity is measured by accelerometers attached to various parts of the child's body. Thus, by measuring the activity as indicated by the accelerometer readings, together with the time in which the activities were spent, it is possible to have an indication of the total amount of energy spent in the physical activity, in kcal, by the child.

Two PA data cohorts, involving subjects of different ages, from 3 to 15 years old (preschool children, school and adolescence (SCA)), were assessed in several laboratory based physical activities. The detailed descriptions of these datasets were given in Chapter 4. The school children and adolescence (SCA) dataset is relatively large; it contains 100 participants. The preschool children dataset contains 11 participants.

The activities performed by both cohorts can be divided into five types, namely

- Sedentary,
- Light activities and games,
- Moderate-vigorous activities,
- Walking, and,
- Running.

The SCA dataset is extracted from a single hip mounted accelerometer. The preschool children dataset is collected from three sensors attached at the hip, left wrist and right wrist of the child.

Several raw data time sequences collected from the hip sensor wearable by preschool children are shown in Figure 8.1, in which accelerometry for each activity type. The plots show the G-forces (vertical axis) for each of the three directions X, Y and Z over a period of 240 seconds.

As can be observed, from left to right and from top to bottom, the X, Y and Z values are increasingly varied since the activities involved would require more energy.

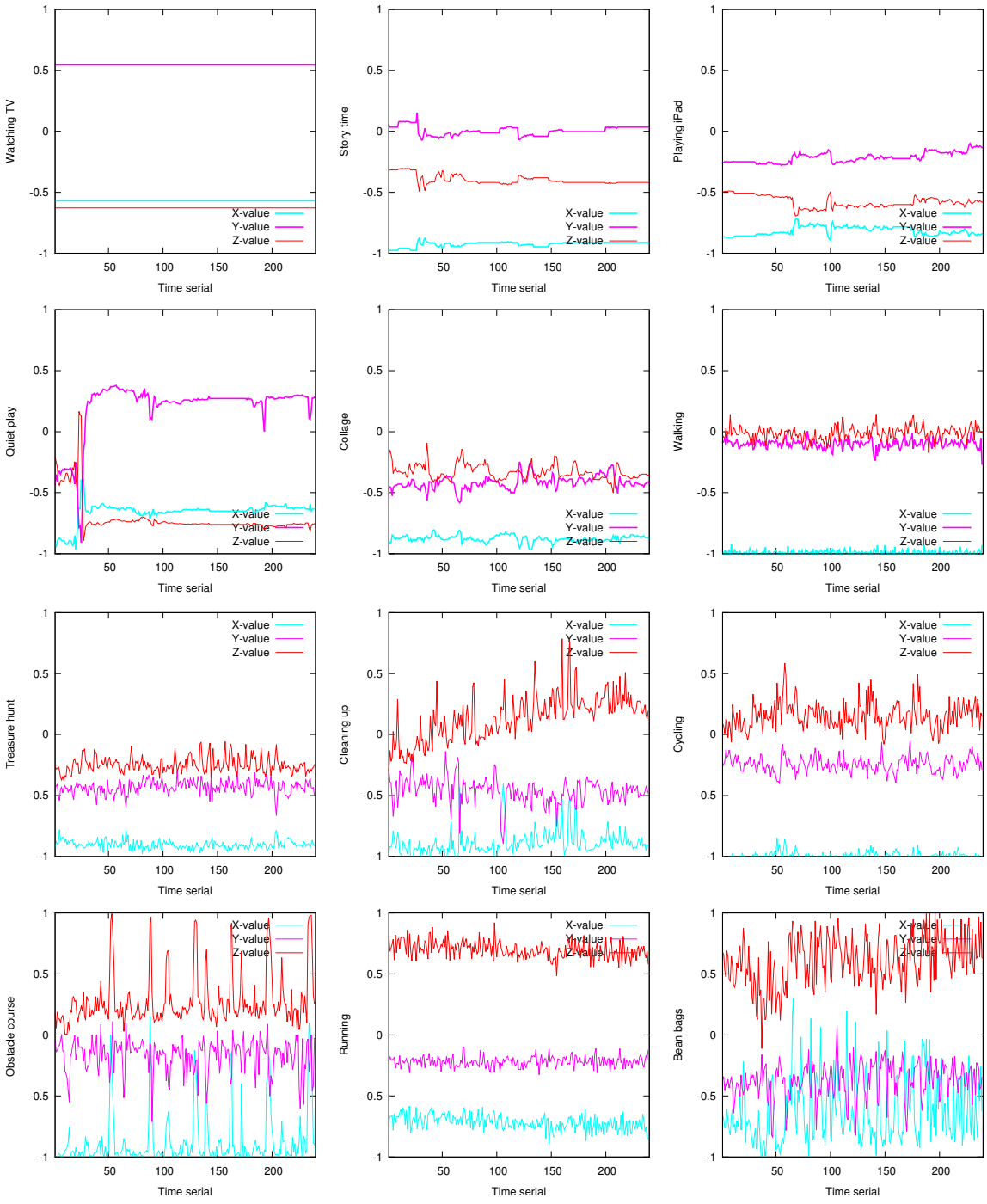

Figure 8.1: The raw time series examples of 12 different physical activities. The horizontal axis is time expressed in seconds.

8.3 Problem description and feature extraction methods

Problem formulation:

Based on the measurements, as given by the accelerometer reading attached to the child's body, as obtained in the datasets, is it possible to predict the activity type of the children?

The prediction of subjects' activity types is likely to be more difficult when the subject's age is < 5. While older aged subjects normally behave in a more disciplined manner in that they would comply with the experimental procedures, the preschool children, however, behave more freely. They are more active and excited with their surroundings. This results in the data collected for preschool children being more noisy.

Feature extraction:

For the preschool children PA data, since each participant performs 12 different types of activities, there are 11 (participants) $\times$ 12 = 132 input sequence samples. For the school and adolescence (SCA) data, there are 100 (participants) $\times$ 12 = 1200 sequence samples. The duration of each activity type is limited to a maximum of 2 minutes; the longest sequence ones is 120 (seconds).

We applied the feature extraction method introduced in [86] and extract two types of features:

- Summary of the distribution of counts features – Consider each minute as a basis. Count the total number of peaks in the time series during this period. One then finds the times when the following percentiles are reached:
  - 10%, t_1
  - 25%, t_2
  - 50%, t_3
  - 75%, t_4
  - 90%, t_5

This gives some idea of the underlying distribution of counts.

- Summary of temporal dynamics. From t_i obtained, $i = 1, 2, 3, 4, 5$, estimate a first order model: $t_i = \alpha t_{i-1} + \epsilon$, where α is a constant, and ϵ is assumed to be Gaussian distributed with $\mathcal{N}(0, \sigma)$, and σ is the unknown variance. In this case, it is quite simple to derive two solutions for α:

  - Case 1:

    $$\alpha = \frac{\sum_{i=2}^{5} t_i^2}{\sum_{i=2}^{5} t_i t_{i-1}}$$

    then, ϵ_i, $i = 2, 3, 4, 5$ can be estimated as follows:

    $$\epsilon_i = t_i - \alpha t_{i-1}$$

    Then $\sum_{i=2}^{5} \epsilon_i \approx 0$, and $\sigma^2 = \sum_{i=2}^{5} \epsilon_i$

  - Case 2:

    $$\alpha = \frac{\sum_{i=2}^{5} t_i t_{i-1}}{\sum_{i=2}^{5} t_i^2}$$

    Then, ϵ_i, $i = 2, 3, 4, 5$ can be computed as follows:

    $$\epsilon_i = t_i - \alpha t_{i-1}$$

    Then, $\sum_{i=2}^{5} \epsilon_i \approx 0$ and $\sigma^2 = \sum_{i=2}^{5} t_i^2$

  For the selection criterion, we choose either case 1 or case 2 dependent on whether $\sum_{i=2}^{5} \epsilon_i$ is closer to 0.

Thus in each minute one would have 6 indicators relating to the distribution and the dynamics of the peaks of the measurements. Note that t_i in the feature set will be expressed in terms of ratio $\frac{t_i}{60}$, and that $0 < \alpha < 1$. Thus there is no need for any scaling of these numbers as they are all in the range $[0, 1]$. Note also that this way of extracting features

will favor those activities with a large number of peaks. For example, in TV watching, it is noted that there is hardly any peak in the time series, and thus the features extracted will be almost all 0. If the time series consists of N minutes, then one would have a feature of $6N$ dimensions. This will give a set of features describing the time series.

Since we are interested in the possibility of predicting the activity types, we will need to use some kind of sliding window to obtain a moving average of these instantaneous features. We divide the time series into non-overlapping intervals of W seconds. Thus, a time series of N data points will consist of $\lfloor \frac{N}{W} \rfloor$ segments, where $\lfloor \cdot \rfloor$ denotes the maximum integer such that $\leqslant \frac{N}{W}$. Then, using W seconds as a window, one could obtain a sliding window version of the features, each time sliding T seconds in the time direction. In our experiments, we choose $T = 10, 30, 60$ seconds. Thus in the 10 seconds case, one will have a 60-dimensional vector, while in the 60 seconds case, one will have a vector of dimension 360. So, say one has a time series of length N seconds, if one has a window of length W seconds, each time it advances by a step of say T seconds, then one has a set of vectors consisting of:

- $\mathbf{V}_1 = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_W]$
- $\mathbf{V}_2 = [\mathbf{v}_{1+T}, \mathbf{v}_{2+T}, \ldots, \mathbf{v}_{W+T}]$
- $\mathbf{V}_3 = [\mathbf{v}_{1+2T}, \mathbf{v}_{2+2T}, \ldots, \mathbf{v}_{W+2T}]$
- $\vdots$

where $\mathbf{V}_i \in \mathcal{R}^{6W}$ and $\mathbf{v}_j \in \mathcal{R}^6$. Note that $\mathbf{V}_i$, are not independent vectors, but correlated vectors. One can denote this set of vectors $\mathcal{V}$. In the situation when there are not enough samples in the last vector of $\mathcal{V}$, this will be padded with 0 so that it will be the same dimension.

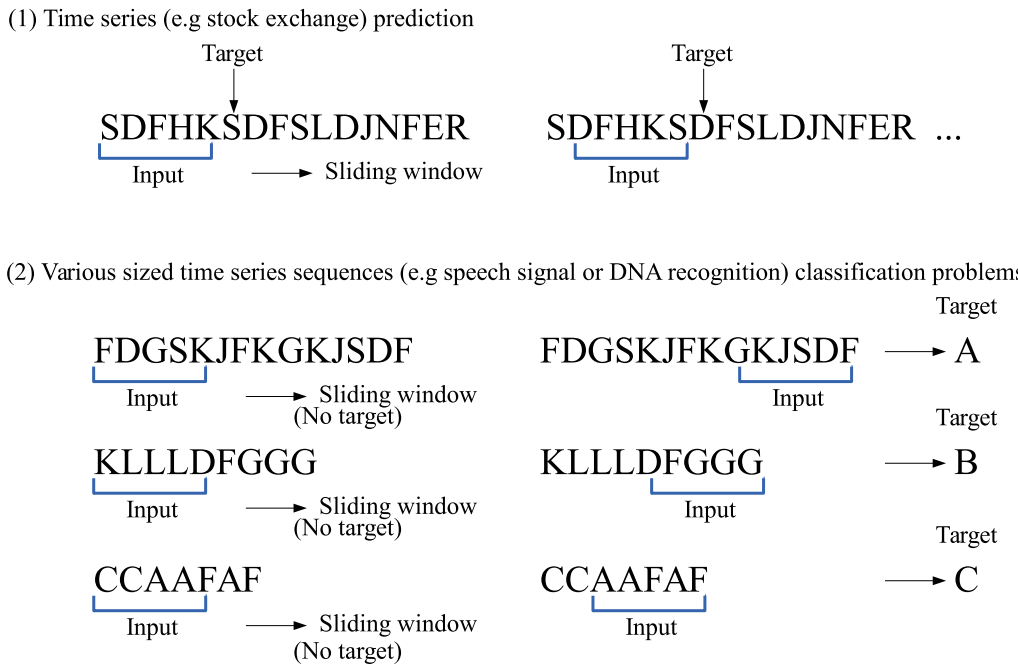

Figure 8.2: (1) The prediction problem, when the next symbol in the sequence is to be predicted given a set of previous sequence. (2) The classification problem: ones have multiple feature vectors formed in each window, where each vector is not necessarily independent, we wish to classify that sequence as belonging to a class label.

8.4 The prediction models

The prediction and classification problems are stated as follows:

Prediction Problem:

Given a set of feature vectors $\mathbf{v}_t$, $t = 1, 2, \ldots, i$, $\mathbf{v} \in \mathcal{R}^6$, can we predict the one step ahead value of $\mathbf{v}_{i+1}$?

Classification problem: Given the sequences $\mathbf{V}_i \in \mathcal{R}^{6W}$, $i = 1, 2, \ldots, N_V$, can one classify the sequence as class C?

Both problems are illustrated in Figure 8.2 using a DNA sequence as an example. This chapter will only consider the classification problem.

Classification of the activity type

Using the features extracted from the time series, it is quite simple to observe that one may train a multilayer perceptron to classify the activity type. In this case, one might have

a training dataset $\mathcal{T}_{train} = \{\mathbf{v}_i, \ell_i\}$, $i = 1, 2, 3, 4, 5$; where i denotes the activity type, and $\mathbf{v}_i \in \mathcal{R}^n$, the n-dimensional feature vector extracted. Then, one can train a multilayer perceptron with, say, one hidden layer, of width, say, N neurons, one can train this model such that the least squared error between the output prediction of the model and the target value of the label to be small, without causing any over-training issues. Then, given a testing dataset $\mathcal{T}_{test} = \{\mathbf{v}_i\}$, without any associated output labels, one could use the trained model to predict the labels associated with the given set of feature vectors.

For modelling the correlated sequences, we consider a number of possible models:

- Elman network or alternatively known as a recurrent neural network
- Recursive multilayer percetron (RMLP) network
- Stable state neural network (SSNN) which combines the recursive and recurrent neural network
- Long short term memory model

All these models have been described in Chapter 3, except the Stable state neural network model, hence it will be explained in the following section. These basic models will form the fundamental building blocks for composing the integrated models.

8.4.1 Stable state neural network

This model is proposed particularly to address temporal sequence classification problems. This model is considered as a special case of the GNN model which was primarily introduced to learn graph based problems [4]. Hence, we only present here the distinct features of the Stable state neural network model.

In learning a temporal sequence, the algorithm can, at each node, take into account both information from the previous steps as well as information in the present step. This model

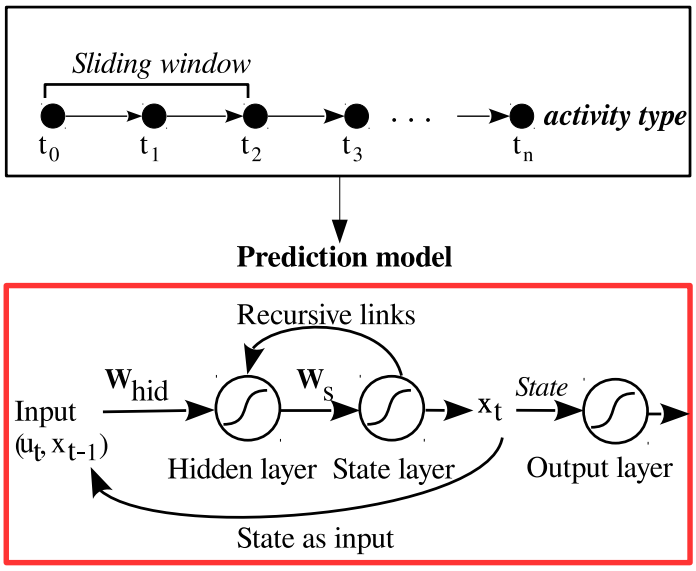

Figure 8.3: The SSNN prediction model

further includes that of the RMLP model which learns only information concurrent at each time step. It also includes the recurrent neural network as a special case.

With respect to Figure 8.3, the SSNN model can be described as follows:

$$\mathbf{z}_t = \mathcal{T}_1\left(\begin{bmatrix} A & B & C \end{bmatrix}\begin{bmatrix} \mathbf{u}_t \\ \mathbf{x}_t \\ \mathbf{x}_{t-1} \end{bmatrix}\right) \tag{8.1}$$

$$\mathbf{x}_t = \mathcal{T}_2(\mathbf{z}_t) \tag{8.2}$$

$$\mathbf{y}_t = \mathcal{T}_3(\mathbf{x}_t), \tag{8.3}$$

where the input $\mathbf{u}_t$ is m-dimensional vector and the output is p-dimensional vector. The internal states $\mathbf{z}_t$ and $\mathbf{x}_t$ are respectively n_1-dimensional and n_2 dimensional vectors. The transformation functions $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ are respectively $n_1 \times (m + 2n_2)$, $n_2 \times n_1$ and $p \times n_2$. These will also include the biases for each of the hidden layer neurons. The hidden layer neurons all have sigmoidal activation functions.

It is noted that in Equation 8.1 if $\mathbf{x}_t$ is missing, then this collapses to the simple recurrent neural network, or alternatively known as the Elman network. On the other hand, in Equation 8.1 if $\mathbf{x}_{t-1}$ is missing, then this will collapse back to an RMLP. Hence, the SSNN model can be observed as a generalization of both the simple recurrent neural network and the RMLP model.

The advantages of having both a recurrent link and a recursive link are that if the time sequence exhibits behaviour which best be described by a recurrent neural network, the SSNN model is capable of handling it. Additionally, if the time sequence exhibits a recursive behaviour, then the SSNN can also handle it. Moreover, it can handle the complex situation where recurrent behaviour interacts with the recursive behaviour of the time sequence.

The states $\mathbf{x}_1$ are guaranteed to be stable if similar to the GNN case, a fixed point theorem is invoked. In practice, this implies that the parameters obtained must be within a certain region, as required by the fixed point theorem. The training of the SSNN follows the usual method. One forms a squared error function at the output end, and then the parameters in the transformation $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ can be updated, using a simple gradient descent algorithm. Similar to the simple recurrent neural network situation, this model could exhibit long term dependency issues.

8.4.2 Composition of models

In this section, we will consider some possible compositions of the basic modules. Moreover, one way of overcoming the long term dependency would be to modify the inputs, and hence here we will consider this in this section as well.

8.4.2.1 Clustering for the pre-training stage

We are provided with a number of time series, e.g., from the acceleromters attached to the subject's body. These provide measurements which might not be decorrelated. A first step

in the preprocessing of the input data can be to cluster, i.e., grouping the measurements together.

The time series input vectors can be mapped into clusters on a two dimensional display space, using a self organizing map (SOM). Once a SOM is trained, then this means that for each input vector/sequence, there will be an associated two dimensional vector (x, y) in the display space. The points on the two dimensional display space may form clusters, i.e., the distance between points within a cluster is smaller than the distance between points in the cluster with those points which are outside it.

Now this way of preprocessing assumes that the data within each window is independent of those in other windows. Such an assumption is not usually valid, and hence the points formed can only be considered to be used as bias, in biasing a solution towards these predisposed locations.

8.4.2.2 The ensemble of SOM for clustering and the SSNN model (SOM+SSNN)

It can make sense to combine the SOM with the SSNN since they have complementary properties. The SOM is different from the SSNN in that its learning algorithm is unsupervised. The SOM model is known to be less sensitive to noise in the data. The SSNN, on the other hand, is supervised, and has much better generalisation ability. Inspired by the concept of deep learning presented in [25], we propose to evaluate the ensemble model SOM+SSNN consisting of a SOM as a first layer, followed by an SSNN as a second layer. Both layers are trained on the same input data. The second layer receives the output of the first layer as an additional input.

The role of SOM as a pre-training module in the ensemble model would have three benefits. The first is that the SOM acts as a filter to reduce the dimensionality of the input feature space to two dimensional feature space. Secondly, the SOM is flexible in setting the mapping size. Mapping results can help to distinguish between samples by diversifying

the data on the map, supporting the SSNN model to recognize potential heterogeneity in the classes and to find an optimal solution. Finally, it is well-known that in the time series learning process, the recurrent network usually forgets the information that occurred far in the past. More technically, the recurrent model because of its parameters, will diminish the effects of information which were far from the current time step, from the stability of the recurrent neural network model property. The output of the SOM is informative to the SSNN model in that it helps to alleviate the problem of learning long sequences by providing the relational information of whole sequence to the input of SSNN. In particular, the SOM output can bring the complete history information of the sequence to the point where a class label is available.

Thus, in this case, we have the input feature vector which is the entire length of the time sequence (say N seconds), and so it will be $6N$ dimensional vector, and it denotes a type of activity by the children. If the time series is less than the maximum we can pad it with 0 so that it will be the same dimension as the maximum length of the time sequences. We can use SOM to train such a set of feature vectors. This will provide some possible clusters in the two dimensional display space. Then, given a time sequence, we can find its associated co-ordinates in the display space. This will be the additional information provided to the augmented inputs of SSNN model, augmented by the additional 2 dimensions. This will bias the SSNN model towards the activity type.

8.4.2.3 The SSNN model with modified input sequences (SSNNin)

The approach is to explore the possibility of feedforward of the inputs. The input sequence to the SSNN model is modified by adding shortcut links from a node to other nodes which appeared previously. Figure 8.4 presents some examples of adding shortcut links to the original input sequence.

One can define a number of channels (or the number of shortcut links) and shortcut steps.

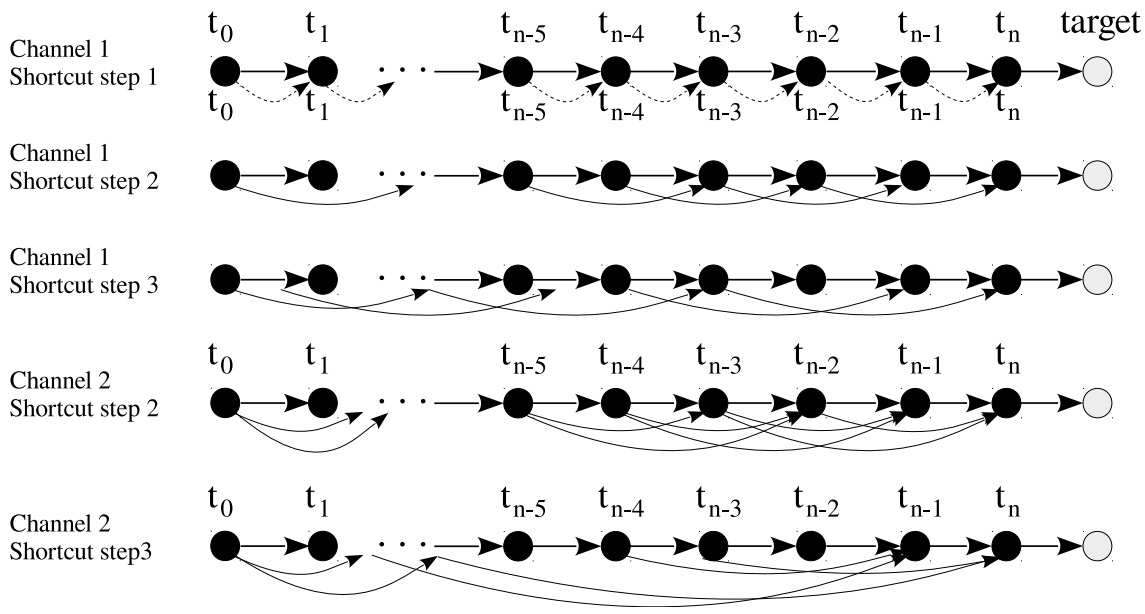

Figure 8.4: The sequence with optional shortcut links

For example, if channel = 1 and step = 1, one obtains the original input sequence. If channel = 1, step = 2, then there will be two inputs: u(t), and u(t-2). If channel = 1, and step =3, then there will be two inputs: u(t), u(t-3). If channel=2, and step = 2, then there will be three inputs, u(t), u(t-1), u(t-2). Note that in this case, the number of steps cannot be less than the number of channels. If channel = 2, one channel at step = 3, and one channel at step = 2, then there will be three inputs: u(t), u(t-2), and u(t-3). In other words, if there is more than one channel, then one must specify the number of delays in the input, which would be available at the current input.

In this manner, the input to the SSNN is augmented, by the direct feedforward of the past inputs to the current input. In general, we will have the following model:

$$\mathbf{x}(t) = f_w\Big(\mathbf{u}(t), \mathbf{u}(t-d_1), \mathbf{u}(t-d_2), \ldots, \mathbf{u}(t-d_p)\Big), \tag{8.4}$$

where are the delays in the i-th channel. $f_w(\cdot)$ is a parametrized function, with weights, which can be determined. The activation function of the hidden layer neurons are assumed

to be sigmoidal functions.

This idea works, because at time t, some of the past inputs are known directly, i.e., not through the network. Thus, if there is long term dependency, this effect is lessened by having direct past inputs to the current input, as such inputs are not modified through the unfolded network architecture. Note that this method does not eliminate long term dependency altogether. It helps to ameleorate the effect of long term dependency by making past inputs available. The number of past inputs and the extent to which it will need to extend to the past will depend on intuition, or experimentation.

8.5 Evaluation metrics and Experimental setting

8.5.1 For classification problems

Three different evaluation metrics are used, including ACC, Recall and F1. Because the model performance is related to different activity types, the ACC evaluation method is considered the most important. Recall and F1 are shown for the purpose of assessing the generalization abilities of the different prediction models, and for the purpose of presenting a rich comparison between models.

Table 8.1 provides an indication of the basic experimental setup the experiment ID, the window size, the step size in each of the experiment.

For preschool children data, the leave-one subject-out cross validation approach is applied. The model is trained on all the input sequences except for one subject's data being left out for the testing set. The learning parameters are tuned based on the training performance. Prediction results are calculated by averaging over all leave-one- out trials. On the other hand, for the SCA data, non-overlapping splits of the whole dataset into three equal sets, namely training, validation and test sets are conducted. The model is trained on the training set, and the learning parameters tuned via the performance on the validation set. For

Table 8.1: The experiment IDs and the corresponding input frame sizes and sliding steps for recurrent NNs.

Experiment ID	Preschool children data		SCA dataset	
	Frame size	Step	Frame size	Step
1	9	9	5	5
2	18	18	5	1
3	18	9	10	10
4	27	27	10	5
5	27	18	10	1
6	27	9	20	10
7	36	27	20	5
8	36	18	20	1
9	36	9	40	10
10	45	27	40	5
11	45	18	40	1
12	45	9	55	5
13	54	27	55	1
14	54	18		
15	54	9		

both datasets, each experiment is run 5 times, and the averages and corresponding standard deviations over 5 runs reported.

In general, the following configurations are used: The learning rate and radius of SOM is selected within $\{0.6, 0.8, 1.0, 1.2\}$ and $\{12, 15, 20, 25\}$, respectively. The SOM map sizes are tried within {19x17, 20x19, 23x20, 25x22} for the preschool children data, and within {53x47, 58x54, 61x58, 63x59} for the SCA data. For both datasets, the number of hidden neurons in the MLP, Elman recurrent network, RMLP and SSNN are tuned within $\{3, 8, 13, 17, 25\}$. The number of state neurons in the state layers of the RMLP and SSNN models is tried within $\{5, 7, 10, 17\}$. For the LSTM model, the number of memory blocks and the number of cells in each block are selected within $\{5, 10, 13, 15, 25\}$ and within $\{1, 2, 3\}$, respectively. The training process of SOM and MLP is both terminated after 10,000 iterations. For all other models, the training duration was chosen to be 5000 epochs, since the convergence was observed well within these number of epochs. The adaptive learning rate is applied for supervised learning models. All the input features are normalized. The experimental results obtained by the traditional MLP will form the baseline for comparison purposes.

Table 8.1 lists the index of experiment (IDs) associated with the frame sizes and sliding steps set for the creation of the input of the recurrent network (the frame size is equivalent to the size of network input, and the sliding steps decide where the next input frame is located). In the following, Exp.ID will be used to indicate the frame as well as step setting here.

8.5.2 For Regression problems

For experimental evaluation, three different evaluation metrics are used, there being root mean square error (RMSE), absolute mean bias (AMB) and mean bias (MB). In particular, RMSE and AMB are used in the prediction of AEE, while RMSE and MB are the evaluation methods for predicting MET values. AMB and MB are fairly similar, although AMB seems more indicative than MB. Using MB instead of AMB is for the purpose of comparing with the results of related studies.

Only the preschool children data contains AEE and MET measurements, hence the experimental results regarding the SCA data will not be available. All experimental settings are set to be the same as in the classification task. In regression learning, we take advantage by using the best selected parameters in the classification task. Additionally, SOMSD is used here instead of the traditional SOM, since the former is capable of encoding contextual information of input data in its learning process, which might be useful for temporal sequence learning. The learning rate and radius of SOMSD are selected from $\{0.6, 0.8, 1.0, 1.2\}$ and from $\{15, 20, 25, 30\}$ respectively. The SOMSD map sizes are tried within {65x56, 77x60, 80x68, 90x70}. In addition, μ values are tried within $\{0.1, 0.3, 0.5, 0.7, 0.9\}$.

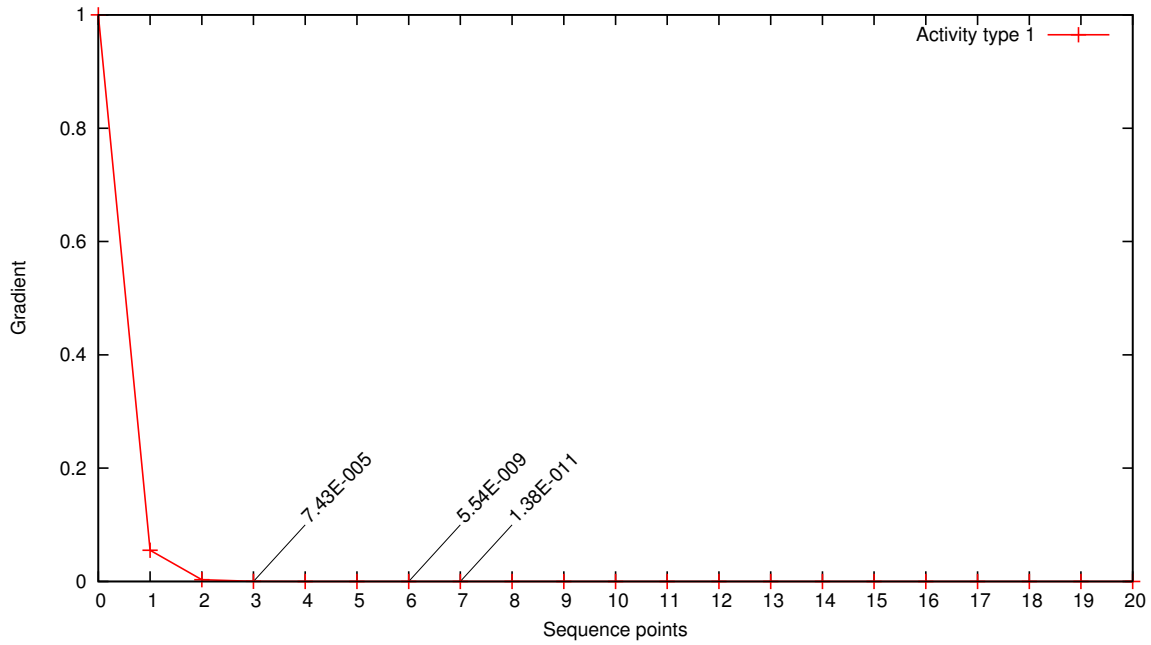

Figure 8.5: The normalized gradient during the back propagation process for the sequence of sedentary.

8.6 The illustrations for long term dependency issue

8.6.1 Gradient in learning long sequences

Using the SCA data for this experiment, we apply 10s window for feature extraction and then apply frame size 3 and sliding step 3 for the input of recurrent network. The following demonstrations are taken at the end of the training process. All the network parameters are set to be the same as in the experimental procedure section. The model used for the experiment is SSNN.

We consider 5 sequences of different types of physical activities. Figure 8.5 displays the normalized gradient during the backpropagation process when learning on the sequence of sedentary activity. Similarly, Figure 8.6 to Figure 8.9 are for the sequences of light activities and games, moderate-vigorous activities, walking and running, respectively.

From these figures, it can empirically be inferred that long term dependency appears

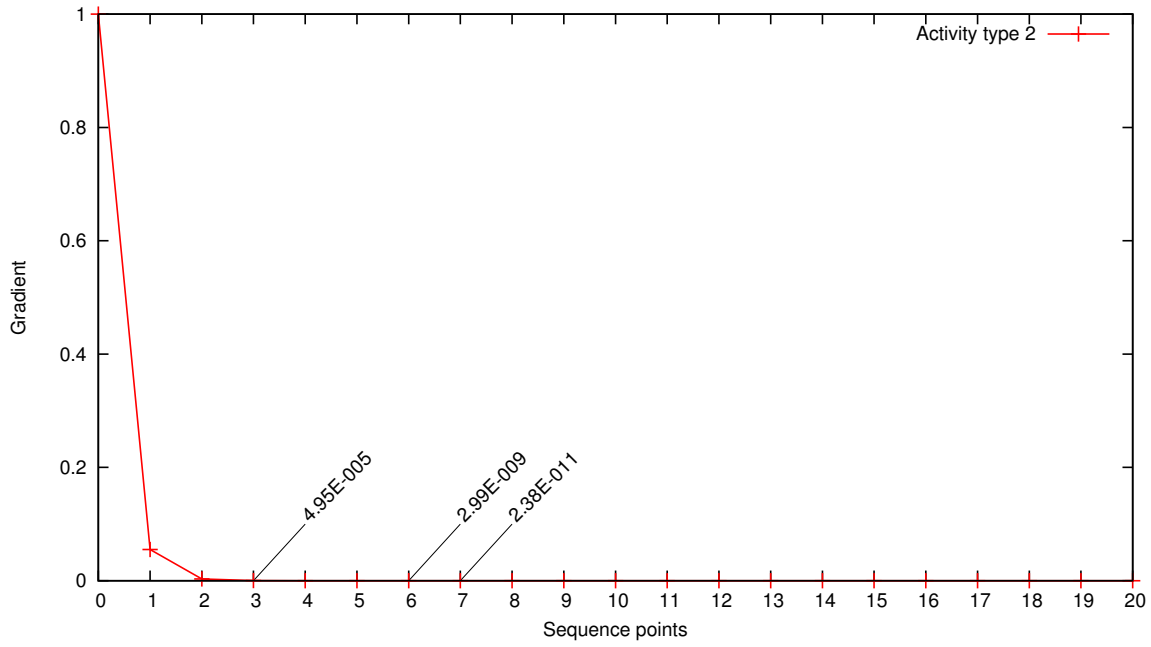

Figure 8.6: The normalized gradient during the back propagation process for the sequence of light activities and games.

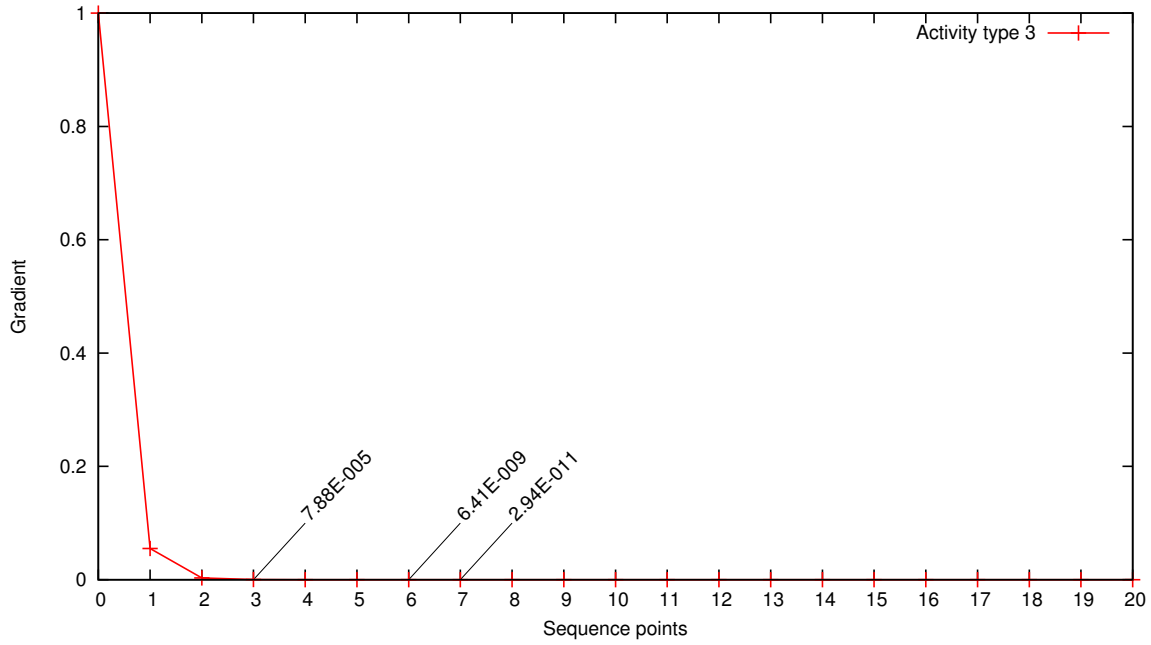

Figure 8.7: The normalized gradient during the back propagation process for the sequence of moderate-vigorous activities.

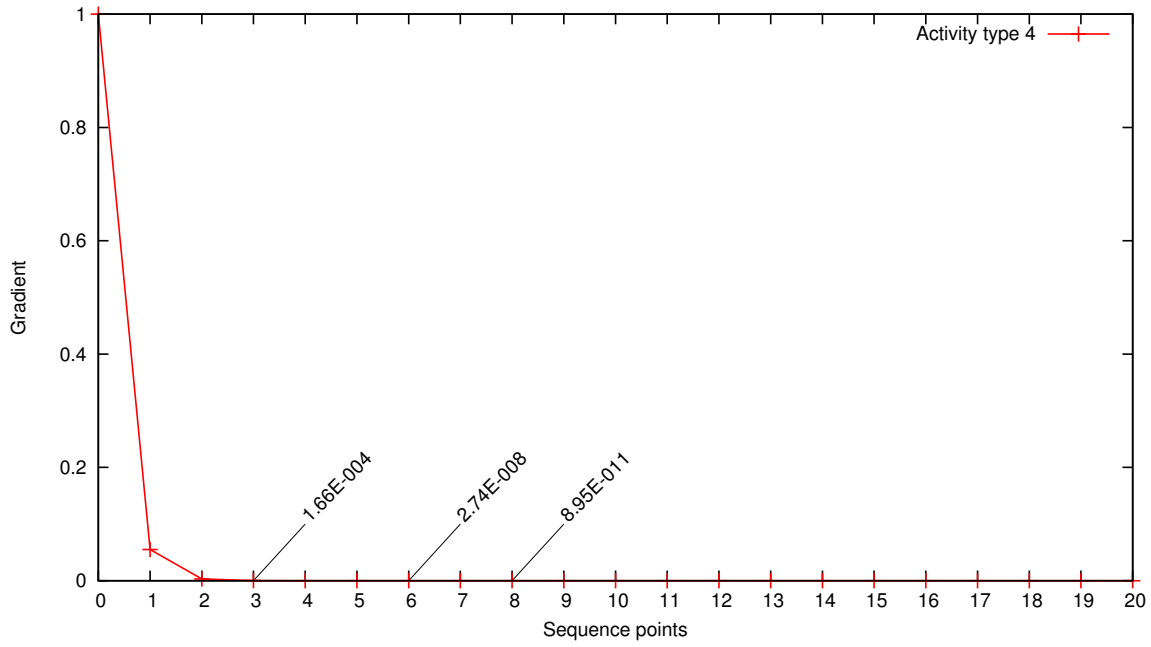

Figure 8.8: The normalized gradient during the back propagation process for the sequence of walking activity.

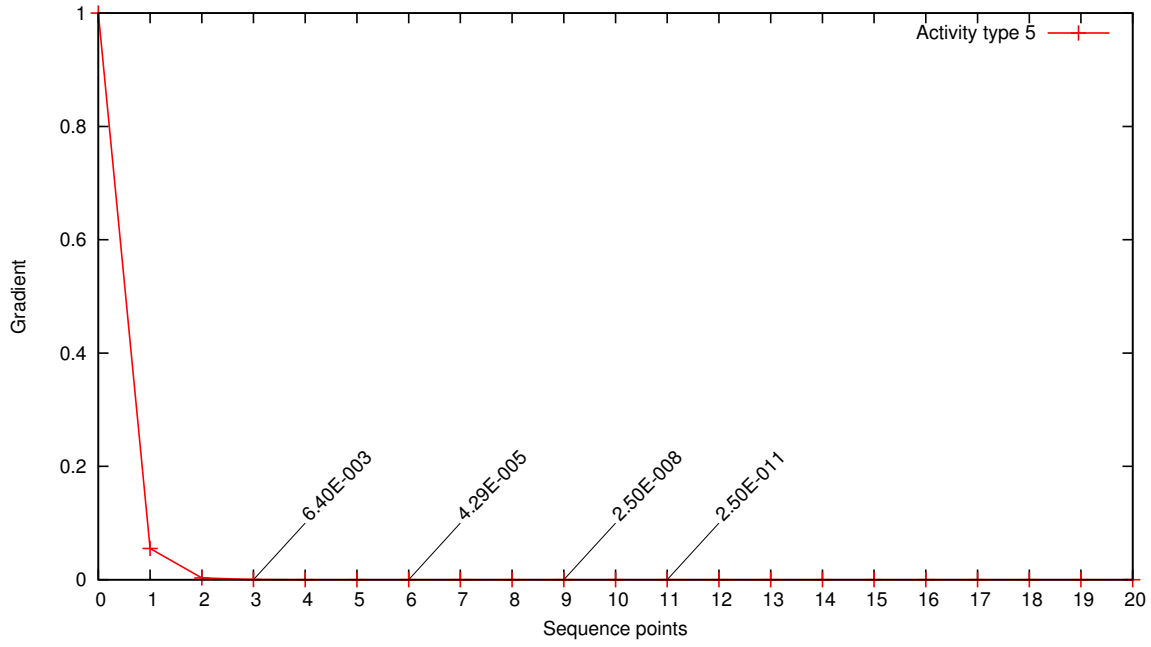

Figure 8.9: The normalized gradient during the back propagation process for the sequence of running activity.

clearly when learning long sequences. It shows that the gradient rapidly decreases and becomes extremely small when the network is in the backpropagation process. If one takes into account the constant 10E-11, then the gradient will be smaller than that value after processing layer 7 to 11 of the sequence, depending on the type of sequence being processed. In particular, the gradients regarding more active activities (i.e. walking and running) decrease slower than that of sedentary activities. The reason for this is that the input sequences of the sedentary class contain more zero values than the other classes. The significant reduction of gradient results in less effectiveness in learning a long sequence, especially when the important information of the sequence lies deep within the sequence. We conclude that the current learning problem is affected by the long term dependency issue.

8.6.2 A solution to long term dependency problem

So far, the experiment sequences have no additional shortcut links. Figure 8.11 displays the normalized gradients when learning on the sequence with shortcut links added. In this case, we take the running sequence for demonstration purposes. Figure 8.10 illustrates a sequence with 21 time step data points. Here we take into account a single particular node t_{20}, the channel here being the number of additional shortcut link (sl). All the other nodes have a similar property, which is intentional though not shown in this figure for clarity. When the channel is expanded, more shortcut links to the distant history nodes are added. Because we investigate the effect of additional links on the problem of long term dependency, we focus on changes in the channel numbers while keeping the shortcut step fixed at 2. Figure 8.11 shows the gradients when learning a sequence which is applied with different channel numbers (i.e channel = 2,4,6,9) compared with the original sequence. It is seen that adding shortcut links is effective in addressing the long term dependency problem. The gradients in the case of applying the input modification method do not vanishingly decline, but decrease gradually. Hence the deep points in the sequence are all taken into account in the learning

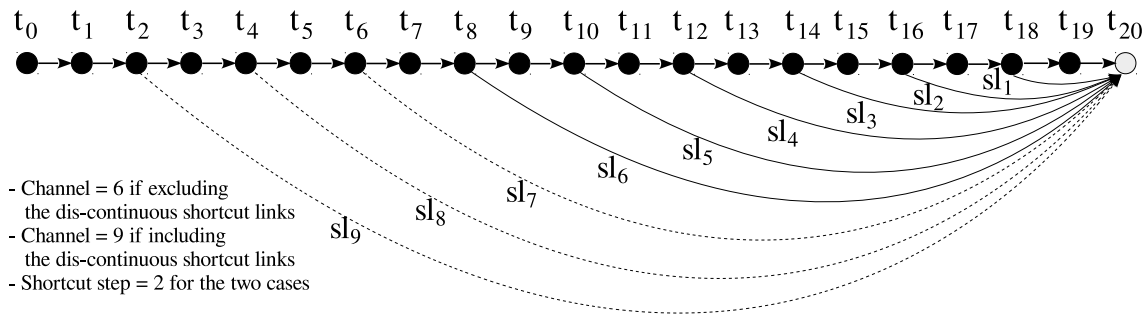

Figure 8.10: Adding shortcut links to original sequence.

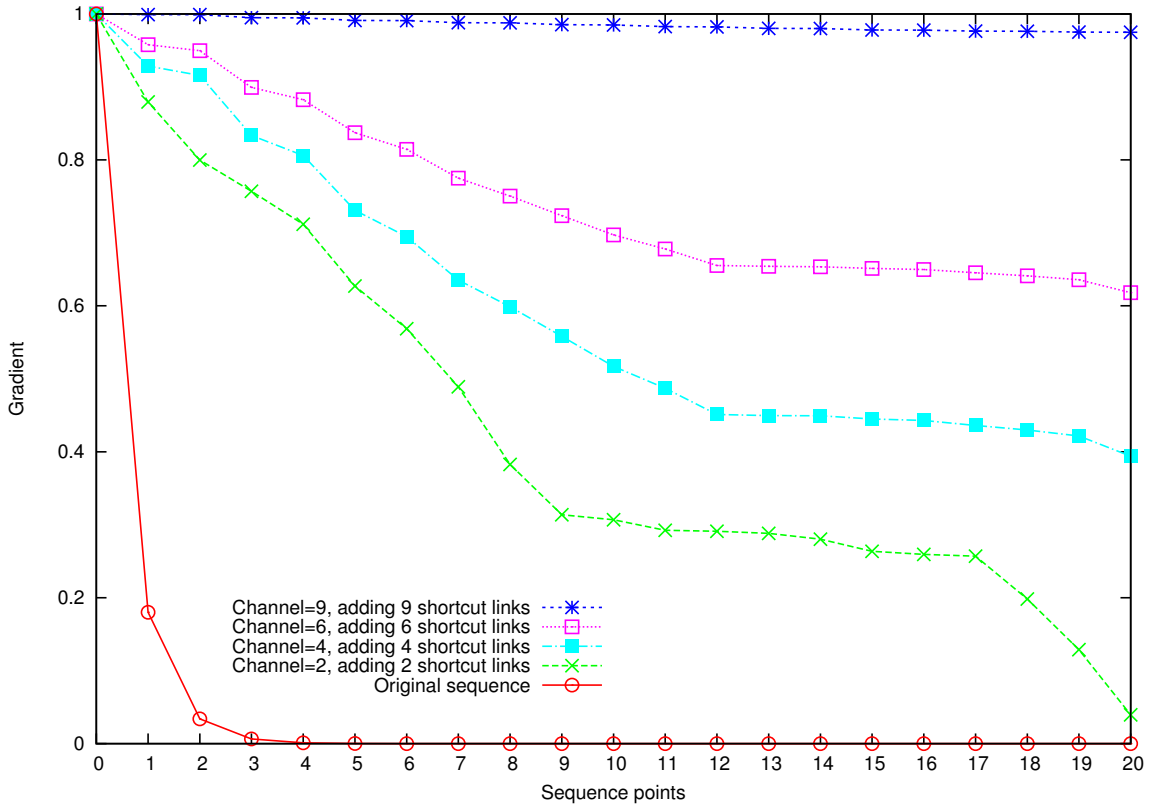

Figure 8.11: The normalized gradient during the back propagation for sequences with different channel numbers.

process. However, it is also shown that the problem of selecting the best number of channels is left for the trail-and-error in the later experimental task.

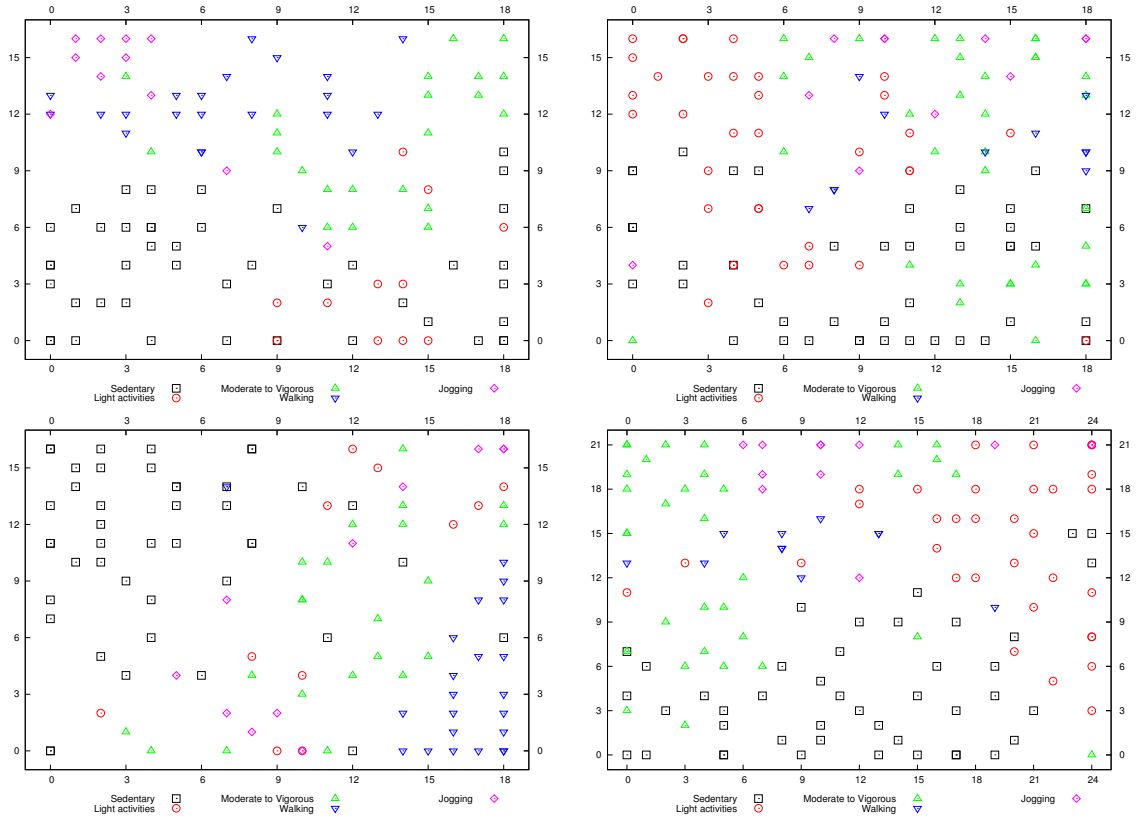

Figure 8.12: SOM activation maps for left wrist (top left), right wrist (top right), hip (bottom left) with same map size 19x17, and three data combination (bottom right) with map size 25x22.

8.7 Experimental results for preschool children data

This section will present prediction results of several neural networks for the preschool children data. The results are shown along with an increment of model complexity. For each table, the results in bold indicate the best performance.

8.7.1 The self organizing map

In this section, the SOM model is implemented. We aim first to show the projection of activity samples on the activation map, and secondly to analyze the possibility of incorporating the SOM model into a layer-wise learning regime. Three sensors' data are separately and interactively learned. Corresponding projection maps are shown in Figure 8.12. It can be

Table 8.2: Performance of MLP on preschool children data.

Data selection	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
Hip data						
10s data	0.882 [0.002]	0.848 [0.005]	0.853 [0.005]	0.630 [0.023]	0.586 [0.027]	0.544 [0.023]
30s data	0.889 [0.003]	0.858 [0.003]	0.861 [0.004]	0.647 [0.026]	0.601 [0.032]	0.564 [0.026]
60s data	0.898 [0.004]	0.871 [0.004]	0.874 [0.004]	0.685 [0.014]	0.634 [0.014]	0.598 [0.018]
Left wrist data						
10s data	0.931 [0.002]	0.896 [0.003]	0.912 [0.004]	0.716 [0.013]	0.682 [0.024]	0.640 [0.028]
30s data	0.937 [0.004]	0.907 [0.006]	0.922 [0.004]	0.722 [0.012]	0.716 [0.009]	0.677 [0.010]
60s data	0.940 [0.004]	0.914 [0.008]	0.928 [0.007]	0.724 [0.009]	0.695 [0.022]	0.655 [0.025]
Right wrist data						
10s data	0.916 [0.004]	0.878 [0.006]	0.885 [0.005]	0.636 [0.016]	0.571 [0.013]	0.530 [0.016]
30s data	0.938 [0.002]	0.902 [0.004]	0.911 [0.004]	0.674 [0.005]	0.594 [0.022]	0.552 [0.022]
60s data	0.952 [0.003]	0.926 [0.006]	0.933 [0.005]	0.683 [0.016]	0.618 [0.032]	0.582 [0.030]
3 data combination						
10s data	0.964 [0.002]	0.944 [0.003]	0.954 [0.003]	0.705 [0.027]	0.609 [0.026]	0.579 [0.026]
30s data	0.965 [0.002]	0.948 [0.003]	0.956 [0.003]	0.722 [0.023]	0.656 [0.029]	0.628 [0.033]
60s data	**0.966 [0.001]**	0.951 [0.003]	0.955 [0.002]	**0.737 [0.009]**	0.683 [0.026]	0.642 [0.028]

observed that the more overlap and confusion between activity samples, the more difficult the prediction task will be. Concerning the projection, the mappings of left wrist data and the combination data could be seen to be more obviously clustered. In the other two cases of right wrist and hip data however, the mappings of samples are inter-weaved. There are some overlaps between sedentary, light activities and games and moderate-to-vigorous activities. The right wrist data seems to contain a certain level of noise. It practically reflects the fact that most participated children move their right hands more frequently while performing actions. Note that, for these little children, the jogging term is equivalent to running in older children.

8.7.2 The multi-layer perceptron

The traditional MLP model is implemented in this section. The MLP can be configured with multiple hidden layers. The standard model normally contains only one hidden layer. Different data extraction based on 10s, 30s and 60s windows are applied. The results of MLP training on these data are indicative to see which feature extraction approach is the best. In fact, smaller windows for feature extraction mean that shorter bit of activities will be validated. The disadvantage of a small window base is that it may capture more noise

Table 8.3: Performance of Elman network on preschool children data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.895 [0.012]	0.878 [0.017]	0.884 [0.015]	0.693 [0.025]	0.611 [0.034]	0.579 [0.037]
2	0.916 [0.001]	0.915 [0.002]	0.915 [0.002]	0.705 [0.014]	0.635 [0.022]	0.604 [0.025]
3	0.894 [0.040]	0.892 [0.039]	0.893 [0.040]	0.679 [0.037]	0.606 [0.056]	0.574 [0.058]
4	**0.917 [0.001]**	0.916 [0.001]	0.916 [0.001]	**0.743 [0.025]**	0.686 [0.031]	0.649 [0.030]
5	0.915 [0.001]	0.912 [0.002]	0.913 [0.002]	0.717 [0.028]	0.663 [0.029]	0.631 [0.033]
6	0.912 [0.005]	0.906 [0.011]	0.908 [0.009]	0.704 [0.011]	0.633 [0.021]	0.598 [0.020]
7	0.915 [0.001]	0.912 [0.003]	0.913 [0.003]	0.692 [0.017]	0.637 [0.038]	0.603 [0.045]
8	0.915 [0.001]	0.914 [0.002]	0.915 [0.002]	0.731 [0.010]	0.665 [0.020]	0.631 [0.020]
9	0.913 [0.002]	0.910 [0.005]	0.911 [0.005]	0.682 [0.019]	0.604 [0.037]	0.573 [0.038]
10	0.916 [0.000]	0.916 [0.001]	0.916 [0.001]	0.718 [0.020]	0.668 [0.039]	0.638 [0.042]
11	0.916 [0.000]	0.916 [0.001]	0.916 [0.001]	0.713 [0.028]	0.648 [0.027]	0.614 [0.029]
12	0.916 [0.001]	0.915 [0.001]	0.915 [0.001]	0.703 [0.029]	0.616 [0.038]	0.583 [0.039]
13	**0.917 [0.000]**	0.916 [0.001]	0.916 [0.001]	**0.717 [0.027]**	0.642 [0.039]	0.609 [0.039]
14	0.916 [0.001]	0.915 [0.002]	0.916 [0.001]	0.722 [0.020]	0.663 [0.033]	0.633 [0.034]
15	0.915 [0.001]	0.914 [0.002]	0.915 [0.002]	0.696 [0.019]	0.641 [0.036]	0.608 [0.037]

for a particular action. On the other hand, large windows for feature extraction can capture overall trends and the direction of activities.

As can be seen from Table 8.2, while the right wrist data is provided with the best training accuracy compared with two other sensors' data, the best generalization performance is related to the left wrist data. Overall, the combination of three sensor data provides the best training and generalization performance. More interestingly, the 60s data is always associated with better prediction results than the 10s and 30s cases. In the later parts, we will use 60s data for different recurrent NNs.

8.7.3 The Elman recurrent neural network and the RMLP learning

Two recurrent neural networks will be deployed in this section. While the Elman network was known as the first recurrent model capable of learning time series, the RMLP was introduced originally for tree structure learning. In this problem, an input sequence is viewed as a simple tree in which the root node is located at the end (or can be created via a super node which connects to all the nodes on the sequence), and the leaf node is located at the beginning of the sequence.

Table 8.4: Performance of RMLP on preschool children data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.878 [0.010]	0.843 [0.016]	0.852 [0.017]	0.729 [0.007]	0.621 [0.022]	0.589 [0.021]
2	0.893 [0.038]	0.887 [0.037]	0.889 [0.038]	0.711 [0.031]	0.639 [0.033]	0.614 [0.038]
3	0.878 [0.041]	0.856 [0.045]	0.861 [0.044]	0.671 [0.034]	0.574 [0.037]	0.539 [0.032]
4	**0.916 [0.000]**	0.916 [0.000]	0.915 [0.000]	**0.757 [0.006]**	0.689 [0.004]	0.665 [0.003]
5	0.912 [0.000]	0.911 [0.003]	0.912 [0.003]	0.717 [0.003]	0.631 [0.000]	0.605 [0.005]
6	0.899 [0.003]	0.882 [0.008]	0.887 [0.006]	0.686 [0.012]	0.578 [0.038]	0.536 [0.042]
7	**0.916 [0.000]**	0.915 [0.000]	0.915 [0.000]	**0.736 [0.000]**	0.696 [0.000]	0.682 [0.000]
8	0.914 [0.000]	0.911 [0.000]	0.911 [0.000]	0.701 [0.000]	0.597 [0.000]	0.567 [0.000]
9	0.889 [0.037]	0.878 [0.037]	0.882 [0.037]	0.696 [0.033]	0.610 [0.025]	0.579 [0.026]
10	0.899 [0.037]	0.899 [0.036]	0.899 [0.037]	0.710 [0.027]	0.638 [0.029]	0.608 [0.027]
11	0.899 [0.037]	0.898 [0.036]	0.899 [0.037]	0.697 [0.035]	0.621 [0.047]	0.594 [0.043]
12	0.898 [0.037]	0.896 [0.038]	0.897 [0.038]	0.675 [0.022]	0.611 [0.039]	0.572 [0.037]
13	0.899 [0.037]	0.898 [0.037]	0.898 [0.037]	0.711 [0.023]	0.645 [0.035]	0.615 [0.034]
14	0.899 [0.037]	0.897 [0.036]	0.898 [0.036]	0.706 [0.035]	0.628 [0.041]	0.599 [0.038]
15	0.896 [0.038]	0.893 [0.038]	0.894 [0.038]	0.704 [0.040]	0.630 [0.054]	0.601 [0.053]

Table 8.3 and Table 8.4 present the prediction results of Elman and RMLP, respectively. As can be observed, the two models' performance is quite similar, although the RMLP testing results are a little better. It is found that RMLP is able to classify roughly an equal number of samples in each class, while the Elman network is more biased on one class or the other. In particular, the F1 indicator in the testing performance of RMLP is almost 4% better than that of the Elman network. An interesting aspect is that one would find it difficult to decide which sequence length (or a selection of window and step size) would bring about the best network performance without an empirical trial and error procedure.

8.7.4 The long short term memory

One of the most powerful recurrent NN models, the LSTM, is implemented in this section. This model is well known in the context of long time lag time series problems. It was proven the long term dependency can be addressed very effectively by this model [26]. The input sequence setting used for LSTM is maintained to be the same as for the RMLP and Elman network. The experimental results of LSTM are given in Table 8.5. It is observed that the LSTM provides fairly stable generalization ability since the oscillation in testing

Table 8.5: Performance of LSTM preschool children data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.803 [0.026]	0.744 [0.040]	0.750 [0.044]	0.765 [0.021]	0.688 [0.037]	0.650 [0.042]
2	0.887 [0.025]	0.833 [0.035]	0.839 [0.034]	0.820 [0.020]	0.756 [0.027]	0.723 [0.026]
3	0.891 [0.027]	0.837 [0.043]	0.845 [0.043]	0.815 [0.020]	0.732 [0.047]	0.697 [0.053]
4	0.901 [0.018]	0.839 [0.036]	0.849 [0.035]	0.808 [0.013]	0.717 [0.028]	0.680 [0.029]
5	0.904 [0.020]	0.844 [0.030]	0.857 [0.034]	0.820 [0.012]	0.739 [0.040]	0.705 [0.041]
6	0.912 [0.017]	0.853 [0.028]	0.864 [0.032]	0.824 [0.015]	0.752 [0.027]	0.716 [0.028]
7	0.892 [0.031]	0.821 [0.052]	0.831 [0.053]	0.805 [0.016]	0.723 [0.035]	0.690 [0.038]
8	0.907 [0.010]	0.845 [0.015]	0.857 [0.014]	0.817 [0.025]	0.757 [0.029]	0.725 [0.033]
9	0.919 [0.024]	0.867 [0.038]	0.883 [0.036]	0.820 [0.020]	0.755 [0.035]	0.723 [0.035]
10	0.907 [0.026]	0.851 [0.041]	0.860 [0.043]	0.806 [0.024]	0.739 [0.045]	0.704 [0.046]
11	**0.937 [0.008]**	0.893 [0.016]	0.908 [0.016]	**0.833 [0.012]**	0.788 [0.029]	0.753 [0.029]
12	0.905 [0.018]	0.850 [0.028]	0.862 [0.031]	0.809 [0.016]	0.748 [0.017]	0.718 [0.015]
13	0.911 [0.020]	0.853 [0.036]	0.867 [0.036]	0.818 [0.014]	0.759 [0.028]	0.722 [0.029]
14	0.925 [0.010]	0.884 [0.017]	0.895 [0.018]	0.800 [0.017]	0.752 [0.021]	0.720 [0.018]
15	0.914 [0.013]	0.867 [0.023]	0.877 [0.021]	0.803 [0.009]	0.742 [0.008]	0.706 [0.016]

performance is small given that the input sequences are changed. Generally, a significant improvement in network performance can be seen when compared with the RMLP and Elman models. In particular, while the training accuracy shows around a 2% improvement, the generalization performance increases by more than 7% for ACC, around 9% for Recall, and 7% for the F1 indicator.

8.7.5 The SSNN based learning models

The SSNN model is first implemented, then the integrated models taking advantages of SSNN will be presented. This allows a comparison between the original SSNN and the integrated complex models, namely SOM+SSNN and the long term dependency solving model (SOM+SSNNin). The experimental performance of the SSNN model is given in Table 8.6. As can be observed, the SSNN training performance is a little poorer than the LSTM model. Nevertheless, its testing accuracy is better than that of the LSTM model. However, the SSNN is not as good as the LSTM in recognizing small class samples. In particular, its Recall and F1 indicators are seen to be 3% lower than LSTM's results. Since ACC is considered the most important evaluation metric for this problem, it is hard to dispute

Table 8.6: Performance of SSNN on preschool children data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.768 [0.023]	0.637 [0.031]	0.621 [0.036]	0.774 [0.006]	0.662 [0.010]	0.626 [0.013]
2	0.866 [0.038]	0.767 [0.050]	0.763 [0.052]	0.821 [0.017]	0.716 [0.019]	0.683 [0.021]
3	0.849 [0.025]	0.734 [0.046]	0.723 [0.052]	0.795 [0.008]	0.674 [0.030]	0.637 [0.033]
4	0.874 [0.016]	0.764 [0.024]	0.773 [0.026]	0.820 [0.020]	0.705 [0.038]	0.672 [0.040]
5	**0.905 [0.018]**	0.820 [0.026]	0.826 [0.028]	**0.848 [0.020]**	0.753 [0.034]	0.728 [0.036]
6	0.875 [0.031]	0.753 [0.046]	0.746 [0.054]	0.827 [0.017]	0.707 [0.038]	0.671 [0.041]
7	0.798 [0.030]	0.645 [0.041]	0.633 [0.049]	0.760 [0.016]	0.594 [0.025]	0.549 [0.025]
8	0.851 [0.029]	0.729 [0.043]	0.730 [0.049]	0.791 [0.008]	0.658 [0.017]	0.619 [0.019]
9	0.867 [0.016]	0.742 [0.025]	0.737 [0.031]	0.809 [0.019]	0.681 [0.030]	0.644 [0.032]
10	0.798 [0.021]	0.635 [0.026]	0.622 [0.031]	0.773 [0.009]	0.602 [0.018]	0.556 [0.017]
11	0.806 [0.028]	0.646 [0.031]	0.634 [0.037]	0.774 [0.018]	0.601 [0.028]	0.556 [0.029]
12	0.844 [0.024]	0.695 [0.034]	0.683 [0.038]	0.783 [0.016]	0.628 [0.028]	0.588 [0.031]
13	0.800 [0.014]	0.630 [0.021]	0.612 [0.026]	0.776 [0.010]	0.600 [0.020]	0.556 [0.022]
14	0.807 [0.024]	0.634 [0.032]	0.613 [0.037]	0.789 [0.010]	0.600 [0.018]	0.554 [0.021]
15	0.785 [0.022]	0.595 [0.025]	0.561 [0.030]	0.773 [0.015]	0.582 [0.022]	0.535 [0.023]

Table 8.7: Performance comparison on preschool children data, when trained with different SSNN based models.

Models	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
SSNN	0.905 [0.018]	0.820 [0.026]	0.826 [0.028]	0.848 [0.020]	0.753 [0.034]	0.728 [0.036]
SOM+SSNN	0.912 [0.015]	0.831 [0.027]	0.826 [0.029]	0.855 [0.014]	0.786 [0.027]	0.761 [0.029]
SOM+SSNNin	**0.920 [0.019]**	0.846 [0.034]	0.845 [0.039]	**0.894 [0.011]**	0.859 [0.028]	0.843 [0.033]

the promising aspects of the SSNN model.

Table 8.7 summarizes the performance of the two SSNN based integrated models. The best SSNN experimental result is also present in this table for comparison purpose. The unsupervised pre-training module SOM is shown to be helpful in the SOM+SSNN model. Such integration helps to increase the generalization ability over the original SSNN. Importantly, the application of input modification in the SOM+SSNNin model is more beneficial. Taking the SSNN as the baseline, the complex model's testing ACC is increased by almost 5%, while more than 10% improvement for Recall and F1 indicators can be seen.

Table 8.8: Performance of MLP on School and Adolescent (Trost) data.

Data selection	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
Temporal sequence data						
10s data	0.909 [0.019]	0.907 [0.024]	0.911 [0.021]	0.824 [0.018]	0.805 [0.027]	0.808 [0.021]
30s data	0.899 [0.018]	0.893 [0.022]	0.897 [0.017]	0.826 [0.020]	0.807 [0.027]	0.811 [0.022]
60s data	**0.910 [0.016]**	0.905 [0.018]	0.910 [0.016]	**0.828 [0.018]**	0.809 [0.025]	0.814 [0.021]
Frequency transformed data						
10s data	0.882 [0.013]	0.882 [0.013]	0.889 [0.011]	0.785 [0.015]	0.760 [0.012]	0.759 [0.016]
30s data	0.864 [0.008]	0.858 [0.014]	0.862 [0.011]	0.800 [0.010]	0.762 [0.018]	0.767 [0.013]
60s data	0.869 [0.008]	0.867 [0.012]	0.877 [0.010]	0.809 [0.006]	0.780 [0.010]	0.788 [0.009]

8.8 Experimental results on SCA data

The prediction results on SCA data will be given in this section. We will show the model performance in the same order as for the preschool children data. The learning parameters and experiment settings are kept the same. Some comparisons could be made to show the result differences between the two datasets learning by the same prediction models.

8.8.1 The self organizing map

Figure 8.13 illustrates the output activation map of SOM for the SCA data, which are obtained on the 60s data. The best map size selected is 58x54, which is larger than the one used for the preschooler data, since this dataset involves a larger number of participants, or a larger number of input samples. It can be observed that only sedentary and light HH and games activities are seen to overlap, while all other samples are intuitively separated. It reflects the fact that the SCA data does not contain as much noise as the small children data, and that it is more likely to be separable by a standard neural network model. However, since several inter-weaves still appear in the map, a perfect classification performance seems not possible for this data set.

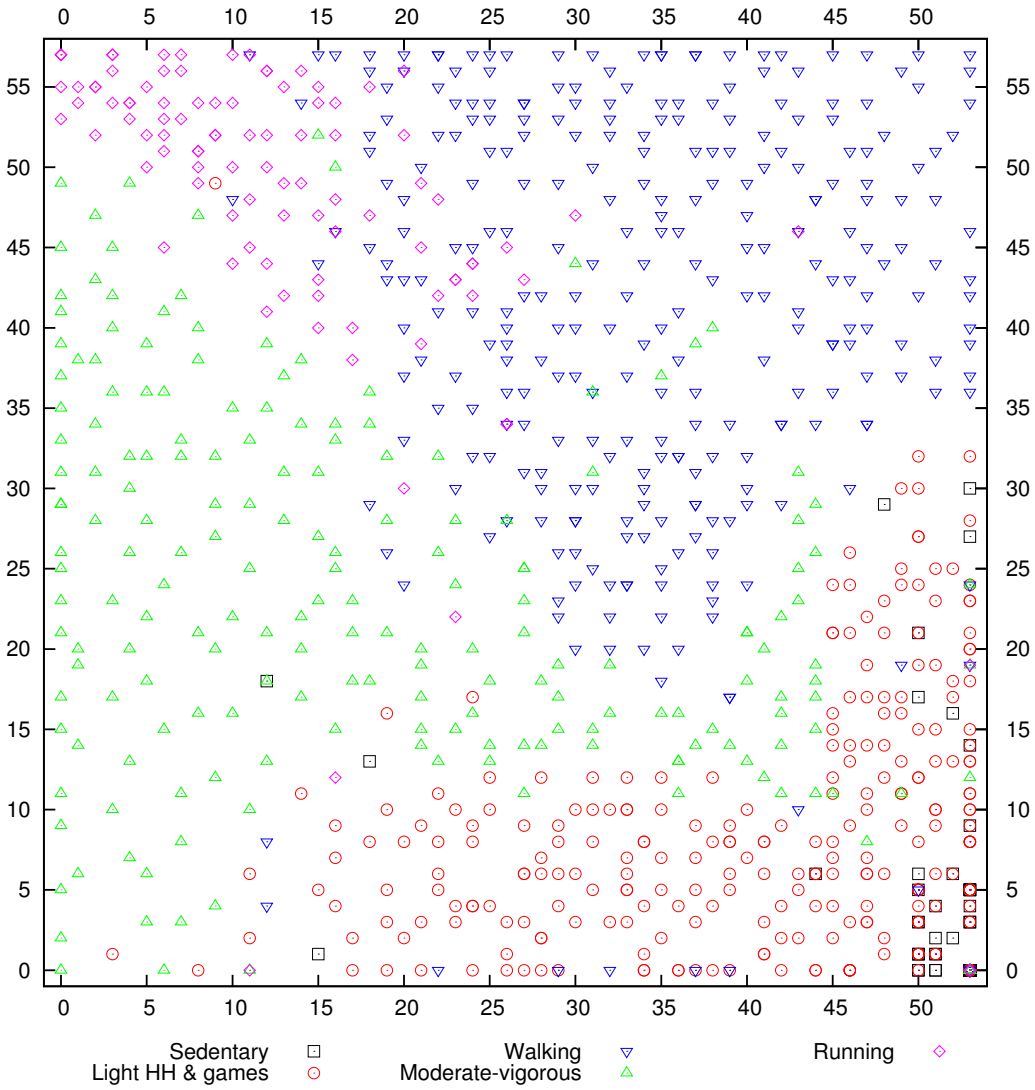

Figure 8.13: SOM activation map for SCA data with map size 58x54.

8.8.2 MLP learning

In this section, a conventional MLP is implemented. We use both the original temporal sequence as well as the frequency data which is transformed from temporal via the Fast Fourier Transform method. The classification results are shown in Table 8.8. As expected, the classification performance for SCA data is significantly higher when compared with the preschool children data. Almost 10% improvement in prediction accuracy is obtained when experimenting with the same MLP learner. It is likewise anticipated that due to the trans-

Table 8.9: Performance of Elman network on the SCA data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.794 [0.063]	0.744 [0.050]	0.757 [0.063]	0.792 [0.031]	0.760 [0.031]	0.774 [0.031]
2	0.778 [0.013]	0.699 [0.024]	0.717 [0.029]	0.779 [0.008]	0.714 [0.015]	0.736 [0.016]
3	0.812 [0.007]	0.755 [0.009]	0.774 [0.008]	0.805 [0.005]	0.761 [0.010]	0.781 [0.009]
4	0.807 [0.006]	0.741 [0.013]	0.762 [0.014]	0.798 [0.009]	0.744 [0.022]	0.765 [0.020]
5	0.804 [0.052]	0.755 [0.045]	0.767 [0.052]	0.799 [0.058]	0.768 [0.049]	0.777 [0.058]
6	0.819 [0.008]	0.766 [0.011]	0.785 [0.011]	0.820 [0.005]	0.787 [0.005]	0.804 [0.006]
7	0.820 [0.007]	0.766 [0.007]	0.786 [0.010]	0.816 [0.007]	0.781 [0.004]	0.799 [0.007]
8	0.829 [0.009]	0.779 [0.010]	0.799 [0.011]	0.819 [0.005]	0.792 [0.005]	0.808 [0.005]
9	**0.841 [0.006]**	0.799 [0.007]	0.820 [0.008]	**0.836 [0.013]**	0.814 [0.012]	0.824 [0.016]
10	0.838 [0.018]	0.798 [0.017]	0.819 [0.016]	0.824 [0.007]	0.799 [0.007]	0.811 [0.009]
11	0.824 [0.007]	0.775 [0.009]	0.793 [0.010]	0.816 [0.006]	0.787 [0.007]	0.803 [0.008]
12	0.840 [0.015]	0.800 [0.020]	0.817 [0.021]	0.830 [0.007]	0.803 [0.007]	0.817 [0.009]
13	0.805 [0.012]	0.749 [0.014]	0.769 [0.016]	0.811 [0.013]	0.777 [0.011]	0.795 [0.015]

formation to frequency data, the MLP performance decreases by 2%-3% for both training and testing performance. The reason might be that some information loss occurs when the temporal sequences are transformed into the frequency domain. It is observable that there is no significant difference in the experimental results when learning with either 10s, 30s or 60s data. Hence, we will use the 10s data (longer sequence length) in the next parts, to compare the model effectiveness in learning temporal sequences.

8.8.3 Elman and RMLP learning

Two early versions of recurrent NNs, Elman network and RMLP, are deployed, and their experimental results compared. Their corresponding training and prediction performances are summarized in Table 8.9 and Table 8.10, respectively. It can be seen that the Elman network is very much influenced by its initialization conditions. The range between the best and the worst testing performance is large, about 9% in accuracy. Nevertheless, the result of the Elman network is better than that of the MLP shown in Table 8.8.

Table 8.10 shows that the RMLP outperforms both the traditional MLP and the Elman network. On average, an improvement of more than 3% regarding the ACC indicator can be observed when compared with the Elman network performance. Both the Elman and RMLP

Table 8.10: Performance of RMLP on the SCA data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.852 [0.014]	0.808 [0.017]	0.827 [0.016]	0.836 [0.015]	0.816 [0.020]	0.826 [0.020]
2	0.828 [0.008]	0.775 [0.013]	0.794 [0.012]	0.807 [0.008]	0.772 [0.021]	0.786 [0.017]
3	0.869 [0.003]	0.833 [0.007]	0.849 [0.006]	0.853 [0.008]	0.835 [0.011]	0.845 [0.012]
4	0.858 [0.011]	0.819 [0.014]	0.836 [0.012]	0.843 [0.007]	0.823 [0.012]	0.833 [0.009]
5	0.849 [0.015]	0.806 [0.018]	0.822 [0.019]	0.835 [0.015]	0.814 [0.020]	0.825 [0.020]
6	0.868 [0.008]	0.833 [0.008]	0.849 [0.009]	0.854 [0.008]	0.837 [0.006]	0.846 [0.007]
7	0.874 [0.015]	0.842 [0.017]	0.856 [0.017]	0.856 [0.005]	0.839 [0.008]	0.850 [0.007]
8	0.874 [0.008]	0.840 [0.011]	0.853 [0.011]	0.856 [0.007]	0.841 [0.010]	0.849 [0.009]
9	0.871 [0.003]	0.841 [0.006]	0.853 [0.005]	0.866 [0.005]	0.852 [0.008]	0.861 [0.008]
10	0.875 [0.007]	0.845 [0.010]	0.858 [0.009]	0.862 [0.001]	0.848 [0.004]	0.855 [0.003]
11	**0.882 [0.009]**	0.852 [0.009]	0.865 [0.008]	**0.859 [0.007]**	0.845 [0.007]	0.852 [0.008]
12	0.873 [0.007]	0.844 [0.007]	0.856 [0.007]	0.861 [0.004]	0.851 [0.005]	0.858 [0.006]
13	0.875 [0.006]	0.843 [0.009]	0.856 [0.009]	0.862 [0.005]	0.852 [0.008]	0.858 [0.007]

algorithms are negatively affected by the length of input sequences. In particular, the poorest performance results from the longest sequence input. The RMLP performance for this data is approximately 10% better than its results for the preschool children data. This implies that the prediction task might be simpler for the case of older aged children activities.

8.8.4 Learning with the LSTM

The well known model being the least effected in learning very long temporal sequence, the LSTM, is studied in this section. In other words, the LSTM model is capable of addressing the long term dependency problem effectively. The reason for this might be attributed to the existence of memory block gates which are properly learned to open whenever the relevant information comes, and to shut otherwise. That information is captured inside the memory block as long as it is useful for the prediction task. The LSTM experimental results are shown in Table 8.11. As can be seen, the LSTM is very stable with changes of input sequence lengths as well as the sizes of input layer. The difference in the prediction performance between the longest and the shortest input sequence is only 1% accuracy. That is also the case between the smallest and the largest sizes of input layer. The standard deviations are also small for generalization performance when compared with that of the MLP or El-

Table 8.11: Performance of LSTM recurrent NN on SCA data.

Exp.ID	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
1	0.874 [0.009]	0.845 [0.013]	0.855 [0.010]	0.864 [0.008]	0.848 [0.009]	0.858 [0.009]
2	0.863 [0.022]	0.831 [0.033]	0.843 [0.033]	0.843 [0.007]	0.819 [0.010]	0.830 [0.009]
3	0.901 [0.008]	0.885 [0.015]	0.892 [0.012]	0.858 [0.004]	0.849 [0.007]	0.850 [0.004]
4	0.900 [0.010]	0.882 [0.019]	0.889 [0.015]	0.854 [0.008]	0.845 [0.006]	0.845 [0.011]
5	0.879 [0.011]	0.855 [0.014]	0.865 [0.011]	0.865 [0.004]	0.849 [0.008]	0.860 [0.004]
6	0.885 [0.014]	0.858 [0.012]	0.869 [0.014]	0.871 [0.005]	0.860 [0.005]	0.868 [0.005]
7	0.895 [0.020]	0.873 [0.022]	0.881 [0.020]	0.873 [0.006]	0.857 [0.004]	0.866 [0.006]
8	0.886 [0.019]	0.860 [0.018]	0.870 [0.019]	0.873 [0.004]	0.855 [0.005]	0.867 [0.004]
9	0.885 [0.019]	0.858 [0.019]	0.869 [0.019]	0.870 [0.006]	0.857 [0.006]	0.865 [0.006]
10	**0.904 [0.018]**	0.888 [0.019]	0.893 [0.018]	**0.884 [0.007]**	0.866 [0.009]	0.872 [0.007]
11	0.898 [0.018]	0.885 [0.020]	0.889 [0.018]	0.876 [0.004]	0.865 [0.006]	0.871 [0.004]
12	0.897 [0.008]	0.872 [0.016]	0.881 [0.011]	0.878 [0.004]	0.868 [0.006]	0.875 [0.005]
13	0.890 [0.016]	0.861 [0.026]	0.874 [0.023]	0.874 [0.005]	0.862 [0.007]	0.871 [0.007]

man networks. In general, the LSTM always outperforms both Elman and RMLP models. On average, the LSTM performance is more than 2% better than Elman and RMLP.

8.8.5 The SSNN based experiments

In this section, we conducted four sets of experiments. The original SSNN model is implemented first, then the long term dependency solver with the SSNNin approach is present. The two final set of experiments are related to the two integrated models (SOM+SSNN and SOM+SSNNin), resulting from an incorporation of SOM with the SSNN and SSNNin models. The integrated models would take advantages of the selected parameters for individual learning units. It would be seen that the prediction results increase when appropriate model cooperation is taken into account.

First, the SSNN experimental results are presented in Table 8.12. The SSNN is seen to be more effective than a bidirectional recurrent neural network in learning a input sequence. While both models attempt to learn contextual information from the past to the present, and also the relational information from the future to the present time, the advanced property of the SSNN model is that it also approximates the stable state of all nodes in the sequence. This state information would enable the model to better exploit the useful information lo-

Table 8.12: Performance of SSNN on SCA data.

Exp.ID	Training performance			Testing performance		
	ACC	**Recall**	**F1**	**ACC**	**Recall**	**F1**
1	0.970 [0.007]	0.967 [0.010]	0.971 [0.008]	0.926 [0.010]	0.922 [0.010]	0.924 [0.011]
2	0.813 [0.066]	0.699 [0.098]	0.666 [0.119]	0.785 [0.068]	0.676 [0.103]	0.643 [0.124]
3	**0.971 [0.010]**	0.969 [0.009]	0.973 [0.009]	**0.936 [0.017]**	0.930 [0.015]	0.936 [0.016]
4	0.964 [0.015]	0.962 [0.014]	0.965 [0.015]	0.926 [0.012]	0.923 [0.012]	0.927 [0.013]
5	0.837 [0.054]	0.734 [0.092]	0.712 [0.116]	0.815 [0.059]	0.719 [0.102]	0.694 [0.121]
6	0.962 [0.012]	0.961 [0.011]	0.962 [0.013]	0.928 [0.015]	0.922 [0.013]	0.925 [0.016]
7	0.941 [0.029]	0.933 [0.042]	0.938 [0.039]	0.913 [0.021]	0.908 [0.026]	0.912 [0.025]
8	0.870 [0.094]	0.798 [0.153]	0.779 [0.184]	0.846 [0.089]	0.781 [0.149]	0.759 [0.178]
9	0.956 [0.012]	0.956 [0.011]	0.960 [0.011]	0.923 [0.010]	0.917 [0.011]	0.923 [0.011]
10	0.963 [0.009]	0.962 [0.009]	0.964 [0.010]	0.928 [0.010]	0.922 [0.011]	0.927 [0.012]
11	0.933 [0.021]	0.925 [0.030]	0.927 [0.028]	0.903 [0.018]	0.896 [0.022]	0.901 [0.021]
12	0.894 [0.009]	0.876 [0.017]	0.887 [0.013]	0.880 [0.005]	0.868 [0.007]	0.878 [0.007]
13	0.960 [0.011]	0.958 [0.013]	0.962 [0.012]	0.924 [0.013]	0.919 [0.013]	0.925 [0.013]

cated far apart on the sequence. The model is quantitatively shown to be effective in this experiment. An obvious improvement on both training and testing performance can be seen, compared with previous models. On average, the SSNN performance increases from 5% to 7% compared with the best results obtained so far for this data cohort.

Secondly, the SSNNin results are shown via heatmaps in Figure 8.14, for the training (upper part) and generalization performance (lower part). We deploy experiments regarding all possible pairs of channel numbers and shortcut-link steps. Bringing past information to the present learning point is theoretically reasonable because a learning system commonly could not remember useful information located far away in the past, i.e for models with no integrated long term memory. Real links established back to the distant history would in fact alter the internal learning mechanism of the SSNN model, in the hope that it may enable the model to recall some helpful information that happened in the past. As can be observed, the results appearing at the bottom left corners of the heatmaps correspond to the original SSNN performance, since both the channel number and shortcut link step are equal to 1. This performance is taken as the base line. Generally, the SSNNin approach can bring almost 5% improvement in prediction accuracy, compared with the base line. Even a small number of channels (e.g from 1 to 3) would normally help. It is interesting that some

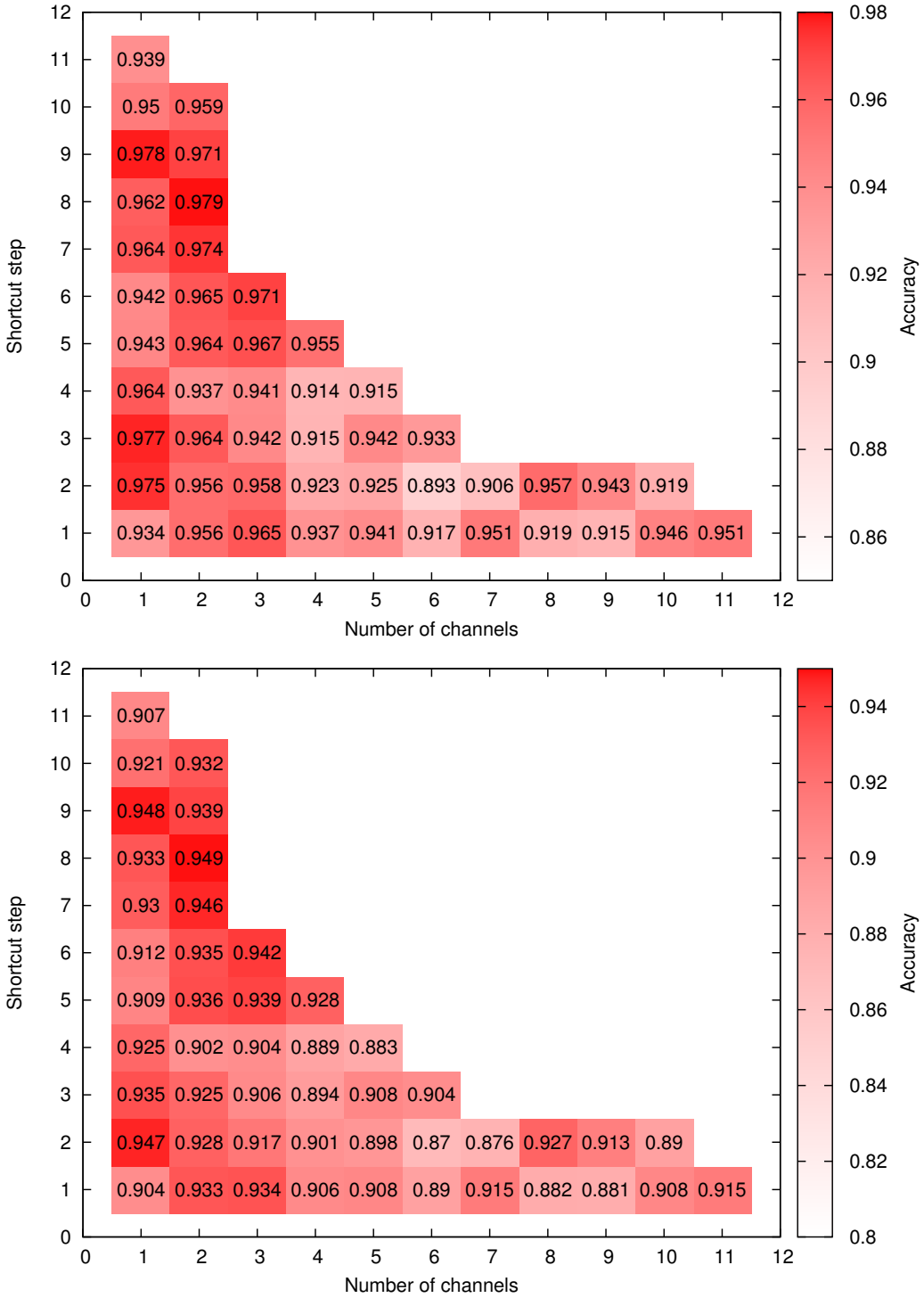

Figure 8.14: The training (top) and testing (bottom) performance of SSNNin approach when learning with frame-step of 3-3 for SCA dataset.

Table 8.13: Performance of different SSNN-based models on the SCA data.

Models	Training performance			Testing performance		
	ACC	Recall	F1	ACC	Recall	F1
SSNN	0.971 [0.010]	0.969 [0.009]	0.973 [0.009]	0.936 [0.017]	0.922 [0.015]	0.924 [0.016]
SOM+SSNN	0.977 [0.008]	0.975 [0.009]	0.978 [0.008]	0.948 [0.004]	0.931 [0.005]	0.936 [0.006]
SOM+SSNNin	**0.989 [0.006]**	0.987 [0.005]	0.990 [0.005]	**0.970 [0.006]**	0.974 [0.007]	0.975 [0.007]

information from the past is sometimes not relevant for learning the current node, resulting in poorer performance. More specifically, when adding more links to the past nodes without ranking the significance of those nodes, the contribution of other important nodes might be faded out. This results in a decrease of classification performance in some cases. It is generally suggested that if the shortcut link steps are large and the number of channels small, the network performance is widely seen to be significantly improved.

The final two experiments in incorporating SOM with either the SSNN or SSNNin model are shown in Table 8.13. The SSNN's best result is presented here for an easy comparison. As can be derived, the layer-wise models once again express robustness when learning the SCA data. The SOM+SSNN model attains more than 1% improvement in generalization accuracy compared with the original SSNN. The best performance is achieved by using the SOM+SSNNin model. At least 2% accuracy improvement compared with the SOM+SSNN model is observed. The final result confirms that older children activities are more likely to correctly be classified than smaller children data. The two age-based cohorts results in about 8% difference in prediction accuracy.

8.9 Conclusion

The current study has a number of strengths. It is the first study to evaluate machine learning approaches to accelerometry data analysis over a wide range of ages, preschool-aged, school-aged and adolescent. The activities include a variety of intensity levels from sedentary to vigorous. Innovative modelling approaches that have not yet been explored in PA research, involving layer-wise neural networks, were examined. The experiments have shown

that the use of a more suitable classifier can improve the accuracy substantially.

A limitation of this study is that the relatively small number of preschool-aged participants might influence the generalization property of the findings. Likewise, the activity trials were completed in a laboratory environment that might not reflect the real life behavior of young children. Therefore, larger studies based on free-living activity protocols are required to test the accuracy of different machine learning approaches for activity type recognition in young children. Another limitation is that only discrete integration among spatial information is made, while consideration of contextual information is neglected. More research should be undertaken to structure the spatial information in an appropriate way, such as using a graph model approach. This could be a graph learning approach, in which for each second we construct a graph with nodes being the hip, left and right wrist. The graphs would be connected as a time-series graph so that a graph-based learning model could actually learn the spatial-temporal information appropriately.

Chapter 9

Conclusion

This thesis considered a number of research questions, with a special emphasis on the effects of encoding relational data (input being described in terms of a graph together with feature vectors associated with each node) in machine learning models. Such models can then be used either for classification or for prediction purposes.

The starting point of the thesis is to investigate the architectural design conditions, specifically how a composite or integrated neural network model can be obtained from a number of building blocks: feedforward neural networks, self organizing map, for considering vectorial inputs, and PMGraphSOM, and Graph Neural Networks for considering graph input data. This thesis also conducted a similar study based on kernel machine techniques, specifically, with the building blocks consisting of support vector machine, kernel machine, for vectorial inputs, and graph Laplacian for graph inputs. The thesis demonstrated that combinations of several machine learning algorithms into integrated models qualitatively improve the learning performance. We have also considered three associated issues related to machine learning algorithms, viz., the high dimensionality of the input feature vectors, the possible imbalance of output label distributions, and the long path dependency (long term dependency) issue. We have empirically verified our proposed combined or integrated models by applying them to benchmark datasets, e.g., UK 2006 and UK 2007 web spam de-

tection datasets, the mutagenesis dataset, and the INEX 2008 XML document classification dataset. In all these cases, we obtained improved results when compared with those obtained using other state-of-the-art approaches. Moreover, we have evaluated our models on a prediction of activity type of preschool and school children based on wearable accelerometer measurements attached to various part of their bodies.

The rest of this chapter is organized as follows: Section 9.1 gives a summary of the major findings of this thesis, Section 9.2 will indicate some of the limitations of our proposed models, while Section 9.3 will provide some indications of future areas of research.

9.1 Summary of major findings

The main contributions of this thesis are summarized as follows:

- A systematic study has been made on different ways of combining fundamental building blocks, of both neural network models, and kernel machine models, with or without graph inputs. It was shown that some combinations of these fundamental building blocks led to improved results when compared with those obtained using other state-of-the-art approaches. Such experimentation led to a set of design principles, which could possibly provide guidelines on ways of how the fundamental building blocks can be combined.

  Moreover, by comparing the results of using neural network fundamental building blocks with those obtained using kernel machine building blocks, we were able to derive some insights into the behaviour of the network, which has not been observed previously.

- We further considered some associated issues which often come with machine learning problems: high dimensionality of the input feature vectors, possible imbalance in the output class distributions, and the long term dependency issue, and found that by

devoting some effort to handling these could lead to further improvements of results than those provided in the previous bullet point.

- We applied the proposed combined models to another practical problem of predicting activity type of preschool and school children from measurements of wearable accelrometers attached to various parts of their bodies. This produces results which are better than when the long term dependency issues are not explicitly considered.

9.2 Limitations of our proposed models

The following are limitations of our proposed models:

- Two models which have been given significant attention are the PMGraphSOM and GNN algorithms. The PMGraphSOM compresses high dimensional feature vectors associated with each node of a graph onto co-ordinates of low dimensional display spaces. It appears that sometimes the two dimensional display space might not be sufficient to contain useful information from the high dimensional feature space. Moreover, sometimes the mapping on the two dimensional display space might be oscillatory, in that even though the algorithm has converged, the map co-ordinates undergo significant changes in between one iteration and the next. We do not know if such instability is caused by the fact that there has never been a satisfactory convergence proof of the self organizing map, and hence we do not know how the algorithm will converge (currently it converges because the learning rate goes to zero as the number of iterations goes to infinity; it is not a model-based convergence, in that the convergence is towards an underlying model), or some other unknown factors which might have affected the stability of the converged results.

  The GNN on the other hand, suffers from long term dependency issues, especially when there are loops in the graph. The long term dependency issue exhibits itself

when the activation function in the hidden layers of the network (used to form the state space of each neuron) becomes vanishingly small due to the backprop error going to zero as it traverses layer after layer of small values of hidden layer neuron activation functions. This causes the parameters in the network to stop updating, even though the backprop error might not be 0.

- Each component of the combined model works independently and sequentially, rather than concurrently. In other words, in our approach, we wait until each component has completed its action (converged) before starting the succeeding module. This is not an issue in our case, as we are developing algorithms to work on benchmark datasets. This *will* be an issue if we work in an online fashion, in other words, the data is streaming in, e.g., in the case of web spam detection, as the information is being crawled, or in the XML document classification problem, when the documents are being collected online and we need to provide a solution immediately based on either information received from the last piece of data, or the last block of received data. Hence there is value in obtaining an online or streaming method.

- It is observed empirically that results obtained by GNN are more oscillatory than ones obtained using the kernel machine counterpart. This might be due to the problem that GNN suffers from long term dependency, while kernel machines do not. The long term dependency problem could be quite severe when there are loops which might involve a large number of nodes. The kernel machine is based on non parametric theory, and hence there are no explicit parameters in the model which need to be adapted.

9.3 Future research directions

There are a number of possible directions for future research:

- It is possible to improve the PMGraphSOM. The PMGraphSOM works well in a wide range of applications but can exhibit shortcommings in situations that require high precision mappings. The shortcoming of the PMGraphSOM is based on the fact that the display space is discrete and is of finite dimension. It will be useful if such cases are delineated, and to consider strategies in which they could be overcome. One way in which such an issue might be overcome is to investigate the possibility of a multi-resolution PMGraphSOM, and that with the multi-resolution decomposition, some of the noise could be filtered out, and that the clusters obtained would be more robust.

- It would be useful if the long term dependency issue in GNN could be considered more carefully. Currently we overcome this, in some cases, using LSTM, or SSNN. But these are implicit models, and not explicit models, with parameters which could be tuned to overcome long term dependency. Hence, it might be useful if some further investigation into GNN can be performed.

- It might be useful if a streaming version or online version of our combined models could be developed. This would allow our proposed combined model to work either in a streaming mode or online mode, which would enable its deployment in practice.

- While there is a GNN2 defined for GoG data applications, there is currently no SOM-based approach to modelling GoGs. A possible way to address this would be using a strategy which is similar to the one that developed the GNN model to the GNN2 model. That is, to use multiple levels of PMGraphSOMs to model the layers in a GoG. The problem which would require attention is how to train these multiple layers of PMGraphSOM as a single coherent system.

References

[1] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *The Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[2] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi, "A self-organizing map for adaptive processing of structured data," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 491–505, 2003.

[3] M. Kc, R. Chau, M. Hagenbuchner, A. C. Tsoi, and V. Lee, "A machine learning approach to link prediction for interlinked documents," in *Focused Retrieval and Evaluation*, S. Geva, J. Kamps, and A. Trotman, Eds. Springer Berlin / Heidelberg, 2010, vol. 6203, pp. 342–354.

[4] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, January 2009.

[5] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, May 1997.

[6] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., M. Horton, Ed. Prentice Hall, 1999.

[7] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[8] F. R. Chung, *Spectral graph theory*. AMS Bookstore, 1997, vol. 92.

[9] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[10] M. Hagenbuchner, A. C. Tsoi, A. Sperduti, and M. Kc, "Efficient clustering of structured documents using graph self-organizing maps," in *Focused Access to XML Documents*. Springer Berlin / Heidelberg, 2008, vol. 4862, pp. 20–221.

[11] S. Yong, M. Hagenbuchner, A. C. Tsoi, F. Scarselli, and M. Gori, "Document mining using graph neural network," in *Comparative Evaluation of XML Information*

Retrieval Systems, N. Fuhr, M. Lalmas, and A. Trotman, Eds. Springer Berlin / Heidelberg, 2007, vol. 4518, pp. 458–472.

[12] F. Scarselli, S. Yong, M. Gori, M. Hagenbuchner, A. C. Tsoi, and M. Maggini, "Graph neural networks for ranking web pages," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, September 2005, pp. 666–672.

[13] A. Pucci, M. Gori, M. Hagenbuchner, F. Scarselli, and A. C. Tsoi, "Applications of graph neural networks to large–scale recommender systems some results," in *Proceedings of the International Multiconference on Computer Science and Information Technology*, 2006, pp. 189–195.

[14] L. Lu, R. Safavi-Naini, M. Hagenbuchner, W. Susilo, J. Horton, S. Yong, and A. C. Tsoi, "Ranking attack graphs with graph neural networks," in *Information Security Practice and Experience*, F. Bao, H. Li, and G. Wang, Eds. Springer Berlin / Heidelberg, 2009, vol. 5451, pp. 34–359.

[15] D. Muratore, M. Hagenbuchner, F. Scarselli, and A. C. Tsoi, "Sentence extraction by graph neural networks," in *Proceedings of the 20th International conference on Artificial neural networks: Part III*, K. Diamantaras, W. Duch, and L. Iliadis, Eds. Springer Berlin / Heidelberg, 2010, vol. 6354, pp. 237–246.

[16] S. Zhang, M. Hagenbuchner, F. Scarselli, and A. C. Tsoi, "Supervised encoding of graph-of-graphs for classification and regression problems," in *Focused Retrieval and Evaluation*. Springer Berlin / Heidelberg, 2010, vol. 6203, pp. 449–461.

[17] L. D. Noi, M. Hagenbuchner, F. Scarselli, and A. C. Tsoi, "Web spam detection by probability mapping graphsoms and graph neural networks," in *Proceedings of the International Conference on Artificial Neural Networks*. Springer, 2010, pp. 372–381.

[18] F. Scarselli, A. C. Tsoi, M. Hagenbuchner, and L. D. Noi, "Solving graph data issues using a layered architecture approach with applications to web spam detection," *Neural Networks*, vol. 48, pp. 78–90, 2013.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, February 2010.

[22] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[23] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[24] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[25] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, vol. 4, no. 2, pp. 234–242, March 1992. [Online]. Available: http://dx.doi.org/10.1162/neco.1992.4.2.234

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the 10th international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 551–556.

[28] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area v2," in *Advances in neural information processing systems*, 2007, pp. 873–880.

[29] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[30] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 131–159, 2002.

[31] M. Varma and B. R. Babu, "More generality in efficient multiple kernel learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1065–1072.

[32] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf, "Large scale multiple kernel learning," *The Journal of Machine Learning Research*, vol. 7, pp. 1531–1565, 2006.

[33] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the rprop algorithm," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 1, 1993, pp. 586–591.

[34] S. Becker and Y. L. Cun, "Improving the convergence of backpropagation learning with second order methods," in *Proceedings of the Connectionist Models Summer School*, T. Hinton and Sejnowski, Eds. San Matteo, CA: Morgan Kaufmann, 1988, pp. 29–37.

[35] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.

[36] K. Ku, M. W. Mak, and W. C. Siu, "Adding learning to cellular genetic algorithms for training recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 2, pp. 239–252, 1999.

[37] Y. Da and G. Xiurun, "An improved pso-based ann with simulated annealing technique," *Neurocomputing*, vol. 63, pp. 527–533, 2005.

[38] Y. Cho and L. K. Saul, "Kernel methods for deep learning," in *Advances in neural information processing systems*, 2009, pp. 342–350.

[39] M. Trebar and N. Steele, "Application of distributed svm architectures in classifying forest data cover types," *Computers and Electronics in Agriculture*, vol. 63, no. 2, pp. 119–130, 2008.

[40] F. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Proceedings of the International Conference on Artificial Neural Networks*. London, UK, UK: Springer-Verlag, 2001, pp. 669–676.

[41] M. Hagenbuchner, G. D. S. Martino, A. C. Tsoi, and A. Sperduti, "Sparsity issues in self-organizing-maps for structures," in *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 2011, pp. 35–40.

[42] M. Hagenbuchner, S. Zhang, A. C. Tsoi, and A. Sperduti, "Projection of undirected and nonpositional graphs using self organizing maps," in *Proceedings of the European symposium on Artificial Neural Networks*, Bruges, Belgium, April 2009, pp. 22–24.

[43] F. Scarselli, A. C. Tsoi, M. Gori, and M. Hagenbuchner, "Graphical-based learning environments for pattern recognition," in *Structural, Syntactic, and Statistical Pattern Recognition*. Springer Berlin / Heidelberg, 2004, vol. 3138, pp. 42–56.

[44] H. Bunke, "Self-organizing map for clustering in the graph domain," *Pattern Recognition Letters*, vol. 23, no. 4, pp. 405–417, February 2002.

[45] M. Hagenbuchner, A. C. Tsoi, and A. Sperduti, "A supervised self-organizing map for structured data," in *Advances in Self-Organising Maps*. Springer London, 2001, pp. 21–28.

[46] C. Goller and A. Küchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proceedings of the International Conference on Neural Networks*, 1996, pp. 347–352.

[47] A. Sperduti, "Labeling raam," Connection Science, Tech. Rep., 1994.

[48] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, W. DC, Ed. Spartan Books, 1961.

[49] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi, "Contextual processing of graphs using self-organizing maps," in *Proceedings of the European symposium on Artificial Neural Networks*, Bruges, Belgium, April 2005, pp. 27–29.

[50] ——, "Contextual self-organizing maps for structured domains," in *Proceedings of the Workshop on Relational Machine Learning*, 2005, pp. 46–55.

[51] M. Hagenbuchner and A. C. Tsoi, "A supervised training algorithm for self-organizing maps for structures," *Pattern Recognition Letters*, vol. 26, no. 12, pp. 1874–1884, 2005.

[52] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi, "Graph self-organizing maps for cyclic and unbounded graphs," *Neurocomputing*, vol. 72, no. 7-9, pp. 1419–1430, March 2009.

[53] F. Aiolli, G. D. S. Martino, M. Hagenbuchner, and A. Sperduti, "Learning nonsparse kernels by self-organizing maps for structured data," *IEEE Transactions on Neural Networks*, vol. 20, no. 12, pp. 1938–1949, December 2009.

[54] F. Aiolli, G. D. S. Martino, A. Sperduti, and M. Hagenbuchner, "Kernelized self-organizing maps for structured data," in *Proceedings of the 15th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 2007, pp. 24–27.

[55] C. Goller, M. Gori, and M. Maggini, "Feature extraction from data structures with unsupervised recursive neural networks," in *Proceedings of the International Joint Conference on Neural network*, Washington, DC USA, July 1999, pp. 425–432.

[56] M. Neuhaus and H. Bunke, "Self-organizing maps for learning the edit costs in graph matching," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 3, no. 35, pp. 503–514, 2005.

[57] M. K. M. Rahman, W. P. Yang, T. W. S. Chow, and S. Wu, "A flexible multi-layer self-organizing map for generic processing of tree-structured data," *Pattern Recognition*, vol. 40, no. 5, pp. 1406–1424, 2007.

[58] A. Sperduti, D. Majidi, and A. Starita, "Extended cascade-correlation for syntactic and structural pattern recognition," in *Advances in structural and syntactical pattern recognition*, P. W. P. Perner and A. Rosenfeld, Eds. Springer-Verlag, Berlin, 1996, vol. 1121, pp. 90–99.

[59] M. Gori, M. Maggini, and L. Sarti, "A recursive neural network model for processing directed acyclic graphs with labeled edges," in *Proceedings of the International Conference on Neural Networks*, vol. 2, July 2003, pp. 1351–1355.

[60] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, January 2009.

[61] S. Zhang, "Impact sensitive ranking of structured documents," Ph.D. dissertation, University of Wollongong, May 2011.

[62] M. Hagenbuchner, "Extensions and evaluations of adaptive processing of structured information using artificial neural networks," Ph.D. dissertation, University of Wollongong, July 2002.

[63] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J. Sheng, G. Soda, and A. Sperduti, "Logo recognition by recursive neural networks," in *Graphics Recognition Algorithms and Systems*, K. Tombre and A. Chhabra, Eds. Springer Berlin / Heidelberg, 1998, vol. 1389, pp. 104–117.

[64] A. Sperduti, *Knowledge-Based Neurocomputing*. The MIT Press, Cambridge, MA, 2000, ch. A Tutorial on Neurocomputing of Structures, pp. 117–152.

[65] A. C. Tsoi, F. Scarselli, M. Gori, M. Hagenbuchner, and S. Yong, "A neural network approach to web graph processing," in *Web Technologies Research and Development*, Y. Zhang, K. Tanaka, J. Yu, S. Wang, and M. Li, Eds. Springer Berlin / Heidelberg, 2005, vol. 3399, pp. 2–38.

[66] R. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *Proceedings of the International Conference on Machine Learning*, vol. 2, 2002, pp. 315–322.

[67] J. Abernethy, O. Chapelle, and C. Castillo, "Graph regularization methods for web spam detection," *Machine Learning*, vol. 81, no. 2, pp. 207–225, 2010.

[68] A. Argyriou, M. Herbster, and M. Pontil, "Combining graph laplacians for semi-supervised learning." in *Proceedings of the Neural Information Processing Systems*, 2005, pp. 67–74.

[69] H. Chang and D.-Y. Yeung, "Graph laplacian kernels for object classification from a single example." in *Proceedings of the Computer Vision and Pattern Recognition*. IEEE Computer Society, 2006, pp. 2011–2016.

[70] S. Melacci and M. Belkin, "Laplacian support vector machines trained in the primal," *Journal of Machine Learning Research*, vol. 12, pp. 1149–1184, March 2011.

[71] ——, "Laplacian support vector machines trained in the primal," *Journal of Machine Learning Research*, vol. 12, pp. 1149–1184, 2011.

[72] S. Y. Cho, Z. Chi, W. C. Siu, and A. C. Tsoi, "An improved algorithm for learning long-term dependency problems in adaptive processing of data structures," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 781–793, July 2003.

[73] T. Lin, B. Horne, P. Tino, and C. Giles, "Learning long-term dependencies in narx recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1329–1338, November 1996.

[74] I. Sutskever and G. Hinton, "Temporal-kernel recurrent neural networks," *Neural Networks*, vol. 23, no. 2, pp. 239–243, 2010.

[75] S. Cho and Z. Chi, "Genetic evolution processing of data structures for image classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 2, pp. 216–231, 2005.

[76] M. Bianchini and F. Scarselli, "Learning long-term dependencies using layered graph neural networks," in *Proceedings of the International Joint conference on neural networks*, 2010, pp. 1–8.

[77] N. V. Tuc, A. C. Tsoi, , and M. Hagenbuchner, "Cost-sensitive cascade graph neural networks," in *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013, pp. 527–532.

[78] C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna, "A reference collection for web spam," *SIGIR Forum*, vol. 40, no. 2, pp. 11–24, December 2006.

[79] Y. Research. (2012, June) Web spam collections. http://chato.cl/webspam/datasets/ crawled by the laboratory of web algorithmics, university of milan. Accessed: 30-09-2012. [Online]. Available: http://law.di.unimi.it/

[80] I. Bíró, D. Siklósi, J. Szabó, and A. A. Benczúr, "Linked latent dirichlet allocation in web spam filtering," in *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*. New York, NY, USA: ACM, 2009, pp. 37–40.

[81] G. Geng, Q. Li, and X. Zhang, "Link based small sample learning for web spam detection," in *Proceedings of the International conference on World wide web*, 2009, pp. 1185–1186.

[82] "Spam labeling guidelines," http://barcelona.research.yahoo.net/webspam/datasets/-uk2007/guidelines/, 2007, accessed: 30-09-2012.

[83] A. Debnath, R. L. de Compadre, G. Debnath, A. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[84] L. Denoyer and P. Gallinari, "Overview of the inex 2008 xml mining track," in *Advances in Focused Retrieval*. Springer, 2009, pp. 401–411.

[85] ——, "The wikipedia xml corpus," in *Comparative Evaluation of XML Information Retrieval Systems*. Springer, 2007, pp. 12–19.

[86] S. G. Trost, W.-K. Wong, K. A. Pfeiffer, and Y. Zheng, "Artificial neural networks to predict activity type and energy expenditure in youth," *Medicine and science in sports and exercise*, vol. 44, no. 9, pp. 1801–1809, 2012.

[87] L. D. Raedt and H. Blockeel, "Using logical decision trees for clustering." in *Proceedings of the 9th international workshop on inductive logic programming, ILP*, ser. Lecture Notes in Computer Science, vol. 1297. Springer, 1997, pp. 133–140.

[88] W. Uwents, G. Monfardini, H. Blockeel, M. Gori, and F. Scarselli, "Neural networks for relational learning: an experimental comparison," *Machine Learning*, vol. 82, no. 3, pp. 315–349, 2011.

[89] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map," *Neural Networks, IEEE Transactions on*, vol. 11, no. 3, pp. 586–600, 2000.

[90] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[91] J. Ramon, "Clustering and instance based learning in first order logic," Ph.D. dissertation, K.U. Leuven. Belgium, 2002.

[92] M. Kirsten, "Multirelational distance-based clustering," Ph.D. dissertation, School of Computer Science, Otto-von-Guericke University, Magdeburg, Germany, 2002.

[93] N. Bandinelli, M. Bianchini, and F. Scarselli, "Learning long-term dependencies using layered graph neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, 2010, pp. 1–8.

[94] J. Abernethy, O. Chapelle, and C. Castillo, "Witch: A new approach to web spam detection," in *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web*. Citeseer, 2008.

[95] G. Cormack, "Content-based web spam detection," in *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web*, 2007.

[96] J. Martinez-Romo and L. Araujo, "Web spam identification through language model analysis," in *Proceedings of the 5th international workshop on adversarial information retrieval on the web*. ACM, 2009, pp. 21–28.

[97] E. Trentin and E. D. Iorio, "Classification of molecular structures made easy." in *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2008, pp. 3241–3246.

[98] H. Lodhi and S. Muggleton, "Is mutagenesis still challenging?" in *Proceedings of the 15th International Conference on Inductive Logic Programming, Late-breaking Papers.*, 2005, pp. 35–40.

[99] S. Kramer and L. D. Raedt, "Feature construction with version spaces for biochemical applications." in *Proceedings of the International Conference on Machine Learning*, 2001, pp. 258–265.

[100] M. Krogel, S. Rawles, F. Zelezny, P. Flach, N. Lavrac, and S. Wrobel, "Comparative evaluation of approaches to propositionalization." in *Proceedings of the 13th international conference on inductive logic programming, ILP*, 2003, pp. 197–214.

[101] R. C. Wilson and P. Zhu, "A study of graph spectra for comparing graphs and trees," *Pattern Recognition*, vol. 41, no. 9, pp. 2833–2841, 2008.

[102] Z. Sun, N. Ampornpunt, M. Varma, and S. Viswanathan, "Multiple kernel learning and the smo algorithm," in *Advances in neural information processing systems*, 2010, pp. 2361–2369.

[103] N. V. Tuc, M. Hagenbuchner, A. C. Tsoi, and S. Zhang, "On the effects of incorporating input graph topology in neural networks," *IEEE Transactions on Neural network and learning systems, submitted*, 2015.

[104] C. Jose, P. Goyal, P. Aggrwal, and M. Varma, "Local deep kernel learning for efficient non-linear svm prediction," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 486–494.

[105] Y. Tang, "Deep learning using support vector machines," *arXiv preprint arXiv:1306.0239*, 2013.

[106] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, "Parallel support vector machines: The cascade svm," in *Advances in neural information processing systems*, 2004, pp. 521–528.

[107] C.-T. Lu, K.-Y. Huang, N. A. Bretaña, W.-C. Chang, and T.-Y. Lee, "Exploiting two-layered support vector machine to predict phosphorylation sites on virus proteins," *International Journal of Bioscience, Biochemistry and Bioinformatics*, vol. 3, no. 5, pp. 460–465, 2013.

[108] Y.-X. Wu, L. Guo, Y. Li, X.-Q. Shen, and W.-L. Yan, "Multi-layer support vector machine and its application," in *Proceedings of the International Conference on Machine Learning and Cybernetics*. IEEE, 2006, pp. 3627–3631.

[109] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[110] S. Hido, H. Kashima, and Y. Takahashi, "Roughly balanced bagging for imbalanced data." *Statistical Analysis and Data Mining*, vol. 2, no. 5-6, pp. 412–426, 2009.

[111] R. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[112] Y. Tang, Y. Zhang, N. Chawla, and S. Krasser, "Svms modeling for highly imbalanced classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 1, pp. 281–288, 2009.

[113] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: a unifying framework for information theoretic feature selection," *The Journal of Machine Learning Research*, vol. 13, pp. 27–66, 2012.

[114] M. Géry, C. Largeron, and C. Moulin, "Ujm at inex 2008 xml mining track," in *Advances in Focused Retrieval*. Springer, 2009, pp. 446–452.

[115] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[116] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *The Journal of Machine Learning Research*, vol. 5, pp. 1205–1224, 2004.

[117] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *The Annals of Statistics*, vol. 32, no. 2, pp. 407–451, 2004.

[118] T. J. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009.

[119] H. Zhang, Z. Xu, X. Chang, and Y. Liang, "Variable selection and sparse reconstruction via L1/2 regularization," *Sparsity in Machine Learning and Statistic, London*, 2009.

[120] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[121] F. Garcıa López, M. Garcıa Torres, B. Melián Batista, J. A. Moreno Pérez, and J. M. Moreno-Vega, "Solving feature subset selection problem by a parallel scatter search," *European Journal of Operational Research*, vol. 169, no. 2, pp. 477–489, 2006.

[122] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[123] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[124] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[125] M. Galar, A. Fernández, E. B. Tartas, H. B. Sola, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches." *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 42, no. 4, pp. 463–484, 2012.

[126] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive." in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*. ACM, 1999, pp. 155–164.

[127] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan, "Adacost: Misclassification cost-sensitive boosting." in *Proceedings of the 6th International Conference on Machine Learning*, I. Bratko and S. Dzeroski, Eds. Morgan Kaufmann, 1999, pp. 97–105.

[128] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique." *Journal of Artificial Intelligence and Research*, vol. 16, pp. 321–357, 2002.

[129] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *Proceedings of the 14th International Conference on Machine Learning*, D. H. Fisher, Ed. Morgan Kaufmann, 1997, pp. 179–186.

[130] C. X. Ling and C. Li, "Data mining for direct marketing: Problems and solutions." in *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1998, pp. 73–79.

[131] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufmann, 1996, pp. 148–156.

[132] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Springer, 2003, pp. 107–119.

[133] H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting and data generation: the databoost-im approach," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.

[134] H. Schwenk and Y. Bengio, "Boosting neural networks." *Neural Computation*, vol. 12, no. 8, pp. 1869–1887, 2000.

[135] B. X. Wang and N. Japkowicz, "Boosting support vector machines for imbalanced data sets," *Knowledge and Information Systems*, vol. 25, no. 1, pp. 1–20, 2010.

[136] M. Kukar and I. Kononenko, "Cost-sensitive learning with neural networks," in *Proceedings of the 13th European conference on artificial intelligence*. Citeseer, Brighton, UK, 1998, pp. 445–449.

[137] B. Zadrozny, J. Langford, and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting," in *Proceedings of the 3rd International Conference on Data Mining*. IEEE Computer Society, 2003, pp. 435–442.

[138] G.-G. Geng, C.-H. Wang, Q.-D. Li, L. Xu, and X.-B. Jin, "Boosting the performance of web spam detection with ensemble under-sampling classification," in *Proceedings of the 4th International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 4. IEEE, 2007, pp. 583–587.

[139] D. Scherer, H. Schulz, and S. Behnke, "Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors," in *International Conference on Artificial Neural Networks*. Springer, 2010, pp. 82–91.

[140] S. Wang, Y. Hong, and J. Yang, "Xml document classification using closed frequent subtree," in *Proceedings of the Web-Age Information Management*. Springer, 2012, pp. 350–359.

[141] C. Largeron, C. Moulin, and M. Géry, "Entropy based feature selection for text categorization," in *Proceedings of the Symposium on Applied Computing*. ACM, 2011, pp. 924–928.

[142] B. Chidlovskii, "Semi-supervised categorization of wikipedia collection by label expansion," in *Proceedings of the Advances in Focused Retrieval*. Springer, 2009, pp. 412–419.

[143] L. M. de Campos, J. M. Fernández-Luna, J. F. Huete, and A. E. Romero, "Probabilistic methods for link-based classification at inex 2008," in *Advances in Focused Retrieval*. Springer, 2009, pp. 453–459.

[144] R. Kaptein and J. Kamps, "Using links to classify wikipedia pages," in *Proceedings of the Advances in Focused Retrieval*. Springer, 2009, pp. 432–435.

[145] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, USA, 2011, pp. 315–323.

[146] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[147] M. Hagenbuchner, D. P. Cliff, S. G. Trost, N. V. Tuc, and G. E. Peoples, "Prediction of activity type in preschool children using machine learning techniques," *Journal of Science and Medicine in Sport*, vol. 18, no. 4, pp. 426–431, 2015.

[148] J. Staudenmayer, D. Pober, S. Crouter, D. Bassett, and P. Freedson, "An artificial neural network to estimate physical activity energy expenditure and identify physical activity type from an accelerometer," *Journal of Applied Physiology*, vol. 107, no. 4, pp. 1300–1307, 2009.