A MULTINOMIAL PROTOTYPE-BASED LEARNING ALGORITHM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET CAN BULUT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2014

Approval of the thesis:

A MULTINOMIAL PROTOTYPE-BASED LEARNING ALGORITHM

submitted by **AHMET CAN BULUT** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

ABSTRACT

A MULTINOMIAL PROTOTYPE-BASED LEARNING ALGORITHM

Bulut, Ahmet Can
M.S., Department of Computer Engineering
Supervisor : Assist. Prof. Dr. Sinan Kalkan

September 2014, 105 pages

Recent studies in machine learning field proved that ideas which were once thought impractical are in fact tangible. Over the years, researchers have managed to develop learning systems which are able to interact with the environment and use experiences for adaptation to new conditions. Humanoid robots can now learn concepts such as nouns, adjectives and verbs, which is a big step for building human-like learners. Behind all these achievements, development of successful learning and classification techniques is one of the key factors. In this thesis, we propose a novel prototype-based learning method which uses the distributional properties of class dimensions. By dealing with the problem of feature dimensions' having multiple polarities, our algorithm can distinguish the dimensions which display unpredictable behaviors from the ones which are composed of multiple predictable patterns. We tested our algorithm on 8 different datasets and compared the results with 9 other algorithms including SVM and AdaBoost. Apart from being insensitive to the ordering of inputs, our method showed that it provides comparable performance in terms of accuracy rate, running time, learning curve and most importantly the ability to resolve multipolarity in dimensions.

Keywords: Machine Learning, Prototype-based Learning, Concept Learning, Cognitive Robotics

ÖZ

ÇOK KUTUPLU BOYUTLAR İÇEREN UZAYLARDA PROTOTİP TABANLI ÖĞRENME

Bulut, Ahmet Can
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Sinan Kalkan

Eylül 2014 , 105 sayfa

Günümüzde, bir zamanlar ulaşılamaz sayılan fikirlerin aslında gerçeklenebilir olduğu bilgisayar bilimindeki gelişmeler sayesinde kanıtlanmıştır. Bilim adamları, çevreyle etkileşim kurabilen, kazandığı tecrübelerden yararlanarak yeni koşullara adapte olabilen öğrenen sistemler geliştirmeyi başarmışlardır. İsim, fiil, sıfat gibi kavramları öğrenebilen insansı robotlar artık programlanabilmektedir. Bütün bu gelişmelerin arkasında, başarılı öğrenme ve sınıflandırma tekniklerinin geliştirilmesi yatmaktadır. Bu tezde, boyutların dağılımsal özelliklerinden faydalanan, prototip tabanlı yeni bir öğrenen sistem sunuyoruz. Prototiplerde görülen "çok kutuplu boyut" problemini çözerek, düzensiz davranışlar sergileyen boyutlarla, birbirinden ayrı istikrarlı desenler içeren boyutların ayrımına varabiliyoruz. Testlerimizde, 8 ayrı veri grubu kullandık ve algoritmamızı aralarında SVM ve AdaBoost gibi algoritmaların bulunduğu 9 ayrı algoritmayla, her bir veri grubu üzerinde karşılaştırdık. Girdilerin sunum sırasına hassasiyet göstermemekle beraber, MNPBL ismini verdiğimiz öğrenme algoritması, isabet oranı, koşu süresi, öğrenme eğrisi ve en önemlisi çok kutuplu boyutları çözümleme alanlarında etkili performans göstermiştir.

Anahtar Kelimeler: Makine Öğrenimi, Yapay Öğrenme, Prototip Tabanlı Öğrenme, Kavram Öğrenme, Robotik

To Mom and Dad

ACKNOWLEDGMENTS

First and foremost, my utmost and sincere gratitude goes to my advisor, Assist. Prof. Dr. Sinan Kalkan, whose encouragement, guidance and continuous support helped me through out this thesis study. It has been a pleasure and privilege working with such a brilliant researcher, great mentor and genuine person who was there for me whenever I needed. Without his immense knowledge, generous efforts and endless patience, completion of this thesis would not be possible. For both his professional and personal contributions, he deserves the best wishes.

An exceptionally talented person, the founder and head of KOVAN Research Lab and an outstanding researcher whom I have always admired, I would like to give my special thanks to Assoc. Prof. Dr. Erol Şahin not only for his contributions to my undergraduate and graduate studies but also introducing our department and university to the entire world with his exceptional work.

I cannot think of this department without Prof. Dr. Göktürk Üçoluk, who is an indispensable member of CENG family. Since I was an undergraduate junior, with his broadest experience and valuable guidance, he showed me and hundreds of other graduates the right path to follow. I would like to offer my gratitude to Göktürk Üçoluk for being one of the people who made me love my profession.

In addition to Erol Şahin and Göktürk Üçoluk, I would also like to thank my other committee members, Prof. Dr. Fatoş Yarman Vural and Assist. Prof. Dr. Aykut Erdem who have modestly agreed to participate in my thesis committee. I am grateful for their invaluable comments, suggestions and contributions to my thesis. Presence and approval of such important and reputable researchers made my thesis study much more meaningful.

I would like to thank Hande Çelikkanat, Güner Orhan and the other members of KOVAN Research Lab for their warm welcome and contributions. They have never turned their back and always offered their support whenever I needed.

My colleagues Dr. Doruk Bozağaç and Dr. Ahmet Kara have always supported me with their experience, guidance and patience during my graduate studies. I owe them a big "thank you" for helping me clear the question marks in my mind, whenever I have some popped up. It has been comforting to know that somebody is there for me, if I need an advice on almost anything. Their tolerant and always supportive attitude towards my graduate studies made TÜBİTAK İLTAREN a better and easier place to work at.

Can a person know everything, well, I know someone who can. Professionally, Zahir Tezcan can be defined by just two words: modest and genius; personally he is even more. Not only me but a lot of people have him in their thank you lists and I am one of these lucky people to have such a friend and colleague.

Dr. Hakkı Bağcı, Hakan Bağcı and Sercan Gök have always been very supportive since my day one at TUBİTAK İLTAREN. I've learned a lot from them, about engineering, about academics, about life. I want to thank them for making my cubicle feel like home and for all the patience with the coffee breaks at which they had to hear about my thesis work. Without their genuine support, I think it would be harder for me to be where I am now. I also would like to thank my colleague and classmate Celal Çığır for being very generous in offering his help whenever I needed during my graduate studies.

I need to mention Hüseyin Dirican who has been more than a colleague for me. It means a lot to have such a person that you can share both good times and hard times. If I need a "man to man" moment for an escape from life or if I need a vacation partner, it's him. As well as personal issues, during my thesis studies, he never hesitated to offer his endless support as he always do. I would like to thank him for being my stubborn but caring brother. Spending years in the very same department of METU without knowing each other, we only managed to meet after graduation but still, it feels like we have never missed those years. I thank him for making me feel that we have been friends since forever.

İsmet Özgür Çolpan, my roommate, my colleague, my good old friend. We fight a lot and we have fun a lot when we are together and we have been together since my day one at METU. We met in Preparatory School and to be honest I didn't think we could be close with this arabesque boy. I didn't know how it happened but then we shared undergraduate and graduate classes, dorm room, gyms, work, apartment. We have been sharing a lot for a long time. All these years, whenever I had trouble, I knew he would help me and he never let me down. During my thesis work, it was not an exception and he stood by me once again. Therefore, I owe him a big thank you for being one of the "classics" in my life.

In a photograph taken in 1990s, we were two little kids, posing, holding hands in a school ceremony. I grew up with my cousin Emre Kaş and we had all those amazing years sharing everything, watching out for each other in case one of us falls down. Despite the distances between us, we never got separated. I want to thank him for supporting me in every step of my life since childhood and never letting me walk alone.

Finally, some feelings can not be expressed by words, they can only be felt. Even if I try to do my best, writing and writing all day long, I can not thank enough to my mom and dad. Yet, I don't know where to start but long story short, they are the

reason of every breath I take, every single heartbeat in my chest, my existence and everything. Without them, without their infinite love, without being raised in such a great family, none of my achievements would have existed, including this thesis. Mom, dad, I know you are proud and you know I will keep on doing everything I can for making you feel that way. This thesis is devoted to you...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LVQ	Linear Vector Quantization
SVM	Support Vector Machine
RBF	Radial Basis Function
DAGSVM	Directed Acyclic Graph Support Vector Machine
NG	Neural Gas Algorithm
GNG	Growing Neural Gas Algorithm
RGNG	Robust Growing Neural Gas Algorithm
GMM	Gaussian Mixture Model
ANN	Artificial Neural Networks
BMU	Best Matching Unit
MDL	Minimum Description Length
AIC	Akaike Information Criterion
PDF	Probability Distribution Function
DEP	Dataset of Effect Prototypes
DEPB	Dataset of Effect Prototypes with Bipolarity
RDEPB	Reduced Dataset of Effect Prototypes with Bipolarity
STD	Short-Tall Dataset
DMP	Synthetic Dataset with High Multipolarity
WD	Wine Dataset
SEED	Seeds Dataset
DOP	Derivation of Prototypes
MNPBL	Multinomial Prototype-based Learning
k-NN	k-Nearest Neighbors
AdaBoost	Adaptive Boosting

CHAPTER 1

INTRODUCTION

1.1 Motivation

Ever since the computer era began, human being has been wondering whether machines could be made to learn. If we could manage to program computers to learn and give them the capability to improve automatically as they gain experience, the impact would be dramatic. Let us imagine computers learning to find out which treatments are most effective for new diseases given the relevant medical records, houses learning from experience to optimize energy cost depending on particular patterns of usage belonging to their occupants, personal assistants updating themselves for the changing needs of their users and robots which are able to perform novel set of commands given by their masters [29].Although we do not know how to make machines learn as well as humans do, algorithms which are effective for certain types of learning tasks have been proposed. A theoretical understanding of how learning has been emerged and continue to emerge [29].

There exists successful studies which highlight machine learning as one of the most important branches in computer science and prove that things which are once, thought to be impossible are in fact achievable.To illustrate, over the years, researchers have been managed to teach robots how to interact with the environment, act and get feedback, use the experience they gain for adapting to new conditions. Humanoid robots are now able to learn certain concepts including noun, adjective, verb concepts and making them earn these abilities is one big step for developing human-like robots.

Today, many machine learning applications, including cognitive robotics algorithms require large datasets for building learning systems which can improve performance by gaining experience. To be able to make successful generalization over experiences via machine learning methods, solving classification problems is an important issue and there exists different classification techniques such as decision tree classifiers, artificial neural network classifiers, support vector machine classifiers and prototype based classifiers. In this thesis, we focus on prototype-based learning and propose an efficient and successful method which overcomes the "multipolar dimensions" sensitivity of a prototype derivation method developed by Kalkan et al. [21].

1.2 Scope

Kalkan et al. already proved in their study [21] that such a method displays successful classification performance in the field of cognitive robotics. They used this method to classify verb concepts using the data collected from iCub humanoid robot. Each class represents a verb concept and class dimension information is one crucial part of the algorithm.

The method uses prototype labels for each feature dimension of a class. A prototype label defines the behavior of a feature in terms of its distributional characteristics. The problem arises when a feature dimension displays a behavioral pattern which is a combination of multiple consistent patterns. In such cases, the feature data shows non-linear behavior. We call it "multipolarity" in dimensions where "bipolarity" is the special case that there exist two peaks in the data. By the proposed method in the study [21], multipolar in other words multinomial dimensions are perceived as unpredictable patterns which is negligible information for class definition. However, multipolar dimensions can carry important information which can affect classification accuracy directly depending on the training data worked on.

In this thesis, we propose a robust prototype based learning algorithm which can eliminate this "multipolarity" problem of class dimensions and achieve promising performances on different types of data. We test the performance of our algorithm in terms of accuracy, precision, recall, generalization and robustness by comparing it with several other algorithms including:

- Linear Vector Quantization (LVQ), which is a well-known prototype based method,
- Support Vector Machine (SVM), which is a well-known binary classifier,
- Multi-SVM, which is an SVM method with the capability of classifying more-than two classes, using different kernel functions.
  - Linear kernel
  - Quadratic kernel
  - Polynomial kernel
  - Radial Basis Function (RBF) kernel
- Adaptive Boosting (AdaBoost), an ensemble learning method.
- Decision Tree Learning, which represents target functions as decision trees.
- k-Nearest Neighbors (k-NN) classifier which is an instance-based method.
- The prototype based method presented and used in the study [21], by Kalkan and his colleagues.

We include several datasets with different levels of multipolarity, different sizes in terms of instance count, class count and dimension count. This way, we evaluate the robustness of our algorithm, with the changing multipolarity levels in class dimensions.

1.3 Organization

The rest of the thesis is organized as follows. Chapter 2 presents background information for a brief understanding of on the notions used in this study, such as machine learning, categorization, concept learning, prototype learning and distance measurement. In Chapter 3, we summarize some of the studies which are related to ours. In Chapter 4, we propose our algorithm and explain the details of it while presenting the experimental results, comparisons and evaluations in Chapter 5. Here, we also include a case study to enhance the understanding of our contribution with this thesis. Finally, we conclude our study with Chapter 6. Detailed experimental results are presented in Chapter A.

CHAPTER 2

BACKGROUND

2.1 Machine Learning

Machine learning answers the question of how to develop computer programs that improve their performance on a given task, by gaining experience [29]. These learning systems are able to learn programs using provided data. Lately machine learning usage is spread throughout and even beyond computer science field. It is used in a lot of applications like spam filters, web search engines, recommender systems, stock trading, ad placement, drug design and robotics [7].

The fundamental aim of machine learning is making generalization over the given training set and going beyond it. Because, most of the time, it is unlikely to catch the same examples over and over again. Data is very important for a learner but there has to be more than that. Only having training data is never enough without further knowledge or assumptions. Luckily, in real world cases, we can achieve providing the learner further information beyond data since the functions to be learned are not gathered uniformly from the complete set of mathematically possible functions. Rather, they expose smoothness and are limited to certain dependencies helping us make assumptions for our goals.

There are two main types of machine learning which are supervised learning and unsupervised learning. For supervised learning, the learner gets supervision from outside while learning; however, in unsupervised learning no outsider help is involved.

2.2 Categorization

Humans apply prior knowledge by considering a new situation as an instance of a previous situation. This process is achieved by categorization. As introduced in "Principles of Categorization" [37] by Rosch, categorization comes with two general and basic principles. The first one is that a category system has to present a function which provides maximum information with least cognitive effort. This phenomenon, which is also called "cognitive economy", saves space by considering the category of objects rather than considering all objects one by one[47]. The second principle is that the world that is perceived by the category system has to be in a structured form and should not present arbitrary or unpredictable features. To be able to fulfill these requirements, categories should map the structure of the perceived world as closely as possible.

One of the most important functions of categorization is classification and a classifier is a system which takes a set of vectors containing features and gives a discrete value, in other words a class. Testing of a classifier is achieved by checking whether it gives the correct class as an output for a given sample input vector or not [4].

The concept of similarity makes a base for most of the approaches in pattern classification which are considered the simplest and the most intuitive among all [5, 18]. It is defined between items which are to be compared and give opinion about how much they overlap. According to this concept, the patterns which are similar, in certain ways, are assigned to the same class. Prototype learning is one of the major learning techniques and classification is achieved based on similarity [7].

The concept of prototype was defined by Eleanor Rosch in her study as the most central member of a category [38]. Prototypes abstract out the central tendency from the experienced examples, and then use it as a basis for categorization decisions [26].

2.3 Concept Learning

As it is stated by Mitchell [29], learning involves problems like generalizing functions from specific training examples. In the presence of negative and positive examples of the category and a sample derived from them, concept learning aims to acquire the definition of a general category. In other words, concept learning is the process of searching for the hypothesis which best fits the training examples in a predefined potential hypothesis space.

2.3.1 Illustration

As an illustration of concept learning task, learning the concept "the days on which the person Y enjoys a sport activity Z" can be defined with a representation of day containing attributes like sky state, air temperature, wind state, water state and weather forecast. In addition to those, a final attribute "EnjoySport" indicates whether or not the person Y enjoys the sport activity Z on a day with corresponding attributes which are described in Table 2.1. The learning task in such a setup is for learning to predict the value of "EnjoySport" in presence of its other attributes. To achieve this task, a hypothesis representation should be provided to the learner and each hypothesis would consist of a conjunction of constraints on the instance attributes. For each attribute, the hypothesis would indicate one of the following:

- "?" : Any value is acceptable for this attribute.
- A single required value (e.g. Sunny).
- "0" : no value is acceptable.

Table 2.1: New instance to be classified (adapted from [29])

Instance	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
A	Sunny	Warm	Normal	Strong	Cool	Change	Yes
B	Rainy	Cold	Normal	Light	Warm	Same	No
C	Rainy	Warm	Normal	Light	Warm	Same	?
D	Rainy	Cold	Normal	Strong	Warm	Same	?

Hypothesis h classifies an instance x as a positive example $h(x) = 1$ if it can manage to satisfy all of the constraints of h, given a training set X, where $x \in X$.

2.3.2 Views on Concept Learning

There are three main views to learn and represent concepts [14, 25, 38, 39]. The first one is the rule based view [2] in which categories are defined with strict boundaries. In this view, there are only two possible decisions about an exemplar belonging to a category, it can either belong to the category or not. The certain rules and boundaries result in exact decisions with no vagueness. The members of the same category have the same properties in common and the common properties should be satisfied for the membership of a category [21]. One of the problems with this view is that people not always agree on categories, therefore there can be unclear cases where the resulting categorization decision is arguable. Another problem is typicality handling. Members of a category may expose variance in how well they satisfy the membership. To illustrate, a falcon and a chicken are both birds but their membership of bird category is different in terms of typicality.

The second view is the prototype based view and for this view, the membership of categories is confidence-based [38]. In contrast to rule-based view, boundaries are not strict in prototype-based view. "Prototype" stimuli which are the ones best representing the category are used in decision making mechanism for membership of a category. Statistical regularities and the frequency distributions of the features are used for representing prototypes [21]. In this thesis, we also adopt the prototype-based view for the method we propose.

Lastly, the third approach is the exemplar-based view and this view suggests representing categories by exemplars which are stored in the memory. To be able to categorize an instance, it is compared to the exemplars. If it is found to be similar to one of the stored exemplars, it is considered as a member of that category.

2.4 Prototype Theory

According to prototype theory, category learning can be achieved via the learning of category prototypes. To be able to assign an unfamiliar stimulus to a category, most similarity is the concern and most similar prototype needs to be found. [20, 32, 33, 35, 42, 46].

Since they are derived based on observations, prototypes include characteristic features that are usually present, not only necessary or sufficient features [19]. Unclear cases which cause problems for rule-based methods are handled by letting an object be in a position with equal distances to prototypes of different categories. Prototype theory also handles typicality by evaluating an object's proximity to the prototype and typicality increases as moving closer to the prototype. With typicality, it is possible to list category members in a desired order since ordering information is present as well as membership decision information. In addition, typicality may give an opinion about the response time of membership decisions. Response time decreases as the instance moves towards the prototype in the feature space. Another issue is family resemblance. Membership of a category is based on the requirement that members should be similar overall. The members may all have something in common but it is not obligatory for membership of a category.

2.5 Distance Measurement

As mentioned by Walters in [46], a distance definition is stating rules to assign positive numbers between pairs of items. Let $\mathbf{i}, \mathbf{j}, \mathbf{k}$ be three point vectors and d be a distance function which maps a pair of vectors to a real positive number. With an illustration in Figure 2.1 function d has the following properties:

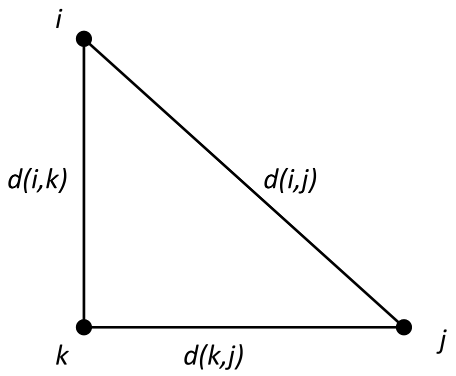

Figure 2.1: Distances of Points

- Identification mark $d(\mathbf{i}, \mathbf{i}) = 0$.

- Non-negativity $d(\mathbf{i}, \mathbf{j}) > 0$.
- Symmetry $d(\mathbf{i}, \mathbf{j}) = d(\mathbf{j}, \mathbf{i})$.
- Triangle inequality $d(\mathbf{i}, \mathbf{j}) <= d(\mathbf{j}, \mathbf{k}) + d(\mathbf{k}, \mathbf{j})$.

Euclidean distance, Manhattan distance and Mahalanobis distance are some popular distance measures and which one to choose depends on the context.

2.5.1 Euclidean Distance

The Euclidean distance, which is given by the "Pythagorean formula", calculates the length of the real straight line between two vectors.

Let $\mathbf{p}, \mathbf{q}$ be vectors, where $\mathbf{p} = p_1, ., ., p_n$ and $\mathbf{q} = q_1, ., ., q_n$, the Euclidean distance is given as follows:

$$EUD(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} \sqrt{(q_i - p_i)^2}}. \tag{2.1}$$

The norm associated with this metric is the Euclidean norm.

2.5.2 Manhattan Distance

Manhattan distance is computed by following a path in a grid-based path, with simple summations of vertical and horizontal components. It is defined as follows:

$$MN(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} |(p_i - q_i)|. \tag{2.2}$$

2.5.3 Mahalanobis Distance

For the calculation of Mahalanobis distance, the components of a vector have weights depending on their variances. The ones with low variability receive higher weights therefore they have more impact on the calculation of the distance [31]. Let p and q be two vectors. The Mahalanobis distance is given by Equation (2.3).

$$MD(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p} - \mathbf{q})^T C^{-1} (\mathbf{p} - \mathbf{q})}. \tag{2.3}$$

where C is a non-singular $n \times n$ covariance matrix which is used for generalization of the variance concept on more than one dimension. C is calculated by taking the element $C(i, j)$ as the covariance between i^{th} and j^{th} value of the sample vector.

CHAPTER 3

RELATED WORK

In this Chapter, we mention the work related to our study in this thesis. Self Organizing Maps, Learning Vector Quantization, Support Vector Machines, Robust Growing Neural Gas Algorithm, Gaussian Mixture Models and Derivation of Prototypes [21] are explained briefly.

3.1 Self Organizing Maps

This study forms a basis and provides better understanding for some of the other studies explained in this Chapter, including Learning Vector Quantization and Robust Growing Neural Gas algorithm.

Self-Organizing Map (SOM) is a type of artificial neural network which was developed by Tuevo Kohonen in 1982 [23]. The "Self-Organizing" term is used because it learns without a supervisor, forming the output by itself. SOM uses unsupervised competitive learning, aiming to become like the given input data by reducing the dimensions of it.

Three layers are used for forming SOMs, the input layer, the computational layer and the output layer. The basic SOM has M neurons placed on a low-dimensional grid of computational layer. Generally 1 to 2 dimensions are used for this layer. The reason of keeping dimensions low while higher dimensions are also possible is that visualization of higher dimensional layers are relatively impractical [48]. Each neuron in the computational layer is fully connected to the neurons in the input layer and they are not connected to each other. Figure 3.1 shows the structure of SOMS.

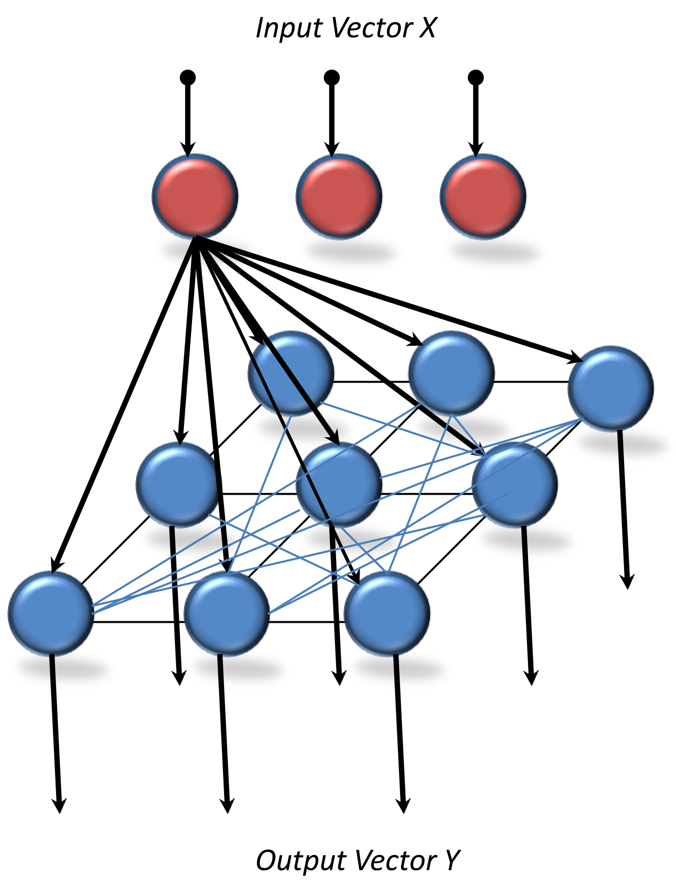

Figure 3.1: SOM Structure

The purpose of developing such a structure is to learn a "feature map" from a continuous input space to a low dimensional discrete output space. This is achieved by adapting the neurons in the computational layer. Furthermore, topology is preserved in SOMs.

The algorithm can be presented in 6 steps as in [17]:

1. The values for the initial weight vectors are initialized.
2. A randomly chosen vector from the training set is presented to the network.
3. Every node in the neural network is checked to find out whose weights are the most similar to the input vector. A winner node is detected which is also known

as $BestMatchingUnit(BMU)$.

$$D = \sqrt{\sum_{i=0}^{n} I_i - W_i}, \tag{3.1}$$

where I is the input vector, W is the weight vector of node, D is the distance for examining proximity by Equation (3.1).

4. The neighborhood radius of BMU is calculated. Typically it is initialized to the radius of the network which is given by Equation (3.2) and it is decreased at each iteration.

$$R = \sigma(t) = \sigma_0 e^{-t/\lambda}, \tag{3.2}$$

where t is the current iteration, λ is the time constant given by Equation (3.3) and σ_0 is the radius of the map.

$$\lambda = numIterations/mapRadius. \tag{3.3}$$

5. The nodes which stay inside the radius of the BMU are adapted. The closer nodes' weights are updated more significantly. The updated node weight is given by Equation (3.4).

$$W(t+1) = W(t) + \Theta(t)L(t)(I(t) - W(t)), \tag{3.4}$$

where the learning rate $L(t)$ is given by Equation (3.5), and $\Theta(t)$ is given by Equation (3.6).

$$L(t) = L_0 e^{t/\lambda}, \tag{3.5}$$

$$\Theta(t) = \exp(-disFromBMU^2/(2\sigma^2(t))). \tag{3.6}$$

6. Repeat previous steps for N iterations.

3.2 Learning Vector Quantization

Learning Vector Quantization (LVQ) is an artificial neural network (ANN) which combines SOM's competitive learning with supervised learning and it is a powerful and intuitive classification algorithm for adaptive nearest prototype classification. LVQ aims to generate optimal reference vectors [40].

Let Q be a training set $\{s^q : t^q\}$ containing training vectors and target outputs. s^q are vectors of with N dimensions and t^q are target output vectors with M dimensions where $q = 1, 2, ., ., .Q$. The target output vectors can be defined as in Equation (3.7):

$$At_i^{(q)} = \begin{cases} 1, & \text{if } s^q \text{ belongs to class } i \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

As stated by Kohonen [24], the structure consists of an input layer, computational layer and an output layer like SOMs do. Nevertheless, LVQ has two parts in its computational layer, namely a competitive layer and a linear layer. In the competitive layer, each one of the neurons is assigned to one class and different neurons may be assigned to the same class. These classes are mapped to one neuron in the linear layer. Therefore $Q \geq M$ condition should be preserved where Q is the number of neurons in the competitive layer and M is the number of neurons in the linear layer.

Competitive layer learns to classify a region in the given input space by letting the neurons learn a prototype vector. The similarity between an input vector and a weight vector is measured by Euclidean distance as it is described in the following equation:

$$n^{(1)} = \begin{bmatrix} \left\| x - W_1^{(1)} \right\| \\ \left\| x - W_2^{(1)} \right\| \\ . \\ . \\ . \\ \left\| x - W_Q^{(1)} \right\| \end{bmatrix}. \tag{3.8}$$

After finding the closest weight vector to the input, the corresponding output element is set to 1, indicating that the input vector lies in the boundaries of that class. This procedure can be represented as $a^{(1)} = compet(n^{(1)})$. These classes are called subclasses because some of them may be identical. The second layer of the computational layer, namely linear layer is used to combine subclasses into one class. It is achieved by using the second weight matrix $W^{(2)}$ where,

$$AW_{ij} = \begin{cases} 1, & \text{if the neuron belongs to a subclass of } j \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

The second weight matrix $W^{(2)}$ stays unchanged after it is set; however, the weights of $W^{(1)}$ needs to be trained. At each iteration, input vector x from the training set is presented to the network and the Euclidean distances between x and each one of the prototype vectors which are forming the columns of the weight matrix, are calculated to find out the closest component. If the neuron with the index j is the winner, the j^{th} element of $a^{(1)}$ is set to 1 while others are set to 0. To get the output of entire network $a^{(2)}$, $a^{(1)}$ is multiplied by $W^{(2)}$. The only non-zero element of $a^{(2)}$ means that the corresponding class is the output for the given input vector.

The LVQ structure is described in Figure 3.2.

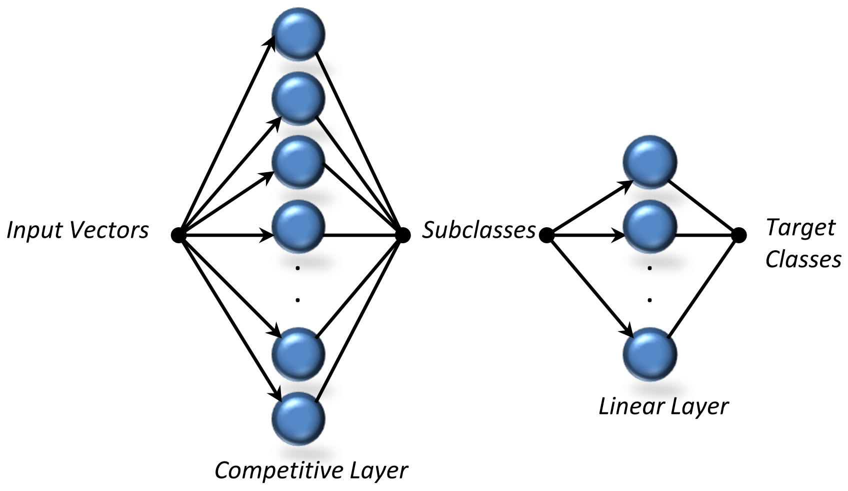

Figure 3.2: LVQ has two layers which are the competitive layer and the linear layer. The competitive layer outputs subclasses which are the inputs of the linear layer. Then, linear layer outputs the target classes.

3.3 Support Vector Machines

A support vector machine is a learning mechanism constructing hyperplanes which separate the classes in feature space. Classes are bounded by edges which are composed of support vectors and these vectors are used for the learning process. Main purpose is to separate the classes between each other by using hyperplanes. In the mean time, class margins need to be maximized[45].

Originally, SVM is binary classifier and a hyperplane which separates the two classes needs to be found based on the idea that these two are separable. The hyperplane that separates the classes best may not be found exactly between two classes. For this case, there exists a trade-off between the minimization of training sample count which are located in the wrong side and maximization of the class margins. To be able to handle this trade-off, an error item is proposed. By replacing the inner product of optimal hyperplane with non-linear functions , SVM becomes capable of handling non-linear classification. There are several kernel functions which are used frequently, including radial basis function (BRF), sigmoid kernel, linear kernel and polynomial kernel [41]. Kernel functions map the data into higher dimensional spaces. This is because in higher dimensions, the data may be separated more easily or structured better.

Class separation by SVM is illustrated in Figure 3.3.

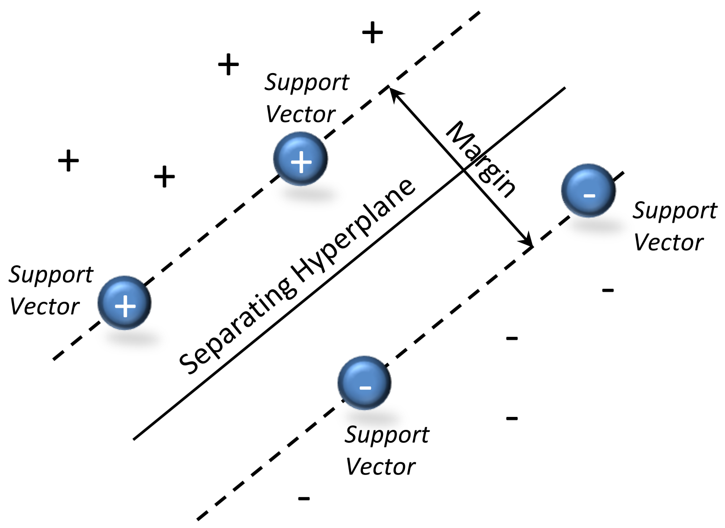

Figure 3.3: SVM maximizes the margin between classes and finds the hyperplane which separates them. The classes are bounded by support vectors.

SVM can also be used for multi-class mapping. This capability is achieved by dividing the class set to several binary sets and applying the original SVM procedure to all. There are three popular methods for making multi-class extension to SVM. The first one is one-against-one method, the second one is one-against-all [8, 22] and the third one is Directed Acyclic Graph-Support Vector Machine (DAGSVM) method.

3.4 K-Nearest Neighbor Algorithm

The k-Nearest Neighbor (k-NN) algorithm is a supervised and instance-based learning algorithm [43]. k-NN makes classification by considering K training vectors which are the closest to the test instance vector.

In the training part, k-NN does sorting and optimizations on the labeled feature vector set. For the classification part, k being a user defined constant, k-NN finds the most frequent k neighbors which have the smallest distance to a given test instance. Euclidean distance, which is explained in Section 2.5.1, is widely used for distance calculation in this algorithm.

A simple explanation of k-NN algorithm is given in Equations 1 and 2.

Algorithm 1 k-Nearest Neighbor Algorithm (training) for approximation of a discrete-valued function. Adapted from [29].

```
Input: training set $TS$,$k$
Output: Boosted hypothesis.
1: procedure K-NEAREST NEIGHBOR TRAINING
2:     for all training sample $\langle x, f(x) \rangle$ in $TS$ do
3:         Add the sample to $tr_{samples}$ list.
4:     end for
5:     return the trained model including $tr_samples$ and $k$.
6: end procedure
```

Algorithm 2 k-Nearest Neighbor Algorithm (testing) for approximation of a discrete-valued function. Adapted from [29].

```
Input: training model including $tr_samples$ and $k$, test vector $x_q$
Output: classification result for $x_q$.
1: procedure K-NEAREST NEIGHBOR TESTING
2:     Find the nearest $k$ vectors to $x_q$ in $tr_samples$.
3:     Store the nearest vectors as $x_1, x_2, ..., x_k$.
4:     return the result using Equation (3.10).
5: end procedure
```

For the testing algorithm of k-NN explained in Algorithm 2, the classification result is calculated as follows:

$$\widehat{f}(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i)), \tag{3.10}$$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise.

3.5 Adaptive Boosting

Adaptive Boosting (AdaBoost) [10, 11] formulated by Yoav Freund and Robert Schapire is an algorithm which is an ensemble-learning type method. AdaBoost uses one or multiple learning algorithms to derive better performance out of them. It boosts the performance of the other algorithms, which are called weak learners, and creates an improved classification ability. Maintaining a set of weights over a given training set, it comes up with a stronger hypothesis with better fitting to the data. There are two versions of Adaboost, called AdaBoost.M1 and AdaBoost.M2. The difference between them is that AdaBoost.M1 does binary classification while AdaBoost.M2 can do classification for more than two classes [9]. AdaBoost.M2 algorithm is described in Algorithm 3.

For the procedure explained in Algorithm 3, final hypothesis is calculated as follows:

$$h_{fin}(x) = arg \max_{y \in Y} \sum_{t=1}^{T} (\log \frac{1}{\beta_t}) h_t(x, y). \tag{3.11}$$

where

$$D_{t+1}(i, y) = \frac{D_t(i, y)}{Z_t} \beta_t^{(1/2)(1+h_t(x_i,y_i)-h_t(x_i,y))}, \tag{3.12}$$

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}, \tag{3.13}$$

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y)), \tag{3.14}$$

and Z_t is a normalization constant.

AdaBoost algorithm is often sensitive to outliers in the data, however it is resistant to overfitting.

Algorithm 3 AdaBoost.M2 Algorithm. Adapted from [9].

```
Input: Training set $(x_1, y_1)$, $(x_2, y_2)$..., $(x_m, y_m)$ with labels $y_i \in Y = 1, ..., k$,
Input: Weak learning algorithm $WLearn$, iteration count $T$
Output: Boosted hypothesis.
 1: procedure AdaBoost.M2
 2:     Let $B = \{(i, y) : i \in \{1, ..., m\}, y \neq y_i\}$
 3:     Initialize $D_1(i, y) = 1/\|B\|$ for $(i, y) \in B$.
 4:     for all $t$ in $1, 2, ..., T$ do
 5:         Call $WLearn$ with $D_t$.
 6:         Get a weak hypothesis $h_t : X \times Y \to [0, 1]$.
 7:         Calculate the pseudo-loss of $h_t$ by Equation (3.14).
 8:         Set $\beta_t$ by Equation (3.13).
 9:         Update $D_t$ by Equation (3.12).
10:     end for
11:     Return the final hypothesis by Equation (3.11).
12: end procedure
```

3.6 Decision Tree Learning

Decision tree learning is a supervised learning method, which creates a tree-like structure by making inferences on a given training set. In other words, they are used for making approximations of target functions and these functions can be visualized as decision trees [29]. Decision trees have internal nodes and leaf nodes. Internal nodes correspond to input features while leaf nodes correspond to classification results.

Decision trees have two types: which are classification trees and regression trees. Classification trees are for nominal predictions while regression trees are for numeric responses. The algorithm partitions the data and represent it as a tree starting from the root node to the leaf nodes. When applying the partitioning procedure, the similar valued instances are collected in the same subset as much as possible for preserving homogeneity. Entropy is used for measuring the homogeneity of the tree. If the tree is completely balanced, the entropy is 0. Therefore, decision tree learning algorithm aims to minimize the entropy.

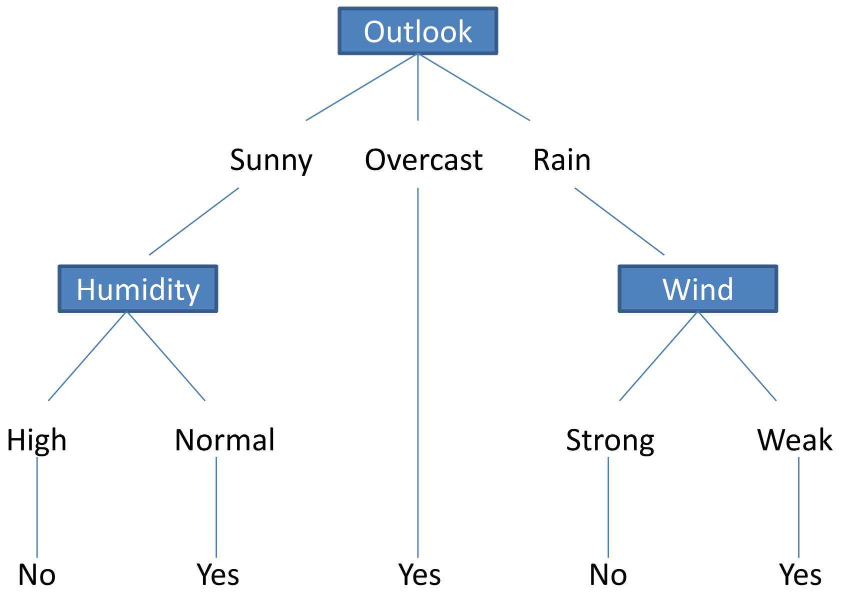

Figure 3.4: An example of decision tree for the concept $EnjoySport$ which is also described in Section 2.3.1. This tree is constructed for classifying Saturday mornings, whether person X enjoys doing sports on that day or not. Adapted from [29].

Information gain is another notion when forming a decision tree. It is the the decrease in the entropy when the tree data is partitioned with respect to an attribute, in other words, information gain is the entropy difference between the first entropy and the total entropy of the branches after the partitioning.

3.7 Robust Growing Neural Gas

The Neural Gas Algorithm (NG) [27] is applied to fields such as vector quantization, pattern recognition and clustering etc. The algorithm shows similarities to Kohonen's Self Organizing Map. The difference is adapting the reference vectors without fixed topological ordering. NG is considered as a soft competitive learning ANN with a single layer. It adjusts the winner vector and other reference vectors given an input vector, taking the proximities to the input vector into account [34].

Originating from the NG algorithm and containing a variable topology, Growing Neural Gas Algorithm (GNG), which is an incremental self-organizing network, was proposed by Fritzke [12, 13]. An edge vector is defined for each reference vector to its direct topological neighbors. It starts with small number of prototypes, typically two, and after a certain number of epochs a new prototype is introduced to the network. The reference insertion mechanism is the reason that the network is called growing neural gas and via this mechanism, the impact of initial prototypes is reduced. When a performance goal is achieved or a predefined reference vector count is reached, the growing procedure is stopped. GNG updates the winning vector and its direct topological neighbors with respect to an input vector. The updating strengths for the winner and its direct topological neighbors are different.

The updating rule is described by Equations (3.15) and (3.16).

$$\Delta w_{s1} = \epsilon_b(x - w_{s1}), \tag{3.15}$$

$$\Delta w_i = \epsilon_n(x - w_i), \tag{3.16}$$

where x is the input vector, w_{s1} is the winner vector, w_i is the i^{th} topological neighbor of the winner vector, ϵ_b and ϵ_n are the updating strengths respectively.

The GNG algorithm is also able to detect the inactive reference vectors which are not updated for a certain predefined amount of time. By removing the edges which are considered old, it modifies the network.

Robustness of an algorithm requires the following properties [34]:

1. For an assumed model, the algorithm should achieve a reasonable accuracy.
2. Small deviations of model assumption should not result in large deviations of performance decrease.
3. Larger deviations of model assumption should not result in a catastrophe.

Traditional prototype based clustering algorithms are most of the time successful considering the first property. But many of them suffer from robustness issues including being very sensitive to initialization, dealing with large number of outliers and the order which input vectors are presented to the system. RGNG algorithm overcomes the robustness defects of GNG algorithm.

RGNG seeks the optimal number of prototypes by employing the minimum description length (MDL) criterion [16]. Furthermore, it aims to enhance the robustness of the updating rule of GNG algorithm by introducing a parameter in addition to the updating strengths ϵ_b and ϵ_b in Equations (3.15) and (3.16). This parameter is used for limiting the force which is amplified by the outliers of the input vectors.

Learning rates of the prototypes are decreased as time step is increased because it is desired that they have more impact at the beginning. In time, the decreased learning rates of the prototypes allow their positions to converge. RGNG algorithm applies different learning rates to the newly inserted prototypes and the older ones while GNG algorithm applies the same learning rates to all prototypes. In RGNG, newly introduced prototypes have larger learning rates than the older ones since they are introduced to the system to find new clusters while others started to converge on some position values.

RGNG presents another feature that it applies a repulsion scheme to solve the prototype coincidence problem [44]. Prototype coincidence problem occurs when two prototypes find the same cluster during the learning procedure.

3.8 Gaussian Mixture Models

A Gaussian Mixture Model is used for capturing the distribution of data using multiple Gaussian functions, i.e. components. GMM is often used for clustering purposes. For each data point, a posterior probability is assigned and this indicates that each data point has some probability to belong to each of the clusters. The number of clusters should be given as input to GMM and the clusters are assigned by selecting the component which maximizes the posterior probability. An algorithm with iterative approach which converges to a local optimum is used in GMM.

In Figures 3.5 and 3.6, clustering with GMM is illustrated.

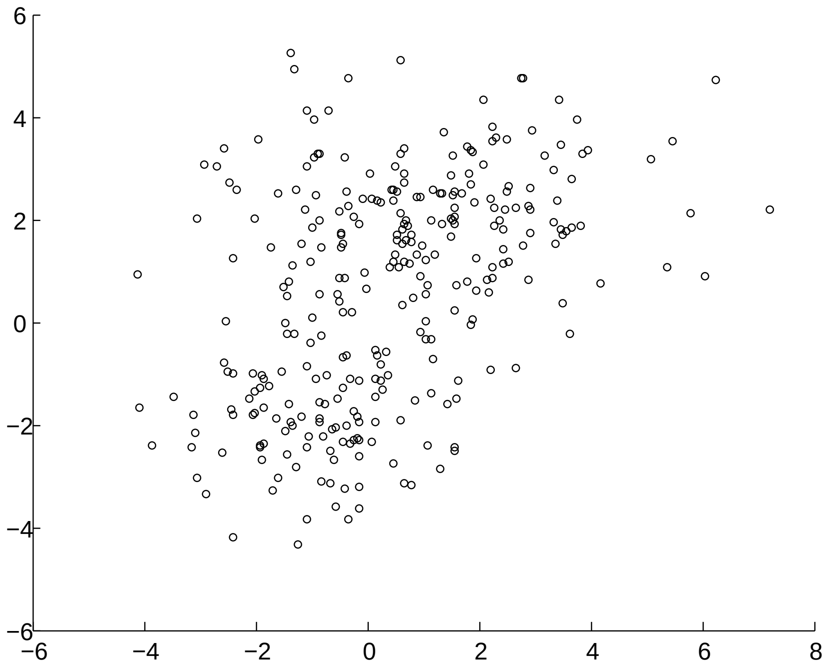

Figure 3.5: Random data without before clustering

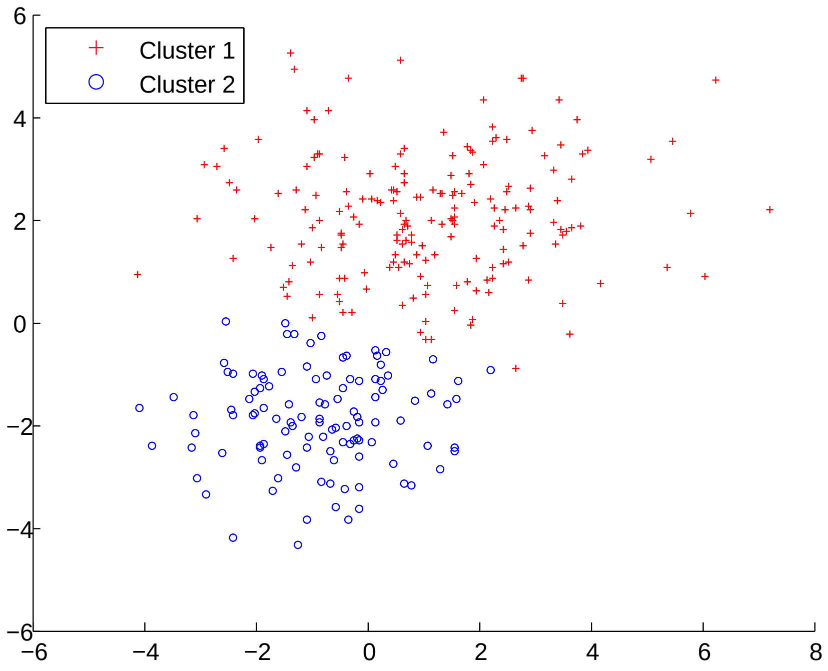

Figure 3.6: Random data clustered with GMM.

GMM is a weighted sum of N component Gaussian densities [36] and it is described as follows:

$$p(x|\lambda) = \sum_{i=1}^{N} w_i g(x|\mu_i, \Sigma_i), \tag{3.17}$$

where x is a data vector of dimension D with continuous values, w_i, $i = 1, 2, ..., N$ are the mixture weights and $g(x|\mu_i, \sum_i)$, $i = 1, 2, ..., N$ are the component Gaussian densities. Each of them is described as:

$$p\left(x|\mu_i, \Sigma_i\right) = \frac{1}{(2\pi)^{D/2} |\Sigma_i|^{1/2}} \exp\left(\frac{-1}{2}(x - \mu_i)' \Sigma_i^{-1} (x - \mu_i)\}\right), \tag{3.18}$$

where μ_i is the mean vector and Σ_i is the covariance matrix and the constraint $\sum_i^N w_i = 1$ is satisfied. The parameterized form of GMM is described by Equation (3.19):

$$\lambda = \{w_i, \mu_i, \Sigma_i\}, \tag{3.19}$$

where $i = 1, 2, .., N$.

3.9 Derivation of Effect Prototypes

Notion of affordance was described by Gibson [15] as the ability of perceiving the environment with the action possibilities available. To illustrate, if we look at a ball, we do not perceive a generic view out of it. Instead, we perceive a set of abilities like throw-ability or lift-ability that the ball offers to us [21].

Sahin et al. [3] formalized the notion for learning and using affordances at autonomous robot control and represented an affordance relation instance tuple as:

$$(entity, behavior, effect),$$

where the term *entity* is defined as a single object, the term *behavior* corresponds to the actions that a robot can take and the *effect* is defined as the perceptual change created in the environment by the corresponding execution of that behavior.

Kalkan et al. [21] described effect instances whose features can be perceived by the iCub humanoid robot. The robot has a repertoire of behaviors including "Move Right", "Move Left", "Move Forward" etc. Using these behaviors for interacting with different objects, a labeled training set is created and different effect prototypes are derived out of it including "Moved Right", "Moved Left", "Moved Forward" etc. Along with a prototype-based learning algorithm 4, details of which is described in Chapter 4, a modified version of Mahalanobis distance is proposed. The proposed distance measurement function is used to evaluate the similarity between a new effect vector and the effect prototypes generated.

CHAPTER 4

A MULTINOMIAL PROTOTYPE-BASED LEARNING ALGORITHM

This thesis aims to offer a solution to the problem of derivation of prototypes in the spaces of multipolar dimensions. In their study Kalkan et al. [21] proposed an algorithm to derive prototypes by examining the mean and variance values of features. The method partitions the input data in the space of means and variances and derives prototype labels for each feature of a prototype. These labels are called prototype labels and they give an insight into the feature characteristics of the prototype. For example, a feature label denoted with the symbol '*' indicates that feature is not a reliable source for describing the class that the corresponding prototype represents. When a new input vector is compared with this prototype the features with the label '*' is omitted. The reason is that the feature displays no informative characteristics but disorder and it is not credible to put it into account deciding whether the given input is similar to that prototype or not. The datasets collected using iCub humanoid robot [28] are used for proving that the method works well on learning verb concepts. The problem, however, is dealing with the datasets which have feature value distributions with more than one peak. In other words, when training instances have features with more than one pattern, those features are perceived as garbage information and do not have impact on describing classes with the derived prototypes.

In this thesis, we address this problem and propose a novel method which detects the existence of multipolar patterns in features that look like one disordered pattern at first glance. We introduce a new prototype structure which keeps all meaningful information about the data it represents, including the mean and variance values of the detected peaks if found any. Moreover, a new distance function, which is built on the basis of Mahalanobis distance, is developed for handling the new properties and the new representation of prototypes. The overall system architecture we propose for this thesis is shown in Figure 4.1.

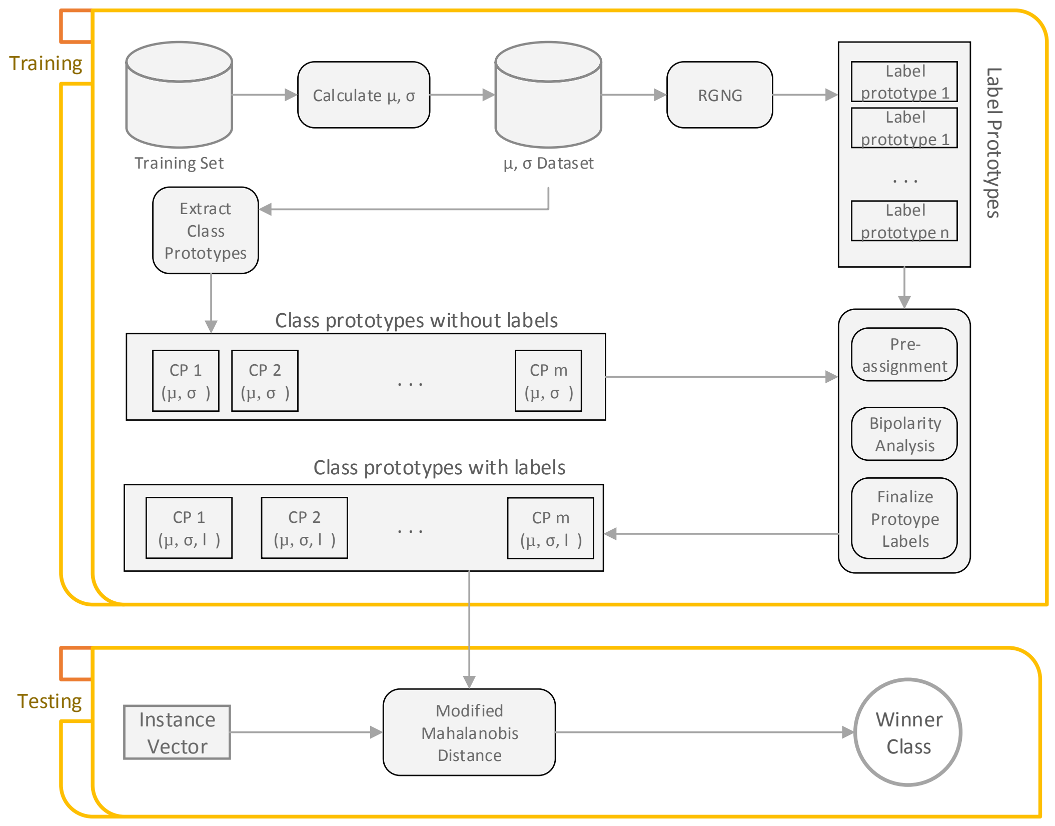

Figure 4.1: Overal System Architecture

4.1 Derivation of Prototypes

Similar to all prototype-based learning methods, the analysis of training samples is crucial for derivation of prototypes method as well. The training set is grouped by class labels, and each group is processed to form a class prototype. The first step for the derivation of prototype labels is to examine the features of the sample vectors. For each class group, mean and variance values of input vector features are calculated. Having the mean and variance values of the features, we can examine their distribution characteristics. This way, the information which is collected out of the features is used for representation of the class prototypes. An important part of this descriptive feature information is the prototype label as well as mean and variance values. In order to extract the labeling information out of the features, we use Robust Growing Neural Gas algorithm [34]. RGNG is an unsupervised clustering algorithm, therefore by using the mean and variance values, features are clustered according to their distribution characteristics. The output is a set of cluster center positions and for each feature cluster, a label is assigned manually.

Algorithm 4 Derivation of Prototypes Algorithm [21].

```
Input: Training dataset of feature vectors and their labels.
Output: Prototypes for each class.
 1: procedure DERIVATION OF PROTOTYPES
 2:     for all $C$ in the set of clusters $\epsilon$ do
 3:         Compute the mean ${}_i\mu_C = 1/N \sum_{f \in C} {}_i f$,
 4:         where $N$ is the cardinality of the set $\{f \in C\}$.
 5:         Compute the variance ${}_i\sigma_C = 1/N \sum_{f \in C} {}_i f -_i \mu_C$
 6:     end for
 7:     Apply RGNG algorithm in the space of $\mu x \sigma$ using Algorithm 5.
 8:     Manually assign labels describing the characteristics of the feature,
 9:     '+', '-', '0', '*' etc.
10: end procedure
```

4.1.1 Robust Growing Neural Gas

RGNG algorithm [34] is an iterative prototype-based clustering algorithm. It starts with a few prototypes and the number of prototypes gradually increases during the iterations. The number of prototypes is usually initialized to two and after some predefined number of epochs, a new prototype is introduced to the network. In the domain of this thesis, RGNG prototypes correspond to our label prototypes, not the overall class prototypes.

Algorithm 5 Robust Growing Neural Gas Algorithm (Adapted from [34]).

Input: Training set to be clustered: X ,desire prototype count: $prenumnode$

Output: Cluster center positions (prototypes): W

```
1: procedure RGNG
2:     Initialize current prototype vectors $W_c$.
3:     Initialize learning rates.
4:     Initialize prototype connection vector, $C$.
5:     Initialize connection age vector, $A$.
6:     Initialize maximum epoch number, $maxEpochNum$.
7:     Initialize other constants.
8:     while $currentnumberofprototypes \leq prenumnode$ and performance measure is not satisfied do
9:         for $m = 0$ to $MaxEpochNum$ do
10:            Calculate $d_k^m(0)$ with Equation (4.5) for each prototype.
11:            Calculate learning rates for current prototypes.
12:            Set step count $t = 1$.
13:            Set $currentTraningSet$ to $X$
14:            while  do
15:                Get a random input $x$ from $currentTraningSet$.
16:                Determine the winner prototype, $w^1$ and second nearest protoype, $w^2$.
17:                Update $C$ w.r.t. $w^1$ and $w^2$.
18:                Update winner prototype and its direct neighbors w.r.t. $x$.
19:                Update $A$ w.r.t $w^1$ and $w^2$.
20:                Remove old edges and prototypes which are not updated for a long time.
21:                Increment $t$ by 1.
22:            end while
23:        end for
24:        Calculate MDL values w.r.t. current prototypes.
```

Algorithm 5 Robust Growing Neural Gas Algorithm (continued)

```
25:         if $currentMDL > previousMDL$ then
26:             Save current prototype positions.
27:         end if
28:         Insert new prototype to the network.
29:         Update $C$.
30:     end while
31:     Set $W = W_c$.
32: end procedure
```

In RGNG, the prototypes, in other words the reference vectors, are updated at each iteration according to an introduced input vector to the network. In time, the positions of the reference vectors converge to their optimal values. When a new input vector is presented to the system, the best matching reference vector is found by calculating the distance of each reference vector to the input vector. Neighborhood information of prototypes is kept in a matrix structure. Using this information, the winning prototype's neighbors are found and both the winner and its neighbors are updated. The updating function does not only depend on the distance to the input vector but also some rules which makes the algorithm robust and reliable. The considerations when updating weights of prototypes are as follows:

- The updating strength of the prototypes should change with their ages. The ages of the prototypes should be kept throughout the life time of the algorithm. Newer prototypes should be updated with larger weights because they are inserted to find out new clusters while the older ones converging to the positions for cluster which are already revealed.

- Outliers should not have a large impact on the adaptation of cluster center positions. Therefore the absolute distance information between the prototype and the input vector should be modulated to identify and eliminate the impact form outlier inputs.

- Two prototypes finding the same cluster at the same time during the training procedure is called prototype coincident problem [44]. If an input vector both attracts the winner prototype and it's neighbor, a repulsion force is applied in the local neighborhood of the winner prototype.

The updating rules for the winner prototype and its neighbors are described in Equations (4.1) and (4.2) respectively.

$$\delta w_{s_1} = \epsilon_b^{s1} \sigma_{s_1}(iter) \left(\frac{x - w_{s_1}}{\|x - w_{s_1}\|} \right), \tag{4.1}$$

$$\delta w_i = \epsilon_n^i \sigma_i(iter) \left(\frac{x - w_i}{\|x - w_i\|} \right) + \exp\left(\frac{-d_{s_1 i}}{\zeta} \right) \times \beta \frac{\sum_i d_{s_1 i}}{|N_{s_1}|} \frac{w_i - w_{s_1}}{\|w_i - w_{s_1}\|} \tag{4.2}$$

where $\forall i \in N_{s_1}$.

$$\begin{aligned} &\sigma_k(iter) = \\ &\sigma_k^m(t) = \begin{cases} d_k^m t, & if \|x_t^m - w_k^{iter}\| \geq d_k^m(t-1) \\ \|x_t^m - w_k^i ter\|, & if \|x_t^m - w_k^{iter}\| < d_k^m(t-1) \end{cases} \end{aligned} \tag{4.3}$$

with

$$d_k^m(t) = \begin{cases} \left\{ \frac{1}{2} \left[\frac{1}{d_k^m(t-1)} + \frac{1}{\|x_t^m - w_k^{iter}\|} \right] \right\}^{-1}, & if \|x_t^m - w_k^{iter}\| \geq d_k^m(t-1) \\ \frac{1}{2} \left[d_k^m(t-1) + \|x_t^m - w_k^{iter}\| \right], & if \|x_t^m - w_k^{iter}\| < d_k^m(t-1) \end{cases} \tag{4.4}$$

and the initial value of the function $d_k^m(t)$ that is described as follows:

$$d_k^m(0) = \left[\frac{1}{N} \sum_{j=1}^{N} \frac{1}{\|x_j - w_k^{mN}\|^{-1}} \right], \tag{4.5}$$

where N is the number of input vectors, x_t^m is the input vector presented at iteration t, training epoch m and w_k^{iter} is the prototype vector k at the iteration $iter$. $d_k^m(t)$ is used for limiting the large absolute distances caused by the presence of outliers.

ϵ_b and ϵ_n,described in Equations (4.6) and (4.7) respectively, are the learning rates applied to the winner prototype and its neighbors respectively and they are not constant values. They are functions which decrease monotonically with the increment of prototypes.

$$\epsilon_b^l = \epsilon_{bi}^l \left(\epsilon_{bf}^l / \epsilon_{bi}^l \right)^{prenode^l / pre\_numnode}, \tag{4.6}$$

$$\epsilon_n^l = \epsilon_{ni}^l \left(\epsilon_{nf}^l / \epsilon_{ni}^l\right)^{prenode^l/pre\_numnode}, l = 1, 2, .., c, \tag{4.7}$$

where c is the current number of prototypes, ϵ_{bi}^l, ϵ_{bf}^l, ϵ_{ni}^l and ϵ_{nf}^l are the initial and final values of ϵ_b^l and ϵ_n^l respectively which are predefined constant values.

During the iterations of the algorithm, inactive prototypes are detected. The age values of edges are incremented by 1 after each iteration and the edges older than some predefined maximum age are removed from edge list. If a prototype has no connected neighbors after this edge updating procedure, the prototype is removed from the network.

The whole learning procedure stops after a certain performance measure is met or the predefined number of prototypes is reached. The result is a set of cluster prototypes for the given training set. In our case, it is the label prototypes representing the clusters in the space of means and variances.

4.1.2 Prototype Labels

In our study, prototype labels play an important role for defining class prototypes. The labeling information gives an insight into the distribution behavior of the features in prototype vectors along with the mean and variance values.

The output of RGNG is a set of reference vectors each containing features of mean and variance values. In their study [21], Kalkan et al. focused on learning verb concepts and they managed to derive effect prototypes out of object feature changes. They used the following labels for the clusters that emerge as the output of RGNG algorithm:

- '+' meaning consistently increasing feature data.
- '-' meaning consistently decreasing feature data.
- '0' meaning no change in the feature data.
- '*' meaning unpredictable change in feature data.

The mean-variance relations of the generated label prototypes are illustrated in Figure 4.2. Consistently increasing features have high mean values with low variances and they are represented by a prototype with label '+'. Likewise, prototypes with '-' and '0' labels represent features with consistent behaviors. However, prototypes with '*' labels have high variances. The distribution patterns of label prototypes are shown in Figure 4.3.

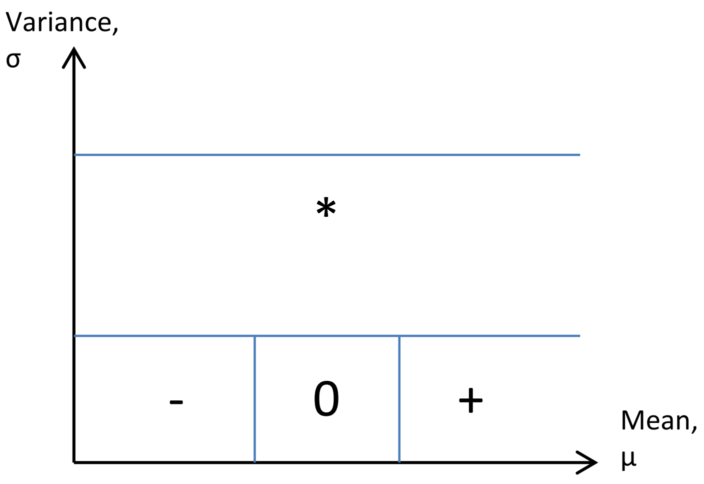

Figure 4.2: Label prototypes on $\mu \times \sigma$ space.

Label cluster number and label choices may change with respect to different types of clustering problems and training sets with different characteristics. Within the scope of this thesis, we focused on classification problems in the spaces of multipolar dimensions. Distinguishing the dimensions which have unpredictable behaviors from those which have multiple meaningful patterns is one of our main concerns. Therefore, in addition to the four label strings described above, we have introduced a new one representing the existence of multipolarity for the feature data, denoted by the symbol '#'. Choosing between the labels '#' and '*' is a critical decision for generating prototypes that can successfully represent the training data.

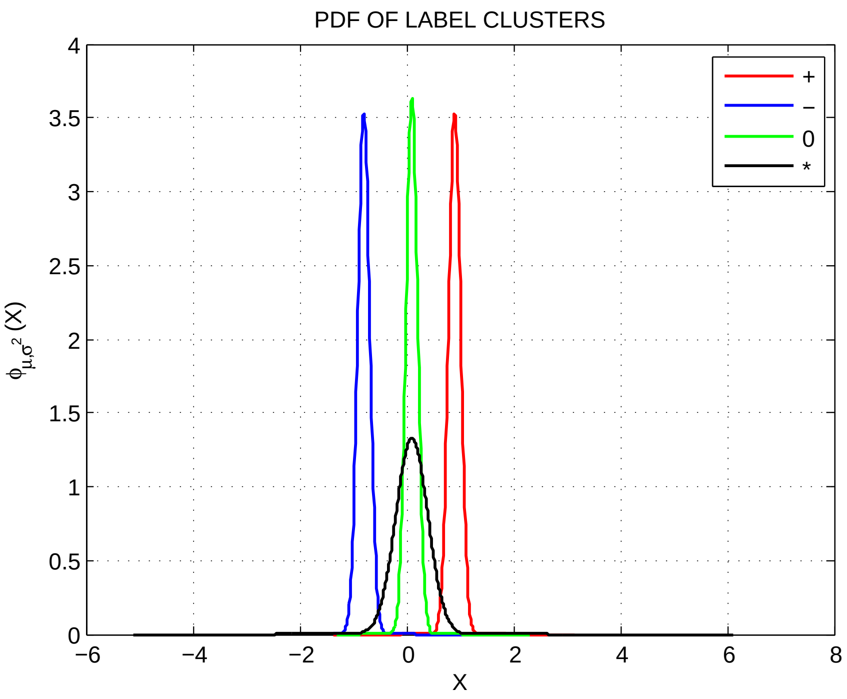

Figure 4.3: Distribution charateristics of label prototypes.

4.2 Multipolar Dimensions

Kalkan and his colleagues [21] showed in their study that the label prototype approach works well for deriving effect prototypes. For example, the algorithm can detect unpredictable changes in feature data, and by denoting a '*' label it uses this information for representing the corresponding feature. This information is valuable because it directly affects the calculation of the distance between an input vector to be classified and the prototype that the particular feature belongs to. It is wise to omit the features with unpredictable behaviors when describing a class, but an important problem arises at this point. Sometimes more than one meaningful pattern exist and look like one disordered pattern. In such a case, instead of retrieving the hidden distribution patterns out of the data and examining it, assigning '*' label and omitting a potentially valuable data may cause precision loss in clustering abilities or a completely wrong class decision which is even worse. Therefore, the existence of multipolar dimensions in input and prototype vectors is an important concern.

In Section 4.2.1, we define multipolar dimensions and in Sections 4.2.2 and 4.2.3, we investigate two possible approaches of dealing with multipolar dimensions for prototype derivation and classification. In this thesis, we adopt the latter which is introduced in Section 4.2.3.

4.2.1 Characteristics

If a dimension of a prototype shows multipolarity, it means, when the prototype is defined as a member of a certain class, there are multiple different observed and expected behaviors for the feature represented by that dimension. However, in the case of unipolarity, there is only one type of behavior, one peak in the distribution of the data and one mean - variance value pair.

Feature clusters with high variances which are labeled to be unpredictable by the RGNG algorithm may in fact have different clusters with small variance values in them. Let $f \in F$ is a feature instance value where F is a set with mean $\mu_F = 0.1149$ and variance $\sigma_F = 0.6921$ within the range of $[-1, 1]$. The distribution behavior of f is shown in Figure 4.4.

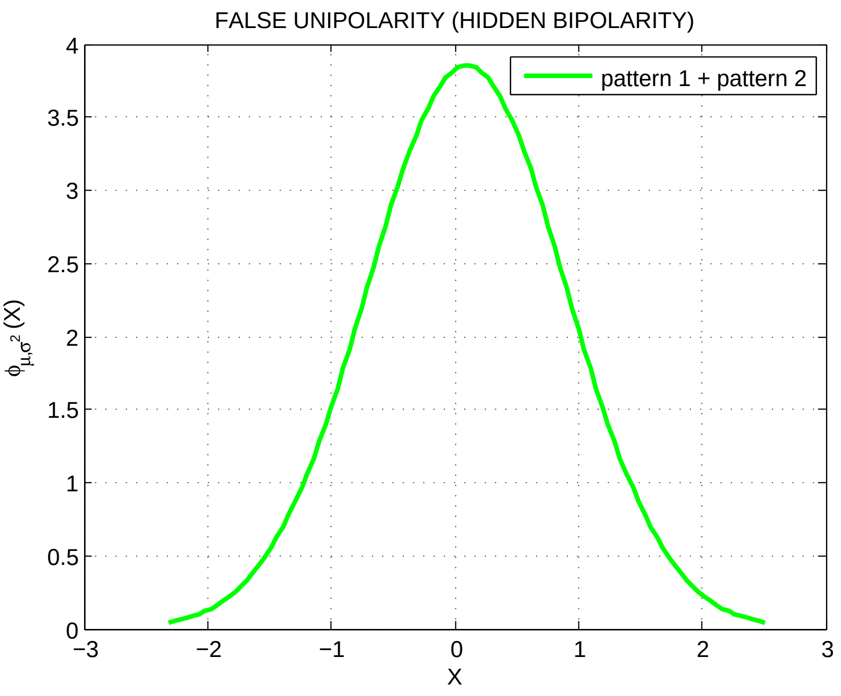

Figure 4.4: Hidden bipolarity in feature value distribution.

Without further analysis on the dataset, the distribution pattern shown in Figure 4.4 and the '*' label which is obtained by RGNG algorithm indicate that the feature values behave unpredictably with high variance, at first glance. It is true that given values do not have one consistent behavior and it is also true that the variance is high enough for the label of that prototype to be classified as '*' by RGNG algorithm. However, an important characteristics, which we call multipolarity (bipolarity in this case), is revealed by further analysis on the feature dataset. There actually exist two consistent behaviors combined together and it delusively looks like a complete disorder. F contains two subsets F_1 and F_2 with mean and variance values $\mu_{F_1} = 0.9332$, $\sigma_{F_1} = 0.0055$, $\mu_{F_2} = -0.7034$, $\sigma_{F_2} = 0.0053$ respectively. Each set shows a consistent behavior with low variance and they both are important for understanding the overall behavior. Figure 4.5 shows the bipolar behavior, where pattern 1 and pattern 2 shows the distributions of the subsets F_1 and F_2 respectively.

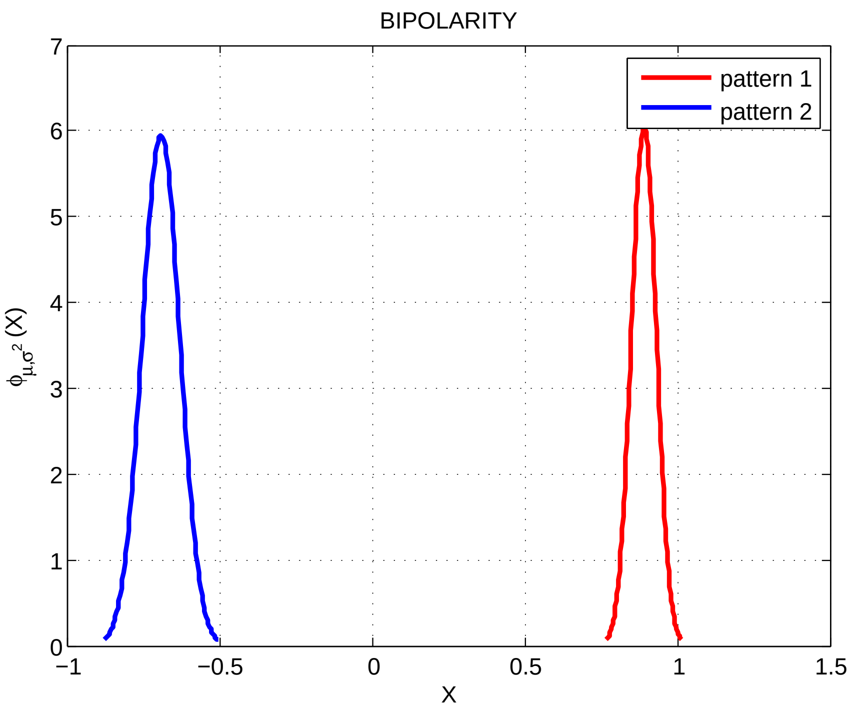

Figure 4.5: Bipolarity in feature value distribution.

To describe the concept of multipolarity of dimensions, some illustrations of effect prototypes from the study [21] can be used. The features of effect prototypes include changes in position information such as Δx, Δy, Δz. For the effect prototype "Moved Right", the feature describing the change in y axis has a large positive mean value and a small variance, while the same feature for "Moved Left" has a large negative mean value with small variance. This indicates that the prototype named "Moved Right" has the information of a consistent positive change on y axis, while "Moved Left" has the information of consistent negative change on the same axis.

In this study we describe a new behavior called "Move Sideways" and a corresponding effect prototype called "Moved Sideways" which displays multipolar behavior in its feature dimensions. The feature dimension corresponding to Δy is a bipolar dimension in this case. Whether an object is decided to be moved right or to be moved left, "Moved Sideways" condition is satisfied for both of the cases. Assume that the aim is to get an object out of the way for some reason and "Move Sideways" behavior is applied to achieve this goal. In such a case, moving the object to the right side or moving it to the left side both do the job and both of their effects on the object can be labeled as "Moved Sideways". Therefore the training data for "Moved Sideways" class can have effect instances labeled both as "Moved Right" and "Moved Left". There are two main approaches to deal with prototype vectors with multipolar dimensions, namely multiple prototype approach and single prototype approach.

4.2.2 Multiple Prototype Approach

The first approach for handling multipolarity of prototype feature dimensions is to split the prototype into multiple prototypes with each feature representing only one type of behavior. By this way, one class may have more than one prototype to be represented.

Let x be an input vector to be classified as one of the classes defined in class set C, where P^{c_i} , S_c , $S_p^{c_i}$ and S_f are the set of prototypes describing class c_i , the number of classes in C, the number of prototypes for representing the class c_i , the number of features in a prototype respectively and $i = 1, .., S_c$. The number of total prototypes is calculated in Equation 4.8.

$$S_p = \sum_{i=1}^{S_c} S_p^{c_i}. \tag{4.8}$$

To be able to classify x, it should be compared to all prototypes. The winning class that input vector x belong to is c_i, where $p_j \in P^{c_i}$ is the winning prototype and

$j = 1, .., S_p^{c_i}$.

Table 4.1 shows the structure of class prototype feature architecture with changing number of bipolar dimensions for the multiple prototype approach.

Table 4.1: Relation of bipolar dimension number and prototype number for multi prototype approach.

Class 1 (0 bipolar dimension)	Prototype			
	Feature 1 (μ_1, σ_1)	Feature 2 (μ_2, σ_2)	…	Feature n (μ_n, σ_n)
Class 2 (1 bipolar dimension)	Prototype 1			
	Feature 1 (μ_1, σ_1)	Feature 2 (μ_2, σ_2)	…	Feature n (μ_n, σ_n)
	Prototype 2			
	Feature 1 (μ_1, σ_1)	Feature 2 (μ_2, σ_2)	…	Feature n (μ_n, σ_n)
…				
Class m+1 (2^m bipolar dimensions where $n \geq m$)	Prototype 1			
	Feature 1 (μ_1, σ_1)	Feature 2 (μ_2, σ_2)	…	Feature n (μ_n, σ_n)
	…			
	Prototype 2^m			
	Feature 1 (μ_1, σ_1)	Feature 2 (μ_2, σ_2)	…	Feature n (μ_n, σ_n)

If a class does not have any multipolar dimensions, then one prototype is enough for that class to be represented. However, with the existance of multipolar dimensions it is not enough. To illustrate, if it has one bipolar dimension, feature representation should be split into two parts and so should the prototype representation. Therefore, for one occurrence of bipolar dimension, there should be two, for two occurrences of bipolar dimension there should be four prototypes and for n occurrences of bipolar dimension, there should be at least 2^n prototypes defined. Each prototype means extra space allocation cost for the features other than the ones with bipolar behaviors. Furthermore, computational cost rises up with a fast increase in the number of prototypes when calculating the distance between prototypes and an input vector because of the reoccurrence of the unipolar dimensions.

Both the computational complexity and the space complexity of this approach is $O(2^n)$. In the case of multipolarity with more than two peaks, the complexity is even higher. For n dimensions with m peaks, there has to be m^n prototypes with this approach.

4.2.3 Single Prototype Approach

The second and a better approach is called the single prototype approach. With this approach, each multipolar dimension is examined and the extracted information is kept in a single feature structure. Therefore, unlike the previous approach, the number of multipolar dimensions does not have an increasing effect on prototype count.

Table 4.2: Relation of bipolar dimension number and prototype number for single prototype approach

Class 1 (0 bipolar dimension)	Prototype			
	Feature 1 (μ_1, σ_1)	Feature 2 (μ_2, σ_2)	…	Feature n (μ_n, σ_n)
Class 2 (1 bipolar dimension)	Prototype			
	Feature 1 $(\mu_1{}^1, \sigma_1{}^1, \mu_1{}^2, \sigma_1{}^2)$	Feature 1 (μ_1, σ_1)	…	Feature n (μ_n, σ_n)
…				
Class m+1 (m bipolar dimensions where $n \geq m$)	Prototype			
	Feature 1 $(\mu_1{}^1, \sigma_1{}^1, \mu_1{}^2, \sigma_1{}^2)$	Feature 2 $(\mu_2{}^1, \sigma_2{}^1, \mu_2{}^2, \sigma_2{}^2)$	…	Feature n (μ_n, σ_n)

For each multipolar dimension, the class prototype has a feature structure which keeps the distribution information of all existing clusters. In addition to the mean and variance values of the clusters, the feature is assigned a '#' label for multipolarity. Table 4.2 shows the structure of class prototype and feature architecture for the single prototype approach.

Since it is much more efficient than the previous approach we adopt the single prototype approach in our implementations for this thesis.

4.3 Detection of Multipolarity

After running RGNG algorithm and receiving the label prototype clusters, at the stage of assigning labels to prototype features, detection of multipolarity procedure is applied. Features which are labeled with '*' symbol have potential to display multipolarity. Therefore the labeling decision of assigning a '*' symbol or a '#' symbol is taken at this point of the prototype derivation method.

If a feature is labeled with ‘*’ symbol in the pre-labeling stage using the label prototypes derived with RGNG algorithm, its distribution information is extracted and used for determining whether it has two clusters displaying consistent and meaningful behavior or the pre-assignment made a correct labeling by assigning the ’*’ symbol. Using the "peak detection function", a possible multiple peak occurrence is detected and labeling is finalized for the prototype feature.

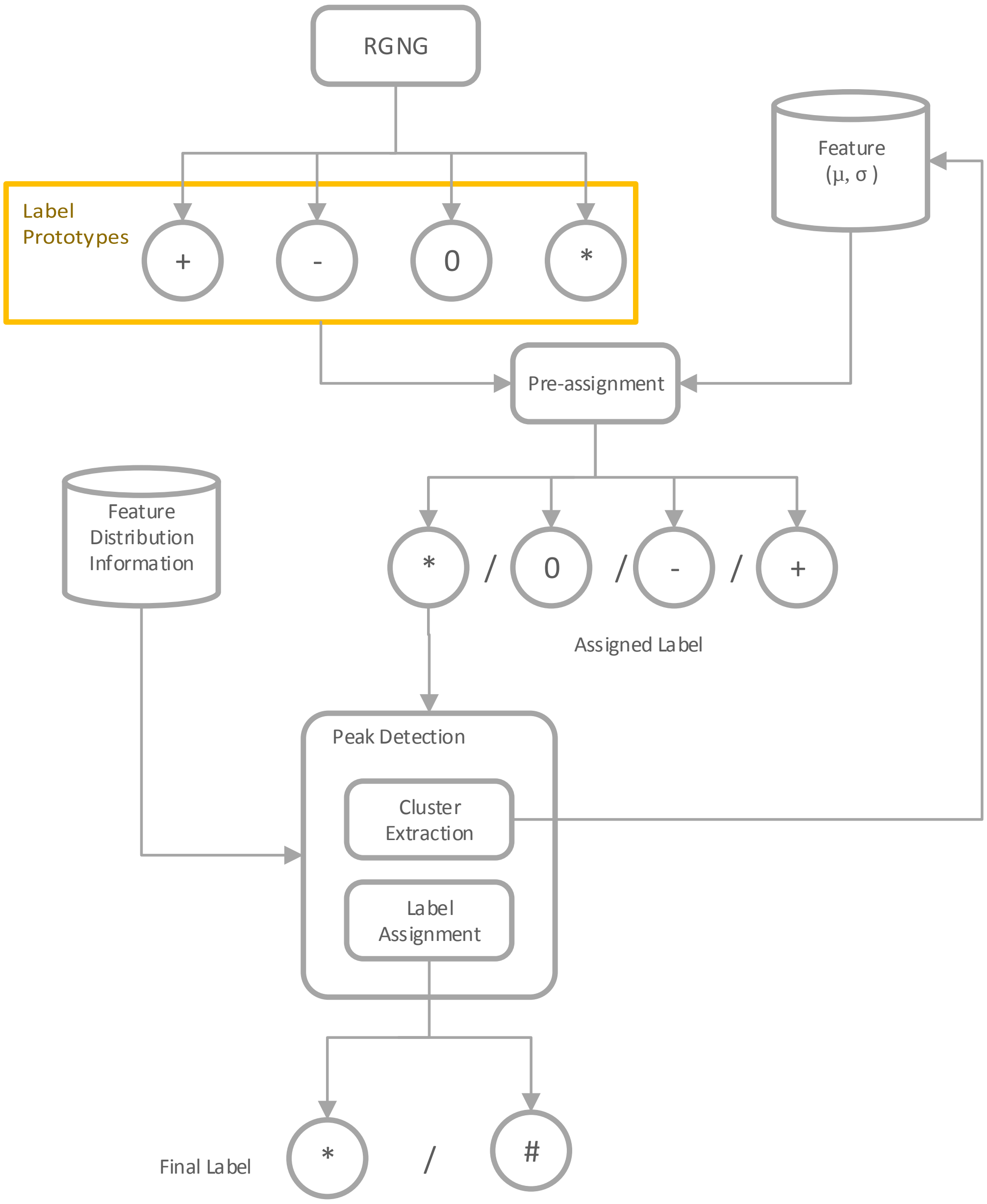

Figure 4.6: Overview of bipolarity detection

An overview of multipolarity detection and labeling process of a feature is provided in Figure 4.6 and the peak detection function is described in detail in the next section.

4.3.1 Peak Detection

Peak detection is one of the key points of this study. The scope of this thesis includes analyzing the existence of peaks in feature dimensions, therefore if two peaks are detected in a feature data, this indicates that bipolarity is detected for the corresponding dimension of the class prototype.

Algorithm 6 DBSCAN Algorithm [6].

```
Input: Dataset $D$, neighborhood radius $\epsilon$,
Input: minimum elements requried to form a cluster $minPts$.
Output: Resultant clusters.
  procedure DBSCAN
    $C = 0$.
    for all unvisited point $P$ in $D$ do
      Set $P$ visited.
      Set $NbrPts$ using $SetRegion(P, \epsilon)$ as in Algorithm 7.
      if $sizeof(NbrPts) < MinPts$ then
        Label $P$ to be noise.
      else
        Set $C$ as next cluster.
        Extend the cluster using $ExtendCluster(P, NbrPts, C, \epsilon, MinPts)$
  as in Algorithm 8.
      end if
    end for
  end procedure
```

A peak detection function is described in Algorithm 9 for examining features and detecting possible peaks if multipolarity occurs. Let f be a prototype feature, F be the set of feature values that f represents. The function takes F as input and it outputs the multipolarity information. This information includes a Boolean value of whether multipolarity is detected or not and if it is detected, μ_1, σ_1, μ_2, σ_2, ...,μ_n, σ_n values

Algorithm 7 SetRegion Algorithm.

```
Input: Point $P$, neighborhood radius $\epsilon$.
Output: Region points.
    procedure SETREGION
        Find the points around $P$ within $\epsilon$.
        Include $P$ and return the points.
    end procedure
```

Algorithm 8 ExtendCluster Algorithm.

```
Input: Point $P$, neighbor points $NbrPts$, cluster $C$, neighborhood radius $\epsilon$, $MinPts$.
Output: Extended cluster.
    procedure EXTENDCLUSTER
        Add point $P$ to cluster $C$.
        for all $P'$ in $NbrPts$ do
            if $P'$ is not processed then
                Set $P'$ visited.
                Set $NbrPts'$ using $SetRegion(P', \epsilon)$ as in Algorithm 7.
                if $sizeof(NbrPts') >= MinPts$ then
                    Increment $NbrPts$ with $NbrPts'$.
                end if
            end if
            if $P$ does not belong to any cluster then
                Set $P'$ as member of $C$.
            end if
        end for
    end procedure
```

indicating mean of the first cluster, variance of the first cluster, mean of the second cluster, variance of the second cluster, mean of the n^{th} cluster and variance of the n^{th} cluster respectively.

The first step is applying the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [6] on the set of feature instances F. Given the neighborhood radius ϵ and minimum number required to form a cluster, algorithm detects the peaks, in other words clusters, in the feature set. The advantage here is that, with this method, we can detect any number of clusters hidden inside the feature set. GMM method, however, needs to know the number of desired clusters before trying to find them. Another advantage is that DBSCAN is very successful on low dimensional data. Although our learning algorithm aims to be successful on high dimensional datasets as well as low dimensional ones, this peak detection module inspects each dimension alone. This way DBSCAN runs on one dimensional data and successfully returns the detected clusters. DBSCAN is briefly explained in Algorithm 6.

Neighborhood radius ϵ is estimated by a comparison among the densities of the original dataset and another uniformly distributed dataset having same size and same volume. The calculation can be showed as in Equation (4.10):

$$\epsilon = \sqrt{\frac{Vk\Gamma(\frac{1}{2}n+1)}{m\sqrt{\pi^n}}}, \tag{4.9}$$

$$V = \prod_{i=1}^{n}\{\max x_i - \min x_i\}, \tag{4.10}$$

where ϵ, m, n, Γ, V are the neighborhood radius with k objects, the number of elements in the dataset, dimensionality of the dataset, the gamma function and the volume of the space of m elements. Gamma function Γ is described as in the following equation:

$$\Gamma(n) = (n-1)!. \tag{4.11}$$

Although radius estimation can be achieved internally without requiring an input, ϵ can be specified manually. For different domains it can be set to exact values for further precision. In other words, ϵ is an optional parameter.

If the algorithm finds more than one cluster in the dataset, this indicates possible multipolarity. We propose approval criterion for the multipolarity decision that comes from DBSCAN algorithm. If this criterion is also fullfilled, it is decided that the given dataset has multipolarity in the examined feature set, which means there exists more than one feature pattern. The values of μ_1, σ_1, μ_2, σ_2, ... , μ_n, σ_n are attached to the output of the function in addition to the boolean value indicating multipolarity in the corresponding feature set, where n is the number of peaks detected.

Algorithm 9 Peak Detection Algorithm.

```
Input: Set of instance values $F$ belonging to a certain feature $f$.
Output: Bipolarity information
1: procedure PEAK DETECTION
2:     Estimate or manually specify neighborhood radius $\epsilon$.
3:     Apply DBSCAN algorithm on $F$ with $\epsilon$.
4:     if $somePerformanceMeasurement$ holds then
5:         Extract the mean and variance values of clusters found by WinnerModel.
6:         Store the mean and variance values as $\mu_1$, $\sigma_1$, $\mu_2$, $\sigma_2$.
7:         Normalize $\mu_1$, $\sigma_1$, $\mu_2$, $\sigma_2$ w.r.t the training data boundaries.
8:     end if
9: end procedure
```

4.3.2 Performance Measurement

After the peak scan part described in Section 4.3.1, if the model offering multiple clusters is chosen, the clusters are judged with respect to some additional performance criteria which is proposed in this thesis.

With the presence of noise in the data, DBSCAN usually gives successful results separating the outlier points by labeling them as so. However,it may still not be adequate to take the decision of multipolarity depending on the context of the problem and the characteristics of the dataset used. To resolve this problem, we define a threshold value called the "ratio threshold", T_r.

Let c_i, N, r_i be the i^{th} cluster of the chosen model, the total number of instances in both clusters and the ratio between the number of instances in c_i and N. N and r_i are given in Equations (4.12) and (4.13) respectively.

$$N = \sum_{i=1}^{2} |c_i|, \tag{4.12}$$

$$r_i = \frac{|c_i|}{N}, \tag{4.13}$$

where $i = 1, 2$ and $|c_i|$ is the cardinality of the set c_i.

To distinguish a meaningful cluster from noise, there has to be some balance between the cardinalities of the clusters. Therefore, each cluster should satisfy the condition in Equation (4.14):

$$r_i > T_r. \tag{4.14}$$

T_r takes a predefined value and it can be optimized with respect to the type of the problem and the dataset used for training and testing.

Another issue with the approval of bipolarity is that the concepts of peak can be perceived differently with respect to different requirements of different problems. In other words, after a normalization procedure, the variance values of the clusters should be checked using a predefined threshold value which we call the "variance threshold", T_σ . Therefore clusters should also satisfy the condition in Equation (4.15):

$$\widehat{\sigma_i} < T_\sigma, \tag{4.15}$$

where $i = 1, 2$ and $\widehat{\sigma_i}$ is the normalized variance value of c_i.

As well as T_r, this predefined value of T_σ can be optimized considering the characteristics of the problem and the dataset for accuracy manners.

If both clusters satisfy the conditions mentioned above, they are considered as valid peaks and the detected bipolarity is approved.

4.4 Classification

Given an instance vector, to be able to find the most similar class prototype, the distances between the instance vector and all class prototypes should be measured and compared to each other. The distance measurement function we propose in this thesis is an improved distance function based on Mahalanobis distance. We improved the original method to fit our prototype models. Since our prototype derivation algorithm makes use of prototype labels, the distance function should be modified so that the characteristics information carried by feature labels is utilized.

The feature information for the ones with the label '*' is omitted because if the feature displays an unpredictable pattern, it is not wise to rely on it. Therefore, the corresponding feature information is omitted for distance calculations.

A feature that has the '#' label on it means that we have more than one different cluster information stored in the feature structure. Unlike in the case of '*' label, these clusters are very important for calculating an accurate distance.

Let x be an input vector, p be the prototype that x is compared to , f_i be the i^{th} feature of p and f_i^x be the i^{th} feature of x, where $i = 1, \ldots, N$ and N is the number of features in a prototype vector. If f_i has the label '#' indicating that it corresponds to a multipolar dimension, there are multiple candidates one of which is a better match for f_i^x. Therefore the distances from f_i^x to the clusters of f_i are calculated and the cluster with the smaller distance is the winner. The prototype distribution information is updated by adopting the distribution information of the winner cluster. The updating operations are done locally and are not permanent. Because the cluster we adopt for modification changes with respect to different input vectors. After manipulating the features and the information they carry, we apply Mahalanobis distance and return the result.

$$MD(x,p) = \sqrt{(\mu^x_{f_i} - \mu^x_{p_i})^T C_p^{-1} (\mu^x_{f_i} - \mu^x_{p_i})}, \tag{4.16}$$

where C is the covariance matrix of the given feature set.

Algorithm 10 Modified Mahalanobis Distance.

Input: instance vector x, prototype p

Output: distance between the given instance vector and the prototype vector, $dist(x, p)$

```
 1: procedure MODIFIED MAHALANOBIS
 2:     for all feature $f_i$ in prototype $p$ do
 3:         if $label_{f_i}$ equals '*' then
 4:             Omit feature $f_i$ from calculations.
 5:         end if
 6:     end for
 7:     for all feature $f_i$ in prototype $p$ do
 8:         if $label_{f_i}$ equals '#' then
 9:             for all $cluster_k$ in $clusters$ do
10:                 Calculate distance between $f_i^{cluster_k}$ and $f_i^x$, store it as $dist_k$.
11:             end for
12:             Determine $max?(dist_1, dist_2, ..., dist_k)$, store winner's index as $j$.
13:             Update $\mu_{f_i}$ with $\mu$ of $cluster_{j_{f_i}}$.
14:             Update $\sigma_{f_i}$ as $\sigma$ of $cluster_{j_{f_i}}$.
15:         end if
16:     end for
17:     Update covariance matrix $C_p$.
18:     By considering all dimensions, calculate the distance using Equation (4.16)
19: end procedure
```

CHAPTER 5

RESULTS & EVALUATION

This chapter includes descriptions of the datasets we used for this study, the tests and the comparison with other methods. We have implemented / re-implemented nine other algorithms for comparison with the algorithm we developed and proposed in this thesis. Test results and observations are discussed.

5.1 Datasets

In this section we describe the datasets which we experimented on for investigation of the performance of the proposed and re-implemented algorithms. The datasets are:

1. IRIS Dataset (IRIS)
2. Dataset of Effect Prototypes (DEP)
3. Short-Tall Dataset (STD)
4. Dataset of Effect Prototypes with Bipolarity (DEPB)
5. Reduced Dataset of Effect Prototypes with Bipolarity (RDEPB)
6. Synthetic Dataset with High Multipolarity (DMP)
7. Wine Dataset (WD)
8. Seeds Dataset (SEED)

Some of the basic properties that describe these datasets are summarized in Table 5.1.

Table 5.1: Summary of the datasets used in experimental work.

Dataset	Number of Classes	Number of Features (Attributes)	Labeled	Contains Missing Values	Average Number of Instances in Each Class
IRIS	3	4	Yes	No	50
DEP	7	67	Yes	No	40
STD	2	92	Yes	No	210
DEPB	6	67	Yes	No	40
RDEPB	4	3	Yes	No	41
DMP	6	3	Yes	No	154
WD	3	13	Yes	No	60
SEED	3	7	Yes	No	70

5.1.1 IRIS Dataset

Iris dataset is one of the popular datasets used in the field of Machine Learning. It is created by R.A. Fisher and taken from UCI Machine Learning Repository [1]. The classes in this dataset refer to the types of iris plant. Attributes are numerical and each refers to characteristics for defining the plant type.

We include tests with this dataset in our study because of comparability and reproducibility purposes since it is a well-known and relatively simple dataset. Other datasets we use in our tests are more domain specific and more complicated than IRIS Dataset.

This dataset contains 3 classes with 50 instances in each class. Each instance has 4 dimensions corresponding to iris flower features.

5.1.2 Dataset of Effect Prototypes

This dataset is created by Kalkan et al. and used in the study [21]. It was created in KOVAN Research Lab which is one of the major research labs in METU Computer Engineering Department and it is used in many other studies as well.

iCub humanoid robot [28] which is designed for cognitive and developmental robotics research is used to generate this dataset. The robot is physically designed in the form of a 4-year old child. iCub's perception of objects is assisted by a Kinect RGB-D camera and sensorimotor data corresponding to perceptual features of objects including surface features, spatial features and object presence are used in the dataset.

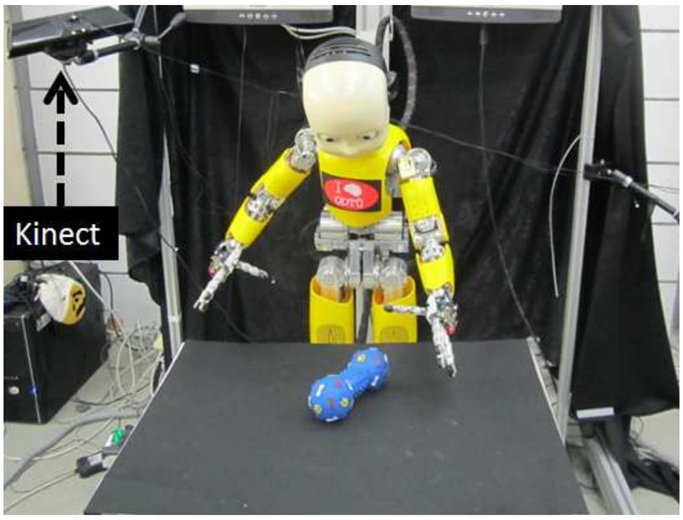

Figure 5.1: iCub humanoid robot.

As it is also explained in Section 3.9, to be able to manipulate the objects, the robot applies behaviors like "Push Right", "Push Left", "Push Forward", "Pull", "Top Grasp" and "Side Grasp". As a consequence of these behaviors, changes in object features occur and then they are recorded. These changes correspond to the features of the effect instances which are used in this dataset. Effects have labels including "Moved Right", "Moved Left", "Moved Forward" etc. and they correspond to the classes of the dataset.

This dataset includes 7 classes, containing 311 instances in total. Each instance has 67 dimensions corresponding to object features. We refer this dataset as DEP in short.

5.1.3 Short-Tall Dataset

This dataset is also created in KOVAN research lab as well as the previously introduced dataset DEP [30]. In our study, we use this dataset to compare several algorithms which we developed and re-implemented. This dataset is also applicable for the SVM algorithm as well as the Multi-SVM algorithms since it has 2 classes and requires binary classification.

This dataset includes 2 classes with 420 instances in total. Each instance has 92

dimensions corresponding to object features. We refer this dataset as STD in short.

5.1.4 Dataset of Effect Prototypes with Bipolarity

Dataset of Effect Prototypes (DEP), which is presented in the previous subsection consists of instances with mostly unipolar dimensions and bipolar dimensions do not have significance. To be able to test the bipolarity/multipolarity detection capability of our proposed algorithm which is called Multinomial Prototype-based Learning (MNPBL), we proposed a new effect which is called “Moved Sideways”. This effect is created by either moving the object to the left side or moving it to the right side. Since both behaviors create effects which are equally valid to be classified as “Moved Sideways”, we created a dataset by merging “Moved Left” and “Moved Right” instances of DEP. This dataset shows stronger multipolarity characteristics in their dimensions compared to the DEP dataset. With this dataset, the aim is to test our algorithms multiple pattern detection abilities by adding multipolarity characteristics to some of the dimensions.

This dataset includes 6 classes, containing 265 instances in total. Each instance has 67 dimensions corresponding to object features. We refer this dataset as DEPB in short.

5.1.5 Reduced Dataset of Effect Prototypes with Bipolarity

This dataset is also created for testing multipolarity in dimensions. Unlike the previous ones it is reduced so that bipolar/multipolar dimensions have more emphasis compared to DEP and DEPB datasets. Both the dimension count and class count is decreased, leaving the ones which carry multipolarity characteristics. With this dataset, the aim is to test the multiple pattern detection ability of our algorithm for the existence of higher multipolarity levels.

This dataset includes 4 classes with 163 instances in total. Each instance has dimensions with reduced count to 3 for the purpose of amplifying the weight of the multipolar dimensions. We call this dataset RDEPB in short.

5.1.6 Synthetic Dataset with a High Level of Multipolarity

This dataset is created for testing high level of multipolarity and includes up to 4 peaks in class dimensions. Classes correspond to different patterns in 3-D space. An important property of this dataset is that different classes seem to have very similar dimension means. Therefore it is very hard to distinguish them without revealing the different patterns inside each dimension. Visual representations are given in Figures 5.2 and 5.3 for a better understanding of the dataset and the patterns of the classes.

This dataset includes 6 classes, containing 924 instances in total. Each instance has 3 dimensions corresponding to position features in 3-D space.

We refer this dataset as DMP in short.

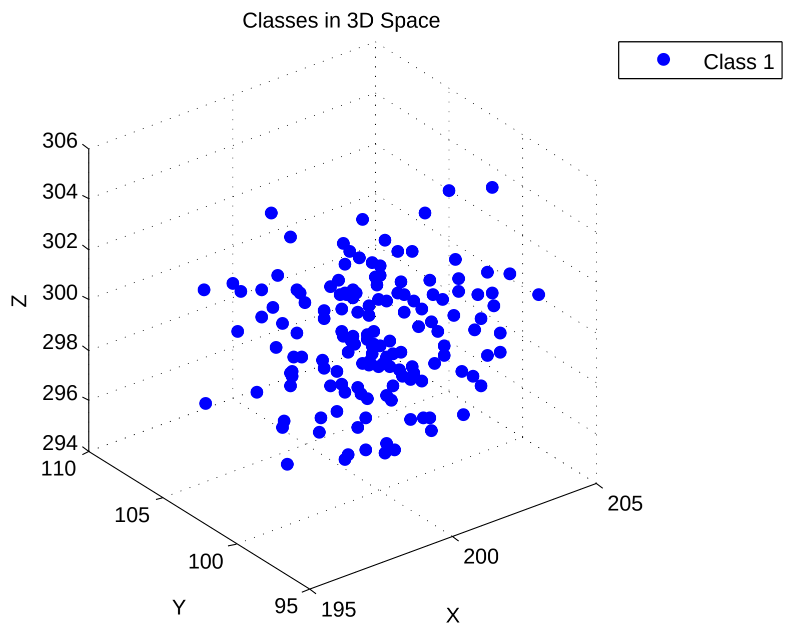

Figure 5.2: Visualization of Class 1 in 3-D space. The dataset is a synthetic one with high degree of multipolarity for testing the multiple pattern sensitivity of the algorithms.

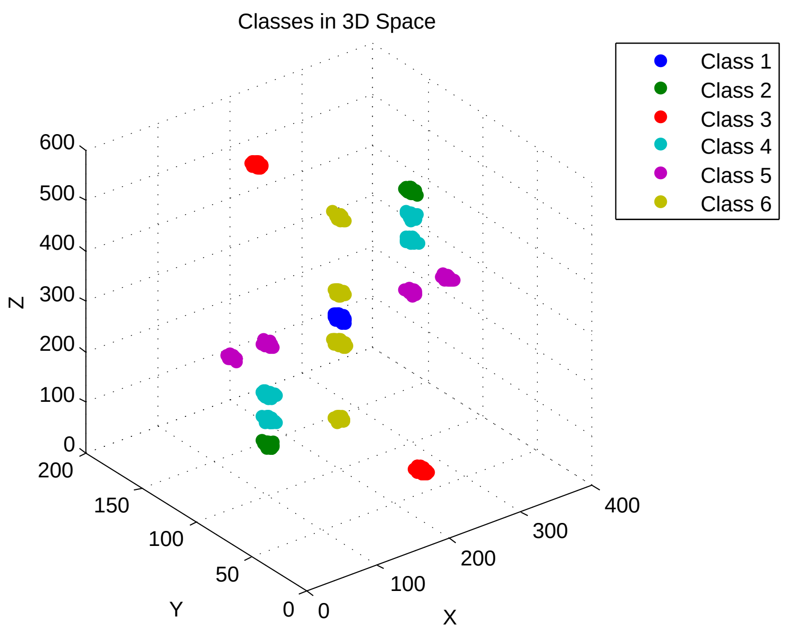

Figure 5.3: Visualization of the entire dataset in 3-D space. The dataset is a synthetic one with high degree of multipolarity for testing the multiple pattern sensitivity of the algorithms.

5.1.7 Wine Dataset

This dataset is originally created by Forina, M. et al and donated to UCI Machine Learning Repository [1] in 1991. The data contains chemical analysis of wines and the attributes correspond to the chemical and physical features like magnesium, phenols, alcohol, color intensity etc. for different types of wines.

The dataset includes 3 classes, containing 178 instances in total. Each instance has 13 dimensions.

We call this dataset WD in short.

5.1.8 Seeds Dataset

This dataset is donated to UCI Machine Learning Repository [1] in 2012 by Charytanowicz, M. and Niewczas, J. from The John Paul II Catholic University of Lublin, Piotr Kulczycki, Piotr A. Kowalski, Syzmon Lukasik, Slowomir Zak from Cracow

University of Technology and Systems Research Institute, Polish Academy of Sciences. The dataset enables analysis on different types of kernels which belong to different types of wheats. The attributes correspond to the kernel features such as area, perimeter, asymmetry coefficient etc.

The dataset includes 3 classes, containing 210 instances in total. Each instance has 7 dimensions.

We call this dataset SEEDS in this thesis.

5.2 Evaluation Metrics

We have divided this section into two subsections to be able to give more details about the experimental work of this thesis. In Section 5.3, we investigate the algorithms that we propose and re-implemented, compare them in terms of accuracies, learning curves and running times on different datasets that have different properties and then we discuss the results. As well as overall performances of algorithms, we also compare the results in class level, with respect to f-measure, precision and recall values for the classes of datasets. Precision is the fraction of retrieved instances which are relevant and recall can be described as the fraction of relevant instances that are retrieved. F-measure is the harmonic average of precision and recall. Let number of true positives, number of true negatives, number of false positives, number of false negatives be TP, TN, FP, FN respectively. Then, precision, recall, f-measure and accuracy can be calculated by Equations (5.3), (5.1), (5.2), (5.4). The relation between accuracy and precision can be shown as in Figure 5.4.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5.1}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5.2}$$

$$\text{f-measure} = 2\frac{(\text{precision})(\text{recall})}{\text{precision} + \text{recall}}, \tag{5.3}$$

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{5.4}$$

In Section 5.3.1, we present a case study of handling multipolar dimensions. Looking only at the accuracy results gives an idea about the overall performance of our algorithm among the others but sometimes it is not enough to understand the dominance of our algorithm in terms of multipolarity handling. Therefore, we compare our proposed algorithm MNPBL with DOP by further analyzing the internal steps during classification and explain the improvements on multipolar dimension handling with numbers from experimental cases. This leads to a better understanding of how our algorithm works and gets better results by coping with multiple pattern challenges in dimensions.

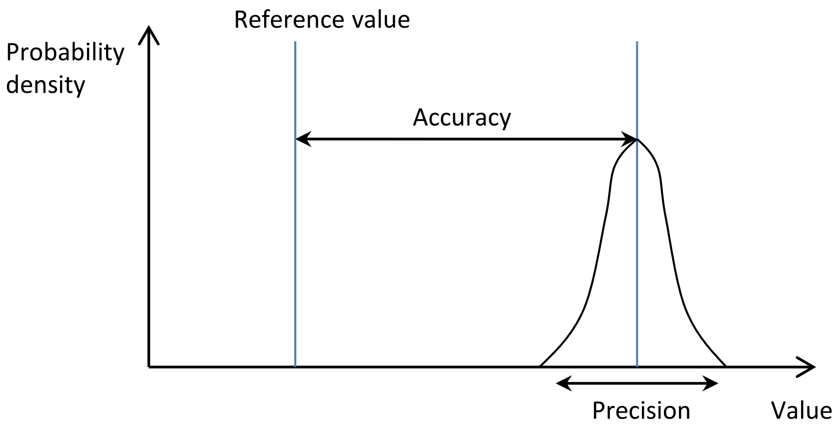

Figure 5.4: Relation between accuracy and precision.

5.3 Comparison with Other Methods

The aim of this comparison is to test different algorithms, including ours, with different datasets and observing their behaviors that they show in different cases. We have experimented on 8 datasets (IRIS, STD, DEP, DEPB, RDEPB, DMP, WD and SEED) with 10 different algorithms (LVQ, Multi-SVM with linear kernel, Multi-SVM with polynomial kernel, Multi-SVM with quadratic kernel, Multi-SVM with RBF kernel, AdaBoost, k-NN, Decision Tree Learning, DOP and MNPBL) and then discuessed the results we derived from these experiments.

For our experiments we used 10-fold cross-validation for our accuracy, f-measure, precision and recall evaluations. We emprically set some values for parameters of the methods we used. LVQ networks that we used was optimezed with 15 to 150 epochs and the class weights were distributed equally. For SVM algorithms, we used 3000 as max iteration count. We used MNPBL to estimate the epsilon value for neighborhood radius when determining the number of peaks in dimensions. Variance threshold T_σ was taken as -0.8 and the ratio threshold T_r was taken as 0.1 for the peak detection mechanism of MNPBL algorithm. For k-NN algorithm we set $k = 3$. AdaBoost algorithm with ensemble cycle value of 100 was used. $AdaBoost.M1$ and $AdaBoost.M2$ algorithms were applied depending on the dataset whether it needs a binary classification or multi-class classification. Pruning was used when applying Decision Tree Learning algorithm with a pruning criterion of 'error' and the minumum parent size value was set to 10 for branch node observations.

The datasets we have choosen display differences in terms of attribute count which refers to the number dimensions of the space worked on, the number of classes, instance count etc. Since we test different algorithms with different characteristics, we are able to observe their behavior with the changing properties of the datasets. But more importantly, we test the algorithms with datasets having different levels of multipolarity and with the obtained results we show that our algorithm, MNPBL, has comparable performance competing the other algorithms investigated in this thesis. MNPBL's performance is high in terms of accuracy, precision, recall and f-measure on datasets with unipolar dimensions, moreover thanks to its multipolarity handling mechanism, the significance of its performance increases with the increasing level of multipolarity in dimensions. Table 5.2 shows the accuracy results.

During our tests, we also measured the algorithm running times for training and compared them. Learning curve analysis is also a part of our experiments for discussing the generalization performances of the algorithms.

Table 5.2: Average accuracies of algorithms (%). 10-fold cross-validation is used. (M=mean,SD=standard deviation)

		IRIS	STD	DEP	DEPB	RDEPB	DMP	WD	SEED	Overall
LVQ	M	92.14	84.04	12.26	17.31	73.89	16.67	34.12	88.57	52.38
	SD	*5.03*	*14.36*	*2.14*	*2.46*	*2.83*	*3.5e-15*	*3.53*	*11.31*	*5.21*
M-SVM (Linear)	M	75.00	**98.25**	79.35	55.77	49.44	16.67	93.53	90.48	69.81
	SD	*11.18*	*5.26*	*7.38*	*8.11*	*1.67*	*3.5e-15*	*6.14*	*8.78*	*6.07*
M-SVM (Poly.)	M	91.43	97.37	71.29	66.54	**99.44**	93.33	91.18	92.38	87.87
	SD	*6.99*	*6.76*	*7.28*	*9.59*	*1.67*	*8.17*	*7.56*	*7.44*	*6.93*
M-SVM (Quad.)	M	**95.71**	93.86	73.23	72.30	**99.44**	91.55	92.94	**93.33**	89.05
	SD	*5.71*	*10.79*	*6.46*	*13.30*	*1.67*	*7.46*	*6.34*	*6.10*	*7.23*
M-SVM (RBF)	M	95.00	**7**3.51	43.87	44.62	**99.44**	91.56	84.12	91.43	77.94
	SD	*7.18*	*21.78*	*7.24*	*6.01*	*1.67*	*7.46*	*5.29*	*8.19*	*8.10*
k-NN	M	**95.71**	93.68	80.97	81.92	**99.44**	**100.0**	87.65	89.05	91.05
	SD	*6.55*	*9.68*	*3.37*	*7.31*	*1.67*	*0.00*	*7.18*	*11.31*	*5.88*
D-Tree	M	**95.71**	93.68	**83.87**	**87.31**	98.89	**100.0**	91.77	91.43	**92.83**
	SD	*5.71*	*7.78*	*3.23*	*5.18*	*2.22*	*0.00*	*9.19*	*7.32*	*5.08*
AdaBoost	M	95.00	96.32	71.29	76.15	**99.44**	74.78	**94.12**	90.48	87.20
	SD	*5.58*	*5.79*	*4.66*	*9.23*	*1.21*	*7.59*	*6.44*	*8.52*	*6.13*
DOP	M	93.57	80.35	65.81	71.92	25.00	38.33	91.18	92.86	69.88
	SD	*5.93*	*14.01*	*18.43*	*4.57*	*6.60*	*5.69*	*6.58*	*4.88*	*8.34*
MNPBL	M	94.29	80.70	**74**.19	**77.**69	**99.44**	94.11	91.18	92.86	88.06
	SD	*5.35*	*13.79*	*11.63*	*5.91*	*1.67*	*5.60*	*6.58*	*4.88*	*6.93*

Our algorithm MNPBL shows good accuracy results in general. It outranks many other algorithms on several datasets. On IRIS dataset, with 94.29% accuracy, MNPBL is more successful than DOP, Linear-kernel SVM, Polynomial-kernel SVM and LVQ. There is only 1.42% difference between the accuracy of MNPBL and the highest accuracy achieved on this dataset. Highest accuracies are achieved by k-NN and Decision Tree Learning algorithm while AdaBoost and RBF-kernel SVM has 95% accuracy rate. Since IRIS dataset is a simpler one compared to many other datasets and does not have high levels of multipolarity, this result shows that MNPBL competes well with other algorithms on datasets with unipolar dimensions on different domains. Since they are both prototype based algorithms with similar structure, DOP also has similar high accuracies except for the dataset with increased multipolarities. For IRIS dataset, DOP achieved 93.57%. The largest standard deviation observed on IRIS dataset belongs to Linear-kernel SVM with the value of 11.18 and the smallest belongs to LVQ with 5.03 after 10-fold cross validation. The results on IRIS dataset are visualized in Figure 5.5.

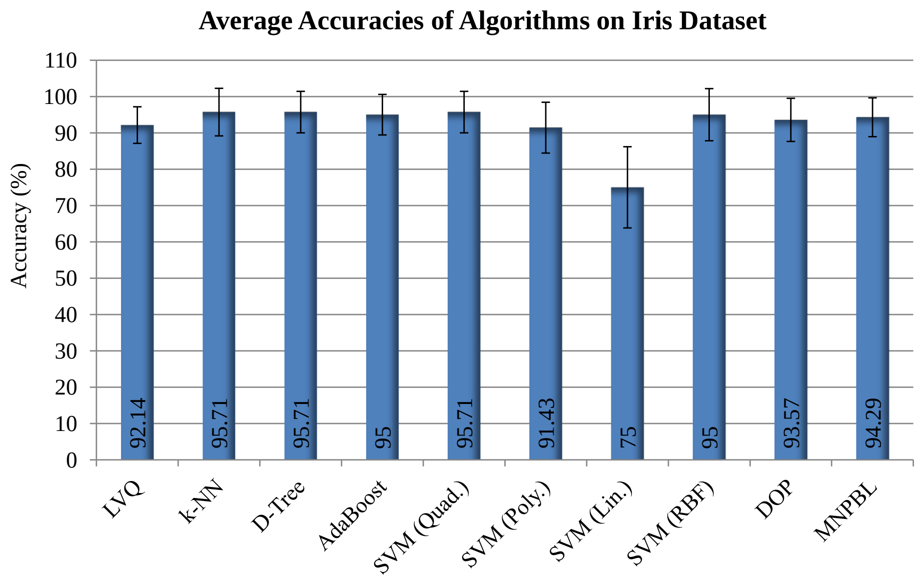

Figure 5.5: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on IRIS dataset. 10-fold cross validation is used.

On STD dataset, MNPBL has over 80% accuracy and outranks DOP and RBF-kernel Multi-SVM algorithm while LVQ has slightly better performance than our algorithms. Other algorithms have accuracies over 90% and Linear kernel-SVM is the most successful algorithm on this dataset. The results on STD dataset is visualized in Figure 5.6.

For the DEP dataset and the DEPB dataset which has increased multipolarity characteristics compared to DEP, MNPBL has promising results with accuracies of 74.19% and 77.69 respectively. For DEP dataset, MNPBL is ranked 4^{th} among 10 algorithms and for DEPB dataset it outranked the Linear-kernel Multi-SVM algorithm and climbed to 3^{th} place. The reason is Linear-kernel Multi-SVM experienced a significant performance decrease with the increased multipolarity from DEP to DEPB. On these datasets, Decision Tree Learning and k-NN algorithms are the first and second most successful ones respectively and they are the only two algorithms that slightly outranks MNPBL on DEPB dataset. Similar to our algorithm MNPBL, Decision Tree Learning and k-NN are not affected by the changing level of multipolarity in the datasets. AdaBoost also achieved good performance on both datasets and not experienced a performance decrease after the multipolarity level change in the datasets. Among the different kernel Multi-SVM's, quadratic kernel Multi-SVM is the most robust one for DEP and DEPB datasets although liner kernel Multi-SVM is the one with highest accuracy on the DEP dataset, when multipolarity level is low. Polynomial-kernel Multi-SVM displayed also good and consistent performance which is slightly under quadratic-kernel Multi-SVM. However RBF-kernel is not successful on either datasets. The lowest accuracy rates belong to LVQ on both datasets. The results on DEP and DEPB datasets are visualized in Figures 5.7 and 5.8 respectively.

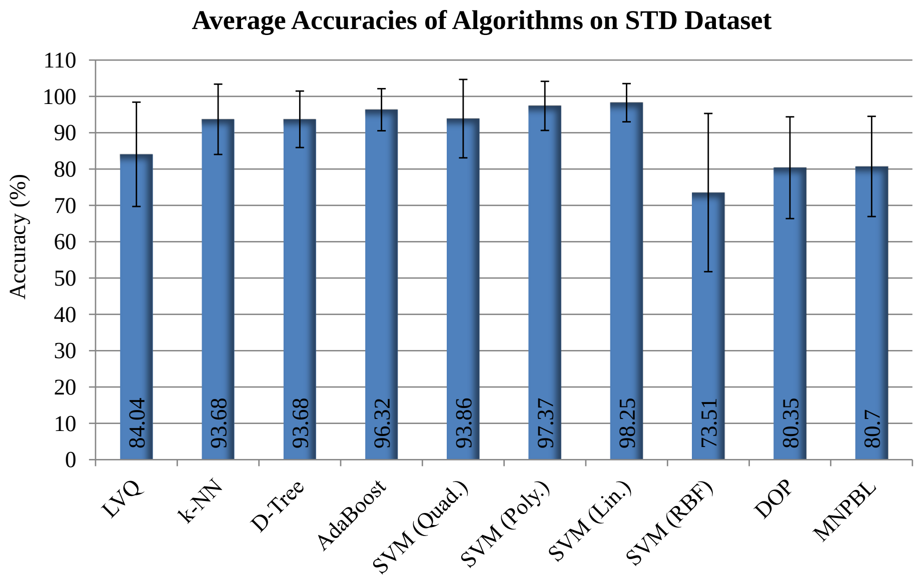

Figure 5.6: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on STD dataset. 10-fold cross validation is used.

For RDEPB dataset, which is a reduced dataset featuring multipolarity, all algorithms other than linear-kernel SVM, DOP and LVQ had very successful results.Although it was not as successful as k-NN, Decision Tree Learning, AdaBoost, quadratic-kernel, polynomial-kernel,RBF-kernel Multi-SVMs and MNPBL, LVQ increased its performance because of the fact that RDEPB is reduced. However, for the same reason, DOP had experienced a dramatic performance decrease. Reducing the dimensions resulted in wrong predictions for DOP because of its multipolarity sensitivity. Linear-kernel SVM also displayed sensitivity of multipolarity on RDEPB. Having 99.44%

average accuracy rate and a standard deviation value of 1.67, MNPBL shared the 1^{st} place in terms of accuracy with the other kernel versions of SVMs, AdaBoost and k-NN. The results on RDEPB dataset are visualized in Figure 5.9.

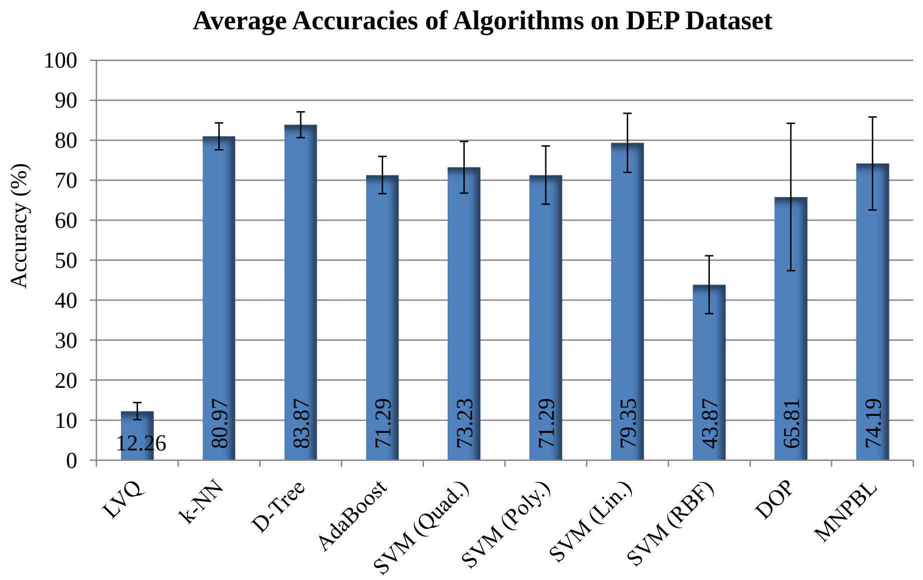

Figure 5.7: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on DEP dataset. 10-fold cross validation is used.

The differences between MNPBL and DOP in accuracies becomes more significant when the unipolar features are excluded from the dataset because this way there is more emphasis on bipolar dimensions and their weight on the overall judgment is increased. For the dataset DEPB, there exists certain amount of multipolarity with the “Moved Sideways” class we proposed in this thesis. Although MNPBL has higher accuracy than DOP on this dataset by doing more accurate calculations for the multipolar dimensions, the difference is still not very dramatic. This is because DEPB dataset has a lot of unipolar dimensions and the bipolarity level is not high enough to compromise its success severely. However, on the RDEPB dataset the MNPBL is far more successful than DOP because DOP perceives multipolarity as unpredictability and this time the weights of multipolar dimensions are larger than before.

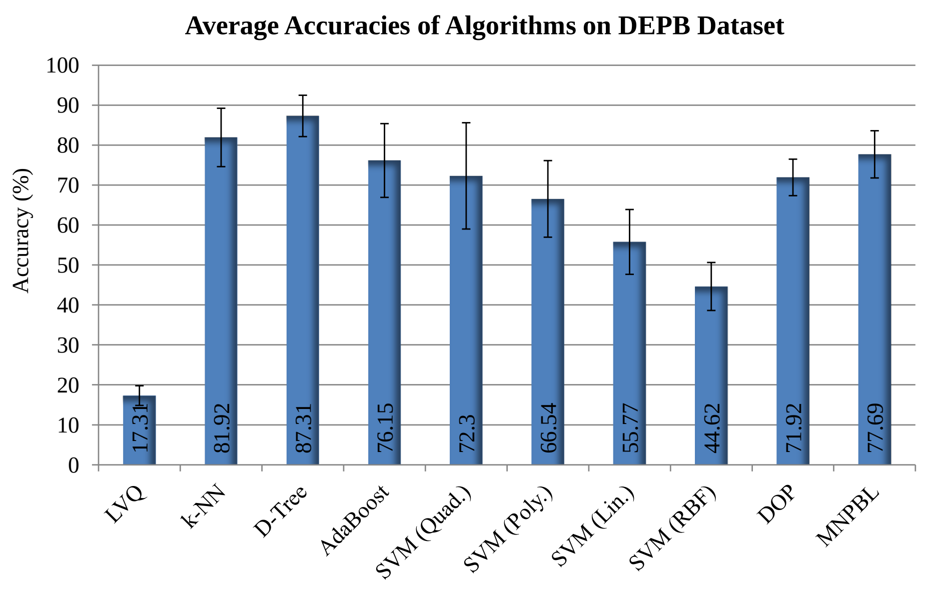

Figure 5.8: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on DEPB dataset. 10-fold cross validation is used.

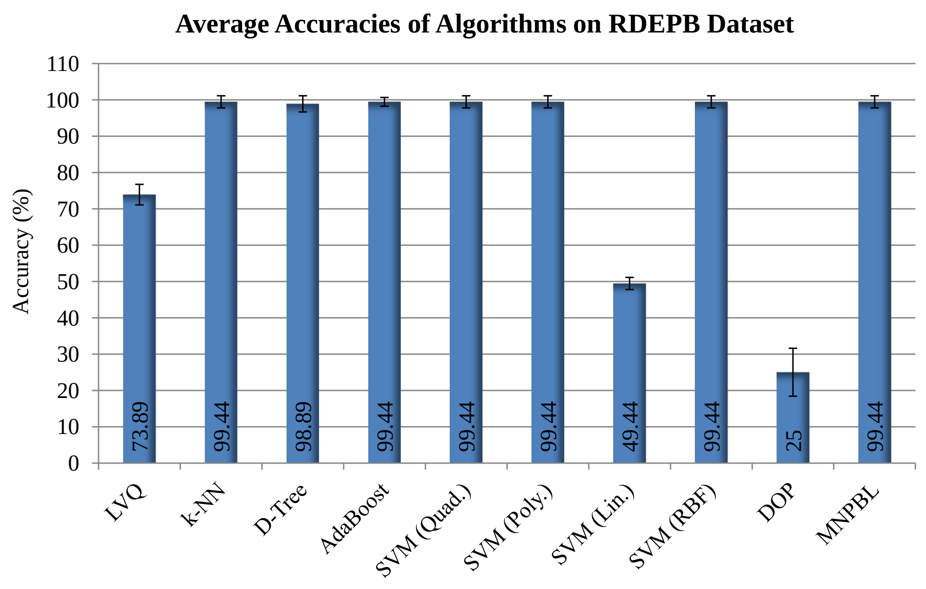

Figure 5.9: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on RDEPB dataset. 10-fold cross validation is used.

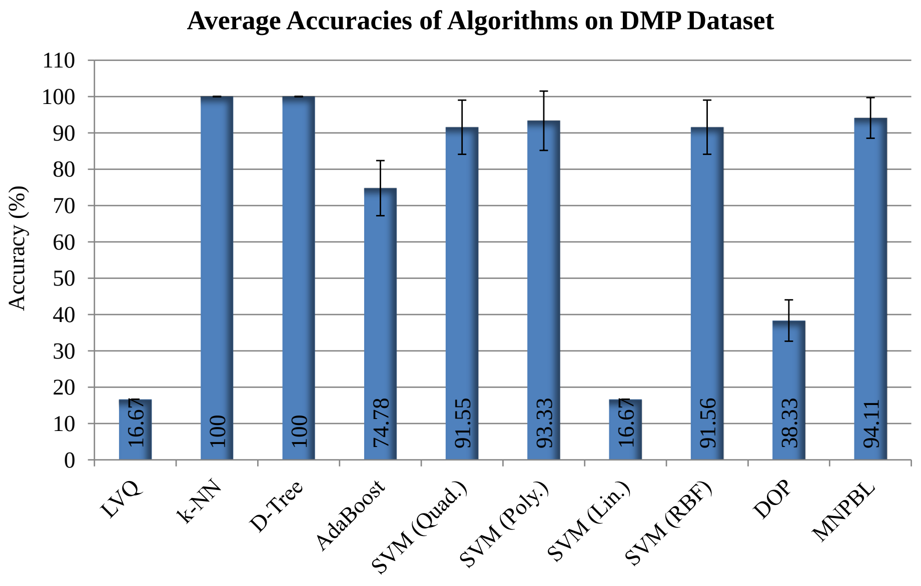

Figure 5.10: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on DMP dataset. 10-fold cross validation is used.

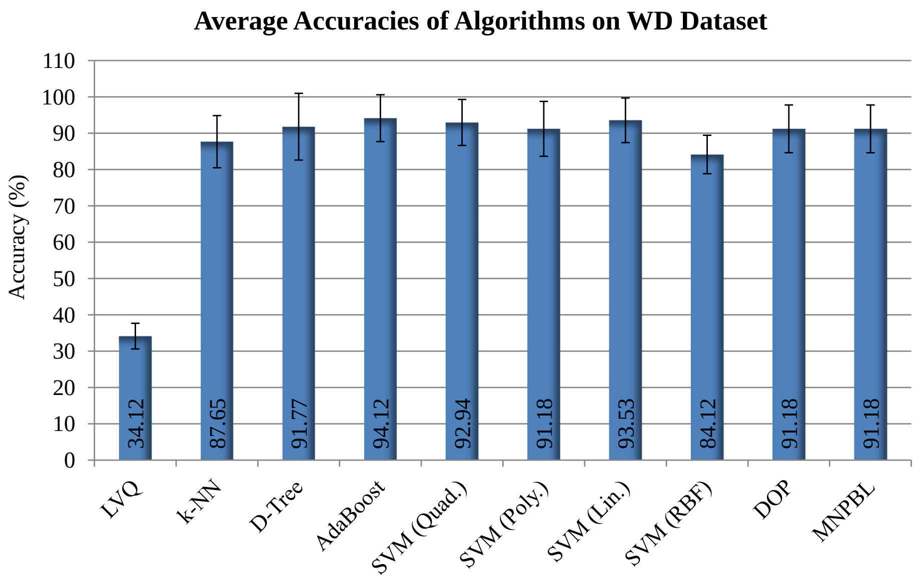

Figure 5.11: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on WD dataset. 10-fold cross validation is used.

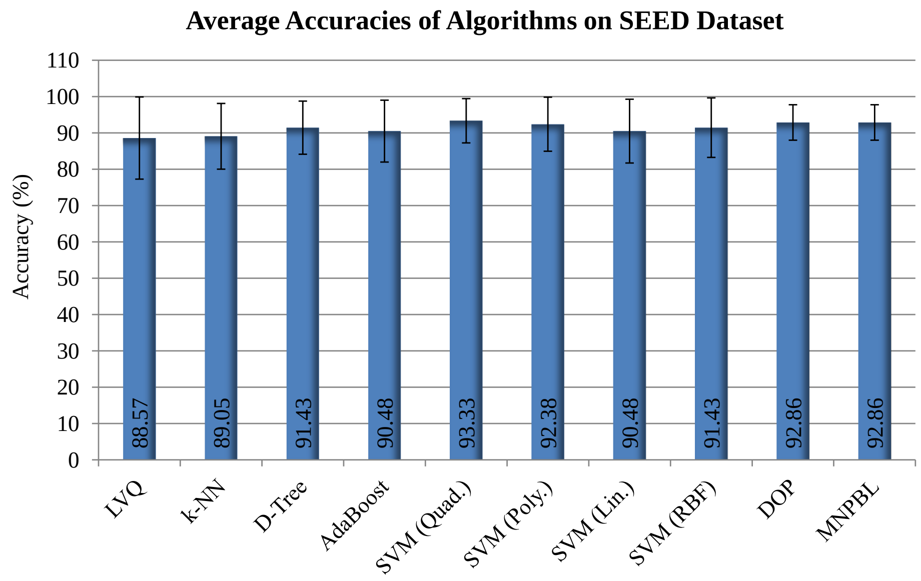

Figure 5.12: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on SEED dataset. 10-fold cross validation is used.

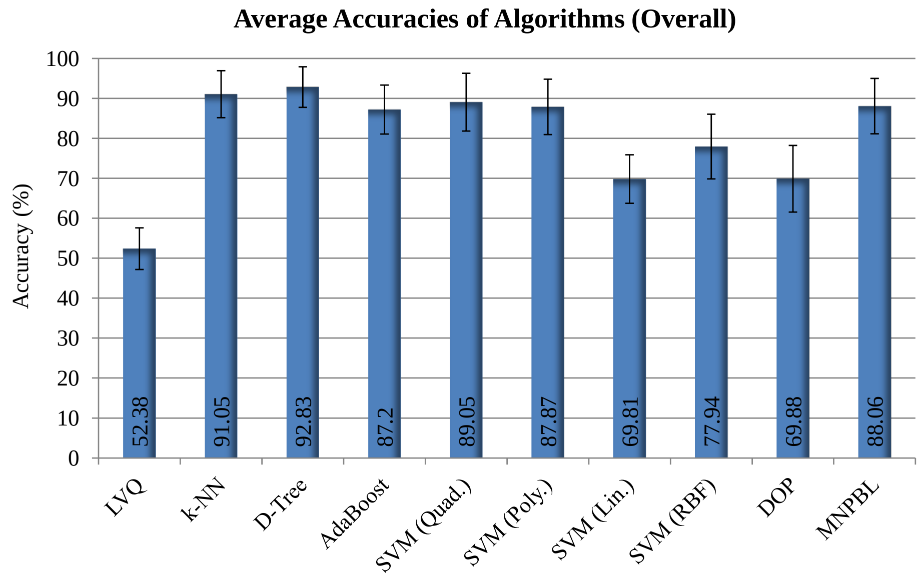

Figure 5.13: Average accuracies of algorithms LVQ, Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL considering all of 8 datasets. 10-fold cross validation is used.

DMP dataset has 6 classes and each class has different patterns with similar mean values in their dimensions, which means high level of multipolarity. Therefore, DOP displayed a low accuracy result on this dataset. Likewise, linear-kernel Multi-SVM and LVQ was not successful. After k-NN and Decision Tree Learning algorithm, MNPBL was ranked 3^{rd} with 99.11% accuracy rate which reveals that it is a highly promising result. After MNPBL, RBF-kernel, quadratic-kernel and polynomial-kernel acheived to be 4^{th}, 5^{th}, 6^{th} respectively with very close success rates while AdaBoost could manage to have 74.78%. The average accuracy results on DMP dataset are visualized in Figure 5.10 and the f-measure values of the algorithms for each class of DMP dataset is shown in Figures 5.14 and 5.15.

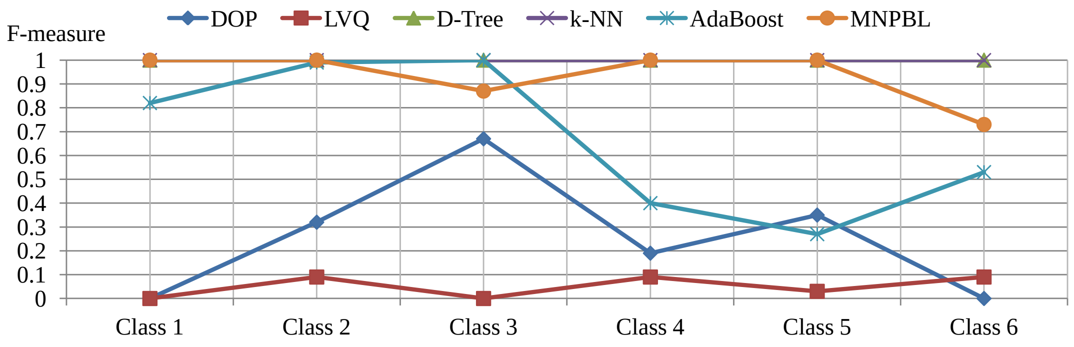

Figure 5.14: Average f-measure values of the algorithms MNPBL, DOP, LVQ, Decision Tree Learning, k-NN and AdaBoost for each class in DMP dataset. 10-fold cross validation is used.

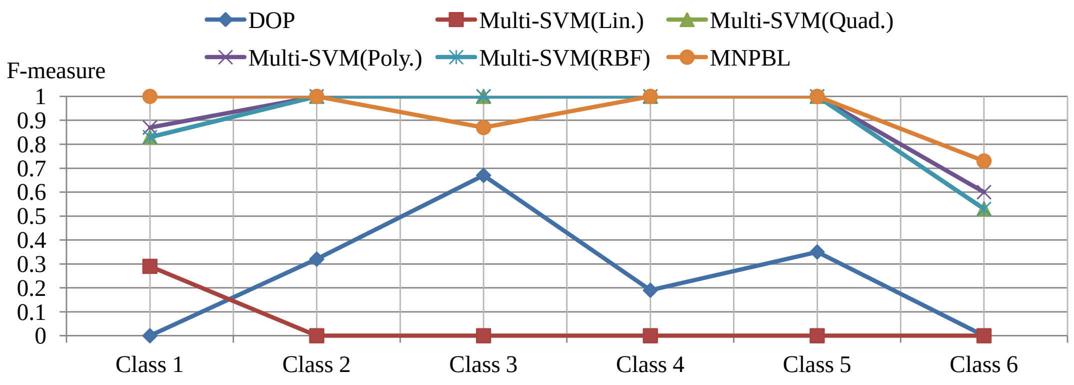

Figure 5.15: Average f-measure values of the algorithms MNPBL, DOP, linear-kernel Multi-SVM, quadratic-kernel Multi-SVM, polynomial-kernel Multi-SVM and RBF-kernel Multi SVM for each class in DMP dataset. 10-fold cross validation is used.

LVQ was not successful also on WD dataset but the other algorithms had high accuracy rates with close values. AdaBoost had the 1^{st} place with 94.12% accuracy rate while MNPBL achieved to have 91.18% which is also a very high success rate. Among the SVM kernels the most successful one was linear-kernel with 93.53% rate and the least successful one was RBF-kernel with 84.12% rate. The results on WD dataset are displayed in Figure 5.11.

On SEED dataset MNPBL achieved an accuracy rate of 92.86% which is very high and ranked 2^{nd} after quadratic-kernel Multi-SVM with 93.33% rate. The difference is only below 0.47% and MNPBL has smaller standard deviation with the value of 4.88. The most successful SVM kernel on SEED dataset was quadratic-kernel and the least successful was linear-kernel with 90.48% which is also a good accuracy result. In this case, AdaBoost and Decision Tree Learning algorithms were below the Multi-SVM and MNPBL averages but they still achieved to have success rates over 90%. k-NN and LVQ had accuracy rates slightly under 90%. The results on SEED dataset are displayed in Figure 5.12.

Overall results on all 8 datasets reveal that LVQ, DOP, linear-kernel Multi-SVM and RBF-kernel Multi-SVM algorithms are not as stable as other algorithms. RBF and linear kernel, DOP are sensitive to multipolarity in datasets. However although it fluctuates a lot, LVQ's performance is not very much dependent on multipolarity. MNPBL has an overall accuracy rate of 88.06% with an average standard deviation value of 6.93 when 10-fold cross validation is applied. It is not sensitive to multipolarity and it is competitive on all datasets outranking many other algorithms each time. For the datasets we used, the overall best algorithm in terms of accuracy rates is Decision Tree Learning algorithm with an accuracy rate of 92.83% and a standard deviation value of 5.08. The difference between Decision Tree Learning and MNPBL is below 5% and our algorithm outranked 6 of 9 other algorithms on the average of 8 datasets, including DOP, linear-kernel Multi-SVM, RBF-kernel Multi-SVM, polynomial-kernel Multi-SVM, LVQ and AdaBoost. These results shown in Figure 5.13 prove that our algorithm has promising results among different types of learning algorithms on different datasets.

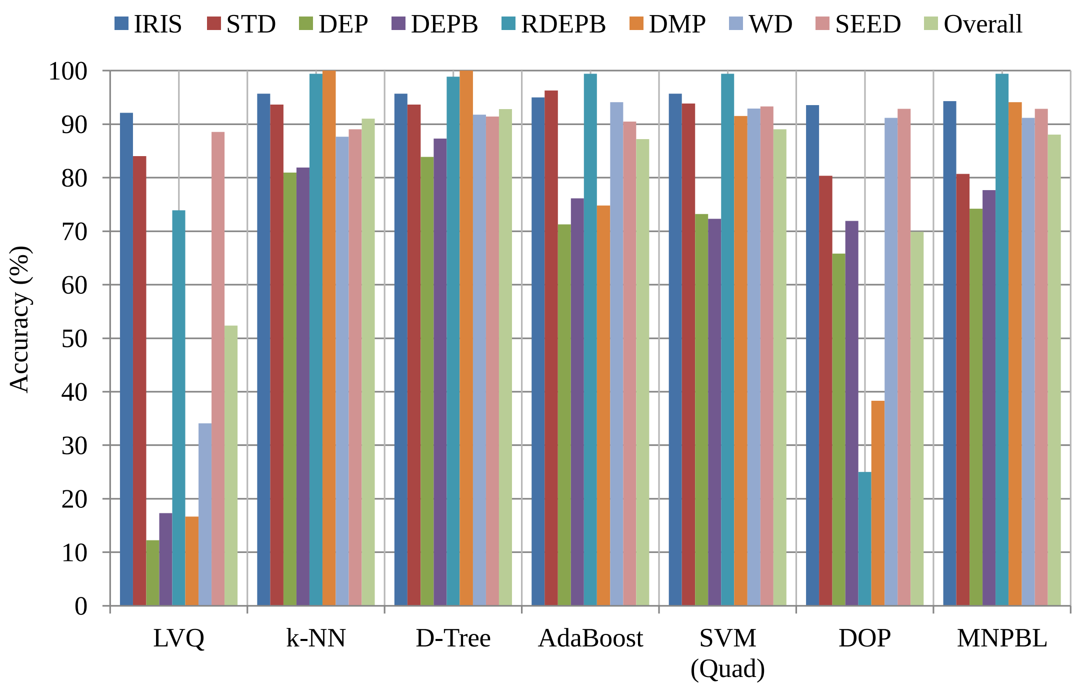

Figure 5.16: Average accuracies of algorithms LVQ, Quadratic kernel Multi-SVM, k-NN, Decision Tree Learning, AdaBoost, DOP and MNPBL on datasets IRIS, STD, DEP, DEPB, RDEPB and DMP. 10-fold cross validation is used.

Table 5.3: Average training times of algorithms (in milliseconds). 10-fold crossed validation is used on datasets.

		IRIS	STD	DEP	DEPB	RDEPB	DMP	WD	SEED	Overall
LVQ	M	5034.60	20371.60	11058.10	9524.60	6666.00	31956.30	6323.70	8310.90	12405.72
	SD	*335.50*	*925.20*	*72.60*	*325.30*	*887.10*	*646.00*	*642.90*	*1088.70*	*615.41*
M-SVM (Linear)	M	55.80	196.70	1318.10	1124.40	126.40	1425.10	209.60	131.50	573.45
	SD	*17.60*	*24.40*	*239.30*	*285.80*	*13.80*	*120.00*	*31.50*	*38.60*	*96.38*
M-SVM (Poly.)	M	95.90	97.20	355.50	249.00	66.80	447.50	85.40	307.60	213.11
	SD	*16.00*	*13.70*	*31.20*	*8.70*	*11.00*	*66.00*	*2.00*	*61.40*	*26.25*
M-SVM (Quad.)	M	81.20	141.40	374.70	265.00	58.40	272.30	142.70	186.50	190.28
	SD	*23.40*	*15.80*	*14.30*	*23.90*	*14.40*	*18.70*	*14.90*	*28.40*	*19.23*
M-SVM (RBF)	M	55.90	431.10	431.10	305.00	56.20	304.50	125.90	76.70	223.30
	SD	*2.70*	*320.10*	*20.40*	*8.90*	*12.30*	*16.10*	*17.50*	*2.80*	*50.10*
k-NN	M	172.40	**8.60**	**8.40**	**105.80**	**32.20**	**10.60**	**8.60**	**9.30**	**44.49**
	SD	*1.60*	*5.30*	*0.20*	*0.30*	*0.20*	*0.70*	*0.60*	*0.90*	*1.23*
D-Tree	M	**14.30**	16.10	22.20	304.90	33.70	15.60	10.90	11.90	53.70
	SD	*1.80*	*0.70*	*3.00*	*0.80*	*01.00*	*0.20*	*4.10*	*0.50*	*1.51*
AdaBoost	M	1537.10	1325.80	1155.60	1332.40	980.00	1218.30	1208.80	1252.20	1251.27
	SD	*270.70*	*207.00*	*8.10*	*17.20*	*1210.90*	*28.20*	*743.80*	*658.10*	*393.00*
DOP	M	56.10	640.80	1468.90	1234.30	99.80	81.90	121.80	80.90	473.06
	SD	*9.30*	*366.20*	*16.40*	*16.40*	*8.70*	*8.50*	*10.70*	*2.50*	*54.84*
MNPBL	M	73.10	797.90	1519.00	1804.20	117.20	148.80	130.40	97.70	586.04
	SD	*8.90*	*29.40*	*399.10*	*19.30*	*11.90*	*9.30*	*11.70*	*9.50*	*62.39*

As it is summarized in Table 5.3, in terms of running times for training, LVQ has the longest by far for all datasets, training the network brings a noticeable amount of running time cost. The second logest training time belongs to AdaBoost algorithm with an average time of 1251.27 milliseconds considering all of the datasets. MNPBL and linear-kernel Multi-SVM have very close running times with 586.04 and 573.45 milliseconds respectively. Other kernels for Multi-SVM algorithm run with smaller values for training times.

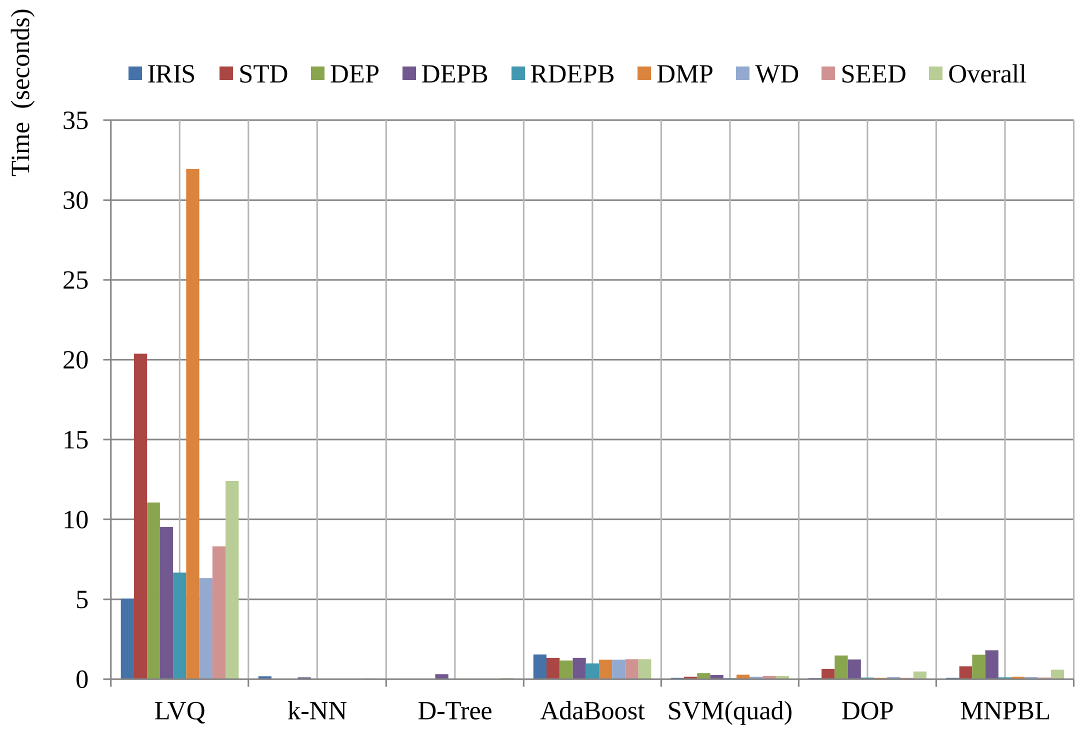

Figure 5.17: Average running times of algorithms for training on DEPB. The values are in milliseconds.

The smallest value of training time belongs to k-NN algorithm which is briefly explained in Algorithm 1, having an average time of 44.49 milliseconds with a value of 1.23 for standard deviation. Decision Tree Learning algorithm, which has the best average accuracy rate on all datasets, has 53.70 milliseconds of average running time and this result makes it the second fastest algorithm of all in terms of training time. Among SVM kernels the fastest is the quadratic kernel and the slowest is the linear kernel for our experiments.

For all datasets we have tested, the running time comparison result is shown in Equation (5.5).

$$\begin{aligned} t_{LVQ} > t_{AdaBoost} > t_{MNPBL} > t_{LinearSVM} > t_{DOP} > t_{RBFSVM} > \\ t_{PolynomialSVM} > t_{QuadraticSVM} > t_{D-Tree} > t_{k-NN}, \end{aligned} \tag{5.5}$$

where t_{LVQ}, $t_{AdaBoost}$, t_{MNPBL}, $t_{LinearSVM}$, t_{DOP}, t_{RBFSVM}, $t_{PolynomialSVM}$, $t_{QuadraticSVM}$, t_{D-Tree} and t_{k-NN} are the training run times of LVQ, AdaBoost, MNPBL, linear-kernel Multi-SVM, DOP, RBF-kernel Multi-SVM, polynomial-kernel Multi-SVM, quadratic-kernel Multi-SVM, Decision Tree Learning algorithm and k-NN algorithm respectively. The comparison is illustrated in Figure 5.17.

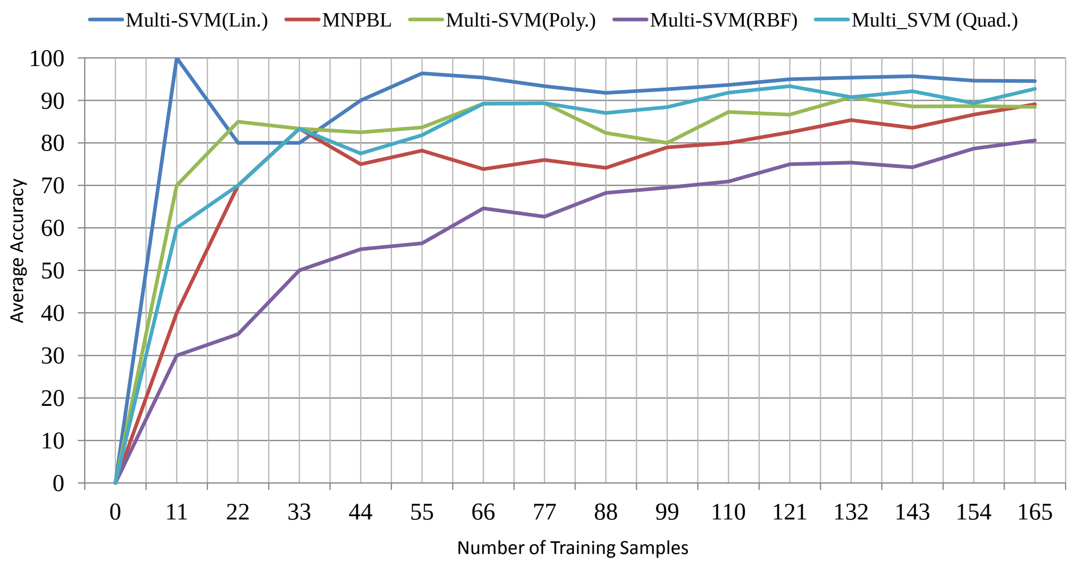

Figure 5.18: Learning curves of Linear kernel Multi-SVM, Polynomial kernel Multi-SVM, Quadratic kernel Multi-SVM, RBF kernel Multi-SVM and MNPBL on WD dataset. 5-fold cross validation is used.

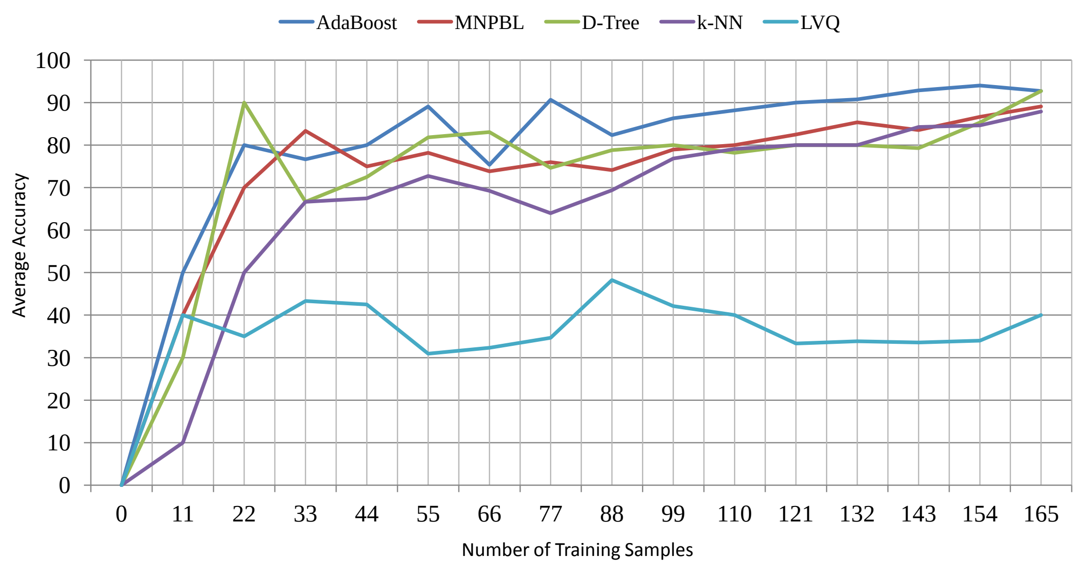

Figure 5.19: Average accuracies of algorithms LVQ, k-NN, Decision Tree Learning, AdaBoost and MNPBL on WD dataset. 5-fold cross validation is used.

To be able to do learning curve analysis, we sampled the SEED dataset with changing sizes. For each sample we used 5-fold cross validation and extracted the learning behavior with respect to the changing data size. MNPBL has once again displayed a competitive performance for this test. It learns and stabilizes quicker than LVQ, k-NN and RBF-kernel Multi-SVM algorithm. The other algorithms have quicker but very similar responses to the change in data size.

AdaBoost and MNPBL has nearly the same tangent value up to 80% accuracy rate but then MNPBL stabilizes better while AdaBoost experiences fluctuations until 50% of data size is reached. Comparing MNPBL with Decision Tree Learning algorithm, they reach 50% accuracy rate at the same data size but until that point, MNPBL runs with higher rate. At around 13.33% of data size, Decision Tree Learning algorithm reaches its first local maximum with 90% before MNPBL reaches its first local maximum with 83.33% accuracy rate at 20% of data size. However, at the same point (20% data size), Decision Tree Learning algorithm's accuracy drops to 66.66% accuracy rate which is below MNPBL's performance. Then the two algorithm stabilizes with minor fluctuations. Among SVM kernels, linear kernel reaches its first local maximum before than any other kernels. RBF kernel learns slower and needs more data reach a maximum. However, together with MNPBL all other kernels stabilizes after around 20% of data size. These behaviors visualized in Figure 5.18 reveal that MNPBL has a competitive generalization performance compared to AdaBoost, Decision-Tree, k-NN and LVQ.

5.3.1 Case Study

In this section, we analyze the algorithms, DOP and MNPBL and show how they behave differently because of their different ways of feature analysis. We derived "Moved Sideways" prototypes with both DOP and MNPBL algorithms on the dataset DEPB. By introducing a "Moved Sideways" effect instance to the both of the algorithms, we investigated the classification step.

5.3.1.1 Prototype Derivation Phase

During prototype derivation process, MNPBL finds the clusters in an multipolar feature set, while DOP treats it as one chunk of data. Therefore, they build different prototypes for "Moved Sideways" effect. Let w_{DOP}, w_{MNPBL}, $f_y^{w_{DOP}}$, $f_y^{w_{MNPBL}}$ be the prototype generated by DOP, the prototype generated by MNPBL, feature of w_{DOP} corresponding to y axis change and ,feature of w_{MNPBL} corresponding to y axis change respectively. The detailed information about $f_y^{w_{DOP}}$ and $f_y^{w_{MNPBL}}$ is given in Tables 5.4 and 5.5.

Table 5.4: $f_y^{w_{DOP}}$, y axis feature of DOP prototype for "Moved Sideways" effect.

$f_y^{w_{DOP}}$ Feature Data	
Mean	0.4184
Variance	0.1021
Normalized Mean	0.1633
Normalized Variance	0.5292
Number of Instances	41
Label	‘*’

With the help of RGNG, DOP decides that the feature $f_y^{w_{DOP}}$ does not have reliable information for a class description. Therefore it assigns ’*’ label to that feature.

Since DOP do not have the ability to detect the peaks in a dimension, the mean and variance values are misleading values for this case. Derived variance value indicates unpredictability although "Moved Sideways" class has two distinct predictable behaviors. The extracted mean value of $f_y^{w_{DOP}}$ lies in a position between the mean values derived by MNPBL. Therefore, this single mean value of 0.4184 does not reflect none of the two behaviors that "Moved Sideways" class displays on y axis. However, DOP algorithm is aware of the fact that it has extracted unreliable mean and variance values. Thus, it assigns the label which causes omission of this misleading data for the corresponding feature in distance calculation process.

MNPBL detects that the input data corresponding to $f_y^{w_{MNPBL}}$ does not display a single consistent behavior as a first step. Then, it further investigates the data for a final decision, whether it is one unpredictable cluster of inconsistent values or it has two clusters of consistent values. By applying Algorithm 10, MNPBL concludes that the feature $f_y^{w_{MNPBL}}$ has two consistent clusters and assigns the '#' label.

Table 5.5: $f_y^{w_{MNPBL}}$, y axis feature of MNPBL prototype for "Moved Sideways" effect.

$f_y^{W_{MNPBL}}$ Feature Data		
	Cluster 1	Cluster 2
Mean	0.1390	0.7117
Variance	0.0011	0.0012
Normalized Mean	-0.7220	0.4234
Normalized Variance	-0.9950	-0.9947
Number of Instances	20	21
Label	'#'	

5.3.1.2 Classification Phase

We introduce a test instance x and its feature f_y^x with a normalized value of 0.5075 and classify it with both of the algorithms. Since they have different prototypes, different distances are calculated by the algorithms.

DOP omits f_y^x and $f_y^{w_{DOP}}$ for distance calculation because it labels $f_y^{w_{DOP}}$ with '*' symbol indicating that "Moved Sideways" class has unpredictable behavior on y-axis. The overall distance from x to w_{DOP} is calculated to be 8.6631 by DOP. However, MNPBL finds the best matching cluster in its prototype w_{MNPBL} for x, by comparing the existing clusters.

Let $dist_1(f_y^x, f_y^{w_{MNPBL}})$ and $dist_2(f_y^x, f_y^{w_{MNPBL}})$ be the distances from the clusters of $f_y^{w_{MNPBL}}$ to f_y^x and they are calculated as:

$dist_1(f_y^x, f_y^{w_{MNPBL}}) = 1.2295$

$dist_2(f_y^x, f_y^{w_{MNPBL}}) = 0.0841$

Since $dist_2(f_y^x, f_y^{w_{MNPBL}}) < dist_1(f_y^x, f_y^{w_{MNPBL}})$, the cluster with the normalized mean value of 0.4234 is chosen as the best matching cluster and $dist_2(f_y^x, f_y^{w_{MNPBL}})$ is decided as the distance from f_y^x to $f_y^{w_{MNPBL}}$. The overall distance from x to w_{MNPBL} is calculated to be 6.7104 by MNPBL. A comparison of DOP and MNPBL in terms of their distance calculations is shown in Table 5.6 and Figure 5.20.

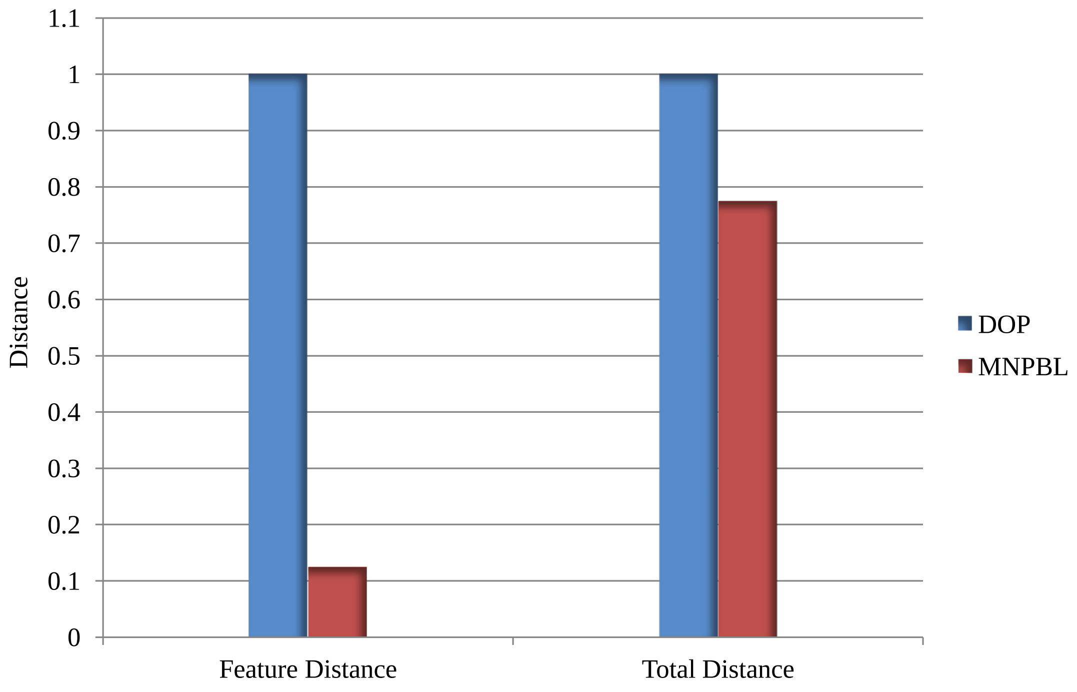

Figure 5.20: Normalized feature distances and total distances between the Input Vector and the prototypes of DOP and MNPBL.

Although DOP has a clever mechanism for avoiding incorrect information, it still suffers from lack of information. This has a significant impact on results especially for cases where more than one prototype compete for classification of an instance vector and the competition depends on that feature distance calculation which is omitted. As an illustration of such cases, in Figure 5.16, DOP experiences a noticeable decrease in accuracy for RDEPB dataset, after exhibiting successful accuracy rates on DEP and DEPB datasets, while MNPBL is stable for all them.

Since we examine the performances of DOP and MNPBL on DEPB dataset in this case study, the accuracy rates of both algorithms are observed to be similar .However, we show that even if the overall results seem to be similar, MNPBL is more precise in distance calculation, benefiting its improved prototype structure and the modified distance calculation that we propose. They are explained in Table 4.2 and Algorithm 10 in detail, respectively.

Table 5.6: Feature distances and total distances calculated by DOP and MNPBL for "Moved Sideways" instance.

	Feature Distance	Total Distance	Normalized Feature Distance	Normalized Total Distance
DOP	0.67 (omitted)	8.66	1 (omitted)	1
MNPBL	0.08	6.71	0.13	0.77

CHAPTER 6

CONCLUSION

In this chapter, we conclude the thesis with a summary of the contributions, the limitations of the current work and the list of potential directions.

6.1 Summary

In this thesis, we propose a prototype based learning method which derives prototypes out of training data by examining the feature dimension values. Making use of the RGNG algorithm, these values are clustered and labeled with symbols such as '+', '-', '*', '#' etc. The purpose of these label prototypes is to describe the distribution properties of class prototypes. However, when a class dimension has multiple separate group of values, each of them having steady behaviors such as consistently increasing or no change, this causes a problem for the prototype-based learning since the corresponding feature is misclassified by assigning '*' label. We overcome this problem in our method by detecting multipolar dimensions and extracting the clusters out of it. A set of performance criteria is applied on the extracted clusters to verify that they are meaningful. If they are clarified after the performance measurement test, a prototype feature structure which stores the distribution information of the extracted clusters is formed. We assign '#' symbol to multipolar dimensions to signify multipolarity such that the classification method which we developed is able to sense the multipolarity in the feature dimension and chooses the correct cluster for making the most accurate distance calculation.

We developed one and re-implemented 9 different learning algorithms for comparison in the scope of this thesis:

- LVQ : A well-know prototype based algorithm
- SVM : A well-known binary classifier.
- Multi-SVM, which is an SVM method with the capability of classifying more-than two classes, using different kernel functions.
  - Linear kernel
  - Quadratic kernel
  - Polynomial kernel
  - Radial Basis Function (RBF) kernel
- AdaBoost: An ensemble learning method which uses weak learners to boost learning performance.
- Decision Tree Learning: A learning method which describes target functions as decision trees.
- k-NN: An instance-based supervised learning method.
- A prototype based method which we call DOP, presented and used in the study [21], by Kalkan and his colleagues.
- The improved prototype based algorithm that we developed for this study. We call it Multinomial Prototype-based Learning (MNPBL).

We test our algorithm on 8 different datasets. Some of them are created for this study and some of them are derived from other studies and UCI Machine Learning Repository [1]. Therefore, we are able to present the performance of the algorithms on variety of datasets.

- Iris Dataset (IRIS)
- Dataset of Effect Prototypes (DEP)
- Short-Tall Dataset (STD)

- Dataset of Effect Prototypes with Bipolarity (DEPB)
- Reduced Dataset of Effect Prototypes with Bipolarity (RDEPB)
- Synthetic Dataset with Multipolar Dimensions (DMP)
- Wine Dataset (WD)
- Seeds Dataset (SEED)

Our algorithm, MNPBL, produces a solid performance in terms of accuracy and f-measure, regardless of the multipolarity levels of datasets, and shows a promising performance on 8 different datasets. DOP, which is the predecessor, being sensitive to multipolarity, experiences a severe performance decrease on datasets such as RDEPB; however, on the other unipolar datasets, it achieves promising accuracy rates overall. With our improvements, we solved the multipolarity problems of DOP and developed a robust algorithm which achieves compelling classification results on different types of datasets.

LVQ, which is also a prototype based algorithm, is sensitive to initialization, and it is not as robust as other algorithms such as MNPBL, Multi-SVM and k-NN. Multi-SVM performs well with convincing accuracy rates on all of the datasets. Except for the linear-kernel version, multipolarity in class dimensions does not have a significant impact on Multi-SVM algorithms. They do not experience severe performance decreases when multipolarity levels increase. AdaBoost is also a robust algorithm which shows good overall performance on all datasets and it is not sensitive to the existence of multiple patterns in class dimensions. k-NN and Decision Tree Learning algorithms display even better performance than AdaBoost in a general perspective. k-NN has the highest average accuracy rate on all 8 datasets.

In addition to its overall success on all of the datasets, MNPBL is also insensitive to initialization and ordering of the training inputs. In Figure 5.16, comparison of the algorithms on IRIS, STD, DEP, DEPB, RDEPB, DMP, WD and SEED datasets is presented. In terms of learning curve analysis, MNPBL learns fast with small amount of data and stabilizes to high accuracy rates early. Therefore, compared to other methods, MNPBL shows a promising generalization performance as well.

In terms of running times, we observed that LVQ runs slowest whereas k-NN is the fastest. When different SVM kernels are compared we see that linear kernel is the slowest kernel for SVM and quadratic kernel is the fastest one on the datasets we have tested. DOP and MNPBL spends reasonable time both for training and classification phases compared to the other algorithms. The running time difference between DOP and MNPBL is acceptably small because MNPBL spends an expected amount of extra time on multipolarity detection and resolution. Figure 5.17 illustrates the running time comparison of the algorithms.

6.2 Future Work

We have demonstrated the effectiveness of our prototype based learning algorithm with the experimental results presented in Chapter 5. It could be extended and further improved in a number of ways.

In our experiments, we have proved that our algorithm has a promising performance on classification. Regression performance could be tested on different datasets and improved further by analyzing the feature behaviors in a more comprehensive way. Parameters such as neighborhood radius value can be assigned individually for each dimension instead of a general assignment.

An optimization algorithm could be applied on the derived mean-variance dataset to reduce possible noise. Especially noise in the variance dimension causes abnormalities after normalization and this affects estimation of high and low variances. Variance of a dimension is an important part of the decision making and label assignment mechanism for our method. Although we solve this problem by eliminating noises in the stage of multipolarity detection and dismissing the false clusters, an early preprocessing for noise reduction could be beneficial.

Last but not least, MNPBL algorithm could be examined in terms of entropy and compared to the other algorithms. Since it is a prototype-based learning algorithm which can operate on multipolar dimensional data, it stores cluster information for representing the training data in a prototype structure. To be able to work on a possible performance increase on prototype structure, data compression could be evaluated.

REFERENCES

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] Jerome S Bruner, Jacqueline J Goodnow, and George A Austin. *A study of thinking*. A Wiley publication in psychology. Wiley, New York, 1956.

[3] Erol Şahin, Maya Çakmak, Mehmet R. Doğar, Emre Uğur, and Göktürk Üçoluk. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 15(4):447–472, December 2007.

[4] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012.

[5] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[6] Martin Ester, Hans peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[7] Hatem A. Fayed, Sherif R. Hashem, and Amir F. Atiya. Self-generating prototypes for pattern classification. *Pattern Recogn.*, 40(5):1498–1509, May 2007.

[8] G.M. Foody and Ajay Mathur. A relative evaluation of multiclass image classification by support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, 42(6):1335–1343, June 2004.

[9] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[10] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

[11] Yoav Freund and Robert E. Schapire. A short introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann, 1999.

[12] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[13] Bernd Fritzke. Some competitive learning methods. *Artificial Intelligence Institute, Dresden University of Technology*, 1997.

[14] Liane Gabora, Eleanor Rosch, and Diederik Aerts. Toward an ecological theory of concepts. In *(D. Aerts, B. D'Hooghe and N. Note, Eds.) Worldviews, Science and Us: Bridging Knowledge and Perspectives on the World, World Scientific*, 2005.

[15] James J Gibson. *The ecological approach to visual perception*. Psychology Press, 2013.

[16] Peter D. Grünwald. *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[17] Shyam M Guthikonda. Kohonen self-organizing maps. *Wittenberg University*, 2005.

[18] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[19] Eleanor Rosch Heider. "focal" color areas and the development of color names. *Developmental psychology*, 4(3):447–455, 1971.

[20] Donald Homa, Sharon Sterling, and Lawrence Trepel. Limitations of exemplar-based generalization and the abstraction of categorical information. *Journal of Experimental Psychology: Human Learning and Memory*, 7(6):418, 1981.

[21] Sinan Kalkan, Nilgün Dag, Onur Yürüten, Anna M Borghi, and E Sahin. Verb concepts from affordances. *Interaction Studies Journal*, 15(1):437–451, 2014.

[22] Taskin Kavzoglu and I Colkesen. A kernel functions analysis for support vector machines for land cover classification. *International Journal of Applied Earth Observation and Geoinformation*, 11(5):352–359, 2009.

[23] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[24] Teuvo Kohonen. The handbook of brain theory and neural networks. chapter Learning Vector Quantization, pages 537–540. MIT Press, Cambridge, MA, USA, 1998.

[25] John K Kruschke. Category learning. *The handbook of cognition*, pages 183–201, 2005.

[26] Mingbo Ma, Ming Shao, Xu Zhao, and Yun Fu. Prototype based feature learning for face image set classification. In *10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, pages 1–6, April 2013.

[27] T.M. Martinetz, S.G. Berkovich, and K.J. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, Jul 1993.

[28] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: An open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 50–56, New York, NY, USA, 2008. ACM.

[29] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[30] G. Orhan, S. Olgunsoylu, E. Sahin, and S. Kalkan. Co-learning nouns and adjectives. In *IEEE Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–6, Aug 2013.

[31] Raquel R. Pinho, João Manuel, R. S. Tavares, and Miguel V. Correia. Efficient approximation of the mahalanobis distance for tracking with the kalman filter. In *CompIMAGE - Computational Modelling of Objects Represented in Images: Fundamentals, Methods and Applications*, pages 84–92, 2006.

[32] Michael I Posner and Steven W Keele. On the genesis of abstract ideas. *Journal of experimental psychology*, 77(3p1):353, 1968.

[33] Michael I Posner and Steven W Keele. Retentation of abstract ideas. *Journal of experimental psychology*, 77:304, 1970.

[34] A. K. Qin and P. N. Suganthan. Robust growing neural gas algorithm with application in cluster analysis. *Neural Netw.*, 17(8-9):1135–1148, October 2004.

[35] Stephen K Reed. Pattern recognition and categorization. *Cognitive psychology*, 3(3):382–407, 1972.

[36] Douglas Reynolds. Gaussian mixture models. *Encyclopedia of Biometrics*, pages 659–663, 2009.

[37] Eleanor Rosch and Barbara B Lloyd. Cognition and categorization. *Hillsdale, New Jersey*, 1978.

[38] Eleanor H Rosch. Natural categories. *Cognitive psychology*, 4(3):328–350, 1973.

[39] Jeffrey N. Rouder and Roger Ratcliff. Comparing exemplar- and rule-based theories of categorization. *Current Directions in Psychological Science*, 15(1):9–13, 2006.

[40] Atsushi Sato and Keiji Yamada. Generalized learning vector quantization. *Advances in neural information processing systems*, pages 423–429, 1996.

[41] Dee Shi and Xiaojun Yang. Support vector machines for landscape mapping from remote sensor imagery. *Proc. AutoCarto 2012*, pages 16–18, 2012.

[42] J. David Smith and John Paul Minda. Prototypes in the mist: The early epochs of category learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, pages 1411–1430, 1998.

[43] N. Suguna and K Thanushkodi. An improved k-nearest neighbor classification using genetic algorithm. *International Journal of Computer Science Issues (IJCSI)*, 7(4), 2010.

[44] Heiko Timm, Christian Borgelt, Christian Döring, and Rudolf Kruse. Fuzzy cluster analysis with cluster repulsion. In *Proc. European Symposium on Intelligent Technologies, Hybrid Systems and Their Implementation on Smart Adaptive Systems (eunite'01, Puerto de la Cruz, Tenerife, Spain)*, Aachen, Germany, 2001. Verlag Mainz.

[45] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[46] Janett Walters-Williams and Yan Li. Comparative study of distance functions for nearest neighbors. In Khaled Elleithy, editor, *Advanced Techniques in Computing Sciences and Software Engineering*, pages 79–84. Springer Netherlands, 2010.

[47] Pr Massimo Warglien and Liudmila Antonova. Categorization and decision making: Focal points and prototypicality. 2010.

[48] AA Zherebtsov and Yu A Kuperin. Application of self-organizing maps for clustering djia and nasdaq100 portfolios. *arXiv preprint cond-mat/0305330*, 2003.

APPENDIX A

DETAILED EVALUATION

In this Chapter, we present the class level analysis of f-score, precision and recall measures for each algorithm on each dataset.

A.1 Results on DEP Dataset

Table A.1: Precision, recall and f-score values of classes in DEP dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.18	0.40	0.25
2	0.70	1.00	0.82
3	0.96	0.96	0.95
4	0.96	1.00	0.98
5	0.27	0.60	0.38
6	0.80	0.98	0.87
7	0.00	0.00	0.00

Table A.2: Precision, recall and f-score values of classes in DEP dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.64	0.40	0.45
2	0.74	0.72	0.71
3	1.00	0.87	0.92
4	0.78	0.76	0.76
5	0.46	0.70	0.53
6	0.80	0.69	0.70
7	0.32	0.47	0.34

Table A.3: Precision, recall and f-score values of classes in DEP dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.76	0.76	0.75
2	0.91	0.94	0.92
3	0.93	0.98	0.95
4	0.98	0.98	0.98
5	0.74	0.71	0.71
6	0.84	0.85	0.83
7	0.53	0.49	0.51

Table A.4: Precision, recall and f-score values of classes in DEP dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.64	0.79	0.70
2	0.76	0.98	0.84
3	1.00	0.93	0.96
4	0.98	1.00	0.99
5	0.75	0.55	0.61
6	0.88	0.93	0.89
7	0.61	0.32	0.40

Table A.5: Precision, recall and f-score values of classes in DEP dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.00	0.00	0.00
2	0.03	0.20	0.05
3	0.00	0.00	0.00
4	0.05	0.40	0.09
5	0.03	0.20	0.04
6	0.00	0.00	0.00
7	0.01	0.10	0.02

Table A.6: Precision, recall and f-score values of classes in DEP dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.78	0.41	0.50
2	0.91	0.74	0.78
3	1.00	0.91	0.95
4	0.98	0.93	0.95
5	0.52	0.84	0.63
6	0.72	0.88	0.79
7	0.49	0.46	0.41

Table A.7: Precision, recall and f-score values of classes in DEP dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.64	0.74	0.64
2	0.84	0.98	0.90
3	0.91	0.94	0.91
4	0.93	1.00	0.96
5	0.77	0.43	0.53
6	0.89	0.98	0.92
7	0.63	0.38	0.44

Table A.8: Precision, recall and f-score values of classes in DEP dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.69	0.73	0.68
2	0.73	0.87	0.79
3	0.90	0.91	0.89
4	0.80	0.91	0.84
5	0.63	0.46	0.52
6	0.92	0.80	0.83
7	0.32	0.19	0.21

Table A.9: Precision, recall and f-score values of classes in DEP dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.59	0.73	0.63
2	0.76	0.81	0.77
3	0.89	0.88	0.88
4	0.94	0.90	0.91
5	0.70	0.49	0.55
6	0.87	0.88	0.87
7	0.37	0.29	0.30

Table A.10: Precision, recall and f-score values of classes in DEP dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.79	0.39	0.51
2	0.84	0.35	0.47
3	0.80	0.30	0.43
4	0.80	0.40	0.52
5	0.40	0.09	0.14
6	1.00	0.71	0.81
7	0.17	0.86	0.28

A.2 Results on DEPB Dataset

Table A.11: Precision, recall and f-score values of classes in DEPB dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.41	0.49	0.42
2	0.96	0.98	0.97
3	0.97	1.00	0.98
4	0.73	0.98	0.81
5	0.79	0.98	0.87
6	0.00	0.00	0.00

Table A.12: Precision, recall and f-score values of classes in DEPB dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.52	0.49	0.49
2	1.00	0.91	0.95
3	0.98	0.89	0.93
4	0.70	0.75	0.68
5	0.92	0.73	0.77
6	0.34	0.43	0.36

Table A.13: Precision, recall and f-score values of classes in DEPB dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.86	0.82	0.82
2	0.93	0.98	0.95
3	0.98	0.98	0.98
4	0.92	0.88	0.88
5	0.83	0.90	0.84
6	0.71	0.54	0.58

Table A.14: Precision, recall and f-score values of classes in DEPB dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.62	0.80	0.69
2	0.98	0.93	0.95
3	0.98	1.00	0.99
4	0.86	0.73	0.77
5	0.87	0.94	0.89
6	0.63	0.41	0.46

Table A.15: Precision, recall and f-score values of classes in DEPB dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.00	0.00	0.00
2	0.01	0.10	0.03
3	0.00	0.00	0.00
4	0.05	0.30	0.09
5	0.05	0.30	0.09
6	0.06	0.33	0.10

Table A.16: Precision, recall and f-score values of classes in DEPB dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.63	0.40	0.45
2	1.00	0.91	0.95
3	0.98	0.91	0.94
4	0.69	0.95	0.77
5	0.81	1.00	0.89
6	0.61	0.36	0.41

Table A.17: Precision, recall and f-score values of classes in DEPB dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.29	0.63	0.39
2	0.71	0.57	0.61
3	0.62	0.28	0.37
4	0.82	0.56	0.62
5	0.82	0.98	0.89
6	0.40	0.26	0.29

Table A.18: Precision, recall and f-score values of classes in DEPB dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.43	0.51	0.46
2	0.86	0.89	0.87
3	0.83	0.90	0.85
4	0.73	0.57	0.61
5	0.91	0.80	0.83
6	0.30	0.21	0.22

Table A.19: Precision, recall and f-score values of classes in DEPB dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.54	0.58	0.54
2	0.87	0.88	0.87
3	0.90	0.92	0.90
4	0.78	0.57	0.64
5	0.89	0.89	0.88
6	0.45	0.45	0.44

Table A.20: Precision, recall and f-score values of classes in DEPB dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.70	0.25	0.35
2	0.90	0.31	0.45
3	0.80	0.43	0.55
4	0.65	0.15	0.24
5	1.00	0.71	0.81
6	0.20	0.86	0.31

A.3 Results on IRIS Dataset

Table A.21: Precision, recall and f-score values of classes in IRIS dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.91	0.93	0.92
3	0.94	0.92	0.93

Table A.22: Precision, recall and f-score values of classes in IRIS dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.80	0.88
3	0.86	1.00	0.92

Table A.23: Precision, recall and f-score values of classes in IRIS dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.96	0.91	0.92
3	0.94	0.96	0.94

Table A.24: Precision, recall and f-score values of classes in IRIS dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.93	0.93	0.92
3	0.95	0.94	0.94

Table A.25: Precision, recall and f-score values of classes in IRIS dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.83	0.98	0.90
3	0.97	0.80	0.87

Table A.26: Precision, recall and f-score values of classes in IRIS dataset, using MNPBL algorithm. 10-fold cross validation is used. Neighborhood radius ϵ is choosen as 1.00.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.82	0.89
3	0.87	1.00	0.93

Table A.27: Precision, recall and f-score values of classes in IRIS dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.45	0.59	0.50
3	0.69	0.67	0.62

Table A.28: Precision, recall and f-score values of classes in IRIS dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.85	0.93	0.88
3	0.94	0.81	0.85

Table A.29: Precision, recall and f-score values of classes in IRIS dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.91	0.98	0.94
3	0.98	0.90	0.93

Table A.30: Precision, recall and f-score values of classes in IRIS dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	0.98	0.99
2	0.90	0.98	0.94
3	0.96	0.90	0.92

A.4 Results on RDEPB Dataset

Table A.31: Precision, recall and f-score values of classes in RDEPB dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.98	0.99
3	0.98	1.00	0.99
4	1.00	1.00	1.00

Table A.32: Precision, recall and f-score values of classes in RDEPB dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.00	0.00	0.00
2	0.00	0.00	0.00
3	0.00	0.00	0.00
4	0.25	1.00	0.40

Table A.33: Precision, recall and f-score values of classes in RDEPB dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	0.98	0.98	0.98
3	0.98	0.98	0.98
4	1.00	1.00	1.00

Table A.34: Precision, recall and f-score values of classes in RDEPB dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.98	0.99
3	0.98	1.00	0.99
4	1.00	1.00	1.00

Table A.35: Precision, recall and f-score values of classes in RDEPB dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.00	0.00	0.00
2	0.78	0.98	0.85
3	0.64	1.00	0.77
4	1.00	0.98	0.99

Table A.36: Precision, recall and f-score values of classes in RDEPB dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.98	0.99
3	0.98	1.00	0.99
4	1.00	1.00	1.00

Table A.37: Precision, recall and f-score values of classes in RDEPB dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.33	1.00	0.50
2	0.00	0.00	0.00
3	0.00	0.00	0.00
4	1.00	0.98	0.99

Table A.38: Precision, recall and f-score values of classes in RDEPB dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.98	0.99
3	0.98	1.00	0.99
4	1.00	1.00	1.00

Table A.39: Precision, recall and f-score values of classes in RDEPB dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.98	0.99
3	0.98	1.00	0.99
4	1.00	1.00	1.00

Table A.40: Precision, recall and f-score values of classes in RDEPB dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	0.98	0.99
3	0.98	1.00	0.99
4	1.00	1.00	1.00

A.5 Results on STD Dataset

Table A.41: Precision, recall and f-score values of classes in STD dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.96	0.98	0.96
2	0.97	0.93	0.94

Table A.42: Precision, recall and f-score values of classes in STD dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.88	0.80	0.82
2	0.62	0.71	0.63

Table A.43: Precision, recall and f-score values of classes in STD dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.94	0.96	0.94
2	0.84	0.78	0.80

Table A.44: Precision, recall and f-score values of classes in STD dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.94	0.95	0.94
2	0.96	0.88	0.90

Table A.45: Precision, recall and f-score values of classes in STD dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.88	0.87	0.86
2	0.62	0.57	0.57

Table A.46: Precision, recall and f-score values of classes in STD dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.88	0.80	0.82
2	0.63	0.71	0.64

Table A.47: Precision, recall and f-score values of classes in STD dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.97	1.00	0.98
2	1.00	0.95	0.97

Table A.48: Precision, recall and f-score values of classes in STD dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.99	0.95	0.97
2	0.96	0.99	0.97

Table A.49: Precision, recall and f-score values of classes in STD dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.97	0.90	0.93
2	0.79	0.85	0.81

Table A.50: Precision, recall and f-score values of classes in STD dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.67	0.90	0.75
2	0.93	0.39	0.42

A.6 Results on DMP Dataset

Table A.51: Precision, recall and f-score values of classes in DMP dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.73	1.0	0.82
2	1.00	0.99	0.99
3	1.00	1.00	1.00
4	0.30	0.60	0.40
5	0.20	0.40	0.27
6	0.60	0.49	0.53

Table A.52: Precision, recall and f-score values of classes in DMP dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.00	0.00	0.00
2	0.21	0.71	0.32
3	0.51	1.00	0.67
4	0.14	0.30	0.19
5	0.64	0.30	0.35
6	0.00	0.00	0.00

Table A.53: Precision, recall and f-score values of classes in DMP dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	1.00	1.00	1.00

Table A.54: Precision, recall and f-score values of classes in DMP dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	1.00	1.00	1.00

Table A.55: Precision, recall and f-score values of classes in DMP dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.00	0.00	0.00
2	0.05	0.30	0.09
3	0.00	0.00	0.00
4	0.05	0.30	0.09
5	0.02	0.10	0.03
6	0.05	0.30	0.09

Table A.56: Precision, recall and f-score values of classes in DMP dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	0.78	1.00	0.87
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	1.00	0.65	0.73

Table A.57: Precision, recall and f-score values of classes in DMP dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.17	1.00	0.29
2	0.00	0.00	0.00
3	0.00	0.00	0.00
4	0.00	0.00	0.00
5	0.00	0.00	0.00
6	0.00	0.00	0.00

Table A.58: Precision, recall and f-score values of classes in DMP dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.80	1.00	0.87
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	0.60	0.60	0.60

Table A.59: Precision, recall and f-score values of classes in DMP dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.73	1.00	0.83
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	0.60	0.49	0.53

Table A.60: Precision, recall and f-score values of classes in DMP dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.73	1.00	0.83
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	1.00	1.00	1.00
6	0.60	0.49	0.53

A.7 Results on WD Dataset

Table A.61: Precision, recall and f-score values of classes in WD dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.94	0.97	0.95
2	0.95	0.91	0.92
3	0.97	0.93	0.93

Table A.62: Precision, recall and f-score values of classes in WD dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.97	0.85	0.89
2	0.83	0.95	0.88
3	0.98	0.94	0.96

Table A.63: Precision, recall and f-score values of classes in WD dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.94	0.92	0.92
2	0.90	0.91	0.90
3	0.95	0.92	0.92

Table A.64: Precision, recall and f-score values of classes in WD dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.86	0.89	0.86
2	0.89	0.85	0.85
3	0.98	0.86	0.90

Table A.65: Precision, recall and f-score values of classes in WD dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.09	0.12	0.77
2	0.20	0.58	0.30
3	0.11	0.33	0.17

Table A.66: Precision, recall and f-score values of classes in WD dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.97	0.85	0.89
2	0.83	0.95	0.88
3	0.98	0.94	0.96

Table A.67: Precision, recall and f-score values of classes in WD dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.96	1.00	0.98
2	0.95	0.88	0.91
3	0.83	0.85	0.84

Table A.68: Precision, recall and f-score values of classes in WD dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.87	1.00	0.93
2	0.95	0.88	0.91
3	0.88	0.74	0.80

Table A.69: Precision, recall and f-score values of classes in WD dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.93	0.98	0.95
2	0.97	0.87	0.91
3	0.84	0.84	0.83

Table A.70: Precision, recall and f-score values of classes in WD dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	1.00	0.81	0.89
2	0.95	0.79	0.84
3	0.63	0.86	0.73

A.8 Results on SEED Dataset

Table A.71: Precision, recall and f-score values of classes in SEED dataset, using AdaBoost algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.91	0.80	0.83
2	0.99	0.97	0.98
3	0.87	0.94	0.89

Table A.72: Precision, recall and f-score values of classes in SEED dataset, using DOP algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.86	0.96	0.90
2	0.99	0.97	0.98
3	0.97	0.86	0.91

Table A.73: Precision, recall and f-score values of classes in SEED dataset, using Decision Tree Learning algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.87	0.87	0.87
2	0.98	0.97	0.97
3	0.91	0.90	0.90

Table A.74: Precision, recall and f-score values of classes in SEED dataset, using k-NN algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.87	0.81	0.83
2	0.93	0.91	0.90
3	0.91	0.94	0.93

Table A.75: Precision, recall and f-score values of classes in SEED dataset, using LVQ algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.86	0.86	0.84
2	0.99	0.89	0.90
3	0.90	0.91	0.89

Table A.76: Precision, recall and f-score values of classes in SEED dataset, using MNPBL algorithm. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.86	0.96	0.90
2	0.99	0.97	0.98
3	0.97	0.86	0.91

Table A.77: Precision, recall and f-score values of classes in SEED dataset, using SVM algorithm. Linear kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.85	0.94	0.88
2	0.99	0.90	0.94
3	0.93	0.87	0.89

Table A.78: Precision, recall and f-score values of classes in SEED dataset, using SVM algorithm. Polynomial kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.91	0.91	0.90
2	0.95	0.91	0.92
3	0.96	0.94	0.95

Table A.79: Precision, recall and f-score values of classes in SEED dataset, using SVM algorithm. Quadratic kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.91	0.94	0.92
2	0.96	0.94	0.95
3	0.96	0.91	0.93

Table A.80: Precision, recall and f-score values of classes in SEED dataset, using SVM algorithm. RBF kernel is used as kernel function. 10-fold cross validation is used.

Class	Precision	Recall	F-Measure
1	0.90	0.87	0.88
2	0.96	0.94	0.95
3	0.92	0.93	0.92