Adaptation-Based Classifiers for Handling Some Problems with Multi-Label Data

A thesis submitted to the *Indian Statistical Institute*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

by

Anwesha Law
Senior Research Fellow

under the supervision of
Prof. Ashish Ghosh

Machine Intelligence Unit
Indian Statistical Institute
Kolkata, India

June, 2022

To my family and friends

Acknowledgement

In my journey as a PhD scholar, I am immensely grateful to multiple individuals who have helped me in various capacities throughout the past six years. I extend my sincere gratitude to my supervisor Prof. Ashish Ghosh for his meticulous guidance and motivation. He has patiently guided us through every obstacle, responded to every query and encouraged our inquisitiveness to gather knowledge. I am thankful to Prof. Susmita Ghosh, Jadavpur University for her academic suggestions and support.

I appreciate the research fellowship (Regn. No. JCSK-CC-0081) awarded to me by the Indian Statistical Institute for smoothly conducting my research. I am also grateful to Prof. Sanghamitra Bandyopadhyay, the honourable director of the institute, for providing great research infrastructure and facilities during the period of my doctoral studies. Heartfelt thanks are due to the Dean of Studies, faculties of Ph.D. & D.Sc. Committee, Research Fellow Advisory Committee (RFAC), Computer and Communication Sciences Division (CCSD), and Machine Intelligence Unit (MIU). My work could not have progressed smoothly without their constant support and invaluable guidance. From having constructive discussions to maintaining an amiable environment throughout my research tenure, I admire my labmates and colleagues, especially, Mr Rahul Roy, Ms Debasrita Chakraborty, Mr Subhadip Boral and Dr Shion Samadder Chaudhury. Additionally, I appreciate all the interns who have collaborated with me during my tenure. I thank the Computer & Statistical Service Center (CSSC) for providing computing facilities, the ISI library for the reference materials, the office staff of the Machine Intelligence Unit for maintaining a friendly environment, and the authorities of ISI for extending various facilities. Unable to include every individual's name in this limited space, I would like to acknowledge every constructive suggestion, motivation and help that I have received throughout my tenure.

I shall forever remain grateful for the love and encouragement I receive from my parents (Mrs Mitali Law and Mr Ajoy Kumar Law), my sisters - (Anupa, Asmita and Ankita), my other family members and close friends. Finally, I thank my husband (Dr Samarjit Ghosh) for his unconditional support and constant motivation through every step of my journey.

Abstract

The concept of multi-label (ML) data generalizes the association of instances to classes by labelling each data sample with more than one class simultaneously. Since this data can belong to more than one class at the same time, instances that are multi-label in nature, should not be forcefully assigned a single label. It needs to be handled in its original form. However, various problems arise while dealing with multi-label data.

In this thesis, four such issues have been highlighted and dealt with. The first problem is the large input dimension that sometimes occurs in multi-label data. Dimensionality reduction of the features help to strike a balance between the feature size, the number of samples and the output dimension. The next limitation is that of a complex decision space with overlapping class boundaries. This occurs due to the instances belonging to multiple classes simultaneously. Various approaches such as improving the feature to class mapping, increasing the class separability and simplifying the decision space have been implemented. The third drawback arises due to a large number of classes and label-sets in multi-label data, most of which are under-represented. This emphasizes the problem of class imbalance that widely prevails in multi-label data. This imbalance has been handled through the usage of customized classifiers suitable for the data at hand. Finally, the problem of class correlation is to be handled in this thesis. Multiple classes simultaneously assigned to every instance indicates a possibility of a few classes co-occurring on numerous occasions. These frequently co-occurring classes might have some correlation among them which have been identified and utilized to improve the multi-label classification performance.

This thesis addresses the above-mentioned issues to perform efficient multi-label classification. Smaller components that target the individual issues have been incorporated to build large classification models. The first work aims to reduce feature dimensions and learn a better feature to class mapping for the complex decision space. A shallow but fast network known as extreme learning machines (ELMs) has been cascaded with autoencoders (AEs) to propose a network that can handle both issues. Two variations of the network have been proposed. To further explore the overlapping boundaries of ML data, the second contribution increases the separability of the complex decision space and also incorporates dimensionality reduction. Functional link artificial neural network (FLANN) has been adopted here for the unique functional expansion capability that transforms the features to a higher dimension thus making it considerably more separable. After identifying the best configuration of the network, it has then been integrated with autoencoders to reduce the functionally expanded feature dimension and

bring additional transformation into the multi-label data. While these classifiers display improved performance, they do not consider the problems of class imbalance or label correlation. Hence, the third work builds a tree of classifiers that handles the problem of class imbalance, simplifies decision space for the ease of learning and preserves label correlations. A novel label-set proximity-based technique has been devised that simplifies boundaries and splits the data while preserving label correlations. Every split is learned by a classifier suited for the balanced or imbalanced data at hand. While handling multiple issues together successfully, this classifier tree model preserves label correlations but does not explicitly use them to improve classification performance. In this regard, the final contribution specifically extracts underlying label correlations from the data and associates them with predictions of existing multi-label classifiers to improve the overall performance. A novel frequent label-set mining technique generates rules that help to improve scores predicted by the existing multi-label algorithms. This thesis incorporates various elements to handle the problems of multi-label data and converges them to create cohesive models for multi-label classification.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Looking into the machine learning paradigm [5], data can be thought of as the key ingredient to the constant development of the domain. The variation in the availability of data has led to the vast improvement in machine learning technology in recent years. Machine learning algorithms were developed to handle and learn from the various types of data that were available. In the past, when access to data was limited, development in this field was comparatively slow-paced. With the advent of social media and online activity throughout the world, today there is an abundance of raw data that is being accumulated every second. Also, the constant improvement and availability of technology is changing the dimension of fields like machine learning. Today, every data related domain is experiencing exponential growth in research over time. This huge availability of data made researchers re-evaluate past ideas and look into them in a new light. Concepts that seemed far-fetched and difficult to achieve are being thoroughly explored.

One such dimension is that of multi-label data [28]. Traditionally, data in machine learning algorithms have always been said to belong to individual predefined categories. Algorithms dealt with data that could belong to a single class at a time. Various concepts have been introduced that have improved on these older ideas. Now, data is known to belong to more than one class simultaneously. This concept falls under the umbrella of multi-label learning. This kind of data occurs frequently around us, hence needs to be

well appreciated and explored.

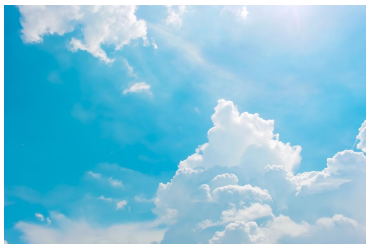
1.1 Multi-label Classification

Multi-label learning is a part of machine learning that deals with data that can belong to more than one category simultaneously. Unlike traditional machine learning techniques [5] which vastly deal with data that belong to one group (also known as single-label/multi-class data), multi-label learning aims to handle data that, as the name suggests, has multiple labels. In the domain of multi-label learning, the supervised learning approach, specifically, classification is being explored by researchers. Currently, multi-label learning is synonymous with multi-label classification. This field of multi-label classification (MLC) [28] is being worked on for the last two decades, and it is mostly used in the fields of text categorization, gene-based classification, learning from images, video, audio, etc. In the past decade, the field of multi-label learning has been explored and some amount of research work has been done in the fields of text categorization, labelling of multi-media data, gene prediction, etc.

For example, traditionally, while categorizing images from a set of classes, say, *sand*, *sky* and *sea*, they are usually assigned a single label. Figure 1.1 shows three images which belong to one of the above-mentioned classes. However, the problem arises if for the same

(a) Class: *sand*

(b) Class: *sea*



(c) Class: *sky*

Figure 1.1: Single-label images (Source: Google Images)

set of labels, the concerned images to be categorized contain more than one of the classes. Figure 1.2 shows four images that have multiple classes like {*sky, sand*}, {*sky, sea*}, {*sea, sand*} and {*sky, sea, sand*}. Clearly, each image belongs to all the classes labelled, thus

(a) Class: {*sky, sand*}

(b) Class: {*sky, sea*}

(c) Class: {*sea, sand*}

(d) Class: {*sky, sea, sand*}

Figure 1.2: Multi-label images (Source: Google Images)

they cannot be passed as single-label data for the same set of classes. These types of data are multi-label in nature. If these images are forcefully assigned to any one of the three labels, it means ignoring the information in the data that might prominently describe other classes. To avoid such issues with data, multi-label classification is a more suitable approach to deal with data that clearly might belong to multiple classes simultaneously. However, unlike single-label data, simultaneous involvement of numerous classes makes multi-label data somewhat ambiguous. This ambiguity arises due to its association with multiple classes at the same time. Thus, it can be said, that ML data is quite complex and hence needs to be handled efficiently to deal with the inherent ambiguity in the data.

The concept of multi-label sometimes might be confused with other similar areas like fuzzy logic [42]. In fuzzy, membership of instances to different classes vary over a certain range of values. However, unlike fuzzy logic, the association of instances to all the labels for ML data is absolutely crisp. It equally belongs to all the classes. Similarly,

ML classification is also quite different from multi-object recognition. Object recognition focusses on segregating and identifying various objects in an image, whereas ML classification is more focused on identifying the underlying concepts in the data, instead of specific objects. For a better understanding, the representation of multi-label data has been discussed below.

Representation of Multi-label data In a multi-label dataset with N samples, the i^{th} instance is represented as a feature vector $\vec{X_i} = \{x_{i1}, x_{i2}, ..., x_{id}\}$ (Table 1.1). Each

Table 1.1: Multi-label data representation

Instance No	Feature 1	Feature 2	..	Feature d	Class 1	Class 2	..	Class C
X_1	x_{11}	x_{12}	$\cdots$	x_{1D}	y_{11}	y_{12}	$\cdots$	y_{1c}
X_2	x_{21}	x_{22}	$\cdots$	x_{2D}	y_{21}	y_{22}	$\cdots$	y_{2c}
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_N	x_{N1}	x_{N2}	$\cdots$	x_{ND}	y_{N1}	y_{N2}	$\cdots$	y_{NC}

element x_{ij} is a feature, where $x_{ij} \in \Re$, $1 \leq j \leq d$ and d is the dimension of the input space. Each of these input patterns $\vec{X_i}$ are associated with a corresponding output vector $\vec{Y_i} = \{y_{i1}, y_{i2}, ..., y_{iC}\}$, where C is the dimension of the output space. Among these C classes, the multi-label data instance can belong to more than one class at a time. Each element of the vector $\mathbf{Y_i}$ is a binary value, indicating if the corresponding label is relevant to the sample or not. $y_{ic} = 1$ indicates that the i^{th} input pattern belongs to the c^{th} class, therefore it is *relevant*, and $y_{ic} = 0$ indicates an *irrelevant* label. Several labels can be active at once, unlike in the case of single-label data where only one label is active at a time. The group of relevant labels associated with an instance is known as its label-set. This relevant label-set indicates complete association of the instance with each of the relevant classes.

Table 1.2: Example of multi-label data representation

Image	Class sky	Class sea	Class sand	Relevant label-set	Irrelevant label-set
a	1	0	1	{sky, sand}	{sea}
b	1	1	0	{sky, sea}	{sand}
c	0	1	1	{sea, sand}	{sky}
d	1	1	1	{sky, sea, sand}	{}

The multi-label representation of the images in Figure 1.2 are shown in Table 1.2. Each class association is represented by 1 and it is a part of the relevant label-set. Despite the various approaches to multi-label classification, the inherently ambiguous nature of the data results in several complexities that arise while dealing with this kind of data. Concerns like complex decision space and correlated classes need to be handled specifically. Researchers approach the issues from different perspectives to handle the shortcomings.

1.2 Motivation

While exploring the field of multi-label learning, few inherent issues seem to cause a bottleneck for multi-label algorithms. These problems are intriguing and gives the motivation to develop techniques that can handle these drawbacks to achieve better multi-label classification.

1. **Large input dimension -** Multi-label data in general is seen to have a large input and/or output dimension. This happens since the source of multi-label data is mostly the internet which has a vast range of content. For example, for text data, the sources may be online articles and blogs. If unique words are considered as features, these sources generate a high input dimension. Similarly, since ML data can belong to multiple classes at the same time, various output possibilities (even the rare classes) are considered, which leads to the output dimension becoming huge as well. To handle and obtain optimum information from the raw multi-label data the input and output dimensions need to be considered fully. Complete removal of rare classes or features might lead to information loss. In this way, almost all types of multi-label data sources tend to generate large dimensions of input and output which needs to be handled seriously. Another issue with multi-label data is the availability of labelled data. When datasets with huge dimensions are being used to train a multi-label classifier, it needs to have sufficient number of instances as well. If there is no balance between the feature and sample size, the training

might not be fruitful. Thus to reduce the bulk and create balance, these large dimensional data need to be handled properly to achieve efficient performance from the ML classifiers. The work in this thesis does not intend to reduce the class size avoiding the chance of multi-label information loss or to increase sample size. Thus, the path of feature dimensionality reduction has been chosen to strike a balance between the feature dimension, sample size and number of classes.

2. **Complex output dimension with overlapping class boundaries -** When dealing with multi-label data, one of the most important issues to keep in mind is its complex decision space [28]. The multi-label classifiers need to learn decision boundaries to separate the various classes that are simultaneously associated with multiple data instances. This situation is much more complex than what is encountered while performing single-label classification. For single-label data, no matter how the decision boundary looks like, at some higher dimensional space, each instance can be distinctly separated from the other instances belonging to different classes. Each class will have a well-defined, non-overlapping decision boundary which makes the task of classification much simpler. However, for ML data, that is not the case. Since each instance belongs to multiple classes, all the classes cannot have completely segregated and well defined decision boundaries. Ideally, it can be said that there is one decision boundary for each of the classes, where it partitions the data in a one-vs-all approach. Therefore, on the whole, there are multiple overlapping class boundaries that help segregate different classes of multi-label data. For example, in Figure 1.3 a toy scenario has been shown with 3 classes. Each class has a separate class boundary and the data which lie within the boundaries have specific label-sets. Thus, each unique label-set can be well-separated and have distinct boundaries, but that does not happen for each class. This makes the task of ML classification much more complex than its single-label counterpart. In this thesis, the aim is to handle this bottleneck through various approaches. Initially, a classification model is built to learn the input to multi-label output mapping in a

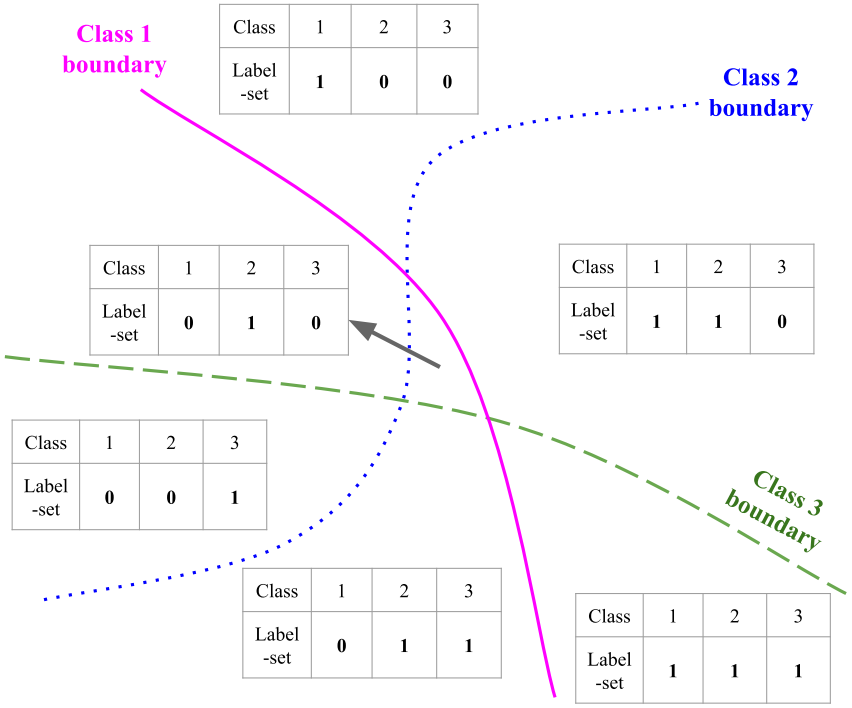


Figure 1.3: An example of multi-label class boundaries

better way. Another approach attempts to increase the separability of the complex decision space. Finally, the classification model tries to simplify the overlapping boundaries of the data in order to perform better multi-label classification.

3. **Imbalanced classes -** The next problem is that of class imbalance, which seems very obvious if it is looked into. When a dataset has a large output dimension with numerous classes, it is quite likely that all those classes may not be equally represented. To encompass the entire essence of multi-label data, even the rarer classes need to be kept under consideration. This leads to an uneven representation of all the classes. Additionally, multi-label data also has the concept of label-sets, which means that each unique set of classes that exist within the dataset also needs to be well-represented. Practically, a sufficient amount of instances from all classes as well as the unique label-sets is quite difficult to achieve. Thus, multi-label data is inherently imbalanced. Section 2.2.1 shows the imbalance in benchmark multi-label datasets. This problem affects the training of ML classifiers, since it creates a bias towards the larger classes. Even if the imbalance cannot be removed

completely, keeping in mind the large output dimension, it is important to improve the balance and reduce the gap among the various classes for a better training of the ML classifier. This problem is handled in one of the contributions, where instead of modifying the data to remove imbalance, the model has been customised to specifically deal with imbalanced classes.

4. **Class correlation -** The final problem is the existence of class correlation among multi-label data which should not be ignored. As multi-label instances belong to multiple classes at the same time, this incurs a probability that in a dataset, a subset of instances may have the same combination of labels. When this occurs repeatedly, it is likely that some correlation exists between the frequently co-occurring classes.

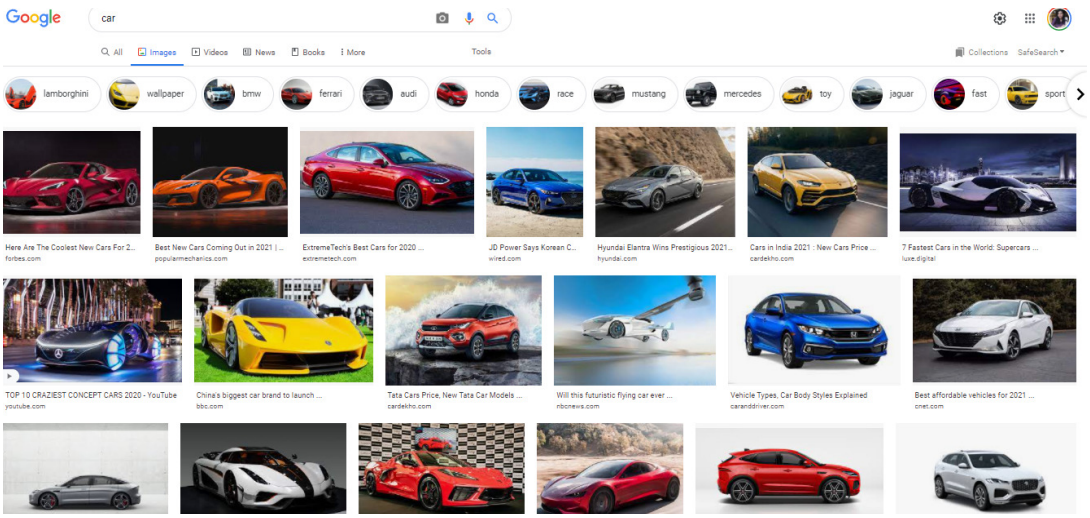


Figure 1.4: Google image search for "car"

Consider an example for classifying images with classes *car*, *road*, *boat* and *sea*. Figures 1.4 and 1.5 are Google image search results for *car* and *boat* respectively. From the images it is seen that classes *car* and *road* are co-occurring more frequently than *car* and *sea*. Similarly, class *boat* frequently co-occurs with class *sea* than with class *road*. Thus, the pair of classes *car* - *road* or *boat* - *sea* should have some correlation among them. The problem with existing ML algorithms is that in most cases these correlations tend to be ignored. Applying the one-vs-all approach

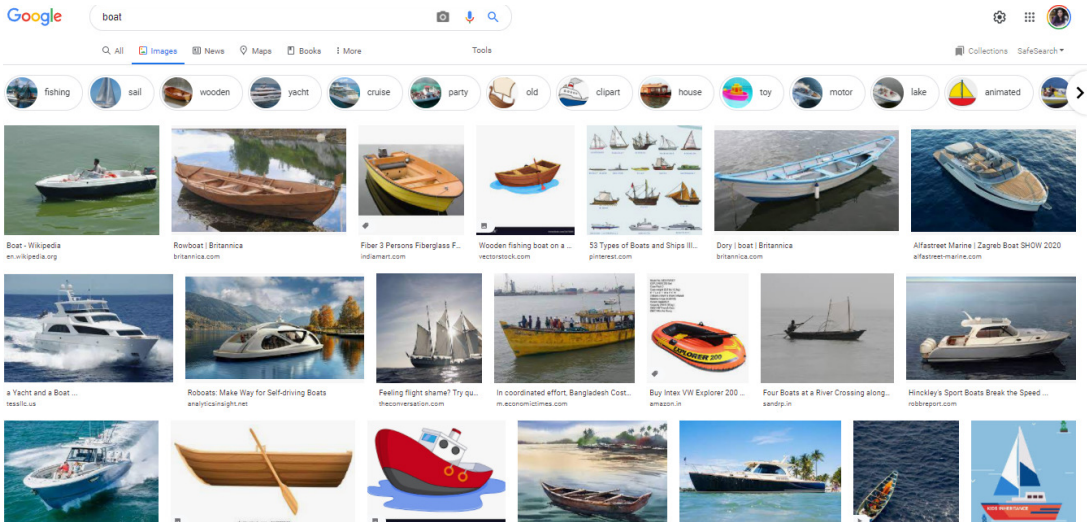


Figure 1.5: Google image search for "boat"

or a single classifier without specifically incorporating class correlations makes the classifiers consider the individual classes to be distinct. This might lead to the loss of information and misclassification. Thus, it is important to acknowledge and incorporate the label correlations that exist within the ML data to improve the classification mechanism. Keeping this in mind, one approach aims to preserve label correlations implicitly while another approach creates a method that would extract and utilize underlying correlations to be combined with existing multi-label classification models.

Handling each of these issues is quite essential in the field of multi-label classification. Researchers are attempting to deal one or more issues through their works. In this thesis, through the endeavour to handle the bottlenecks in multi-label data, attempts to include a combination of these above-mentioned problems has been made in each of the proposed works. Smaller components targeted to handle the individual problems are in turn combine to build the large classification model.

1.3 Scope of the Thesis

This thesis is a comprehensive attempt to deal with the various bottlenecks that occur while classifying multi-label data. Different models have been developed to handle a combination of the four drawbacks while improving the multi-label classification performance. The models developed in this thesis are specifically adaptation-based models, that use existing traditional models and adapt them to suit the multi-label requirement without modifying the data. Some theoretical and experimental results have been presented to demonstrate the effectiveness of the proposed techniques. Among the four specific problems targeted in this thesis, a subset of problems has been handled through different approaches by the proposed models in each chapter. The first problem of large feature dimension has been handled with the use of autoencoders incorporating it with other shallow networks that are adapted to handle the second problem of ML decision space complexity. Thus, initially, dimensionality reduction and improved feature to decision space mapping has been done using autoencoders and extreme learning machines. Then, another approach of handling the complex decision space has been explored through the use of functional link networks which projects the data to a higher dimension and improves the separability. Alongside, autoencoders have been utilized to reduce the expanded input dimension. The next model aims to handle the complex decision space problem by simplifying it through partitions. Additionally, here, the third problem of class imbalance is approached through the use of custom classifiers and parameters and the final problem of class dependencies is implicitly preserved through a proximity-based label-space partitioning technique. The final work focusses on explicitly utilizing the existing label dependencies in the data to improve traditional multi-label classifier performance through frequent label-set mining and association.

Figure 1.6 shows the flow of the thesis highlighting the problems dealt with in each proposed work. The current chapter deals with an introduction to multi-label classification. The next chapter briefs on the background and some key information followed by four

contributory chapters and the final concluding chapter.

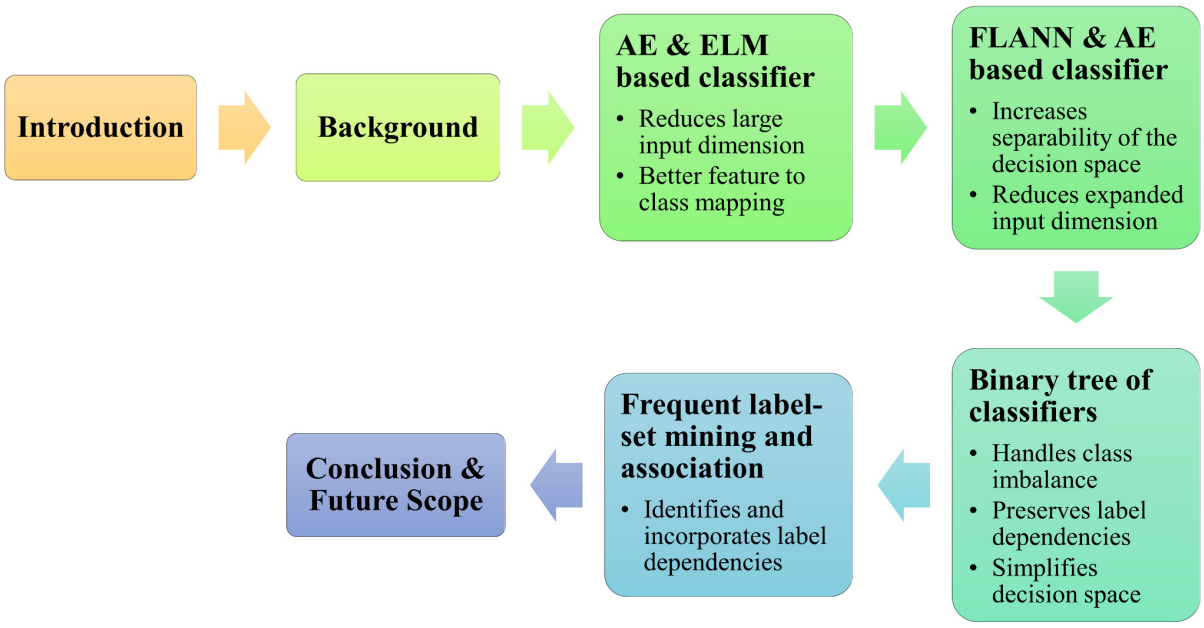


Figure 1.6: Flow of the thesis

Chapter 2

This chapter provides a background that is essential for a better understanding of multi-label data and the work done in this thesis. It starts with a literature survey section, that includes popular ML algorithms along with other relevant methods. The datasets to be used later for experimental analysis have also been described here. Finally, the performance metrics, specific to multi-label data have been included.

Chapter 3

This chapter introduces the first proposed work which is a cascade of neural networks for the classification of multi-label data. Two types of networks, namely, autoencoder (AE) and extreme learning machine (ELM) have been incorporated in the proposed systems. ELM is a compact and efficient single-label classifier that seems to lose its efficiency while dealing with multi-label data. This happens due to the complex nature of the multi-label data, which makes it difficult for the smaller networks to interpret it accurately. This

work attempts to deal with two of the bottlenecks faced while handling multi-label data. The first challenge is to handle the dimensionality and representation of the input space for multi-label data, and the other challenge is the effective mapping of the input to the complex output space. Thus, the aim is to enhance the performance of a stand-alone multi-label extreme learning machine (MLELM) by collaborating it with other networks. For the initial model, there are three basic phases: feature encoding, soft classification and class score approximation. In the first step, a stacked autoencoder (SAE) network is employed to generate a discriminating and reduced input representation of the multi-label data. This makes the data compact and more manageable for the successive stages. This data in turn is used by an MLELM in the next phase for the prediction of soft labels. In the final step, to improve the prediction capability of the previous network, a novel approach of approximating the class score is proposed using an additional MLELM. This model has been further improved to develop a deeper cascaded network that portrays improved performance. This model incorporates a deep autoencoder (DAE) and stacked MLELMs. In the first phase, the DAEs handle the large feature space of ML data and the subsequent stacked MLELMs intricately learns the input to output mapping and performs ML classification. Comprehensive experimental evaluation of the proposed approaches has been performed on various datasets and overall it displays a promising performance. To improve the learning capability of the classification model, in the next chapter the possibility of increasing the class separability of the complex multi-label decision space has been explored.

Chapter 4

To handle the inherent complexity of multi-label data, a compact and efficient network known as functional link artificial neural network (FLANN) has been explored in this chapter. FLANNs are known to functionally transform the input space to introduce non-linearity into the data, thus making the task of separating the classes in the output space comparatively simpler. In this chapter, a multi-label functional link artificial

neural network (MLFLANN) has been developed to efficiently perform multi-label data classification. The input data is functionally expanded to a higher dimension, followed by iterative learning of the MLFLANN using the training set. The architecture of the network is less complex and the input space dimension is improved in an attempt to overcome the non-linear nature of the multi-label classification problem. Furthermore, six multi-label FLANN models have been devised for multi-label classification to procure the optimal configuration. These six variations of the network have been built using three basis functions - trigonometric, Chebychev, power polynomial and two learning techniques - backpropagation and particle swarm optimization. Finally, an extended model has been developed where the input features are subjected to two transformations adapted from MLFLANN and autoencoders. First, a functional expansion of the original features is made using basis functions. This is followed by an autoencoder-aided transformation and reduction on the expanded features. This network is capable of improving separability for the multi-label data owing to the two-layer transformation while reducing the expanded feature space to a more manageable amount. This balances the input dimension which leads to a better classification performance even for a limited amount of data. Since this novel network does not exist in the single-label domain, the single-label variation of the proposed network has been formulated simultaneously. All the developed models have been tested on ML datasets to establish their effectiveness. After increasing the separability of the multi-label decision space, the following chapter aims to simplify the decision boundary to some extent to ease the task of classification. Alongside, attempts to handle the problems of class imbalance and label correlation which were not dealt with in the previous contributions has also been made.

Chapter 5

This chapter proposes a binary tree of classifiers model for multi-label classification that aims to preserve label dependencies within the data and handle class imbalance. It also attempts to simplify the complex decision boundary by splitting the data appropriately.

A tree is built stepwise, where within each node, the input data is strategically split into two subsets for its subsequent child nodes, keeping the label correlations intact. Unlike other decision tree approaches, a best split is not aimed for since it might increase the computational complexity of the algorithm, instead an approximate split in the data is opted to serve a similar purpose in lesser time. A novel approach of partitioning the data based on label-set proximity has also been proposed. It groups the data based on their Hamming distance in the decision space which keeps similar label-sets and co-occurring classes together. Various data appropriate classifiers are trained to learn the binary partition at every internal node. The tree of classifiers grows iteratively depending on two parameters – multi-label entropy and sample cardinality that are computed on the data at the current node. During training, the decision at any node is based on these parameters and the branching out is restricted, if deemed unnecessary. Specific classifiers at the leaf nodes perform the final classification task and assign appropriate label-sets to the unlabelled data. The proposed system aims to appropriately split the data and build the hierarchical structure such that the training and classification tasks become simpler. Also, the problem of class imbalance leads to the irregular splitting of data and excessive branching out of the tree which is handled through the novel use of suitable classifiers and parameters at the intermediate and leaf nodes. The proposed method has shown significant performance improvement on fourteen datasets against fourteen existing multi-label classifiers. Among the different problems handled in this work, the issue of class dependencies has been handled implicitly. It is an important characteristic of multi-label data that often is ignored. To extract more information from the multi-label data, the underlying label dependencies need to be specifically identified and explored further to incorporate more relevant information that can in turn improve the overall classification performance.

Chapter 6

Multi-label data deals with multiple classes associated with individual samples at the same time. This leads to the co-occurrence of several classes repeatedly, which indicates some existing correlation among them. In this chapter, the correlation among classes has been explored to improve the classification performance of existing ML classifiers. A novel approach of performing frequent label-set mining to extract correlated classes has been done. Both co-presence (CP) and co-absence (CA) of classes have been taken into consideration. The rules mined from the ML data has been further used to incorporate class correlation information into existing ML classifiers. The soft scores generated by an ML classifier are modified through a novel approach using the CP-CA rules. A concept of *certain* and *uncertain* scores has been defined here, where the proposed method aims to improve the uncertain scores with the help of the certain scores and their corresponding CP-CA rules. This has been experimentally analysed on ten ML datasets for few existing ML classifiers which shows substantial improvement in their overall performance.

Chapter 7

In the final chapter, the various contributions presented in this thesis are briefly summarized. The limitations of these studies are highlighted. Also, several open areas are enumerated, which present the scope of further extending the developed models discussed in this thesis.

1.4 Contribution

This thesis encompasses various approaches to solve four concerns with ML data. For a better understanding, here the specific novelty in the thesis, i. e., the chapter-wise contributions have been highlighted in this section.

Chapter 3 The first proposed work aims to handle the large input dimension and complex decision space through novel models incorporating autoencoders and MLELMs. The contributions are as follows.

- Building a cascade of networks with autoencoders (AEs) and MLELMs, where the AEs are used for dimensionality reduction and MLELM are utilized for improved learning.
- Incorporating stacked autoencoders (SAEs) in the main model and extending it to a deep version which utilizes deep autoencoders (DAEs).
- Stacking MLELMs to learn the mapping of the class scores obtained from the previous classifier to the hard labels.
- Optimizing the network configuration of the proposed model for varying depths of the network.
- Determining the better-suited component among DAE and SAE for dimensionality reduction.

Chapter 4 The second proposed work also aims to handle similar problems as the previous work. The large input dimension of ML data is handled through the use of autoencoders, whereas the complex decision space is dealt using functional link networks. This work unfolds stepwise.

- First, a novel multi-label functional link artificial neural network (MLFLANN) model is introduced that deals with the complex decision space of multi-label data by improving separability among classes.
- Then, various combinations of basis functions and learning techniques are explored to experimentally identify the optimal configuration of MLFLANN.
- Finally, the simple MLFLANN network is extended to a novel two-layer network based on MLFLANN and AE specifically for multi-label classification. Use of the

functional expansion layer followed by additional transformation by autoencoder layer improves class separability and reduces the feature dimension expanded by the first layer. This maintains a balance between the feature space and sample size, which leads to a good training of the classifier with limited data.

- Introducing the single-label variation of this novel two-layer network.

Chapter 5 The third work attempts to deal with the complex decision space, class imbalance and label correlation issue of ML data through a binary tree of classifiers model. The contributions are as follows.

- A tree of classifiers for multi-label classification that utilizes suitable classifiers at the intermediate and leaf nodes to handle various bottlenecks.
- A novel label-space partitioning technique to implicitly handle the underlying class correlations in the data.
- Approximate splitting of data to achieve faster convergence and simplifying the decision space through broad partition that makes the problem of complex boundaries and class imbalance less prominent.
- Explicitly handling imbalanced classes that cause uneven splitting of data through the use of appropriate classifiers at the intermediate and leaf nodes. Building the tree based on parameters that facilitate restrictions to prevent its unnecessary branching for smaller imbalanced classes.

Chapter 6 The final work specifically identifies the label dependencies that exist in the ML data through association rule mining and utilizes them to improve ML classification performance. The main contributions are:

- Introduce the concept of "frequent label-set mining" for finding class/label correlations. This identifies co-presence (CP) and co-absence (CA) among classes to generate rules for relevant and irrelevant label-sets.

- Novel computation to improve classification score for any classifier by incorporating class correlations with the help of frequent label-set mining.

Each of the individual models proposed in the four chapters target some of the identified problems of ML data. In some cases the outcome is better than the other, however, each of the works are a novel attempt to deal with the bottlenecks through the construction of adaptation-based classifiers.

1.5 Organization of the Thesis

The brief description regarding the organization of the thesis is given here.

Chapter 2 gives the relevant background before discussing the proposed works. First, the literature survey (Section 2.1) is discussed based on the approaches taken in literature, namely, data transformation, problem adaptation and ensemble approaches. These encompass the relevant literature for all the proposed works. Only multi-label algorithms have been explored in this chapter to keep the literature survey brief and relevant. Next, in Section 2.2 all the ML datasets that have been used for experimentation have been discussed. The label-set partitions have been shown to give an idea regarding the existing imbalance in the data. Finally, the various ML performance metrics (Section 2.3) have been briefly discussed for a better understanding in the later chapters.

Chapter 3 is the first contributory chapter that begins by introducing the problems (Section 3.1) and the proposed works briefly. This work is based on autoencoders and extreme learning machines, so Section 3.2 discusses the basics of AE and ELM networks. This is followed by the proposed works, Section 3.3 describes the stacked AE and MLELMs based model and Section 3.4) describes the deep ML classification network autoencoders and stacked MLELMs model, each of which contain the model description and elaborate experimental analysis. Section 3.5 concludes the chapter.

Chapter 4 proposes the second contributory work beginning with Section 4.1 giving an introduction to the problem. This work is mainly based on FLANN, hence Section 4.2 gives the preliminaries of the FLANN network. Section 4.3 discusses the first proposed work, the multi-label FLANN model. Section 4.4 discusses the optimization of the MLFLANN model and Section 4.5 proposes the autoencoder-integrated MLFLANN model. Each of these proposed work sections has an elaborate description of the model architecture and in-depth experimental analysis. Section 4.6 concludes the chapter.

Chapter 5 proposes a tree of classifiers model, where Section 5.1 introduces the problem, Section 5.2 discusses the proposed model in detail. This section gives the model description with elaborate details about building of the tree and the methodology and also an experimental analysis. Section 5.3 concludes the chapter.

Chapter 6 proposes a frequent label-set mining and association technique to improve ML classification performance. Section 6.1 is an introduction to the problem, Section 6.2 discusses the proposed work in detail with the model description discussing the steps of the algorithm individually, and then an elaborate experimental analysis. Finally Section 6.3 concludes the chapter.

Chapter 7 is the concluding chapter of the thesis that highlights the chapter-wise conclusions in Section 7.1, discusses some findings in Section 7.2, and the limitations and future scope in Section 7.3.

Chapter 2

Background

Multi-label classification in itself is quite an extensive area which is being thoroughly explored in recent years. In this chapter, a few relevant areas and background have been discussed, to have a better understanding of multi-label classification and the proposed methods in the following chapters. Firstly, a brief literature survey highlighting the popular and state-of-the-art research in the field of ML classification. Most of these algorithms have been used for comparative analysis in the later chapters. This is followed by an in-depth description of the multi-label datasets that have been used for the experiments of the proposed models. Finally, the performance metrics specific to multi-label data have been described in detail.

2.1 Literature Survey

Multi-label classification is being worked with by researchers mostly in the last decade. Looking into the literature, various interesting algorithms being frequently developed by researchers for efficient multi-label classification. In this thesis, the focus has been kept mainly on the popular multi-label algorithms that have been developed over the years and are still being used for comparison in the current literature. Alongside, some recent

algorithms that are relevant to this thesis and the proposed methods have also been included in this chapter. Generally, the literature survey for multi-label classification is presented by researchers based on the various approaches used. This is mainly due to the evolution of the ML classification techniques used over the years. Initially, when multi-label data was being considered to be dealt with alongside single-label and binary data, the simplest approach was opted, i.e., using the existing classification algorithms. This meant modifying the multi-label property of the data to convert it to single-label/binary such that it can be used with the traditional algorithms. This approach was named data-transformation. Since these approaches might have led to some loss of information, another variant of algorithms were developed by researchers that became quite popular. Here, instead of modifying the data, the algorithms were changed to adapt to the ML property to handle the data as is. This was the problem-adaptation approach. Finally, with the popularity of ML classification using these previous approaches, researchers developed ensemble models like in traditional classifiers to improve the efficiency of the ML models. In this way, a gradual evolution of ML classifiers is seen in the literature, and hence, the literature survey in this thesis has been presented based on the approaches chosen by researchers. Thus, as already discussed, there are broadly three categories (Figure 2.1), namely, data transformation, problem adaptation and ensemble approaches. Each of these approaches and their relevant algorithms have been briefly discussed in the following sections.

2.1.1 Data Transformation Methods

Data transformation was an initial approach opted by researchers that involved converting the original multi-label data into one or multiple simpler datasets. These were then used with traditional binary/single-label classifiers. In a certain way, these methods act as a preprocessing phase, producing new datasets from the original ones. These methods can also be thought to be tampering with the original data while altering the unique multi-label property of the data. These techniques either lead to loss of information

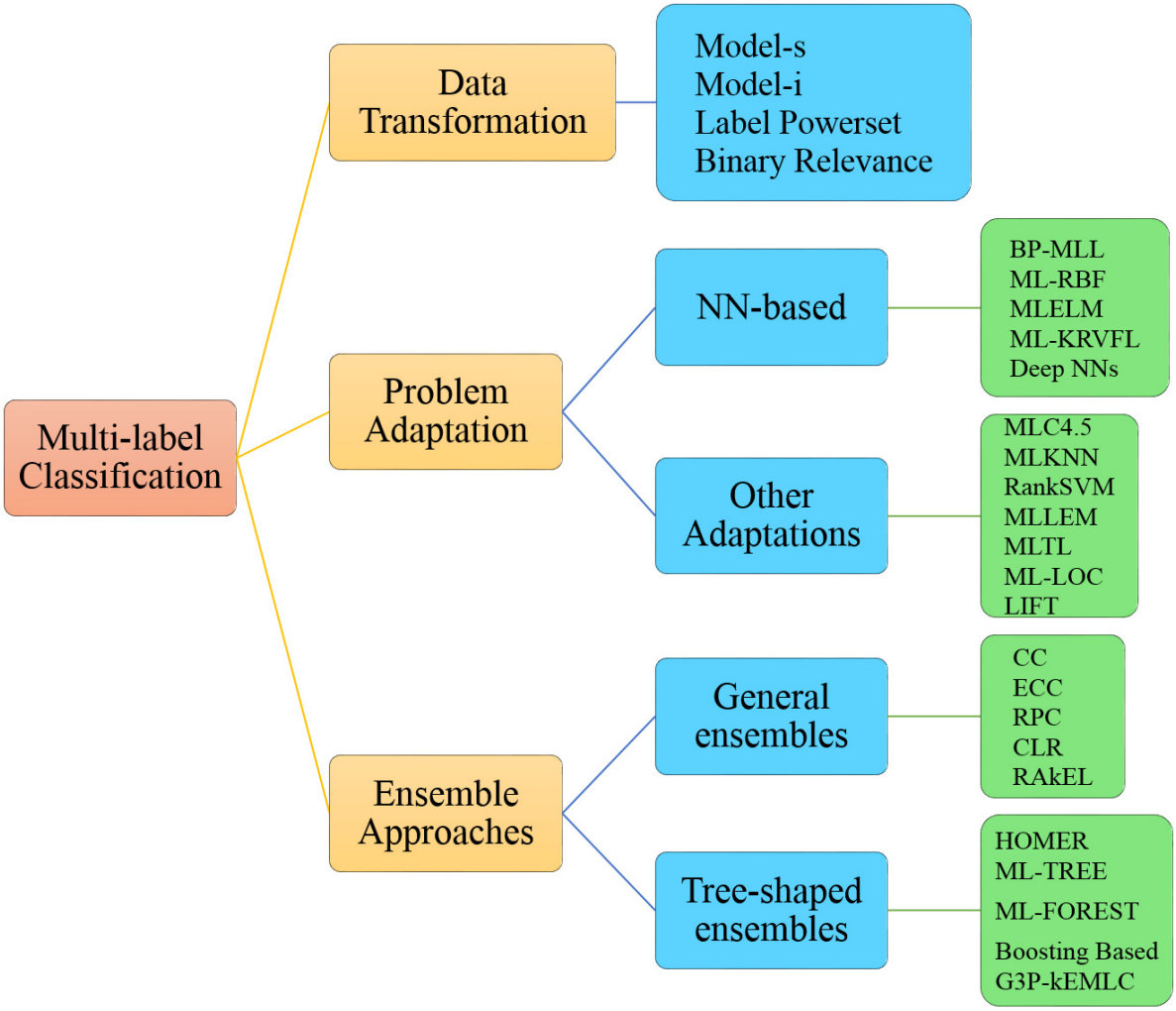


Figure 2.1: Approaches and methods of Multi-label classification

which eventually increases misclassification, or they include a lot of redundancy in the data, thus unnecessarily increasing the computation complexity. Some popular data transformation techniques have been discussed here.

Model-s and Model-i – In [7], the authors developed few basic algorithms to forcefully convert the multi-label data into single-label. Here s in Model-s stands for single-label. Multi-label data has C number of classes each represented by 0 or 1 for irrelevant and relevant labels respectively. Model-s retains the information for only one class, i.e, any one of the classes remain 1 while the rest are changed to 0. This modification can be based on some heuristics like retaining the class which has maximum instances, etc. This

is a pretty naïve approach and is not feasible for real-life classification.

Model-i on the other hand chooses to "**i**gnore" all the instances that have more than one labels and only retain the ones with a single label. This method chooses to remove all multi-label instances, thus is not appropriate to be termed as a multi-label algorithm.

Label Powerset (LP) – Similar to Model-s and Model-i, there is a Model-n also known as Label powerset (LP) [7]. This model encodes the multi-label information to single-label without completely losing it. Here, each unique label-set of the multi-label data is replaced by a unique single-label thereby, converting the multi-label classification to single-label classification. For a C-class data, a maximum of 2^C unique label-sets are possible (including all zeros as a unique label-set), however, all of these 2^C label-sets may not occur in the multi-label dataset. Each unique label-set is replaced by unique single labels. Thus C multi-label classes can be replaced by a maximum of 2^C single-label classes. Thus, the larger the initial C number of classes, the larger will be the corresponding SL classes.

Binary Relevance (BR) – One of the most popular approaches is binary relevance (BR) [24, 101]. It is a practical approach that splits the multi-label class information into multiple binary classes. For C number of classes, the data classified C number of times, once for each class. It is a one-vs-all concept that creates C number of independent classifiers, and each classifier works on the same number of input instances in the original ML data, but the i^{th} classifier positively labels the i^{th} label and others as negative. Thus each classifier is trained with C_i vs rest binary data. This has received much popularity but it succumbs to certain inconveniences. It dismisses any label correlations that might exist within the data since it handles each class independently. Also, the already imbalanced multi-label classes create highly imbalanced binary class partitions which creates a biased training of the classifier. Although BR is a basic approach, it still achieves comparable results for various loss functions. There are other variations of

the basic BR algorithm that utilizes the strengths of BR. Recently, in [92] the authors combined strengths of RankSVM (discussed in Section 2.1.2.2) and BR. It handles the problem of error accumulation of RankSVM and class imbalance issue in BR by creating a robust low-rank learning (RBRL). It learns to minimize ranking loss like in Rank-SVM and Hamming loss like BR.

[63] gives an extensive experimental analysis of methods based on binary transformation from multi-label data. The authors group their study into one-round, stacking and ensemble approaches. They have included BR, classifiers chains (CC), ensemble of classifier chains (ECC) (discussed in Section 2.1.3.1) along with other binary transformation classifiers. It highlights the influence of base classifiers such as support vector machines (SVM), random forest (RF), eXtreme Gradient Boosting (XGB), etc on the final performance of these techniques. Similarly, other data transformation approaches have also been developed over the years, however, the focus of this thesis is towards the next category of multi-label classifiers.

2.1.2 Problem Adaptation based Classifiers

Problem adaptation based classifiers, contrary to data transformation methods, mainly adapt the algorithm to the ML data. They modify traditional classifiers in a way that they can tackle the original multi-label data without tampering with it. It aims to retain the multi-label property of the data without having to convert them to single-label or binary. Most of the problem adaptation approaches rely on modifying traditional algorithms based on trees [13]), neural networks [103], instance-based learning [104], etc. In [97], various popular classifiers for multi-label data (discussed in this section) have been tested experimentally. Studies like this help to analyse the performance of different methods under varied scenarios for multiple datasets and metrics. This particular study highlights the dependance of performance on the size of the datasets, correlation between the base classifiers and the entire model, and a trade off between performance and training

time. In this section, some popular problem adaptation models have been discussed. Additionally, some of the proposed methods discussed in the following chapters deal with neural networks, hence few models from the sub-category has been highlighted.

2.1.2.1 Neural Network based adaptations

Neural networks are one of the most popular machine learning tools that have gained a lot of attention over the years [27]. Today, various types of networks exist and are being continuously adapted in numerous domains to perform efficient classification. Similarly, while exploring the literature of multi-label classification, it is seen that researchers have developed different neural network based techniques by tailoring the original models to suit the problem at hand. However, in comparison to single-label NN based models being developed, there are very few multi-label NN models. Here, few popular and recent NN models for multi-label classification have been discussed.

Backpropagation for multi-label learning (BP-MLL) – One of the first neural network based model developed for ML classification is backpropagation based multi-label learning (BP-MLL) [103]. It is an adaptation of the traditional multi-layer perceptron with a modified error computation technique, that is more suited for multi-label data. It takes into consideration the selection of multiple labels at the output layer. The two-layer feed-forward network has been trained using backpropagation with a cost function that incorporates the ranking of labels. The appropriate number of hidden neurons needed in that single hidden layer is found experimentally. In the end, a unique threshold for each testing instance is calculated based on the processing of the training set. The performance of BP-MLL depends on the configuration of the network and its parameters. Also, a substantial amount of time is required for training the network.

Multi-label radial basis function network (ML-RBF) – ML-RBF [100] is an adaptation of radial basis networks for ML classification which incorporates neighbourhood information. It executes K-means clustering and uses the cluster centres in the

RBF model. There is no way to know the distribution of the dataset beforehand, thus, K-means may not always cluster the data well. The two-step process also increases the computational complexity of the problem. There are other variations of ML-RBF [105] that exist in the literature.

Multi-label Extreme Learning Machine (MLELM) – Extreme learning machines (ELM) are simple two-layer feed-forward networks that have a one pass training phase. It does not use backpropagation for learning, instead learns the weight matrix through a pseudo-inverse. The most simple form is referred to as multi-label ELM (MLELM), which is an adaptation of the simple ELM network. These models do not require any architectural change, but the output provided at the last layer is multi-label in nature. This ELM classifier has been adapted by authors in [81] to develop a multi-label extreme learning machine (MLELM) for fast classification. They used the MLELM with a bipolar representation of labels to improve the learning of the simple ELM. Authors in [74] develop a model termed ELM-ML using two consecutive ELMs. The first MLELM is trained as a multi-label classifier with multiple class output nodes. The second ELM is described as a sample-wise threshold function with a single output node. The threshold determined by this ELM is then used on the outcome of the first ELM to generate the final output. Canonical Correlation Analysis based ELM (CCA-ELM) was introduced in [43]. Correlations among the input features and the set of labels are computed using CCA, then the input space and label space are mapped to the new space. An ELM is used to classify and finally map the original input space. L_{21}-norm Minimization ELM [36] is an ELM based algorithm which combines the smallest training error of ELM with L_{21}-norm minimization of the hidden to output layer weight matrix. In [105], the authors proposed a multi layer ELM-RBF for multi-label learning (ML-ELM-RBF). It is built from radial basis function for multi-label learning (ML-RBF) and weight uncertainty ELM-AE (WuELM-AE). This model stacks WuELM-AE to form a deep network, and then it performs clustering analysis on sample features of each possible class to compose

the last hidden layer.

Multi-label Random Vector Functional Link Network (ML-KRVFL) – Random Vector Functional Link Network (RVFLN) [53] is a shallow feed-forward network that uses non-iterative learning. Nodes from the input layer are connected to the output layer in two ways. One is a direct connection and the other is connected through some enhancement nodes. These enhancement nodes are similar to the hidden nodes in a simple neural network. The weights in the network are learnt through the computation of pseudo-inverse. Various applications have been seen in the single-label domain using this network. However, very few multi-label models based on RVFLN exist in literature. The multi-label adaptation of this network in [10] creates a kernelized version of the traditional RVFL (ML-KRVFL). Here, a kernel function replaces the enhancement layer that provides more stabilization to the network and reduces randomness.

Deep neural networks Among the neural network based approaches, deep learning models are slowly finding application in the field of multi-label data. Owing to the complex nature of ML data, deep models are capable of extracting underlying information. However, lack of labelled data leads to lesser exploration of this branch of approaches. Deep models mostly find application in real-life multi-label image classification problems and domains like medical image processing, emotion recognition, etc. To deal with images, most researchers explore convolutional neural networks (CNNs) for multi-label image classification. In this context, [8] have developed a CNN-based network called Multi-ECGNet that analyzes ECGs and identifies patients who are suspected to multiple cardiac ailments simultaneously. The proposed model embraces benefits of CNN and residual networks to detect 55 symptoms of heart diseases. In [2], proposed a bidirectional network that is capable of extracting high-level features and labels from the data. There are multiple stacked pooling layers that identify and combine pairs of low-level features to create higher level features. These in turn compose the next pooling layer and so on. This approach helps to propagate association in both features and labels. In

[88] hierarchical multi-label classification networks (HMCN) were proposed that would tree-structured hierarchies. They adopt a hybrid strategy that can optimize both global and local loss functions in parallel while penalizing hierarchical violations. There is one output layer per hierarchical level of the data and an additional global output layer. Both recurrent and non-recurrent neural networks have been utilized to build the model. A feed-forward version, HMCN-F and a recurrent version HMCN-R were developed in this work as well. In [85] a label graph superimposing framework was proposed to improve the performance of graph convolutional network (GCN) and CNN developed for ML recognition. A label graph model is created to capture the label relationships that exist in the data and a lateral connection (LC) operation is presented to inject GCN embeddings into CNN features, thus creating the proposed Knowledge and Statistics Superimposing Network (KSSNet). The LC operation aids the model in learning label-anchored feature representations by acting as label-feature correlation modelling. There are not too many deep learning models in the ML domain, and they are slowly being developed with the availability of diverse data.

2.1.2.2 Other popular adaptations

Apart from the above-mentioned categories, there are other well-known classifiers that have been adapted in numerous ways to perform multi-label classification.

Multi-label C4.5 – Decision tree-based techniques have been widely used for single-label classification [1], but have not been extensively explored in the ML scenario. Decision-based building of the tree structure is quite simple and robust, which makes it an apt candidate for ML data. In general, it is seen that decision tree (DT) based techniques mostly vary on the way the data is split in each intermediate node. Some popular tree-based methods have been included here. ML C4.5 [11] is one of the first adaptations of decision trees for multi-label classification. Traditional C4.5 was modified to handle multiple labels at the same time. It performs multi-label gene expression data classification.

Here, the splitting of the tree is based on a modified entropy measure to incorporate the irrelevant class probability as well. The leaves of the tree indicate label-sets instead of single classes which performs the final classification.

Multi-label k Nearest Neighbour (ML-kNN) – Among the instance-based multi-label classification algorithms that exist in the literature, ML-kNN [104] is one of the best-known algorithms. It is a lazy learning technique and is a multi-label adaptation of the traditional k-nearest neighbours classifier. It primarily works on the principle of MAP (maximum a-posteriori) while incorporating first and second-degree neighbourhood information for ML classification. It internally works as a binary relevance classifier, since a separate set of apriori and conditional probabilities are independently computed for each label. It involves a large number of computations in the second-order neighbourhood of each training pattern before the actual classification is done. For a dataset with a huge number of samples, this method would be computationally expensive and quite slow. Recently, in [58] a generalized version of prototype weighting has been proposed to improve the performance of the kNN classifier for multi-label data. It aims to minimize error rate and optimize the F-measure metric. They aim to provide smooth boundaries which would improve the classification capability especially for overlapping boundaries.

RankSVM – In [21], authors develop a method based on the support vectors machine (SVM) principle. This aims to incorporate label correlations through an approximation of the Hamming loss metric. The SVM based model generates a ranking for the classes and the final label-set is predicted using a thresholding function. Although Rank-SVM considers label correlations, its performance is not exceptionally enhanced as compared to other binary classification models. Simplified constraints RankSVM (SC-RankSVM) [83], as its name suggests, is a modification of the RankSVM with milder constraints to minimize the ranking loss metric and have a large margin.

MLLEM – In [41], ML classification using Laplacian Eigen Map (MLLEM) has been proposed as a non-linear embedding technique that incorporates instances and labels in the same lower dimensional space. It considers instance-instance, label-instance and label-label relationship in the data. For instance-instance relationship, instances that are similar in the feature space are closer in the embedded space. For label-label relationship, co-occurring label pairs are closer and for instance-label relationship, the actual associations mapped in the data are represented in the embedded space. This method attempts to provide a simultaneous visualization for the patterns as well as the labels. It provides both linear and non-linear mapping.

Multi-Label Tomek Link (MLTL) – Multi-Label Tomek Link (MLTL) [55] is a recent work using Tomek link approach that handles the class imbalance problem of multi-label data. It is a resampling based approach that has been used with existing algorithms to improve their performance. It has been applied with algorithms like binary relevance, label powerset, HOMER, CC, ML-kNN, RakEL. The Tomek-link for single-label data has been adapted to handle multi-label data by defining the difference between label-sets. It can be applied as an undersampling method where it removes samples to reduce imbalance or as a post-processing clean-up step.

ML-LOC – In [33], authors performed multi-label learning by exploiting the label correlations locally (ML-LOC), where a local correlation (LOC) code is determined and used as additional features. The code vector for each instance encodes the influence of class correlations with an assumption that similar instances have similar label-sets. This similarity is measured in the label space. It algorithm uses the local correlation sensitivity and global discrimination fitting into a single framework.

LIFT – ML learning with label-specific features (LIFT) [102] conducts clustering analysis on positive and negative instances and constructs features that are specific to the individual classes. k-means algorithm is used to determine the cluster centres which rep-

resent the structure of the training data. It then queries the clustering results to perform training and testing of the classifier. To eliminate the effect of class imbalance, LIFT gives similar importance for both the positive and negative data with respect to their cluster sizes.

2.1.3 Ensemble Approaches

Along with the evolution of stand-alone classifiers, the concept of ensemble of classifiers have also proven to be quite efficient in the field of ML classification. These ensembles are built from the basic data transformation models or the problem adaptation models, hence it may or may not be considered as a completely distinct category. Some popular ensemble techniques in the literature have been mentioned with a separate focus on tree-shaped ensembles, since this is more relevant to one of the proposed approaches.

2.1.3.1 General Ensembles

There are few popular ensemble techniques for ML data that have been used extensively over the years.

Classifier Chains (CC) – Classifier Chains (CC) [61, 62] is one of the best-known ensembles of binary classifiers for multi-label classification.It is seen as an improvement over binary relevance (BR) where it attempts to take into account the label information at the same time. It also uses the same number of classifiers as in BR, i.e., same as the number of classes, but chosen at random order. It uses C classifiers for the C individual classes while incorporating the class information along with the features to have a better classification. The first classifier is trained using only the original input attributes. The first output label is then added as new input attribute, and the new input space is used to train the second classifier, and so on. This forms a chain of classifiers which incorporates label information. The final outcome is dependent on the order of training of the classifiers. Probabilistic Classifier Chains (PCC) [18] is an extension of CC that

incorporates Bayesian methods to optimize the order of chains, thus aiming to improve the overall performance. An ensemble extension to CC, known as the Ensemble Classifier Chains (ECC) [62] the authors aim to handle the drawbacks of CC by introducing an ensemble model. It is used to reduce the error propagating through the CC model. This is done by several classifier chains trained with a random ordering of labels and subsets of training instances for ML classification. ECC, like CC, also utilizes class information. While CC is an ensemble of binary classifiers, ECC can be thought of as an ensemble of multi-label classifiers.

Another adaptation of CC is the label specific features based classifier chain for multi-label classification (LSF-CC) [89]. It is a very recent modification of CC to handle some drawbacks like random label ordering and noises in the original and additional features.

Ranking by pairwise comparison (RPC) – Unlike the previously discussed ensembles, ranking by pairwise comparison (RPC) [35] is a one-vs-one (OVO) approach where every pair of classes has a binary classifier. This increases the number of classifiers to $\frac{C(C-1)}{2}$, which might be a bottleneck if the data has a large number of classes. Linear perceptron models were used as the base classifiers for RPC. The job of each OVO classifier is to individually identify the rank among the two classes, and later all the rankings are merged to generate the label ranking of the entire data.

Calibrated Label Ranking (CLR) – Another ensemble of binary classifiers, calibrated label ranking [22] was developed as an extension of RPC. CLR includes the concept of a virtual label along with the actual labels in the dataset. This virtual label works as a threshold that helps to calibrate the final classifier to separate the relevant and irrelevant labels. The classes that rank above the virtual label are considered in the final label-set.

Random k-labelsets (RAKEL) – Random k-labelsets (RAKEL) [78, 79] is another popular ensemble technique that employs multiple single-label classifiers to perform multi-

label classification. It aims to consider the existing label correlations within the data. It generates random subsets of labels and trains a single-label classifier for each subset. The working of RAKEL is also quite similar to BR, the number of classifiers is the same as the number of subsets. The number and length of subsets can be altered as per requirement. The base classifier used for RAKEL can be changed as per requirement, but the original RAKEL model uses label powerset. [84] modifies the RAKEL algorithm to improve the concept of randomized label subsets by utilizing the active learning model. A selection criteria is proposed that evaluates separability and balances the level of classes. The first class is chosen randomly and the following labels are selected based on the active learning concept.

2.1.3.2 Tree-shaped ensembles

This type of ensemble technique splits the data into partitions, to form a tree-like structure and uses multiple classifiers as an ensemble to perform multi-label classification.

Hierarchy of multi-label classifiers (HOMER) – Hierarchy of multi-label classifiers (HOMER) [77] splits the classes into several groups such that at the end each leaf node corresponds to a particular class. Multiple label powerset (LP) classifiers are used at intermediate stages to learn the subset of classes. This model is specifically designed to handle the large number of classes in ML data.

ML-TREE and ML-FOREST – ML-TREE [94] is another tree-based algorithm that splits the data into multiple branches using one-vs-all SVM for each class at intermediate nodes. The data is thought of as a hierarchy and thus it is split into multiple partitions like in a hierarchical tree structure. At the final level, each leaf node contains a set of classes which provides the predicted labels. ML-FOREST [93] is an ensemble of ensembles where multiple ML-TREEs are combined. Both of these approaches use SVMs as there base classifier to train the data while building the tree structure.

Boosting based – AdaBoost is a boosting based decision tree model which helps to improve the performance of multiple weak classifiers. AdaBoost.MH [68] is a multi-label version of the boosting model. In [69], the authors proposed the BoosTexter approach which is also a multi-label extension of the popular ensemble learning approach, AdaBoost. Alternate decision trees (ADT) are a generalization of traditional DT models. It modifies AdaBoost models to propose an alternative way to using techniques like boosting to improve their performance. Similarly, ADTBoost.MH [13] employs a one-vs-all strategy to split the instances and then trains multiple ADTs on these data to perform ML classification.

G3P-kEMLC – In [50], a tree-shaped ensemble of ML classifiers has been built which is based on grammar-guided genetic programming (G3P-kEMLC). Each of these classifiers model dependencies among a sub-group of k labels. The predictions from the children nodes are aggregated at each node, while the leaf nodes contain a classifier to perform final classification.

A comparison among some of the popular ML algorithms has been shown in Table 2.1 stating their advantages and disadvantages. Most of the algorithms discussed in this chapter have been relevantly used for comparative analysis with the different proposed methods on various multi-label datasets.

2.2 Datasets

For all the experimental analysis throughout the thesis, benchmark multi-label datasets have been used. These datasets are available at multiple sources.

- http://www.uco.es/kdis/mllresources/
- http://mulan.sourceforge.net/datasets-mlc.html
- https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multilabel.html

Table 2.1: Comparison of some popular ML methods

Method	Description	Advantage	Disadvantage
Label powerset (LP) [7]	Unique label-set transformed to a new class	Simple model	Large number of classes
Binary Relevance (BR) [24]	Binary classifier for each label	Simple model	Ignores label dependencies
Classifier Chain (CC) [61]	Extension of BR	Includes class dependencies	Performance depends on the order of classifiers in the chain
Ensemble of Classifier Chains (ECC) [62]	Extension of CC	Chain ordering is less likely to negatively affect performance	Potentially large redundancy in learning space
Ranking by pairwise comparison (RPC) [35]	Ranking by pairwise comparison method	Handles class imbalance issue	Number of classifiers = C(C-1)/2
Calibrated Label Ranking (CLR) [22]	Extension of RPC	Additional virtual class concept improves performance	Large number of classifiers
Hierarchy Of Multilabel classifiERs (HOMER) [77]	Tree-based method that uses LP as its base classifier	Computationally efficient for datasets with a large number of labels	Additional parameter to tune, i.e. the number of clusters
RAndom k -labELsets (RAkEL) [78]	Extension of the label powerset method	An improvement over LP for a large number of labels and training examples	Random nature may include models that affect the ensemble in a negative way
Multi-label kNN (MLkNN) [104]	Adaptation of the traditional kNN with second-order neighbourhood information	Works well with small data	Large computational complexity

Experiments have been done using MATLAB 2017a on a Windows OS with Intel Core i7 processor and 16GB RAM.

2.2.1 Description of Datasets

Details of the benchmark datasets has been given in Table 2.2. They are available in pre-processed forms, which have been used after normalization for the experiments. The table contains domain names of the data with the number of instances, features and classes. It also has label cardinality (Label Card) which indicates the average size of

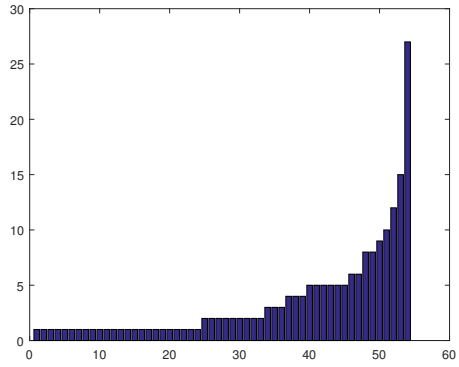
the label-set, i.e., the average number of relevant labels per instance. Diversity indicates number of unique label-sets of the data divided by the number of possible label-sets. Multi-label datasets inherently have the issue of imbalanced classes. To provide a better understanding of the class imbalance present in the data, Table 2.2 contains maximum and mean imbalance ratios as MaxIR and MeanIR respectively [28].

Table 2.2: Dataset details

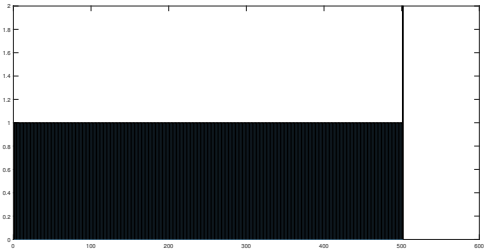
Dataset	*Domain*	*Reference*	*Instances*	*Features*	*Classes*	*Label Card*	*Diversity*	*MaxIR*	*MeanIR*
Flags	Image	[23]	194	19	7	3.392	0.422	5.88	2.25
CAL500	Music	[80]	502	68	174	26.04	1.000	88.80	20.58
CHD49	Medicine	[71]	555	49	6	2.58	0.531	26.38	5.77
Emotions	Music	[76]	593	72	6	1.869	0.422	1.78	1.48
Water Quality	Chemistry	[6]	1060	16	14	5.073	0.778	2.84	1.77
GNegativePSEAAC	Biology	[96]	1392	440	8	1.046	0.074	69.63	18.448
Enron	Text	[60]	1702	1001	53	3.378	0.442	913.00	73.95
Image	Image	[104]	2000	294	5	1.236	0.625	1.42	1.19
Scene	Image	[7]	2407	294	6	1.074	0.234	1.46	1.25
Yeast	Biology	[21]	2417	103	14	4.237	0.082	53.41	7.20
HumanPSEAAC	Biology	[96]	3106	440	14	1.185	0.027	46.41	15.289
Slashdot	Text	[59]	3782	1079	22	1.181	0.041	194.67	19.462
Corel	Image	[19]	5000	499	374	3.522	0.635	1120.00	189.57
Bibtex	Audio	[38]	7395	1836	159	2.402	0.386	20.43	12.50
Yelp	Text	[67]	10810	671	5	1.638	1.000	7.57	2.88
Delicious	Text	[77]	16110	500	983	19.02	0.981	309.29	71.13
Eurlex	Text	[47]	19350	5000	201	2.213	0.129	4290.00	536.98

Additionally, the frequencies of unique label-sets have also been plotted for all the datasets to visualize the imbalance in the data. Here, each unique label-set is considered as a distinct class and the histograms for the datasets have been given in Figure 2.2, 2.3 and 2.4. The X-axis for all the plots are the unique label-sets and the Y-axis is the number of data points belonging to that label-set. The datasets have also been marked as small, medium and large based on the total number of instances in the dataset.

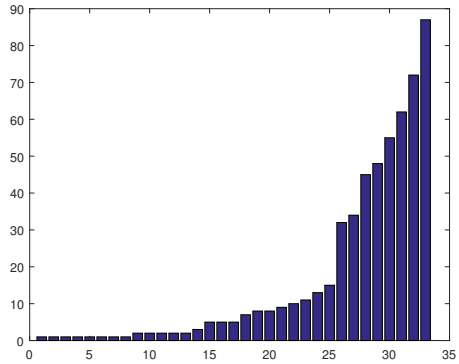
From these plots it is seen that all these ML datasets are highly imbalanced. CAL500 (Figure 2.2b) has a single data from each label-set, whereas delicious dataset (Figure 2.4d) has almost 16000 unique label-sets mostly with one or two data points from each. Most of the datasets have very few samples from individual label-sets which make the task of multi-label classification difficult. The problem of imbalance has been handled in one of the proposed methods.



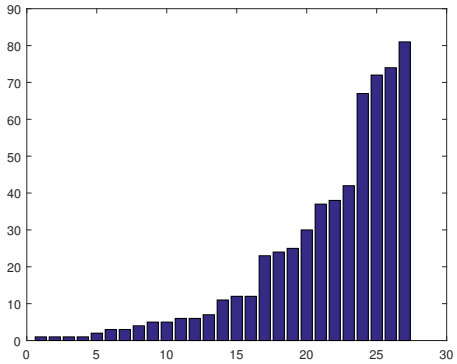
(a) Flags



(b) CAL500



(c) CHD49



(d) Emotions

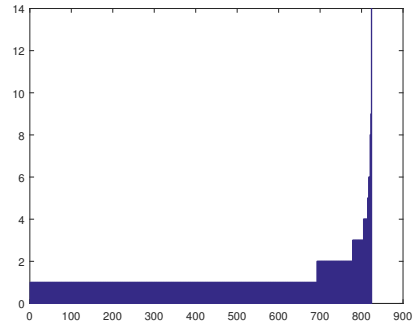
Figure 2.2: Histograms of small datasets

2.3 Performance Metrics

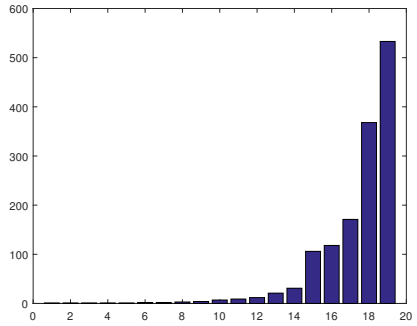
While computing performance of multi-label classifiers there is a need of separate performance metrics from those used for traditional single-label data. These should be able to handle the occurrence of multiple labels in the data. They are broadly divided into few categories as follows. Each of these metrics are computed using the actual labels Y and the predicted labels Z.

2.3.1 Example-based Metrics

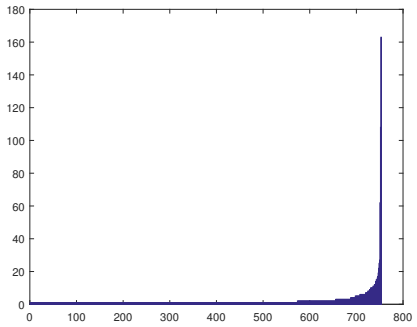
These scores are calculated separately for each sample, and then an average score is computed. This indicates that each sample has equal weight, irrespective of whether it belongs to a rare or frequent label-set.



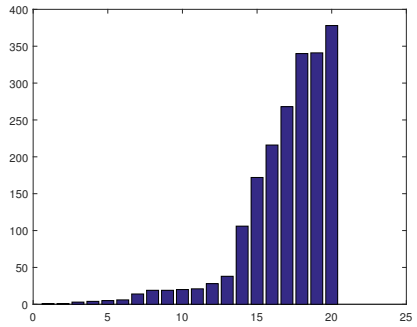
(a) Water Quality



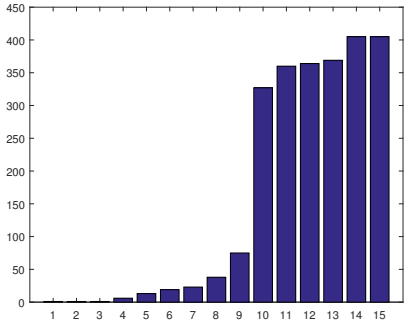
(b) GNegativePseAAC



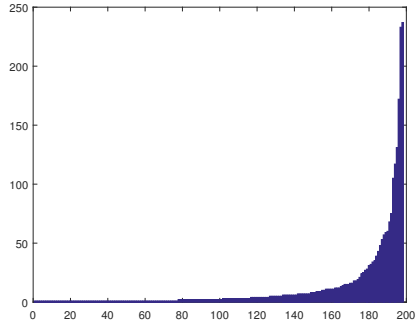
(c) Enron



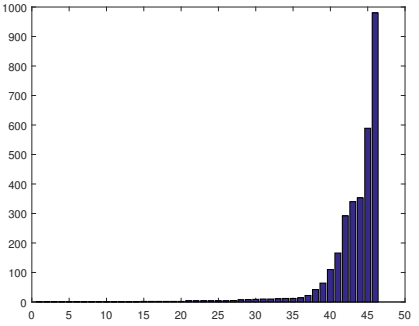
(d) Image



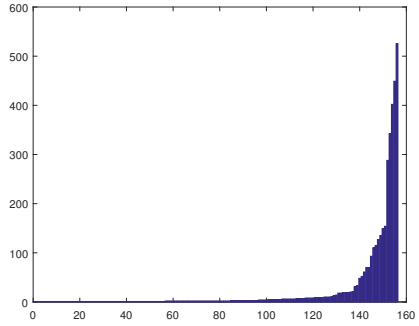
(e) Scene



(f) Yeast

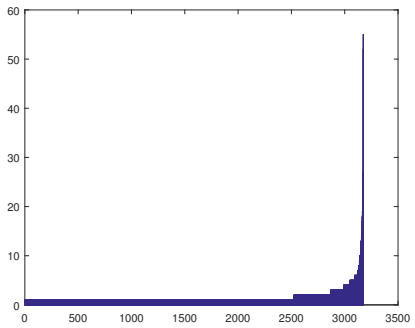


(g) HumanPseAAC

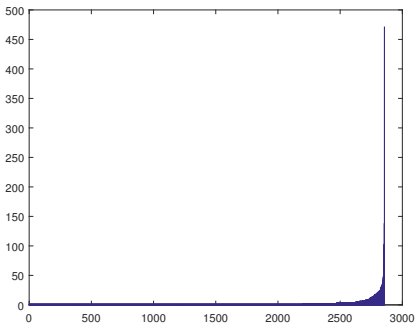


(h) Slashdot

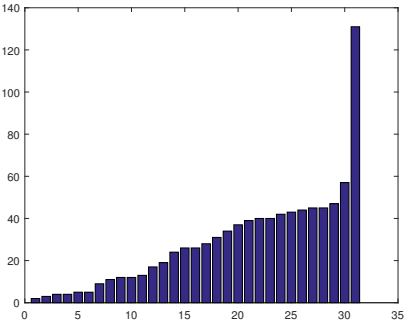
Figure 2.3: Histograms of medium datasets



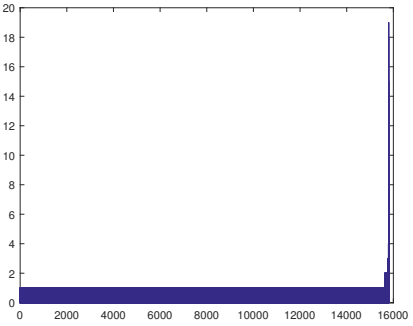
(a) Corel



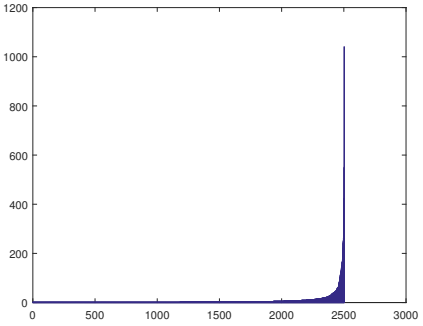
(b) Bibtex



(c) Yelp



(d) Delicious



(e) EUR-Lex

Figure 2.4: Histograms of large datasets

2.3.1.1 Hamming Loss

It is a popular metric widely used by researchers to evaluate ML classifiers.

$$\text{Hamming Loss} = \frac{1}{N}\frac{1}{C}\sum_{i=1}^{N}|Y_i \vartriangle Z_i| \tag{2.1}$$

where, $\vartriangle$ is the symmetric distance between Y and Z, N is the total number of instances and C is the number of classes. This score computes the Hamming distance between the two output vectors. It gives the number of wrong predictions with respect to the total number of classes.

2.3.1.2 Accuracy

It is computed as a ratio of the correctly predicted classes to the total number of relevant classes. Computation is done for every instance and then an average is made.

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{2.2}$$

2.3.1.3 Precision

It is the ratio between number of correctly predicted labels vs the total number of predicted labels.

$$\text{Precision} = \frac{1}{N}\sum_{i=1}^{N}\frac{Y_i \cap Z_i}{Z_i} \tag{2.3}$$

2.3.1.4 Recall

It is the ratio between number of correctly predicted labels vs the total number of relevant labels.

$$\text{Recall} = \frac{1}{N}\sum_{i=1}^{N}\frac{Y_i \cap Z_i}{Y_i} \tag{2.4}$$

2.3.1.5 F-Measure

It is a joint measure computed by the harmonic mean of precision and recall.

$$\text{F-Measure} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.5}$$

2.3.1.6 Subset Accuracy

It is a strict metric which does not consider partial correctness of predicted labels. This counts the number of completely correct predicted label-sets. This metric is quite difficult to achieve, and has lower possibility of correctness with a larger number of classes.

$$\text{Subset Accuracy} = \frac{1}{N}\sum_{i=1}^{N}[[Y_i = Z_i]] \tag{2.6}$$

2.3.2 Ranking-based Metrics

These metrics are based on the ranks of the score for each class. The scores are rearranged in a descending order based on their $rank(x, y)$ where x is the instance belonging to the class y.

2.3.2.1 Ranking Loss

It considers all combinations of relevant and irrelevant labels for an instance and checks the number of times an irrelevant label has ranked higher than a relevant label. Lower ranking loss, indicates better performance.

$$\text{Ranking Loss} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|Y| \cdot |\bar{Y}_i|}|y_a, y_b : rank(x_i, y_a) > rank(x_i, y_b), (y_a, y_b) \in Y_i \times \bar{Y}_i| \tag{2.7}$$

2.3.2.2 Coverage

This metric counts the number of steps in the ranking of classes that cover all the relevant labels. Lower the coverage, better the performance. This means that all relevant labels have the highest scores, or in other words, all classes with highest scores are actually relevant.

$$\text{Coverage} = \frac{1}{N}\sum_{i=1}^{N} \underset{y \in Y_i}{\text{argmax}} \langle rank(x_i, y) \rangle - 1 \tag{2.8}$$

2.3.2.3 One Error

This counts the number of wrong predictions for the top-most ranked classes. For each instance, this metric returns 1 if the top-ranked class does not belong to the actual label-set. Lower one error indicates better performance.

$$\text{One Error} = \frac{1}{N}\sum_{i=1}^{N} [[[\underset{y \in Z_i}{\text{argmax}} \langle rank(x_i, y) \rangle \notin Y_i]]] \tag{2.9}$$

2.3.2.4 Average Precision

For every label of an instance, this metric computes the amount of relevant labels that have predicted rank above it.

$$\text{Average Precision} = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|Y_i|} \sum_{y \in Y_i} \frac{|y' | rank(x_i, y') \geq rank(x_i, y), y' \in Y_i|}{rank(x_i, y)} \tag{2.10}$$

2.3.3 Label-based Metrics

Unlike the previous metrics which are calculated individually for each instance, the label-based metrics are computed for each label and then averaged by the number of labels. There are two techniques for averaging label-based metrics, macro-averaging and micro-averaging. Metrics like Precision, Recall and F-measure can be micro or macro averaged.

Here the F-Measure metric has been considered.

2.3.3.1 MacroF1

For macro-averaging, the FMeasure metric is evaluated once per class and then the values are averaged.

$$\text{Macro-F1} = \frac{1}{C}\sum_{c \in C} FMeasure_c \tag{2.11}$$

2.3.3.2 MicroF1

For micro-averaging, it computes all the parameters class-wise for the entire dataset and does the final computation only once.

$$\text{Micro-F1} = \frac{1}{C} * 2 * \frac{\sum_{c \in C} Precision_c * \sum_{c \in C} Recall_c}{\sum_{c \in C} Precision_c + \sum_{c \in C} Recall_c} \tag{2.12}$$

All the above described performance metrics have been used in the following contributory chapters for experimental analysis. Using a wide variety of metrics help to analyse the strengths and weaknesses of the different approaches in comparison with the existing ML techniques. Since the background is set, the next chapters describe the problems of multi-label data to be handled and the various approaches.

Chapter 3

Autoencoders and Extreme Learning Machines based Multi-label Classifiers

3.1 Introduction

In order to deal with the challenges encountered with multi-label data, various approaches are adopted by researchers. Since multi-label data belongs to more than one classes at a time, the corresponding class boundaries invariably overlap thus making the decision space quite complex. To handle this problem efficiently, many researchers prefer using artificial neural networks (ANNs) [27, 29, 49, 72] to learn the complex multi-label class boundaries. In literature, adaptations of multi-layer perceptron (MLP) [103], radial basis functions (RBF) [100], extreme learning machine (ELM) [43], deep neural networks (DNN) [91], etc have been done by researchers to classify multi-label data efficiently. From these works, it is seen that more complex architectures become computationally bulky, whereas extremely simple networks may not be able to work as desired. Thus,

while developing a classifier, efficiency and simplicity both need to be handled simultaneously. Keeping the above aspects in mind, in this chapter the focus is directed towards one such simple yet effective model known as an extreme learning machine (ELM) [31]. ELMs are quite compact and perform efficiently when dealing with single-label data. Hence, adaptations of this single layer model have been made by various researchers to perform multi-label classification [36, 81, 105]. However, it is seen that the performance of a stand-alone multi-label ELM (MLELM) relatively deteriorates when it comes to multi-label data classification. It is not able to approximate the weights for data with multiple outputs as efficiently as it can do for single-label data. The complex multi-label data prove to be bulky for the simple one-pass ELM network. This served as the motivation to build a model that utilizes the strengths of ELMs and handles their drawbacks to improve the overall performance in the field of multi-label classification.

In this chapter, a cascade of neural networks has been proposed which attempts to handle a few of the setbacks in the field of multi-label data classification. As mentioned previously, the focus of this model is directed towards ELMs. To strengthen the performance of ELMs, two specific issues faced due to the complexity of multi-label data has been handled using a couple of complementing networks. One challenge is to handle the dimensionality and representation of the input space for multi-label data. Simple ELMs have the requirement of a huge number of nodes in the hidden layer as compared to the input layer for efficient approximation of weights. This proves to be a bottleneck while using ELM for multi-label classification. Since the input dimension is often quite large, it drastically increases the number of weights to be learnt by the network in one pass, thus reducing the approximation capability of the network. To battle this problem, another simple yet efficient neural network, namely autoencoder (AE) [29], is used in the proposed model. Autoencoders are unsupervised networks that are capable of learning effective encoding of data. They produce a good representation of the actual data which makes the consecutive learning steps more functional. The other challenge faced is the effective mapping of the input to the complex output space. Due to this com-

plexity, the one-pass learning scheme of ELMs falters while learning all the overlapping relations among classes. To give this mechanism an additional scope of learning a novel and slightly modified MLELM in coalition with the original MLELM is introduced to effectively approximate the class scores. Utilizing the simplistic nature of ELMs and the effectiveness of AEs, a classifier system based on stacked AEs and ELMs is presented to perform multi-label classification efficiently.

The first proposed model is a cascade of stacked autoencoders and multi-label ELMs. It has three phases: feature extraction [25], soft classification [4], class score approximation. The first phase of the proposed model is feature extraction: this is performed using a stacked autoencoder. The large input dimension is thus transformed to a smaller, appropriate space which is used in the subsequent classification phase. In the classification phase of the model, a multi-label ELM is employed to predict the soft class labels for the data. The outputs generated in this phase is then fed to another MLELM in the final step. The last step is for class score approximation in which the second MLELM learns to map the soft class labels to their corresponding hard class labels. After the final class mapping is learned, a global threshold is calibrated to assign the hard class labels to the data. The three-step cascade of classifiers proposed in this chapter is a novel approach to handling a couple of the challenges faced by multi-label classifiers. This approach of stacking networks has been used in accordance with the concept of stacked generalization [90]. However, the main contribution lies in the technique of using an extreme learning machine to learn the mapping of the class scores obtained from the previous classifier to the hard labels. The proposed model has been validated with seven multi-label datasets along with comparative analysis against eleven relevant algorithms.

In the second approach, the above-proposed model has been further explored to create a deeper version. Using deep learning in the context of ML classification can be readily assumed to have increased efficiency compared to that of a simpler classifier. However, looking into ML literature, it is seen that deep neural networks are still being explored

and do not have sufficient work in this area [34]. Considering it as a scope to delve into this field, DNNs have been considered to handle the complex ML data. Among the various types of AEs that exist, namely deep AE (DAE), stacked AE (SAE), variational AE, etc, contrary to the SAE in the previous model, in this work, a DAE has been specifically considered, which can map complicated non-linear representation of the input feature set. This seems apt to handle ML data, which inherently have a large number of features and is difficult to manage. Integrating deep autoencoders and multi-label ELMs the proposed deep neural model is created for effective multi-label classification. The effort in this chapter has been to create a deep network that first performs feature transformation-cum-reduction and then performs consecutive ML classification. First, a DAE with multiple hidden layers is implemented for effective feature engineering. A large part of generalization capability depends upon the features; hence feature engineering becomes an integral part of a good classification system. The transformed features are then sent to a stacked network of MLELM (St-MLELM) for successfully classifying ML data. Features are isolated from the bottleneck of the DAE and fed into the St-MLELM network. The St-MLELM model stacks multiple MLELM networks sequentially to create a deeper network that has improved input-output mapping capability. The main motivation of this work is to create a deeper network for the classification task of handling ML data. Dimensionality reduction is one of the aspects that is addressed through the use of DAE; this network mainly targets feature extraction and better representation of the features. The MLELM model is also being used, which is a fast and one-pass classifier, focused primarily on the handling of multi-label outputs. A stacked network of MLELM is being used, to increase the learning power of the network architecture. The performance of the proposed deep network has been thoroughly experimented with on five benchmark ML datasets against various performance measures. Depth alterations have been tested with respect to the number of hidden layers for the DAE component and the number of MLELMs in the stack to get the optimum configuration for eleven performance measures. The proposed method has also been compared with six state-of-the-art algorithms

to analyse the performance. Also, the effectiveness of DAE vs SAE has been tested.

The primary contributions of this chapter involve:

a. Building a cascade of networks with stacked autoencoders (SAEs) and MLELMs.

b. Stacking MLELMs to learn the mapping of the class scores obtained from the previous classifier to the hard labels.

c. Building a deep neural network model for ML classification integrating deep autoencoders (DAEs) and extreme learning machines.

d. Optimizing the network configuration of the proposed model by analysing its performance for varying depths of the network.

e. Determining the better-suited component among DAE and SAE for dimensionality reduction.

The rest of the chapter is outlined as follows. Section 3.2 discusses some preliminaries that are required to understand the proposed works. Section 3.3 elaborates on the first proposed model and Section 3.4 explains the second proposed model along with their related results discussed in the respective sections. Finally, Section 3.5 concludes the chapter.

3.2 Preliminaries

Among the problem adaptation techniques which exist in the literature, few of the multi-label classifier models have used ELMs and autoencoders. Since the proposed methods are based on MLELM, stacked autoencoder (SAE) and deep autoencoder (DAE), a brief description of the related models are given in the following sub-sections.

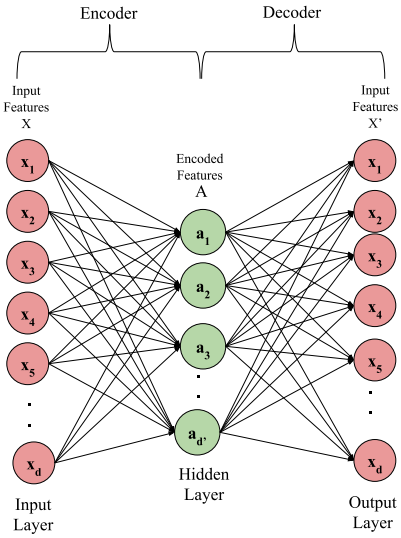


Figure 3.1: Architecture of an autoencoder

3.2.1 Autoencoders

In this era of deep learning, the use of autoencoders (AEs) [44, 45] have increased in various domains. Among the different types of autoencoders in the field of machine learning like denoising AE [82], variational AE, etc., the focus of this work is on stacked autoencoders (SAEs) [9, 98] and deep AEs [29]. A simple AE is a variation of a feedforward neural network that has three layers: input, hidden and output. For N samples, each with feature vector $\mathbf{X_i}$, for $i = 1, .., N$ and $\mathbf{X_i} \in R^d$, both the input and the output layer of the autoencoder has d nodes. The autoencoder works in an unsupervised fashion, unlike a regular feed-forward network.

An example of a simple autoencoder is shown in Figure 3.1. The autoencoder takes $\mathbf{X} = \{x_1, x_2, x_3, .., x_d\}$ as both input and output. The task of an autoencoder is to encode the d-dimensional data to d' dimension in the first part (i.e., encode) and then decode the features from d' to d dimension in the decoder part. When d' is smaller than d, the autoencoder compresses the data to a smaller dimension and then uncompresses it in the next layer. If the input to the autoencoder is $\mathbf{X}$, the encoder maps it to a set of

hidden nodes $\mathbf{A} = \{a_1, a_2, .., a_{d'}\}$ where output at a node a_j is computed as

$$a_j = \phi\left(\sum_{i=1}^{d} w_{ij}x_i + b_j\right), \tag{3.1}$$

where, ϕ is the transfer function for the encoder, w_{ij} is the connection weight between the x_i and a_j, and b_j is the bias. The function of the decoder is to map the encoded representation $\mathbf{A}$ back to an estimate $\mathbf{X}'$ of the original input $\mathbf{X}$. Thus, the value at the k^{th} output node x'_k is computed as follows.

$$x'_k = \psi\left(\sum_{j=1}^{d'} w_{jk}a_j + b'_k\right), \tag{3.2}$$

where, ψ is the transfer function for the decoder, w_{jk} is the connection weight between a_j and x'_k, and b'_k is the bias for the k^{th} node of the decoder. The autoencoder is trained iteratively and the weights are updated through backpropagation like a multi-layer perceptron.

In the scenario of a stacked autoencoder, multiple autoencoders are sequentially placed one after the other. The encoded data from one AE is passed on as input to the next AE and so on, to further extract prominent features from the data. On the other hand, for deep autoencoders, instead of having one hidden layer (as shown later in Figure 3.11), they have multiple hidden nodes. The learning happens through backpropagation for both the variations of AE. However, each AE in an SAE learns sequentially i.e., once the first AE finishes training, its weights do not get updated again in that training phase. Whereas, all the layers in a DAE are updated throughout a training phase. Autoencoders are used for various purposes, such as encoding of features, weight initialization of other networks, etc. In the proposed models, for both types of AEs, their under-complete architecture has been used where the number of nodes gradually reduce starting from the input layer to the final encoder layer. This is suitable since the SAE and the DAE both have been used in the form of a feature extractor before performing classification using

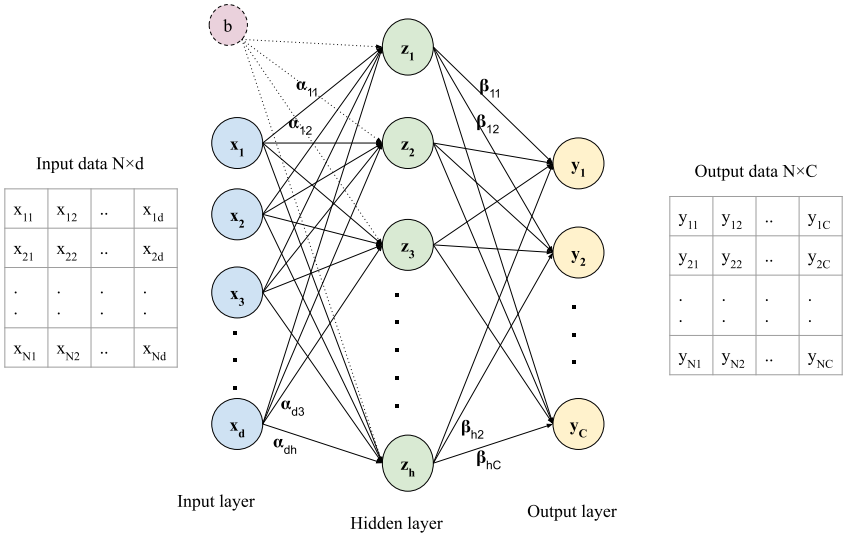


Figure 3.2: Architecture of ELM

ELMs.

3.2.2 Extreme Learning Machine

Extreme learning machine is a compact and efficient single-layer feed-forward neural network that has been quite popular in the past decade. This unique network performs classification task efficiently and fast. It has been found that the learning speed of an ELM can be a lot higher than the traditional feed-forward network while obtaining better generalization performance [32]. The single-label ELM has been adapted for multi-label classification in [73, 74]. The structure of a single-label ELM and a multi-label ELM is the same. The output layer in an ELM handles single outcomes, whereas, in the MLELM, it handles multiple outcomes.

The architecture of the ELM has 3 layers: input, hidden and output. For a single-label sample $\mathbf{X}$, the class is denoted as Y, where $\mathbf{X} \in R^d$ and $\mathbf{Y} \in R^C$. The input layer of an ELM has d nodes, the hidden layer has h nodes and the output layer has C nodes. The network has input to hidden layer connection weights α, biases b and hidden to output layer weights β. An illustration of the ELM network is given in Figure 3.2.

The uniqueness of this network is that the input layer weights and biases are randomly

initialized and unlike most other ANNs, they are not updated any further. Learning from the data occurs in the hidden layer weights alone. An activation function ϕ is used at the hidden nodes. The output at any hidden node z_j is computed as

$$z_j = \phi\left(\sum_{i=1}^{d} \alpha_{ij} x_i + b_j\right), \tag{3.3}$$

where, $j = 1, .., h$, α_{ij} is the connection weight between x_i and hidden node z_j, and b_j is the bias. Similarly, at any output node y_k the outcome is

$$y_k = \sum_{j=1}^{h} z_j \beta_{jk}, \tag{3.4}$$

where, β_{jk} is the weight from hidden node z_j to output node y_k. The above model is compactly represented as

$$\mathbf{Y} = \mathbf{Z}\beta, \tag{3.5}$$

where, $\mathbf{Y} = \{y_1, y_2, .., y_C\}$, $\mathbf{Z} = \{z_1, z_2, .., z_h\}$ and β is the $h \times C$ weight matrix between the hidden layer and the output layer. The outputs from the hidden layer $\mathbf{Z}$ and the output layer $\mathbf{Y}$ are already known. Thus, the hidden to output layer weight matrix β is approximated as:

$$\beta = \mathbf{Z}^{\dagger}\mathbf{Y}, \tag{3.6}$$

where, $\mathbf{Z}^{\dagger}$ is the Moore-Penrose inverse of $\mathbf{Z}$ [57, 70]. Once the weight matrix β is obtained, the ELM model has completed its training phase and is ready for testing. Class predictions for unknown samples are then performed like in any other feed-forward network.

The architecture of a basic ELM model is quite simple and has similarities with older neural network models. This has raised various doubts in the research community which led to difference of opinions [86, 87]. In [30], the author of ELM explains their philosophy and idea behind the proposed model. NN models like RVFL and RBF might be considered

as predecessors of ELM, since they have similar architecture and approach. Despite their similarities, these models are seen to work better than the other in some specific cases that make each of the models unique in their own way. Keeping this in mind, here, the architecture of ELM has been explored mainly for its simplicity and fast computation. The concept of one-pass learning is quite intriguing and has worked moderately for ML data. Hence, our focus is to utilize the strengths of this network to build a better model for multi-label classfication.

Along with the ELM-based techniques specifically tailored for multi-label classification (discussed in Section 2.1.2.1), other ELM-based single-label models can also be employed as multi-label classifiers. Since ELMs and neural networks, in general, do not require much architectural modification to switch from single-label to multi-label classification, they can be adapted to solve the multi-label classification problem. In [66], a fast pruned ELM was proposed that can automatically generate the number of hidden nodes. The relevance of the initial large number of hidden nodes is measured and the irrelevant ones are pruned. [64] developed an online sequential fuzzy ELM (OS-Fuzzy-ELM) which trains the ELM in an online batch-wise mode. The initial training is done with a chunk of data and the later chunks are used to update the parameters. In [65], an aircraft recognition system had been built which extracts three moment-invariant features from the input aircraft images and feeds them as inputs to three separate modules of neural networks. These modules consist of ELMs which in turn perform classification, and the outcome from the modules are combined. [51] proposed a self-adjusting ELM (SA-ELM) which learns the input to hidden layer weights using an ameliorated teaching learning based optimization instead of using the random weights. Apart from these, many other methods also exist in literature which can be adapted for multi-label classification. Among the many applications of ELM, one is being used as an auto-encoder (ELM-AE) [37, 106]. In general, ELM-AE is a sparse autoencoder and the input data is expanded in the hidden layer due to the presence of a large number of hidden nodes in ELM. The proposed method requires feature reduction, hence it has opted for a stacked autoencoder instead of an ELM-

AE. In [37], the authors built an ELM-AE as a part of a deep model called multi-layer ELM specifically for image classification. In the stacked ELM model [106], the authors have divided a single large ELM network into multiple stacked serially connected smaller ELMs. They have utilized ELM-AE in each iteration of the stacked-ELM algorithm to improve performance.

However, unlike the above mentioned ELM-based techniques, the proposed methods aim to improve the performance of ELMs while handling multi-label data specifically. In general, multi-label data tends to have a large number of features, which in turn requires the MLELM to use a larger number of hidden nodes. Not only does this increase the cost, but it also degrades the performance of the network. Hence, it is not able to approximate the weights for data with multiple outputs as efficiently as it can do for single-label data. Also, the decision space of multi-label data is inherently quite complex. A simple MLELM is unable to map the input to the output space efficiently and learn the separating decision boundaries. The complex multi-label data prove to be bulky for the simple one-pass ELM network. Thus, the proposed methods aim to handle these shortcomings with the use of a cascade of networks, where an individual network is employed to handle separate issues.

3.3 Proposed Multi-label Classifier with Stacked Autoencoder and Extreme Learning Machines

In this work, a cascade of stacked autoencoders and extreme learning machines has been proposed for the classification of multi-label data (MLSAEELM). In the first phase, a stacked autoencoder network is used to reduce the dimension of the data. Stacked autoencoders (SAEs) are widely used for deep neural networks since they are capable of efficiently extracting underlying features from any given data. Although the proposed classifier isn't too deep, a reduced set of well-encoded features obtained from the SAE are beneficial to retain the underlying property of the data. Once the relevant features have

been extracted, these are passed on to a multi-label ELM for soft label prediction. This MLELM handles the multi-label data in a way similar to the simple ELM. To improve the classification performance of the MLELM, another MLELM is concatenated in the final phase of the model. The task of this network is to learn the mapping of the soft labels to hard labels. The final class scores predicted by the approximation MLELM are then used to assign hard class labels to the data.

3.3.1 Model Description

A detailed description of the proposed model regarding its architecture, training and testing phases are given in this section.

3.3.1.1 Architecture of the network

The architecture of the proposed model includes a stacked autoencoder followed by a multi-label extreme learning machine for classification. Figure 3.3 shows the overview of the model.

Stacked autoencoder The stacked autoencoder part of the network contains individual autoencoders stacked sequentially. The first autoencoder AE_1 takes the original input $\mathbf{X_i} = \{x_1, x_2, .., x_d\}$, trains itself iteratively and encodes the data to a smaller number of features (say, $\mathbf{a_i} = \{a_1, a_2, .., a_{d'}\}$). These encoded features are then passed on to train the next autoencoder AE_2 and the features are further encoded to $\mathbf{g}$ which has d'' number of features and so on. The final set of encoded features from the autoencoder AE_n is used as input in the next phase. In Figure 3.3, only two layers have been shown in the stacked autoencoder.

Multi-label ELM for soft classification (MLELM-C) In this phase, a simple MLELM network (referred here as MLELM-C) is used, which is an ELM network performing multi-label classification. The encoded features obtained from the previous stacked

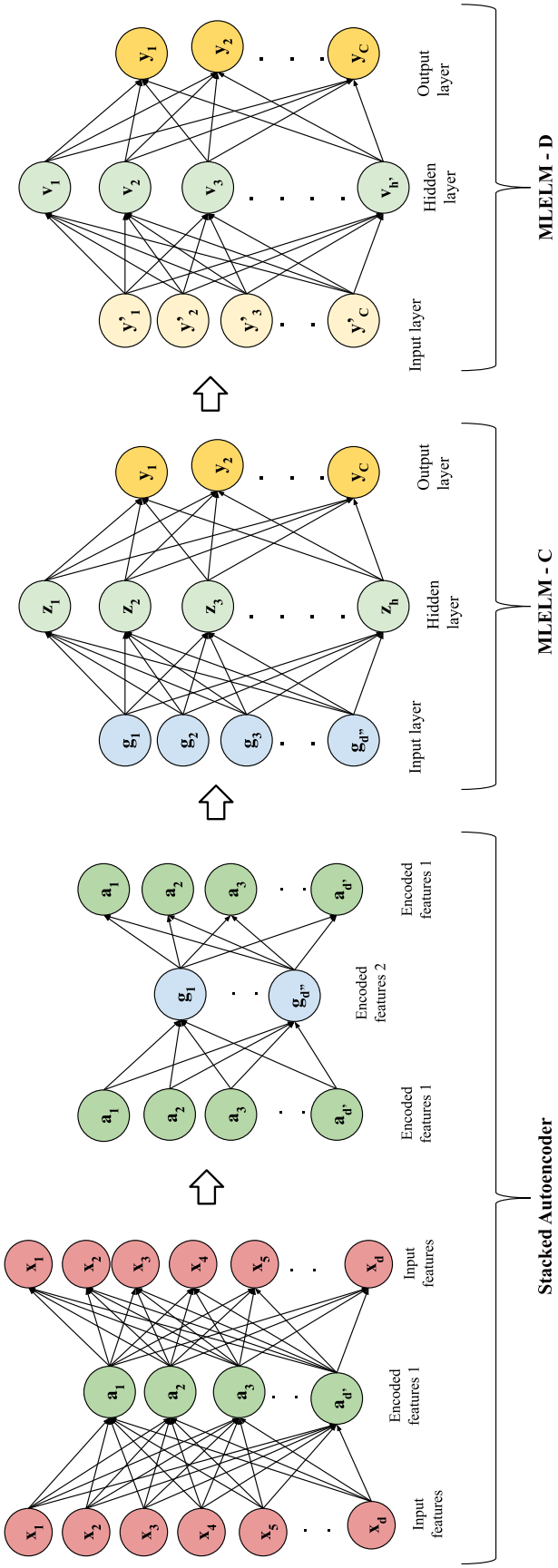


Figure 3.3: Architecture of MLSAEELM

autoencoder phase is taken as input and the class labels for the multi-label data are provided as output. Thus, the MLELM-C maps d'' number of input nodes to C output nodes with h hidden nodes. The hidden layer output vector $\mathbf{Z} = \{z_1, z_2, .., z_h\}$ is computed using Equation 3.3. The MLELM-C model can thus be compactly represented as

$$\mathbf{Y} = \mathbf{Z}\beta, \tag{3.7}$$

where, $\mathbf{Y} = \{y_1, y_2, .., y_C\}$ is the actual output vector and β is the $h \times C$ weight matrix between the hidden layer to the output layer. β is then determined using Equation 3.6.

A large number of hidden nodes are required in comparison to the number of input nodes for the MLELM-C to be able to learn efficiently from the data. Since the number of features of the input data have been reduced, the input layer is not as large as a stand-alone MLELM. Thus, the hidden layer does not need to be extremely large and the weight matrix can be approximated comfortably. Once the weight matrix β has been approximated, the soft label scores at the k^{th} output node can be computed as

$$y'_k = \sum_{j=1}^{h} z_j \beta_{jk}, \tag{3.8}$$

where, β_{jk} is the weight from the hidden node z_j to output node y_k. Each node in the output layer now contains the soft classification score for that particular class. These scores are usually converted to hard labels in a regular MLELM using a threshold. In the proposed method, these soft classification scores are used in the next phase.

Multi-label ELM for score mapping (MLELM-D) The output generated from the MLELM-C is a set of scores, one for each class. For single-label data, the class label corresponding to the highest value is assigned to the sample. While labelling multi-label data, several class labels may be assigned to one sample depending on the obtained score. The general method of determining the hard multi-labels is by setting a threshold. If the predicted value is higher than the threshold, the class is relevant, hence the label is 1,

else the label is 0 for the irrelevant class. This threshold selection is very crucial for multi-label data. A very low threshold can assign more labels than required and a very high threshold might end up under-assigning labels to the data instances, both eventually leading to misclassification. Therefore, it is important that the model can predict correct hard labels from the scores obtained from the classifier. To improve the prediction ability of the model, a modified MLELM is proposed to be used in the third phase that learns the mapping of the scores predicted by MLELM-C to the final class labels. This MLELM-D takes the predicted output values $\mathbf{Y}'$ from the MLELM-C as input and the original target labels $\mathbf{Y}$ as output. The input weights and biases are randomly initialized and the output at the hidden layer V is computed using Equation 3.3. The hidden weights α are similarly learned from $\mathbf{V}$ and the original output labels $\mathbf{Y}$ as

$$\alpha = \mathbf{V}^{\dagger}\mathbf{Y}. \tag{3.9}$$

This MLELM-D helps to map the soft scores predicted by the previous network to the hard labels instead of using a specific threshold like in traditional ELMs. The final predictions made by this MLELM-D network is then converted to hard class labels using a calibrated threshold.

3.3.1.2 Training Phase

In this phase, the training data is passed through three networks sequentially. Each of these networks has a specific task and is trained iteratively.

Feature Extraction At the beginning of the training phase, the input data is fed to the stacked autoencoder to perform feature extraction. The reduced number of features and the number of layers in the autoencoder is predetermined. The SAE trains iteratively till it learns the input thoroughly. Once the SAE has learnt from the training data, the encoded features are obtained from the network. The extracted features of the training data are then passed on to the second network.

Soft Class Prediction The multi-label ELM allocated for soft class prediction (MLELM-C) uses the reduced features from the previous network as input and the original multi-label output in the output layer. The number of nodes in the hidden layer is determined depending on the number of input nodes. This network works in batch mode, where it takes all the input instances together and trains itself in one pass. Once the MLELM-C has learned the hidden layer weights, the outputs need to be predicted. The MLELM-C is again fed with the features encoded training data $\mathbf{g}$, only to generate class scores. The predicted score vector $\mathbf{Y}'$ at the output layer are calculated as

$$\mathbf{Y}' = \beta\phi(\gamma\mathbf{g} + \mathbf{B}), \tag{3.10}$$

where, γ is the input-hidden weight matrix, $\mathbf{B}$ is the bias vector, ϕ is the activation function and β is the hidden-output weight matrix computed by the MLELM-C. The predicted outputs $\mathbf{Y}'$ that is obtained from the MLELM-C are used to train the third network MLELM-D.

Class Score Approximation Another MLELM (named MLELM-D) is used to improve the predictions of MLELM-C by mapping the class scores to actual class labels. The predicted values computed from MLELM-C are provided as input to the MLELM-D and the original class labels are used as output. This MLELM-D network also learns the hidden layer weights in one pass.

3.3.1.3 Testing Phase

Once the complete multi-label stacked encoder and ELM model (MLSAEELM) are trained, it is ready for testing. In this phase, the test data is fed individually to the first SAE network. The SAE generates a set of encoded features for the test data in an unsupervised manner. The reduced input features are then passed on to the second phase. In the soft classification step, the trained MLELM-C computes the individual class scores for each of the test patterns. These intermediate predicted values are then given as input to

the final network MLELM-D which then maps the soft class scores to actual class labels. This third network outputs the final predicted values which are then used to determine the hard class labels for the test samples.

3.3.2 Experimental Analysis

To evaluate the effectiveness of the proposed MLSAEELM technique, it has been tested on seven multi-label datasets using ten performance metrics. These outcomes have been compared with eleven other relevant techniques to judge the overall performance of the proposed model.

3.3.2.1 Setup

Seven well-known multi-label datasets have been used to evaluate the performance of the proposed model. These datasets are Emotions (music), Scene (image), Flags (image), Slashdot (text), Yeast (biology), Delicious (web text) and EUR-Lex (text).

Ten performance measuring indices [28] have been computed with k-fold cross-validation on the above datasets. These metrics are namely, Hamming loss (HL), ranking loss (RL), one error (OE), subset accuracy (SA), average precision (AP), macro-F1 ($MacF1$), micro-F1 ($MicF1$), accuracy (Acc) and average recall (AR).

The proposed model has been compared with eleven related models. The first method of comparison is a multi-label adaptation of the original ELM (MLELM) with a global threshold of 0.5. A bipolar MLELM [81] (B-MLELM) and ELM-ML [73] have been included for comparison. Among non-ELM methods, binary relevance (BR) [24], classifier chains (CC) [62] and random k label-sets (RAKEL) [79] along with one of their modifications (BRq, CCq and RAKEL-d) have been used for comparison. Apart from these existing techniques, partial structures of the proposed method have also been included for evaluation. The method referred to as S-MLELM includes only the first and second phase of the proposed model, i.e., an SAE followed by MLELM-C. Comparison with

this model shows the utility of the MLELM-D network of the proposed work. Another model MLELM+ which contains the second and the third phase of the proposed model (MLELM-C + MLELM-D) has been used for comparison to show the importance of using an SAE. A detailed analysis of the results is given in the following section.

3.3.2.2 Analysis of Results

The proposed classification model (represented as MLSAEELM) has been compared with eleven other relevant multi-label classification methods (including some ELM based techniques) from literature. The 5-fold cross-validation results corresponding to these algorithms for seven datasets and ten performance measures are shown below. Tables 3.1–3.7 show the results for Emotions, Scene, Flags, Slashdot, Yeast, Delicious and EUR-lex datasets respectively for nine measures.

Table 3.1: Comparative results for Emotions dataset

	HL	**RL**	**OE**	**SA**	**AP**	**AR**	**MacF1**	**MicF1**	**Acc**
MLELM	0.4138	0.4039	0.5678	0.3960	0.6037	0.4774	0.6507	0.6505	0.3024
B-MLELM	0.4034	0.3943	0.5882	0.4272	0.5996	0.5322	0.6653	0.6615	0.3335
ELM-ML	0.4774	0.4683	0.6186	0.3757	0.5475	0.4972	0.6259	0.6250	0.2822
MLELM+	0.3655	0.3601	0.5042	0.4541	0.6385	0.4552	0.6633	0.6594	0.3545
S-MLELM	0.2792	0.2608	0.3833	0.5267	0.7225	**0.5625**	0.6986	0.6976	0.4360
BR	0.2640	0.3270	0.4470	0.1720	0.6190	0.1654	0.2785	0.2825	0.4220
BRq	0.2830	0.3040	0.4820	0.1550	0.5400	0.1435	0.2885	0.2915	0.4460
CC	0.2610	0.2940	0.4200	0.2210	0.5790	0.2091	0.2865	0.2925	0.4670
CCq	0.2780	0.2970	0.4470	0.2140	0.5680	0.1879	0.2835	0.2870	0.4590
RAKEL	0.3450	0.2230	0.3170	0.0930	0.7100	0.0879	0.3051	0.3060	0.4710
RAKEL-d	0.2610	0.3060	0.4590	0.1650	0.5560	0.1376	0.2830	0.2855	0.4330
MLSAEELM	**0.1852**	**0.1580**	**0.2735**	**0.5695**	**0.8065**	0.5598	**0.7245**	**0.7232**	**0.4957**

Analysing the performance measures for all the methods in the above tables, it is seen that they vary in a similar manner for all the datasets across all the metrics. The algorithms MLELM and B-MLELM, both solo networks, are seen to perform quite close to each other for all the datasets, whereas their performance is quite low compared to other algorithms. Thus, it can be seen that using a stand-alone MLELM and its variations is unable to predict multi-label classes well. The single network is not sufficient for learning the intricacies that are present in the data.

Table 3.2: Comparative results for Scene dataset

	HL	RL	OE	SA	AP	AR	MacF1	MicF1	Acc
MLELM	0.1819	0.4699	0.7713	0.0014	0.4430	0.0211	0.8181	0.8181	0.0010
B-MLELM	0.1778	0.4676	0.7672	0.0010	0.4471	0.0251	0.8225	0.8226	0.0007
ELM-ML	0.2763	0.4722	0.7905	0.1916	0.4339	0.1826	0.7721	0.801	0.1815
MLELM+	0.1753	0.3895	0.7469	0.0145	0.4966	0.0166	0.8259	0.8259	0.0135
S-MLELM	0.1459	0.1340	0.3410	0.5773	0.7892	0.6112	0.8259	0.8267	0.5442
BR	0.1350	0.2320	0.3890	0.4230	0.6870	0.5426	0.3175	0.312	0.5340
BRq	0.1440	0.2070	0.3930	0.4020	0.6010	0.3981	0.3115	0.3185	0.5410
CC	0.1440	0.2200	0.3810	0.5310	0.6540	0.5571	0.3081	0.3010	0.5840
CCq	0.1430	0.2110	0.3650	0.5290	0.6590	0.4872	0.3115	0.3060	0.5930
RAKEL	0.2210	0.1510	0.3160	0.2290	0.7450	0.2765	0.3025	0.2930	0.5140
RAKEL-d	0.1440	0.2260	0.4080	0.4470	0.6050	0.4367	0.3025	0.2981	0.5320
MLSAEELM	**0.0918**	**0.0760**	**0.2225**	**0.6126**	**0.8677**	**0.6351**	**0.8352**	**0.8355**	**0.6019**

Table 3.3: Comparative results for Flags dataset

	HL	RL	OE	SA	AP	AR	MacF1	MicF1	Acc
MLELM	0.4029	0.4162	0.4615	0.5242	0.6598	0.5406	0.5852	0.5602	0.4045
B-MLELM	0.3985	0.3754	0.3158	0.5889	0.7227	0.5921	0.5657	0.5481	0.4504
ELM-ML	0.5201	0.5256	0.6410	0.4476	0.5877	0.4547	0.5230	0.4907	0.3226
MLELM+	0.3666	0.3537	0.3493	0.5846	0.7250	0.5444	0.5866	0.5669	0.4734
S-MLELM	0.3679	0.3013	0.2051	0.5807	0.7671	0.5682	0.5751	0.5637	0.4474
BR	**0.2530**	0.2690	0.3090	0.1750	0.7720	0.1862	0.3345	0.3735	0.6060
BRq	0.2540	0.2720	0.2110	0.1650	0.6710	0.1655	0.3541	0.3830	**0.6270**
CC	0.2720	0.2940	0.2320	0.2580	0.6800	0.2351	0.3285	0.3610	0.5860
CCq	0.2650	0.2960	0.2110	0.1600	0.6680	0.1462	0.3401	0.3750	0.6120
RAKEL	0.2830	0.2670	0.2680	0.1490	0.7490	0.1311	0.3482	0.3745	0.6070
RAKEL-d	0.2750	0.3140	0.2320	0.2110	0.6780	0.1892	0.3241	0.3605	0.5860
MLSAEELM	0.2612	**0.1895**	**0.1429**	**0.6758**	**0.8420**	**0.6248**	**0.6221**	**0.6014**	0.5503

Table 3.4: Comparative results for Slashdot dataset

	HL	RL	OE	SA	AP	AR	MacF1	MicF1	Acc
MLELM	0.1803	0.2995	0.7530	0.2222	0.3855	0.4714	0.8441	0.9386	0.1571
B-MLELM	0.1750	0.3233	0.7701	0.2235	0.3713	0.459	0.8681	0.9392	0.1605
ELM-ML	0.1347	0.2308	0.6640	0.2837	0.4770	**0.5007**	0.8721	0.9417	0.2182
MLELM+	0.2092	0.3269	0.7734	0.1994	0.3658	0.4357	0.8658	0.9365	0.1410
S-MLELM	0.0738	0.2165	0.5854	0.3143	0.5302	0.3603	0.8996	0.9453	0.2876
BR	0.0430	0.1440	0.5080	0.2990	0.5950	0.2891	0.1175	0.2301	0.3460
BRq	0.0450	0.3000	0.6170	0.2880	0.3820	0.2677	0.1245	0.2295	0.3480
CC	0.0560	0.3010	0.5390	0.3630	0.4540	0.3491	0.1245	0.2215	**0.4230**
CCq	0.0590	0.2970	0.5660	0.3450	0.4290	0.3167	0.1265	0.2085	0.3970
RAKEL	0.0490	0.3980	0.8070	0.1290	0.2000	0.1472	0.0745	0.1271	0.1540
RAKEL-d	0.0420	0.3030	0.6000	0.3110	0.3920	0.3418	0.1251	0.2365	0.3570
MLSAEELM	**0.0412**	**0.1078**	**0.4276**	**0.3748**	**0.6768**	0.3767	**0.9024**	**0.9470**	0.3593

Table 3.5: Comparative results for Yeast dataset

	HL	RL	OE	SA	AP	AR	MacF1	MicF1	Acc
MLELM	0.2177	0.1938	0.2541	0.5855	0.7419	0.5568	0.7332	0.7159	0.4751
B-MLELM	0.2178	0.1885	0.2464	0.5709	0.7412	0.5462	0.7372	0.7214	0.4595
ELM-ML	0.2228	0.1782	0.2583	**0.6442**	0.7424	**0.7203**	0.7395	0.7214	**0.5292**
MLELM+	0.2091	0.1875	0.2611	0.5942	0.7429	0.5588	0.7409	0.7233	0.4891
S-MLELM	0.2129	0.1838	0.2333	0.5900	0.7480	0.576	0.7402	0.7230	0.4789
BR	0.2510	0.3150	0.4380	0.0620	0.5820	0.0632	0.1951	0.2841	0.4200
BRq	0.2610	0.2660	0.4940	0.0620	0.5310	0.0671	0.2095	0.3060	0.4690
CC	0.2660	0.3000	0.4820	0.1440	0.5330	0.1231	0.1985	0.2765	0.4250
CCq	0.2740	0.2830	0.5330	0.1250	0.5320	0.1253	0.2051	0.2945	0.4600
RAKEL	0.3200	0.2860	0.4140	0.0480	0.5510	0.0433	0.2120	0.2845	0.4250
RAKEL-d	0.2740	0.3120	0.4820	0.0640	0.4950	0.0598	0.1940	0.2725	0.3980
MLSAEELM	**0.1943**	**0.1648**	**0.2222**	0.6104	**0.7649**	0.5707	**0.7444**	**0.7267**	0.5021

Table 3.6: Comparative results for Delicious dataset

	HL	RL	OE	SA	AP	AR	MacF1	MicF1	Acc
MLELM	0.0207	0.2073	0.4915	0.1552	0.2663	0.1246	0.9805	0.9805	0.0998
B-MLELM	0.0204	0.2074	0.4845	0.1532	0.2628	0.1253	0.9807	0.9808	0.0985
ELM-ML	0.0234	0.1588	0.4519	**0.2629**	0.2812	**0.2388**	0.9799	0.9806	0.1840
MLELM+	0.1117	0.2765	0.5652	0.1276	0.2210	0.2204	0.9786	0.9787	0.1451
S-MLELM	0.0183	**0.1392**	0.3607	0.1747	0.3476	0.1211	0.9809	0.9807	0.1672
BR	0.0182	0.8824	0.3414	0.0118	0.2681	0.1218	0.0235	0.0967	0.1089
BRq	0.0215	0.8654	0.3541	0.0109	0.2542	0.1209	0.0231	0.0929	0.1310
CC	0.0187	0.1780	0.3059	0.0255	0.1641	0.0925	0.0617	0.1285	0.1467
CCq	0.0209	0.1654	0.3373	0.0256	0.1746	0.0875	0.0573	0.1256	0.1358
RAKEL	0.0182	0.2002	0.3426	0.0261	0.1881	0.0926	0.0507	0.1241	0.1429
RAKEL-d	0.0199	0.1823	0.3456	0.0271	0.1785	0.1013	0.0581	0.1131	0.1265
MLSAELM	**0.0147**	0.1499	**0.3075**	0.1855	**0.3580**	0.2242	**0.9846**	**0.9809**	**0.1881**

Table 3.7: Comparative results for EUR-Lex dataset

	HL	RL	OE	SA	AP	AR	MacF1	MicF1	Acc
MLELM	0.0232	0.0886	0.4007	0.4267	0.6072	0.5995	0.9216	0.9889	0.4135
B-MLELM	0.0186	0.0450	0.2027	**0.5340**	0.7763	0.5868	0.9344	0.9890	0.4703
ELM-ML	0.0109	0.0397	0.2412	0.2108	0.7530	0.0000	0.9294	0.9891	0.4582
MLELM+	0.0215	0.1112	0.4390	0.4009	0.5617	0.5086	**0.9365**	0.9889	0.3439
S-MLELM	0.0110	0.1319	0.1778	0.3862	0.2211	0.6154	0.9343	0.989	0.5162
BR	0.0153	0.9478	0.2200	0.2701	0.7631	0.3562	0.0977	0.2209	0.4515
BRq	0.0202	0.8751	0.2118	0.2618	0.7281	0.3721	0.0941	0.2178	0.4987
CC	0.0050	0.7735	0.1937	0.4796	0.7254	0.4567	0.2454	0.3819	**0.5986**
CCq	0.0166	0.7543	0.2032	0.4511	0.7203	0.4435	0.2681	0.3491	0.5672
RAKEL	0.0046	0.8107	0.1854	0.4982	0.7747	0.4873	0.2557	0.3905	0.5118
RAKEL-d	0.0199	0.7968	0.1955	0.5012	0.7683	0.4967	0.2526	0.3777	0.5098
MLSAELM	**0.0031**	**0.0295**	**0.1677**	0.4865	**0.7773**	**0.6252**	0.9393	**0.9892**	0.5208

ELM-ML and MLELM+ both use two consecutive networks instead of one MLELM, and their performance is better than that of the single MLELMs in most of the cases. ELM-ML uses the second network for threshold approximation, whereas MLELM+ uses the second MLELM to map class score (MLELM-D). Among these two, MLELM+ seems to perform better than ELM-ML, which indicates the benefit of using the proposed MLELM-D network. Among MLELM, MLELM+, S-MLELM and MLSAEELM it is seen that the proposed model has significant improvement in performance by using SAE for feature encoding and MLELM-D for class score approximation. The proposed method is also seen to perform better than the other existing methods BR, BRq, CC, CCq, RAKEL and RAKEL-d. These results have been further confirmed by testing MLSAEELM against these methods by varying the amount of training data as shown in Figures 3.4–3.10. These box plots show the average precision of the methods for all the datasets over k-fold cross-validation, ranging k from 2 to 10.

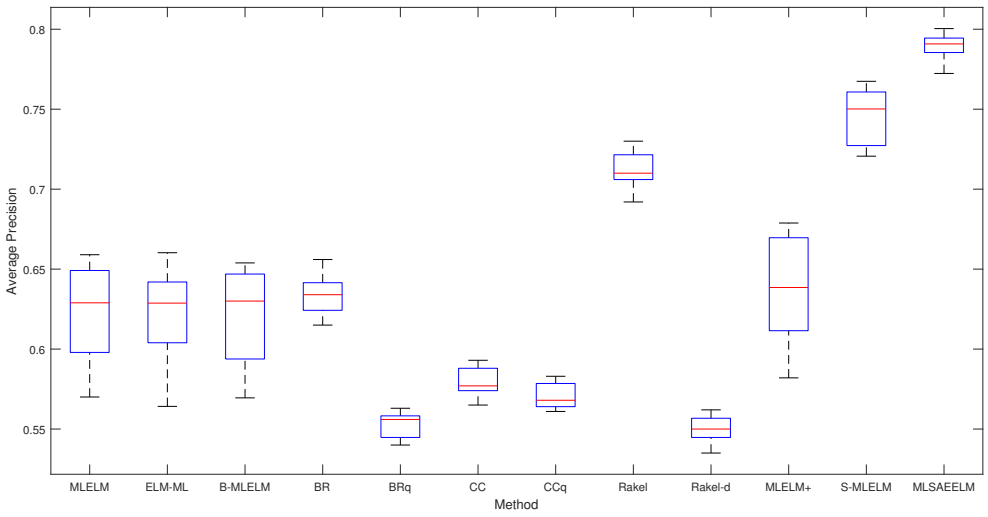


Figure 3.4: Average Precision of all algorithms over k-fold cross-validation for Emotions dataset

Among all the methods used for comparison, the proposed approach is seen to perform significantly better over the existing ones for all the seven datasets and all the performance metrics. T-test statistics on the proposed approach against all the compared methods

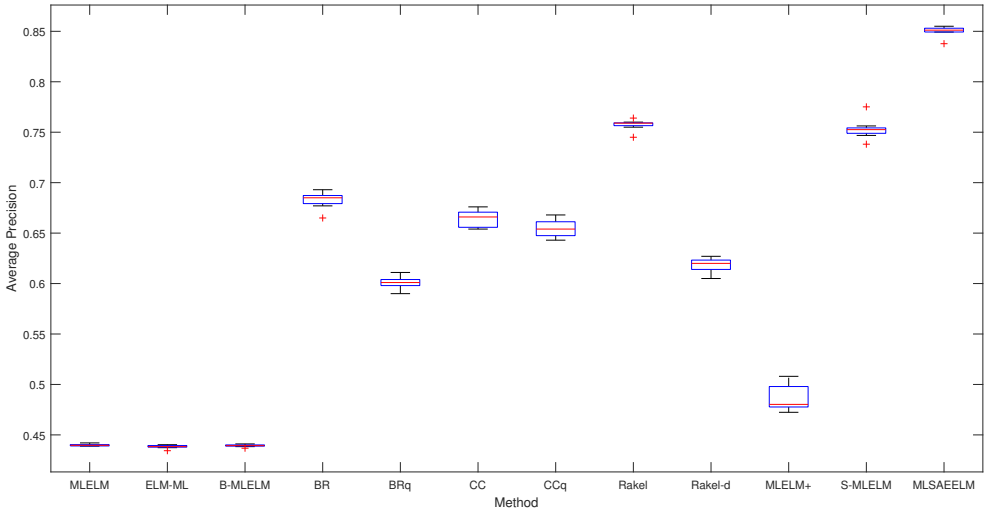


Figure 3.5: Average Precision of all algorithms over k-fold cross-validation for Scene dataset

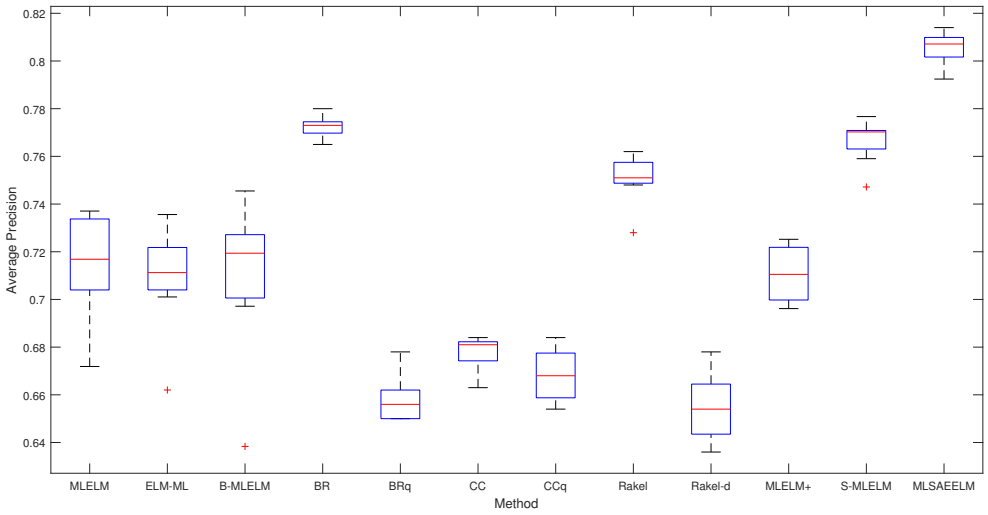


Figure 3.6: Average Precision of all algorithms over k-fold cross-validation for Flags dataset

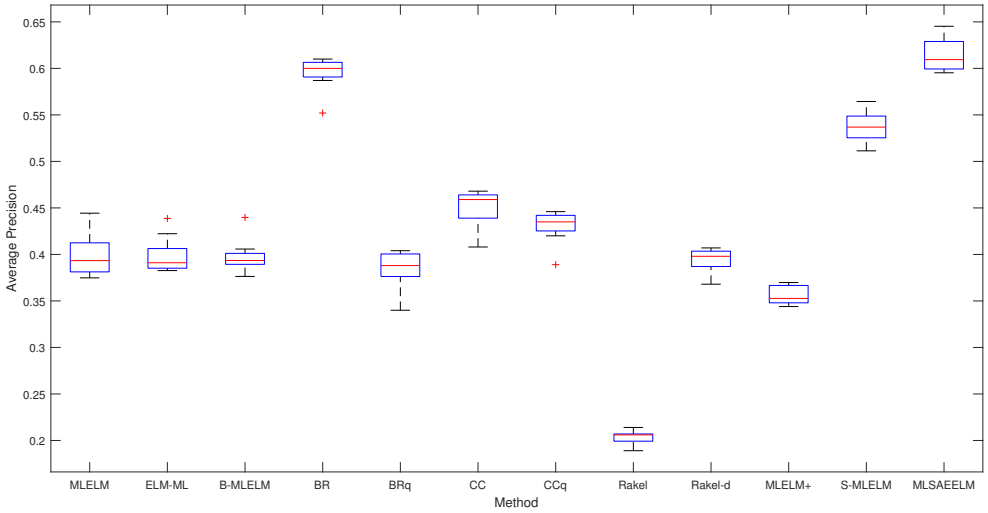


Figure 3.7: Average Precision of all algorithms over k-fold cross-validation for Slashdot dataset

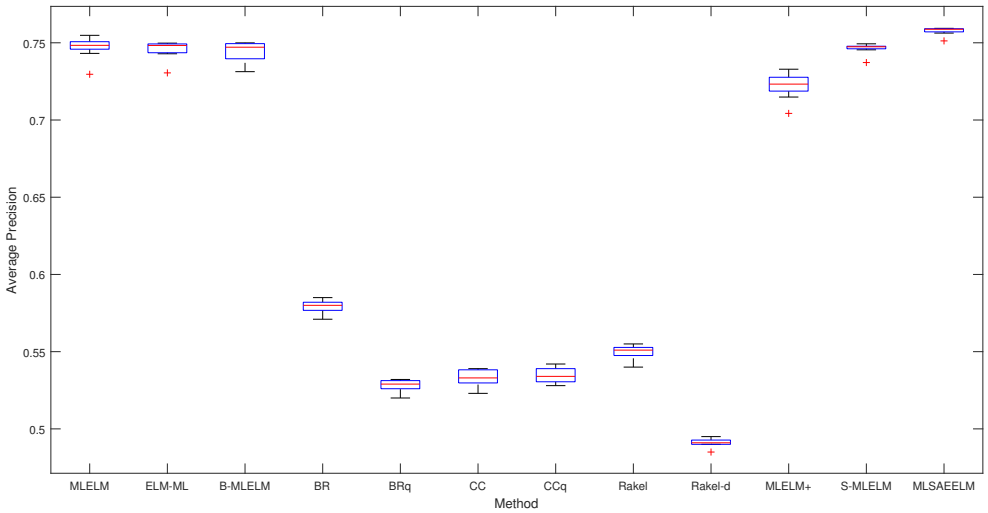


Figure 3.8: Average Precision of all algorithms over k-fold cross-validation for Yeast dataset

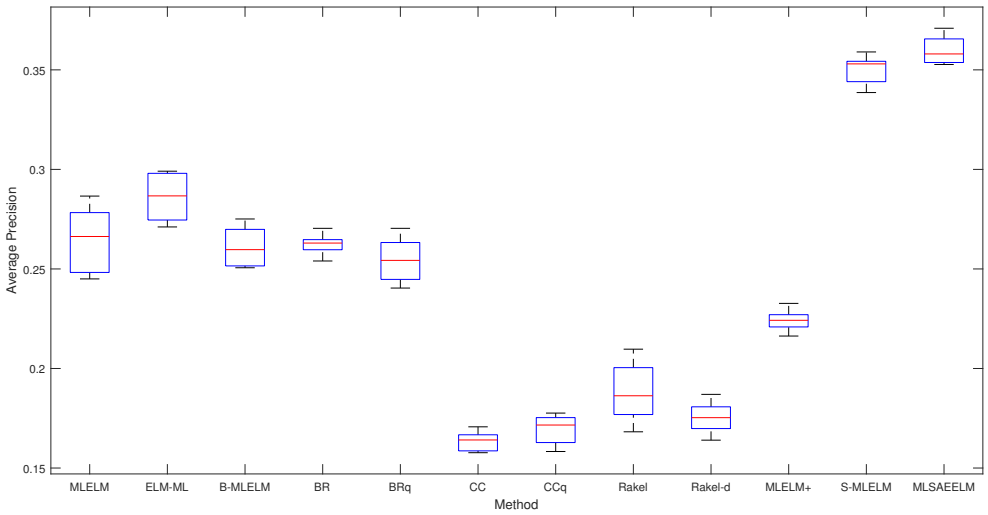


Figure 3.9: Average Precision of all algorithms over k-fold cross-validation for Delicious dataset

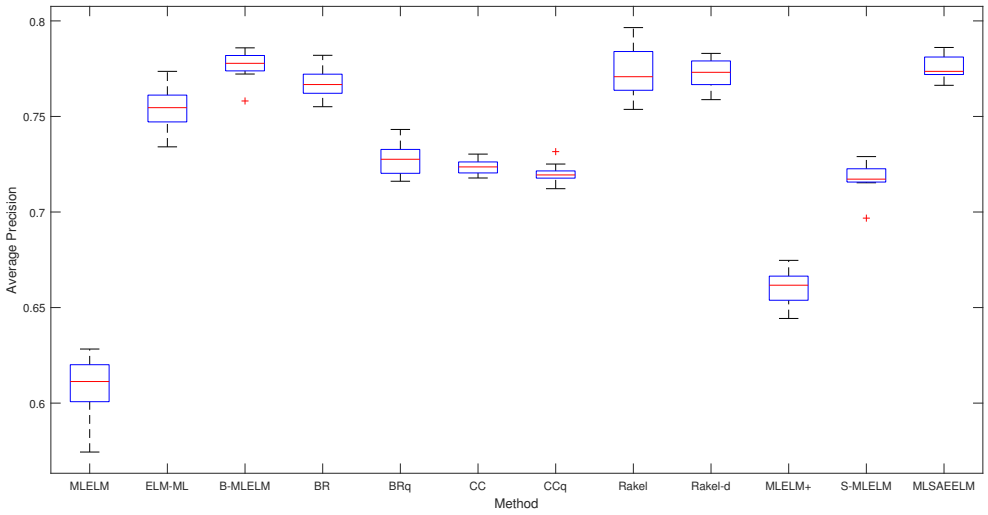


Figure 3.10: Average Precision of all algorithms over k-fold cross-validation for EUR-Lex dataset

for the average precision metric is shown in Table 3.8. For $t_{0.90} = 1.440$ with degrees of freedom = 6, the proposed method MLSAEELM outperforms all the other models.

Table 3.8: T-test statistics for all the algorithms against the proposed MLSAEELM

Method	**T-test value**
MLELM	4.1996
B-MLELM	3.0272
ELM-ML	3.5019
MLELM+	4.7244
S-MLELM	1.6514
BR	4.6123
BRq	5.7343
CC	4.3426
CCq	4.7881
RAKEL	2.4001
RAKEL-d	4.5857

To evaluate further, a non-parametric two-tailed Wilcoxon signed-rank test has been performed on the accuracy metric for all the datasets against all other methods (Table 3.9). It is seen that for alpha=0.20, $T_{Wilcoxon}(7) = 5$, the proposed MLSAEELM outperforms the other algorithms for the accuracy metric.

The computational complexity of the proposed algorithm has been computed experimentally and compared with the other existing methods and shown in Table 3.10. Both the training times and testing times have been recorded separately. It is seen that the training time for the proposed algorithm is higher compared to the other algorithms, especially for the larger datasets. This is mainly due to the stacked autoencoder network which is iteratively trained in the proposed algorithm, thus adding on some extra amount of training time. Due to this reason, S-MLELM also has a large training time. Since the training phase is performed offline, a larger training time does not quite affect the actual testing speed of the algorithm. From the recorded testing times, it can be seen that the proposed method is quite fast and its speed is comparable to the rest of the methods. Moreover, its testing time for some datasets especially the larger ones is faster than most of the other compared algorithms.

Table 3.9: Two-tailed Wilcoxon signed-rank test statistics for all the methods vs proposed MLSAEELM (based on accuracy)

Method	***p* Value**	**Value of sign rank**
MLELM	0.015625	28
B-MLELM	0.015625	28
ELM-ML	0.046875	26
MLELM+	0.015625	28
S-MLELM	0.015625	28
BR	0.046875	26
BRq	0.296875	21
CC	0.937501	13
CCq	0.937500	13
RAKEL	0.109375	24
RAKEL-d	0.078125	25

Table 3.10: Run-time (in seconds) of all the algorithms for all datasets

		Emotions	**Scene**	**Flags**	**Slashdot**	**Yeast**	**EUR-Lex**	**Delicious**
MLELM	*Train*	0.1749	1.1105	0.0124	2.1116	1.5371	161.9306	215.9715
	Test	0.0011	0.0088	0.0018	0.0629	0.0221	3.8547	0.1599
B-MLELM	*Train*	0.1778	1.1031	0.0194	2.1183	1.4245	181.0913	234.7683
	Test	0.0013	0.0079	0.0011	0.0672	0.0034	3.7205	0.1713
ELM-ML	*Train*	0.1328	4.2526	0.0183	66.7521	1.6929	936.4543	755.9038
	Test	0.0035	0.0416	0.0028	0.6456	0.0179	0.4021	0.3654
MLELM+	*Train*	0.0632	2.0992	0.0163	44.4069	0.7496	167.9387	405.1314
	Test	0.0037	0.0242	0.0027	0.2026	0.0131	0.2066	0.1894
S-MLELM	*Train*	2.3339	24.9166	1.2771	338.0252	7.4861	12856.6601	528.0802
	Test	0.0852	0.0637	0.0596	0.0763	0.0609	0.5123	0.0833
BR	*Train*	0.3346	0.6719	0.3063	0.6518	0.3785	47.5633	40.1136
	Test	0.0151	0.0417	0.0405	0.1096	0.1362	3.7268	30.9803
BRq	*Train*	0.3202	0.6551	0.2917	0.6265	0.3905	49.1082	40.1021
	Test	0.0145	0.0322	0.0336	0.1077	0.1159	3.0465	29.5411
CC	*Train*	0.3622	0.9818	0.3211	1.4761	0.7788	973.2005	40.2038
	Test	0.0180	0.0348	0.0154	0.1085	0.0752	4.1138	33.1692
CCq	*Train*	0.3396	0.9481	0.3202	1.4655	0.7791	864.5708	39.2140
	Test	0.0154	0.0301	0.0132	0.0998	0.0663	4.0213	32.9651
RAKEL	*Train*	0.6254	4.4748	0.3752	1.3193	4.0614	6467.4840	347.1959
	Test	0.0211	0.1389	0.0185	0.1424	0.0872	3.9647	31.6542
RAKEL-d	*Train*	0.5889	4.4734	0.3776	1.3330	4.0544	6542.3890	347.9101
	Test	0.0204	0.1134	0.0166	0.1326	0.0803	3.7465	30.2645
MLSAEELM	*Train*	2.5726	25.7914	1.2392	334.4661	8.2951	18353.8901	677.5598
	Test	0.0801	0.0592	0.0575	0.0876	0.0741	0.8718	0.1609

Thus, from all the results obtained, it can be said that the application of a stacked encoder for feature extraction and additionally learning the soft class scores to hard labels mapping in the proposed method significantly improves the performance of the MLELM compared to other algorithms.

3.4 Proposed Deep Multi-label Classifier with Deep Autoencoder and Extreme Learning Machine

In continuation of the above-proposed work, extension of the previous model has been done to build a deep network for efficient ML classification. The main aim of this network is to handle the large multi-label feature space by reducing it to a well-represented lower dimension and then perform fast classification while learning to map the input to the output suitably. It is mainly a two-phase model. The first part of the network comprises a DAE for the initial feature extraction. After that, these relevant features are fed through a network of stacked MLELMs (St-MLELM) for multi-label classification. Subsequent addition of multiple networks of ELM has been done to improve the learning of the ML data. The soft classification scores from the final ELM output layer is passed through a threshold limit in order to obtain a hard classification of the predicted class labels.

3.4.1 Model Description

In this section, the components and the architecture of the proposed network are discussed in detail.

3.4.1.1 Feature Reduction by DAE

The first component of this network model is feature extraction by DAE. It is a feed-forward multi-layer neural network, where the output is mapped as the input itself for reconstruction. The structure for the DAE network has been shown in Figure 3.11. x represents the input features and x' represents the reconstructed input in the network.

The intermediate layer a generates the encoded version of the actual input x, which can

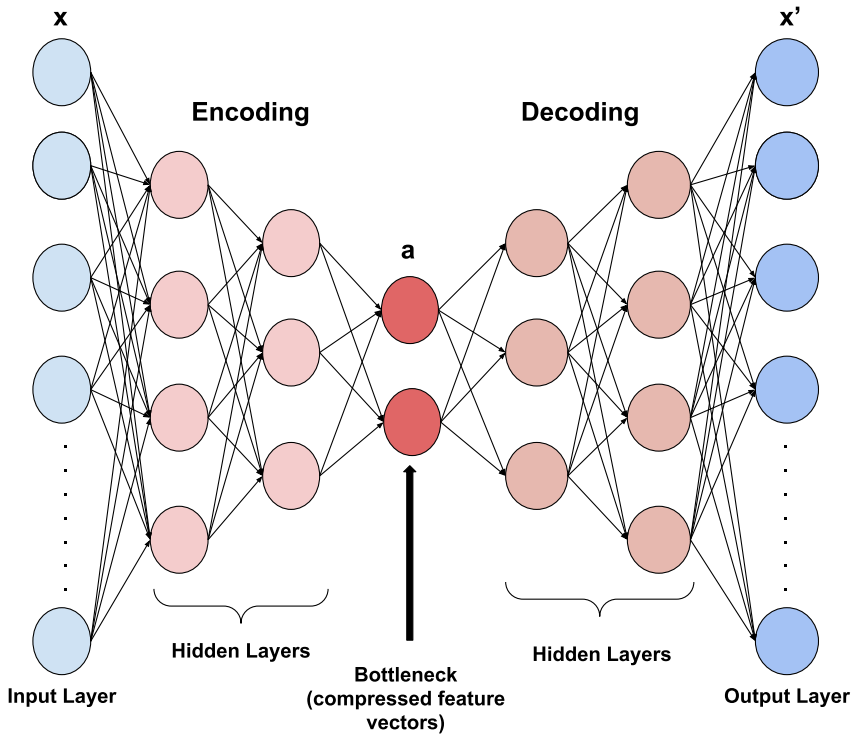


Figure 3.11: Structure of Deep Autoencoder

be considered as an unsupervised transformation of the features. In the training phase, the DAE adjusts all its weights using the backpropagation mechanism. In this step of the proposed algorithm, a DAE is preferred over SAE, since the latter greedily trains and stacks AEs one after the other. Whereas, a DAE tries to optimize all the layers simultaneously, thus improving the overall learning. While creating a deep network, it is important to consider components that would enhance the performance of the overall model. The performance of SAE vs DAE has been experimentally compared in Section 3.4.2.2. After training of the DAE is completed, the encoded input a generated from this component is used in the next phase.

3.4.1.2 Sequential classification by St-MLELM

The second part of the network is the ML classification with the extracted features from the encoded inputs of the DAE. This phase uses a network of stacked multi-label ELM networks (St-MLELM) which are capable of efficient input to output mapping. The

individual components in this stack are MLELMs which are single-layer feed-forward network (SLFN). The architecture and description of a single ELM (same as an MLELM) network are given in Section 3.2.2. The St-MLELM component is built by stacking multiple individual MLELM networks. Stacking the consecutive networks of MLELM helps to observe the performance of the network and how it behaves with increasing complexity being introduced. While looking closely into the architecture of the network, it is seen that as the first layer of a simple ELM network initializes weights randomly, there is no real learning involved in that part. The actual learning occurs in the second section of the ELM network. Hence, if the classifier architecture is restricted to just one MLELM classifier, the learning is restricted as well. When multiple ELM networks are stacked for the same purpose, the learning considerably increases, which shows improved performance. After the classification is performed by the St-MLELM network, a soft-class score approximation is generated at this stage. This soft class score is again passed through a thresholding function to assign a hard classification label for each of these instances. This threshold has been considered as 0.5 for the experiments.

3.4.1.3 Architecture

The training and testing architectures of the proposed model are depicted below. They are essentially the same model, but the working is different in both phases, hence it is depicted separately for ease of understanding.

Training phase The training phase of the network consists of both feature extraction by DAE and multi-label classification by stacked MLELMs. Figure 3.12 shows the entire architecture of the training network. The training phase for the deep autoencoder involves the encoder and decoder sections since the training is done on the entire network. Here a six-layer DAE has been shown, however, the depth of the DAE can be varied as required. After training of the DAE, the encoded input a is fed as input to the St-MLELM network. Here a stack of 3 MLELMs has shown, which can also be increased as per requirement.

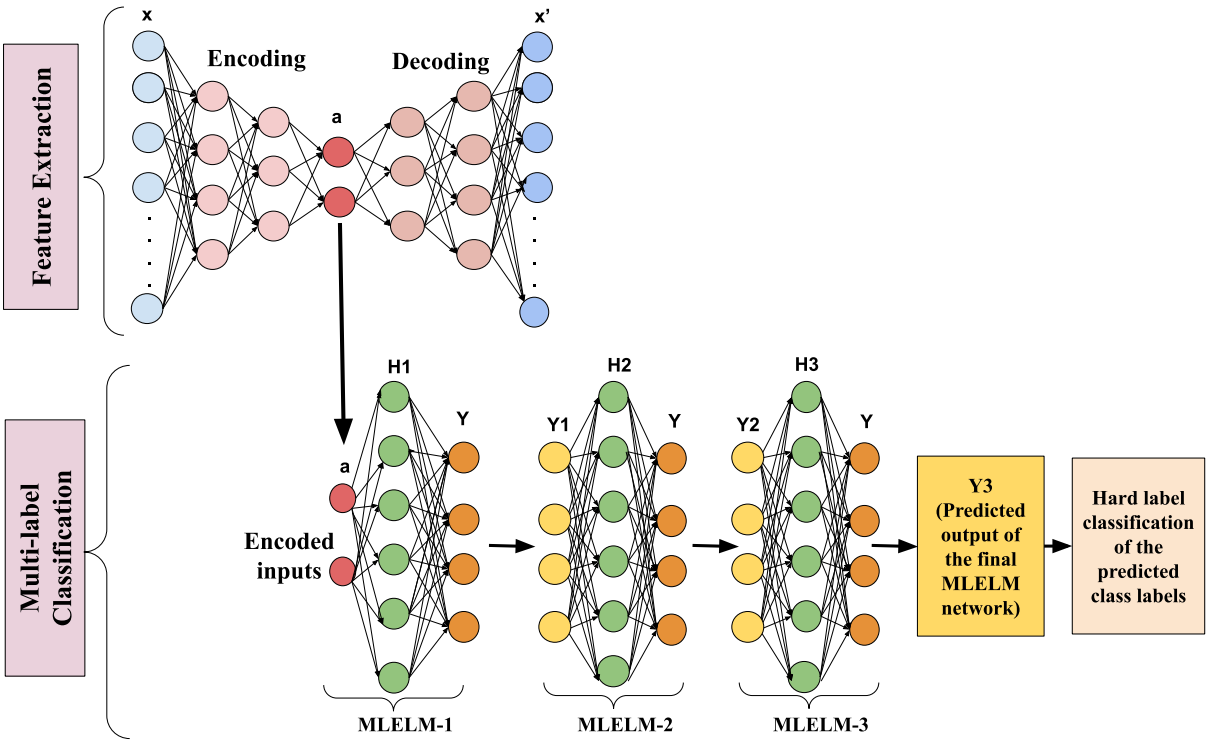


Figure 3.12: Representation of the training architecture of the network

Each of the MLELM networks maps the input to the target output, denoted by Y. The predicted outputs for each of these MLELM networks (MLELM-1, MLELM-2, MLELM-3) are given by $Y1$, $Y2$ and $Y3$ respectively. After training of MLELM-1, the predicted output, $Y1$, is fed to MLELM-2. Again, for the subsequent network, $Y2$, is fed to MLELM-3, and so on. This is done to improve the input to output mapping capability of the network, while not increasing the learning time drastically. Here, the three MLELM stack predicts the final soft-label classification score as $Y3$. This score is further passed through a global threshold, which then determines the relevant and irrelevant hard label-sets.

Testing phase The testing architecture can be thought of as a fully connected network as depicted in Figure 3.13. It utilizes all the layers trained in the previous phase. First, it only takes the encoder portion of the trained DAE which generates the encoded inputs. Thus original input is pushed forward through the encoder layers to the St-MLELM layers directly. The trained St-MLELM layers transform the encoded input further through the

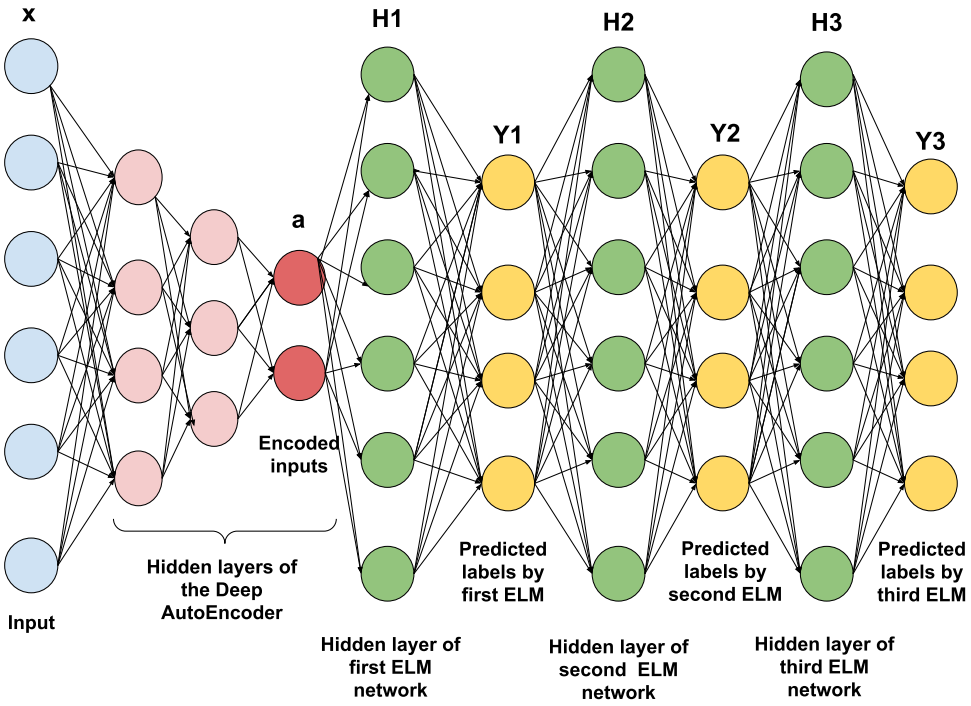


Figure 3.13: Representation of the testing architecture of the network

multiple layers which map the input to the output successfully. The stack of MLELMs can now be thought of as a fully connected network since the output of one is used as an input to the next. The testing network uses all the weights learned in the training phase. The output layer predicts classification scores for the test data which is then converted to hard labels by a global threshold.

3.4.2 Experimental Analysis

The proposed deep ML classifier based on deep autoencoders and stacked MLELM has been tested on five benchmark ML datasets, namely, Scene, Flags, Emotions, Delicious and Yeast. Eight performance measures have been used for the experiments, i.e., Hamming loss (HL), one error (OE), ranking loss (RL), average precision (AP), accuracy (A), subset accuracy (SA), macro-F1 (Mac-F1) and micro-F1 (Mic-F1) metrics. Experiments have been conducted on the proposed model such that different concerns can be addressed. It is mainly divided into the following categories.

3.4.2.1 Varying the depth of the entire network

To judge the performance of the entire network six different configurations have been tried. The proposed model has two components, namely, DAE and St-MLELM. For testing purpose, one component is kept constant while the depth of the other component is varied.

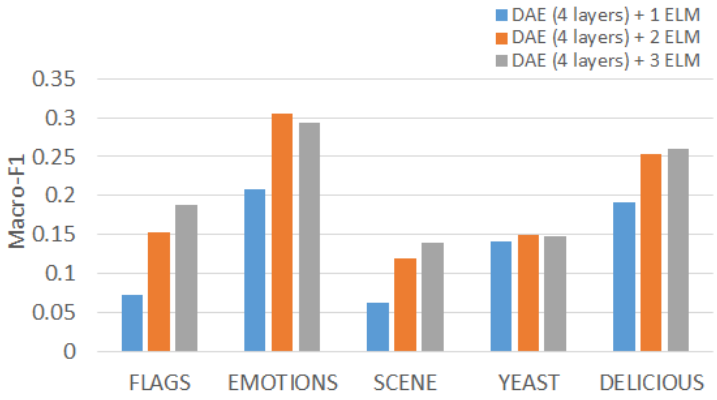


Figure 3.14: Macro-F1 across all datasets with increasing number of MLELM networks

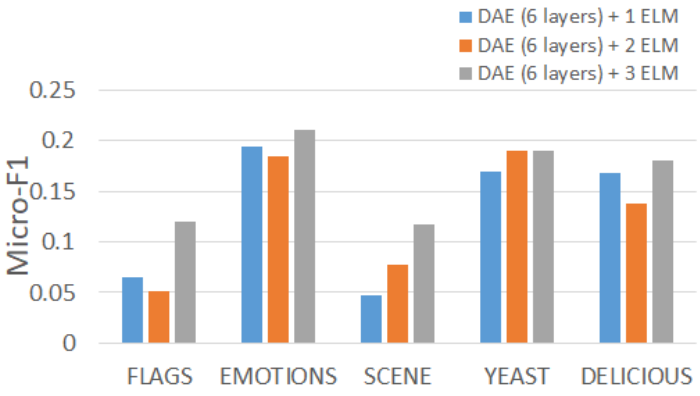


Figure 3.15: Micro-F1 across all datasets with increasing number of MLELM networks

Varying St-MLELM depth - Here, the depth of the DAE has been kept constant and the number of MLELMs in the stack has been increased from 1 to 3. A total of six different network configurations has been shown on the five datasets. Figure 3.14 shows Macro-F1 results for DAE (4 layers). For the training network, DAE contributes 4 layers, and one MLELM contributes 2 layers each. Thus, the size of the entire network is varied

from 6, 8 and 10 layers. Figure 3.15 shows the Micro-F1 results for DAE (6 layers). The size of the training network is varied from 8, 10 and 12. In most cases, the performance is seen to improve with the increase in the number of MLELMs.

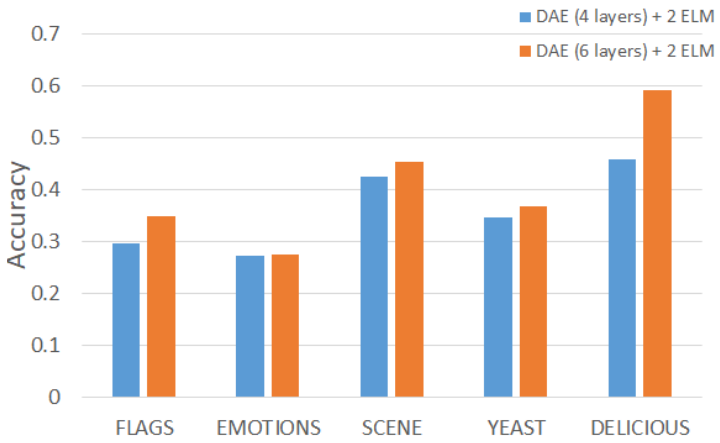


Figure 3.16: Accuracy score with increasing depth of DAE, keeping the number of ELM network constant

Varying DAE depth - Figure 3.16 depicts the variation of accuracy measure when the MLELM stack is fixed at 2, and the depth of DAE is changed from 4 to 6 layers. This makes the train network depth 8 and 10 layers. Here, the performance of the network is seen to improve with the increase in depth.

3.4.2.2 DAE vs SAE

To justify the use of DAE in the network for feature extraction, a comparison with SAE has been done. Figure 3.17 shows that in all the cases, DAE performs better than SAE. This is due to the greedy nature of learning for SAE, which learns one AE at a time and stacks them. However, for DAE the entire network updates its weights simultaneously to improve its learning.

3.4.2.3 Comparison with state-of-the-art methods

Finally, the results for the best model with respect to some state-of-the-art techniques have been shown in Table 3.11 for six performance measures on five datasets. Com-

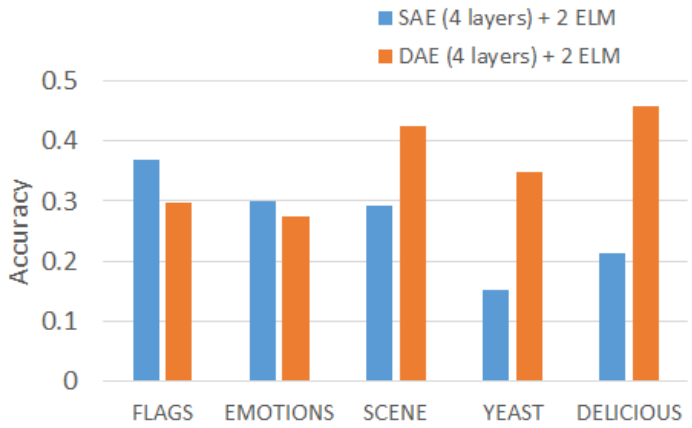


Figure 3.17: Accuracy score of DAE vs SAE

parisons have been made with well-known ML classifiers, MLELM, bipolar MLELM (B-MLELM) [81], ELM-ML [73], BR [46], CC [62] and RAKEL [79]. MLELM and B-MLELM are single ELM models, ELM-ML is a two ELM system and BR, CC, RAKEL are ensemble models which use multiple classifiers. Here, it is seen that the proposed model performs comparatively better than the others for some metrics. The ensemble models, RAKEL and CC are seen to perform close to the proposed method in few cases.

Additionally, from the T-test statistical analysis of the proposed method against the state-of-the-art algorithms it is seen that MLDAEELM outperforms the others for $t_{0.75} = 0.741$ with degrees of freedom = 4 for the average precision metric. For the two-tailed Wilcoxon signed-rank test on the accuracy metric, for alpha=0.20, $T_{Wilcoxon(5)} = 2$, MLDAEELM also outperforms the other state-of-the-art methods.

3.5 Conclusion

This chapter proposed novel multi-label classifiers that could handle large input dimension and complex decision space. For dimensionality reduction purpose, autoencoders were used, whereas for enhanced input to output mapping multi-label extreme learning machines were incorporated. ELMs are well-known for their one-pass fast classification capability that helped to make the proposed network less bulky. AEs have also gained

Table 3.11: Comparison of proposed model with the state of the art networks for all the datasets

Dataset	Methods	HL	OE	RL	AP	A	SA
	MLELM	0.1819	0.7713	0.4699	0.4430	0.0010	0.0014
	B-MLELM	0.1778	0.7672	0.4676	0.4471	0.0007	0.0010
	ELM-ML	0.2763	0.7905	0.4722	0.4339	0.1815	0.1916
Scene	BR	0.1350	0.7469	0.2320	0.6870	0.5340	0.4236
	CC	0.1440	0.3890	0.2200	0.6540	**0.5840**	**0.5310**
	RAKEL	0.2210	0.3810	0.1710	0.7450	0.5140	0.2290
	MLDAEELM	**0.1339**	**0.3780**	**0.1667**	**0.7731**	0.4356	0.3094
	MLELM	0.4029	0.4165	0.4162	0.6598	0.4045	0.5242
	B-MLELM	0.3985	0.3158	0.3754	0.7227	0.4504	0.5289
	ELM-ML	0.5201	0.6410	0.5256	0.5877	0.3226	0.4476
Flags	BR	**0.2530**	0.3090	0.2690	0.6710	**0.6060**	0.1750
	CC	0.2720	0.2520	0.2940	0.7720	0.5860	0.2580
	RAKEL	0.2830	0.2680	0.2670	0.7490	0.6120	0.1490
	MLDAEELM	0.3308	**0.2432**	**0.2432**	**0.7885**	0.4308	**0.5360**
	MLELM	0.4138	0.5678	0.4039	0.6037	0.3024	0.3960
	B-MLELM	0.4034	0.5882	0.3943	0.5996	0.3335	**0.4272**
	ELM-ML	0.4774	0.6186	0.4683	0.5475	0.2822	0.3757
Emotions	BR	0.3640	0.4470	0.3270	0.6190	0.4220	0.1720
	CC	0.3610	0.4200	0.2940	0.5790	0.4670	0.2210
	RAKEL	0.3450	**0.3170**	0.3230	**0.7100**	**0.4710**	0.0930
	MLDAEELM	**0.3366**	0.5248	**0.2782**	0.5567	0.3831	0.3421
	MLELM	0.2577	0.2941	0.2938	0.6019	0.4751	**0.5455**
	B-MLELM	0.2578	0.2964	0.2885	0.6012	0.4595	0.5309
	ELM-ML	0.2528	0.2983	0.2782	0.6024	**0.5292**	0.4442
Yeast	BR	0.2510	0.4380	0.3150	0.5820	0.4200	0.0620
	CC	0.2660	0.4820	0.3000	0.5330	0.4250	0.1440
	RAKEL	0.3200	0.4140	0.2860	0.5510	0.4250	0.0480
	MLDAEELM	**0.2462**	**0.2893**	**0.2686**	**0.6127**	0.4135	0.1534
	MLELM	0.0207	0.4915	0.2073	0.2663	0.2998	0.1552
	B-MLELM	0.0204	0.4845	0.2074	0.2628	0.2985	0.1532
	ELM-ML	0.0234	0.4519	0.1588	0.2512	0.3840	**0.2629**
Delicious	BR	0.0182	0.3414	0.3824	0.2681	0.3089	0.0118
	CC	0.0187	0.3459	0.1781	0.1641	0.3467	0.0255
	RAKEL	0.0182	0.3426	0.2002	0.1881	0.3429	0.0261
	MLDAEELM	**0.0181**	**0.3196**	**0.1406**	**0.2841**	**0.6082**	0.1931

Table 3.12: T-test statistics for all the algorithms against MLDAEELM (based on average precision)

Method	**T-test value**
MLELM	1.3377
B-MLELM	1.2224
ELM-ML	1.9602
BR	1.2452
CC	1.6454
RAKEL	0.6951

Table 3.13: Two-tailed Wilcoxon signed-rank test statistics for all the methods vs proposed MLDAEELM (based on accuracy)

Method	**p-value**	**Value of sign rank**
MLELM	0.1875	13
B-MLELM	0.3125	12
ELM-ML	0.3125	12
BR	0.6250	5
CC	0.6250	5
RAKEL	0.6250	5

popularity as great unsupervised feature extractors. Utilizing these concepts, first, a three-phase model was developed including stacked AEs and two MLELMs. This model was further enhanced to create a deeper network having a two-phase system that utilized the same principles as above. Here, instead of SAE, a deep AE was used, that experimentally seemed to be better at feature transformation and reduction. Then a stack of MLELMs was created that would sequentially learn the input to output mapping thoroughly. Both the proposed models showed a good performance in comparison to various existing models and were able to handle the intended drawbacks of ML data.

Once the approach of enhanced feature to class mapping has been done in this chapter, it is seen that there might be some scope of improvement from the perspective of learning the decision boundaries. The MLELMs employed here, for learning the feature to class mapping better, have good generalization capability and are able to learn in one pass. However, another approach of handling the decision space also needs to be explored. That is to increase the separability of the decision boundaries in the output space. To explore this idea further, a transformation based network has been explored in the next chapter.

Chapter 4

Functional Link Artificial Neural Network based Multi-label Classifiers

4.1 Introduction

Due to multiple classes being linked with each multi-label data, there is an increase in the complexity of the decision space. The class boundaries are much more convoluted and overlapping due to the increased generality of ML classification. Focussing on the overlapping boundaries of multi-label data, it would be beneficial if the separability of the decision space could be improved. In this context, while solving the problem of classification in general, various models are being developed and explored by researchers. Among such methods, artificial neural networks have shown the capability of solving very complex, non-linear data relationships [27]. As seen in the previous chapter, the flexibility exhibited by neural networks (NNs) makes it capable of learning any type of pattern. They are loosely inspired by how the human brain works; specifically, its interconnected

neuron structure and its activation resembles the biological synaptic connections and its firing. This makes it capable of handling complex classification tasks. Additionally, they are quite capable of handling complexities without unnecessarily creating a bulky model. They inherently bring non-linearity by increasing the number of layers in the model which are able to follow disparate structures in the datasets.

In the ML domain, some of the existing NN models are computationally expensive due to their complex architecture, while few of them are not able to efficiently classify multi-label data due to a very simplistic approach. Keeping these issues in mind, the aim is to adopt a simple yet efficient architecture for the class prediction of multi-label data. A model which is simplistic like the extreme learning machine adopted in the previous chapter, but has different strengths which can be explored and enhanced to build a strong classification model for multi-label data. Among the various networks explored for single-label classification, functional link artificial neural network (FLANN) is one such network that is compact yet efficient. Multiple FLANN adaptations for single-label classification [14, 17, 48] exist in the literature, but it is yet to be sufficiently experimented with in the multi-label domain. FLANNs are feed-forward networks where the use of a hidden layer is omitted by non-linearly transforming the input features with some basis functions. The expanded input layer portrays a higher dimension projection of the input with better discriminating characteristics. Multi-label data is inherently quite complex, mostly due to its multiple overlapping class boundaries. This calls for models that can handle this bottleneck and improve on it. Therefore, FLANN is an apt choice in this scenario.

In this chapter, two specific drawbacks of ML data are to be handled. First is the complex decision space with overlapping class boundaries. To overcome this issue, the FLANN model is explored. Later in the chapter, AEs are incorporated with FLANN to perform additional feature transformation and dimensionality reduction. The works are split into three parts. First, a multi-label FLANN has been proposed. The model is structurally quite simple, thus involving less computational complexity, but incorporates a higher di-

mension projection of the features which make the output space more discriminated, thus leading to efficient classification. In the second part, various models of FLANN has been devised to analyse each of their importance as a multi-label classifier and identify the optimal configuration of the MLFLANN model. Three types of functional expansions viz. trigonometric, Chebychev and power polynomial expansions and two weight optimization techniques, namely, backpropagation and particle swarm optimization (PSO) have been explored. The combination of one functional expansion and one weight optimization technique has led to six FLANN models for multi-label classification. These multi-label FLANN models have been tested on four datasets and ten performance metrics to appreciate the overall performance for each of them. Comparative analysis indicates optimal performance from a few of the models over the others making them efficient multi-label classifiers.

Finally, a novel two-layer transformation network is constructed that is adapted from MLFLANN and the well-known autoencoders (AE). This AutoEncoder integrated MLFLANN (AE-MLFLANN) network is capable of overcoming a few drawbacks faced in multi-label classification previously. In the first layer of the network, functional expansion of features is performed that is inspired by MLFLANN. The input features are functionally expanded to a higher dimension, thus giving rise to a decision space with increased separability. It is an attempt to improve the input space, thereby, increasing the convergence. However, the transformed data helps to improve the multi-label feature space only to some extent. The expansion of input space might not always give rise to an optimal representation. There is still some scope for further transforming the data which will lead to more improved performance. Also, the functional expansion leads to an increase in the input dimension, which poses a problem for multi-label data. To handle both these issues, a second feature transformation-cum-reduction layer incorporating autoencoders is introduced. The second layer is created from the encoder section of an AE, which is capable of transforming the features to a comparatively reduced and improved space. Autoencoders are widely known to implicitly extract features for classification tasks, while successfully

transforming the data. It is capable of generating a suitable representation through unsupervised learning. To concisely describe the proposed network, it can be said that, the input features are functionally expanded in the first layer. In the second layer, these expanded features are passed through an AE which generates a favourable representation in a reduced feature space. These reduced and transformed features are then mapped to the output layer. This AE-MLFLANN model is capable of good multi-label classification as it can handle the complex decision space better by transforming the data through two consecutive layers. The proposed work has highlighted the importance of the two transformation layers, which can be extended further to build deeper networks in future. AE-MLFLANN has been experimentally shown to perform better than six benchmark methods over five multi-label datasets. The application of the proposed method has been further explored from multi-label domain to traditional single-label domain as well. This single-label version of the proposed model, named AutoEncoder integrated single-label FLANN (AE-SLFLANN), has been separately tested on four relevant datasets to analyse its success.

The contribution of this chapter can be highlighted as follows.

- Introducing a novel multi-label functional link artificial neural network (MLFLANN) model that deals with the complex decision space of multi-label data by improving separability among classes.
- Various combinations of basis functions and learning techniques are explored to identify the optimal configuration of MLFLANN.
- Introducing a novel two-layer network based on MLFLANN and AE specifically for multi-label classification.
- Improving separability in multi-label data by first applying functional expansion layer, followed by additional transformation by autoencoder layer.
- The increased feature dimension caused by the first layer is reduced by consecutive

AE in the second layer. This maintains a balance between the feature space and sample size, which leads to a good training of the classifier with limited data.

- Introducing the single-label variation of this novel two-layer network.

The rest of the chapter is organized as follows. Section 4.2, describes the traditional FLANN model and some related works. In Section 4.3, 4.4 and 4.5 the proposed works are discussed in details. Description of the proposed model and experimental result have been included for the individual works. Section 4.6 concludes the chapter.

4.2 Preliminaries

In this section, the existing FLANN model is briefly discussed.

4.2.1 Functional Link Artificial Neural Network (FLANN)

Neural networks are widely used to handle complex classification problems. Various models of ANNs have been used in the past to solve different types of problems. Functional link artificial neural network (FLANN) is one such neural network model, which is simple yet efficient and has been used to solve classification tasks. FLANN is a flat feed-forward neural network with a functionally expanded input layer, no hidden layers and an output layer with one neuron. It follows a simple learning rule and uses the single error generated by the network to train itself iteratively. Its low architectural complexity makes it easier to train and helps to gain more insight into the classification problem. FLANN uses functionally expanded features to increase the dimensionality of the input data, thus overcoming the non-linear nature of the given problem. From Cover's theorem [12], it is known that given a set of training data that is not linearly separable, one can with high probability transform it into a training set that is linearly separable by projecting it into a higher-dimensional space via some non-linear transformation. Hence, the hyper-planes that are generated by FLANN should be able to efficiently discriminate between the input

patterns.

In literature, it is seen that FLANN has been modelled for single-label data classification. The initial idea of FLANN originated in [52] and [54], where it has been shown that functional links neurons may be conveniently used for function approximation with lesser computational load and faster convergence rate than multi-layer perceptron. Various models like random vector functional link networks (RVFLN) [53] have been developed alongside FLANN which might have certain similarities in the overall structure. However, the functional expansion concept of FLANN makes it somewhat unique. Over the years, FLANN has proven quite effective in classification and has been combined with other popular models from genetic algorithms in [15, 16] and PSO in [17] for enhancing the classification accuracy. Different combination of basis functions and learning mechanisms have been experimented with in the past. In [48], FLANN was used with backpropagation (BP) learning for training where a different set of orthonormal basis function was suggested for trigonometric feature expansion. The method developed in [17] also uses expansion but instead of BP, particle swarm optimization (PSO) with Cauchy and Gaussian mutation operator has been used for training of the weights. [14] used a Chebychev expansion along with an adaptive version of PSO (aPSO) combined with BP as their learning scheme for single-label classification. The initial weights were first found using aPSO and then this was used as a starting point for the BP learning for fine-tuning.

4.3 Proposed Multi-label FLANN

Exploring the various domains and variations of FLANN, it was seen that this network has proven to be quite efficient in single-label classification tasks, thus should be explored in the multi-label domain as well. The major characteristic of FLANN by which it projects the input vector efficiently to a higher dimension to improve separability makes it quite suitable for multi-label data. The class boundaries of multi-label datasets are inevitably overlapped and the data eventually seems to be quite difficult to classify. Projecting the

input vectors to a higher dimension might make this problem comparatively simpler. In this work, an adaptation of functional link artificial neural network has been proposed for class prediction of multi-label data where the functional expansion of features helps to generate hyperplanes that have a higher discrimination capability suitable for multi-label data.

4.3.1 Model Description

The details of the proposed model describing the architecture, the training and the testing phases are discussed here.

4.3.1.1 Architecture of the network

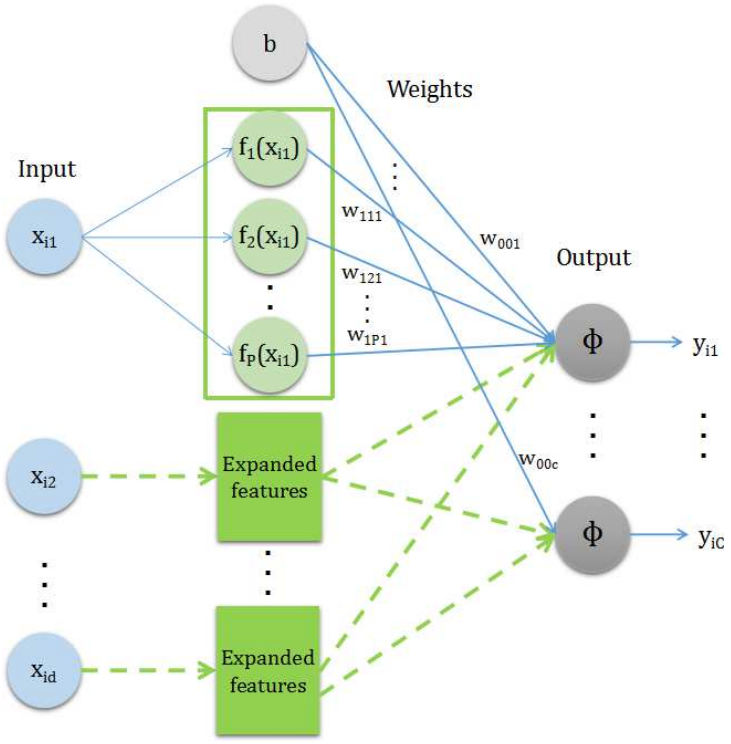


Figure 4.1: Architecture of MLFLANN

The architecture of the existing FLANN has been modified in the proposed work to incorporate classification of multi-label data (Figure 4.1). The basic feed-forward network model has two layers, the expanded input layer and the output layer. The actual input to the network has d features. Each input feature x_{ij} is functionally expanded as

$\{f_1(x_{ij}), f_2(x_{ij}), ..., f_P(x_{ij})\}$, where P is the total number of basis functions used for each input element. This increases the dimension of the input space from d to $P * d$, by a set of basis functions F applied on an input pattern X_i. $F(X_i)$ can be expanded as,

$$\begin{aligned} F(X_i) &= \{f_1(x_{i1}), f_2(x_{i1}), .., f_p(x_{i1}), \\ &\quad f_1(x_{i2}), f_2(x_{i2}), .., f_p(x_{i2}), .., \\ &\quad f_1(x_{id}), f_2(x_{id}), .., f_p(x_{id})\}. \end{aligned} \tag{4.1}$$

Once the new expanded features are obtained from the input data, they are then fed to the network. Since there are no hidden layers, the other layer in the network is the output layer. Unlike the existing FLANN which has only one output node, the proposed MLFLANN has C output neurons, one for each class. Since the existing FLANN architecture generates only one global error, representation of multiple outputs and learning multiple labels is not possible using the earlier network architecture. Hence, adaptation is necessary. The proposed model is a feed-forward network, with $d * P * C$ number of connections between the input layer and the output layer.

4.3.1.2 Training Phase

In the proposed MLFLANN architecture, a set of basis functions F, and a fixed number of weight parameters W have been used to represent the output $\vec{Y}$. The output of multi-label data can be represented as a vector of individual class outputs, i. e., $\vec{Y_i} = \{y_{i1}, y_{i2}, ..., y_{ic}\}$. With a specific set of basis functions F, the challenge is to find the weight parameters W that provide the best possible approximation of $\vec{Y}$ on the given input-output samples. This can be achieved by iteratively updating W.

At the beginning of the training phase, the network weights are initialized randomly. Then, the input patterns $\vec{X_i}$ are fed to the MLFLANN one at a time. Each of the input features is functionally expanded; some trigonometric basis functions have been used in

the problem to expand an input feature x_{ij}:

$$\begin{aligned} F(X_i) &= \{\sin\pi(x_{ij}), \cos\pi(x_{ij}), \\ &\quad \sin 2\pi(x_{ij}), \cos 2\pi(x_{ij}), \\ &\quad ..., \\ &\quad \sin m\pi(x_{ij}), \cos m\pi(x_{ij})\} \end{aligned} \tag{4.2}$$

A weighted sum of these non-linear outputs is computed through the network. The induced local fields for each class is obtained by adding a bias b to this sum. An activation function ϕ is applied at each of the output nodes to obtain the estimated outcomes for all the classes. The output vector $\vec{Y_i'}$ can be written as,

$$\vec{Y_i'} = \{y'_{i1}, y'_{i2}, ..., y'_{iC}\}. \tag{4.3}$$

$$y'_{ic} = \phi\left(\sum_{j=1}^{d}\sum_{p=1}^{P} f_p(x_{ij}).w_{jpc} + b\right), \quad 1 \leq c \leq C. \tag{4.4}$$

This actual output $\vec{Y_i'}$ is compared to the corresponding desired output $\vec{Y_i}$ and the resultant error vector $\vec{E}$ for the i^{th} pattern is,

$$\begin{aligned} \vec{E} &= \vec{Y_i} - \vec{Y_i'} \\ &= \{y_{i1} - y'_{i1}, y_{i2} - y'_{i2}, ..., y_{iC} - y'_{iC}\} \\ &= \{e_1, e_2, ..., e_C\} \end{aligned} \tag{4.5}$$

At the $t+1^{th}$ iteration, the weight matrix W is updated depending on the error computed at the t^{th} iteration. The change in weight $\triangle W_j^{(t)}$ given by,

$$\triangle W_j^{(t)} = \mu f_j(x)^{(t)}\delta^{(t)}, \tag{4.6}$$

where, μ is the learning rate, $f_j(x)^{(t)}$ is the functionally expanded input at the t^{th} iteration

and the gradient is,

$$\delta^{(t)} = (\vec{Y'})(1 - \vec{Y'})\vec{E}, \tag{4.7}$$

where, $\vec{Y'}$ is the output vector and $\vec{E}$ error vector at the output layer. Then the connection weights for the $t + 1^{th}$ iteration can be updated as,

$$W_j^{(t+1)} = W_j^{(t)} + \triangle W_j^{(t)}, \tag{4.8}$$

where, $W_j^{(t)}$ is the j^{th} weight at the t^{th} iteration.

At the end of the training phase, the learned classifier is able to generate a set of outputs for a given pattern, but this output vector needs to be mapped to a label set. In the case of multi-label data, a suitable threshold needs to be selected that will be able to correctly map all the class labels. Instead of selecting a single threshold, a set of thresholds is determined from the outputs obtained from the trained classifier. A set of C thresholds are computed by averaging the relevant and irrelevant scores obtained for each class.

4.3.1.3 Testing Phase

Once the MLFLANN is well trained, the validation/testing phase begins. In this phase, each multi-label test pattern is taken at a time and fed to the trained MLFLANN. The same set of basis functions are used to expand the features of the test pattern. The trained network computes the output at each node which denote each class. Once the outputs have been obtained, the heuristically determined threshold is applied to each of the class outputs, which give the actual results.

4.3.2 Experimental Analysis

To assess the performance of the proposed approach, experiments have been conducted on four multi-label datasets and has been compared to two other NN-based multi-label classification algorithms. One is a multi-label single-layer perceptron (MLSLP) model

which works without modifying the input features. The other model used for comparison is MLRBF [100], this technique modifies the input features using radial basis functions. Results corresponding to the above two techniques and the proposed work MLFLANN have been shown in Table 4.1.

Table 4.1: Comparative results on four datasets

Dataset	Method	**HL ↓**	**AP ↑**	**RL ↓**	**Cov ↓**	**OE↓**
	MLSLP	0.1638	0.5884	0.2941	1.5627	0.6149
Scene	MLRBF	**0.0771**	**0.8877**	0.1678	**0.4299**	**0.1834**
	MLFLANN	0.1185	0.8251	**0.1106**	0.6476	0.2794
	MLSLP	0.2378	0.7076	0.2242	7.4033	0.2521
Yeast	MLRBF	0.2361	0.7278	0.2228	7.1468	0.2105
	MLFLANN	**0.2246**	**0.73771**	**0.2129**	**7.0105**	**0.1898**
	MLSLP	0.2211	0.7873	0.1928	2.0169	0.2798
Emotions	MLRBF	**0.1936**	0.7961	0.1616	1.8233	**0.2583**
	MLFLANN	0.2031	**0.7983**	**0.1607**	**1.8099**	0.2677
	MLSLP	0.2867	0.3297	0.4603	169.3410	0.2980
CAL500	MLRBF	0.3045	0.2435	0.4725	168.5221	0.3146
	MLFLANN	**0.1872**	**0.4152**	**0.2678**	**154.7961**	**0.1245**

As performance measuring indices, Hamming loss, average precision, ranking loss, coverage and one error have been used [28]. To compute an average performance, the 5-fold cross-validation results have been recorded. From the results obtained, it is seen that the proposed method performs better than the single-layer perceptron model for all the datasets. It is safe to say that the functional expansion in the proposed model has made the multi-label data more discriminable. On the other hand, MLFLANN and MLRBF have comparable performance. Both the techniques do not use the input features as it is and perform some transformation of the input space to make the classification task simpler. MLFLANN performs better than MLRBF for Yeast and CAL500 datasets and is a close competitor for the other two datasets. The results strengthen the fact that functional expansion of features and iteratively adapting the network weights in MLFLANN has proven to be beneficial for the classification of multi-label data.

4.4 Proposed Optimization of MLFLANN

From Cover's theorem [12], it can be said that a higher dimensional transformation makes the original data more separable, thus, simplifying the task of classification. This is a primary reason for exploring FLANN models to handle the complex multi-label data. The work in this section is aimed to optimize the MLFLANN proposed in Section 4.3. Multiple variations of single-label FLANNs exist in the literature, which also needs to be explored to identify the optimal configurations in the multi-label domain. Thus, in the proposed work, three basis functions, namely, trigonometric, Chebychev and power polynomial have been focused upon, along with two learning approaches – backpropagation and particle swarm optimization (PSO). The current work combines the basis functions and learning mechanisms to build six models of FLANN for multi-label classification. Details of the approaches have been discussed in the following sections.

4.4.1 Model Description

The proposed variations of the MLFLANN are discussed in detail.

4.4.1.1 Architecture

The basic model of multi-label FLANN is shown in Figure 4.1. It contains an input layer for d features $x_{i1}, ..., x_{id}$. These input features are expanded by the functional expansion unit with the help of some basis functions which introduce non-linearity in the data. Now, each input feature x_{ij} is expanded to p corresponding features which represent the final amplified input. There are various types of basis functions for traditional FLANN in the literature that serve different purposes. Few of them have been discussed in Section 4.4.1.2. After expansion, the features are directly mapped to the C nodes of the output layer with the help of weighted connections. The nodes at the output layer compute the predicted score for each class at the corresponding nodes. A weighted sum of the expanded inputs is computed at the output layer. This predicted score is iteratively

moved towards the target scores by updating the weights and reducing the error. These weights W are learned iteratively during the training phase of the network. Researchers have incorporated different weight optimization mechanisms, two of which are discussed in Section 4.4.1.3.

4.4.1.2 Basis functions

This section describes the three basis functions explored in this work.

Trigonometric The trigonometric expansion is one of the most popular basis functions for FLANN. Trigonometric orthonormal basis, *sin* and *cos* functions are used for this expansion. According to [14], a trigonometric basis function forms a more compact representation than the other functions. For all the polynomials of the n^{th} order with respect to an orthogonal system $\varphi(u)_{i=1}^{n}$, the best approximation in the metric space of L^2 is given by the n^{th} partial sum of its Fourier series with respect to the system. The expansion representation with input feature X_i used for the current work is as follows:

$$X_i' = \{X_i, \sin(\pi X_i), \cos(\pi X_i), \sin(2\pi X_i), \cos(2\pi X_i), \ldots, \sin(q\pi X_i), \cos(q\pi X_i)\} \quad (4.9)$$

This function was used for expansion in the proposed model in Section 4.3.

Chebychev This series is easier to calculate in comparison to the trigonometric functions. As stated in [14], the non-linear approximation capacity of the Chebychev orthogonal polynomial is very powerful by the best approximation theory. Chebychev functions are orthogonal for the range $[-1, 1]$. The finite set of Chebychev polynomials can be generated in the following recursive way for an input feature X_i:

$$X_i' = \{1, X_i, \{2 \cdot X_i \cdot f_1(X_i) - f_0(X_i)\}, \ldots, \{2 \cdot X_i \cdot f_{q-1}(X_i) - f_{q-2}(X_i)\}\} \quad (4.10)$$

Power Polynomial The use of power polynomial basis functions for expansion is quite simple. Any polynomial functional form can be used for this approach. In the current work the expansion has been carried out in the following way:

$$X_i' = \{1, X_i, X_i^2, \ldots, X_i^q\} \tag{4.11}$$

4.4.1.3 Learning Mechanisms

The subsequent section gives a brief description of two learning techniques widely used for FLANN.

Backpropagation The backpropagation mechanism is one of the most popular weight optimization techniques for neural networks. FLANN models for single-label classification [48] are known to use this learning technique quite often. Gradient descent provides a computationally efficient method of changing the weights in a feed-forward network with differentiable activation function units to learn a training set of input-output pairs. In this work, the network weight matrix, $\vec{W}$, is initialized randomly with values close to zero. The input patterns, $\vec{X}$, are then fed to the network iteratively. A weighted sum of the functionally expanded non-linear inputs, X', is added with a bias term, b. The summed result is then subjected to an activation function $\phi(.)$ in the output neurons. The output vector, $Y_i' = [y_{i1}',\ y_{i2}', ...,\ y_{iC}']$ for the i^{th} pattern, representing the predicted labels, is then obtained. For N data instances or patterns, the labels are obtained by mapping the input to the C output dimensions as follows:

$$y_c' = \phi\Big(\sum_{k=1}^{D'} X_k' \cdot W_{kc} + b\Big),\ where\ 1 \leq c \leq C. \tag{4.12}$$

The error E_i is the calculated using the original set of labels, for the i^{th} pattern as:

$$E_i = \ Y_i \ - \ Y_i^{'} \tag{4.13}$$

The weights are then updated using gradient descent. The change in weights, ΔW^t, which is a set of weight vectors, ΔW_{kc}^t at t^{th} iteration is given as,

$$\Delta W_{kc}^t = \mu \cdot X_k^{'t} \cdot \delta^t, \tag{4.14}$$

where, μ is the learning rate and the gradient δ^t at t^{th} iteration is given as,

$$\delta^t = Y^{'} \cdot (1 - Y^{'}) \cdot E \tag{4.15}$$

The updated connections weights, W at the $(t+1)^{th}$ iteration is given by,

$$W^{t+1} = W^t + \Delta W^t \tag{4.16}$$

At the end of the learning phase, the model is able to generate class scores using Equation (4.12) at the output nodes in the range [0, 1]. These scores are then converted to relevant labels for each data instance by using a global threshold.

Particle Swarm Optimization (PSO) PSO [20] is a population-based search procedure in which individuals called "particles" change their position with time, each particle in the swarm represents a "solution" to the optimization problem. The particles fly around in a multi-dimensional search space following the personal best and the global best particle positions. As stated above, a particle referred to here is a potential solution to the optimization problem. Thus, a particle k in the multi-label FLANN scenario is represented with two parameters, its velocity and position. The velocity v_k represents the movement of the particle in space towards the optimal solution. Whereas, the position of a particle is represented by the weight matrix of the FLANN, W_k, in the C-dimensional problem space. The particle position is denoted as a vector $\vec{W_k} = (W_{k1},\ W_{k2},\ ..., W_{kC})$, for the k^{th} particle out of all P randomly initialized ones in the problem space, i. e., the swarm. Each k^{th} particle maintains records of its personal best position, $\vec{W_k^L} = (W_{k1}^L,\ W_{k2}^L,\ ..., W_{kC}^L)$,

its current velocity, v_k, and its current position, W_k. In each iteration a global best $\vec{W}_g = (W_{g1}, W_{g2}, ..., W_{gC})$ is found based on the positions of all the members of the swarm. Any particle in the swarm then changes or moves in the space according to the information from the global best and its own personal best. In each iteration, W_k^L and W_g of the current swarm are combined with some weights γ_1 and γ_2, where γ_1 is known as the cognitive factor or the self-confidence factor and γ_2 is referred to as the social factor or the swarm confidence factor. This is done to adjust the velocities of the particles of the swarm.

Figure 4.2 represents the movement of the particle. The particle moves in the direction of the resultant vector, $\vec{W}'_k$, w.r.t the personal and global best positions. ω represents the inertia factor, and it plays a key role in the process of providing a balance between exploration and exploitation process in PSO. If the value of ω is large then PSO tends to be in the global search mode, hence, providing little resistance to the velocity. Whereas, if the value is less, then it provides greater resistance to the previous velocity of the particle, hence, tending to be in a targeted search mode.

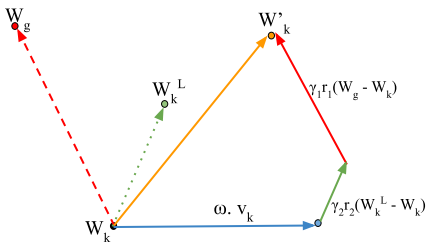


Figure 4.2: Vector Representation of PSO

Now, considering the multi-label classification problem at hand, the mean squared error (MSE) objective function has been selected for carrying out the PSO routine. Hence, the task is to minimize the MSE between the target output and the generated output. With

each particle position $\vec{W_k}$ represented as:

$$\begin{aligned}\vec{W_k} = (&W_{11}, W_{12}, ..., W_{1C},\\ &W_{21}, W_{22}, ..., W_{2C}..., \\ &W_{D'1}, W_{D'2}, ..., W_{D'C}).\end{aligned} \tag{4.17}$$

The objective function can be defined as:

$$J(W) = \frac{1}{D'} \sum_{i=1}^{D'} \sum_{j=1}^{C} ||\phi(W_{ij} \cdot X'_{ij} + b) - Y_{ij}||^2, \tag{4.18}$$

where, $\phi(.)$ is the activation function. In the current work, the activation function $\phi(x)$ is selected to be the sigmoid function. The velocity $\vec{v_k}$ and position $\vec{W_k}$ for each particle k is updated in the t^{th} iteration as,

$$\begin{aligned}\vec{v_k}(t+1) = \omega \odot \vec{v_k}(t) + \gamma_1 \odot \vec{r_1}(t) \odot (\vec{W_k^L}(t) - \vec{W_k}(t)) \\ + \gamma_2 \odot \vec{r_2}(t) \odot (\vec{W_g}(t) - \vec{W_k}(t)),\end{aligned} \tag{4.19}$$

$$\vec{W_k}(t+1) = \vec{W_k}(t) + \vec{v_k}(t+1), \tag{4.20}$$

where, $\odot$ represents the Hadamard product, $\vec{r_1}$ and $\vec{r_2}$ are vectors of random numbers which introduce randomness for search space exploitation and γ_1, γ_2 are acceleration coefficients. The personal best and global best positions for the $(t+1)^{th}$ iteration are computed as follows,

$$\vec{W_k^L}(t+1) = \begin{cases} \vec{W_k^L}(t) & \text{if } J(\vec{W_k}(t+1)) \geq J(\vec{W_k^L}(t)) \\ \vec{W_k}(t+1) & \text{otherwise} \end{cases} \tag{4.21}$$

$$\vec{W}_g(t+1) = \begin{cases} \vec{W}_g(t) & \text{if } J(\vec{W}_k(t+1)) \geq J(\vec{W}_g(t)) \\ \vec{W}_i(t+1) & \text{if for any } i \\ & J(\vec{W}_i(t+1)) < J(\vec{W}_g(t)) \\ & \forall\ i \in \text{swarm} \end{cases} \tag{4.22}$$

In the present work, the inertia factor, ω, has been changed in a linearly decreasing fashion [95], to employ exploration in the initial iterations and slowly convert it to exploitation when approaching the best particle. The linearly decreasing strategy for ω, that enhances the efficiency and performance of PSO, is defined as,

$$\omega(t) = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{itr_{max}} \times t \tag{4.23}$$

where, itr_{max}, is the total number of iterations for the PSO routine. In addition, the implemented PSO also uses the adaptive cognitive acceleration coefficient (γ_1) and the social acceleration coefficient (γ_2) [14]. γ_1 has been decreased from an initial value, γ_{1i}, to a final value, γ_{1f}, and γ_2 has been increased from γ_{2i} to γ_{2f} using the following equations,

$$\gamma_1^t = (\gamma_{1f} - \gamma_{1i})\frac{t}{itr_{max}} + \gamma_{1i} \tag{4.24}$$

$$\gamma_2^t = (\gamma_{2f} - \gamma_{2i})\frac{t}{itr_{max}} + \gamma_{2i} \tag{4.25}$$

Once the optimal weight matrix is obtained through the learning techniques, the final classification is done in the testing phase. The test data is functionally expanded and the class scores are predicted through their weighted aggregation. These predicted scores are finally converted to crisp labels using a global threshold. The following section describes the experimental findings for the different multi-label FLANN models using the basis functions and learning techniques discussed here.

4.4.2 Experimental Analysis

To evaluate the effectiveness of all the six multi-label FLANN models proposed in this section, they have been tested on four datasets with ten performance metrics. Comparison among these six methods has been made to determine their effectiveness in the field of multi-label classification. Four datasets viz. Flags, Emotions, Yeast, and Scene were used. Among the many metrics for multi-label classification [28], average precision (AP), precision (P), recall (R), hamming loss (HL), one error (OE), coverage (Cov), ranking loss (RL), micro-F1 (MicF1), macro-F1 (MacF1) and subset accuracy (SA) metrics have been used. Three basis functions – trigonometric, Chebychev and power polynomial, and two learning techniques – backpropagation and PSO were used in combination to form six FLANN models for multi-label classification. Table 4.2 provides the acronyms along with their corresponding basis function and learning mechanism.

Table 4.2: The six modelled multi-label FLANN with their corresponding basis functions and learning mechanisms

Method	Basis Function	Learning Mechanism
ML-T-PSO	Trigonometric	PSO
ML-C-PSO	Chebychev	PSO
ML-P-PSO	Polynomial	PSO
ML-T-BP	Trigonometric	BP
ML-C-BP	Chebychev	BP
ML-P-BP	Polynomial	BP

4.4.2.1 Analysis of Results

The six models for multi-label FLANN have been tested on four datasets and ten performance measures. 5-fold and 10-fold cross-validation results for the datasets are shown in Tables 4.3 and 4.4 respectively. From the results throughout all the datasets and performance metrics, ML-P-PSO is seen to outperform the other methods in most of the cases. It is followed by ML-T-BP. These two combinations of power polynomial with PSO and trigonometric with backpropagation seem to perform well for multi-label classification.

Analysis of each category of FLANN models for multi-label classification reaches a few

Table 4.3: 5 fold CV results for multi-label datasets

FLAGS	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.7848	**0.6319**	0.6467	**0.2807**	0.2814	3.9290	0.2504	**0.7101**	**0.6293**	**0.1594**
ML-C-PSO	0.7632	0.5235	0.5946	0.2998	0.2448	4.2800	0.3280	0.6797	0.5252	0.0928
ML-P-PSO	**0.7976**	0.5906	**0.6678**	0.2851	0.2499	**3.8057**	**0.2309**	0.6992	0.6023	0.1390
ML-T-BP	0.7696	0.6228	0.6291	0.3051	0.2707	4.0323	0.2739	0.6836	0.6188	0.1286
ML-C-BP	0.6695	0.4960	0.4036	0.3754	**0.2132**	4.7457	0.5289	0.6151	0.4100	0.0874
ML-P-BP	0.7503	0.4641	0.5728	0.3106	0.2189	4.4834	0.3654	0.6536	0.4661	0.0930

EMOTIONS	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.7612	**0.6547**	0.5977	0.2392	0.3289	2.0153	0.2005	0.6305	0.6206	0.2075
ML-C-PSO	0.7553	0.5655	0.6336	0.2314	0.3458	2.0253	0.2109	0.6091	0.5912	0.2310
ML-P-PSO	**0.7814**	0.6034	**0.6654**	**0.2130**	**0.2850**	1.9156	**0.1815**	**0.6437**	**0.6265**	**0.2463**
ML-T-BP	0.7619	0.6205	0.6144	0.2381	0.3423	1.9678	0.1979	0.6188	0.6133	0.2124
ML-C-BP	0.6100	0.3144	0.4610	0.3210	0.5245	**3.0389**	0.4513	0.3897	0.3499	0.0877
ML-P-BP	0.7690	0.6262	0.6343	0.2265	0.3171	2.0170	0.2023	0.6356	0.6262	0.2294

SCENE	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.8320	0.6446	0.7055	0.1143	0.2759	0.5958	0.1015	0.6649	0.6715	0.4857
ML-C-PSO	0.7055	0.4306	0.6112	0.1515	0.4474	1.1836	0.2185	0.4995	0.5000	0.3212
ML-P-PSO	0.8012	0.5768	0.6868	0.1242	0.3149	0.7537	0.1320	0.6204	0.6237	0.4549
ML-T-BP	**0.8451**	**0.7153**	**0.7377**	**0.1014**	**0.2534**	**0.5629**	**0.0942**	**0.7129**	**0.7250**	**0.5596**
ML-C-BP	0.4391	0.0959	0.2134	0.2077	0.7852	2.5148	0.6799	0.1429	0.1180	0.0719
ML-P-BP	0.7808	0.5766	0.6612	0.1295	0.3390	0.8284	0.1527	0.6124	0.6129	0.4375

YEAST	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.6344	0.4375	0.3711	0.3144	0.3670	8.3231	0.2864	0.5222	0.3913	0.0463
ML-C-PSO	0.7069	0.3257	0.4105	0.2315	0.2822	7.3090	0.2155	0.5849	0.3454	0.0861
ML-P-PSO	**0.7213**	0.3490	**0.4488**	0.2240	**0.2669**	**7.0632**	**0.2014**	0.6016	0.3714	0.1026
ML-T-BP	0.6454	**0.4406**	0.3832	0.3073	0.3583	8.3078	0.2891	0.5286	**0.4034**	0.0501
ML-C-BP	0.6083	0.2396	0.3195	0.2492	0.4382	10.3469	0.4451	0.5191	0.2513	0.0203
ML-P-BP	0.7074	0.3745	0.4467	**0.2238**	0.2784	7.9258	0.2431	**0.6099**	0.3957	**0.1117**

conclusions. Results for trigonometric expansion indicates a close competition between ML-T-PSO and ML-T-BP. This is very evident especially for the Scene dataset, where ML-T-BP exceeds all other methods, closely followed by ML-T-PSO. This shows that the trigonometric basis function improves the feature space for Scene data compared to the other basis functions. It forms a more compact representation of the data, making it easier to comprehend. Similarly, for power polynomial expansion, the ML-P-PSO model displayed a good performance for Flags, Emotions and Yeast datasets. ML-P-PSO and ML-P-BP have commendable results for the Emotions dataset, indicating polynomial expansion fruitful for this specific dataset. Also, from the results for Chebychev expansion, ML-C-PSO is seen to perform significantly better than gradient descent optimization, ML-C-BP. However, the overall trend shows that Chebychev expansion models cannot

Table 4.4: 10 fold CV results for multi-label datasets

FLAGS	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.7863	**0.6462**	**0.6416**	0.2834	0.2696	3.9326	0.2493	0.7129	**0.6325**	**0.1450**
ML-C-PSO	0.7681	0.4919	0.5307	0.3047	0.2343	4.1932	0.3241	0.6712	0.4855	0.0882
ML-P-PSO	**0.7959**	0.6155	0.6296	**0.2766**	0.2538	**3.8174**	**0.2344**	**0.7132**	0.6065	0.1137
ML-T-BP	0.7693	0.6169	0.6116	0.3136	0.2702	4.0061	0.2763	0.6773	0.6000	0.1342
ML-C-BP	0.7044	0.3880	0.3544	0.3738	0.2861	4.9176	0.5111	0.5616	0.3281	0.0413
ML-P-BP	0.7255	0.4918	0.5797	0.3120	**0.2133**	4.5521	0.3781	0.6604	0.4875	0.0671

EMOTIONS	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.7476	0.6188	0.5686	0.2556	0.3561	2.0279	0.2087	0.6034	0.5855	0.1801
ML-C-PSO	0.7543	0.5353	0.6072	0.2432	0.3491	2.0388	0.2116	0.5820	0.5575	0.1736
ML-P-PSO	**0.7856**	0.6164	**0.6549**	**0.2136**	**0.2968**	**1.8734**	**0.1809**	**0.6450**	**0.6262**	**0.2581**
ML-T-BP	0.7599	0.6216	0.5901	0.2496	0.3172	2.0328	0.2051	0.6088	0.5965	0.1838
ML-C-BP	0.6203	0.3138	0.5110	0.3153	0.5140	2.8734	0.4231	0.3945	0.3526	0.0828
ML-P-BP	0.7640	**0.6219**	0.6266	0.2305	0.3340	1.9344	0.1946	0.6305	0.6131	0.2379

SCENE	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.8285	0.6503	0.6941	0.1170	0.2771	0.6220	0.1060	0.6607	0.6685	0.4853
ML-C-PSO	0.7058	0.4281	0.6252	0.1491	0.4420	1.2002	0.2198	0.5013	0.5023	0.3299
ML-P-PSO	0.7893	0.5525	0.6933	0.1260	0.3361	0.7848	0.1390	0.6053	0.6096	0.4375
ML-T-BP	**0.8448**	**0.7216**	**0.7360**	**0.1015**	**0.2538**	**0.5650**	**0.0943**	**0.7132**	**0.7256**	**0.5525**
ML-C-BP	0.4332	0.1132	0.2768	0.2112	0.7956	2.5162	0.6890	0.1556	0.1373	0.0760
ML-P-BP	0.7903	0.5847	0.6784	0.1267	0.3294	0.8006	0.1467	0.6197	0.6231	0.4553

YEAST	AP	P	R	HL	OE	Cov	RL	MicF1	MacF1	SA
ML-T-PSO	0.6324	**0.4410**	0.3773	0.3137	0.3873	8.2594	0.2832	0.5230	0.3963	0.0443
ML-C-PSO	0.7069	0.3178	0.4057	0.2317	0.2759	7.3282	0.2154	0.5820	0.3421	0.0823
ML-P-PSO	**0.7262**	0.3422	0.4481	**0.2219**	**0.2668**	**6.9045**	**0.1954**	0.6023	0.3711	0.0985
ML-T-BP	0.6503	0.4405	0.3845	0.3032	0.3517	8.2581	0.2850	0.5335	**0.4040**	0.0567
ML-C-BP	0.6198	0.2524	0.3090	0.2429	0.4240	10.2136	0.4308	0.5383	0.2676	0.0310
ML-P-BP	0.7063	0.3831	**0.4556**	0.2226	0.2822	7.9551	0.2439	**0.6131**	**0.4040**	**0.1187**

compete with the trigonometric and power polynomial ones. The Chebychev functional expansion seems to be somewhat inadequate to bring about separability in the data. Thus, it is less suitable for multi-label classification, making trigonometric and polynomial expansions more favourable. Moving on to the learning techniques, overall, PSO is seen to perform slightly better than its BP counterparts for multi-label classification. However, the choice of parameters plays a vital role in the performance of PSO. The numbers of particles that make up the swarm in the current work, has been fixed after trial and error. A larger number of particles corresponds to a much larger computation time. The optimal value to stop the PSO (itr_{max}) is also not known, hence, it was experimentally set. Keeping in mind the drastic increase in the number of dimensions after the functional expansion, which thereby increases the overall computation time.

However, experimentally, BP was faster than PSO, while compensating slightly on its performance. Thus, both BP and PSO can be used in multi-label FLANN depending on the requirement of the problem.

4.5 Proposed Autoencoder Integrated Multi-label FLANN

In this final section, a two-layer transformation based neural network has been proposed for multi-label classification that incorporates MLFLANN and autoencoders. In the basic model devised in Section 4.3, the data undergoes a single-level functional expansion which improves the multi-label data separability to some extent. However, the single transformation provided by the basis functions seems to improve the separability and thus the classification, only to a certain extent. It is a motivation to take it a step further and introduce another layer of transformation that can further improve the classification capability of the network. It led to the conception of a two-layer network with the second transformation credited to an autoencoder. It seemed suitable to use an AE in this scenario since the aim is to transform as well as reduce the number of features. Without having to manually select features, AEs are capable of generating a good representation of the original data in a new space. Thus, the multi-label data in the proposed AE-MLFLANN undergoes two feature transformations and reduction followed by learning and final classification.

4.5.1 Model Description

The section describes the training architecture and testing phase of the proposed model.

4.5.1.1 Architecture

The proposed model employs autoencoders (AEs) in coalition with MLFLANN (AE-MLFLANN) to perform efficient multi-label classification. The architecture of the proposed model has been shown in Figure 4.3. The model consists of three phases as follows.

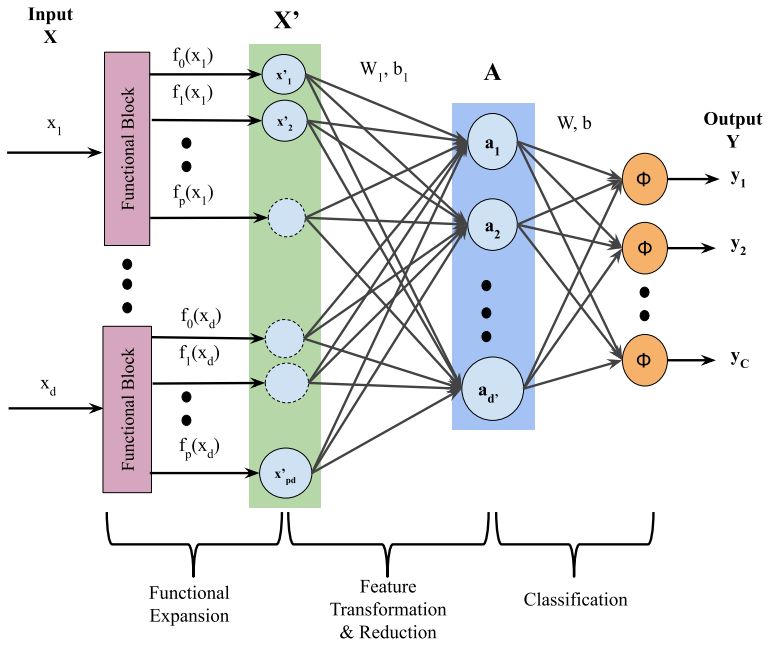


Figure 4.3: Proposed model AE-MLFLANN

Functional Expansion The first layer is inspired by a multi-label functional link artificial neural network (MLFLANN). Neural networks have been the most popular models to solve complex classification tasks. However, due to the varying complexities of problems, the number of hidden layers and neurons in the hidden layers need to be changed, giving rise to a more and more complex model. To overcome the bottlenecks that are associated with multi-layer perceptrons, higher-order neural networks (HONNs) have been considered as an alternative. It also has the ability to achieve convergence faster with a lower training complexity.

The first layer of AE-MLFLANN takes the original input and functionally expands it using some basis functions. The functional expansion acts on an element of a pattern or the entire pattern by generating a set of linearly independent functions. Thereby, it effectively increases the dimension of the input and brings a greater discriminating capability to the feature space. It comprises of d input nodes corresponding to the original input features, $X = \{x_1, ..., x_d\}$, each of which are expanded using p basis functions. This results in a

$d \times p$ functionally expanded feature space. The input pattern X is enhanced with the functional expansion block which results in the following pattern X'.

$$\begin{aligned} X' &= \{f_0(x_1), f_1(x_1), ..., f_p(x_1), \\ &\quad \vdots \\ &\quad f_0(x_d), f_1(x_d), ..., f_p(x_d)\} \end{aligned} \tag{4.26}$$

Figure 4.3 shows the functional blocks, which transform each feature, x_i, to its corresponding higher-order representation. The functional expansion performed in the first layer can be done using various basis functions, like, trigonometric, Chebychev, power polynomials, etc. Keeping that in mind the results achieved in Section 4.4.2, trigonometric basis functions have been used for functional expansion in the first layer.

The functional expansion provided by an MLFLANN is capable of transforming the data to a higher dimension but in a restricted way. The purpose of this expansion is to bring about non-linearity in the data, which is otherwise impossible for a shallow neural network to achieve. However, this expansion alone can only introduce a restricted amount of separability in the complex decision space of multi-label data. If another level of transformation can be introduced in the data, it might be even more separable. To achieve this, an AE is to be used.

Autoencoder Transformation and Feature Reduction Once the new set of features, X', are obtained through functional expansion, they are then fed to an autoencoder. The AE brings about another level of implicit transformation in the network, which is capable of encoding the input features into a new set of features A. The aim of using an autoencoder is to learn an efficient data representation or encoding in an unsupervised manner, typically for dimensionality reduction. The structure of an AE has been discussed in Section 3.2.1. It is trained to reconstruct the input as closely as possible. The

intermediate layer learns an encoded latent representation of the input in the course of its training. It performs the task of an encoder, whereas, the decoding happens from the intermediate layer to the output layer. The aim of such a network as mentioned earlier is to minimize the error between the input vector and the output vector. If the number of neurons in the encoding layers are lesser than the original number of features, the autoencoder is said to be under-complete and if it is greater than the input it is said to be over-complete. It manipulates the dimension of the encoded features, thus leading to dimensionality reduction or expansion. The working of the AE is quite similar to that of a multi-layer perceptron. The forward propagation which maps the input vector X' to the latent representation $A \in \mathbb{R}^{d'}$ and then maps A to the output X'' (reconstruction) can be formulated as:

$$A = \sigma_1(W_1 \cdot X' + b_1) \tag{4.27}$$

$$X'' = \sigma_2(W_2 \cdot A + b_2) \tag{4.28}$$

Here, W_1 and b_1 represents the weight matrix and bias from input to hidden layer and W_2 and b_2 represents the weight matrix from hidden layer to reconstruction layer. σ_1, σ_2 represents the activation functions which can be chosen from sigmoid activation, rectified linear unit (ReLU), hyperbolic tangent function, etc. The AE iteratively modifies the weights in the network using backpropagation, to finally learn to reconstruct the original input in the output layer.

In the AE-MLFLANN model (Figure 4.3), the autoencoder transformation is incorporated by the use of the encoder weights, W_1, b_1, from the trained AE (Figure 3.1). These weights are for the connections between the first transformation layer to the second. X' can be now modified to A by using the autoencoder weights, just like it was done in the standalone autoencoder.

Along with feature space transformation, dimensionality reduction is also performed with the help of the AE. In general, AEs are known to extract good features from data. If

the size of the intermediate layer of the AE is varied, it might result in feature expansion or reduction. However, in the proposed scenario, to handle the already expanded feature space by the basis functions, feature reduction by the AE is opted. This helps with the additional bottleneck of increased input dimensionality in the previous layer. This was a drawback of MLFLANN which is being overcome in AE-MLFLANN. This feature encoding reduces the size of the network at this stage, thus in turn, having lesser weights to learn. Once the data has been transformed completely, the encoded and reduced representation $A = \{a_1, a_2, ..., a_{d'}\}$, is further connected to the output layer of the network. These d' nodes are fully connected to the C nodes of the output layer of the AE-MLFLANN (Figure 4.3). The weights W, b connecting the AE transformation layer to the output layer are learnt in the training phase.

Learning In the final phase of the network, the last layer weights are updated. These are the connections between A and Y layers, which have been initialized randomly. In AE-MLFLANN, backpropagation has been used as the learning mechanism to update the weights in the training phase. First, the original input X is sequentially transformed to X', then A in the transformation phase. Once the transformation is done, the output in the last layer is computed as:

$$Y'_j = \phi\left\{\sum A \cdot W_j\right\}, \tag{4.29}$$

where ϕ is the activation function of the output layer. Thus, for each data instance, a corresponding label-set is obtained as,

$$Y_i^{'} = [\ y_{i1}^{'},\ y_{i2}^{'}, ...,\ y_{iC}^{'}\ ], \tag{4.30}$$

If, Y is the target label, the error, E, calculated using the original set of labels is,

$$\begin{aligned} E &= Y_i - Y_i^{'} \\ &= [\ y_{i1} - y_{i1}^{'},\ y_{i2} - y_{i2}^{'}, ...,\ y_{iC} - y_{iC}^{'}\ ] \end{aligned} \tag{4.31}$$

The weights are updated using this error with the backpropagation technique discussed in Section 4.4.1.3. The weights are updated iteratively throughout the training phase. Once the training is complete, the trained network can now predict multi-label outputs for any unknown data sample.

4.5.1.2 Testing AE-MLFLANN

In the testing phase, an unknown sample is fed to the AE-MLFLANN. This input is first functional expanded from X_{test} to X'_{test}, then the autoencoder transformation-reduction is processed.

$$A_{test} = \sigma_1 \left\{ \sum W_1 \cdot X' + b_1 \right\}, \tag{4.32}$$

where, σ_1 is the activation function of the encoder. At the final output the classification scores are predicted as,

$$Y'_{test} = \phi \left\{ \sum W \cdot A + b \right\}. \tag{4.33}$$

The scores are then converted to hard labels using a global threshold. This threshold has been set to 0.5 as a midpoint of irrelevant class score 0 and relevant class score 1. Thus, all classes with a score higher than the threshold are marked as relevant, while the rest become irrelevant. In this way, the AE-MLFLANN is able to predict multi-label output for any unseen data.

4.5.2 Experimental Analysis

Two phases of experiments have been performed on the proposed model. The first and the more elaborate experimentation has been done on multi-label data, using the proposed

AE-MLFLANN model, since it is the main focus of the work. The second, more concise, set of experiments has been done using a single-label variation of the proposed model, named, AE-SLFLANN, to evaluate its performance on single-label data. As per the knowledge of the authors, this kind of two-layer network does not exist in the literature, hence, its efficiency has been tested on both types of data.

4.5.2.1 AE-MLFLANN for Multi-label data

The proposed AE-MLFLANN has been experimentally verified on five standard multi-label datasets and compared against six state-of-the-art multi-label classifiers. Emotions, Flags, Scene, CAL500 and Yeast datasets have been used for the experiments. Eight standard multi-label performance metrics, namely, average precision (AP), Hamming loss (HL), one error (OE), coverage (Cov), ranking loss (RL), micro F1 (MicF1), macro F1 (MacF1) and subset accuracy (SA) have been used here with 5-fold and 10-fold cross-validation over the above mentioned datasets. These metrics include example-based metrics (HL and SA), ranking-based metrics (AP, OE, Cov and RL), and label-based metrics (MacF1 and MicF1).

Table 4.5 and 4.6 show the 5-fold and 10-fold cross-validation results respectively for AE-MLFLANN on all datasets against the six other algorithms. These include three data transformation methods, BR, CC and ECC, and three problem adaptation techniques, ML-KNN, MLFLANN and BPMLL. From a general overview, the proposed method is seen to perform substantially better than the other algorithms in comparison with all the eight performance metrics. Delving deeper into specific comparisons, AE-MLFLANN is outright better than its single layer version, MLFLANN, for all the datasets and metrics. This indicates that the inclusion of the transformation layer has definitely proven to be fruitful. It is now able to perform multi-label classification more efficiently than before. BP-MLL is a two-layer MLP model, which performs more weight adjustments and computations than AE-MLFLANN. However, AE-MLFLANN is seen to surpass BP-MLL for all the datasets in almost all aspects. This shows that the simple transformations of

Table 4.5: 5-fold CV results

Dataset	Method	AP	HL	OE	Cov	RL	MicF1	MacF1	SA
	BR	0.6190	0.2640	0.4470	0.4331	0.3270	0.5570	0.5650	0.1720
	CC	0.5790	0.2610	0.4200	**0.4163**	0.2940	0.5730	0.5850	0.2210
	ECC	0.6735	0.2409	0.4300	2.7559	0.6695	0.4761	0.4259	0.1517
Emotions	ML-KNN	0.8101	0.1925	0.2581	1.7486	0.1565	0.6692	0.6321	0.2867
	MLFLANN	0.7619	0.2381	0.3423	1.9678	0.1979	0.6188	0.6133	0.2124
	BP-MLL	0.7982	0.2089	0.2969	1.7673	0.1622	0.6913	0.6624	0.2849
	AE-MLFLANN	**0.8217**	**0.1863**	**0.2395**	1.6861	**0.1420**	**0.6925**	**0.6769**	**0.3221**
	BR	0.7295	0.2764	0.2184	4.5837	0.4882	0.7153	0.6206	0.1547
	CC	0.7212	0.2830	0.2289	4.6103	0.4840	0.7050	0.6053	0.1800
	ECC	0.6803	0.3212	0.2340	4.5574	0.6015	0.7217	**0.6706**	0.1310
Flags	ML-KNN	0.8117	0.2924	0.2138	3.7480	0.2128	0.7007	0.5183	0.1497
	MLFLANN	0.7696	0.3051	0.2707	4.0323	0.2739	0.6836	0.6188	0.1286
	BP-MLL	0.8003	0.3247	0.2134	3.9796	0.2338	0.6904	0.5054	0.1256
	AE-MLFLANN	**0.8235**	**0.2656**	**0.1867**	**3.7200**	**0.2025**	**0.7219**	0.5802	**0.1758**
	BR	0.7075	0.1131	0.4483	1.2746	0.5593	0.5785	0.5700	0.4067
	CC	0.7092	0.1126	0.4437	1.2692	0.5546	0.5821	0.5748	0.4117
	ECC	0.6648	0.1240	0.5056	1.4802	0.6440	0.5053	0.4879	0.3249
Scene	ML-KNN	0.8717	0.0839	0.2181	0.4504	0.0731	0.7438	0.7476	0.6386
	MLFLANN	0.8451	0.1014	0.2534	0.5629	0.0942	0.7129	0.7250	0.5596
	BP-MLL	0.7461	0.1608	0.4163	0.8687	0.1562	0.5738	0.5776	0.3947
	AE-MLFLANN	**0.8752**	**0.0761**	**0.2119**	**0.4379**	**0.0707**	**0.7562**	**0.7702**	**0.6398**
	BR	0.2757	0.1414	0.3886	169.3485	0.7879	0.3085	0.0953	0.0000
	CC	0.2769	0.1388	0.3866	169.3507	0.7913	0.3078	0.0838	0.0000
	ECC	0.2742	**0.1381**	0.3667	169.2636	0.7954	0.3040	0.0760	0.0000
CAL500	ML-KNN	0.4904	0.1491	0.1216	129.7710	0.1832	0.3112	0.0867	0.0000
	MLFLANN	0.3391	0.2147	0.4462	166.5194	0.3636	0.3436	0.1311	0.0000
	BP-MLL	0.4891	0.1543	0.1156	**128.5593**	0.1755	0.3669	0.1227	0.0000
	AE-MLFLANN	**0.4996**	0.1467	**0.1070**	130.1255	**0.1623**	**0.3852**	**0.1473**	0.0000
	BR	0.6692	0.1952	0.3136	9.0016	0.4622	0.6375	0.3695	0.1684
	CC	0.6710	0.1933	0.3049	8.9102	0.4576	0.6412	0.3723	0.1763
	ECC	0.6684	**0.1900**	0.2685	8.9814	0.4646	0.6381	0.3523	**0.1779**
Yeast	ML-KNN	0.7436	0.1943	0.2346	6.6215	0.1871	**0.6441**	0.3693	0.1771
	MLFLANN	0.6454	0.3073	0.3583	8.3078	0.2891	0.5286	0.3590	0.0501
	BP-MLL	0.7334	0.2312	0.2747	6.7463	0.1951	0.6322	**0.4172**	0.1220
	AE-MLFLANN	**0.7542**	0.2071	**0.2338**	**6.5337**	**0.1766**	0.6325	0.4034	0.1541

MLP may not be as efficient for multi-label data as the novel combination of functional expansion and feature transformation. ML-KNN is a multi-label adaptation of KNN, which is computationally expensive and is seen to surpass AE-MLFLANN in rare cases. As a problem adaptation technique, AE-MLFLANN establishes its performance quite well as compared to the above three methods. A similar performance situation occurs for the three data transformation methods. BR uses multiple classifiers (one for each class) but it is seldom seen to perform the best for any of the metrics. CC also uses multiple classifiers

Table 4.6: 10-fold CV results

Dataset	Method	AP	HL	OE	Cov	RL	MicF1	MacF1	SA
	BR	0.6911	0.2308	0.3693	2.7446	0.6177	0.5146	0.4713	0.1737
	CC	0.7019	0.2227	0.3609	2.7044	0.5887	0.5447	0.5032	0.1921
	ECC	0.6759	0.2395	0.4298	2.7382	0.6619	0.4838	0.4302	0.1551
Emotions	ML-KNN	0.8016	**0.1906**	0.2732	1.7773	0.1628	0.6703	0.6281	0.3070
	MLFLANN	0.7599	0.2496	0.3172	2.0328	0.2051	0.6088	0.5965	0.1838
	BP-MLL	0.7977	0.2159	0.2866	1.7722	0.1613	**0.6818**	**0.6681**	0.2511
	AE-MLFLANN	**0.8156**	0.1962	**0.2445**	**1.7320**	**0.1491**	0.6747	0.6536	**0.3068**
	BR	0.7466	0.2685	0.2227	4.5250	0.4773	0.7248	0.6457	0.1711
	CC	0.7390	0.2639	0.2327	4.5042	0.4638	0.7247	0.6262	0.2174
	ECC	0.6780	0.3265	0.2490	4.4987	0.6115	0.7241	**0.6567**	0.1261
Flags	ML-KNN	0.8175	0.2764	0.2346	3.7221	0.2043	0.7242	0.5701	0.1145
	MLFLANN	0.7693	0.3136	0.2702	4.0061	0.2763	0.6773	0.6000	0.1342
	BP-MLL	0.7984	0.3225	0.2330	3.9437	0.2280	0.6975	0.5197	0.1258
	AE-MLFLANN	**0.8235**	**0.2656**	**0.1867**	**3.7200**	**0.2025**	**0.7249**	0.5802	**0.1758**
	BR	0.7175	0.1098	0.4375	1.2360	0.5402	0.5958	0.5903	0.4246
	CC	0.7190	0.1096	0.4350	1.2314	0.5370	0.5977	0.5928	0.4275
	ECC	0.6719	0.1228	0.4981	1.4449	0.6329	0.5155	0.4977	0.3328
Scene	ML-KNN	0.8723	0.0893	**0.2135**	0.4587	0.0743	0.7415	0.7448	**0.6335**
	MLFLANN	0.8448	0.1015	0.2538	0.5650	0.0943	0.7132	0.7256	0.5525
	BP-MLL	0.7361	0.1605	0.4261	0.8675	0.1652	0.5534	0.5876	0.4147
	AE-MLFLANN	**0.8728**	**0.0881**	0.2164	**0.4417**	**0.0714**	**0.7506**	**0.7659**	0.6215
	BR	0.2756	0.1426	0.3844	169.2574	0.7892	0.3061	**0.1766**	0.0000
	CC	0.2790	0.1391	0.3844	169.2872	0.7911	0.3083	0.1710	0.0000
	ECC	0.2756	0.1384	0.3705	169.2458	0.7977	0.3016	0.1617	0.0000
CAL500	ML-KNN	0.4910	0.1393	0.1174	**130.5269**	0.1894	0.3148	0.1734	0.0000
	MLFLANN	0.3326	0.2181	0.4916	167.4366	0.3735	**0.3379**	0.1745	0.0000
	BP-MLL	0.4741	0.1633	0.1256	138.5593	0.1955	0.3369	0.1227	0.0000
	AE-MLFLANN	**0.4984**	**0.1373**	**0.1134**	139.8022	**0.1888**	0.3142	**0.1766**	0.0000
	BR	0.6689	0.1962	0.3165	8.9787	0.4613	0.6366	0.3780	0.1746
	CC	0.6708	0.1944	0.3099	8.9183	0.4578	**0.6400**	0.3773	0.1804
	ECC	0.6693	**0.1903**	0.2751	8.9211	0.4621	0.6390	0.3615	**0.1783**
Yeast	ML-KNN	0.7518	0.1951	0.2356	6.3082	0.1795	0.6382	0.3768	**0.1783**
	MLFLANN	0.6503	0.3032	0.3517	8.2581	0.2850	0.5335	0.4040	0.0567
	BP-MLL	0.7326	0.2323	0.2714	6.7200	0.1945	0.6329	**0.4180**	0.1109
	AE-MLFLANN	**0.7545**	0.2032	**0.2341**	**6.5348**	**0.1747**	0.6360	0.3668	0.1485

and performs very close to BR. However, very rarely does it outperform the proposed model. The ensemble technique, ECC, performs better than BR and CC sometimes, but AE-MLFLANN surpasses it in almost all cases.

Table 4.7 depicts the T-test values for average precision of all the six methods against the proposed AE-MLFLANN. For $t_{.90} = 1.533$ with degrees of freedom = 4, AE-MLFLANN outperforms all the other methods. Table 4.8 shows the two-tailed Wilcoxon signed-rank test statistics for Hamming score metric where the proposed method surpasses the others

Table 4.7: T-Test for all methods against AE-MLFLANN

Method	**T-Test Value**
BR	3.8329
CC	4.1265
ML-KNN	5.4046
MLFLANN	2.3138
ECC	3.4028
BP-MLL	2.0090

Table 4.8: Two-tailed Wilcoxon signed-rank test statistics for all the methods vs proposed AE-MLFLANN (based on Hamming Score)

Method	**p-Value**	**Value of sign rank**
BR	0.4375	11
CC	0.3125	12
ECC	0.3125	12
ML-KNN	0.4375	11
MLFLANN	0.0625	15
BP-MLL	0.0625	15

for $\alpha = 0.20, T_{Wilcoxon} = 2$. Overall assessment of the results obtained shows that the proposed AE-MLFLANN has proven to be quite efficient in the domain of multi-label classification.

4.5.2.2 Comparison with the previous proposed works

Along with the state-of-the-art comparisons, the proposed AE-MLFLANN is compared with the MLSAEELM algorithm proposed in the previous chapter (Table 4.9). Both the proposed works handle similar problems of ML data, i.e., large input dimension and complex decision space. This comparative analysis is aimed to identify the superior method that can handle both these issues well. Both the proposed networks utilize an autoencoder framework for dimensionality reduction to handle the feature dimension. However, for the complex decision space problem, MLSAEELM uses stacked MLELMs for improved input to output mapping and AE-MLFLANN transforms the data to increase separability of the data. From the comparison in Table 4.9 for the four datasets it is seen that the performance of both the algorithms is at par with each other. Specifically for the Scene dataset AE-MLFLANN scores high, whereas for the other three data, MLSAEELM

Table 4.9: Comparison of AE-MLFLANN with the previously proposed MLSAEELM algorithm

Dataset	Method	HL	RL	OE	AP	MacF1	MicF1	SA
Scene	MLSAEELM	0.0918	0.0760	0.2225	0.8677	**0.8352**	**0.8355**	0.6126
	AE-MLFLANN	**0.0761**	**0.0707**	**0.2119**	**0.8752**	0.7702	0.7562	**0.6398**
Emotions	MLSAEELM	**0.1852**	0.1580	0.2735	0.8065	**0.7245**	**0.7232**	**0.5695**
	AE-MLFLANN	0.1863	**0.1420**	**0.2395**	**0.8217**	0.6769	0.6925	0.3221
Flags	MLSAEELM	**0.2612**	**0.1895**	**0.1429**	**0.8420**	**0.6221**	0.6014	**0.6758**
	AE-MLFLANN	0.2656	0.2025	0.1867	0.8235	0.5802	**0.7219**	0.1758
Yeast	MLSAEELM	**0.1943**	**0.1648**	**0.2222**	**0.7649**	**0.7444**	**0.7267**	**0.6104**
	AE-MLFLANN	0.2071	0.1766	0.2338	0.7542	0.4034	0.6325	0.1541

is seen to do slightly well. This can be due to the size of the network, MLSAEELM is a deeper network with more number of layers, thus leading to more learning. The AE-MLFLANN, on the other hand is a shallow network which might restrict the learning capacity comparatively. This can be overcome by increasing the transformation layers in future.

4.5.2.3 AE-SLFLANN for single-label data

Looking at the proposed AE-MLFLANN, it is seen to have various scopes of exploration and improvement. Although AE-MLFLANN has been specifically developed for multi-label classification here, and this network does not exist in the single-label domain, its single-label version AE-SLFLANN has been tested as well to analyse its effectiveness.

Table 4.10: Testing Accuracy for single-label data

Datasets	MLP	FLANN	ELM	AE-SLFLANN
Parkinson	0.8215	0.8717	0.8042	**0.8757**
Ionosphere	0.8632	0.9002	0.8776	**0.9031**
PIMA	0.7721	0.7501	0.7383	**0.7618**
Vertebral 2C	0.8323	0.8226	0.8419	**0.8452**

Table 4.10 shows the testing accuracy of AE-SLFLANN on four datasets from the UCI repository. AE-SLFLANN works in a similar way as AE-MLFLANN, only the multi-label output layer at the end is replaced by a single-label one. The final classification

is done by assigning a hard label to the class with the maximum classification score, instead of using a threshold like in multi-label. Here, when compared to a single-label FLANN, the proposed AE-SLFLANN performs somewhat better than it. This happens since the AE transformation used for AE-MLFLANN helps the multi-label data to be more separable, whereas, for AE-SLFLANN non-linearity introduced by the functional expansion was quite good, and the second layer contributes marginally but positively to it. Additionally, AE-SLFLANN performs better than a two-layer MLP and an ELM in all cases. This strengthens the claim of the proposed network that the two consecutive layers performing expansion and transformation lead to an improved representation of the data which eventually leads to improved classification performance.

4.6 Conclusion

This chapter was first aimed at increasing the separability of classes in multi-label data. For this, a compact but efficient functional link NN was considered. It was successfully adopted to classify multi-label data through functional expansion of the original features to a higher dimension which helps to improve separability within the data. Six configurations of the proposed MLFLANN model was then explored, which included three different basis functions and two weight learning techniques. It identified two combinations, namely trigonometric expansion with backpropagation learning and power polynomial with PSO learning, to be quite effective. Due to the faster computational speed of the former combination, it was used in the extended model, AE-MLFLANN. Along with increasing separability, with AE-MLFLANN, additionally the aim was to reduce the feature dimension that was getting increased by the functional expansion. This two-layer network was proposed to further transform the features expanded by MLFLANN using AEs while reducing their dimension. Additionally, a single-label version of the AE-MLFLANN model was also developed, named AE-SLFLANN, and tested on few datasets. All of the proposed models were experimentally tested and they display improved classification per-

formance.

Successfully transforming the data to a higher dimension leading to increased separability and better classification capability makes functional expansion quite suitable for multi-label classification. However, after pursuing this route, the next approach of simplifying the decision space needs to be explored. Additionally, the other problems of multi-label data, such as the class imbalance and label correlations need attention. The proposed models in this chapter and the previous dealt with two issues only. The imbalanced classes degrade the performance of the proposed classifier and need to be handled. Similarly, gaining some additional information from the underlying label correlations should enhance the classification performance. Thus the approach of simplifying decision space, preserving class correlation and handling class imbalance is to be done in the next chapter.

Chapter 5

Binary Tree of Classifiers for Multi-label Data

5.1 Introduction

Moving on to the other drawbacks of multi-label data, the imbalance among classes is seen to be a major issue that exists in almost all ML datasets. The presence of imbalanced classes makes the classifier biased towards the larger classes and is insufficiently trained for the smaller classes. This leads to inevitable misclassifications thus degrading the performance of the ML classifier. Hence, the class imbalance issue is to be handled in this chapter. Additionally, it is seen that multi-label data also has some correlations among classes that are not always taken into consideration. To improve the classification capability of the developed classifier, label correlations are to be incorporated while predicting label-sets.

In this chapter, a tree-based hierarchical multi-label classifier has been proposed, that aims to handle the problem of class imbalance, complex decision space and label correlations from a different perspective. Here, an ML classification model is built that

will utilize the existing class correlations in the data to improve the classification performance. Instead of assuming the classes to be independent and learning one label at a time, it aims to utilize and preserve the label associations as much as possible. This will be done by localizing the data according to the class labels and then using multiple small classifiers for training instead of employing a massive complex classifier. This localization also leads to the simplification of the multi-label decision space. The basic hierarchical model of the proposed classifier is inspired by decision trees, which contains the entire training data at the root node. In the proposed Multi-Label Binary Tree of Classifiers (ML-BTC), this data is split into two discrete chunks every time as the levels in the hierarchy increase, eventually forming a binary tree-like structure. Unlike other models in literature [93, 94], here, specifically a binary tree structure has been adapted instead of an n-ary tree with a variable number of children. It essentially means that at each intermediate node, only one split is necessary; it focuses on the major partition in the data. Subsequent child nodes look for further divisions in the partitioned subspace. This step-wise division of the data simplifies the decision space without having to learn all the class boundaries together. Moreover, contrary to the decision tree approach of exhaustively searching for the "best" split at each node, the proposed model seeks an approximate split that strives to keep label dependencies unchanged. Thus, the proposed model takes the data at a node and approximately splits it into two sets aiming to keep the correlation among classes intact. This consistent correlation is achieved with a novel approach to compute label-set proximity among the data. It forms subsets that contain data instances whose label-sets are similar to each other. This helps in the preservation of label correlations without computing class-wise correlations explicitly.

The motivation to opt for a decision-tree-like approach is to be able to manipulate, combine or split the ML datasets conveniently to help deal with the class imbalance issue [39, 40]. At the intermediate nodes of the tree, similar label-sets are grouped to form two comparatively larger subsets which have increased sample size as compared to the individual label-sets. This makes the class imbalance issue less prominent. However, sometimes

when the data is highly imbalanced, the class-based splitting at these nodes might lead to uneven subsets. This issue is also handled in the proposed technique by utilizing different types of classifiers and parameters for appropriate scenarios. Preliminary classification is performed at the intermediate nodes while the final classification is done at the leaf nodes. Unlike other tree-based models, the proposed tree is not built continuously until each node consists of an individual class or label-set. Since in the worst case, imbalanced classes might end up with a singleton training sample in a leaf node. To avoid unnecessary partitioning of imbalanced classes, the expansion of the tree is restricted with the help of parameters at a much earlier stage. Thus, suitable parameters and classifiers have been used to orchestrate a strategic yet efficient formation of the hierarchical tree of classifiers, sometimes also referred to as the "classifier tree" in this chapter. This name is apt since it is an adaptation of the traditional decision tree approach with the "decision" at most of the nodes being made by some classifier. The overall goal of this work is to build a decision-tree-based model that utilizes the underlying label correlations in the data and simplifies the decision space to perform efficient ML classification while handling the class imbalance issue. The main contributions of this chapter can be highlighted as follows.

- Build a tree of classifiers for multi-label classification that utilizes suitable classifiers at the intermediate and leaf nodes to handle various bottlenecks.

- While building the tree, a novel label-space partitioning technique is applied to the data to implicitly handle the underlying class correlations which otherwise might get ignored.

- Approximate splitting of data is done to achieve faster convergence instead of following the traditional exhaustive best split approach. ML data often have large input and output dimensions which is overwhelming for an exhaustive search. Also, the broad partition simplifies the decision space and allocates more data with a group than the imbalanced classes and label-sets, thus making the problem of complex boundaries and class imbalance less prominent.

- Explicitly handling imbalanced classes that cause uneven splitting of data at the intermediate nodes. The use of appropriate classifiers at the intermediate and leaf nodes based on the data at hand ensures proper attention to the imbalanced data. Also, the building of the tree is based on parameters that facilitate restrictions to prevent its unnecessary branching for smaller imbalanced classes.

The proposed model has been tested on fourteen standard ML datasets, with nine performance metrics, compared with fourteen relevant state-of-the-art techniques and it shows significant improvement with respect to most of the existing methods.

The organisation of the remainder of the chapter is as follows. Section 5.2 contains a detailed description of the proposed work along with the architecture and methodology. Experimental analysis has been included in this section. Finally, Section 5.3 concludes the chapter.

5.2 Proposed Multi-label Binary Tree of Classifiers

In this chapter, a novel binary tree of classifiers for multi-label classification (ML-BTC) has been developed. This method attempts to overcome a few limitations that are faced while handling multi-label data. In the literature, researchers are seen to approach the problem of multi-label classification from various perspectives. Among the existing obstacles, utilizing and preserving the dependencies among classes are quite important. Correlated classes that frequently occur together can be mapped simultaneously to validate their occurrences and reduce misclassification. Many of the existing methods may not explicitly handle this problem. In the proposed method, the class dependencies that exist within the data are being utilized by grouping the data that belong to a similar set of classes. This helps the classifier to identify the related classes and guide an unknown data sample through the ML-BTC to its closest label-set. This grouping and splitting of data occurs without having to learn the overlapping boundaries and leads to a simpler representation of the decision space. Along with this, the problem of class imbalance

that commonly occurs in multi-label data has also been handled in the proposed method. Grouping of related label-sets at the intermediate steps help to increase the sample size in comparison to that of individual label-sets, thus diminishing the magnitude of the imbalance problem. However, the presence of highly imbalanced classes can still cause uneven partitioning of data and unnecessary branching of the tree which have been handled effectively through the appropriate use of certain classifiers and restricting the growth of the tree with the help of some parameters.

5.2.1 Model Description

Details of the proposed approach have been discussed in the following sections.

5.2.1.1 Building of the tree

Starting from the root node, the tree is formed by successively splitting the available data at every internal node. Initially, the root node contains all the training instances N_{tr}, and the initial split at the root node separates the data into two subsets, say, N_{tr1} and N_{tr2}, where $N_{tr} = N_{tr1} \bigcup N_{tr2}$ and $N_{tr1} \bigcap N_{tr2} = \phi$. This splitting of the data leads to the creation of two child nodes that contain one subset of N_{tr} each. The data present at any node is used as the training set of that particular node. In this way, the data at each internal node is iteratively split into smaller partitions and the tree eventually grows until the data cannot be split any further. The division at each internal node represents a prominent split in that data, which simplifies the boundaries while keeping the correlation between labels intact. A novel proximity-based label-space partitioning scheme has been employed to preserve the label correlations. Once an approximate split has been determined at the current node, a classifier is trained to learn the decision boundary between the two subsets of data. In the subsequent steps, data chunks at the nodes are further split and specific classifiers are trained to learn the broad classification. The choice of the classifier is based on the type of partition that is made at an internal node. Researchers who developed similar tree-based multi-label classifiers [93, 94] in the

past, have preferred to use a single type of classifier for the entire data. However, while dealing with ML data, the class imbalance problem leads to irregular partitioning in the intermediate steps. This, in turn, makes larger classifiers quite impractical to be trained on a smaller quantity of data. Keeping this in mind, different classifiers are incorporated to suit the data at hand and improve effectiveness.

Every decision at any internal node is made depending on two parameters: ML entropy and sample cardinality – which determine the status of the current data chunk. If the parameters suffice, the data at a node is split further, otherwise, the node is converted to a leaf node. It is at these leaf nodes that the final multi-label class prediction is performed. The data that is at a leaf node might belong to a single set of labels or a varied set of labels. If there are different combinations of labels among the data in a leaf node, a suitable multi-label classifier is used in that node. If there is a unique label-set in a leaf node, no further classifier is needed. The intuition behind developing this model is that multi-label data is inherently imbalanced. It becomes expensive and increases the number of nodes in a decision tree if the tree grows until each leaf node contains a unique label-set. In this case, instances whose label-sets are quite similar to each other may be found residing in a leaf node if the data size is not large enough to be partitioned. Hence, a multi-label classifier, that can be trained on smaller data, at that leaf node performs efficient classification instead of the repetitive splitting of the data. The actions associated with the building of the classifier tree are discussed below in detail.

Splitting strategy Most decision tree-based algorithms split the data depending on a single feature at a time [11] or the entire feature space [77, 94]. Although this seems efficient for single-label classification, for multi-label classification with the increase in the input and output dimensions, the problem becomes more complex to handle. In the proposed method, instead of using the feature space for partitioning the data, the label information has been utilized in a novel way. For example, while dealing with

multi-label images, a broad classification would mean images with one or more labels like "sea", "sand", "beach", etc. would belong together whereas images with labels like "grass", "mountains", etc. should be a part of another large group. To enable this type of broad classification, grouping of the data has been proposed in the label space instead of the feature space. This means that data whose class labels are similar will belong to one group and the classifier will train itself based on this partition.

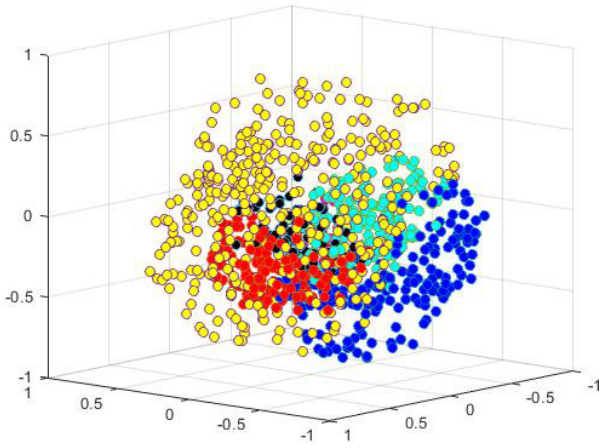


Figure 5.1: Representation of data in the feature space

Figure 5.1 shows an example of a synthetic multi-label dataset with three features and three classes. Each of the label-sets is depicted in different colours to visualize the complexity of the data. For real multi-label data, with higher input and output dimensions, the data is far more complex. To deal with this data more simply, the proposed method visualizes the data in the label space and obtain a split that preserves label correlations approximately.

Figure 5.2 represents the same dataset in the 3-dimensional output space, which appears to be far simpler and organized. Thus, the data to be split at any node is partitioned into two groups in the output dimension. Data with the same label-sets are represented with the same point in the label space; hence, data having the same label-set always belong to the same group. This simplifies the classes and reduces the scope of misrepresentation of data. This approach ensures that data of similar class labels will be closer in the decision space and thus make the task of partitioning simpler. Figure 5.3 depicts the splitting

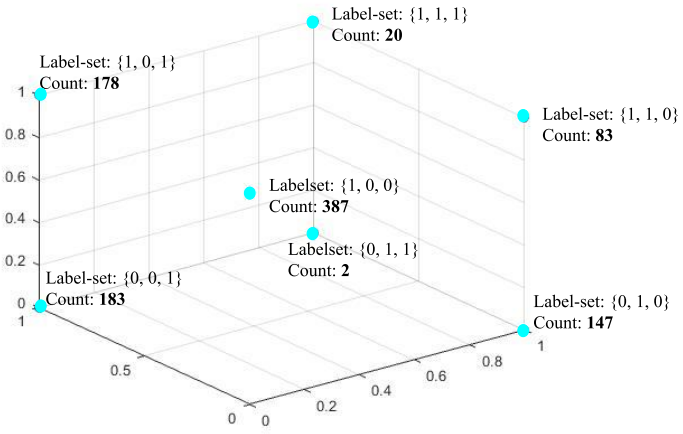
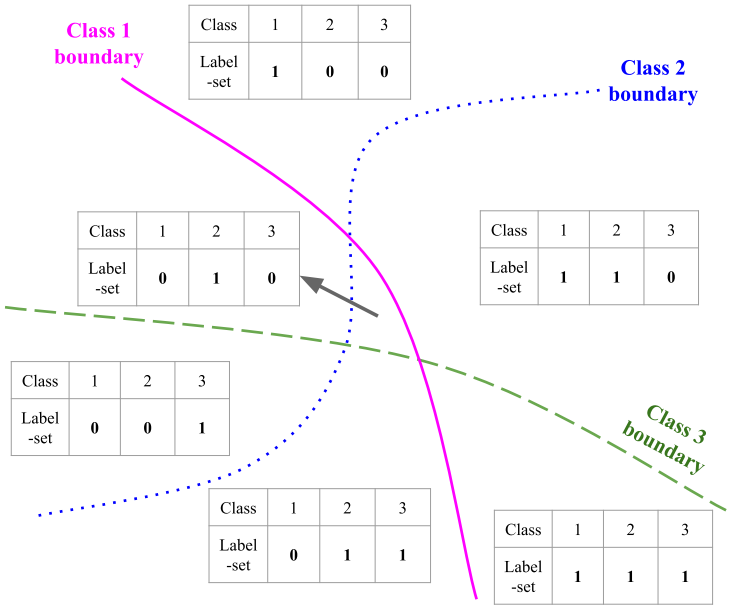


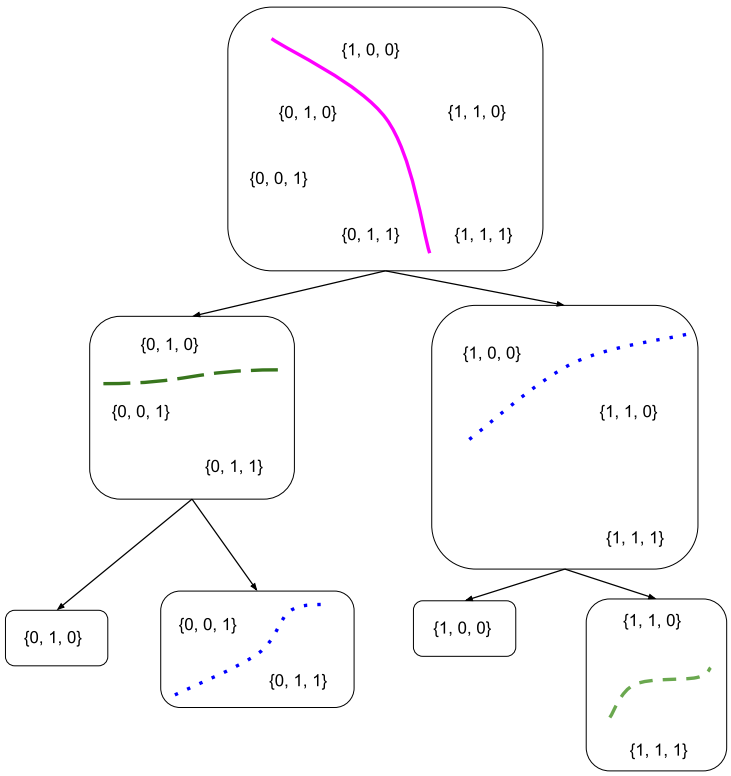
Figure 5.2: Representation of data in the label space

strategy developed here. The decision space and the original class boundaries for the three-class label-sets are shown in Figure 5.3a. Figure 5.3b shows that the entire data (represented by the label-sets) available at a node is split into two subsets based on the proximity of their label vectors. This proximity among any two data points is measured by computing the Hamming distance among their respective label-sets. Hamming distance is one of the easiest techniques to compute the difference between two label-sets which are same-length vectors of 1s and 0s. The difference of bit values indicates dissimilarity. Lower Hamming distance between two points indicates greater similarity among the label-sets. In the proposed ML-BTC, to find co-occurring classes, the label-sets with low Hamming distance are considered. Depending on these distances, the entire data is consecutively split into two groups at each intermediate node. The consecutive splitting of data at every node that leads to the formation of a tree-like structure calls for a divisive hierarchical clustering algorithm that can partition the available data into two chunks at every step. For this purpose, a hierarchical clustering algorithm, namely, bisecting k-medoids, is employed step-wise on the proposed tree structure.

Bisecting k-Medoids This algorithm is a variation of the existing hierarchical clustering technique, bisecting k-means. Bisecting k-medoids algorithm has been used in the proposed method to split the data sequentially at each internal node. This results in the



(a) Original multi-label class boundaries



(b) Data split at each node

Figure 5.3: Splitting strategy of ML-BTC

formation of the binary tree-like structure. The bisecting k-medoids is as follows:

Step 1 Begin with the entire data in one cluster, say, Cl

Step 2 Use the well-known k-medoids clustering technique with k=2 on the current data to generate 2 distinct clusters, say Cl_1 and Cl_2.

Step 3 Select one cluster at a time and repeat *Step 2*.

Step 4 Stop when the desired number of clusters has been reached. Each individual cluster resides in a particular leaf node.

The proposed multi-label binary tree of classifiers (ML-BTC) model incorporates the bisecting k-medoids algorithm within the binary tree structure. Thus, each intermediate node of the ML-BTC corresponds to the clustering tasks in the bisecting k-medoids algorithm itself. The clustering algorithm has been employed in the label space of the data to generate appropriate clusters among similar label-sets while preserving correlation among them. Also, Hamming distance has been used as a similarity measure among label-sets for the bisecting k-medoids in the proposed algorithm.

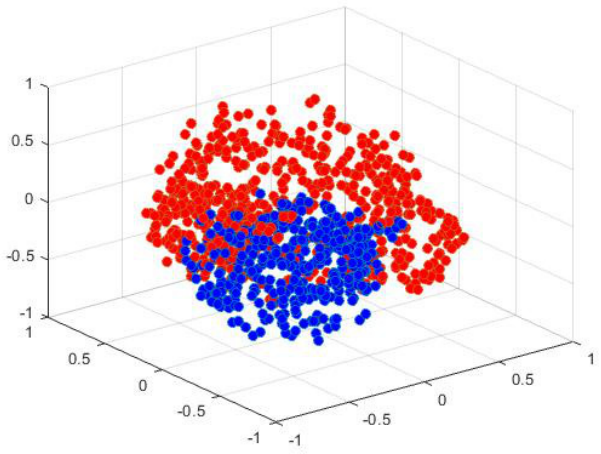


Figure 5.4: Approximate partition of the data as per the output space

Figure 5.4 represents the synthetic dataset after the application of bisecting k-medoids with the approximate split generated in the output space of the data mapped to the input space. After the splitting is done, data with similar label-sets are seen to belong to the

same subset. The intuition behind this approach is that, while dealing with multi-label data instances may be encountered whose features are similar but the label-sets are very different. Similarly, instances having similar label-sets may not have identical feature values. Hence, basing the decision at a node on feature values, may not be quite appropriate while dealing with multi-label data. Using the label information in this way attempts to preserve and utilize the underlying correlation between classes in a particular dataset. If a set of labels were inter-related, i.e., they occur together frequently, they remain intact in the same subset. On the other hand, using a method to split the data based on each label or feature loses any label correlation information that was present in the data. The proposed method aims to utilize this label correlation information and assign the closely related labels to the relevant sample in turn improving the final classification performance. Here, the original overlapping class boundaries are not specifically used. Instead the label-sets are utilized which have discrete boundaries. A major approximate split is created from the label-sets which is discrete and non-overlapping. This making the decision space much simpler. Additionally, this approximate splitting strategy helps to handle the class imbalance issue to some extent. When two broad groups are created, it contains a larger number of data points than the individual imbalanced classes or label-sets. This helps to make the imbalance problem less prominent at this stage. However, if still, the data splitting is uneven due to imbalanced classes, it is handled separately.

In this novel splitting strategy, the correlation among labels is not explicitly computed, but it serves a similar purpose. Identical or similar label-sets end up in the same group, whereas dissimilar label-sets will have a higher Hamming distance, thus, they are most likely to be well separated. In the case of equidistant label-sets, depending on the parameters, initially, one approximate split is made in the current node. Later, in the children nodes, further splits are made if necessary, eventually leading to few equidistant label-sets in the same group or as individual leaf nodes. However, once a split has been determined, it is the job of the binary classifier to train itself to be able to classify future samples in the same way.

Classifiers In the proposed model, different classifiers have been used to handle specific scenarios. Unlike existing tree-based multi-label classification models, the proposed hierarchical ML-BTC uses different classifiers depending on the data at a node that it needs to handle. The intuition behind using different classifiers is that each splitting of data at an intermediate node may not result in similar partitions. Existing techniques [93, 94] use a single classifier in the entire model. Since multi-label data inherently is class imbalanced, few label-sets have insufficient data and splitting up of this set may not result in equal-sized partitions. If the split consists of a tiny and a large subset, using a generic classifier like an SVM might not prove to be quite useful. The uneven partition might cause the selection of insufficient support vectors thus leading to biased results. In this scenario, a simpler classifier is opted that can work with irregular as well as lesser quantities of data. A classifier like k-nearest neighbour (k-NN) generates a piece-wise decision boundary based on a small number of data points. Thus, the uneven sizes of the partitions do not affect the final classification boundary [99]. Thus, if the data at hand is split into two uneven partitions, a k-NN classifier suffices. On the other hand, if the split results into two large-sized groups, an SVM classifier is used. SVM has been chosen since it is a strong classifier that can efficiently perform classification for ML data [93, 94].

Apart from the intermediate nodes, the proposed model attempts to employ classifiers in some leaf nodes when needed. Due to the class imbalance problem, the subsequent splitting of data might be quite irregular thus resulting in a node containing a smaller number of data, but from various label-sets. At such a node, where the data cannot be split any further, yet its entropy is on the higher side, a multi-label classifier is employed at that node. This ensures that the classifier tree need not be expanded further, however, the data at that leaf node gets correctly classified. Since this job is of a simpler classifier that can work well with a smaller number of data, a multi-label k-NN classifier is used. In a similar scenario with a larger amount of data that cannot be split further,

a shallow multi-layer perceptron (MLP) is used. In this way, to determine which classifier is suitable at the current node, few parameters have been used in the proposed model.

Parameters The proposed binary tree of classifiers model for multi-label classification proceeds to split the available data at an internal node into two subsets. The data available at each node may not be sufficient or suitable for further splitting. The decision to be taken at any node is thus quite crucial and is based on the following two parameters.

a) **Multi-label entropy**: Entropy determines the degree of variation in the label-sets of the data available at a node. If entropy = 0, the chunk of data in the node belongs to a single label-set, i.e., all the instances at that node have the same set of labels. With the increase in entropy, it can be said that the data available at a node has increased variation. In other words, there are multiple instances whose label-sets are quite different. Entropy is defined as

$$H = -\sum_{i}(p_i \log p_i) \tag{5.1}$$

where p_i is the probability of an instance belonging to the i^{th} class. Multi-label entropy can be described [11] as

$$H = -\sum_{i}(p_i \log p_i + q_i \log q_i) \tag{5.2}$$

where $q_i = 1 - p_i$. At every child node, the entropy should be lower than its parent node. An entropy threshold, $H_{threshold}$, is provided in the model, which is used to determine the decision to be taken at any node. At the point where entropy cannot be reduced further, the tree stops growing.

b) **Sample Cardinality**: This parameter, named SC, is used along with entropy to make decisions at any node. It represents the number of data points present at the

current node. This is essential since decisions cannot be made based only on the variation in the data, the size of the available data also needs to be assessed. A cardinality threshold, $SC_{threshold}$, determines if the amount of data at any node is sufficient to be partitioned further or the process should be stopped to form a leaf node. The importance of this parameter lies in the simple notion that there is no need to unnecessarily keep on expanding the tree even if the data size becomes trivial.

A combination of the above two parameters helps to make a decision at any given node in the training phase. Splitting of the data at every node follows a simple strategy; if there is sufficient data at the node and the variation is high, the data needs to be split and the tree can be expanded. Sometimes, even if the entropy is high, the amount of data at a node may not be enough for splitting, i.e., since the sample cardinality is already low, splitting it into further small groups serves no purpose and simply increases the number of nodes for traversal in the testing phase. The aim of building this tree is not to have unique label-sets at each leaf node, but to keep building the tree as long as it is necessary to perform efficient multi-label classification. If a situation arises where every label-set has one training sample each, it is pointless to keep on splitting to end up with leaf nodes having singleton label-sets. Instead, the tree would first recognize prominent splits and employ a smaller classifier at a much earlier stage to do the same job and reduce unnecessary branching out. Thus, the actions within the tree, at any internal node have high importance.

5.2.1.2 Methodology

The main purpose of this contribution is to construct a tree-like structure step-wise capable of performing multi-label classification. When the test data is fed to the root node of the trained tree, based on some decisions at the intermediate nodes, the data traverses smoothly from the root node to a leaf node which eventually ends up classifying the data sample to a specific label-set.

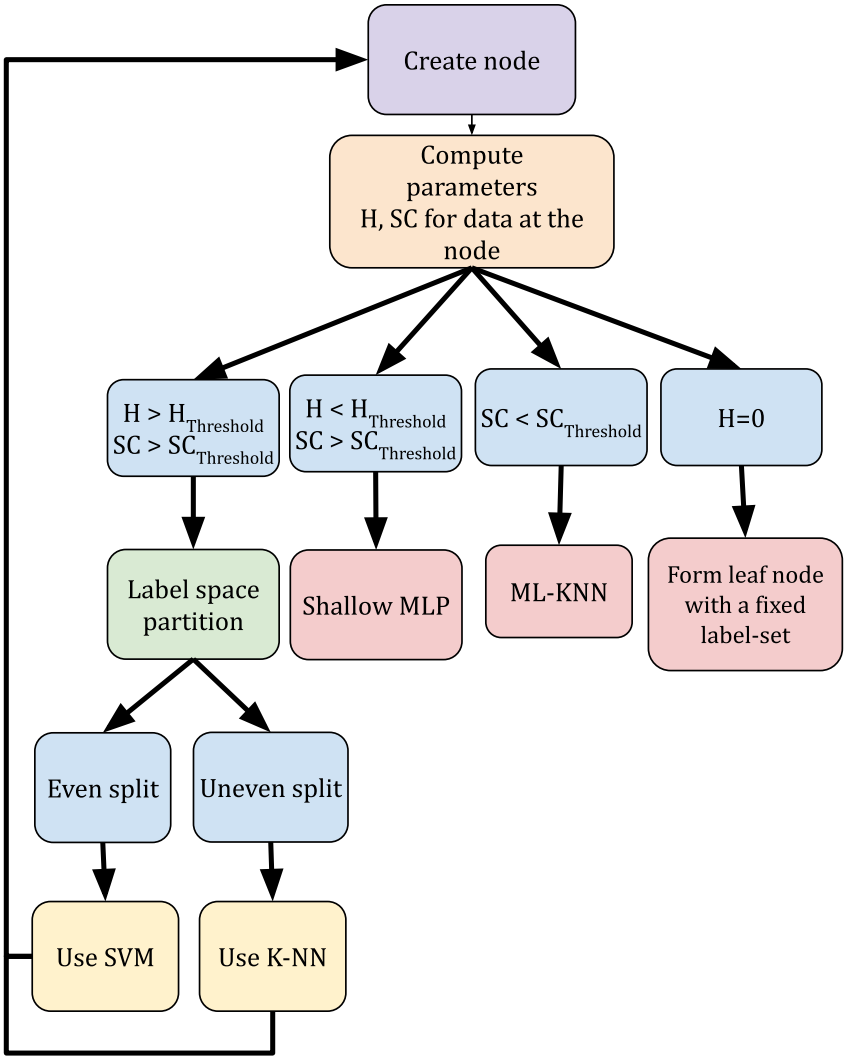


Figure 5.5: Flow diagram for the creation of ML-BTC

Training Phase In the training phase, the hierarchical binary tree of classifiers is created in a step-wise fashion (Figure 5.5). Initially, the empty root node contains the entire training dataset. The action-determining parameters (namely entropy and cardinality) are computed at the current node ($node_{Curr}$). Depending on these parameters, H and SC, the next action is selected. The parameters (H & SC) are evaluated based on threshold values, $H_{threshold}$ and $SC_{threshold}$. There are a few conditions that may arise depending on the parameters:

- $H > H_{threshold}$ and $SC > SC_{threshold}$: This indicates the presence of a large number of diverse data which should be split in order to expand the tree. Thus, a label

space partition is performed using bisecting k-medoids at $node_{Curr}$ thereby creating a strategic split in the data.

  - If this splitting of data is uneven leading to one of the groups being too small, a k-NN classifier is used for this partition.

  - If the data has been split into even proportions, an SVM is used to learn a decision boundary between the two parts.

- $SC < SC_{threshold}$: If a node has a lesser amount of data and splitting it is not possible, the node is converted to a leaf node and an MLKNN classifier is used. This ensures that any data that land up in this node while testing gets properly classified. Even though the amount of training data might be small, the label-sets are not completely ignored. This ensures the handling of imbalanced classes that have a fewer number of samples.

- $H < H_{threshold}$ and $SC > SC_{threshold}$: If there is enough data with lower variation, it indicates a group of closely related label-sets. Splitting it further results in the separation of correlated label-sets. The low entropy is further reduced as a result of splitting, eventually leading to individual leaf nodes for each label-set. This elongated procedure is stopped early by employing a shallow MLP on the sufficient data available, to learn different classes at that stage.

- $H = 0$, this indicates that the dataset present at the current node belongs to the same label-set, hence no further action is required. This node also becomes a leaf node with a fixed label-set.

The depth of the tree is not fixed beforehand. It grows till there is sufficient data to be split at any intermediate node. If the conditions determine the data unfit to be split any further, that node is converted to a leaf node. In this way, the entire tree grows to some extent depending on the two parameters.

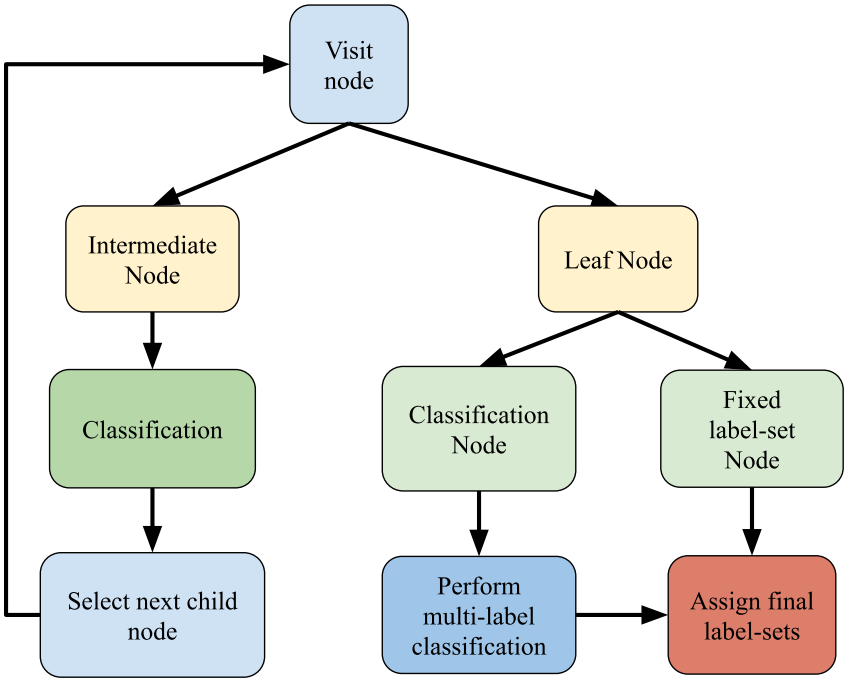


Figure 5.6: Flow diagram for the testing phase of ML-BTC

Testing Phase After the hierarchical tree structure has been built in the training phase, the validation/testing phase begins. There are a few varied scenarios that may arise for the test data (Figure 5.6). The testing process begins with an unknown sample at the root node. The binary classifier at the root classifies the test pattern to either the left child or the right child of the current node. In this way, the data moves downwards through the binary tree structure getting classified by the designated classifier at every intermediate node. Finally, at some point, the test pattern reaches a leaf node which performs the final multi-label classification. There are 3 scenarios that a test pattern may face at a leaf node:

- Fixed label-set: The leaf node has a fixed label-set and the test pattern is assigned that particular set of labels without any further investigation.
- MLP: The test pattern is fed to the trained network to output its final label-set.
- ML-KNN: The ML-KNN classifier finalizes the label-set to be assigned to the test pattern that reaches this leaf node.

In this way, any unknown sample gets gradually classified as it traverses through the ML-BTC and receives its final label-set at the leaf nodes. In the case of test data that belongs to a label-set unknown to the classifier tree, it moves step-wise through the intermediate nodes to reach a leaf node that assigns a label-set nearest to its actual, thus minimizing the classification error. However, in the case of a stark domain shift, i.e., the addition of data from new domains/classes, the classifier tree might need to be retrained and restructured with the new information. This is especially because, in the multi-label scenario, the addition of one new class indicates the possibility of twice the number of existing label-sets. If there were C number of classes with the possibility of 2^C label-sets, adding one more class would change the number of possible label-sets to 2.2^C or 2^{C+1}. Hence, this new class definitely needs to be included in the classifier tree.

5.2.1.3 ML-BTC vs few tree-based multi-label classifiers

Comparing the existing tree-based multi-label classifiers with the proposed ML-BTC, the methodologies are seen to differ at a few elements. Firstly, the concept of label-space partition of the data at each intermediate node is unique which implicitly preserves the label correlations and simplifies the decision space. Due to this practical approach, the proposed method employs a maximum of one classifier at every node (both, intermediate and leaf nodes) and results in the generation of approximate splits. This is a major improvement when compared to existing tree-based techniques, which looks for the best split, i.e., best feature or best class partition, by exhaustively searching the entire space. For example, ML-TREE [94] and ML-FOREST [93] use one-vs-all SVMs for every class at each intermediate node resulting in the use of numerous classifiers. G3P-kEMLC [50] uses a pool of label-powerset classifiers, each predicting a set of k labels. A tree structure is built utilizing context-free grammar and an evolutionary algorithm. Another multi-label decision tree model, MLC4.5 [11] finds the best attribute that optimizes information gain. Whereas, the HOMER algorithm [77] creates distinct subsets of classes at every level of the tree and the final leaf nodes contain one class each. For a dataset with a huge

number of classes, this would lead to the formation of a bulky tree with at least that many branches.

Unlike these methods, the proposed ML-BTC prefers to reduce the burden of extensively searching for the best, when it can produce similar results with a good approximation. Unlike methods like ML-TREE, ML-FOREST, HOMER which use one classifier per class, ML-BTC chooses to handle one of the drawbacks of multi-label data, i.e., the possibility of a large number of classes and a larger number of unique label-sets (sometimes in the order of hundreds). In such a case, employing even a small classifier for every class/label-set would increase the size of the overall system. Another novelty in ML-BTC is catering to the class imbalance problem which may lead to uneven class partitions while building the tree. This mainly handled by using appropriate classifiers at different nodes of the tree depending on the amount of available data. Existing tree-based ML classifiers [93, 94] use a single type of classifier for the entire model, say SVM, which may not be customisable for the data at hand. In some cases, even the ML-Forest may not perform better than the tree of classifiers model in all cases since it does not consider the imbalanced classes. Also, since all the label-sets are not equally well-represented, the ML-BTC tree is not expanded till all the leaf nodes correspond to a unique label-set. Instead, the splitting is strategically stopped at an earlier stage to reduce the unnecessary expansion and complexity of the system. Also, the class correlations are incorporated into the ML-BTC algorithm which might otherwise get overlooked, to assemble the data that have similar label-sets. Keeping these in mind, the proposed technique has drawn inspiration from the existing algorithms and has aimed to handle the deficiencies related to the problem of multi-label classification.

5.2.2 Experimental Analysis

Various experiments have been performed on the proposed hierarchical model and have been compared with fourteen state-of-the-art algorithms in the domain of multi-label clas-

sification. The details of the experiments performed have been discussed in the following sections.

5.2.2.1 Setup

Fourteen ML datasets have been used for the experimental analysis of the proposed algorithm, namely, Emotions, Flags, CAL500, CHD49, Scene, Yeast, Enron, Image, Water Quality, Corel, Delicious, Bibtex, EUR-Lex, and Yelp datasets. The proposed work has been executed with 6 runs of 5-fold cross-validation on the fourteen datasets for nine performance measures [28], namely, Hamming loss (HL), ranking loss (RL), one error (OE), subset accuracy (SA), F-measure (FM), macro F1 (MacF1), micro F1 (MicF1), accuracy (Acc), G-mean (GM). Among these, HL, SA, FM, Acc and GM are example-based metrics evaluated instance-wise. They need the crisp output label-set to compute correctness. FM and GM metrics are used to assess algorithms based on the class imbalance problem. RL and OE are ranking-based metrics which only require a ranking of classes and not the actual labels for assessment. Finally, MacF1 and MicF1 are two label-based metrics that compute class-wise performance instead of sample-wise. Results for fourteen well-known multi-label classification techniques have been included in the comparative analysis. Out of these fourteen, four are tree-based algorithms, namely, ML-TREE (2015) [94], ML-FOREST (2016) [93], MLTL-HOMER (2020) [55] and G3P-kEMLC (2020) [50] which have been used for comparing the proposed tree-based ML-BTC. The MLTL-HOMER technique also handles class imbalance problem of ML data, making it apt for comparison. Next four methods are binary or ML ensemble classifiers, CC (2011) [62], ECC (2011) [62], LSF-CC (2020) [89] and RAKEL (2010) [78] which use multiple base classifiers (specifically SVMs) which seem fit for comparison with ML-BTC. Two methods utilize label correlations, LIFT (2014) [102] and ML-LOC (2012) [33]. One is a data transformation technique, BR (2018) [101] which also uses SVM as its base classifier, and the rest are well-known problem adaptation based ML classifiers, namely, ML-KNN (2007) [104], SC-RankSVM (2014) [83] and MLLEM (2016) [41]. Since ML-KNN and

SVM are a part of the proposed method, the focus is to compare with existing works based on these algorithms. Like ML-BTC, MLLEM and RAKEL also aim to preserve label correlations, hence, has been considered for comparison.

5.2.2.2 Analysis of Results

To analyse the performance of the proposed algorithm, six runs of five-fold cross-validation results for all the fifteen methods on fourteen datasets with nine performance measures have been shown in Table 5.1 to Table 5.14 for the small (Emotions, CAL500, Flags, CHD49), medium (Scene, Yeast, Enron, Image, Water quality) and large (Delicious, Corel, Bibtex, EUR-Lex, Yelp) datasets respectively. The values of the threshold have been fixed for all datasets, $H_{threshold} = \frac{H_{initial}}{5}$ and $SC_{threshold} = \frac{N_{tr}}{5}$, where, $H_{initial}$ is the ML entropy of the data at the root node and N_{tr} is the training sample size.

Table 5.1: Results for Emotions (small)

	HL	**RL**	**OE**	**SA**	**MacF1**	**MicF1**	**FM**	**Acc**	**GM**
ML-TREE	0.3083	0.3323	0.5189	0.2428	0.3322	0.3479	0.4958	0.2763	0.4392
ML-FOREST	0.2755	0.3216	0.3129	0.3532	0.5011	0.4981	0.5124	0.3761	0.4892
MLTL-HOMER	0.2208	0.3425	0.3033	0.2834	0.6181	0.6598	0.6506	0.5217	0.6228
CC	0.2206	0.3852	0.2956	0.2464	0.5616	0.5964	0.5346	0.462	0.5072
ECC	0.2141	0.4029	0.2713	0.2651	0.5618	0.6092	0.5734	0.4819	0.5198
LSF-CC	0.2130	0.3542	0.2721	0.4176	0.6211	0.6398	0.5982	0.4982	0.5821
SC-RankSVM	0.2345	0.2181	0.2873	0.1944	0.4291	0.4342	0.4184	0.3125	0.4019
MLLEM	0.2712	0.2752	0.3679	0.1839	0.4319	0.4421	0.4377	0.3736	0.4229
MLKNN	0.2618	0.3009	0.3355	0.1816	0.4105	0.5257	0.4804	0.4017	0.4382
BR	0.2199	0.3421	0.2715	0.2867	0.6303	0.6579	0.6289	0.5438	0.5967
RAKEL	0.2369	0.3645	0.3052	0.2548	0.5686	0.6078	0.562	0.4837	0.2321
LIFT	0.2161	0.3776	**0.2443**	0.6034	0.6126	0.6331	0.6622	0.5099	0.6869
ML-LOC	0.2743	**0.1895**	0.3285	0.6241	0.6011	0.6180	0.6574	0.5264	0.1170
G3P-kEMLC	0.2291	0.3081	0.2902	0.3093	0.6261	0.6347	0.5831	0.4999	0.2566
ML-BTC	**0.2126**	0.2164	0.2639	**0.6384**	**0.6352**	**0.6601**	**0.6704**	**0.5518**	**0.7018**

At the first glance, the proposed ML-BTC is seen to perform better than the other fourteen algorithms for all fourteen datasets in most of the cases. specifically the example-based and the label-based metrics. Only for the ranking-based metrics, RL and OE, ML-BTC does not score the highest in most cases but lies very close to the best performing algorithm. Among all the performance metrics used for comparison, one, in particular,

Table 5.2: Results for CAL500 (small)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.1424	0.3545	0.1508	0.0000	0.0465	0.0614	0.2181	0.0333	0.4506
ML-FOREST	0.1891	0.3412	0.1322	0.0639	0.1271	**0.4305**	0.2412	0.0521	0.4761
MLTL-HOMER	0.2166	0.5084	0.6832	0.0000	0.0641	0.3306	0.3255	0.2157	0.3281
CC	0.1581	0.3783	0.3721	0.0000	0.0638	0.329	0.3321	0.2032	0.4391
ECC	0.1583	0.5697	0.3256	0.0000	0.0821	0.192	0.3308	0.2033	0.4253
LSF-CC	0.1411	0.3872	0.3239	0.0000	0.0901	0.2011	0.3421	0.2098	0.4112
SC-RankSVM	0.1552	0.3619	0.2901	0.0000	0.2259	0.2879	0.3282	0.1023	0.3901
MLLEM	0.6253	0.5964	0.5583	0.0000	0.0675	0.3371	0.2862	0.1819	0.4182
MLKNN	0.1523	0.5796	**0.1245**	0.0000	0.0823	0.3092	0.3373	0.2052	0.4711
BR	0.1396	0.4995	0.3693	0.0000	0.0595	0.324	0.3483	0.2161	0.3901
RAKEL	**0.1372**	0.3549	0.2006	0.0000	0.0381	0.0662	0.3261	0.1991	0.2312
LIFT	0.1453	0.4706	0.1982	0.1997	0.1026	0.3248	0.3618	0.2022	0.4051
ML-LOC	0.1443	0.3509	0.2114	0.1988	0.0831	0.3043	0.3631	0.1861	0.4264
G3P-kEMLC	0.1992	0.3562	0.4236	0.0000	0.0982	0.1082	0.3372	0.2018	0.2516
ML-BTC	0.1931	**0.3445**	0.4195	**0.2052**	**0.1343**	0.2458	**0.3639**	**0.2241**	**0.5089**

Table 5.3: Results for Flags (small)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.4542	0.2524	0.2564	0.3165	0.3459	0.5187	0.6636	0.2043	0.5208
ML-FOREST	0.4051	**0.2415**	0.2411	0.3241	0.3928	0.5721	0.6518	0.2341	0.5312
MLTL-HOMER	0.3751	0.6164	0.2498	0.1341	0.6031	0.6136	0.6517	0.5035	0.4632
CC	0.3532	0.5181	0.2363	0.0927	0.4834	**0.6909**	0.6649	0.5409	0.4372
ECC	0.3683	0.2997	0.2414	0.1093	0.1638	0.6291	0.3308	0.2025	0.4891
LSF-CC	0.3527	0.3512	0.2301	0.1287	0.3213	0.6122	0.4322	0.3544	0.4992
SC-RankSVM	0.5542	0.4482	0.2259	0.1011	0.4882	0.5401	0.5199	0.3732	0.4561
MLLEM	0.4285	0.6104	0.3133	0.1125	0.4095	0.4448	0.4461	0.3211	0.3893
MLKNN	0.3592	0.6562	0.2263	0.0912	0.4795	0.6763	0.6619	0.5182	0.5981
BR	0.3529	0.5769	0.2185	0.1339	0.5051	0.6318	0.6017	0.5061	0.6015
RAKEL	**0.3645**	0.4632	0.2281	0.1236	0.4551	0.6739	0.6513	0.5104	0.5801
LIFT	0.3533	0.5551	0.2256	0.5291	0.4897	0.6513	0.6543	0.5068	0.6031
ML-LOC	0.3497	0.3709	0.3846	0.5221	0.5011	0.6457	0.6231	0.4965	0.6092
G3P-kEMLC	0.3731	0.3982	0.2315	0.2091	0.5009	0.6019	0.6511	0.4572	0.5982
ML-BTC	**0.3438**	0.3135	**0.2158**	**0.5922**	**0.5085**	0.6114	**0.6697**	**0.5194**	**0.6115**

Table 5.4: Results for CHD49 (small)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.3562	0.2516	0.3831	0.5821	0.3627	0.6157	0.5941	0.4655	0.6156
ML-FOREST	0.3501	0.2981	0.3998	0.5901	0.3782	0.6102	0.5899	0.4561	0.5782
MLTL-HOMER	0.3245	0.3414	0.2924	0.1441	0.4214	0.6276	0.6042	0.4893	0.5688
CC	0.3325	0.4711	**0.0831**	0.1712	0.4102	0.6309	0.6117	0.4713	0.5794
ECC	0.3322	0.3928	0.0992	0.1712	0.3990	0.6333	0.6087	0.4773	0.5893
LSF-CC	0.3521	0.3092	0.3981	0.2091	0.4281	0.6009	0.5983	0.4664	0.5899
SC-RankSVM	0.4182	0.3708	0.4476	0.0306	0.4186	0.4912	0.4618	0.3498	0.5783
MLLEM	0.4301	0.2851	0.4152	0.0054	0.4092	0.5637	0.5763	0.3981	0.5711
MLKNN	0.3421	**0.2308**	0.2274	0.1459	0.4044	0.6391	0.6107	0.4819	0.6002
BR	0.3262	0.2563	0.2725	0.1748	0.4205	0.6288	0.6138	0.4803	0.6032
RAKEL	0.3484	0.3739	0.1606	0.1525	0.4161	0.6336	0.6058	0.4847	0.6109
LIFT	0.3303	0.5421	0.1412	0.5834	0.4066	0.6259	0.6006	0.4814	0.6152
ML-LOC	0.3438	0.2451	0.3194	0.5661	0.3152	0.5918	0.6102	0.4339	0.6167
G3P-kEMLC	0.3492	0.3569	0.3078	0.1892	0.4075	0.4933	0.5714	0.4557	0.6005
ML-BTC	**0.3244**	0.2659	0.3463	**0.6054**	**0.4338**	**0.6398**	**0.6139**	**0.4901**	**0.6288**

Table 5.5: Results for Scene (medium)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.1982	0.1377	0.2871	0.6618	0.5681	0.5784	0.7071	0.6067	0.6573
ML-FOREST	0.1872	0.1102	0.2701	0.6682	0.5743	0.5889	0.6632	0.6033	0.6392
MLTL-HOMER	0.1378	0.3036	0.3012	0.5933	0.6893	0.6797	0.6873	0.6634	0.6473
CC	0.1372	0.3537	0.3575	0.5526	0.6974	0.6898	0.6468	0.6231	0.6281
ECC	0.1452	0.3555	0.2712	0.5929	0.7029	0.7015	0.6941	0.6182	0.6709
LSF-CC	0.1324	0.2743	0.2572	0.5821	0.6352	0.6401	0.6478	0.6092	0.6382
SC-RankSVM	0.5629	0.3994	0.2846	0.1099	0.4107	0.4095	0.4186	0.3025	0.4089
MLLEM	0.1384	0.3535	0.2918	0.6143	0.7099	0.6762	0.6901	0.6102	0.6875
MLKNN	0.1452	0.2191	0.2411	0.6512	0.7217	0.7122	0.7219	0.6981	0.6901
BR	0.1343	0.3796	0.2671	0.6436	0.7161	0.7091	0.7133	0.6958	0.6877
RAKEL	0.1191	0.3707	0.2443	0.6568	0.7246	0.7126	0.7156	0.6808	0.6893
LIFT	0.1122	0.3198	**0.1247**	0.7166	0.7209	0.7116	0.7256	0.6796	0.6904
ML-LOC	0.2035	**0.0576**	0.1659	0.7163	0.6078	0.6088	0.7245	0.6878	0.6901
G3P-kEMLC	0.1342	0.2091	0.2691	0.6471	0.7082	0.7062	0.7162	0.6811	0.6782
ML-BTC	**0.1291**	0.1336	0.2658	**0.7291**	**0.7248**	**0.7153**	**0.7268**	**0.7093**	**0.7175**

Table 5.6: Results for Yeast (medium)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.2994	0.2145	0.5021	0.2067	0.1912	0.1945	0.5062	0.1346	0.4341
ML-FOREST	0.2901	0.2119	0.4952	0.2198	0.1091	0.2098	0.5165	0.1472	0.5262
MLTL-HOMER	0.2817	0.4687	0.3128	0.1643	0.3303	**0.6515**	0.6407	0.4273	0.6397
CC	0.2611	0.3538	0.4034	0.1622	0.3364	0.6389	0.6145	0.4073	0.6045
ECC	0.2718	0.3637	0.3452	0.1849	0.3383	0.6376	0.6137	0.4152	0.6022
LSF-CC	0.3368	0.3526	0.3902	0.1772	0.3362	0.5261	0.6092	0.4242	0.5902
SC-RankSVM	0.3027	0.3334	0.2989	0.0008	0.3432	0.5264	0.5141	0.3728	0.5982
MLLEM	0.2452	0.3618	0.5006	0.0095	0.1716	0.1909	0.2031	0.1783	0.2381
MLKNN	0.2593	0.2882	0.2939	0.1899	0.3371	0.6432	0.6167	0.4182	0.6002
BR	0.2496	0.3827	0.2928	0.1506	0.3405	0.6363	0.6113	0.4027	0.5993
RAKEL	0.2981	0.4543	0.2889	0.1593	0.3446	0.6404	0.6189	0.4106	0.5912
LIFT	0.2322	0.4657	0.2974	0.5113	0.3177	0.6261	0.7182	0.3925	0.6445
ML-LOC	0.2493	**0.1911**	0.2989	0.5157	0.3149	0.5852	0.7125	0.4261	0.6374
G3P-kEMLC	0.2672	0.3051	0.3082	0.5083	0.3021	0.5463	0.6281	0.4096	0.6082
ML-BTC	**0.2342**	0.2671	**0.2891**	**0.5275**	**0.3443**	0.5605	**0.6339**	**0.4319**	**0.6425**

Table 5.7: Results for Enron (medium)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.0891	0.3438	0.6029	0.1218	0.0491	0.1363	0.2971	0.1784	0.2877
ML-FOREST	0.0921	0.3718	0.4582	0.1324	0.0535	0.1372	0.3092	0.1891	0.2983
MLTL-HOMER	0.0671	0.4985	0.4156	0.1017	0.1926	0.3662	0.4752	0.2707	0.4872
CC	0.0585	0.5142	0.1152	0.1211	0.2066	0.3694	0.5147	0.3058	0.5019
ECC	0.0584	0.5407	0.1382	0.1198	**0.2099**	0.3523	0.5192	0.3097	0.5008
LSF-CC	0.0899	0.5012	0.1109	0.1213	0.1672	0.3281	0.3872	0.2562	0.3982
SC-RankSVM	0.0971	0.4691	0.6921	0.0059	0.1448	0.2587	0.2209	0.1499	0.2432
MLLEM	0.8526	0.7816	0.8011	0.2018	0.1019	0.1306	0.129	0.0699	0.1625
MLKNN	0.0627	0.7791	0.5081	0.1141	0.0545	0.2807	0.3057	0.2461	0.3241
BR	0.0661	0.8417	0.3532	0.1251	0.1892	0.3692	0.5043	0.2962	0.5092
RAKEL	0.0562	0.6089	0.1852	0.1281	0.1941	0.3245	0.5206	0.3098	0.5021
LIFT	**0.0461**	0.5064	**0.0924**	0.3516	0.2077	0.3666	**0.6952**	0.3101	0.5179
ML-LOC	0.0531	0.3276	0.2324	0.3403	0.1763	0.3722	0.6705	0.3068	0.5181
G3P-kEMLC	0.0673	0.3569	0.4081	0.2081	0.1572	0.3298	0.4381	0.3022	0.5096
ML-BTC	0.0889	**0.3257**	0.3892	**0.3622**	0.1679	**0.3724**	0.4557	**0.3119**	**0.5193**

Table 5.8: Results for Image (medium)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.4112	0.2007	0.3298	0.5423	0.4901	0.4843	0.5001	0.4595	0.4078
ML-FOREST	0.3701	0.3811	0.4421	0.4253	0.1899	0.4361	0.5002	0.2831	0.5435
MLTL-HOMER	0.3656	0.6228	0.3405	0.4545	0.5545	0.5526	0.6192	0.5071	0.5699
CC	0.2829	0.5362	0.2039	0.3979	0.5452	0.5436	0.5497	0.5108	0.4673
ECC	0.3501	0.6316	0.2705	0.4341	0.5552	0.5512	0.5937	0.5531	0.4891
LSF-CC	0.3527	0.3821	0.2531	0.0212	0.3982	0.4803	0.4577	0.3225	0.4903
SC-RankSVM	0.4044	0.8241	0.3491	0.1231	0.5457	0.5367	0.5393	0.4195	0.2549
MLLEM	0.4144	0.8043	0.3745	0.4555	0.5616	0.5595	0.5677	0.4993	0.1092
MLKNN	0.2848	0.7992	0.3297	0.4792	0.5483	0.5365	0.5951	0.5058	0.5226
BR	0.3031	0.8213	0.3281	0.4761	0.5406	0.5571	0.6226	0.5053	0.4539
RAKEL	0.3013	0.6842	0.3155	0.4951	0.5354	0.5613	0.6258	0.5124	0.4401
LIFT	**0.2075**	0.5077	**0.1154**	0.5583	0.5581	0.5474	0.6051	0.4817	0.5297
ML-LOC	0.3298	**0.1432**	0.2682	0.5657	0.5481	0.5494	**0.6319**	0.5035	0.6026
G3P-kEMLC	0.3271	0.4821	0.4102	0.5023	0.5493	0.5692	0.5825	0.5193	0.4983
ML-BTC	0.2189	0.2429	0.4008	**0.5729**	**0.5637**	**0.5713**	0.5979	**0.5241**	**0.6089**

Table 5.9: Results for Water Quality (medium)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.3729	0.3929	0.4545	0.4101	0.1868	0.4262	0.4919	0.2781	0.5301
ML-FOREST	0.3701	0.3883	0.4342	0.4198	0.2012	0.4311	0.5002	0.3019	0.5291
MLTL-HOMER	0.3656	0.6215	0.4151	0.0028	0.4026	0.5533	0.5281	0.3397	0.5699
CC	0.3992	0.3486	0.1809	0.0162	0.3822	0.4939	0.4549	0.3264	0.4762
ECC	0.3501	0.4269	0.1726	0.0171	0.3767	0.4838	0.4466	0.3187	0.4553
LSF-CC	0.3499	0.4362	0.1766	0.0391	0.3361	0.3221	0.3872	0.3021	0.3992
SC-RankSVM	0.4044	0.7015	0.3953	0.0047	0.2018	0.2105	0.1981	0.1421	0.2029
MLLEM	0.4144	0.6751	0.4566	0.0047	0.1036	0.1058	0.0987	0.0701	0.1034
MLKNN	0.3448	0.7283	0.2406	0.0217	0.4247	0.4521	0.5181	0.3218	0.5224
BR	0.3531	0.7326	0.3198	0.0094	0.3594	0.4786	0.4351	0.3099	0.4555
RAKEL	0.3513	0.4955	0.2019	0.0151	0.3581	0.4599	0.4151	0.2964	0.4198
LIFT	0.3458	0.7343	**0.1141**	0.3699	0.2408	0.4143	0.5215	0.2562	0.5297
ML-LOC	0.3501	**0.2847**	0.3711	0.4671	0.4012	0.5065	0.5348	0.3429	0.6326
G3P-kEMLC	0.3581	0.4838	0.4192	0.0921	0.3762	0.4689	0.4266	0.3201	0.4226
ML-BTC	**0.3436**	0.3177	0.4039	**0.4796**	**0.4273**	**0.4955**	**0.5365**	**0.3431**	**0.5769**

Table 5.10: Results for Delicious (large)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.0213	0.7871	0.6786	0.0537	0.0538	0.0559	0.2065	0.0284	0.3333
ML-FOREST	0.0201	0.7628	0.4532	0.0673	0.0598	0.0582	0.2383	0.0293	0.3621
MLTL-HOMER	0.0308	0.3297	0.7291	0.0007	0.1045	0.2904	0.2345	0.1577	0.3092
CC	0.0287	0.1781	0.1459	0.0025	0.1023	0.2257	0.2243	0.1467	0.2561
ECC	0.0269	0.1864	**0.1226**	0.0025	0.1123	0.2466	0.2241	0.1466	0.2583
LSF-CC	0.0208	0.2034	0.2387	0.0031	**0.1298**	0.2473	0.2322	0.1372	0.2313
SC-RankSVM	0.0563	0.3721	0.6873	0.0032	0.0653	0.1982	0.1654	0.1221	0.1782
MLLEM	0.0292	0.5961	0.9757	0.0009	0.0126	0.0321	0.1326	0.0436	0.1028
MLKNN	0.0278	0.7371	0.3573	0.0019	0.0414	0.1829	0.1651	0.1051	0.1322
BR	0.0189	0.8824	0.3414	0.0012	0.0471	0.1933	0.1691	0.1089	0.1547
RAKEL	**0.0182**	0.2002	0.1426	0.0026	0.1013	0.2082	0.2181	0.1429	0.2093
LIFT	0.0489	0.3265	0.6632	0.0209	0.0782	0.0835	0.1782	0.1282	0.1872
ML-LOC	0.0193	**0.1637**	0.6034	0.0441	0.0533	0.0465	0.1963	0.0253	0.3925
G3P-kEMLC	0.0452	0.3082	0.6832	0.0018	0.0732	0.2109	0.2271	0.1342	0.2481
ML-BTC	0.0381	0.2816	0.6549	**0.2152**	0.0824	**0.2488**	**0.2439**	**0.1609**	**0.4012**

Table 5.11: Results for Corel (large)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.0344	**0.2824**	0.7633	0.1505	0.1495	0.1475	0.1015	0.0897	0.2457
ML-FOREST	0.0332	0.3281	0.7601	0.1023	0.1499	0.1501	0.1036	0.0901	0.2342
MLTL-HOMER	0.0227	0.3187	0.7636	0.0061	0.0464	0.1709	0.1074	0.1055	0.2231
CC	0.0225	0.3335	0.5130	0.0076	0.0476	0.1571	0.1024	0.0986	0.2291
ECC	0.0223	0.3454	0.5590	0.0072	0.0471	0.1691	0.1114	0.0998	0.2353
LSF-CC	0.0243	0.4326	0.5722	0.0083	0.0542	0.1503	0.1126	0.0953	0.2245
SC-RankSVM	0.0256	0.4692	0.5328	0.0091	0.0782	0.1461	0.1092	0.0896	0.2135
MLLEM	0.0951	0.6971	0.9738	0.0012	0.0177	0.0194	0.0194	0.0098	0.0162
MLKNN	0.0309	0.6165	0.7291	0.0004	0.0051	0.1076	0.1014	0.0656	0.1132
BR	0.0292	0.7519	0.6412	0.0066	0.0227	0.1701	0.1116	0.1053	0.1642
RAKEL	0.0316	0.3651	0.6458	0.0092	0.0454	**0.1722**	0.1077	0.1035	0.1527
LIFT	0.0228	0.9893	**0.0153**	0.0214	0.1493	0.0244	0.0449	0.0146	0.0451
ML-LOC	0.0432	0.3542	0.8721	0.0127	0.0321	0.0453	0.1064	0.0873	0.2099
G3P-kEMLC	0.0312	0.3581	0.8089	0.0032	0.1093	0.1321	0.1019	0.1071	0.1823
ML-BTC	**0.0221**	0.3101	0.7592	**0.1566**	**0.1527**	0.1521	**0.1129**	**0.1096**	**0.2483**

Table 5.12: Results for Bibtex (large)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.0314	0.3784	0.4511	0.1567	0.1414	0.3257	0.2549	0.2367	0.3761
ML-FOREST	0.0301	0.3421	0.4982	0.2001	0.1231	0.3091	0.4021	0.2319	0.3533
MLTL-HOMER	0.0312	0.4037	0.5857	0.0949	0.1497	0.3186	0.3239	0.2139	0.3382
CC	0.0315	0.4131	0.2597	0.1532	0.1285	0.4331	0.4058	0.2247	0.3502
ECC	0.0309	0.4245	**0.2504**	0.1559	0.1259	0.4293	0.4065	0.2155	0.3609
LSF-CC	0.0308	0.4372	0.3092	0.1872	0.1099	0.4019	0.4122	0.2281	0.3521
SC-RankSVM	0.0391	0.5764	0.6972	0.1789	0.1088	0.1342	0.2002	0.1543	0.2091
MLLEM	0.0829	0.8182	0.7921	0.0871	0.0849	0.1223	0.1481	0.1292	0.1562
MLKNN	0.0762	0.5846	0.6531	0.0552	0.0562	0.1882	0.1837	0.1433	0.1892
BR	0.6823	0.3961	0.3602	0.2151	0.1504	0.4313	**0.4519**	0.2139	0.3728
RAKEL	0.0321	0.5261	0.3505	0.1813	0.1232	**0.4378**	0.4275	0.2283	0.3592
LIFT	0.0308	0.7466	0.3607	0.2734	0.1424	0.2885	0.4072	0.2295	0.3825
ML-LOC	0.0312	**0.2858**	0.7872	0.1201	0.0563	0.1264	0.2128	0.0961	0.2115
G3P-kEMLC	0.0304	0.5628	0.5879	0.2415	0.1035	0.2473	0.2891	0.2029	0.3741
ML-BTC	**0.0299**	0.5524	0.5736	**0.2919**	**0.1586**	0.2597	0.3014	**0.2391**	**0.3911**

Table 5.13: Results for EUR-Lex (large)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.2478	0.1378	0.7369	0.0489	0.2817	0.0493	0.0257	0.0254	0.1405
ML-FOREST	0.2082	0.13471	0.7223	0.0557	0.2899	0.0532	0.0281	0.0277	0.1498
MLTL-HOMER	0.0353	0.4985	0.5064	0.1147	0.2091	0.1407	0.1443	**0.1787**	0.1402
CC	0.0445	0.7851	0.5373	0.1479	0.2908	0.1376	0.2631	0.1786	0.1601
ECC	0.0405	0.7861	0.4433	0.1401	0.2914	0.1415	**0.2634**	0.1783	0.1608
LSF-CC	0.0392	0.7212	0.3981	0.1321	0.2739	0.1548	0.1621	0.1534	0.1512
SC-RankSVM	0.1028	0.7328	0.7621	0.1092	0.1087	0.1101	0.0081	0.0721	0.1024
MLLEM	0.1986	0.7736	0.7759	0.0021	0.0208	0.0218	0.0217	0.0112	0.0342
MLKNN	0.0367	0.9355	0.5332	0.1316	0.2154	0.1509	0.1115	0.1427	0.1526
BR	0.0353	0.8478	0.3221	0.1201	0.1953	0.1417	0.1132	0.1515	0.1526
RAKEL	0.0346	0.7107	0.3543	0.0982	0.2914	0.1281	0.1775	0.1718	0.1631
LIFT	0.0411	0.5628	**0.3182**	0.1834	0.2823	0.1521	0.1315	0.1026	0.1609
ML-LOC	0.0337	**0.1312**	0.7621	0.1316	0.2899	0.1575	0.2374	0.0943	0.1606
G3P-kEMLC	0.0423	0.6619	0.7732	0.1082	0.2781	0.1462	0.1247	0.0932	0.1622
ML-BTC	**0.0315**	0.6015	0.7689	**0.1551**	**0.2968**	**0.1579**	0.1334	0.1083	**0.1699**

Table 5.14: Results for Yelp (large)

	HL	RL	OE	SA	MacF1	MicF1	FM	Acc	GM
ML-TREE	0.2649	0.2028	0.2816	0.5364	0.3104	0.5198	0.7066	0.4639	0.6203
ML-FOREST	0.2561	0.2003	0.2781	0.5463	0.3201	0.5261	0.7098	0.4873	0.6571
MLTL-HOMER	0.1856	0.3269	0.2432	0.3789	0.6513	0.7076	0.658	0.5842	0.6753
CC	0.1805	0.3847	0.1432	0.3905	0.6511	0.7063	0.652	0.5821	0.6657
ECC	0.1804	0.3524	0.1583	0.3882	0.6513	0.7061	0.6503	0.5803	0.6601
LSF-CC	0.1782	0.3658	0.1408	0.3801	0.6572	0.6741	0.6577	0.5899	0.6709
SC-RankSVM	0.2584	0.2064	0.3254	0.2291	0.4526	0.4539	0.3919	0.3402	0.3892
MLLEM	0.3275	0.3236	0.5555	0.0745	0.4638	0.4766	0.4189	0.3902	0.4093
MLKNN	0.274	0.2626	0.3351	0.1904	0.1581	0.3299	0.2664	0.2281	0.2761
BR	0.1785	0.1556	0.2476	0.3899	0.6348	0.7035	0.6444	0.5757	0.6579
RAKEL	0.1663	0.2271	0.1872	0.4555	0.6855	0.7266	0.6908	0.6271	0.6866
LIFT	0.1951	0.3981	0.1641	0.6336	0.6203	0.6961	0.7037	0.5846	0.6749
ML-LOC	**0.1095**	0.1579	0.1425	0.6345	**0.6735**	**0.7162**	0.6988	**0.6513**	0.6844
G3P-kEMLC	0.2231	0.1629	0.1572	0.4781	0.5893	0.6521	0.6858	0.5538	0.6583
ML-BTC	0.2167	**0.1461**	**0.1387**	**0.6436**	0.5944	0.6602	**0.7104**	0.5619	**0.6872**

is quite difficult to achieve. While the others base their outcome on partial correctness, the "subset accuracy" metric determines the full correctness, i.e., the correct prediction of the entire label-set. For multi-label data, this is a difficult feat to achieve, even for a small number of test instances. However, from the results, it is seen that the proposed ML-BTC surpasses all the other methods by a significant margin for SA. Especially for the CAL500 dataset, where eleven of other methods have scored SA = 0. Similarly, metrics F-Measure and G-Mean are used to better assess the class imbalance issue in ML data. It is seen that for most datasets, ML-BTC has surpassed the others in terms of FM and GM, indicating its ability to handle class imbalance.

From a dataset perspective, ML-BTC has performed equally well for datasets of small, medium and large sizes. Only for the ranking-based metrics, ML-LOC is seen to have a better RL, whereas, LIFT shows a better OE compared to the other methods. ML datasets in general have the drawback of class imbalance, and all the fourteen datasets used for experimentation are quite imbalanced (shown in Section 2.2.1). Out of the fourteen datasets, CAL500, Delicious and Yelp, in particular have 1, 0.981 and 1 diversity respectively, indicating a high ratio of distinct label-sets to the number of instances. This makes the train set and test set completely different and thus are likely to get misclassified.

Yet, the proposed ML-BTC performs substantially well compared to the other algorithms, especially for SA, FM and GM metrics.

For datasets like CAL500, Delicious and Yelp, where the training set and test set will have mostly distinct label-sets, the propagation of error from the root node to the leaf nodes in a tree structure can be a major concern. Any misclassification at an intermediate node might propagate to the leaf nodes, affecting the final classification results. However, through the performance achieved by ML-BTC, it is clear that it has not succumbed to the problem of error propagation. In the proposed ML-BTC, since there is a separate binary classifier at every intermediate node which learns different decision boundaries, any data point that is misclassified at one level can be rectified in the later steps. At every node, since an approximate partition is being estimated, some amount of misclassification is expected to occur. However, the novel approach of label-space partitioning ensures that data from similar classes will belong to the same side of the partition. Since the final classification does not take place until the sample reaches some leaf node, even if some data have been misclassified at the intermediate nodes, it still has a chance to be correctly classified at the leaf nodes. Either an entire leaf node is allocated for that label-set, or the ML classifiers at the leaf nodes handle these anomalous points, thus preventing the error from affecting the results.

To evaluate the sensitivity of ML-BTC to the two parameters, namely multi-label entropy, H, and sample cardinality, SC, that orchestrate the building of the tree-like structure, experiments have been performed for different combinations of $H_{threshold}$ and $SC_{threshold}$. Figure 5.7 depicts the G-Mean values (Y-axis) obtained for the fourteen datasets by considering three combinations of $H_{threshold}$ and $SC_{threshold}$. The thresholds have been set to a fraction of the initial entropy, ($H_{initial}$), and training sample size, N_{tr}. In each case, $H_{threshold} = \frac{H_{initial}}{den}$ and $SC_{threshold} = \frac{N_{tr}}{den}$. The X-axis represents three values for the denominator of thresholds, $den = 3, 5, 10$. On the X-axis, when $den = 3$ it depicts results from ML-BTC experimented with parameters, $H_{threshold} = \frac{H_{initial}}{3}$ and $SC_{threshold} = \frac{N_{tr}}{3}$,

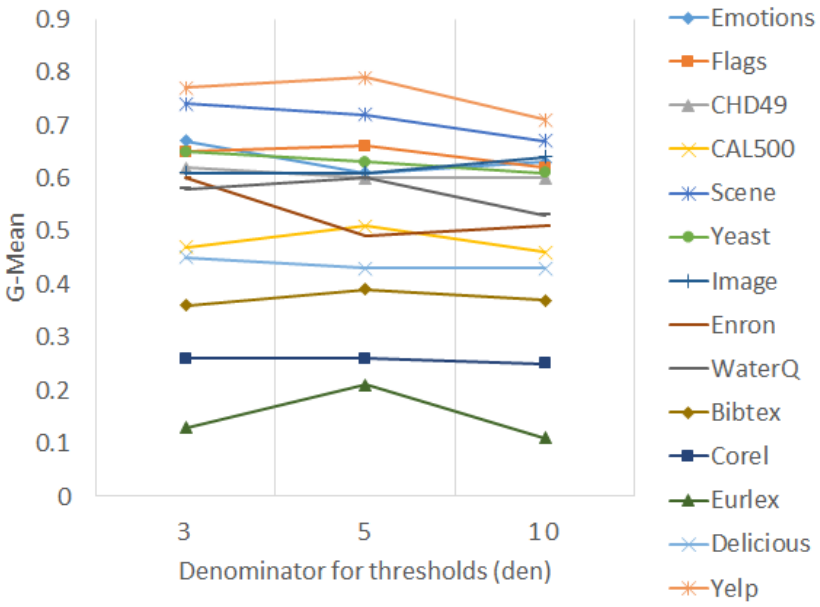


Figure 5.7: Effect of the two parameters $H_{threshold}$ and $SC_{threshold}$ on G-Mean metric

and so on. Analysis of the performance of the proposed algorithms for the various combinations of thresholds on the datasets shows that the G-Mean does not vary drastically with the variation of the thresholds. Within the range tested, the exact choice of thresholds do not seem crucial. This shows that the proposed algorithm is not too dependent on the thresholds and can perform efficiently with any approximate value chosen within the range.

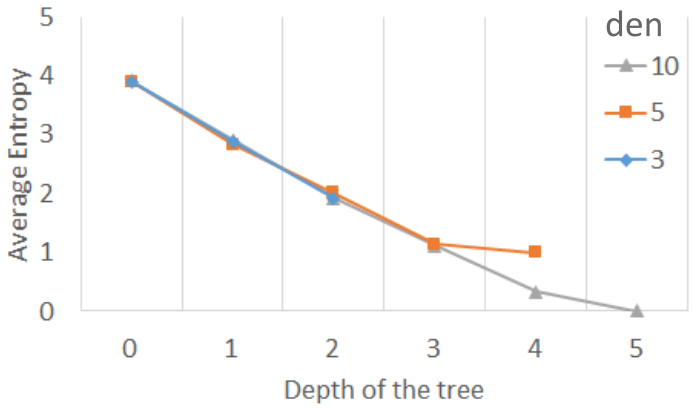


Figure 5.8: Average Entropy at every depth of the ML-BTC tree for different ($H_{threshold}$, $SC_{threshold}$) combination for Scene dataset

To investigate the tree structure built by ML-BTC algorithm, in Figure 5.8 and Figure 5.9

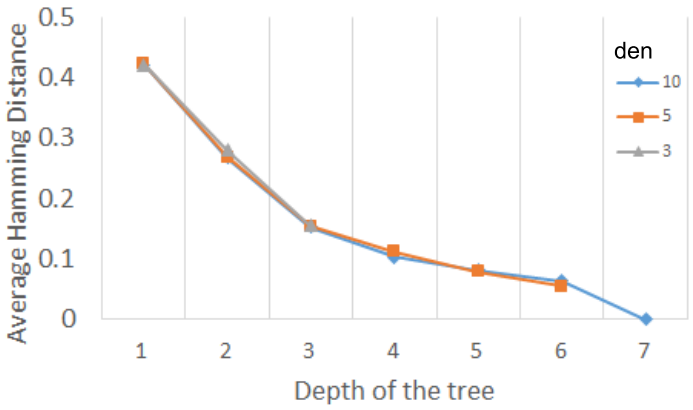


Figure 5.9: Average Hamming distance at every depth of the ML-BTC tree for different $(H_{threshold}, SC_{threshold})$ combination for Emotions dataset

plots of the average entropy and average Hamming distance at each depth of the trees have been shown respectively. These results have been obtained from different runs with different training sets. Three combinations of thresholds have been considered. $H_{threshold} = \frac{H_{initial}}{den}$ and $SC_{threshold} = \frac{N_{tr}}{den}$, where den = 3, 5, 10. From the graphs, it is seen that the average entropy and Hamming distance at the nodes of a particular depth of the tree keeps systematically decreasing. Entropy determines the diversity of the data, while Hamming distance portrays the existing closeness and label dependencies. The tree building is effectively stopped before all the nodes attain entropy and Hamming distance 0.

To evaluate the performance of the proposed method even further, T-test statistics have been computed for the proposed ML-BTC against all the other algorithms for the subset accuracy metric. It is seen that for $t_{.95} = 0.771$ with degrees of freedom = 13 the proposed method outperforms the others. Additionally, the non-parametric two-tailed Wilcoxon signed-rank test [3] (Table 5.16) has been performed on the accuracy metric for all datasets against all the other methods. It indicates that with alpha=0.20, $T_{Wilcoxon}(14)$=31, the proposed ML-BTC is statistically superior to all the other algorithms for the accuracy metric.

Table 5.15: T-Test for all methods against ML-BTC (based on subset accuracy)

Method	**T-test value**
ML-TREE	4.0477
ML-FOREST	5.3618
MLTL-HOMER	6.3063
CC	6.0099
ECC	5.8155
LSF-CC	6.0655
SC-RankSVM	9.6692
MLLEM	7.7906
MLKNN	6.6234
BR	5.4556
RAKEL	5.7183
LIFT	1.7334
ML-LOC	2.1339
G3P-kEMLC	4.2790

Table 5.16: Two-tailed Wilcoxon signed-rank test statistics for all the methods vs proposed ML-BTC (based on accuracy)

Method	***p* Value**	**Value of sign rank**
ML-TREE	0.000122	105
ML-FOREST	0.000122	105
MLTL-HOMER	0.067627	82
CC	0.172607	75
ECC	0.135254	77
LSF-CC	0.019043	89
SC-RankSVM	0.000122	105
MLLEM	0.000122	105
MLKNN	0.003052	97
BR	0.040039	85
RAKEL	0.090576	80
LIFT	0.002319	98
ML-LOC	0.008545	93
G3P-kEMLC	0.000122	105

5.2.2.3 Computational Analysis

For an overall computational analysis of the proposed algorithm in comparison with the fourteen state-of-the-art techniques, Table 5.17 depicts the runtime for a single run of the model comprised of the training and testing phase. For the proposed algorithm, both the *train time* and the *test time* have been separately indicated. Train time is the time elapsed to build and train the ML-BTC structure using the training set and the test time is the total classification time for the entire test set. From the noted computational times, it is seen that the proposed algorithm maintains an average computational time with respect to the other methods. However, methods like MLLOC, ML-FOREST, G3P-kEMLC, SC-RankSVM, LSF-CC, ECC have a comparatively higher computational time than ML-BTC. Comparing the train time and the test time of ML-BTC it is seen that the test time is quite lower than the training time, thus if training is performed offline, the model can perform efficiently in the testing phase.

Thus, analysing the overall performance of the proposed method, it can be said that the ML-BTC fares well above the other state-of-the-art algorithms for multi-label classification. Significant improvement in most cases when compared to the tree-based models, ensemble classifiers, data transformation techniques and other standard ML algorithms can be observed.

5.2.2.4 Comparison with the previous proposed works

The proposed works in the previous chapters, namely MLSAEELM and AE-MLFLANN were built to handle the large input dimension and the complex decision space. The ML-BTC also deals with complex decision space, while handling class imbalance and label correlations. Thus, the comparison with the previous methods can be made on the grounds of handling the complex decision space problem. Each of the three proposed methods handle the problem through different approaches, hence, it is interesting to analyse which approach performs the best among these three. From Table 5.18 it is

Table 5.17: Average run time of all methods for all datasets

		Small				*Medium*					*Large*			
	Emotions	**CAL500**	**Flags**	**CHD49**	**Scene**	**Yeast**	**Enron**	**Image**	**WQ**	**Corel**	**Delicious**	**Bibtex**	**Eurlex**	**Yelp**
ML-TREE	1.66	312.18	1.84	9.68	22.43	32.17	583.62	21.57	11.13	339.09	2295.27	3377.69	3707.45	399.1
ML-FOREST	4.63	408.34	5.62	15.27	26.89	27.54	697.32	25.43	17.78	501.24	3526.87	4032.89	4573.89	790.32
MLTL-HOMER	3.05	6.35	4.09	5.92	20.54	20.03	316.41	19.02	9.02	345.66	3164.61	1218.24	3587.43	593.65
CC	2.78	80.82	1.15	1.15	23.85	23.63	436.31	24.37	6.58	493.21	2385.13	657.68	3894.08	768.91
ECC	5.98	100.32	2.98	7.43	26.73	27.76	598.64	23.44	7.36	570.53	2873.76	1673.65	3945.87	836.45
LSF-CC	5.62	110.87	3.82	8.87	22.98	28.65	673.71	21.54	10.73	489.32	2981.98	1572.98	3598.43	840.76
SC-RankSVM	60.15	101.42	4.26	20.41	609.84	657.62	1493.49	756.13	956.93	1792.54	7986.43	10879.34	11857.49	3181.45
MLLEM	2.33	81.37	3.19	4.32	20.32	23.65	122.66	18.69	13.87	398.34	323.08	545.16	703.92	500.31
MLKNN	96.77	258.76	157.37	179.87	158.43	150.98	342.17	143.33	137.25	371.98	1504.19	1171.82	2083.92	421.44
BR	7.95	35.16	1.52	8.61	17.54	20.88	474.06	14.43	9.44	459.03	2437.43	924.38	3398.21	544.44
RAKEL	2.82	45.58	2.9	4.97	21.33	25.31	570.06	20.62	10.62	298.39	2664.8	958.57	3574.98	396.17
LIFT	3.44	79.61	2.19	4.04	11.17	18.6	401.07	18.46	11.8	1768.08	2763.98	4239.6	4770.7	3607.61
MLLOC	18.1	33.37	2.73	21.68	668.69	567.43	788.61	434.45	223.17	402.43	8349.4	1569.76	5774.63	820.37
G3P-kEMLC	4.83	350.45	4.98	6.52	27.49	30.21	700.83	25.76	8.92	547.82	2872.76	1487.56	3587.92	789.23
	Train Time	2.36	78.4	2.61	3.02	24.01	24.78	647.36	16.45	6.57	354.38	2342.75	1293.25	3304.75	719.92
ML-BTC	*Test Time*	0.09	0.44	0.13	0.16	0.57	0.24	1.23	0.33	0.36	5.67	13.03	2.09	3.06	26.25
	Total	2.45	78.84	2.74	3.19	24.58	25.02	648.59	16.78	6.93	360.05	2355.78	1295.34	3307.81	746.17

Table 5.18: Comparison with the other proposed methods

Dataset	Method	HL	RL	OE	MacF1	MicF1	SA
	MLSAEELM	0.0918	0.0760	0.2225	**0.8352**	**0.8355**	0.6126
Scene	AE-MLFLANN	**0.0761**	**0.0707**	**0.2119**	0.7702	0.7562	0.6398
	ML-BTC	0.1291	0.1336	0.2658	0.7248	0.7153	**0.7291**
	MLSAEELM	**0.1852**	0.1580	0.2735	**0.7245**	**0.7232**	0.5695
Emotions	AE-MLFLANN	0.1863	**0.1420**	**0.2395**	0.6769	0.6925	0.3221
	ML-BTC	0.2126	0.2164	0.2639	0.6352	0.6601	**0.6384**
	MLSAEELM	**0.2612**	**0.1895**	**0.1429**	**0.6221**	0.6014	**0.6758**
Flags	AE-MLFLANN	0.2656	0.2025	0.1867	0.5802	**0.7219**	0.1758
	ML-BTC	0.3438	0.3135	0.2158	0.5085	0.6114	0.5922
	MLSAEELM	**0.1943**	**0.1648**	**0.2222**	**0.7444**	**0.7267**	**0.6104**
Yeast	AE-MLFLANN	0.2071	0.1766	0.2338	0.4034	0.6325	0.1541
	ML-BTC	0.2342	0.2671	0.2891	0.3443	0.5605	0.5275

seen that MLSAEELM performs best in most cases for the four datasets compared. The effectiveness of the network can be credited to the improved input to output mapping approach that focussed on learning the data well. Intuitively, ML-BTC is aimed to handle more issues of ML data, and is seen to perform well compared to the state-of-the-art, but is underachieving than expected when compared to the other proposed models. In this algorithm, an approximate approach has been opted to simplify the complex decision space, handle the class imbalance issue of ML data and implicitly include label correlations in the data. These have been handled to some extent in this model and can be investigated further in the future.

5.3 Conclusion

In this chapter, the problem of class imbalance in multi-label data was aimed to be handled. This would assure improved performance of the classifier by reducing biased training towards the larger classes. Additionally, the intention was to utilize the correlations among ML classes that exist but are often ignored and simplify the decision space while doing so. This would also help improve classification by incorporating the depen-

dencies that exist among labels. In this regard, a tree of classifiers was proposed that would handle these issues simultaneously. As the tree is built, the data is strategically split at each intermediate node and specific classifiers are trained. The imbalance is handled by the use of different classifiers suitable for the data at a node. The novel approach of label-space partitioning helps to retain label correlations and simplify class boundaries without having to deal with the overlaps. Experimentally, the proposed algorithm has shown good outcomes against various relevant methods.

Administering the solutions for complex decision space, class imbalance and label correlations, the proposed model is seen to be quite capable of handling the different issues. However, the concept of class correlations were not completely utilized in this method since they were used implicitly to preserve the correlations. To explore them in a better way, and use them to improve classification performance, the next chapter extracts class correlations to use them explicitly to improve the performance of other classifiers.

Chapter 6

Improved Multi-label Classification with Frequent Label-set Mining and Association

6.1 Introduction

After exploring the solutions to various problems of multi-label data in the previous chapters, it is seen that the avenue of label correlations have not been utilized to its full potential. Since, most existing multi-label algorithms do not focus on this aspect, a dedicated method is to be developed here that incorporates class correlations to build a strong classification model. Due to the simultaneous occurrence of multiple classes in multi-label data, some frequently co-occurring labels might have some dependencies among them. However, in multi-label literature, usually every class is considered to be independent and the predictions for each class is performed separately. This loses out on any class correlation that might exist in the data. For example, while analysing user movie choices it can have genre classes like 'action', 'comedy', 'thriller', 'romance',

'science fiction (sci-fi)' etc. Among these, users preferring 'action' movies are highly likely to prefer 'thriller' as well. Movies genres are multi-label in nature hence not considering 'romantic-comedy', 'action-thriller' or 'action-sci-fi' as correlated classes might be a lack of judgement. This would eventually lose a lot of information, thus leading to deceiving results. Similar issues might be seen in other types of multi-label data as well.

In this chapter, an approach has been proposed to incorporate class correlations that exist in the multi-label data. This is done to improve the overall classification performance of any multi-label classifier that otherwise might have ignored these dependencies. In this regard, a concept named "frequent label-set mining" (FLM) has been introduced. Here, the traditional concept of frequent itemset mining which finds the relation among frequently occurring itemsets has been utilized. Most researchers determine frequent itemsets and association rules based on the feature space of the data. Authors in [75] developed a multi-class, multi-label associative classification (MMAC) approach, which detects all frequent itemsets from sample features occurring in the data and the classes associated with them. An application of the apriori algorithm has been done in [56]. Here, the label-sets have been searched for correlations and the compound labels that have a strong association, have been replaced by single labels before classification. ML-KNN has been then used for multi-label classification, after which the labels have been reverted to their original form. However, unlike the common approach of considering each feature as an item, the perspective is changed to a fresh angle, where the labels are considered as items. Thus, turning the label-sets of multi-label data into itemsets. This ensures that the class correlations are found among frequently co-occurring labels when they are considered as frequent itemsets. FLM is performed on the training data to extract the correlated classes. In this approach, concepts of co-presence (CP) and co-absence (CA) of classes have been presented. These help to extract relevant and irrelevant associations among classes in the form of association rules. This FLM approach is then used in unison with a few benchmark ML classifiers to improve their classification accuracy. The independent classifier trains itself on the training set and generates the

soft classification scores. Depending on the ambiguity of the scores, they are marked as *certain* or *uncertain*. The *uncertain* scores are then improved with the help of the important CP-CA rules extracted from the classes with *certain* scores. The CP rules include frequently co-occurring classes, while the CA rules discard those classes which are frequently absent together. A novel score improvement formula has been defined to modify the soft classification scores. Once the soft scores are updated depending on the class associations, they become more relevant or irrelevant and thus less ambiguous. After this, a hard classification is performed using a threshold function to generate the final label-sets.

The contribution of the proposed work is as follows.

- Introduce the concept of "frequent label-set mining" for finding class/label correlations. This identifies co-presence (CP) and co-absence (CA) among classes to generate rules for relevant and irrelevant label-sets.

- Define a formula for the improvement of *uncertain* scores incorporating CP-CA rules of *certain* classes.

- Improve classification score for any classifier by incorporating class correlations with the help of frequent label-set mining.

The proposed algorithm, frequent label-set mining and association (FLMA), has been tested in combination with three ML classifiers to improve their performance on ten benchmark datasets. Experimental analysis indicates substantial improvement with the application of the proposed approach to incorporate class correlations on existing ML classifiers.

The rest of the chapter is organized as follows. Section 6.2 elaborates on the proposed work, including the model description and all the experimental results done for this work. Section 6.3 concludes the chapter.

6.2 Proposed Multi-label Frequent Label-set Mining and Association

The proposed system is designed to aid multi-label classifiers with frequent label-set mining to utilize class correlations. This aims to improve the classification performance achieved by a regular classifier that does not consider existing relations among classes.

6.2.1 Model Description

In this section the major steps of the proposed method has been discussed in detail.

6.2.1.1 Frequent label-set mining

Itemset mining is done to find frequently occurring items in a given dataset. In the scenario of multi-label data, each sample is associated with multiple classes at the same time. This set of relevant classes constitute the label-set for a particular sample. Similarly, the classes that are not associated with a particular sample, constitute its irrelevant label-set. To map an ML dataset with itemset mining, each of the classes is considered as individual items, the label-sets are itemsets and then mining is performed in the output space. Frequent label-set mining would identify the classes that frequently occur together, hence can be considered to be correlated. The aim of FLM is to generate positive rules for classes occurring together and negative rules for classes that are always absent together. This ensures that all grounds are covered for classes with a similar occurrence pattern. Thus two concepts are introduced here:

a. Co-presence (CP) label-sets: These indicate classes that frequently occur together. It helps to identify frequent relevant label-sets.

b. Co-absence (CA) label-sets: These indicate classes that are frequently missing together. It helps to identify frequent irrelevant label-sets.

FLM generates both CP and CA rules in this step. However, if there are a large number of classes, the number of CP-CA label-sets would be huge and their corresponding rule generation might be computationally expensive. For C number of ML classes, the possible relevant and irrelevant label-sets is 2^C. ML data tend to have a large number of classes, which in turn would create a large number of label-sets. Hence, to reduce the bulk of the classes, some non-frequent classes are ignored at first to reduce the number of possible label-sets to some extent. This is an optional step taken for larger datasets. All classes with above-average occurrence frequency are considered for rule generation. The aim is to primarily handle classes that have the highest probability of occurrence. From the reduced set of classes, the frequent label-sets are identified and their corresponding association rules are generated using the well-known FP-growth algorithm [26]. FP-growth builds a tree structure that is fast and can handle a large output dimension.

FP-Growth Algorithm FP-growth is a popular method of mining frequent patterns (FP) in data that is specifically preferred over the Apriori algorithm since it can handle larger dimensional data. This is typically used on transactions related to a set of items to identify the frequently occurring itemsets from the entire list. The data can be accessed in two formats; the horizontal format, where the data is represented through transaction ids, and each id contains multiple items purchased in that transaction. The other is a vertical format, where each item is listed along with all the corresponding transactions in which they occur.

This algorithm has been adapted to be utilized in the proposed work to identify frequent label-sets in multi-label data. The data at hand is represented in a horizontal format, where each instance is associated with multiple classes which constitute its relevant label-set. Similarly, the classes that are not relevant to the instance constitute its irrelevant label-set. These relevant and irrelevant sets of classes are used to identify the frequent label-sets that exist in the data. For this purpose, the FP-growth algorithm has been employed on the label-sets of the multi-label data. The FP-growth algorithm first requires

the building of the FP-Tree. The steps to construct the FP-Tree for relevant label-sets are as follows:

a. Scan the data and compute the support count for every individual label. The support of a class Y is computed as,

$$Support(Y) = \frac{\text{Samples with label } Y}{N}. \tag{6.1}$$

This can be done through one scan of the data.

b. Sort the labels in descending order of support count. min_sup is the minimum support threshold which determines the frequent label-sets with support count above this threshold. All non-frequent labels can be eliminated.

c. Begin creating the FP-Tree by forming an empty root node. Any node that gets added, has a *support counter*.

d. For each instance i,

i Sort its corresponding relevant labels in the order recorded in Step 2 and form a list Rel_i.

ii Start at the root node, take labels from Rel_i sequentially and traverse the tree.

iii If a label exists in order of the sequence, increment its counter and traverse to the next branch.

iv Add a branch to the tree, for every label in the sequence that does not exist and set its counter to one.

e. Stop expanding the tree once all instances have been traversed.

Once the FP-tree has been created, frequent itemsets can be extracted by traversing from the root node to the leaf nodes. The support counter at each node indicates the frequency of that particular path from the root. Using a minimum support threshold, the

paths with high support count can be selected, thus identifying the frequent co-presence label-sets. The FP-Tree for irrelevant label-sets is built and frequent co-absent label-sets are extracted in a similar way.

After the frequent label-sets have been identified association rules are generated from them. For every frequent label-set $\{Y_1, Y_2\}$, association rules $\{Y_1\} \rightarrow \{Y_2\}$ and $\{Y_2\} \rightarrow \{Y_1\}$ can be formed. Each rule generated is of the form $\{Y_1\} \rightarrow \{Y_2\}$, where $\{Y_1\}$ and $\{Y_2\}$ can be single or multiple classes and the rule indicates an association between them. Each of these rules has two parameters associated with them as follows.

$$Support(\{Y_1\} \rightarrow \{Y_2\}) = \frac{\text{Samples with label-set}\{Y_1, Y_2\}}{N} \tag{6.2}$$

$$Confidence(\{Y_1\} \rightarrow \{Y_2\}) = \frac{\text{Samples with label-set}\{Y_1, Y_2\}}{\text{Samples with label}Y_1} \tag{6.3}$$

These parameters indicate the importance of the rules, depending on which they have been used at a later stage. To select most important association rules, a minimum confidence threshold, min_conf is also set. All rules above the min_sup and min_conf thresholds constitute the final set of rules. The min_sup and min_conf thresholds for irrelevant label-sets is kept much higher since the frequency of 0's in the label space is higher than the frequency of 1's. This causes the support and confidence ranges for irrelevant labels to be higher than the relevant labels. The rules generated from the relevant label-sets are the co-presence rules and the ones generated from the irrelevant label-sets are the co-absence rules.

Thus, the FLM step is performed in the training phase, and it helps to identify the relationship between sets of frequently occurring and frequently absent classes. It generates two sets of rules, one for relevant and one for irrelevant label-sets some of which are used later in the testing phase to improve the classification performance of a multi-label classifier.

6.2.1.2 Soft classification

In this step, an existing multi-label classifier is trained on the ML data in the training phase of the method to get the classification scores. Most classifiers do not explicitly consider class correlations or incorporate class information while training the classifier. Classifiers like ML-KNN, ML-RBF, multi-layer perceptrons, etc. are standard methods that can be improved using the proposed approach. Final or hard classification is not performed in this step. The soft classification scores obtained from the ML classifier will be combined later with the rules generated in the FLM step. Since the MLC performs the training phase separately, it can be done in parallel while performing FLM. In the testing phase, the trained model is used to predict classification scores, which are later improved using the proposed approach.

6.2.1.3 Associative Correlation

This step combines the outputs generated from the FLM and soft classification steps. The soft scores obtained from the classifier are aimed to be improved by correlating the CP-CA label-set rules generated in the first step. The scores can lie within the range of 0 to 1 or -1 to 1; here, [0,1] is considered. As shown in Figure 6.1, 0 indicates irrelevance while 1 indicates relevance. The soft scores obtained can be segregated into two categories.

a. **Certain scores** - The scores lying close to the boundary values are the ones the classifier is most certain about. Thus, classes that have scores lying close to 0 are surely irrelevant, and scores close to 1 are relevant.

b. **Uncertain scores** - On the other hand, the region around the mean of the score range (in this case, 0.5) can be considered ambiguous. Thus, classification scores which lie close to the mean (0.5) are quite uncertain. A slight error by the classifier might lead to misclassification.

Thus, it can be said that the *certain* scores do not need much intervention and can be considered to be correct. Most misclassifications occur due to the scores that lie in the

uncertainty region. In Figure 6.1, a "region of uncertainty" has been demarcated. Two

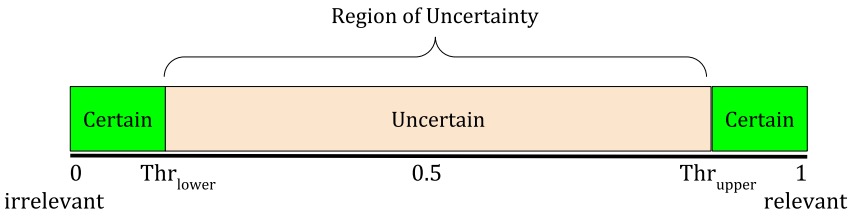


Figure 6.1: Region of Uncertainty

regions close to the boundaries (0 and 1) are considered to be *certain*, while the region around the mean (here, 0.5) is *uncertain*. Boundary thresholds are determined which can distinguish the *certain* regions from the *uncertain* region. Thr_{lower} determines the lower threshold closer to 0, separating the *certain* and *uncertain* region for irrelevant classes, while Thr_{upper} determines the upper threshold closer to 1, separating the *uncertain* and *certain* scores for the relevant classes. These thresholds are computed by fitting an S-membership function on the soft scores obtained from the classifier. This work aims to improve the *uncertain* scores with the help of the *certain* scores and the relevant-irrelevant class correlations generated in the FLM step.

Steps of Associative Correlation This phase combines the soft classification scores with the CP and CA rules.

a. Assign hard labels to the *certain* scores, i.e $score > Thr_{upper}$ and $score < Thr_{lower}$.

b. Clean the overlapping CP and CA rules for the same set of classes. Keep the rules that have higher support and confidence.

c. Sort the CP and CA rules in descending order of their confidence and support. Higher the confidence, more important is the rule.

d. Sort the *certain* classes and *uncertain* classes based on their distance from the boundary. The shorter the distance from either boundary, the more confident is the score.

e. Take a *certain* class and find all the CP and CA rules involving that class.

f. From the soft scores, find the most *uncertain* class and improve its score.

g. Score improvement: CP rule moves the score towards 1, CA rule moves the score towards 0.

Improving Uncertain Scores Using the *certain* labels and the CP-CA rules, the scores of the *uncertain* samples are improved. With the help of CP rules, scores are increased to be closer to 1, and with CA rules, scores are moved towards 0. The chosen rule $\{YC\} \rightarrow \{YU\}$ has {YC} as a *certain* label-set and {YU} as an *uncertain* label-set. Equation 6.4 determines the change Δ by which the *uncertain* scores will be improved.

$$\Delta = Confidence(\{YC\} \rightarrow \{YU\}).\frac{|YU_{score} - NB|}{|YC_{score} - NB|}, \tag{6.4}$$

where, NB is the nearest boundary. The numerator incorporates the score of the *uncertain* class (YU_{score}) and the confidence of the rule in question. Higher the confidence of the rule and the distance of YU_{score} from the nearest boundary (i.e., a measure of ambiguity), the more the impact. The denominator determines the distance of YC_{score} (*certain* class) from the nearest boundary. Better the *certain* score, higher the impact. Δ determines the shift from the original score to its desired score.

For CP rules, the scores are increased to move towards 1.

$$YU_{score-new} = YU_{score-old} + \Delta. \tag{6.5}$$

For CA rules, the score is decreased to move towards 0.

$$YU_{score-new} = YU_{score-old} - \Delta. \tag{6.6}$$

This computation weighs the confidence of the rule vs the ambiguity of the score. For a particular *uncertain* class, the new score is computed incorporating all relevant CP-CA

rules. Addition and/or subtraction of Δ might occur several times for each rule applied.

6.2.1.4 Multi-label classification

This is the final step. Once the changes are made by associative correlation, the new scores are recomputed from all the modifications that have been incorporated by the previous step. These scores are then converted to hard labels using a global threshold of 0.5. While predicting labels for unseen data, first the ML classifier predicts scores for the test data. From these scores, the *certain* and *uncertain* scores are separated. The CP-CA rules from the training set are used for the *certain* classes to improve the *uncertain* scores.

6.2.2 Experimental Analysis

Experimental analysis of the proposed work has been performed on ten standard ML datasets namely, Emotions (music), Scene (image), Flags (image), Yeast (biology), Image (image), CHD_49 (medicine), Yelp (web-text), Water quality (chemistry), Human_PseAAC (biology) and GpositivePseAAC (biology). Seven performance metrics [28] have been used for comparison, namely, Hamming loss (HL), ranking loss (RL), one error (OE), subset accuracy (SA), macro F1 (MacF1), micro F1 (MicF1) and accuracy (Acc). The results are aggregated from multiple runs of 5-fold cross-validation.

The proposed model has been applied to benchmark ML classifiers to include label correlations information alongside classification. Figures 6.2 to 6.4 compare the performance of three algorithms namely, MLP, MLKNN and MLRBF respectively with and without the application of the proposed FLMA approach. The results for three metrics, macro F1, micro F1 and accuracy have been shown in the form of a stacked column chart for all ten datasets to analyse the impact of the proposed method. The coloured portions separately indicate the performance of each method. From the results, it is seen that the performances of the FLMA improved algorithms are significantly better than their

stand-alone versions.

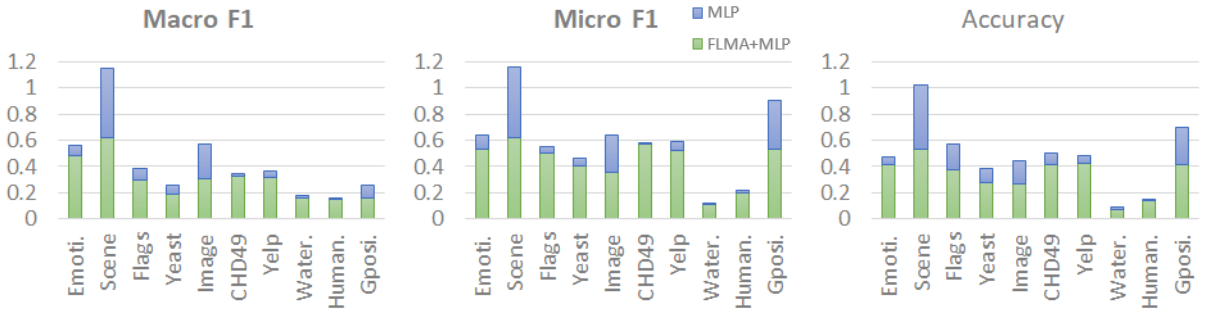


Figure 6.2: Improvement of MLP on application of FLMA

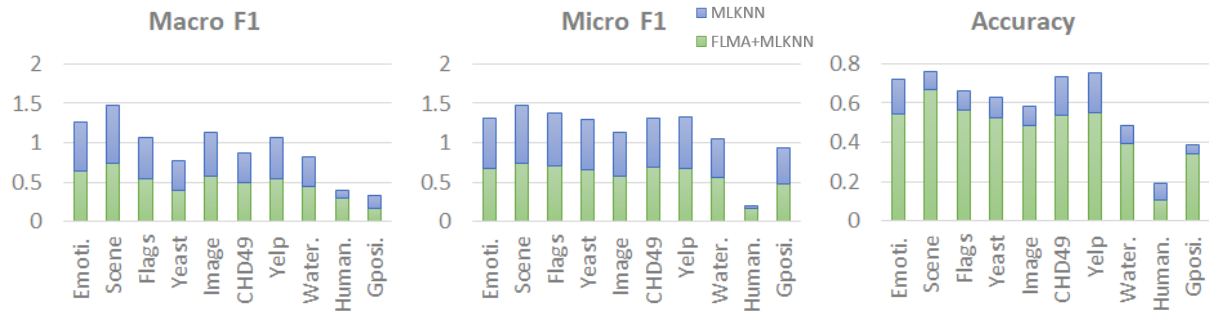


Figure 6.3: Improvement of ML-KNN on application of FLMA

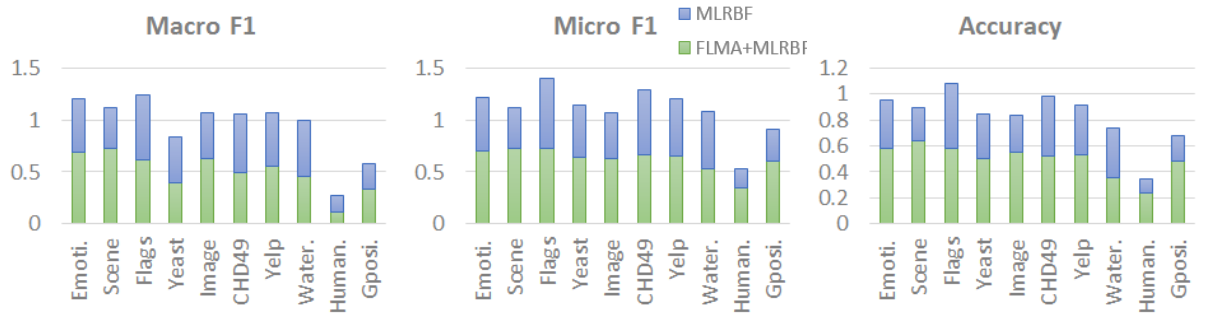


Figure 6.4: Improvement of ML-RBF on application of FLMA

Figure 6.5 shows the consolidated performance of the three FLMA improved algorithms for ten datasets and six metrics in the form of radar plots. The boundary covering a larger area on a radar plot depicts better performance. From the plots, it is seen that for all the six metrics, FLMA+MLKNN and FLMA+MLRBF perform very closely. However, the performance of FLMA+MLP is seen to falter for three metrics.

Table 6.1 to 6.10 compare the FLMA improved methods with four benchmark ML algorithms, namely RAKEL, CC, ECC and MLLEM. These are few well-known ML algo-

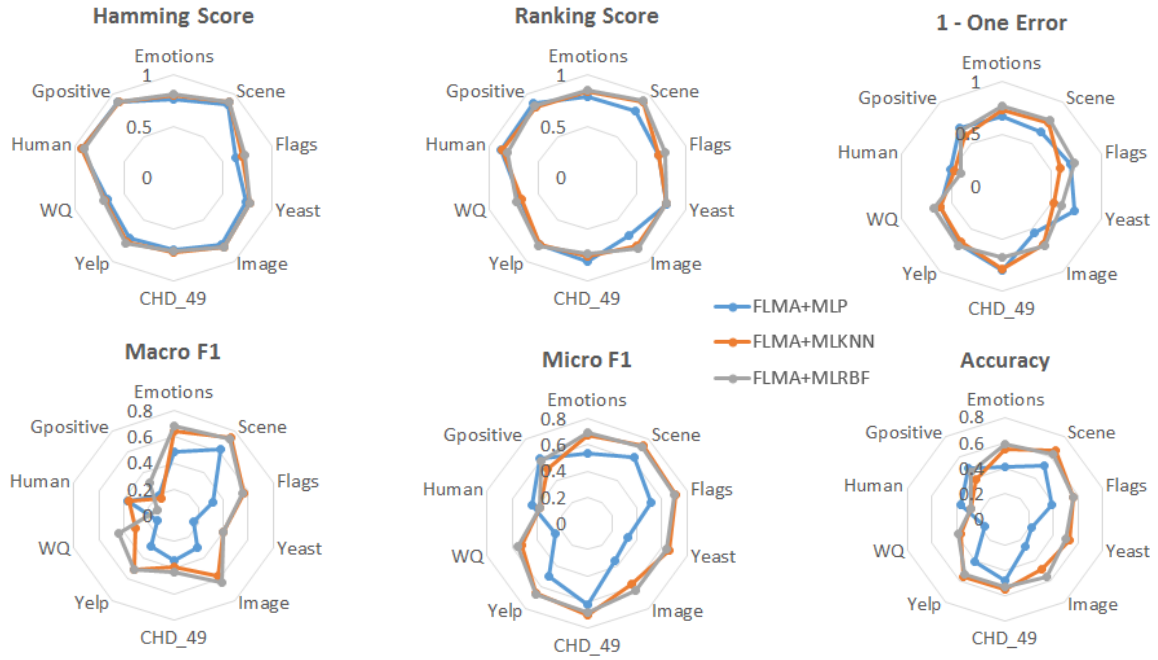


Figure 6.5: Comparison among the improved algorithms for ten datasets on six metrics

rithms that utilize label information during classification. This makes these relevant to the proposed work and are thus used for comparative analysis. From the results, at a glance, it is seen that the FLMA improved algorithms perform better than the other ML classifiers in most cases. Looking into specific performance metrics, it is seen that for the ranking-based one error metric, the outcome is lower than the other algorithms. However, the other ranking based metric, the ranking loss seems to achieve the best scores for the proposed approach. Subset accuracy is a strict metric, which considers only those results which have completely correct labels. The proposed FLMA is seen to improve the SA metric quite well.The other metrics like HL, MacF1, MicF1, Acc show improvement with the application of FLMA. Among the three improved algorithms, FLMA+MLRBF achieves a majority of the highest scores, followed by FLMA+MLKNN. Overall, for all the ten datasets, the proposed FLMA is seen to improve the performance of the standard ML classifiers by incorporating class correlations that might have been otherwise ignored.

To further analyse the data, T-test statistics have been computed for the proposed FLMA model. Among the three FLMA enhanced algorithms, MLRBF seems to be the best per-

Table 6.1: Comparison of methods for Emotions dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.2339	0.2116	0.3305	0.1871	0.4835	0.5333	0.4114
FLMA+MLKNN	0.1948	0.1615	0.2731	0.2969	0.6413	0.6717	0.5472
FLMA+MLRBF	**0.1832**	**0.1479**	0.2310	**0.3306**	**0.6843**	**0.6964**	**0.5843**
RAKEL	0.2369	0.6645	0.3052	0.2548	0.5686	0.6078	0.4837
CC	0.2206	0.4852	**0.1956**	0.2464	0.5616	0.5964	0.4620
ECC	0.2341	0.6290	0.2513	0.2649	0.5686	0.6149	0.4964
MLLEM	0.2712	0.7752	0.3793	0.1383	0.4231	0.4327	0.3573

Table 6.2: Comparison of methods for Scene dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.1107	0.1889	0.3436	0.5072	0.6187	0.6236	0.5308
FLMA+MLKNN	**0.0872**	0.0830	0.2414	**0.6249**	**0.7401**	**0.7338**	0.6665
FLMA+MLRBF	0.0886	**0.0750**	**0.2135**	0.5933	0.7276	0.7211	0.6353
RAKEL	0.0991	0.7701	0.2443	0.6068	0.7246	0.7176	**0.7008**
CC	0.1072	0.6537	0.2575	0.5526	0.6974	0.6898	0.6230
ECC	0.1062	0.7555	0.2202	0.5929	0.7135	0.7058	0.6685
MLLEM	0.1093	0.9035	0.2908	0.6052	0.6984	0.6838	0.6771

Table 6.3: Comparison of methods for Flags dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.3661	0.2413	0.2979	0.0617	0.2915	0.5035	0.3747
FLMA+MLKNN	0.2998	0.2527	0.3845	**0.1184**	0.5486	0.7004	0.5612
FLMA+MLRBF	**0.2710**	**0.2257**	0.2912	0.0876	**0.6107**	**0.7294**	**0.5836**
RAKEL	0.2945	0.6632	0.0781	0.1136	0.4551	0.6739	0.5304
CC	0.2932	0.5181	0.0382	0.0927	0.4834	0.6909	0.5409
ECC	0.2911	0.5672	**0.0363**	0.1134	0.4838	0.6908	0.5439
MLLEM	0.4685	0.7104	0.3073	0.0000	0.4095	0.4448	0.3211

Table 6.4: Comparison of methods for Yeast dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.2431	**0.1876**	0.2656	0.0372	0.1888	0.4021	0.2801
FLMA+MLKNN	0.2115	0.1909	0.4344	**0.1498**	0.3937	**0.6545**	**0.5241**
FLMA+MLRBF	0.2098	0.1925	0.3744	0.1351	**0.4003**	0.6458	0.5055
RAKEL	**0.1996**	0.6543	0.1489	0.1593	0.3644	0.6404	0.5106
CC	0.2017	0.5538	**0.1034**	0.1422	0.3664	0.6389	0.5073
ECC	0.2011	0.6371	0.1452	0.1447	0.3583	0.6376	0.5052
MLLEM	0.3027	0.6801	0.5006	0.0095	0.1716	0.1909	0.1338

Table 6.5: Comparison of methods for Image dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.2093	0.3126	0.4565	0.2369	0.3073	0.3549	0.2640
FLMA+MLKNN	0.1714	0.1814	0.3235	0.4015	0.5747	0.5768	0.4840
FLMA+MLRBF	**0.1637**	**0.1595**	**0.2720**	**0.4429**	**0.6284**	**0.6274**	0.5471
RAKEL	0.1789	0.6842	0.3155	0.4095	0.6254	0.6213	**0.5924**
CC	0.1829	0.5362	0.2903	0.3971	0.5952	0.5936	0.5108
ECC	0.1819	0.6316	0.2805	0.4034	0.6150	0.6112	0.5531
MLLEM	0.1975	0.8043	0.3745	0.4155	0.5616	0.5595	0.5393

Table 6.6: Comparison of methods for CHD_49 dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.3180	0.2590	0.3117	0.1006	0.3298	0.5669	0.4182
FLMA+MLKNN	0.3158	0.2255	0.2754	0.1061	0.4933	**0.6841**	**0.5354**
FLMA+MLRBF	**0.2941**	**0.2176**	0.2862	**0.1431**	**0.4969**	0.6676	0.5233
RAKEL	0.3084	0.6261	0.1606	0.0952	0.4861	0.6436	0.5047
CC	0.2925	0.5289	**0.0831**	0.1112	0.4902	0.6609	0.5213
ECC	0.2922	0.6072	0.0992	0.1112	0.4791	0.6633	0.5273
MLLEM	0.4399	0.7159	0.4152	0.0054	0.0000	0.0000	0.0000

Table 6.7: Comparison of methods for Yelp dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.2723	0.2063	0.3519	0.2309	0.3147	0.5229	0.4285
FLMA+MLKNN	0.2430	0.1892	0.3449	0.2362	0.5453	**0.6675**	**0.5527**
FLMA+MLRBF	0.2193	**0.1570**	0.2731	0.2962	**0.5569**	0.6554	0.5272
RAKEL	**0.1663**	0.7729	**0.1879**	**0.4555**	0.5455	0.6566	0.5471
CC	0.1805	0.6153	0.1432	0.3905	0.5211	0.6463	0.5221
ECC	0.1804	0.6476	0.1583	0.3882	0.5213	0.6460	0.5203
MLLEM	0.3275	0.6764	0.5555	0.0745	0.0095	0.0071	0.0032

Table 6.8: Comparison of methods for Water Quality dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.3526	0.3120	0.3434	0.0019	0.0624	0.1141	0.0704
FLMA+MLKNN	0.3149	0.3001	0.4924	**0.0152**	0.4382	**0.5649**	**0.3927**
FLMA+MLRBF	**0.3074**	**0.2668**	0.3824	0.0104	**0.4602**	0.5309	0.3569
RAKEL	0.3113	0.4955	0.2019	0.0145	0.3581	0.4599	0.2964
CC	0.3192	0.3486	**0.1009**	0.0149	0.3822	0.4939	0.3264
ECC	0.3101	0.4269	0.1726	0.0150	0.3767	0.4838	0.3187
MLLEM	0.4144	0.6751	0.4566	0.0047	0.1036	0.1058	0.0701

Table 6.9: Comparison of methods for Human_PseAAC dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	**0.0840**	0.1780	0.6135	0.1224	0.0456	0.2013	0.1347
FLMA+MLKNN	0.0870	0.1839	0.6668	0.0888	0.0299	0.1604	0.1041
FLMA+MLRBF	0.0913	**0.1779**	0.6017	**0.1674**	**0.1159**	**0.3431**	**0.2429**
RAKEL	0.0993	0.3396	0.4674	0.1606	0.0935	0.3245	0.2326
CC	0.0996	0.2389	**0.3267**	0.1484	0.0853	0.2811	0.2050
ECC	0.1027	0.3071	0.4282	0.1555	0.0936	0.3015	0.2377
MLLEM	0.1182	0.7521	0.7349	0.2135	0.1069	0.2425	0.2384

Table 6.10: Comparison of methods for GPositivePseAAC dataset

Method	*HL*	*RL*	*OE*	*SA*	*MacF1*	*MicF1*	*Acc*
FLMA+MLP	0.0926	0.1416	0.4030	0.4109	0.1557	0.5318	0.4184
FLMA+MLKNN	0.0979	0.1438	0.4303	0.3283	0.1662	0.4695	0.3384
FLMA+MLRBF	**0.0826**	**0.1106**	**0.3326**	**0.4669**	0.3316	**0.6035**	**0.4847**
RAKEL	0.0971	0.6898	0.3634	0.4297	0.3565	0.5905	0.4196
CC	0.1022	0.6059	0.3478	0.3928	0.3689	0.5513	0.3630
ECC	0.9979	0.0154	0.0134	0.0138	0.0080	0.0118	0.0134
MLLEM	0.0873	0.9068	0.3462	0.4501	**0.4109**	0.5887	0.4570

forming one, hence the comparisons have been made based on FLMA+MLRBF. From Table 6.11 it is seen that for $t_{.90} = 1.383$ with degrees of freedom = 9 the proposed method outperforms the others. Additionally, the non-parametric two-tailed Wilcoxon signed-rank test (Table 6.12) for the macro F1 metric indicates that with $\alpha = 0.20$, $T_{Wilcoxon}(10) = 14$, the proposed FLMA is statistically superior to all the other algorithms.

Table 6.11: T-Test for all methods against FLMA (based on Macro F1)

Method	**T-test value**
RAKEL	2.571410
CC	2.812371
ECC	1.656584
MLLEM	2.740044

Table 6.12: Two-tailed Wilcoxon signed-rank test statistics for all the methods vs proposed FLMA (based on accuracy)

Method	p **Value**	**Value of sign rank**
RAKEL	0.431641	36
CC	0.003906	54
ECC	0.130859	43
MLLEM	0.013672	51

6.2.2.1 Comparison with the previous proposed works

To have a comparative analysis of the algorithm proposed in this chapter with the previously proposed works, the performance can be compared based on the problem they are handling. The proposed FLMA is specifically dealing with the label dependency issue in ML data. The only other work in this thesis that aims to do a similar job is the previously proposed multi-label binary tree of classifiers (ML-BTC) model. The class dependencies were implicitly preserved in ML-BTC whereas in FLMA, the class associations are explicitly extracted and incorporated to improve a standard ML algorithm. In Table 6.13 the best version of proposed FLMA (FLMA+MLRBF) has been compared with ML-BTC for eight ML datasets. From the comparison it is seen that the FLMA improved algorithm performs better in most cases. Thus, the explicit approach of extracting class

Table 6.13: Comparison with the other proposed method

Dataset	Method	HL	RL	OE	SA	MicF1	MacF1	Acc
Emotions	FLMA	**0.1832**	**0.1479**	**0.2310**	0.3306	**0.6843**	**0.6964**	**0.5843**
	MLBTC	0.2126	0.2164	0.2639	**0.6384**	0.6601	0.6352	0.5518
Scene	FLMA	**0.0886**	**0.0750**	**0.2135**	0.5933	**0.7276**	0.7211	0.6353
	MLBTC	0.1291	0.1336	0.2658	**0.7291**	0.7153	**0.7248**	**0.7093**
Flags	FLMA	**0.2710**	**0.2257**	0.2912	0.0876	0.6107	**0.7294**	**0.5836**
	MLBTC	0.3438	0.3135	**0.2158**	**0.5922**	**0.6114**	0.5085	0.5194
Yeast	FLMA	**0.2098**	**0.1925**	0.3744	0.1351	0.4003	**0.6458**	**0.5055**
	MLBTC	0.2342	0.2671	**0.2891**	**0.5275**	**0.5605**	0.3443	0.4319
Image	FLMA	**0.1637**	**0.1595**	**0.2720**	0.4429	**0.6284**	**0.6274**	**0.5471**
	MLBTC	0.2189	0.2429	0.4008	**0.5729**	0.5713	0.5637	0.5241
CHD49	FLMA	**0.2941**	**0.2176**	**0.2862**	0.1431	0.4969	**0.6676**	**0.5233**
	MLBTC	0.3244	0.2659	0.3463	**0.6054**	**0.6398**	0.4338	0.4901
Yelp	FLMA	0.2193	0.1570	0.2731	0.2962	0.5569	**0.6554**	0.5272
	MLBTC	**0.2167**	**0.1461**	**0.1387**	**0.6436**	**0.6602**	0.5944	**0.5619**
WQ	FLMA	**0.3074**	**0.2668**	**0.3824**	0.0104	0.4602	**0.5309**	**0.3569**
	MLBTC	0.3436	0.3177	0.4039	**0.4796**	**0.4955**	0.4273	0.3431

dependencies in the data seems to be quite efficient in improving the overall classification performance.

6.3 Conclusion

In this chapter, the aim was to incorporate the label correlations that exist in multi-label data and are often not taken into consideration for classification. Here, a frequent label-set mining concept has been introduced, which identifies frequently co-present and co-absent labels in the data. This information is further used to improve the classification scores of instances that have been ambiguously predicted by existing classifiers. With the help of *certain* confident scores, these ambiguous *uncertain* scores have been tactfully improved. Using the proposed method in association with a few existing ML classifiers over ten ML datasets, it is seen that this additional class correlation information brings

substantial improvement to the performance.

The various models developed across the previous chapters are capable of handling the few problems identified for multi-label data. However, there is still a vast scope of research in this field that remains unexplored. A summary of the work done, limitations and scope of future work has been discussed in the following concluding chapter.

Chapter 7

Conclusion & Future Scope

7.1 Conclusions

This thesis is a comprehensive attempt to develop different multi-label models that are capable of handling the various bottlenecks that exist in this domain. Among the numerous problems that exist, a few have been considered and worked around to create models that can reduce the effect of these drawbacks on the multi-label classification results. The issues handled in this thesis were mainly regarding the large input dimension, complex decision space, class imbalance and label correlations of multi-label data. A subset of these problems was handled in each of the chapters. Here, the summary of the contributions along with their findings has been highlighted.

Chapter 3 proposed two novel models for multi-label classification that can handle the drawbacks of large input dimensions and complex output space in ML data. For the dimensionality reduction aspect, autoencoders have been used, whereas, for the fast learning of the complex output space and the feature to class mapping, multi-label extreme learning machines (MLELMs) have been used. ELM is a compact neural network that learns from the training data in one pass. Applications of this network have been made in various domains, but its use is limited in the field of multi-label classification. The

network is innovative and unique but it is not able to handle multi-label data efficiently on its own. To explore the possibilities in the area of multi-label classification, by utilizing the strengths and coping with the challenges faced by ELM, cascaded network models have been proposed here. The first work proposed a novel stacked autoencoder and extreme learning machine network for multi-label classification (MLSAEELM). It uses stacked autoencoders for feature encoding, MLELM for soft classification and a novel class score approximation method, which finally results in multi-label classification. Comparative analysis of the proposed algorithm on seven datasets with ten performance metrics against eleven other algorithms has shown encouraging performance improvement. The above model was further extended to create a deep multi-label classifier based on deep autoencoder (DAE) and multi-label extreme learning machines. DAE performed feature extraction, while the stacked MLELMs improved the learning of the classifier. It is an improvement on the first approach from a deep learning perspective. In-depth experimental analysis showed that increasing the depth of the model to some extent can improve classification performance in most cases. The proposed model also fares well compared to a few state-of-the-art ML classifiers. A comparison between SAE and DAE shows that DAE can extract better features from larger datasets, whereas SAE worked better for the smaller data. Thus, the individual components as well as the entire proposed model has proven to be quite efficient.

Chapter 4 was aimed to develop models that increase the separability of overlapping multi-label class boundaries to improve the classification. Alongside, dimensionality reduction and transformation of the large feature space has also been performed. The first work presented a multi-label functional link artificial neural network (MLFLANN) for the class prediction of multi-label datasets. The proposed MLFLANN model expands the input data to a higher dimension which helps to classify the multi-label data well. This type of data is quite complex and has overlapping class boundaries which make multi-label classification more challenging than single-label classification tasks. The network learns to map the functionally expanded features to the output nodes through weighted connec-

tions that are updated through some learning techniques. The literature for single-label FLANN shows the use of different basis functions and learning methods. To have a similar understanding of the configurations that work best for ML data, six configurations of MLFLANN had been evaluated to determine the optimal one. These models were created by pairing three functional expansion units – trigonometric, Chebychev and power polynomial, and two learning procedures – backpropagation and particle swarm optimization (PSO). All the combination models have been tested on some multi-label datasets to generate concluding results. It was found that few models that were successful in single-label classification, have not been able to handle multi-label classification equally well. Whereas, some techniques have surpassed the others in almost every scenario, making them suitable multi-label classifiers. Combinations of trigonometric function with backpropagation learning and power polynomial function with PSO learning have performed better than the other four models. Chebychev expansion does not increase separability for ML data, hence is not usable. Among backpropagation and PSO, the latter is computationally more expensive, especially if the population size is huge. While developing MLFLANN, although the functional expansion improved class separability it also increased the feature dimension multiple folds. This is contradicting for the large input dimension issue for multi-label data. Hence, a two-layer transformation network was proposed, which performs functional expansion followed by reduction-cum-transformation of features to handle the large input dimension as well as the complex decision space. It is an amalgamation of a multi-label functional link artificial neural network and an autoencoder that performs feature transformation in two stages. The first layer functionally expands the original input, thus introducing non-linearity to it, while the second layer performs feature extraction and transformation through autoencoders to improve separability while reducing the bulk of the previously expanded input dimension. The final network is more compatible to handle multi-label data while transforming it to a further separable space. This network combines the strengths of FLANN and AE to overcome the bottlenecks of multi-label classification. A single-label version of the pro-

posed model has also been built and evaluated. Experimental analysis of the proposed two-layer AE-MLFLANN has proven to surpass some standard classifiers. Overall, the proposed model experimentally establishes that the novel two-layer functional expansion and feature transformation is beneficial for both multi-label and single-label classification.

In Chapter 5, a binary tree of classifiers had been proposed that handles the problem of class imbalance, simplifies decision boundaries and preserves label correlation in the data. The unique binary tree structure is built in the training phase by approximately partitioning the data into two discrete chunks without performing an exhaustive search. This is achieved by a novel label-space partitioning approach that strives to preserve class dependencies implicitly. A suitable classifier is trained to learn the partitioned data. This label-space splitting technique also helps to simplify the complex decision boundaries by segregating them into compact groups. Moreover, the inherent problem of imbalanced classes in multi-label data has been looked after by broadly partitioning the data if possible, otherwise using different classifiers specifically suited for the unevenly split data at an internal node and some proposed parameters. Two appropriate parameters have been included to help the action-decision at every node of the tree. As the tree grows in the training phase, restrictions are imposed strategically to avoid redundant and unnecessary branches. Finally, the leaf nodes are capable of assigning the final labels to samples taking the class dependencies and imbalance into consideration. In this way, once the binary tree of classifiers is formed it can be successfully used for efficient multi-label classification. The experimental results for the proposed model in comparison to fourteen other state-of-the-art algorithms display improvement in the majority of the scenarios thus establishing the success of the proposed model.

Multi-label classifiers mostly disregard the label correlations that might exist in the data. However, to extract some information from these correlations, Chapter 6 proposes a frequent label-set mining technique that can be associated with any existing multi-label classifier to include the correlation information from the data. For multi-label data,

groups of co-occurring classes constitute frequent label-sets which indicates a correlation between these labels. A novel approach of frequent label-set mining for ML data has been proposed which generates co-present and co-absent rules. The proposed method then improves the classification scores from an existing ML classifier by incorporating these rules. The *certain* scores from the existing classifier indicate confident predictions that are used to select relevant rules. These rules are used to improve the *uncertain* and ambiguous scores which enhances the overall classification performance of the multi-label classifier. The results indicate substantial improvement in the application of the proposed technique with respect to the existing ML classifiers.

7.2 Findings

In the four contributory chapters, numerous ML models were built to achieve a good classification while aiming to handle the various problems of ML data focussed in this thesis. While exploring the field, developing the models and performing experiments there were various findings that are note-worthy.

- **Unpredictability -** The nature of ML data is not as predictable as single-label/binary data. Due to the overlapping classes, the algorithms sometimes underachieve than expected. The proposed methods have also behaved differently in certain cases than their expected performance.

- **Imbalance -** The class imbalance in the data is so severe that algorithms might not function as expected. It is better to handle the imbalance in the data before applying the proposed work to achieve unbiased performance.

- **Disproportion -** Some of the benchmark ML datasets have large number of features or classes compared to the number of available samples. That makes it quite difficult to train the algorithm, since the disproportionate sample to feature/class size often leads to issues of overfitting or underfitting. In such cases, for complex

algorithms, these unbalanced datasets were avoided.

Keeping these in mind, the next section states the limitations and future scope.

7.3 Limitations & Future Scope

Analyzing the proposed models individually and the thesis as whole, various limitations and scope of future exploration has been identified.

- **Parameter tuning** The models proposed in this thesis have certain parameters that are determined experimentally. Although the parameters have been broadly deduced from the data, fine-tuning is needed to improve its impact on the classification. The threshold parameters of H and C in the tree of classifier ML-BTC, bounding thresholds the region of certainty in FLMA, etc are broadly determined from the data. Although a slight change of parameters does not drastically affect the performance of the classifiers, future studies can attempt to adapt these parameters in a way that they are specifically tuned for the dataset at hand.

- **Network configurations** The proposed neural network models do not have any specific configuration regarding the number of layers in the network, or the number of hidden nodes in an intermediate layer. The MLFLANN model was evaluated for six configurations, however, it is not feasible to do the same for the larger networks. Thus, the size of the intermediate layer for autoencoders and ELMs, the number of functions for feature expansion in the MLFLANN, etc need to be specified. These have been experimentally determined and in future can be explored to find good configurations with respect to the feature dimension, sample size and the number of classes.

- **Remove class imbalance** The problem of class imbalance is quite prominent in multi-label data. This thesis explores one possibility of accommodating the classifier to suit the imbalanced data. However, future possibilities include tackling

the imbalance itself and removing its effect from the data before any classifier is applied. In such a scenario, the contributions in this thesis that do not consider class imbalance specifically might be able to achieve superior performance.

- **High output dimension and low sample size** The limitations of multi-label data include disproportionate feature, sample and class size. One problem of the three, i.e., the large feature dimension has been handled in the thesis using several autoencoders in different scenarios. However, if the output dimension is too large and/or the sample size low in comparison, the proposed classifiers may not train properly, leading to problems like overfitting or underfitting. In future, all the three aspects of the data, i.e., the number of features, samples and classes should be balanced with respect to each other such that the classifier is not biased in any way. The large output space needs to be reduced without losing out on the multi-label information and the sample size should be tackled without introducing redundancy.

- **Explore other models** In the limited scope of the thesis, it was not possible to explore all the interesting models that exist in literature. There is always a scope of improvement for the proposed works to take inspiration from models that are being developed currently and might achieve a better performance. Various networks like RVFL, etc that exist can be thoroughly explored in future.

- **Application of multi-label data** There were no specific application areas where the proposed algorithms have been applied. There is a huge scope for applying multi-label algorithms in a plethora of domains. The proposed methods can be modified to suit the application area and type of data being used. Domains like geoscience and remote sensing, sentiment analysis from social media posts, etc, still have a good scope of employing multi-label algorithms.

- **Other branches of multi-label learning** The focus of this thesis was towards general multi-label classification and handling specific drawbacks in the domain.

However, this field is now being thoroughly explored which has opened up multiple new avenues for research. One such branch is multi-instance multi-label learning where each data sample is described by multiple instances and its corresponding multiple labels. This type of representation helps to deal with more complex data which hold multiple semantic meanings.

The future scopes discussed above range from minor modifications to exploring new domains. This is because the field of multi-label learning is still being thoroughly explored by researchers. Although a significant amount of work has been done in this thesis, it still leaves a huge scope to explore the field of multi-label learning.

List of Publications

Journal Articles

a. **A. Law**. and A. Ghosh, Multi-label classification using a cascade of stacked autoencoder and extreme learning machines. *Neurocomputing*. 2019 Sep 17;358:222-34.

b. **A. Law**, B. Evani and A. Ghosh, Optimized functional link artificial neural network for multi-label classification. *Australian Journal of Intelligent Information Processing System (AJIIPS), Special Issue: Neural Information Processing, 26th International Conference on Neural Information Processing (ICONIP 2019), Sydney, Australia*. 2019;16(4):56-63.

c. **A. Law** and A. Ghosh, Multi-label classification using binary tree of classifiers. *IEEE Transactions on Emerging Topics in Computational Intelligence*. 2021;1-13.

Conference Papers

a. **A. Law**, K. Chakraborty and A. Ghosh, Functional link artificial neural network for multi-label classification. In *International Conference on Mining Intelligence and Knowledge Exploration* 2017 (pp. 1-10). Springer, Cham.

b. **A. Law**, R. Ray and A. Ghosh, Autoencoder and extreme learning machine based deep multi-label classifier. In *9th International Conference on Pattern Recognition*

and Machine Intelligence 2021. (accepted)

Bibliography

[1] R. C. Barros, M. P. Basgalupp, A. De Carvalho, and A. A. Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312, 2011.

[2] M. Bello, G. Nápoles, R. Sánchez, R. Bello, and K. Vanhoof. Deep neural network to extract high-level features and labels in multi-label classification problems. *Neurocomputing*, 413:259–270, 2020.

[3] A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *Journal of Machine Learning Research*, 18(1):2653–2688, 2017.

[4] E. Binaghi, P. A. Brivio, P. Ghezzi, and A. Rampini. A fuzzy set-based accuracy assessment of soft classification. *Pattern Recognition Letters*, 20(9):935–948, 1999.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[6] H. Blockeel, S. Džeroski, and J. Grbović. Simultaneous prediction of multiple chemical parameters of river water quality with TILDE. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 32–40. Springer, 1999.

[7] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene

classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[8] J. Cai, W. Sun, J. Guan, and I. You. Multi-ECGNet for ECG arrythmia multi-label classification. *IEEE Access*, 8:110848–110858, 2020.

[9] D. Chakraborty, V. Narayanan, and A. Ghosh. Integration of deep feature extraction and ensemble learning for outlier detection. *Pattern Recognition*, 89:161–171, 2019.

[10] V. Chauhan, A. Tiwari, and S. Arya. Multi-label classifier based on kernel random vector functional link network. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[11] A. Clare and R. King. Knowledge discovery in multi-label phenotype data. *Principles of Data Mining and Knowledge Discovery*, pages 42–53, 2001.

[12] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, 1965.

[13] F. De Comité, R. Gilleron, and M. Tommasi. Learning multi-label alternating decision trees from texts and data. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 35–49. Springer, 2003.

[14] S. Dehuri and S. B. Cho. A comprehensive survey on functional link neural networks and an adaptive PSO–BP learning for CFLNN. *Neural Computing and Applications*, 19(2):187–205, 2010.

[15] S. Dehuri and S. B. Cho. Evolutionarily optimized features in functional link neural network for classification. *Expert Systems with Applications*, 37(6):4379–4391, 2010.

[16] S. Dehuri and S. B. Cho. A hybrid genetic based functional link artificial neural network with a statistical comparison of classifiers over multiple datasets. *Neural Computing and Applications*, 19(2):317–328, 2010.

[17] S. Dehuri, R. Roy, S. B. Cho, and A. Ghosh. An improved swarm optimized functional link artificial neural network (ISO-FLANN) for classification. *Journal of Systems and Software*, 85(6):1333–1345, 2012.

[18] K. Dembczynski, W. Cheng, and E. Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *27th International Conference on Machine Learning (ICML)*, 2010.

[19] P. Duygulu, K. Barnard, J. F. G. de Freitas, and D. A. Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *European Conference on Computer Vision*, pages 97–112. Springer, 2002.

[20] R. Eberhart and J. Kennedy. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[21] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems*, pages 681–687, 2002.

[22] J. Fürnkranz, E. Hüllermeier, E. L. Mencía, and K. Brinker. Multilabel classification via calibrated label ranking. *Machine learning*, 73(2):133–153, 2008.

[23] E. C. Gonçalves, A. Plastino, and A. A. Freitas. A genetic algorithm for optimizing the label ordering in multi-label classifier chains. In *25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 469–476. IEEE, 2013.

[24] T. Gonçalves and P. Quaresma. A preliminary approach to the multilabel classification problem of portuguese juridical documents. In *Portuguese Conference on Artificial Intelligence*, pages 435–444. Springer, 2003.

[25] I. Guyon and A. Elisseeff. *An Introduction to Feature Extraction*, pages 1–25. Springer, Berlin, Heidelberg, 2006.

[26] J. Han, M. Kamber, and J. Pei. Data mining concepts and techniques third edition. *The Morgan Kaufmann Series in Data Management Systems*, 5(4):83–124, 2011.

[27] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice - Hall of India, New Delhi, 2008.

[28] F. Herrera, F. Charte, A. J. Rivera, and M. J. Del Jesus. *Multilabel Classification: Problem Analysis, Metrics and Techniques.* Springer, 2016.

[29] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[30] G. B. Huang. What are extreme learning machines? Filling the gap between frank rosenblatt's dream and john von neumann's puzzle. *Cognitive Computation*, 7(3):263–278, 2015.

[31] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks*, volume 2, pages 985–990. IEEE, 2004.

[32] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[33] S. J. Huang and Z. H. Zhou. Multi-label learning by exploiting label correlations locally. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[34] Y. Huang, W. Wang, L. Wang, and T. Tan. Multi-task deep neural network for multi-label learning. In *2013 IEEE International Conference on Image Processing*, pages 2897–2900. IEEE, 2013.

[35] E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16):1897–1916, 2008.

[36] M. Jiang, Z. Pan, and N. Li. Multi-label text categorization using L21-norm minimization extreme learning machine. *Neurocomputing*, 261:4–10, 2017.

[37] L. L. C. Kasun, H. Zhou, G. B. Huang, and C. M. Vong. Representational learning with ELMs for big data. *IEEE Intelligent Systems*, 28(6):31–34, 2013.

[38] I. Katakis, G. Tsoumakas, and I. Vlahavas. Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD*, volume 18, page 5. Citeseer, 2008.

[39] P. Kaur and A. Gosain. Robust hybrid data-level sampling approach to handle imbalanced data during classification. *Soft Computing*, 24(20):15715–15732, 2020.

[40] P. Kaur and A. Gosain. GT2FS-SMOTE: An intelligent oversampling approach based upon general type-2 fuzzy sets to detect web spam. *Arabian Journal for Science and Engineering*, 46(4):3033–3050, 2021.

[41] K. Kimura, M. Kudo, and L. Sun. Simultaneous nonlinear label-instance embedding for multi-label classification. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 15–25. Springer, 2016.

[42] G. Klir and B. Yuan. Basic concepts and history of fuzzy set theory and fuzzy logic. In *Handbook of Fuzzy Computation*, pages A1–1. CRC Press, 2020.

[43] Y. Kongsorot and P. Horata. Multi-label classification with extreme learning machine. In *6th International Conference on Knowledge and Smart Technology (KST)*, pages 81–86. IEEE, 2014.

[44] C. Y. Liou, W. C. Cheng, J. W. Liou, and D. R. Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 2014.

[45] C. Y. Liou, J. C. Huang, and W. C. Yang. Modeling word perception using the Elman network. *Neurocomputing*, 71(16-18):3150–3157, 2008.

[46] O. Luaces, J. Díez, J. Barranquero, J. J. del Coz, and A. Bahamonde. Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, 1(4):303–313, 2012.

[47] E. L. Mencia and J. Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 50–65. Springer, 2008.

[48] B. B. Misra and S. Dehuri. Functional link artificial neural network for classification task in data mining. *Journal of Computer Science*, 3:948–955, 2007.

[49] A. Mondal, S. Ghosh, and A. Ghosh. Partially camouflaged object tracking using modified probabilistic neural network and fuzzy energy based active contour. *International Journal of Computer Vision*, 122(1):116–148, 2017.

[50] J. M. Moyano, E. L. Gibaja, K. J. Cios, and S. Ventura. Tree-shaped ensemble of multi-label classifiers using grammar-guided genetic programming. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[51] P. Niu, Y. Ma, M. Li, S. Yan, and G. Li. A kind of parameters self-adjusting extreme learning machine. *Neural Processing Letters*, 44(3):813–830, 2016.

[52] Y. Pao. *Adaptive Pattern Recognition and Neural Networks*. Reading, MA (US); Addison-Wesley Publishing Co., Inc., 1989.

[53] Y. H Pao, G. H. Park, and D. J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180, 1994.

[54] Y. H. Pao, S. M. Phillips, and D. J. Sobajic. Neural-net computing and the intelligent control of systems. *International Journal of Control*, 56(2):263–289, 1992.

[55] R. M. Pereira, Y. M. G. Costa, and C. N. Silla Jr. MLTL: A multi-label approach for the Tomek link undersampling algorithm. *Neurocomputing*, 383:95–105, 2020.

[56] F. Qin, X. J. Tang, and Z. K. Cheng. Application of apriori algorithm in multi label

classification. In *2013 International Conference on Computational and Information Sciences*, pages 717–720. IEEE, 2013.

[57] C. R. Rao and S. K. Mitra. Generalized inverse of a matrix and its applications. In *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Theory of Statistics*, pages 601–620. The Regents of the University of California, 1972.

[58] N. Rastin, Mansoor Z. Jahromi, and M. Taheri. A generalized weighted distance k-nearest neighbor for multi-label problems. *Pattern Recognition*, 114:107526, 2021.

[59] J. Read. *Scalable Multi-label Classification.* PhD thesis, University of Waikato, 2010.

[60] J. Read, B. Pfahringer, and G. Holmes. Multi-label classification using ensembles of pruned sets. In *Eighth IEEE International Conference on Data Mining (ICDM'08)*, pages 995–1000. IEEE, 2008.

[61] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 254–269. Springer, 2009.

[62] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[63] A. Rivolli, J. Read, C. Soares, B. Pfahringer, and A. C. de Carvalho. An empirical analysis of binary transformation strategies and base algorithms for multi-label learning. *Machine Learning*, 109(8):1509–1563, 2020.

[64] H. J. Rong, G. B. Huang, N. Sundararajan, and P. Saratchandran. Online sequential fuzzy extreme learning machine for function approximation and classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(4):1067–1072, 2009.

[65] H. J. Rong, Y. X. Jia, and G. S. Zhao. Aircraft recognition using modular extreme learning machine. *Neurocomputing*, 128:166–174, 2014.

[66] H. J. Rong, Y. S. Ong, A. H. Tan, and Z. Zhu. A fast pruned-extreme learning machine for classification problem. *Neurocomputing*, 72(1-3):359–366, 2008.

[67] H. Sajnani, V. Saini, K. Kumar, E. Gabrielova, P. Choudary, and C. Lopes, 2013.

[68] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[69] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine learning*, 39:135–168, 2000.

[70] D. Serre. *Matrices: Theory and Applications*. Springer-Verlag New York, 2010.

[71] H. Shao, G. Z. Li, G. P. Liu, and Y. Q. Wang. Symptom selection for multi-label data of inquiry diagnosis in traditional Chinese medicine. *Science China Information Sciences*, 56(5):1–13, 2013.

[72] B. N. Subudhi, S. Ghosh, and A. Ghosh. Application of Gibbs–Markov random field and Hopfield-type neural networks for detecting moving objects from video sequences captured by static camera. *Soft Computing*, 19(10):2769–2781, 2015.

[73] X. Sun, J. Wang, C. Jiang, J. Xu, J. Feng, S. S. Chen, and F. He. ELM-ML: Study on multi-label classification using extreme learning machine. In *Proceedings of ELM-2015*, volume 2, pages 107–116. Springer, 2016.

[74] X. Sun, J. Xu, C. Jiang, J. Feng, S. S. Chen, and F. He. Extreme learning machine for multi-label classification. *Entropy*, 18(6):225, 2016.

[75] F. A. Thabtah, P. Cowling, and Y. Peng. MMAC: A new multi-class, multi-label associative classification approach. In *4th IEEE International Conference on Data Mining (ICDM'04)*, pages 217–224. IEEE, 2004.

[76] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas. Multi-label classification

of music into emotions. In *9th International Conference on Music Information Retrieval (ISMIR)*, volume 8, pages 325–330, 2008.

[77] G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proceeding of ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, volume 21, pages 53–59, 2008.

[78] G. Tsoumakas, I. Katakis, and I. Vlahavas. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089, 2010.

[79] G. Tsoumakas and I. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *European Conference on Machine Learning*, pages 406–417. Springer, 2007.

[80] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):467–476, 2008.

[81] R. Venkatesan and M. J. Er. Multi-label classification method based on extreme learning machines. In *13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 619–624. IEEE, 2014.

[82] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, December 2010.

[83] J. Wang, J. Feng, X. Sun, S. S. Chen, and B. Chen. Simplified constraints Rank-SVM for multi-label classification. In *Chinese Conference on Pattern Recognition*, pages 229–236. Springer, 2014.

[84] R. Wang, S. Kwong, X. Wang, and Y. Jia. Active k-labelsets ensemble for multi-label classification. *Pattern Recognition*, 109:107583, 2021.

[85] Y. Wang, D. He, F. Li, X. Long, Z. Zhou, J. Ma, and S. Wen. Multi-label classification with label graph superimposing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12265–12272, 2020.

[86] Y. Wang, L. Wang, Q. Chang, and C. Yang. Effects of direct input–output connections on multilayer perceptron neural networks for time series prediction. *Soft Computing*, 24(7):4729–4738, 2020.

[87] Y. Wang, L. Wang, F. Yang, W. Di, and Q. Chang. Advantages of direct input-to-output connections in neural networks: The elman network for stock index forecasting. *Information Sciences*, 547:1066–1079, 2021.

[88] J. Wehrmann, R. Cerri, and R. Barros. Hierarchical multi-label classification networks. In *International Conference on Machine Learning*, pages 5075–5084. PMLR, 2018.

[89] W. Weng, D. H. Wang, C. L. Chen, J. Wen, and S. X. Wu. Label specific features-based classifier chains for multi-label classification. *IEEE Access*, 8:51265–51275, 2020.

[90] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[91] F. Wu, Z. Wang, Z. Zhang, Y. Yang, J. Luo, W. Zhu, and Y. Zhuang. Weakly semi-supervised deep learning for multi-label image annotation. *IEEE Transactions on Big Data*, 1(3):109–122, 2015.

[92] G. Wu, R. Zheng, Y. Tian, and D. Liu. Joint ranking SVM and binary relevance with robust low-rank learning for multi-label classification. *Neural Networks*, 122:24–39, 2020.

[93] Q. Wu, M. Tan, H. Song, J. Chen, and M. K. Ng. ML-FOREST: A multi-label tree

ensemble method for multi-label classification. *IEEE Transactions on Knowledge and Data Engineering*, 28(10):2665–2680, 2016.

[94] Q. Wu, Y. Ye, H. Zhang, T. W. S. Chow, and S. S. Ho. ML-Tree: a tree-structure-based approach to multilabel learning. *IEEE Transactions on Neural Networks and Learning Systems*, 26(3):430–443, 2015.

[95] J. Xin, G. Chen, and Y. Hai. A particle swarm optimizer with multi-stage linearly-decreasing inertia weight. In *International Joint Conference on Computational Sciences and Optimization*, volume 1, pages 505–508. IEEE, 2009.

[96] J. Xu, J. Liu, J. Yin, and C. Sun. A multi-label feature extraction algorithm via maximizing feature variance and feature-label dependence simultaneously. *Knowledge-Based Systems*, 98:172–184, 2016.

[97] E. K. Yapp, X. Li, W. F. Lu, and P. S. Tan. Comparison of base classifiers for multi-label learning. *Neurocomputing*, 394:51–60, 2020.

[98] J. Zabalza, J. Ren, J. Zheng, H. Zhao, C. Qing, Z. Yang, P. Du, and S. Marshall. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185:1–10, 2016.

[99] B. Zang, R. Huang, L. Wang, J. Chen, F. Tian, and X. Wei. An improved KNN algorithm based on minority class distribution for imbalanced dataset. In *2016 International Computer Symposium (ICS)*, pages 696–700. IEEE, 2016.

[100] M. L. Zhang. ML-RBF: RBF neural networks for multi-label learning. *Neural Processing Letters*, 29(2):61–74, 2009.

[101] M. L Zhang, Y. K. Li, X. Y. Liu, and X. Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12(2):191–202, 2018.

[102] M. L. Zhang and L. Wu. LIFT: Multi-label learning with label-specific features.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(1):107–120, 2014.

[103] M. L. Zhang and Z. H. Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.

[104] M. L. Zhang and Z. H. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

[105] N. Zhang, S. Ding, and J. Zhang. Multi layer ELM-RBF for multi-label learning. *Applied Soft Computing*, 43:535–545, 2016.

[106] H. Zhou, G. B. Huang, Z. Lin, H. Wang, and Y. C. Soh. Stacked extreme learning machines. *IEEE Transactions on Cybernetics*, 45(9):2013–2025, 2015.