NON-INCREMENTAL CLASSIFICATION LEARNING ALGORITHMS BASED ON VOTING FEATURE INTERVALS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

by

Gülşen Demiröz

August, 1997

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Halil Altay Güvenir (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. İlyas Çiçekli

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst.Prof. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

NON-INCREMENTAL CLASSIFICATION LEARNING ALGORITHMS BASED ON VOTING FEATURE INTERVALS

Gülşen Demiröz
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Halil Altay Güvenir
August, 1997

Learning is one of the necessary abilities of an intelligent agent. This thesis proposes several learning algorithms for multi-concept descriptions in the form of feature intervals, called *Voting Feature Intervals* (VFI) algorithms. These algorithms are non-incremental classification learning algorithms, and use feature projection based knowledge representation for the classification knowledge induced from a set of preclassified examples. The concept description learned is a set of intervals constructed separately for each feature. Each interval carries classification information for all classes. The classification of an unseen instance is based on a voting scheme, where each feature distributes its vote among all classes. Empirical evaluation of the VFI algorithms have shown that they are the best performing algorithms among other previously developed feature projection based methods in terms of classification accuracy. In order to further improve the accuracy, genetic algorithms are developed to learn the optimum feature weights for any given classifier. Also a new crossover operator, called *continuous uniform crossover*, to be used in this weight learning genetic algorithm is proposed and developed during this thesis. Since the explanation ability of a learning system is as much important as its accuracy, VFI classifiers are supplemented with a facility to convey what they have learned in a comprehensible way to humans.

Keywords: machine learning, supervised learning, classification, inductive learning, non-incremental learning, feature intervals, voting, genetic algorithms.

ÖZET

OYLAYAN ÖZNİTELİK BÖLÜNTÜLERİNE DAYALI TOPLU SINIFLANDIRMA ÖĞRENME ALGORİTMALARI

Gülşen Demiröz
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Halil Altay Güvenir
Ağustos, 1997

Öğrenmek akıllı bir bireyin en gerekli özelliklerinden biridir. Bu tezde çoklu kavram tanımlarını öznitelik aralıkları şeklinde öğrenen yeni algoritmalar önerilmektedir. *Oylayan Öznitelik Aralıkları* (*VFI*) olarak isimlendirilen bu algoritmalar toplu sınıflandırma öğrenme algoritmalarıdırlar. Daha önceden sınıflandırılmış olan örneklerden sınıflandırma bilgisini çıkarmak için öznitelik izdüşümlerine dayalı bilgi gösterim yöntemini kullanırlar. Öğrenilen kavram tanımı her öznitelik için ayrı ayrı öğrenilen aralıklar şeklindedir. Her bir aralık bütün sınıflar için sınıflandırma bilgisi içerir. Yeni bir örneğin sınıflandırılması her özniteliğin oyunu bütün sınıflara dağıttığı bir oylama sistemine dayanır. Gerçek hayattan alınan veri kümeleri üzerinde yapılan deneylerde VFI algoritmaları daha önce geliştirilmiş öznitelik izdüşümlerine dayalı diğer metodlardan daha yüksek sınıflandırma doğruluğu elde etmişlerdir. Ayrıca sınıflandırma doğruluğunu daha çok arttırmak için optimum öznitelik ağırlıklarını öğrenen genetik algoritmalar geliştirilmiştir. Aynı zamanda bu genetik algoritmalarda kullanılmak üzere yeni bir çaprazlama operatörü de geliştirilmiştir. Bir öğrenme sisteminin açıklama yeteneği de en az doğruluğu kadar önemli olduğundan, VFI algoritmaları öğrendiklerini insanların anlayabileceği bir şekilde gösterebilmektedirler.

Anahtar Sözcükler: öğrenme, tümevarımsal öğrenme, sınıflandırma, toplu öğrenme, denetimli öğrenme, öznitelik izdüşümleri, oylama, genetik algoritmalar.

ACKNOWLEDGMENTS

I would like to express my gratitude to Assoc. Prof. H. Altay Güvenir, from whom I have learned a lot, due to his supervision, suggestions, and understanding throughout the development of this thesis.

I am also indebted to Assist. Prof. Özgür Ulusoy and Assist. Prof. İlyas Çiçekli for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to thank to Halime Büyükyıldız for everything, Esra Erdem for her mails at any time from Texas, Gökmen Gök especially for his poems, my yoga friends, Antal van den Bosch for his friendship at the conferences, Serap Yılmaz, Bahtiser Kuş, Aynur Akkuş, Bilge Aydın, my sister Ayşen Demiröz, my younger sisters, and my parents for their morale support and friendship. I should not forget to thank to the bars in Ankara to which usually I and Halime were used to go.

I would also like to thank Bilkent University, which enabled this research environment and supported the presentation of this work at conferences.

This thesis was supported by TUBITAK (Scientific and Technical Research Council of Turkey) under Grant EEEAG-153.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

1PCO	: One-Point Crossover
1PCO–WNN	: WNN learning feature weights using a GA which uses 1PCO
2PCO	: Two-Point Crossover
2PCO–WNN	: WNN learning feature weights using a GA which uses 2PCO
1R	: System whose input is training examples and output is 1-rule
CFP	: Classification by Feature Partitioning
COFI	: Classification by Overlapping Feature Intervals
C_i	: Label of the i^{th} class
CUCO	: Continuous Uniform Crossover
CUCO–WNN	: WNN learning feature weights using a GA which uses CUCO
CUCO–WVFI1	: WVFI1 learning feature weights using a GA which uses CUCO
C4.5	: Decision tree algorithm
d	: Number of features in the dataset
D_f	: Generalization distance for feature f in the COFI algorithm
D_{eH}	: Euclidean distance between example e and exemplar H
e	: A training example
e_f	: f^{th} feature value of the example e
e_c	: class label of the example e
f_j	: j^{th} feature
g	: Generalization ratio
EACH	: Exemplar-Aided Constructor of Hyperrectangles
FIL	: Feature Interval Learning Algorithms
FI1	: Feature Interval Learning Algorithm
FI2	: Feature Interval Learning Algorithm
FI3	: Feature Interval Learning Algorithm
FI4	: Feature Interval Learning Algorithm
FPB	: Feature Projection Based Learning Algorithms
GA	: Genetic Algorithm
GA-CFP	: Hybrid CFP Algorithm
H	: Hyperrectangle
H_f	: f^{th} feature value of the exemplar H
$H_{f,lower}$	: Lower end of the range for the exemplar H for feature f

$H_{f,upper}$: Upper end of the range for the exemplar H for feature f

IBL : Instance-based learning

IB1 : Instance-based learning algorithm

IB2 : Instance-based learning algorithm

IB3 : Instance-based learning algorithm

IB4 : Instance-based learning algorithm

IB5 : Instance-based learning algorithm

ID3 : Decision tree algorithm

k : Number of classes in the dataset (unless otherwise specified) or number of neighbors

k-NN : K Nearest Neighbor

k-NNFP : K Nearest Neighbor on Feature Projections

log : Logarithm in base 2

m : Number of training instances

max_f : Maximum value for the feature f

min_f : Minimum value for the feature f

NBC : Naive Bayesian Classifier

NBCN : Naive Bayesian Classifier assuming normal distribution

NGE : Nested-Generalized Exemplars

NN : Nearest Neighbor Algorithm (1-NN)

OC1 : Oblique Classifier 1

$p(\mathbf{x}|w_j)$: Conditional probability density function for $\mathbf{x}$ conditioned on given w_j

$P(w_c)$: Prior probability of being class c for an instance

$P(w_c|\mathbf{x})$: The posterior probability of an instance being class c given the observed feature value vector $\mathbf{x}$

SADT : Simulated Annealing of Decision Trees

t : A test example

t_c : Class label of the test example t

t_f : f^{th} feature value of the test example t

T2 : Agnostic PAC learning decision tree algorithm with at most two levels

UCO : Uniform Crossover

UCO–WNN : WNN learning feature weights using a GA which uses UCO

$\mathbf{v}$: Total vote vector

$\mathbf{v}_f$: The vote vector of the f^{th} feature

VFI	: Voting Feature Intervals
VFI1	: Voting Feature Intervals Algorithm Version 1
VFI2	: Voting Feature Intervals Algorithm Version 2
VFI3	: Voting Feature Intervals Algorithm Version 3
VFI4	: Voting Feature Intervals Algorithm Version 4
VFI5	: Voting Feature Intervals Algorithm Version 5
$\mathbf{x}$	: Instance vector
$\mathbf{x}_i$	: Value vector of i^{th} instance
w_f	: Weight of feature f
w_H	: Weight of exemplar H
WNN	: Weighted Nearest Neighbor Classifier
WVFI1	: Weighted VFI1 Classifier
$\triangle$	: Weight adjustment rate of the CFP algorithm

Chapter 1

Introduction

Since learning is one of the necessary abilities of an intelligent agent, machine learning has played an important role in artificial intelligence. Simon [66] has defined *learning* as changes in a system that enable it to do the same task or tasks drawn from the same population more efficiently and more effectively the next time. There are two ways in which a system can change [65]:

1. The system can acquire new knowledge from external sources (*knowledge acquisition*)

2. The system can modify itself to exploit its current knowledge more effectively (*refinement of skills through practice*)

The first type of learning acquires new knowledge from external sources in order to solve a problem, perform a new task or improve the performance of an existing task. The second kind of learning is often called *speedup learning* or *skill acquisition*. This kind of learning is usually used for improving the efficiency of search-base problem-solving systems. One way to speed up search is to introduce macro operators that take "big steps" in the search space. Another way to speed up search is to introduce meta level control knowledge. *Explanation–based learning* (EBL) [19] is a technique that has been applied to learn macro operators and search control knowledge.

Michalski, Carbonell, and Mitchell classify Machine Learning (ML) approaches according to their learning strategies as follows [49]:

- *Rote learning* is also called as *learning by being programmed* and consists of just recording the different objects supplied by a teacher. Classical database systems illustrate this strategy.
- *Learning by instruction* is learning by being told some new knowledge from an external source.
- *Inductive learning* or *empirical learning* is accomplished by reasoning from externally supplied examples to produce more general descriptions.
- *Learning by observation* is learning by observing the environment and making discoveries.

Inductive learning or *empirical learning* has been heavily investigated in ML literature. Inductive learning can be described as learning by drawing inductive inference from facts that are provided by a teacher or an environment. Acquiring knowledge involves operations of generalizing, specializing, transforming, correcting and refining knowledge representations [49]. Learning a *concept* usually means to learn its description, that is, a relation between the name of the concept and a given set of features by making some inferences. This learning strategy requires that a sufficient number of examples made available to the learner. We focus in general on inductive learning — learning from examples— in this thesis. Inductive learning can be categorized into two categories: *supervised learning* and *unsupervised learning*.

Supervised learning, also known as *classification*, is the primary task studied in machine learning research. A supervised learning algorithm receives a set of preclassified training instances (examples), each labeled with a particular class. The goal of such a learning algorithm is to learn a classification rule that will correctly assign new instances to these classes. For example, instances could be descriptions of the symptoms of diseased and healthy patients. The classes here are "diseased" and "healthy", and the task of the learning system is to produce a set of rules for accurately predicting whether new patients are diseased or healthy.

In unsupervised learning, the training instances have not been assigned to classes by a teacher. Only the descriptions of these instances are given and the goal of the inductive learning system is to search for some regularities and natural groupings (clustering) among these instances. Unsupervised learning differs from supervised learning in the measure of success. To test whether a supervised learning algorithm has succeeded, we can simply apply it to a set of test examples and see if they are correctly classified that is, the classification of the system agrees with the classification of the teacher. But with unsupervised learning, we must examine the test examples and see if they exhibit the same regularity that was discovered in the training instances.

Supervised learning is also called as *concept learning* or *concept acquisition*, and the classes are called as *concepts*. The word "concept" is derived from the Latin word "concipere" meaning "to seize (a thought)" and the learning system seizes the concept by learning a set of conditions sufficient to decide whether a given object is or is not an instance of it. The two types of concept learning are *single concept learning* and *multi-concept learning*. In single concept learning, the teacher either provides only the positive instances (instances of the concept) or both the positive and negative instances to the learning system. For example, the records of healthy patients can be viewed as the positive instances and the records of diseased (non-healthy) patients can be viewed as the negative instances of the "healthy" concept. The set of rules learned by the concept learning system from the given examples is the description of the "healthy" concept. Single-concept learning is a special case of multi-concept learning, where there are more than one concept to be learned. For example, there are several brands of cars some of which are Opel, Renault, Mazda, Volkswagen, etc. In this multi-concept learning domain, instances do not belong to more than one class (a car can not be Opel and Mazda at the same time), that is, classifications of instances are mutually disjoint. But in some other multi-concept learning tasks, instances may belong to more than one class, that is, classifications of instances are possibly overlapping.

Several concept learning systems that learn multi-concept descriptions from instances where the concepts are mutually disjoint have been developed. The Nearest Neighbor algorithm [17, 18, 24], Decision Tree Inducers [37, 28, 13, 55,

60, 14, 36, 12], Bayesian Classifier originating from work in pattern recognition [24, 29], learning by EACH (Exemplar-Aided Constructor of Hyperrectangles) [62], and instance-based learning algorithms [5, 9] are some of them and explained in Chapter 2.

This thesis proposes several new multi-concept learning algorithms, called Voting Feature Intervals (VFI) algorithms. The VFI algorithms are non-incremental classification learning algorithms that learn the concept descriptions by constructing feature intervals on each feature dimension from a set of preclassified examples provided by a teacher. Classification of a new example is performed by a voting scheme where the feature intervals distribute their vote among classes. The features are considered separately both in learning and classification which provides faster classification times. Processing each feature separately enables a simple and effective way of handling missing feature values which is a problem for decision tree inductive learning and the nearest neighbor algorithms.

The VFI classifiers always achieve higher classification accuracies than all other classification algorithms that use the feature projection based knowledge representation and usually perform better than the Naive Bayesian Classifier. Another advantage of the VFI classifiers is that it is possible to make a general classification returning a probability distribution over all classes instead of a categorical classification [45].

The representation of concepts learned by VFI classifiers is similar to that of other concept learning models using feature projection based knowledge representation scheme such as CFP [32], FIL algorithms [7], COFI [73], and k-NNFP [8] all of which are described in detail in Chapter 3. The voting scheme used to classify a new instance has also evolved from the voting schemes used in these related methods. Chapter 4 explains the details of this new classification method. Since induction of multi-concept descriptions from classified examples have large number of applications to real-world problems, we will empirically evaluate VFI classifiers on some real-world datasets from the UCI-Repository [51] and artificially generated datasets in Chapter 5. For this purpose, we have also compiled two medical datasets, one for the description of arrhythmia characteristics from ECG signals, and the other for the histopathological

description of a set of dermatological diseases. The classification performance of VFI algorithms are compared with that of other classification algorithms discussed in Chapters 2 and 3. Chapter 5 also presents the complexity analysis of the VFI algorithms. Chapter 6 describes and presents the experimental results of a feature weight learning genetic algorithm combined with the Nearest Neighbor and the VFI algorithms. Chapter 7 presents how we visualize the concept learned by the VFI algorithms and the explanation of classification of a new instance. The final chapter presents a summary of this thesis and some ideas for future work.

Chapter 2

Supervised Inductive Learning Models

Supervised inductive learning (concept learning) from examples has been the most active research area in machine learning. It can be defined as process of acquiring knowledge by drawing inductive inferences from teacher or environment-provided facts by generalizing, specializing, transforming, correcting and refining knowledge representations [49].

The necessary input to a concept learning system is a set of training examples correctly assigned to classes by a teacher. All the concept learning systems mentioned in this thesis use *feature-value* description for the input training instances. Feature-value representation expresses all the information about one instance in terms of a fixed collection of properties or features. Each feature may have either discrete (nominal) or continuous (linear) values. For example, *color* feature having values "red", "blue", or "green" is a nominal feature whereas *age* feature is a linear feature which can take any numerical value (integer or real) in some range and in general has a linearly ordered set of feature values. One important restriction is that the features used to describe an instance must not vary from one instance to another. Since a teacher assigns classes to instances in supervised learning, the input instances have a class label in addition to the feature values. The learning systems in this thesis can learn multi-concepts requiring that an instance can not belong to more

than one class and the classes are discrete. There are tasks that do not have discrete classes and concerned with the prediction of continuous value such as the price of gold. The multi-concept learning of discrete classes is very often called as *classification* since the concept learning system (classifier) will predict a class value for the new instance among those discrete classes.

For concept learning tasks, one of the widely used representation technique is the *exemplar-based* representation. Either representative instances as in Instance-Based Learning [5] or generalizations of instances as in Nested-Generalized Exemplars [62] form the concept descriptions in exemplar-based models. Another useful knowledge representation technique for concept learning is decision trees [55]. Statistical concept learning algorithms also use training instances to induce concept descriptions based on certain probabilistic approaches [24]. In the following sections, these concept learning models are presented and most of them will be later used to compare with the new learning methods developed in this thesis.

2.1 Exemplar-Based Learning

Exemplar-Based Learning was originally proposed as a model of human learning by Medin and Schaffer [48]. In the simplest form of exemplar-based learning, every example is stored in memory verbatim, with no change of representation. An example is defined as a vector of feature values along with a label which represents the category (class) of the example.

Knowledge representation of exemplar-based models can be maintained as representative instances [2, 5], hyperrectangles [62, 63], or feature projection based representations [7, 8, 22, 32, 73]. Unlike Explanation-Based Generalization (EBG) [19, 50], little or no domain specific knowledge is required in exemplar-based learning.

Figure 2.1 presents a hierarchical classification of exemplar-based learning models. Instance-Based Learning (IBL) and Exemplar-Based Generalization

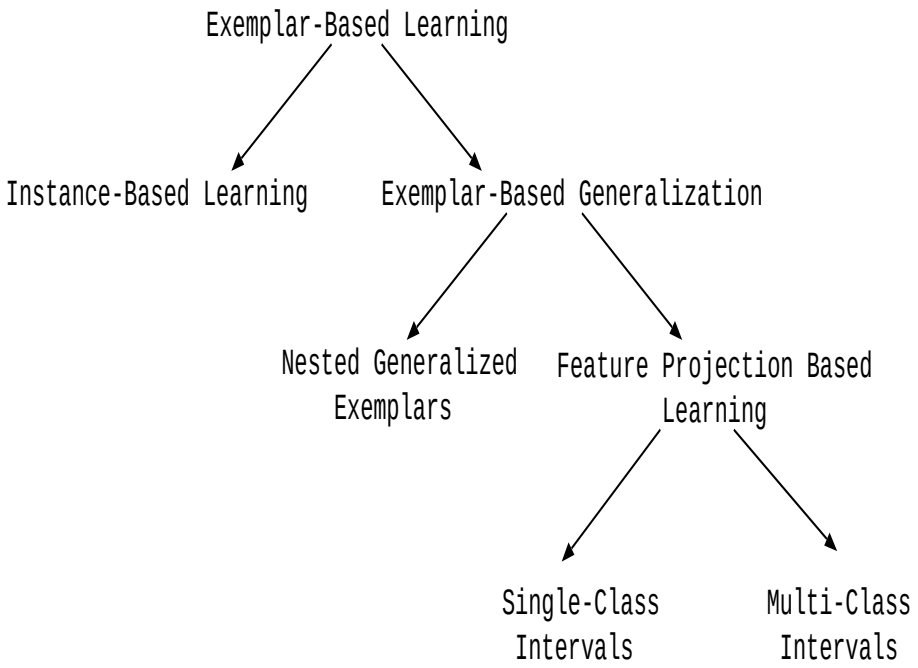

Figure 2.1. Classification of Exemplar-Based Learning models.

are two types of exemplar-based Learning. For example, instance-based learning methods [5] retain examples in memory as points, and never changes them. On the other hand, exemplar-based generalization methods make certain generalizations on the training instances. One category of the exemplar-based generalization is the Nested-Generalized Exemplars (NGE) model [62]. This model changes the point storage model of the instance-based learning and retains examples in the memory as axis-parallel hyperrectangles. Feature Projection Based (FPB) learning models are the basis for this thesis and can be classified as exemplar-based generalization methods. The FPB algorithms learn the concept descriptions by generalizing the projections of the training examples separately on each feature. In this thesis, we will study several supervised inductive learning methods that can be also categorized as feature projection based algorithms. In the following sections, we will describe the IBL and NGE methods briefly. Previously developed FPB methods will be discussed in detail in Chapter 3.

2.1.1 Instance-Based Learning

Instance-Based Learning (IBL) methods extend the classical nearest neighbor algorithm, which has large storage requirements [5, 9]. IBL algorithms generate classification predictions using only specific instances. Aha calls them also as *lazy learning* algorithms since the concept description is a set of stored instances [76]. All instances are represented as points on the d-dimensional Euclidean space, where d is the number of features. The concept descriptions can change incrementally after each training instance is processed. IBL algorithms do not construct extensional concept descriptions. Instead, concept descriptions are determined by how the IBL algorithm's selected *similarity* and *classification* functions use the current set of saved instances. There are three components in the framework which describe all IBL algorithms as defined by Aha and Kibler [5]:

1. The *similarity function* computes the *similarity* between two instances (similarities are real-valued).
2. The *classification function* receives the output of the similarity function and the classification performance records of the instances in the concept description, and yields a classification for instances.
3. The *concept description updater* maintains records on classification performance and decides which instance are to be included in the concept description.

These similarity and classification functions determine how the set of instances in the concept description are used for prediction. So, IBL concept descriptions contain not only a set of instances, but also these two functions.

Several IBL algorithms have been developed: IB1, IB2, IB3, IB4 and IB5 [3, 5]. IB1 is the simplest one and it uses the similarity function computed as

$$similarity(x,y) = -\sqrt{\sum_{f=1}^{n} diff(f,x,y)^2} \tag{2.1}$$

$$diff(f,x,y) = \begin{cases} |x_f - y_f| & \text{if } f \text{ is linear} \\ 0 & \text{if } f \text{ is symbolic and } x_f = y_f \\ 1 & \text{if } f \text{ is symbolic and } x_f \neq y_f \end{cases} \tag{2.2}$$

where x and y are the instances.

IB1 is identical to the nearest neighbor algorithm except that it processes training instances incrementally and simply ignores instances with missing feature value(s). Since IB1 stores all the training instances, its storage requirement is quite large. IB2 is an extension of IB1, it saves only misclassified instances reducing storage requirement. On the other hand, its classification accuracy decreases in the presence of noisy instances. IB3 aims to cope with noisy instances. IB3 employs a *significance test* to determine which instances are good classifiers and which ones are believed to be noisy. Once an example is determined to be noisy, it is removed from the description set. IB2 and IB3 are also incremental algorithms. IB1, IB2, and IB3 algorithms assume that all features have equal relevance for describing concepts.

To study the effect of relevances of features in IBL algorithms, IB4 has been proposed by Aha [3]. In this study, different feature weights are learned for different concepts; a feature may be highly relevant to one concept and completely irrelevant to another. So, IB4 has been developed as an extension of IB3 that learns a separate set of feature weights for each concept. Weights are adjusted using a simple feedback algorithm to reflect the relative relevances of the features to describe instances. These weights are then used in IB4's similarity function which is a Euclidean weighted-distance measure of the similarity of two instances. Multiple sets of weights are used because similarity is concept-dependent, the similarity of two instances varies depending on the target concept. IB4 decreases the effect of irrelevant features on classification decisions. Therefore, it is quite successful in the presence of irrelevant features.

The problem of novelty is defined as the problem of learning when novel features are used to help describe instances. IB4, similar to its predecessors, assumes that all the features used to describe training instances are known before training begins. However, in several learning tasks, the set of describing features is not known beforehand. IB5 [3], is an extension of IB4 that tolerates the introduction of novel features during training. To simulate this capability during training, IB4 simply assumes that the values for the (as yet) unused feature are missing. During training, IB4 fixes the expected relevance of the feature for classifying instances. IB5 instead updates the weight of a feature only when its value is known for both of the instances involved in a classification attempt. IB5 can therefore learn the relevance of novel features more quickly than IB4.

Also noise-tolerant versions of instance-based algorithms have been developed by Aha and Kibler [4]. These learning algorithms are based on a form of significance testing, that identifies and eliminates noisy concept descriptions.

2.1.2 Nested-Generalized Exemplars

Nested-generalized exemplar (NGE) theory is a variation of exemplar-based learning [62]. In NGE, an exemplar is a single training example, and a generalized exemplar is an axis-parallel hyperrectangle that may cover several training examples. These hyperrectangles may overlap or nest. Hyperrectangles are grown during training in an incremental manner.

Salzberg implemented NGE in a program called EACH (Exemplar-Aided Constructor of Hyperrectangles) [63]. In EACH, the learner compares new examples to those it has seen before and finds the most similar generalized exemplar in memory.

NGE theory makes several significant modifications to the exemplar-based model. It retains the notion that examples should be stored verbatim in memory, but once it stores them, it allows examples to be *generalized*. In NGE theory, generalizations take the form of hyperrectangles in d-dimensional Euclidean space, where the space is defined by the feature values measured for

each example. The hyperrectangles may be nested one inside another to arbitrary depth, and inner rectangles serve as *exceptions* to surrounding rectangles [62]. Each new training example is first classified according to the existing set of classified hyperrectangles by computing the distance from the example to each hyperrectangle. If the training example falls into the nearest hyperrectangle, then the nearest hyperrectangle is extended to include the training example. Otherwise, the second nearest hyperrectangle is tried. This is called as *second match heuristic*. If the training example falls into neither the first nor the second nearest hyperrectangle, then it is stored as a new (trivial point) hyperrectangle.

A new example will be classified according to the class of the nearest hyperrectangle. Distances are computed as follows: If an example does not fall into any existing hyperrectangle, a weighted Euclidean distance is computed. If the example falls into a hyperrectangle, its distance to that hyperrectangle is zero. If there are several hyperrectangles having equal distances, the smallest of these is chosen. The EACH algorithm computes the distance between a new data point e and a hyperrectangle H, by measuring the Euclidean distance between these two objects as follows:

$$D_{e,H} = w_H \sqrt{\sum_{f=1}^{n} (w_f \frac{d(e,H,f)}{max_f - min_f})^2} \tag{2.3}$$

where

$$d(e,H,f) = \begin{cases} e_f - H_{f,upper} & e_f > H_{f,upper} \\ H_{f,lower} - e_f & e_f < H_{f,lower} \\ 0 & otherwise \end{cases} \tag{2.4}$$

where w_H is the weight of the exemplar H, w_f is the weight of the feature f, e_f is the value of the fth feature on example e, $H_{f,upper}$ or $H_{f,lower}$ are the upper end of the range and lower end, respectively, on fth feature on exemplar H, max_f and min_f are the minimum and maximum values of that feature, and n is the number of features recognizable on e.

The EACH algorithm finds the distance from e to the nearest face of H. There can be several alternatives to this, such as using the center of H. If

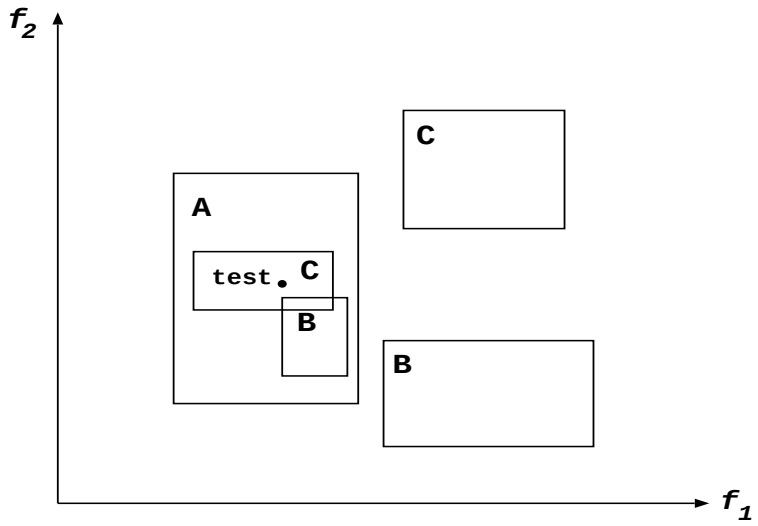

Figure 2.2. An example concept description of the EACH algorithm in a domain with two features.

the hyperrectangle H is a point hyperrectangle, representing an individual example, then the upper and lower values becomes equal.

If a training instance e and generalized exemplar H are of the same class, that is, a correct prediction has been made, the exemplar is generalized to include the new instance if it is not already contained in the exemplar. However, if the closest hyperrectangle has a different class then the algorithm modifies the weights of features so that the weights of the features that caused the wrong prediction is decreased. This is how the EACH algorithm learns feature weights.

The original NGE was designed for domains with continuous features only. Discrete features require a modification of the distance and area computations for NGE.

In Figure 2.2, an example concept description constructed by the EACH algorithm is presented for a domain with two features f_1 and f_2. Here, there are three classes, A, B and C, and their descriptions are rectangles (exemplars) as shown in Figure 2.2. The rectangle A contains two smaller rectangles, B and C, in its region. Therefore, B and C are *exceptions* inside the rectangle A. The NGE model allows exceptions to be stored quite easily inside hyperrectangles, and exceptions can be nested any number of levels. The test instance, that is

marked as *test* in Figure 2.2, falls into the rectangle C, since it is smaller, so the prediction will be the class value C for this test instance.

2.1.3 Feature Projection Based Learning

The Feature Projection Based Learning algorithms all generalize the feature projections of the training instances in learning the concept descriptions. The previously studied techniques categorized as feature projection based learning methods under exemplar-based generalization are the Classification by Feature Partitioning (CFP) [31, 32, 71], the Classification by Overlapping Feature Intervals (COFI) [73], Feature Intervals Algorithms (FIL) [7], and the k Nearest Neighbor on Feature Projections (k-NNFP) [8] algorithms. The FPB models are further classified as Single-Class Intervals and Multi-Class Intervals as shown in Figure 2.1. The CFP and the FIL algorithms are Single-Class Intervals algorithms. The COFI algorithm is a Multi-Class Intervals algorithm. On the other hand, the k-NNFP algorithm also based on feature projections can be categorized as both Single-Class and Multi-Class. The classification of unseen instances in the FPB models are based on a voting among the individual predictions made by using the local information individually stored on each feature. The discussion of these algorithms are presented in Chapter 3 in detail.

2.2 The Nearest Neighbor Classifier

One of the most common and simplest classification algorithms is the Nearest Neighbor (NN) algorithm. In the literature, nearest neighbor algorithms for learning from examples have been studied extensively [17, 18, 24]. Although other machine learning techniques such as decision trees [55] have been the subject of much recent experimental work, the nearest neighbor algorithms continues to stay as an accurate learning technique [64]. The nearest neighbor learning algorithms have been shown to work as well as other machine learning methods despite their simplicity [16, 18, 68]. It seems that nearest neighbor

methods will continue to be cited as a basis of comparison with other methods.

The NN classification algorithm is based on the assumption that examples which are closer in the instance space are of the same class. That is, unclassified ones should belong to the same class as their nearest neighbor in the training dataset. After all the training set is stored in memory, a new example is classified as the class of the nearest neighbor among all stored training instances. Although several distance metrics have been proposed for NN algorithms [64], the most common one is the Euclidean distance metric. The Euclidean distance between two instances $x =< x_1, x_2, ..., x_d, C_x >$ and $y =< y_1, y_2, ...y_d, C_y >$ on an d dimensional space is computed as:

$$dist(x, y) = \sqrt{\sum_{f=1}^{d} w_f \times diff(f, x, y)^2} \tag{2.5}$$

$$diff(f, x, y) = \begin{cases} |x_f - y_f| & \text{if } f \text{ is linear} \\ 0 & \text{if } f \text{ is nominal} \quad \text{and } x_f = y_f \\ 1 & \text{if } f \text{ is nominal} \quad \text{and } x_f \neq y_f \end{cases} \tag{2.6}$$

Here w_f denotes the weight for feature f and for all features $w_f = 1$ in standard NN and $diff(f, x, y)$ denotes the difference between the values of instances x, and y on feature f. Note that this metric requires the normalization of all feature values into a same range by computing the maximum and minimum.

Although several techniques have been developed for handling unknown (missing) feature values [57, 58], the most common approach is to set them to the mean of the values on corresponding feature.

Stanfill and Waltz introduced the Value Difference Metric (VDM) to define the similarity for discrete (nominal) features and empirically demonstrated its benefits [68]. The VDM computes a distance for each pair of the different values a nominal feature can assume. It essentially compares the relative frequencies of each pair of distinct values across all classes. Two feature values have a small distance if their relative frequencies are approximately equal for all output classes. Cost and Salzberg presented a nearest neighbor algorithm that uses a

modification of VDM, called MVDM (Modified Value Difference Metric) [16]. The main difference between MVDM and VDM is that their method's feature value differences are symmetric. This is not the case for VDM. A comparison of MVDM and Bayesian classifier is presented in [59].

NN algorithm can be quite effective when the features of the domain are equally important. However, it can be less effective when many of the features are misleading or irrelevant to classification. To avoid this problem, weakly relevant features should have lower weights and strongly relevant features should have higher weights in Equation 2.5 where the weight of a feature f is represented by w_f. Assigning different weights to the features of the instances before applying the NN algorithm distorts the feature space, modifying the importance of each feature to reflect its relevance to classification. In this way, similarity with respect to relevant features becomes more critical than similarity with respect to irrelevant features. A weight learning method for the NN algorithm will be described in Chapter 6.

In fact NN is a specialization of a more general algorithm called the k–Nearest Neighbor algorithm (k–NN), which classifies a new instance by a majority voting among its k (≥ 1) nearest neighbors using some distance metrics in order to reduce the effect of noisy training instances.

An average-case analysis of k-NN classifiers for Boolean threshold functions on domains with noise-free Boolean features and a uniform instance distance distribution is given by Okamoto and Satoh [53]. They observed that the performance of the k-NN classifier improves as k increases, then reaches a maximum before starting to deteriorate, and the optimum value of k increases gradually as the number of training instances increases.

2.3 Decision Trees

One of the most well known and widely experimented inductive learning approaches is decision trees. The original idea goes back to the work by Hunt, Marin and Stone [37]. Other researchers have arrived independently at similar

methods such as the CART system [13]. This same idea also produces ID3 [55], PLS1 [60], ASSISTANT [14]. The principal name for Quinlan's famous decision tree induction program is C4.5 [58], which is the descendant of an earlier version called ID3.

Given a set of preclassified instances, decision tree learning systems generate a tree structure that can be used to classify new instances. Each instance is described by a set of feature values, which can have either continuous (linear) or discrete (nominal) values, with the corresponding class (category) label. A decision tree is either

- a *leaf*, indicating a class, or
- a *decision node* that specifies some test to be carried out on a single feature value, with one branch and subtree for each possible outcome of the test.

A new test instance is classified by starting at the root of the tree and moving through it until a leaf is encountered. At each nonleaf decision node, the test at the node shifts the search to the branch determined by the corresponding feature value of the test instance. When this process finally reaches a leaf node, the class label stored at the leaf node is returned as the predicted class value of the test instance.

Decision trees are built using a divide and conquer approach. The skeleton of decision tree construction from a set T of training instances is simple. Let the classes be denoted C_1, C_2, ..., C_k. There are three possibilities:

- T contains one or more instances, all belonging to a single class C_j:
  The decision tree for T is a leaf identifying class C_j.
- T contains no cases:
  The decision tree is again a leaf, but the class to be associated with the leaf must be determined from information other than T. For example, C4.5 uses the most frequent class at the parent of this node.

- T contains cases that belong to a mixture of classes:
  In this situation, the idea is to refine T into subsets of instances that seem to be single–class collections of instances. A *test* is chosen, based on a single feature, that has one or more mutually exclusive outcomes O_1, O_2, ..., O_n. T is partitioned into subsets T_1, T_2, ..., T_n, where T_i contains all the cases in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identified by *test*, and one branch for each possible outcome. The same procedure is applied recursively for each subset of training instances produced by this test. That is, the i^{th} branch leads to the decision tree constructed from the subset T_i.

Each internal node contains a test that will partition the training instances. The most important decision criteria in decision tree induction is how to decide the best test on a given node. One must use some heuristics to find the best decision nodes because the problem of finding the best decision tree is NP-complete. C4.5 uses *information–gain criterion* to evaluate the goodness of a test. Given a set of training instances T and a test X with n outcomes, the information can be found as the weighted sum over the subsets:

$$info_X(T) \;=\; \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times info(T_i) \tag{2.7}$$

The information gain that is gained by partitioning T according to this test X is:

$$gain(X) = info(T) \;-\; info_X(T) \tag{2.8}$$

The information–gain criterion selects the test to maximize this information gain. Although this criterion gave quite good results, it has a strong bias for tests with many outcomes. To overcome this bias problem, another criterion, called *gain ratio criterion* is introduced [58]:

$$gain\ ratio(X) = gain(X)\ /\ split\ info(X) \tag{2.9}$$

where $split\ info(X)$ is defined as:

$$split\ info(X) = -\ \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times log_2(\frac{|T_i|}{|T|}) \tag{2.10}$$

$split\ info$ represents the potential information generated by dividing T into n subsets. Information gain measures the information relevant to classification

arising from the division. Then, the gain ratio is the proportion of information generated by the division that appears helpful.

There are three tests considered in C4.5:

- The standard test on a discrete feature, with one outcome and branch for each possible value of that feature,
- A more complex test, based on a discrete feature, in which the possible values are allocated to a variable number of groups with one outcome for each group rather than each value. This form of test is optionally invoked in C4.5.
- For a continuous feature f, a binary test with outcomes $f \leq V$ and $f > V$, based on comparing the value for f against a threshold value V. To find a threshold value, the instances are first sorted with respect to the values of the feature f. Let those sorted values be v_1, v_2, ..., v_m. The midpoint between each v_i and v_{i+1} is considered as a representative threshold and $m - 1$ such midpoints are all examined as a candidate threshold.

The construction process of a decision tree makes use of a hidden assumption that the outcome of a test for any instance can be determined. The outcome of a test is both required when partitioning a set T into subsets T_i and when classifying a test instance using a decision tree. Since every test is based on a single feature, the outcome of a test can not be determined unless the value of that feature is known. The solution of C4.5 to overcome the problem of unknown (missing) feature values in training, is to evaluate the tests by simply ignoring the instance with unknown value i.e. excluding that instance in the gain calculations. Then the partition is done according to the selected test and the instance with missing value is inserted in all subsets with a probability to be in that subset. When classifying a test instance, if a decision node having a test that is unknown is reached, all possible outcomes are explored and the probabilistic classifications are combined arithmetically. Then the class with the highest probability is the predicted class of the test instance.

Another problem with decision trees is that the resulting tree of C4.5 is often a very complex tree that "overfits the data" by inferring more structure than is justified by the training instances. A decision tree is not usually simplified by deleting the whole subtree in favor of a leaf. Instead, the idea is to remove parts of the tree that do not contribute to classification accuracy on unseen instances, producing something less complex and thus more comprehensible. This process is known as the *pruning* [58].

A simpler decision tree learning approach, called 1R system, is later proposed by Holte [36]. It is based on the rules that classify an object on the basis of a single feature that is, they are 1-level decision trees, called *1-rules* [36]. The input of the 1R algorithm is a set of classified training instances and the output is a concept description in the form of 1-rule. Since each feature is considered separately in 1R system, missing feature values can be simply ignored instead of ignoring the instance containing missing value. Then, one of the concept descriptions on a feature is chosen as the final concept description by selecting the one that makes the smallest error on the training dataset.

Holte used sixteen datasets, fourteen of which were selected from the collection of UCI-Repository [51], to compare 1R and C4.5 [36]. The main result of comparing 1R and C4.5 was an insight into the tradeoff between simplicity and accuracy. 1R rules are only a little less accurate (about 3 percentage points) than C4.5's pruned decision trees on almost all of the datasets. Decision trees formed by C4.5 are considerably larger in size than 1-rules. Holte shows that simple rules such as 1R are as accurate as more complex rules such as C4.5.

Another decision tree algorithm is T2 (decision trees of at most 2-levels) [12]. Its computation time is almost linear in the size of training set. The T2 algorithm is evaluated on 15 common real-world dataset. It is shown that the most of these datasets, T2 provides simple decision trees with little or no loss in accuracy compared to C4.5.

SADT [34] and OC1 [52] are decision tree induction methods, which partition instances using oblique hyperplanes. Standard decision tree techniques, such as C4.5 [58], partition a set of points with axis-parallel hyperplanes whereas oblique decision tree algorithms attempts to find hyperplanes at any

orientation. SADT and OC1 use a randomized approach for generating decision trees using non-axis-parallel hyperplanes. The purpose of these more general techniques is to find smaller but more accurate decision trees and the experiments have shown that in some cases they produce small trees without losing predictive accuracy.

2.4 Naive Bayesian Classifier

Bayesian classifier originating from work in pattern recognition is a probabilistic approach to inductive learning [24, 29]. Given the observed feature values for an instance and the prior probabilities of classes, the *a posteriori* probability that an instance belongs to a class is estimated. The class with the highest estimated probability is predicted as the class of the instance. Bayesian classifiers assume that features may be statistically dependent. On the other hand, *Naive Bayesian Classifier* assumes that features are independent.

Bayes Decision Theory is a probabilistic approach to the problem of pattern classification. The prediction of a class label depends on probability values and it is assumed that all of the relevant probability values are known.

Suppose we are given a domain defined by d features and with k classes. The classification problem is to predict a class among k classes for the unseen example using the concept description induced from training instances. The probabilistic representation of a concept stores probabilistic information about each class. This information includes $P(C_i)$, which specifies the *a priori* probability that one will observe a member of class C_i, and a set of conditional probabilities, specifying a probability distribution for each feature. From this probabilistic concept description and a given feature value vector $\mathbf{x} =< x_1, \ldots, x_d >$ of the new example to be classified, the *a posteriori* probability $P(C_i|\mathbf{x})$ for each class are computed. Bayes rule allows us to compute the *a posteriori* probability $P(C_i|\mathbf{x})$ using the *a priori* probability $P(C_i)$ and the class conditional density $p(\mathbf{x}|C_i)$:

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})} \tag{2.11}$$

where

$$p(\mathbf{x}) = \sum_{i=1}^{k} p(\mathbf{x}|C_i)P(C_i) \tag{2.12}$$

There are many different ways to represent classifiers. One way is in terms of a set of discriminant functions $g_i(\mathbf{x})$, $i = 1, .., k$ where k is the number of classes. The classifier is set to assign a feature vector $\mathbf{x}$ to class C_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \qquad for \;\; all \;\; i \neq j. \tag{2.13}$$

Thus, the classifier can be viewed as a machine that computes discriminant functions and selects the class (category) whose discriminant function has the largest value.

For the general case we can let $g_i(\mathbf{x}) = P(C_i|\mathbf{x})$, so that the maximum discriminant function corresponds to the maximum *a posteriori* probability. The classifier would simply select the class C_i with maximum $P(C_i|\mathbf{x})$.

The choice of discriminant functions is not unique. More generally, if every $g_i(x)$ is replaced by $f(g_i(x))$, where f is a monotonically increasing function, the resulting classification is unchanged. This observation can lead to significant analytical and computational simplifications. In particular, for minimum-error-rate classification, any of the following choices gives identical classification results, but some can be much simpler to understand or to compute than others [24]:

$$g_i(\mathbf{x}) = P(C_i, \mathbf{x}) \tag{2.14}$$

$$g_i(\mathbf{x}) = \frac{P(\mathbf{x}|C_i)P(C_i)}{\sum_{j=1}^{c} P(\mathbf{x}|C_j)P(C_j)} \tag{2.15}$$

$$g_i(\mathbf{x}) = P(\mathbf{x}|C_i)P(C_i) \tag{2.16}$$

$$g_i(\mathbf{x}) = logP(\mathbf{x}|C_i) + logP(C_i) \tag{2.17}$$

Even though the discriminant functions can be written in a variety of forms, the decision rules are equivalent. The effect of any decision rule is to divide

```
train(TrainingSet):
begin
    for each feature f
        for each class c
            find all distinct values of f in examples of class c in TrainingSet
            for each distinct value v
                count the number of examples of class c /* call count[v, c] */
end.
```

Figure 2.3. The Training in the NBC Algorithm.

the feature space into k decision regions, $R_1, .., R_k$. If $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $i \neq j$, then $\mathbf{x}$ is in R_i and the decision rule calls for us to assign $\mathbf{x}$ to C_i. The regions are separated by decision boundaries, surfaces in feature space where ties occur among the largest discriminant functions. If R_i and R_j are contiguous, the equation for the decision boundary separating them is

$$g_i(\mathbf{x}) = g_j(\mathbf{x}). \tag{2.18}$$

While this equation may appear to take different forms depending on the forms chosen for the discriminant functions, the decision boundaries are, of course, the same. For points on the decision boundary, the classification is not uniquely defined. For a Bayes classifier, the conditional risk associated with either decision is the same, and it does not matter how ties are broken. No matter which discriminant function is used, $P(\mathbf{x}|C_i)$ has somehow to be computed. Since Naive Bayesian Classifier assumes that features are independent, it can be computed as follows:

$$P(\mathbf{x}|C_i) = \prod_{f=1}^{n} P(x_f|C_i). \tag{2.19}$$

The training in a particular implementation of the Naive Bayesian Classifier is given in Figure 2.3 and the classification is given in Figure 2.4. This particular implementation, which is called as NBC, estimates the conditional probability density functions $p(x_f|C_i)$ for a given feature value x_f for the f^{th} feature using the frequency of observed instances around x_f. This probability density estimation algorithm is given in Figure 2.5. $P(x_f|C_i)$ for nominal

```
classify(e):
begin
    for each class c
        /* class_count[c]: number of examples that have the class value c */
        g[c] = class_count[c] / (number of training examples)

        for each feature f
            g[c] = g[c] * probability(e, f, c)

    return class c with highest g[c]
end.
```

Figure 2.4. The Classification in the NBC Algorithm.

features is the ratio of the number of training examples of class C_i with value x_f for feature f over total number of training examples of class C_i. $P(x_f|C_i)$ for continuous features is computed using the frequency of examples of class C_i with the smallest value larger than x_f and the largest value smaller than x_f. Instead of this approach for continuous features used in NBC, continuous features are discretized into 10 bins of uniform size in MLC++ [43] and the conditional probability is computed as done for nominal features using the frequency counts. If there is a class value with zero counts, that class is ignored and never be predicted. If there are no examples for class C_i with feature value x_f for feature f, the conditional probability, $P(x_f|C_i)$, will be zero. In our current NBC implementation, for continuous features $P(x_f|C_i)$ is estimated using the frequency of examples of class C_i having values around x_f. But for nominal features, the zero conditional probability is kept as it is and the conditional probability, $P(\mathbf{x}|C_i)$, becomes zero, which eliminates C_i from consideration. Some other approaches to avoid a zero estimate for $P(\mathbf{x}|C_i)$ are proposed by Kohavi [42].

Some other implementations of Naive Bayesian Classifier assume a particular distribution such as normal distribution for continuous features. The structure of a Bayes classifier is determined primarily by the conditional probability densities $p(x_f|C_i)$ and the probability density function for normal distribution

```
probability(e, f, c):
begin
    if f is a nominal feature
        return ( count[e_f, c] / class_count[c] )

    else /* f is continuous */

        if e_f has seen in the training examples of class c then

            if e_f is the only value of f in the training set
                return (1.0)

            else if e_f is the smallest value of f in the training set
                count = count[e_f, c] / (smallest_of_larger - e_f)

            else if e_f is the largest value of f in the training set
                count = count[e_f, c] / (e_f - largest_of_smaller)

            else /* e_f is in the middle of some values */
                difference = smallest_of_larger - largest_of_smaller
                count = count[e_f, c] / difference * 2

        else if e_f is smaller than the smallest value of f in examples of class c
            count = count[smallest, c] / (smallest - e_f)

        else if e_f is larger than the largest value of f in examples of class c
            count = count[largest, c] / (largest - e_f)

        else /* there are values smaller and larger than e_f */
            difference = smallest_of_larger - largest_of_smaller
            count = (count[smallest_of_larger, c]+count[largest_of_smaller, c])
                                                / 2 / difference

        return ( count / class_count[c] )
end.
```

Figure 2.5. Computing the *a posteriori* probabilities in the NBC Algorithm.

is as follows:

$$p(x_f) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-(x_f-\mu)^2/2\sigma^2}. \tag{2.20}$$

The normal density is completely specified by two parameters, the *mean* μ and the *variance* σ^2. We say that x_f is normally distributed with mean μ and variance σ^2. Since we are trying to find the conditional probability distribution of values x_f given that the class value is C_i, we compute the mean and variance for each class separately. That is, the mean and the variance of the x_f values of each class C_i examples define the conditional probability distribution of feature f given that the class is C_i. Another Naive Bayesian Classifier developed during this thesis assumes normal distribution for continuous features and treats nominal features as usual, which we call NBCN in short throughout this thesis.

Naive Bayesian Classifiers handle missing (unknown) feature values by simply ignoring the feature with the missing value instead of ignoring the whole instance. When an instance x has an unknown value for feature f, the conditional probabilities ($P(x_f|C_i)$) of each class C_i are assigned to 1, which has no effect on the product of probabilities distributed by each feature. Therefore, the probabilities of classes are computed by only the features having known values and the features having unknown values are simply ignored.

Naive Bayesian Classifier assumes that the features are independent from each other. It is a classical classification algorithm originating from work in pattern recognition and has been found successful in terms of classification accuracy in many domains, including medical diagnosis, compared with Assistant, which is an ID3-like [55] inductive learning system. It has also been concluded that induction of decision trees is relatively slow as compared to Naive Bayesian Classifier [44].

Chapter 3

Feature Projection Based Learning Models

Feature projections for knowledge representation constitutes the background for this thesis. Given a set of training instances with class labels, knowledge for concepts (or classification) is maintained as the projections of the training set on each feature dimension separately. The rationale behind this knowledge representation is that humans maintain knowledge in this form, especially in medical domains. An example for this approach is the CRiteria Learning System (CRLS) [72], which aims to learn decision rules in the form of criteria tables as humans do. The most important advantage of this representation is that the projections of the feature values can be organized for each feature in a way that it reduces the time for the computation of similarity to all training instances for nearest neighbor like techniques. An additional advantage is the easy and natural handling of missing feature values. On the other hand, the disadvantage of the representation by feature projections is that it is possible to lose the knowledge conveyed by the combination of the individual information encoded by several features together.

First, I will describe the k–Nearest Neighbor on Feature Projections (k-NNFP) algorithm [8], which is a new version of the classical k-NN algorithm and uses feature projections knowledge for representation.

After describing the k-NNFP algorithm, the first feature projection based classifier called as the Classification by Feature Partitioning (CFP) algorithm [32] is explained. The basic unit of representation —a *disjoint feature interval*— in the CFP algorithm represents only one class for a range of values in a feature. Then the Classification by Overlapping Feature Intervals (COFI) algorithm [73] was developed to make the basic unit of representation —an *overlapping feature interval*— more powerful by allowing it to represent more than one class. The common property of the CFP and the COFI algorithms is that they both consider each feature separately in an incremental manner. Incremental learning algorithms are sensitive to the presentation order of the instances. In order to prevent such an effect, the next feature projection based algorithms are developed in non-incremental fashion. One of them is the set of Feature Intervals Learning (FIL) algorithms [7] and the other is the set of Voting Feature Intervals (VFI) algorithms [22], which make up the main subject of this thesis. Both FIL and VFI algorithms do not require any domain dependent parameters as the CFP and COFI algorithms do.

Next section describes the k-NNFP algorithm. Section 3.2 and 3.3 discusses the CFP and the FIL algorithms respectively. The COFI algorithm is explained in Section 3.4.

3.1 K Nearest Neighbor Classification on Feature Projections

K Nearest Neighbor on Feature Projections (k-NNFP) [8] is a non-incremental supervised learning algorithm which also represents the concept descriptions as the projections of the training instances on each feature dimension. The classification is based on a majority voting on individual classifications of each feature. To determine the individual classifications of each feature, k–nearest neighbor algorithm is applied on that feature projection of instances. The k-NNFP algorithm based on feature projections can be categorized as both Single-Class and Multi-Class under FPB learning models (see Figure 2.1). It can be categorized as Single-Class because the projections of single instances

are kept without any generalization in the k-NNFP algorithm. It can also be categorized as Multi-Class when $k > 1$ because in the majority voting several classes are represented by the k neighbors of the instance to be classified.

In the training phase, each training instance is stored simply as its projections on each feature dimension. If the value of a training instance is missing for a feature, that instance is not stored on that feature. In order to classify an instance, a preclassification separately on each feature dimension is performed. During this preclassification, the k–NN algorithm on that single dimension is used. That is, for a given test instance t and feature f, the preclassification for $k = 1$ will be the class of the training instance whose value on feature f is the closest to that of the t. For a larger value of k, the preclassification is a bag (multiset) of classes of the nearest k training instances. In other words, each feature has exactly k votes, and gives these votes for the classes of the nearest training instances. In some cases, especially for nominal features, there may be ties to determine the first k nearest neighbors. In such cases ties are broken randomly. For the final classification of the test instance t, the preclassification bags of each feature are collected using bag union. Finally, the class that occurs most frequently in the collection bag is predicted to be the class of the test instance. In other words, each feature has exactly k votes, and gives these votes for the classes of the nearest training instances. Since each feature is processed separately, no normalization of feature values is needed.

The distance between the values on a feature dimension is computed using $diff(f, x, y)$ metric as follows:

$$diff(f, x, y) = \begin{cases} |x_f - y_f| & \text{if } f \text{ is linear} \\ 0 & \text{if } f \text{ is nominal and } x_f = y_f \\ 1 & \text{if } f \text{ is nominal and } x_f \neq y_f \end{cases} \tag{3.1}$$

The k-NNFP algorithm handles unknown feature values in a straight forward manner. If the value of a test instance for a feature f is missing, then feature f does not participate in the voting for that instance. The final voting is done between the features for which the test instance has a known value. That is, unknown feature values are simply ignored.

A set of experiments have been performed to evaluate the k-NNFP algorithm on some real-world datasets for $k = 1, 2, \ldots 10$ [8]. These experiments show that the k-NNFP algorithm achieves comparable accuracy with the k-NN algorithm. On the other hand, the average running time of the k-NNFP algorithm is much less than that of the k-NN algorithm. The reason for this is that the k-NNFP algorithm treats feature values independently, whereas the k-NN algorithm treats all instances as points in d-dimensional Euclidean space. The k-NNFP algorithm stores the feature projection of the training instances in a sorted order. Therefore, the classification of a new instance requires a simple search of the nearest training instance value. On the other hand, in the k-NN algorithm, a new search must be done for each test instance in the whole Euclidean space. The experiments also have shown that the classification accuracy of the k-NNFP algorithm usually increases when the value of k increases [8]. On the other hand, it has been observed that the increase in the value of k does not result in a parallel increase in the accuracy of the k-NN algorithm.

3.2 Classification by Feature Partitioning

The Classification by Feature Partitioning (CFP) algorithm is a method for learning from examples that uses feature projections for knowledge representation [31, 32, 71]. It is an incremental supervised inductive learning algorithm where instances are stored by their feature projections over each feature dimension. In the training phase, *disjoint feature intervals* (also called as *interval* in this section shortly) of concept definitions are constructed by generalization and specialization. An interval representing a single class is a basic unit of knowledge representation in this algorithm, therefore the CFP algorithm can be categorized as a Single-Class FPB learning algorithm (see Figure 2.1. For each interval, lower and upper bounds of the feature values, the associated class, and the number of instances it represents are maintained.

Initially, an interval is a point on a feature dimension. It can be extended

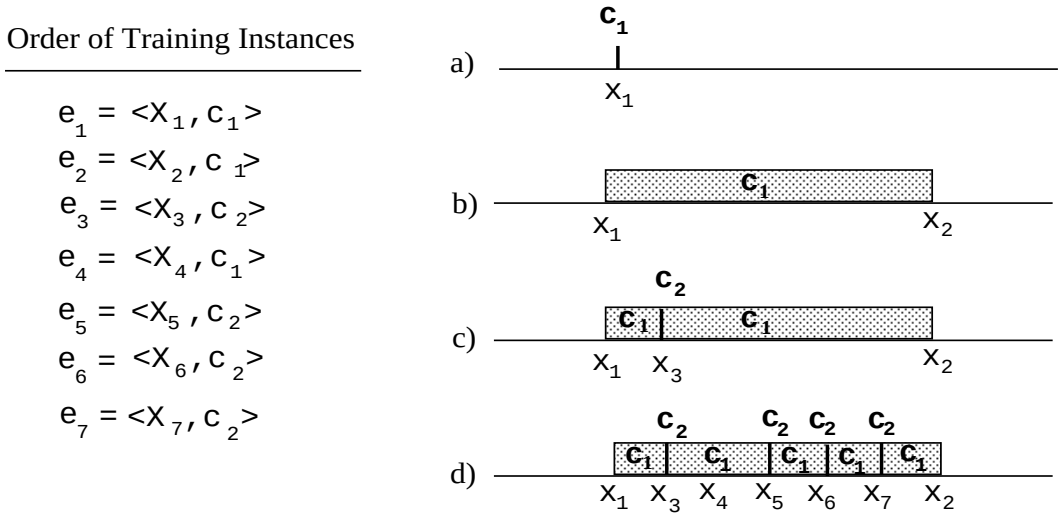

Figure 3.1. Construction of intervals in the CFP algorithm: (a) after e_1 is processed, (b) after e_2 is processed, (c) after e_3 is processed, (d) after all training instances are processed.

through generalization with other neighboring points in the same feature dimension. In order to avoid overgeneralization, a parameter, called generalization limit (D_f), is provided by the user as a domain dependent parameter. Before generalizing an interval on a feature dimension f to cover a new point, the distance between the interval and the new point must be less than D_f. Otherwise, the new value forms a new point interval on that feature dimension. During training, if the feature value of a training instance falls into an interval properly with the same class, simply the representativeness value is incremented by one. However, if it falls into an interval with a different class than that of the instance, specialization of that interval is made by dividing it into subintervals and inserting a point interval for the new value in between them. The representativeness values of these new intervals are updated according to their sizes.

Figure 3.1 shows the construction of intervals in the CFP algorithm. Let us consider a training dataset with only one feature. The first instance e_1 forms a point interval at the feature value x_1 on this feature dimension. After the second instance e_2, a range interval is constructed and its lower and upper bounds are x_1 and x_2, respectively, since these two instances have the same

class, as shown in Figure 3.1.b. Here, we assume that the generalization distance is greater than the difference between x_1 and x_2. The third instance with different class, C_2, specializes the interval into two subintervals by inserting a new point interval in between them. In Figure 3.1.c, the fourth instance e_4 with class C_1 just increases the representativeness count of the interval that covers it. Let us assume the next three instances belong to class C_2, and their related feature values are between x_4 and x_2. In this case, the interval $[x_3, x_2]$ in Figure 3.1.b is partitioned into four intervals for class C_1 and point intervals are constructed for the second class C_2 as shown in Figure 3.1.d.

During the training process in the CFP algorithm, feature weights and feature intervals of each concept are learned in an incremental manner. Initially, all feature weights are taken as 1. Assume that a new training example is misclassified by a feature f. Then the weight of that feature (w_f) is decreased by multiplying it by (1 - $\triangle$). Otherwise, it is increased by multiplying it by (1 + $\triangle$). Here, $\triangle$ is the global feature adjustment rate, given as a parameter to CFP.

Classification of an unseen instance is based on a vote taken among the predictions made by each feature separately. The prediction of a feature is determined by the value of that instance on that feature. If it falls into an interval with a known class, then the prediction is the class of that interval. If it falls on a point interval, the class with highest representativeness value is chosen among all the intervals at that point. If it doesn't fall in any interval, then no prediction for that feature is made. The effect of the prediction of a feature in the voting is proportional to the weight of that feature. The final classification is based on a weighted majority voting taken among local predictions of features.

In the CFP algorithm, feature intervals are constructed as disjoint sets of feature values. However, intervals may have common boundaries. In such cases, the representativeness values of the intervals are used to determine the prediction: the class label of the interval which has the maximum representativeness value is predicted.

Several extensions to the CFP algorithm have been presented in order to

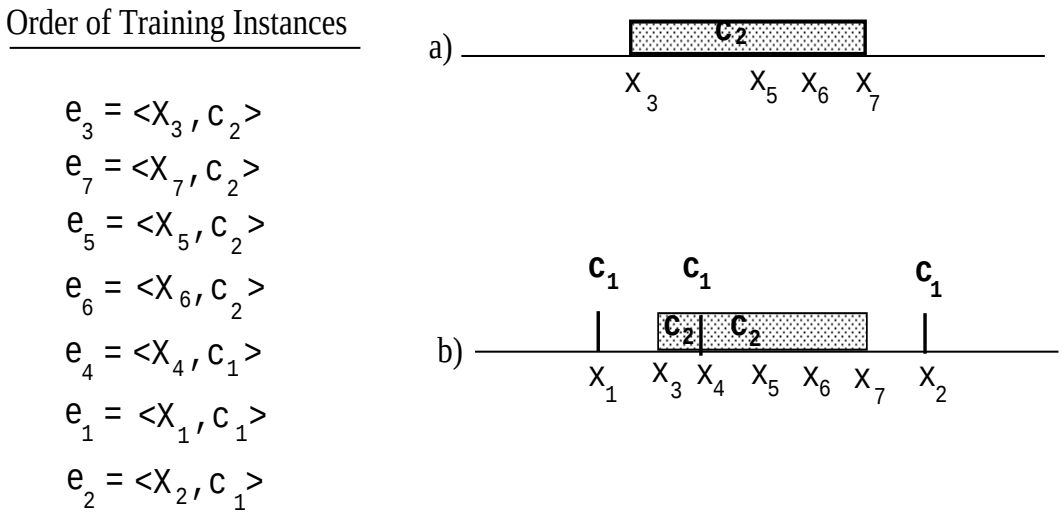

Figure 3.2. Construction of intervals in the CFP algorithm by changing the order of the training instances. Note that here the same set of instances in Figure 3.1, but in a different order, is used as the training set: (a) after e_3, e_7, e_5 and e_6 are processed, (b) after all instances are processed.

handle noisy values [70, 71] and determine the domain dependent parameters (D_f and $\triangle$) of the CFP algorithm [31].

In the noise-tolerant version of the CFP algorithm, feature intervals that are believed to be introduced by noisy examples are removed from the memory [71]. A new parameter, called confidence threshold (or level) is introduced to control the process of removing the intervals from the concept description. The confidence threshold and observed frequency of the classes are used together to decide whether an interval is noisy or not.

In order to learn feature weights and domain dependent parameters of the CFP algorithm, a hybrid system, called GA-CFP, which combines a genetic algorithm with the CFP algorithm has been developed [31]. The genetic algorithm is used to determine a very good set of domain dependent parameters ($\triangle$ and D_f for each feature) of the CFP, even when trained with a small set of the data set[1]. An algorithm that hybridizes the classification power of the feature partitioning CFP algorithm with the search and optimization power of the genetic algorithm, called GA-CFP, requires more computation than the

[1]For example, 20% of all the training data set might be used.

CFP algorithm, but achieves improved classification performance.

Figure 3.2 illustrates a limitation for the CFP algorithm. In order to see the effect of the order of presentations of training instances, let us construct intervals by the CFP algorithm by changing the order of training instances. In this case, all instances with class C_2 were processed before other instances with class C_1 in the previous example, then the intervals would have been constructed as shown in Figure 3.2. Firstly, a range interval is constructed for the class C_2 from the first four instances as shown in Figure 3.2a, and then three point intervals are constructed for the last three instances of class C_1 as in Figure 3.2b. The concept descriptions (intervals) in Figure 3.1 and Figure 3.2 are very different from each other although the same training instances were processed. This indicates that the order of the instances is very important and it affects the resulting concept descriptions considerably. The different concept descriptions can classify a test instance as of different classes. For example, the test instance $< x_8, ? >$ where $x_5 < x_8 < x_6$ will be classified as C_1 by the intervals constructed in Figure 3.1 and as C_2 according to feature intervals in Figure 3.2.

The FIL algorithms [7] are non-incremental learning algorithms, thus offer a solution to this problem. They are given all the training instances at once, and constructs intervals independently from the presentation order of the training instances. Next section will describe the FIL algorithms and illustrate the construction of intervals for the same dataset in Figures 3.1 and 3.2.

3.3 Feature Intervals Learning Algorithms

Feature Intervals Learning (FIL) algorithms are a set of non-incremental supervised inductive learning algorithms that also use feature projections to represent the concept description [7]. From a set of training instances, FIL algorithms construct *disjoint intervals* (also called as *interval* in this section shortly) for each feature. An interval in the FIL algorithms represents a single class, therefore the FIL algorithms can be categorized as Single-Class FPB learning algorithms. In the basic FIL algorithm FI1, an interval is represented

by four parameters: lower bound, upper bound, representativeness count and associated class label. Lower and upper bounds of an interval are the minimum and maximum feature values that fall into the interval respectively. Representativeness count is the number of the instances that the interval represents, and the class label is the associated class of the interval. An interval is either a *point interval*, whose lower and upper bounds are the same or a *range interval*, whose upper bound is greater than its lower bound. A point interval is either constructed from a *single-class point*, which is a value on a feature dimension that belongs to a single class label or a *multi-class point*, which is a value on a feature dimension that belongs to more than one class label. The FIL algorithms construct the concept description by generalizing neighboring same single-class points into range intervals. These range intervals are disjoint; that is, a range interval represents only one class. However, multi-class points represent more than one class and in that case a set of point intervals are constructed for multi-class points. Therefore, both point and range intervals constructed by the FIL algorithms represent a single class.

The classification is based on a majority voting taken among the individual predictions of features. The classification of a feature is based not only on the value of the test instance on that feature dimension but also on the feature intervals constructed during the training phase. Each feature predicts only a single class. FIL algorithms assume that features may have different levels of relevances. Assuming equal relevance is a special case of weighted-voting; that is, each feature contributes to voting process with equal weights. The feature weights are given to the FIL algorithms externally by the user. If they are not given, then all features assume equal weights; so, each feature has the same voting power in the determination of the final class prediction.

The classification on a feature is simply a search process on that feature dimension. If the feature value of the test instance on that feature is contained by an interval, then the prediction will become the class of that interval. If it falls in a multi-class point, the class of the interval with the maximum representativeness count will be predicted. Otherwise, if it is not contained by any interval, then no prediction is made by that feature, hence no vote is taken from that feature. In order to determine the final classification, the

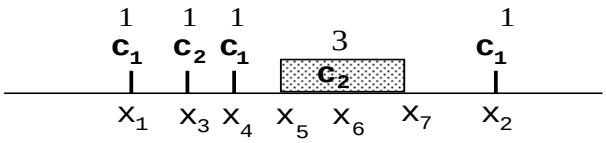

Figure 3.3. Construction of the intervals in the FIL algorithms with using the same dataset as used in Figure 3.1 and Figure 3.2.

individual vote of each feature are summed up. The class which receives the maximum vote is the classification for the test instance. The voting mechanism of the FIL algorithms are similar to that of the CFP algorithm, where each feature votes for only one class and when all the features are equally relevant, the sum of these individual votes determine the final classification.

Figure 3.3 shows the intervals with their representativeness counts constructed by the FIL algorithms from the training datasets given in Figures 3.1 and 3.2. The first three intervals are point intervals constructed from single-class points, the fourth is a range interval, and the last one is again a point interval. The initial version FI1 [7] keeps the representativeness counts —the number of training instances in the corresponding interval— of each interval as shown above each interval in Figure 3.3. Since the FIL algorithms process all the training instances at once, the different orderings of the same set of training instances shown in Figures 3.1 and 3.2 do not result in different set of intervals. On the other hand, CFP might construct different set of intervals for the same training dataset with different orderings. Moreover, FIL algorithms do not require domain dependent parameters such as D_f and $\triangle$ as in the case of the CFP algorithm.

FI2 is the slightly modified version of the initial algorithm FI1 [7]. FI2 keeps the *relative representativeness count*, which is the ratio of the number of training instances to the total number of training instances of the corresponding class rather than absolute representativeness count. This might only change the classification on multi-class points and FI2 is more fair than FI1 in the sense that classes that appear less frequently have now a greater chance to be predicted on multi-class points that they had in FI1.

Since after the training phase is completed, always the same class is predicted from multi-class points in classification, it is unnecessary to store several point intervals for multi-class points. To eliminate this unnecessary storage, FI3 [7] is investigated. The point interval having the maximum representativeness count is chosen as the class of the interval on a multi-class point. The elimination of the point intervals with lower representativeness counts is a kind of *pruning*, but FI3 is careful with this pruning when two classes have very close representativeness counts. Therefore, the point interval kept is assigned a weight equal to the the difference between two maximum representativeness counts divided by the total number of representativeness counts of multi-class points at that feature value. With this modification, the point intervals constructed after pruning contributes to the voting with that weight whereas range intervals and point intervals constructed from single-class points have a voting weight of 1. This weight assigning step in FI3 causes the point intervals constructed from single-class points to have the maximum weight of 1. But these single-class points might be noisy intervals and to decrease the effect of such intervals, FI4 is developed [7]. Training in FI4 is identical to FI3 except normalization of feature interval weights according to class distributions in the training set.

3.4 Classification with Overlapping Feature Intervals

The Classification with Overlapping Feature Intervals (COFI) algorithm is another incremental concept learning algorithm that uses feature projections to generalize knowledge. Classification knowledge learned is maintained in the form of *overlapping feature intervals* (also called as *interval* in this section shortly). The COFI algorithm makes generalizations to construct the concept descriptions from a set of preclassified training instances. Concept descriptions learned by the COFI algorithm are represented as intervals on the class dimensions for each feature. Since the overlapping feature intervals in the COFI algorithm allow the representation of several classes instead of a single class, it can be categorized as a Multi-Class FPB learning algorithm.

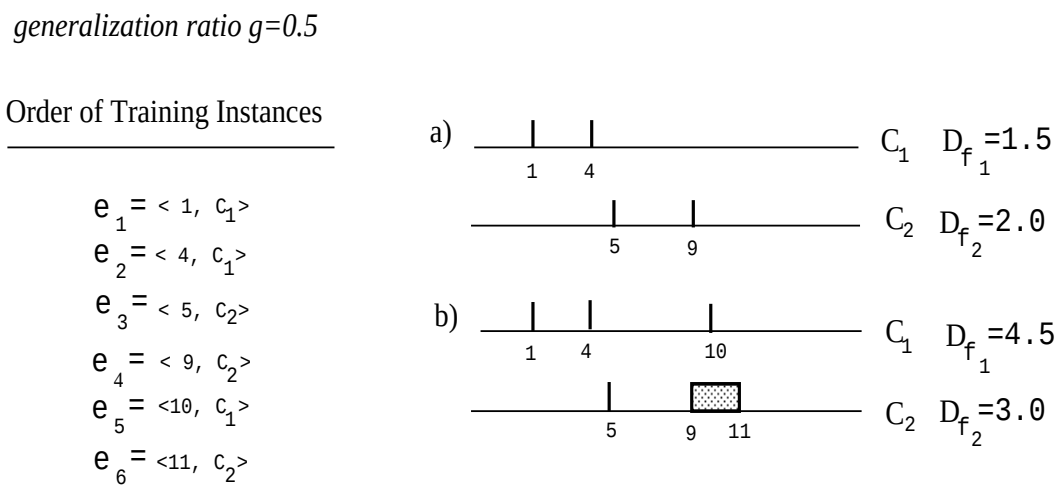

Figure 3.4. An example of construction of intervals in the COFI algorithm: (a) after e_1, e_2, e_3 and e_4 are processed, (b) after e_5 and e_6 are processed.

In the training process, examples are processed one by one and the corresponding intervals on each class dimension for each feature are constructed. The COFI algorithm performs the learning task by constructing the projection of the concepts over each class dimension for each feature, that is, the COFI algorithm learns the overlapping feature intervals for each feature. Learning overlapping feature intervals is done by storing the objects separately in each class dimension for each feature as class intervals of values. An interval consists of four parameters: lower and upper bounds, representativeness count and a class label. Lower and upper bounds of the interval are the minimum and maximum feature values that fall into the interval respectively. Representativeness count is the number of the instances that the interval represents, and finally the class label is the associated class of the interval.

The first task of the training process is the estimation of the current generalization distances, D_f, for each feature f. They are computed as:

$$D_f = (current\_max_f - current\_min_f) * g. \tag{3.2}$$

Here the current maximum and current minimum feature values are the maximum and minimum values of the related feature seen up to the current example and g is the generalization ratio in the range $[0, 1]$. D_f values are

updated by each new training example. Since current maximum and minimum of features change through out the training process, the COFI algorithm is affected also by the presentation order of the training instances. In the first training instance, the maximum and the minimum values are equal to each other and they are the first feature values of the related feature of the training instance. Therefore, initially all the generalization distances are 0 for each feature. If the feature values of the next training instance are different from the previous example's feature values, then one of the maximum and minimum value of the related feature is updated so the generalization distance will also be updated.

After deciding the generalization distance D_f, the intervals should be updated according to D_f. If the distance between the feature value of the new example and the previously constructed intervals is greater than the D_f, then the new example constructs a new point interval. Otherwise, simply the representativeness count of the interval containing it is incremented by 1. The COFI algorithm handles both the linear and nominal feature values. However, the generalization process is applied only to linear type features. Nominal feature values are not generalized, taking D_f as 0 for nominal features.

Figure 3.4 illustrates the construction of overlapping feature intervals in the COFI algorithm. This sample training set with one feature and two classes. The incremental computation of $D_{f,c}$ for each class dimension is also shown in the Figure 3.4. For this example, on the C_1 class dimension only point intervals are constructed since the difference between feature values do not exceed $D_{f,1}$. On the other hand, on the second class dimension, the value of the last training instance forms a range interval, since the difference between feature values is greater than $D_{f,2}$.

The classification of an unseen test instance is based on a majority voting taken among the individual predictions based on the votes of the features. The vote of a feature is based solely on the value of the test instance for that feature. The vote of a feature is not for a single class but rather a vector of votes, called *vote vector*. The size of the vector is equal to the number of classes. An element of the vote vector represents the vote given by the feature to the corresponding class. The vote that a feature gives to a class is the relative representativeness

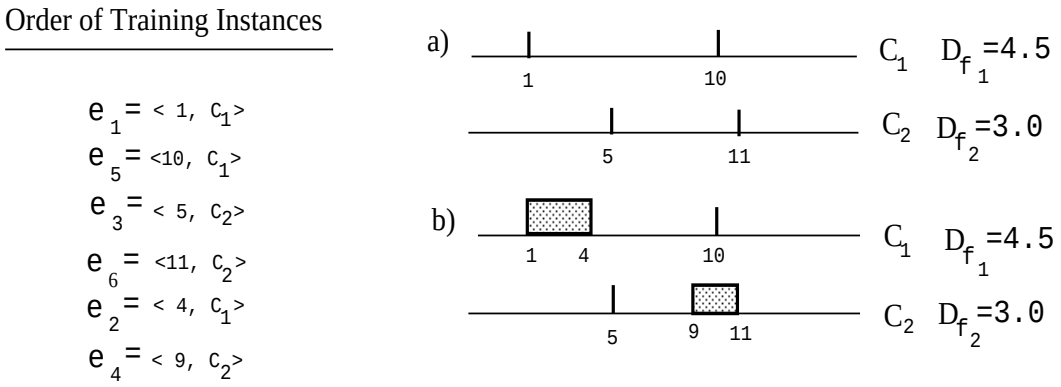

Figure 3.5. An example of construction of intervals in the COFI algorithm using the same set of training instances as in Figure 3.6, but in a different order: a) after e_1, e_5, e_3, and e_6 are processed, b) after e_2 and e_4 are processed.

count of the class interval. The relative representativeness count is the ratio of the representativeness count to the number of examples of the corresponding class label. Since for most of the datasets, the instances are not distributed normally in terms of their class values, this kind of normalization is required. The vote vectors of each feature are added to determine the predicted class. The class which receives the maximum vote is the final class prediction for the test instance.

Generalization in the COFI algorithm is sensitive to the order of the training instances as shown in Figure 3.5, as in the CFP algorithm. Here, the order of training instances are changed among same classes. We get a different construction of overlapping intervals from this ordering of training instances, as shown in Figure 3.5 since the initial generalization distances change.

Chapter 4

Classification by Voting Feature Intervals

This chapter introduces the new classification algorithms developed during this thesis. The common name for a set of non-incremental classifiers is *Voting Feature Intervals* (VFI) and they all use the *feature projections* knowledge representation scheme used in CFP, COFI, k-NNFP, and FIL algorithms described in Chapter 3. They are called "Voting Feature Intervals" because *feature intervals* are constructed on each feature dimension in the learning phase and the corresponding intervals on each feature *votes* for each class in the classification phase of the VFI algorithms. VFI algorithms also consider each feature separately as in the case of Naive Bayesian Classifier as well as the other feature projection based learning methods. A voting scheme is used in classification to combine the individual classifications of each feature similar to other feature projection based methods. The encouraging results and the advantages of the feature projections knowledge representation and classification voting schemes such as speed and handling missing feature values motivated us to come up with this new classification technique. The concept is still represented as projections on each feature dimension separately, but the basic unit of representation —a feature interval— in the VFI algorithms is somewhat different from the intervals of the CFP and the FIL classifiers. Unlike disjoint segments in CFP and disjoint intervals in FIL algorithms, a feature interval can represent instances

from a set of different classes instead of a single class. Thus, the VFI algorithms can be categorized as Multi-Class feature projection based algorithms (see Figure 2.1). The voting scheme used in classification is also modified in the VFI algorithms, such that each feature distributes its vote among classes, whereas in the CFP and the FIL classifiers, a feature votes for only one class. The COFI and the k-NNFP algorithms also have a similar voting mechanism as that of the VFI algorithms such that features also vote for more than one class. But construction of overlapping feature intervals is performed for each class independently from other classes on a feature dimension. The projections of instances with the same class value on a feature dimension are generalized to form intervals by using a dynamic generalization distance computed according to a given generalization parameter. On the other hand, the VFI algorithms use the projection of instances from all classes on a feature dimension at the same time and forms intervals from these instances without requiring a parameter. VFI algorithms do this non-incrementally, i.e. processing all the training instances at once, whereas the COFI and the CFP algorithms are incremental; that is, processes each instance one by one.

The Naive Bayesian Classifier also considers each feature separately both in training and classification as well as feature projection based classification methods. The voting scheme used in the classification phase of the VFI algorithms is also analogical with the probability estimation in Naive Bayesian Classifier. In Naive Bayesian Classifier, each feature participate in the classification by assigning probability values for each class, and the final probability of a class is the product of individual probabilities measured on each feature. On the other hand, in VFI classifiers each feature distributes its vote among classes and the final vote of a class is the sum of all individual votes given by the features.

There are several advantages of feature projection based knowledge representation, which also holds for the VFI classifiers. One of them is that these methods provide faster classification than the nearest neighbor and the decision tree algorithms described in Chapter 2. Second one is that they enable the classifier to simply ignore the missing feature values occurring both in training and classification, where a value should be provided to replace a missing value

in both nearest neighbor and decision tree algorithms [57]. Naive Bayesian Classifier can also ignore the missing values similar to feature projection based techniques by simply excluding that feature from the product of individual probability distributions of each feature. Another one is that since each feature is considered separately, no normalization of values to the same range for all the features is required as in the case of nearest neighbor and instance-based algorithms described in Chapter 2.

We have developed five versions of the VFI algorithms and called them as VFI1, VFI2, VFI3, VFI4, and VFI5. First, I will give some basic definitions. Then, I will explain the training and classification process in the initial version VFI1 [22] and then continue with the modifications on the basic idea of the VFI algorithms towards the other versions in sequence. Then properties of the VFI algorithms according to some important dimensions of machine learning will be given.

4.1 Basic Definitions

All Feature Projection Based (FPB) algorithms (CFP, COFI, k-NNFP, FIL, and VFI) in essence learn, separately for each feature f, a mapping from the set of values that f can take on, V_f, to a set of intervals I_f, that carries the classification information about the domain for that set of values of f. This mapping is represented as:

$$g_f : V_f \rightarrow I_f \tag{4.1}$$

where g_f is a surjection (onto function) in the VFI algorithms.

Definition 1. An **interval** i on a feature f is defined as:

$$i = \langle V, \mathbf{v} \rangle,$$

where V is a set of values for feature f and $\mathbf{v}$ is the vote vector of interval i.

An interval i defines a vote vector $\mathbf{v}$ for a given set of values V on the feature that i is defined. Here a vote vector $\mathbf{v} = \langle v_1, v_2, \ldots, v_c, \ldots, v_k \rangle$ specifies the votes for each class c in the domain where k is the number of classes.

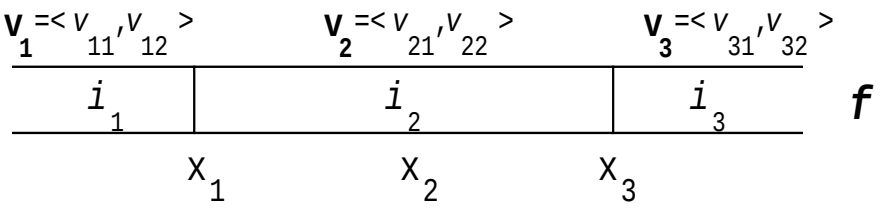

Figure 4.1. An example for three intervals on a feature dimension **f**.

Note that an interval represents several classes by these votes for each class. Therefore, the intervals of the VFI algorithms are multi-class intervals rather than single-class intervals.

Definition 2. A **range interval** is an interval $i = \langle V, \mathbf{v} \rangle$ defined on a linear feature, where the set of values V represents a range of consecutive values.

Figure 4.1 is an example with three range intervals defined on linear feature dimension f where $x_1 \neq x_3$. The interval $i_1 = \langle V_1, \mathbf{v}_1 \rangle$ is defined on a range of values from $-\infty$ to x_1 and represents the classes with a vote vector $\mathbf{v}_1 = \langle v_{11}, v_{12} \rangle$ where v_{11} and v_{12} are the votes of i_1 for the first and second class respectively. The interval $i_2 = \langle V_2, \mathbf{v}_2 \rangle$ is defined on a range of values from x_1 to x_3 and represents the classes with a vote vector $\mathbf{v}_2 = \langle v_{21}, v_{22} \rangle$. The interval $i_3 = \langle V_3, \mathbf{v}_3 \rangle$ is defined on a range of values from x_3 to ∞ and represents the classes with a vote vector $\mathbf{v}_3 = \langle v_{31}, v_{32} \rangle$. The set of intervals in this example is $I_f = \{i_1, i_2, i_3\}$ and the value x_2 is shown in Figure 4.1 to illustrate an example mapping where $g_f(x_2) = i_2$. On the other hand, the mapping of the boundary values such as x_1 and x_3 requires a special treatment and differs in each version of the VFI algorithms.

Definition 3. A **point interval** is an interval $i = \langle V, \mathbf{v} \rangle$, where V is a singleton set $V = \{x\}$.

All the intervals of a nominal feature are point intervals. On the other hand, a point interval might also exist on a linear feature only in some versions of the VFI algorithms. An example nominal feature of a two-class domain along with three point intervals is shown in Figure 4.2. The interval $i_1 = \langle V_1, \mathbf{v}_1 \rangle$ is defined on a singleton set of values $V_1 = \{red\}$ and represents the classes with

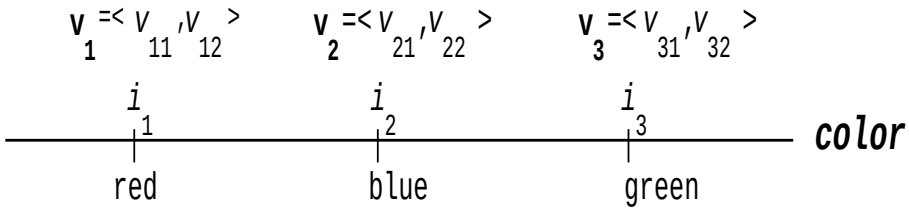

Figure 4.2. An example for three point intervals on feature dimension *color*.

the vote vector $\mathbf{v}_1 = \langle v_{11}, v_{12} \rangle$. Similarly, the interval $i_2 = \langle \{blue\}, \mathbf{v}_2 \rangle$ and $i_3 = \langle \{green\}, \mathbf{v}_3 \rangle$ are defined on a single value of the *color* feature. Suppose that only v_{12} is zero and other votes of all intervals are nonzero, this means that instances of the first class take values *red*, *blue*, and *green* values whereas instances of the second class take *blue* and *green* values for the *color* feature.

The set of values for the *color* feature is $V_{color} = \{red, blue, green\}$ and the set of intervals is $I_{color} = \{i_1, i_2, i_3\}$. A mapping g_{color} is learned by the VFI algorithms and the mappings for all values in V_{color} are:

$$g_{color}(red) = i_1$$
$$g_{color}(blue) = i_2$$
$$g_{color}(green) = i_3$$

where g_{color} is a one-to-one mapping as well as it is onto, thus g_{color} is a bijection. This holds for the g_f of all nominal features.

For point intervals, only a single value is used to represent V of that interval. For range intervals, on the other hand, since all range intervals on a feature dimension are linearly ordered, it suffices to maintain only one value (either lower or upper bound) for the range of values that a range interval represents. In our implementation of the VFI algorithms, we have chosen to keep the lower bound of the range. The upper bound of a range interval is the lower bound of the next higher interval.

4.2 Description of the VFI Algorithms

This section will describe the training and classification process of the VFI algorithms. It will explain how feature intervals on each feature dimension are constructed and how each feature participates in the classification by the voting scheme used.

The only input to the VFI classifiers is a set of preclassified training instances each represented as a vector of feature values plus a label that represents the class of the instance. An instance $\mathbf{x}$ is represented as $\langle x_1, x_2, \ldots, x_d, C_j \rangle$ where $x_1, \ldots, x_d$ are the corresponding feature values of feature $f_1, \ldots, f_d$ and C_i is the associated class label where $1 \leq j \leq k$. Here, k is the total number of classes and d is the number of features in the given domain. Therefore, the dimension of the instance vector is $d + 1$.

4.2.1 The VFI1 Algorithm

The VFI1 classification algorithm [22] is the initial version of VFI algorithms. The next two subsections will describe the training and the classification in the VFI1 algorithm.

4.2.1.1 Training in the VFI1 Algorithm

Since all VFI algorithms are non-incremental, the VFI1 algorithm takes all these training instances and processes them at once. It constructs feature intervals on each feature dimension in the training phase. The training process in the VFI1 classifier is given in Figure 4.3. First, the *end points* for each class c on each feature dimension f are found. End points of a given class c are the lowest and highest values on a linear feature dimension f at which some instances of class c are observed. On the other hand, end points on a nominal feature dimension f of a given class c are all distinct values of f at which some instances of class c are observed. The end points of each feature f is kept in an array $EndPoints[f]$. There are $2k$ end points for each linear feature, where k

```
train(TrainingSet):
begin
    for each feature f
        for each class c
            EndPoints[f] = EndPoints[f] ∪ find_end_points(TrainingSet, f, c);
        sort(EndPoints[f]);

        if f is linear
          each pair of consecutive distinct points in EndPoints[f] form
                                                          a range interval
        else /* f is nominal */
          each distinct point in EndPoints[f] forms a point interval

        for each interval i on feature dimension f
            for each class c
                interval_class_count[f, i, c] = 0
        count_instances(f, TrainingSet);
        for each interval i on feature dimension f
            for each class c
                interval_class_vote[f, i, c] = interval_class_count[f, i, c] / class_count[c]
                normalize interval_class_vote[f, i, c];
                          /* such that Σ_c interval_class_vote[f, i, c] = 1 */
end.

count_instances(f, TrainingSet):
begin
   for each instance e in TrainingSet
       if e_f is known
         i = find_interval(f, e_f)
         e_c = class of instance e

         if i is a point interval */
           if e_f = lower bound of i
             interval_class_count[f, i, e_c] += 1
         else /* i is a range interval */
           if e_f = lower bound of i
             interval_class_count[f, i − 1, e_c] += 0.5
             interval_class_count[f, i, e_c] += 0.5
           else /* e_f falls into i */
             interval_class_count[f, i, e_c] += 1
end.
```

Figure 4.3. Training phase in the VFI1 Algorithm.

```
classify(e): /* e: example to be classified */
begin
    for each class c
        vote[c] = 0

    for each feature f
        for each class c
            feature_vote[f, c] = 0 /* vote of feature f for class c */

        if e_f value is known
            i = find_interval(f, e_f)

            if i is a point interval
              if e_f = lower bound of i
                for each class c
                    feature_vote[f, c] = interval_class_vote[f, i, c]
            else /* i is a range interval */
              if e_f = lower bound of i
                for each class c
                    feature_vote[f, c] = (interval_class_vote[f, i-1, c] + interval_class_vote[f, i, c]) / 2
              else /* inside the interval i */
                for each class c
                    feature_vote[f, c] = interval_class_vote[f, i, c]

            for each class c
                vote[c] = vote[c] + feature_vote[f, c];

    return class c with highest vote[c];
end.
```

Figure 4.4. Classification in the VFI1 Algorithm.

is the number of classes. Then, for linear features the list of end-points on each feature dimension is sorted. If the feature is a linear feature, then each pair of consecutive distinct end points constitutes a range interval. If the feature is a nominal feature, each distinct end point constitutes a point interval.

Then the number of training instances in each interval will be counted and the count of class c instances in interval i of feature f is represented as $interval\_class\_count[f,\ i,\ c]$ in Figure 4.3. These counts for each class c in each interval i on feature dimension f are computed by the *count_instances* procedure. For each training example, the interval i in which the value for feature f of that training example e (e_f) falls is searched. If interval i is a

point interval and e_f is equal to the lower bound (same as the upper bound for a point interval), the count of the class of that instance (e_c) in interval i is incremented by 1. If interval i is a range interval and e_f is equal to the lower bound of i (falls on the lower bound), then the count of class e_c in both interval i and $(i-1)$ are incremented by 0.5. But if e_f falls into interval i instead of falling on the lower bound, the count of class e_c in that interval is incremented by 1 normally. There is no need to consider the upper bounds as another case, because if e_f falls on the upper bound of an interval i, then e_f is the lower bound of interval $i+1$. Since all the intervals for a nominal feature are point intervals, the effect of *count_instances* is to count the number of instances having a particular value for nominal feature f.

To eliminate the effect of differences in the class counts of training instances, the count of instances of class c in interval i of feature f is then normalized by $class\_count[c]$, which is the total number of instances of class c. It is important because 5 instances out of a total of 10 instances is not the same as 5 instances out of a total of 100 instances. The former means that the 50% of that class is in that interval whereas the latter means that only 5% of that class is in that interval. Thus, the relative counts are 0.5 and 0.05 respectively. This relative number of instances in that interval is assigned to $interval\_class\_vote[f,\ i,\ c]$, since this value represents the vote of interval i to class c. To find the final individual vote given to class c by feature f for a future unseen instance with an f value falling into interval i, the $interval\_class\_vote[f,\ i,\ c]$ values are normalized such that the the sum of the votes distributed to several classes is 1. Hence, the vote of interval i on feature f for class c is a real-valued vote less than or equal to 1. This final normalization provides that each feature will have an equal voting power in the classification process independent of its size, since every feature has the equal chance of distributing its votes that sum up to 1. The features might have different voting powers when feature weights are provided from an external source. In that case, the sum of the votes distributed by a feature would be equal to the weight of that feature.

4.2.1.2 Classification in the VFI1 Algorithm

The classification phase of the VFI1 algorithm is given in Figure 4.4. The process starts by initializing the votes of each class to zero. For each feature f, the interval on feature dimension f into which e_f falls is searched, where e_f is the f value of the test example e. If e_f is unknown (missing), that feature does not participate in the voting (gives a vote zero for each class). Hence, the features containing missing values are simply ignored. Ignoring the feature about which nothing is known is a natural and plausible approach.

If e_f is known, first the interval i into which e_f falls is found. If i is a point interval and e_f is equal to the lower bound of that point interval, then for each class c, feature f gives a vote equal to

$$feature\_vote[f, c] \;=\; interval\_class\_vote[f,\; i,\; c] \tag{4.2}$$

where $interval\_class\_vote[f,\; i,\; c]$ is the vote of feature f given to class c. Since point intervals consist of a single value, a point interval i is found such that e_f must fall on that point interval i. This means that e_f must be equal to the lower bound of that point interval (same as its upper bound) in order to say that e_f falls into that interval. If i is a range interval and e_f falls on the lower bound (i.e. e_f is equal to the lower bound) of range interval i, then a vote equal to

$$feature\_vote[f, c] = \frac{interval\_class\_vote[f, i-1, c] + interval\_class\_vote[f, i, c]}{2} \tag{4.3}$$

is given. This is because the instances falling on the lower bound of interval i —which is the upper bound of interval $(i-1)$— were both included in interval $(i-1)$ and i in the counting process of the training phase explained in Section 4.2.1.1. On the other hand, if e_f falls into a range interval i (i.e. e_f is not equal to any lower bound), then for each class c, feature f gives a vote equal to

$$feature\_vote[f, c] \;=\; interval\_class\_vote[f,\; i,\; c] \tag{4.4}$$

as in the case of a point interval. Each feature f collects its votes in a vote vector $\langle feature\_vote[f,\; C_1],\; \ldots\; , feature\_vote[f,\; C_j],\; \ldots, feature\_vote[f,\; C_k]\rangle$, where $feature\_vote[f,\; C_j]$ is the individual vote of feature f for class C_j and

k is the total number of classes. Then these d individual vote vectors, where n is the total number of features, are summed up to get a total vote vector $\langle vote[C_1],\ \ldots\ , vote[C_k]\rangle$. Finally, the class with the highest total vote is predicted to be the class of the test instance.

With this implementation, the VFI1 algorithm is a *categorical classifier*, since it returns a unique class for a test instance [45]. A unique class is predicted for the test instance in order to compare this predicted class with the actual class of the test instance. This enables us to measure the performance of our classifiers according to the most commonly used metric, which is the the percentage of correctly classified test instances over all test instances (see Chapter 5 for more detail). Instead,

$$\frac{vote[C_j]}{\sum_{i=1}^{k} vote[C_i]}$$

can be used as the probability of class C_j which makes the VFI1 algorithm a more general classifier. In that case, the VFI1 algorithm returns a predicted probability distribution over all classes. Although a class is returned as the prediction of the test instance as an output of the VFI1 algorithm , the votes received by each class is also available as an output to the user enabling him/her with the level of confidence in the prediction.

4.2.1.3 An Example

In order to describe the VFI1 algorithm, consider the sample training dataset in Figure 4.5. In this dataset, we have two linear features f_1 and f_2, and there are 3 examples of class A and 4 examples of class B. The intervals with their class counts constructed in the training phase of the VFI1 algorithm are shown in Figure 4.6. There are 5 intervals for each feature. The lower bound of the leftmost intervals is $-\infty$ and the upper bound of the rightmost intervals is ∞. For example, the second interval on feature dimension of f_1 has a class count 2 for for class A and 0 for class B. The count of 2 for class A comes from half count of the class A instance with f_1 value 2, full count of the class A instance with f_1 value 3, and half count of the class A instance with f_1 value 4. The training process continues with computing the interval class votes determined

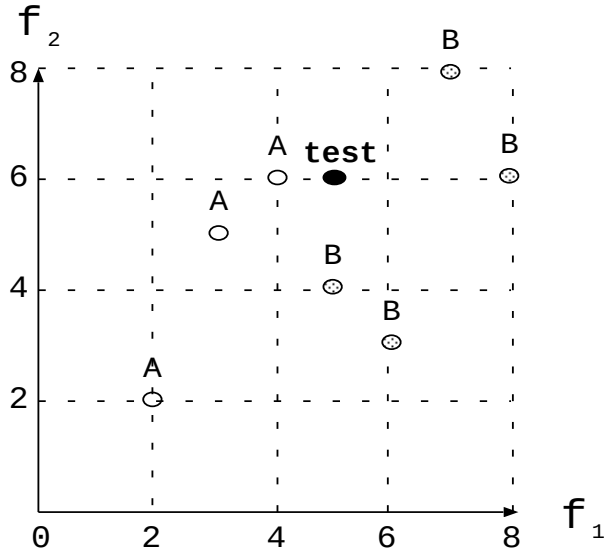

Figure 4.5. A sample training dataset with two features and two classes.

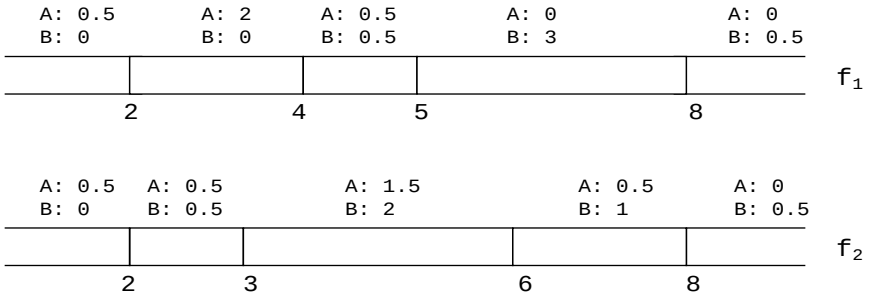

Figure 4.6. The constructed intervals by VFI1 with their class counts for the sample dataset.

by the relative class counts after a normalization. The normalized class votes for the constructed intervals by VFI1 is shown in Figure 4.7. Let us look at one interval to see how the normalized votes are computed from the class counts. The interval i_{24} on feature dimension f_2 has $class\_count[\mathsf{A}] = 0.5$ and $class\_count[\mathsf{B}] = 1$ as shown in Figure 4.6. The class votes are $\frac{0.5}{3} = 0.17$ for class A and $\frac{1}{4} = 0.25$ for class B. Then these votes are normalized to make the sum of votes distributed to classes equal to 1, and the normalized vote for class A is 0.4 and 0.6 for class B.

In order to illustrate the classification phase in the VFI1 algorithm, consider a test example $t = \langle 5, 6, ? \rangle$. On feature f_1 dimension, the $t_1 = 5$

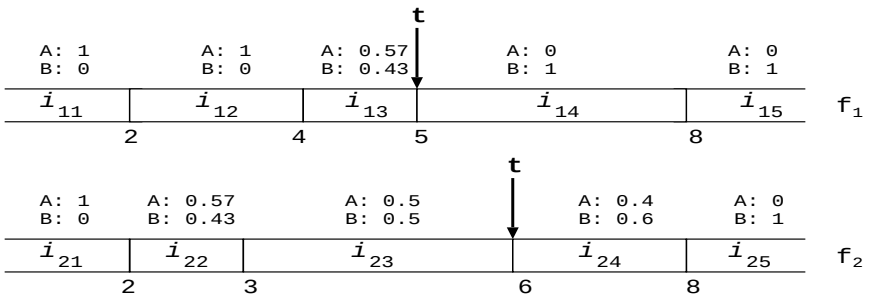

Figure 4.7. The constructed intervals by VFI1 with their class votes for the sample dataset.

falls on the lower bound of the interval i_{14} as shown with an arrow in Figure 4.7. The interval i_{13} has a vote $interval\_class\_vote[f_1, i_{13}, \mathsf{A}] = 0.57$ for class A and $interval\_class\_vote[f_1, i_{13}, \mathsf{B}] = 0.43$ for class B. The interval i_{14} has a vote equal to $interval\_class\_vote[f_1, i_{14}, \mathsf{A}] = 0$ and a vote equal to $interval\_class\_vote[f_1, i_{14}, \mathsf{B}] = 1$. Since the t_1 is on the lower bound of interval i_{14}, the half of votes from both intervals i_{13} and i_{14} determines the individual vote of feature f_1. The votes of feature f_1 are $feature\_vote[f_1, \mathsf{A}] = \frac{0.57+0}{2} = 0.285$ and $feature\_vote[f_1, \mathsf{B}] = \frac{0.43+1}{2} = 0.715$. Thus, the vote vector of f_1 is $\mathbf{v}_1 = \langle 0.285,\ 0.715 \rangle$. If f_1 had been given a chance to make a prediction, it would have predicted class B which has received higher vote than class A. On the feature dimension of f_2, $t_2 = 6$ falls on the lower bound of interval i_{24} as shown with an arrow in Figure 4.7. The interval i_{23} has a vote equal to $interval\_class\_vote[f_1, i_{23}, \mathsf{A}] = 0.5$ and $interval\_class\_vote[f_1, i_{23}, \mathsf{B}] = 0.5$ for class A and class B respectively. The interval i_{24} has a vote equal to $interval\_class\_vote[f_1, i_{24}, \mathsf{A}] = 0.4$ and $interval\_class\_vote[f_1, i_{24}, \mathsf{B}] = 0.6$ for class A and B respectively. Since the t_2 is on the lower bound of interval i_{24}, the average of votes from both intervals i_{23} and i_{24} determines the individual vote of feature f_2. The votes of feature f_2 are $feature\_vote[f_2, \mathsf{A}] = \frac{0.5+0.4}{2} = 0.45$ and $feature\_vote[f_1, \mathsf{B}] = \frac{0.5+0.6}{2} = 0.55$. Thus, the vote vector of f_2 is $\mathbf{v}_2 = \langle 0.45,\ 0.55 \rangle$. If f_2 had also been given the chance to make a prediction, it would have predicted class B but not as much confident as feature f_1. Finally, the individual votes of the two features are summed up correspondingly and the total vote vector is $\mathbf{v} = \langle 0.735, 1.265 \rangle$. VFI1 votes 0.735 for class A and 1.265 for class B, so class B with the highest vote is predicted as the class of the test example. If VFI1 were used to make probabilistic classifications

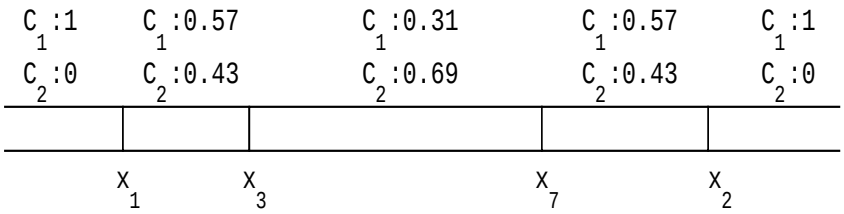

Figure 4.8. The constructed intervals by VFI1 with their class votes for the training dataset in Figure 3.1.

instead of categorical (see Section 4.2.1.2), VFI1 would predict class A with 37% probability and class B with 63% probability.

In order to compare the concept description learned by the VFI1 algorithm with that of the CFP algorithm, the intervals along with their votes constructed from the training dataset in Figure 3.1 by the VFI1 algorithm are shown in Figure 4.8. The intervals constructed by the VFI1 algorithm represent both class C_1 and C_2 with their corresponding votes whereas the intervals constructed by the CFP algorithm represent only a single class. When a new instance falls between x_3 and x_7, it would be predicted as class C_2 in the VFI1 algorithm since that is the range of values in which class C_2 training instances were observed. On the other hand, the CFP algorithm has lost this information and is just aware of some single points on which class C_2 training instances appeared as shown in Figure 3.1.

The single-class intervals constructed by the FIL algorithms using the same dataset was shown in Figure 3.3. The FIL algorithms construct a range interval representing class C_2 between x_5 and x_7 unlike the CFP algorithm. While all the values are mapped to an interval in the VFI algorithms, both the CFP and the FIL algorithms might have many empty range of values between the intervals such as the range of values between the point intervals in Figure 3.3.

Figure 4.9 shows the intervals along with their votes constructed by the VFI1 algorithm from the dataset in Figure 3.4 where the intervals constructed by the COFI algorithm are also shown. Like the CFP and the FIL algorithms, the COFI algorithm also has many empty range of values. For example, a new instance with value equal to 6 would be predicted as class C_2 in the VFI1 algorithm whereas the COFI algorithm would predict nothing for this instance.

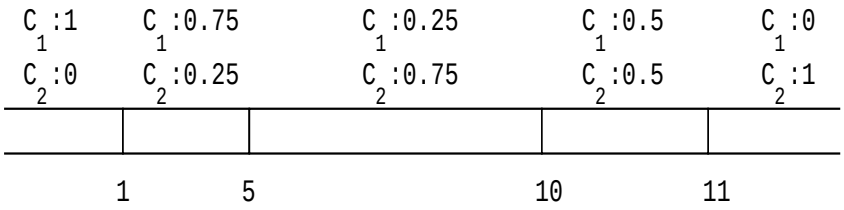

Figure 4.9. The constructed intervals by VFI1 with their class votes for the training dataset in Figure 3.4.

4.2.2 The VFI2 Algorithm

The VFI2 algorithm is the next version after the initial VFI algorithm. The VFI2 algorithm will be explained by only pointing out its differences than the VFI1 algorithm. The only difference is in finding the lower bounds of intervals on linear feature dimensions. In the VFI1 algorithm, the interval lower bounds are distinct end points which are the lowest and highest points of each class on a given feature dimension. This causes lots of instances fall on interval boundaries and we thought that it would be better to fall into one interval instead of being affected by two neighboring intervals. Therefore, to decrease the amount of hits on interval boundaries, the interval lower bounds are determined as the mid points of the end points instead of the end points themselves as in VFI1. This helps a lot especially in visualizing the Dermatology dataset (see Chapter 7). The idea of using the mid points has been also used in C4.5 to find the best split on a linear feature dimension [58].

The training algorithm for the VFI2 algorithm is shown in Figure 4.10. Algorithm is the same as the VFI1 algorithm, except the determination of the lower bounds of the intervals on a linear feature dimension. The classification process is exactly the same as that of the VFI1 algorithm.

The intervals with their class counts constructed from the example training dataset in Figure 4.5 by the VFI2 algorithm is shown in Figure 4.11. The lower bounds of the intervals are the mid points of the lower bounds of the intervals constructed by the VFI1 algorithm shown in Figure 4.6. The training instances falling on the lower bounds are similarly treated as in the VFI1 algorithm, so there might still be half counts but this usually occurs less than it occurs in

```
train(TrainingSet):
begin
    for each feature f
        for each class c
            EndPoints[f] = EndPoints[f] ∪ find_end_points(TrainingSet, f, c);
        sort(EndPoints[f]);

        if f is linear
          each pair of mid points of two consecutive distinct points in EndPoints[f]
                                                                  form a range interval
        else /* f is nominal */
          each distinct point in EndPoints[f] forms a point interval

        for each interval i on feature dimension f
            for each class c
                interval_class_count[f, i, c] = 0
        count_instances(f, TrainingSet);
        for each interval i on feature dimension f
            for each class c
                interval_class_vote[f, i, c] = interval_class_count[f, i, c] / class_count[c]
                normalize interval_class_vote[f, i, c];
                          /* such that Σ_c interval_class_vote[f, i, c] = 1 */
end.
```

Figure 4.10. Training phase in the VFI2 Algorithm.

VFI1. The normalized votes from these class counts of each interval is shown in Figure 4.12.

Let us go through the classification of the same test instance $t = \langle 5, 6, ? \rangle$ classified by VFI1 in Section 4.2.1.3. The intervals into which this test example falls on each feature are indicated in Figure 4.12 with arrows. On feature f_1 dimension, the $t_1 = 5$ falls into interval i_{13}. Remember that t_1 falls on the lower bound of an interval constructed by the VFI1 algorithm (see Figure 4.7). The interval i_{13} has a vote $interval\_class\_vote[f_1, i_{13}, \mathsf{A}] = 0$ for class A and a vote $interval\_class\_vote[f_1, i_{13}, \mathsf{B}] = 1$ for class B. Thus, the vote vector of f_1 is $\mathbf{v}_1 = \langle 0,\ 1 \rangle$. If f_1 had been given the chance to make a prediction, it would have predicted class B with no doubt because B has received all the votes of feature f_1 and class A has received none. On the feature dimension of f_2, $t_2 = 6$

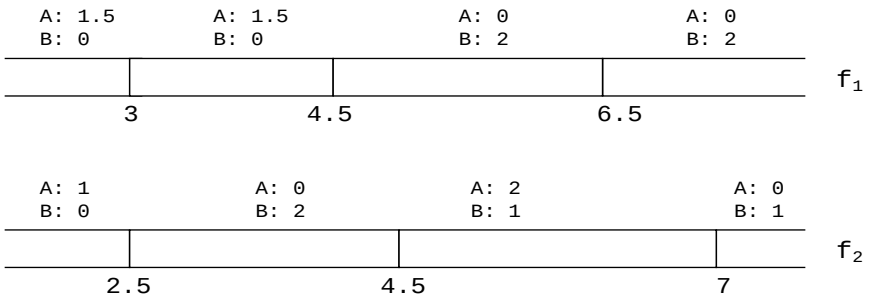

Figure 4.11. The constructed intervals by VFI2 with their class counts for the sample dataset.

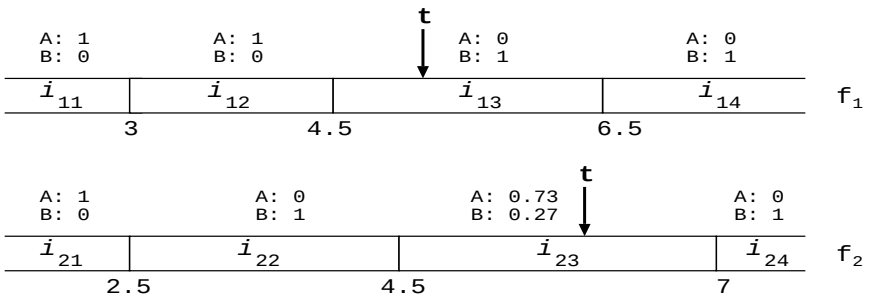

Figure 4.12. The constructed intervals by VFI2 with their class votes for the sample dataset.

falls into interval i_{23} which has $interval\_class\_vote[f_1, i_{23}, \mathsf{A}] = 0.73$ for class A and $interval\_class\_vote[f_1, i_{23}, \mathsf{B}] = 0.27$ for class B. Thus, the vote vector of f_2 is $\mathbf{v}_2 = \langle 0.73,\ 0.27 \rangle$. If f_2 had been given the chance to make a prediction, it would have predicted class A, which has received higher votes than that of class B. The reason for this is that 2 training instances of class A out of a total of 3 are observed in the range [4.5 .. 7] of feature f_2, whereas only 1 training instance of class B out of a total of 4 are in that range. Finally, the individual votes of the two features are summed up respectively and the total vote vector is $\mathbf{v} = \langle 0.73, 1.27 \rangle$. That is, the VFI2 algorithm votes 0.73 for class A and 1.27 for class B, so class B, receiving the highest votes is predicted as the class of the test example t.

4.2.3 The VFI3 Algorithm

The VFI3 algorithm is not something that is developed over the VFI2 algorithm, instead it is again a modification to the VFI1 algorithm in determining

under what conditions the lower bound of an interval i can be included to only one interval instead of both interval i (the right interval) and interval $i-1$ having that lower bound as the upper bound (the left interval). Remember that the intervals are formed by a pair of consecutive distinct end points of each class. The lower bound of an interval is either the lowest or the highest point of a class on that dimension. The lower bounds of a range interval can be classified into three types according to a given class c:

1. The lower bound of the interval is the lowest point of a class c,
2. The lower bound of the interval is the highest point of a class c,
3. Neither of the above two types, that is, another class ($\neq c$) determines the lower bound.

Suppose that during the training a training instance of class C_j falls on the lower bound of a range interval i. If the lower bound is of first type according to class c, the current training instance is counted in the right interval since the lower bound is the start point of observing class c instances. If it is of second type, the current training instance is counted in the left interval since the lower bound is the last point of observing class c instances. If it is of the last type, since class c instances are observed before and after that lower bound, the current training instance is counted half for the right interval and half for the left interval as done in the VFI1 algorithm. Thus, by replacing the *count_instances*(f, i, c) function in Figure 4.3 with *count_instances_vfi3*(f, i, c) shown in Figure 4.13, we get the training algorithm for the VFI3 algorithm. The difference in *count_instances_vfi3*(f, i, c) is that it now counts the instances on the lower bounds taking care of the types of the lower bounds as described just above.

The classification in the VFI3 algorithm is also modified in order to consider three lower bound types for range intervals. The modified classification algorithm is shown in Figure 4.14. First the total votes and individual votes of each feature for each class are initialized to zero as usual. Then, given a test instance, for each feature f, the interval in which the value of the test example for feature f (t_f) falls is sought. If t_f falls on a point interval and

```
count_instances_vfi3(f, TrainingSet):
begin
  for each instance e in TrainingSet
    if e_f is known
      i = find_interval(f, e_f)
      e_c = class of instance e

      if i is a point interval
        if e_f = lower bound of i
          interval_class_count[f, i, e_c] += 1

      else /* i is a range interval */
        if e_f = lower bound of i
          if e_f = lowest point of e_c on f
            interval_class_count[f, i, e_c] += 1
          else if e_f = highest point of e_c on f
            interval_class_count[f, i - 1, e_c] += 1
          else
            interval_class_count[f, i - 1, e_c] += 0.5
            interval_class_count[f, i, e_c] += 0.5
        else /* inside the interval */
          interval_class_count[f, i, e_c] += 1
end.
```

Figure 4.13. The algorithm for counting the training instances in the training phase of the VFI3 classifier.

is equal to the lower bound of that point interval, then feature f gives a vote equal to $interval\_class\_vote[f,\ i,\ c]$ for each class c. If t_f falls into a range interval without falling on its lower bound, then feature f gives a vote equal to $interval\_class\_vote[f,\ i,\ c]$ for each class c. However, if t_f falls on the lower bound of range interval i (i.e. equal to the lower bound of range interval i), then each class c receives a vote according to the type of the lower bound with respect to c. There are three cases:

1. If the lower bound of interval i is equal to the lowest point of class c on feature dimension f, then class c receives its vote from interval i.

2. If the lower bound of interval i is equal to the highest point of class c on feature dimension f, then class c receives its vote from interval $i - 1$.

```
classify(e): /* e: example to be classified */
begin
    for each class c
        vote[c] = 0

    for each feature f
        for each class c
            feature_vote[f, c] = 0 /* vote of feature f for class c */

        if e_f value is known
          i = find_interval(f, e_f)

          if i is a point interval
            if e_f = lower bound of i
              for each class c
                  feature_vote[f, c] = interval_class_vote[f, i, c]

          else /* i is a range interval */
            if e_f = lower bound of i
              for each class c
                  if e_f = lowest point of c on f
                    feature_vote[f, c] = interval_class_vote[f, i, c]
                  else if e_f = highest point of c on f
                    feature_vote[f, c] = interval_class_vote[f, i - 1, c]
                  else
                    feature_vote[f, c] = (interval_class_vote[f, i-1, c] + interval_class_vote[f, i, c]) / 2

            else /* inside the interval i */
              for each class c
                  feature_vote[f, c] = interval_class_vote[f, i, c]

          for each class c
              normalize feature_vote[f, c] /* s.t. sum_c feature_vote[f, c] = 1 */
              vote[c] = vote[c] + feature_vote[f, c];

    return class c with highest vote[c];
end.
```

Figure 4.14. Classification in the VFI3 Algorithm.

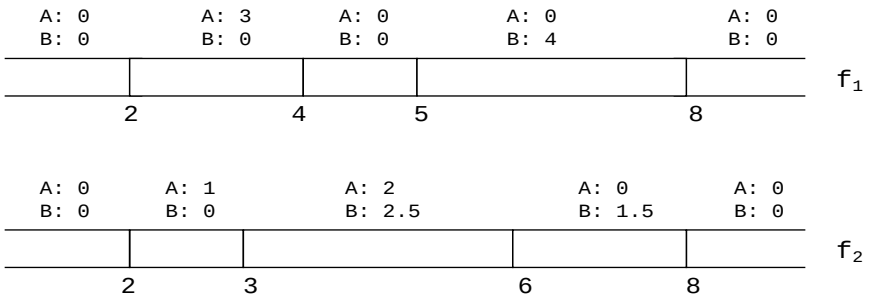

Figure 4.15. The constructed intervals by VFI3 with their class counts for the sample dataset.

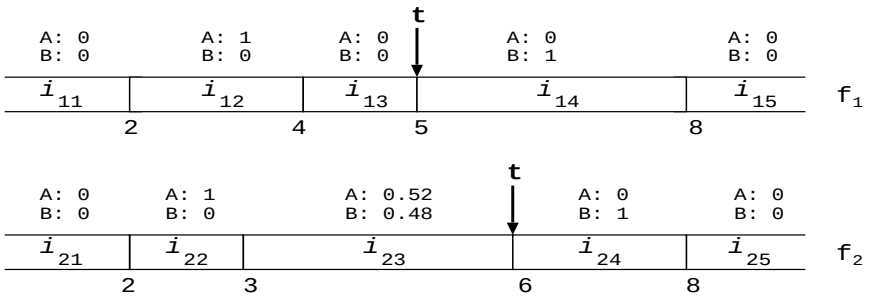

Figure 4.16. The constructed intervals by VFI3 with their class votes for the sample dataset.

3. If the lower bound of interval i is equal to neither the lowest nor the highest point of class c on feature dimension f, then class c receives its vote as the average of the votes from intervals $i-1$ and i.

Since classes might take their votes from different intervals or even as the average of two intervals, the sum of votes for each class is no more equal to 1 as normalized in training. Therefore, the votes of each feature are once again normalized before combining to compute the total vote. This final normalization is not required in the VFI1 and the VFI2 algorithms because if a vote is taken from an interval or from two neighboring intervals, that is the same for all the classes and does not change from class to class. However, in VFI3 a class might take its vote from interval i, whereas another class might take its vote from both interval i and $i-1$. Examples for this case will be shown. Finally, the class with the highest total vote is returned as the prediction.

To illustrate the training and classification in VFI3, the intervals with their class counts constructed from the example training dataset in Figure 4.5 by

VFI3 are shown in Figure 4.15. The lower bounds of all intervals are the same as the lower bounds of the intervals constructed by VFI1 shown in Figure 4.6, however the class counts of the intervals are changed. For example, the training instance of class A with f_1 value 4 is not now counted half for the second and half for the third intervals of feature f_1, instead it is fully included in the second interval because the lower bound of the third interval is of the second type i.e. it is the highest point of class A on feature f_1.

Let us go through the classification of the same test instance $t = \langle 5, 6, ? \rangle$ classified by VFI1 in Section 4.2.1.3. The intervals into which this test example falls on each feature is shown in Figure 4.16 with arrows. On feature f_1 dimension, the $t_1 = 5$ falls on the lower bound of interval i_{14} as shown with an arrow in Figure 4.16. Since this lower bound (5) is the lowest point of class B on feature dimension f_1, the vote for class B will be taken from interval $i14$. Thus, the vote for class B is $feature\_vote[f_1, \mathsf{B}] = 1$. On the other hand, since this lower bound is neither the lowest nor the highest point of class A on f_1, the vote for class A is the average of the votes of intervals i_{13} and $i14$. Since both intervals i_{13} and i_{14} have a vote 0 for class A, their vote is $feature\_vote[f_1, \mathsf{A}] = 0$. A final normalization would change nothing in the distribution of votes and class A receives a vote 0 and class B receives a vote 1 from f_1. On the feature dimension f_2, $t_2 = 6$ falls on the lower bound of interval $i24$. Since this lower bound (6) is the highest point of class A on feature f_2, the vote for class A will be taken from interval i_{24}. Therefore, the vote of feature f_2 for class A is $feature\_vote[f_2, \mathsf{A}] = 0.52$. However, since this lower bound is neither the lowest nor the highest point of class B on f_2, the vote for class B is the average of the votes of intervals $i23$ and $i24$. Then the vote for class B is $feature\_vote[f_2, \mathsf{B}] = (0.48 + 1)/2 = 0.74$. Note that the feature votes given to class A and B do not sum up to 1 and if we leave them as they are, this feature will have a higher voting power with no reason. Therefore, in the classification of the VFI3 algorithm these votes are normalized and a vote equal to $feature\_vote[f_2, \mathsf{A}] = 0.41$ for class A and a vote $feature\_vote[f_2, \mathsf{B}] = 0.59$ are given. Finally, the individual votes of the two features are summed up correspondingly and total vote vector is $\mathbf{v} = \langle 0.41, 1.59 \rangle$. The VFI3 algorithm votes 0.41 for class A and 1.59 for class B, so class B with the highest vote is predicted as the class of the test example.

Figure 4.17. The projection of a sample dataset with two classes on linear feature dimension f_1.

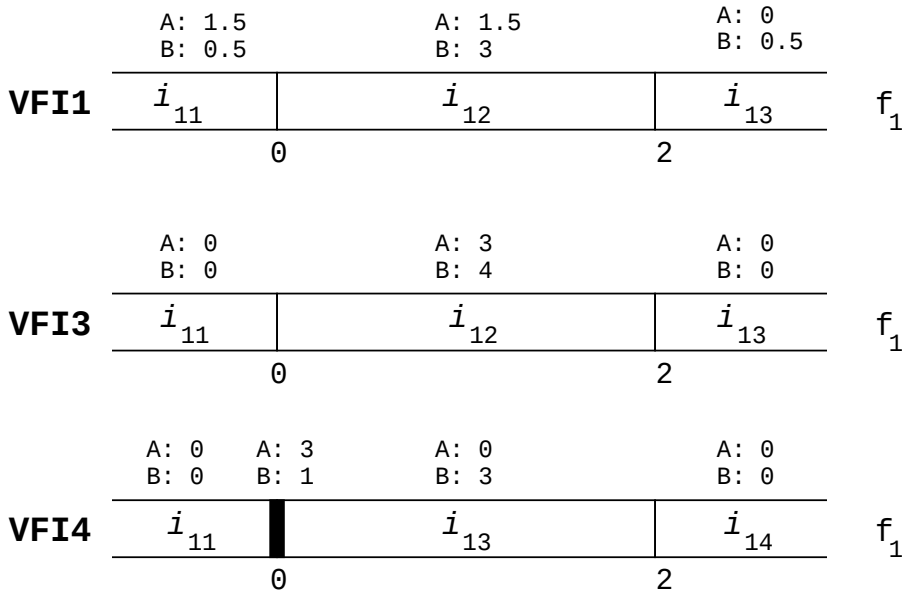

Figure 4.18. The constructed intervals by VFI1, VFI3, VFI4 with their class counts for the second sample dataset.

4.2.4 The VFI4 Algorithm

The VFI4 algorithm is the version developed over the VFI3 algorithm when we realized that in real-world datasets there are classes instances of which always take the same value for a feature. That is, the lowest and highest points of such a class are the same on that feature dimension. But in all previous versions of the VFI algorithms, we do not represent this knowledge. On the other hand, it is not a loss of knowledge in the case of a nominal feature because all the instances of that class are counted on that point interval constructed by the lowest and highest points of that class. But when it occurs on a linear feature dimension, an interval starting from that lowest (equal to highest) point and continuing up to a distinct point is constructed. This will result in a concept description which represents that class in the range of values in which it never

appeared.

To illustrate the problem, suppose that we have a sample dataset with the projection shown in Figure 4.17 on linear feature dimension f_1. There are two classes, class A and B in this domain. All of the three training instances of class A are observed on value 0 for feature f_1. There are four training instances of class B, which are observed on values 0, 1, and 2 for feature f_1. The lowest point of class A is 0 on f_1 dimension, which is also the highest point of class A. The lowest point of class B is 0 and the highest point of class B is 2. Therefore, we have two distinct end points —0 and 2— from which the range intervals will be constructed in both the VFI1 and the VFI3 algorithms. The intervals with their class counts constructed by the VFI1 and the VFI3 algorithms are shown in Figure 4.18. The instances of class A having f_1 value 0 (falling on the lower bound of interval i_{12}) are counted half for interval i_{11} and half for interval i_{12} in the VFI1 algorithm. On the other hand, since the VFI3 algorithm tries to count the instance on the lower bounds according to the types of the lower bounds, all the instances of class A are counted for interval i_{12}. When a new instance with value 1 for feature f_1 is to be classified, class A will get a nonzero vote from feature f_1 both in the VFI1 and the VFI3 algorithms. However, class A instances never had a value different than 0 for feature f_1 and the inductive result from this should be that a class A instance can not have a value other than 0 for feature f_1. One might say that 0 is also the highest point of class A, so the VFI3 algorithm might count those instances in interval i_{11} as well. Both might have done, but the VFI3 algorithm and all the other versions of the VFI algorithms do not realize that a lower bound is both the highest and the lowest at the same time, and that's why we came up with the VFI4 algorithm, which takes care of this special situation that might occur in real-world datasets. In fact, a feature always getting the same value for a class is very informative and should be discovered by a classifier.

The VFI4 algorithm constructs a *point interval* from the end point 0, which is both the lowest and highest point of class A as shown with a filled narrow rectangle at point 0 in Figure 4.18. This point interval is exactly the same as the point intervals constructed for nominal features, that is, the lower and upper bound of this interval is both 0 and instances having 0 value for feature

```
train(TrainingSet):
begin
    for each feature f
        for each class c
            EndPoints[f] = EndPoints[f] ∪ find_end_points(TrainingSet, f, c);
        sort(EndPoints[f]);

        if f is linear
          for each end point p in EndPoints[f]
              if an end point p = both lowest and highest point of a class
                 form a point interval from end point p
                 form a range interval between p and the next endpoint≠ p
              else
                  form a range interval between p and the next endpoint≠ p
        else /* f is nominal */
           each distinct point in EndPoints[f] forms a point interval

        for each interval i on feature dimension f
            for each class c
                interval_class_count[f, i, c] = 0

        count_instances_vfi3(f, TrainingSet);

        for each interval i on feature dimension f
            for each class c
                interval_class_vote[f, i, c] = interval_class_count[f, i, c] / class_count[c]
                normalize interval_class_vote[f, i, c];
                          /* such that ∑_c interval_class_vote[f, i, c] = 1 */
end.
```

Figure 4.19. Training phase in the VFI4 Algorithm.

f_1 are counted in this interval. When a new instance with value 0 for feature f_1 is to be classified, class A will get a vote 0.8 for class A and 0.2 for class B from feature f_1. On the other hand, the same test instance will get a vote 0.5 for class A and 0.5 for class B in the VFI3 algorithm. Although the training instances carry the information that all the instances of class A occurred on value 0 of feature f_1, the VFI3 algorithm somehow loses this and votes equally for both classes. The VFI4 algorithm is designed to overcome this loss of knowledge in the VFI3 algorithm.

The training process of the VFI4 algorithm is the same as that of the VFI3 algorithm except for the special situation illustrated by an example above. The modified training for the VFI4 algorithm is shown in Figure 4.19. When an end point p is both the lowest and highest points of a class, a point interval with lower bound and upper bound equal to p is constructed. Then a range interval between p and the next end point different than p is constructed. This end point p becomes the upper bound for the left neighboring range interval and the lower bound for the right neighboring range interval. To exclude the training instances falling on p from both the right and the left range intervals, the training instances falling on p are counted only for the point interval. While counting the instances in training, if there exists a point interval for the value being searched for, that point interval is returned by $find\_interval$ function used in $count\_instances\_vfi3$ procedure. Therefore, such an end point p is excluded both from the left interval and from the right interval. Since the same counting procedure used by the VFI3 algorithm is used in the VFI4 algorithm, the other lower bounds that are not point intervals have the same treatment as they had in the VFI3 algorithm. As a summary, the VFI4 algorithm checks for equal lowest and highest points to construct a point interval from such an end point and excludes that point from the neighboring range intervals.

In the classification of a new instance, if the value of the instance on that feature dimension is equal to a lower bound of a point interval, then the individual votes of that feature are simply taken from that point interval. Although that value is also the lower bound of the next range interval, the votes are taken from only the point interval if there is a point interval with that value. This is again handled by the $find\_interval$ function, which returns the point

interval with lower bound equal to that value if a point interval with that lower bound exists. When a test instance falls on the lower bound of a range interval or inside an interval, the classification process is the same as that of the VFI3 algorithm.

The intervals constructed by the VFI4 algorithm from the sample training dataset in Figure 4.5 are exactly the same as those constructed by the VFI3 algorithm, since there are not classes with equal lowest and highest points on any feature dimension. The VFI4 algorithm differs from the VFI3 algorithm only when there are end points on any feature dimension which are both lowest and highest points for the same class. Such situations might be observed in real-world datasets and for example it occurs in the Dermatology dataset.

4.2.5 The VFI5 Algorithm

The VFI5 algorithm is the final version of VFI algorithms that generalizes the construction of point intervals to all end points. The VFI5 algorithm constructs a point interval from each distinct end point and a range interval between a pair of distinct end points excluding the end points. The training algorithm for VFI5 is shown in Figure 4.20. The intervals along with their class counts constructed from the sample training dataset in Figure 4.5 are shown in Figure 4.21. The lower bounds of all intervals are now point intervals and there are range intervals between those lower bounds exclude the lower bounds. For example, the training instance of class A with f_1 value 4 is counted for point interval i_{14} on f_1 dimension with lower and upper bound equal to 4, and the instance of class A with f_1 value 3 is counted for range interval i_{13} on feature f_1 with lower bound 2.

The classification process is the same as that of the VFI1 algorithm. The VFI5 algorithm finds the point intervals on linear feature dimensions by the $find\_interval$ function. The point intervals on linear features are same as those of the nominal features. Since the lower bound of a range interval is also the lower bound of a point interval, $find\_interval$ function returns the point interval when the value of a test instance for a feature is equal to the lower bound of an interval. Therefore, there is no decision required to take about

```
train(TrainingSet):
begin
    for each feature f
        for each class c
            EndPoints[f] = EndPoints[f] ∪ find_end_points(TrainingSet, f, c);
        sort(EndPoints[f]);

        if f is linear
          for each end point p in EndPoints[f]
              form a point interval from end point p
              form a range interval between p and the next endpoint≠ p
        else /* f is nominal */
           each distinct point in EndPoints[f] forms a point interval

        for each interval i on feature dimension f
            for each class c
                interval_class_count[f, i, c] = 0
        count_instances(f, TrainingSet);
        for each interval i on feature dimension f
            for each class c
                interval_class_vote[f, i, c] = interval_class_count[f, i, c] / class_count[c]
                normalize interval_class_vote[f, i, c];
                          /* such that ∑_c interval_class_vote[f, i, c] = 1 */
end.
```

Figure 4.20. Training phase in the VFI5 Algorithm.

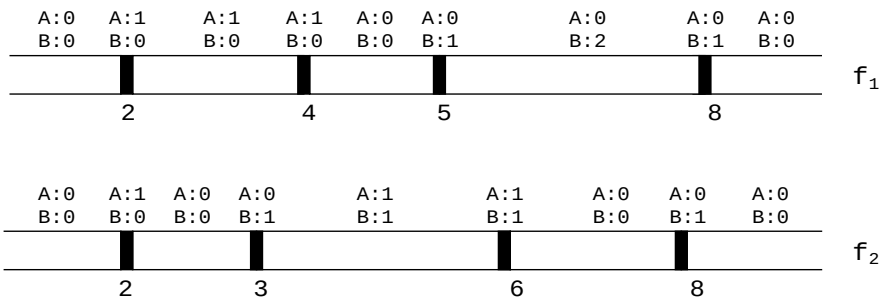

Figure 4.21. The constructed intervals by VFI5 with their class counts for the sample dataset.

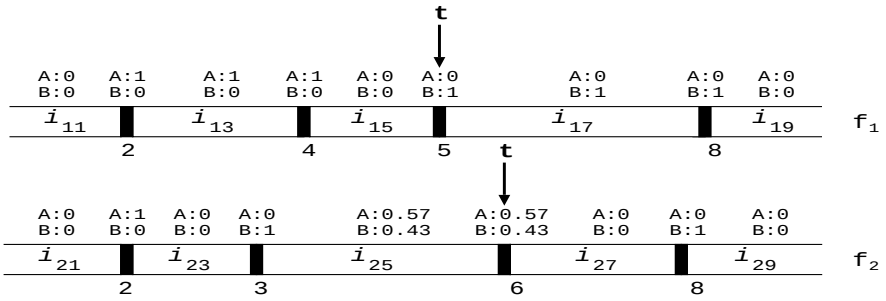

Figure 4.22. The constructed intervals by VFI5 with their class votes for the sample dataset.

the lower bounds as done in all other versions of VFI algorithms.

To illustrate the classification of the VFI5 algorithm on an example, let us classify the same test example $t = \langle 5, 6, ? \rangle$ also classified by other VFI classifiers. This test example falls on point interval i_{16} with lower bound 5 on feature dimension f_1 and on point interval i_{26} with lower bound 6 on feature dimension f_2 shown with arrows in Figure 4.22. Since there are point intervals on which both $t_1 = 5$ and $t_2 = 6$ fall, the individual votes of features are taken from the corresponding point intervals.

The point interval i_{16} of feature f_1 on which $t_1 = 5$ falls votes equal to $interval\_class\_vote[f_1, i_{16}, \mathsf{A}] = 0$ and $interval\_class\_vote[f_1, i_{16}, \mathsf{B}] = 1$ for class A and class B respectively. Thus, the individual vote vector of f_1 is $\mathbf{v}_1 = \langle 0, 1 \rangle$. If f_1 had been given the chance to make a prediction alone, it would have predicted class B with certainty because B has received all the vote of feature f_1 and class A has received none. On the feature dimension of f_2, the point interval i_{26} on which $t_2 = 6$ falls has a vote equal to $interval\_class\_vote[f_1, i_{26}, \mathsf{A}] = 0.57$ for class A and a vote equal to $interval\_class\_vote[f_1, i_{26}, \mathsf{B}] = 0.43$ for class B. Thus, the individual vote vector of f_2 is $\mathbf{v}_2 = \langle 0.57, 0.43 \rangle$. If f_2 had been given the chance to make a prediction, it would have predicted class A. Finally, the individual votes of the two features are summed up correspondingly and total vote vector is $\mathbf{v} = \langle 0.57, 1.43 \rangle$. The VFI5 algorithm votes 0.57 for class A and 1.43 for class B, so class B with the highest vote is predicted as the class of the test example.

4.3 Characteristics of VFI Algorithms

In this section, the general properties of learning methods are presented in order to characterize the VFI algorithms.

4.3.1 Knowledge Representation

Knowledge representation is one of the most important dimensions in classifying machine learning techniques. Many machine learning systems acquire knowledge in the form of *rules*. Another way to represent what is learned is with decision trees as described in Chapter 2. Naive Bayesian classifier represents the learned concept with a set of conditional probabilities. On the other hand, knowledge representation in exemplar-based learning models is sets of representative instances [1, 2, 5] or hyperrectangles which represent generalizations [62, 63].

In Chapter 3, we presented a new knowledge representation scheme based on feature projections. Generalization and specialization are made on the basis of feature projections. This allows faster classification of test instances by preventing the similarity computation to each training instance because feature projections can be sorted for continuous valued features. One shortcoming of this representation is that descriptions involving a conjunction between two or more features cannot be represented. However, the prior research has shown that this knowledge representation is quite powerful in the classification of real-world tasks and does not cause any significant drop on the accuracy [32, 73, 8, 7]. All algorithms described in Chapter 3 represent the concept in some generalized form of feature projections of the training instances. The CFP algorithm [32] generalizes the projections of training instances in the form of disjoint feature intervals (single-class) on each feature. The FIL algorithms [7] also represent the concept with disjoint feature intervals on each feature. The k-NNFP algorithm [8] uses feature projections of instances just as they are on each feature dimension without any generalization, it only sorts the projected values on each feature dimension in the training phase. The COFI algorithm [73] generalizes the projections of instances of each class separately

and produces a set of overlapping feature intervals (multi-class) on each feature.

The VFI algorithms also acquire concept descriptions by using feature projection based knowledge representation. Learned concept descriptions are in the form of multi-class intervals. These intervals are able to represent more than one class as in the case of overlapping feature intervals of COFI, but the procedure to construct the intervals in the VFI algorithms is different than the COFI algorithm. The number of intervals on a feature dimension in the VFI algorithms does not depend on the number of training examples as it does in other feature projection based learning algorithms. Instead, it depends on the number of classes in the domain for linear features and on the number of distinct values for nominal features.

4.3.2 Supervised Inductive Learning

We have defined supervised inductive learning (concept learning) in Chapter 1 as learning generalized descriptions from examples supplied by a teacher or an environment. From a set of training instances described with a set of feature values and labeled correctly with a class label among mutually disjoint classes, the supervised inductive learning system learns a concept description which will enable the system correctly classify new instances. VFI algorithms are supervised inductive learning algorithms that take a set of preclassified training instances provided by a teacher as input and make generalizations on the feature projections of these instances to construct the concept description in the form of feature intervals.

4.3.3 Non-incremental (Batch) Learning

Inductive learning can be performed in two alternative ways: *incremental* or *non-incremental* (*batch*) [55]. An incremental learning system processes each instance one by one and aims at improving its internal model with each new instance at each step. Incremental learning is the way humans learn, thus researchers who explore the incremental approach are typically concerned with

developing plausible models of human learning. The inevitable deficiency of this approach is that it is sensitive to the presentation order of the training examples. On the other hand, non-incremental learning systems construct concept descriptions after seeing all training instances to maximize the performance of the learning system. But a non-incremental learning system might also be sensitive to the presentation order of the instances.

Incremental variations of non-incremental algorithms can usually be created and many incremental learning methods also have non-incremental counterparts. For example, IB1 is an incremental variation for the Nearest Neighbor algorithm and FIL algorithms are somewhat non-incremental variations of the CFP algorithm with slight differences.

VFI algorithms are non-incremental, that is, all the training instances are presented to the VFI algorithms before training. The construction of intervals is unique for that training set, that is, they are independent of presentation order of training instances. However, the concept description learned by some learning algorithms might not be unique and change with the order of presentation such as the CFP algorithm (see Chapter 3 for an example).

4.3.4 Domain Independence in Learning

In some learning methods, such as Explanation-Based Generalization (EBG), considerable amount of domain specific knowledge is required to construct explanations [19]. In EBG, some domain specific knowledge is applied to formulate valid generalizations from a single training example. The characteristic common to these methods is their ability to explain why the training instance is a member of the concept being learned.

An advantage of domain independence is that systems can be adapted to new domains quickly without any extra domain knowledge. The CFP and COFI algorithms use domain specific parameters. These parameters in the CFP algorithm are $\triangle$ (feature weight-adjustment rate) and D_f (generalization distances of features). In the COFI algorithm, the only domain dependent parameter is g (generalization ratio). The k-NNFP algorithm and the FIL

algorithms do not use any domain specific parameters. Similarly, the VFI algorithms also do not require any domain specific parameters, thus can be quickly adapted to any domain from which a set of training instances are drawn and presented to the VFI algorithms as input. On the other hand, feature weights are domain specific knowledge and a feature weight learning method can be adapted to all feature projection based learning algorithms.

4.3.5 Multi-concept Learning

Many early concept learning algorithms have been developed for exactly one concept and the instances are either instances belonging to the concept (positive) or not belonging to the concept (negative). Later, many learning algorithms have been developed that induce multi-concept descriptions from examples. Multi-concept learning is more general than single-concept learning, since the descriptions for any number of concepts can be learned. The VFI algorithms as well as all the other algorithms mentioned in this thesis have been designed for learning multi-concept descriptions. The focus of most classification algorithms is multi-concept learning of disjoint concepts, that is, instances do not belong to more than one class. But in some other multi-concept learning tasks, instances may belong to more than one class, that is classifications of instances are possibly overlapping. VFI algorithms are capable of learning multi-concept descriptions instead of only single-concept descriptions.

4.3.6 Properties of Feature Values

The representation of the input training instances to a classification system is described at the beginning of Chapter 2. The instances are described with a vector of feature values and a class (concept) label which they belong to. The features might either be nominal (discrete, categorical) or linear (continuous). The VFI algorithms can handle both linear and nominal features. Linear features may take on values from $-\infty$ to ∞ and all possible values are linearly ordered. Nominal features take on discrete feature values, for example, color of

```
; Information about the Hungarian dataset
Features l n n l l n n l n l n n n
Classes  0 1
```

Figure 4.23. An example for the information provided to the FIL algorithms.

an object is a nominal feature, or binary values such as answers to yes/no questions are also nominal feature values. The only difference in handling linear features and nominal features is that only point intervals are constructed for nominal features whereas mostly range intervals —except some point intervals constructed in the VFI4 and the VFI5 algorithms— are constructed for linear features.

4.3.7 Handling Missing (Unknown) Feature Values

One of the most important advantages of the VFI algorithms is the natural handling of missing feature values. There is no need to fill in missing values with some arbitrary value in the VFI algorithms. This affects neither the construction of the feature intervals in training nor the voting mechanism used in the classification process. In addition, this is a natural approach because in real life if nothing is known about a feature, it can be ignored rather than assigning an average or expected value.

4.4 Implementation and User Interface

The VFI algorithms have been written in C language and implemented in Unix environment. The input to the VFI algorithms is a file of training instances, a file of test instances, and an information file. Figure 4.23 shows the information file given to the VFI algorithms as input for the *Hungarian* dataset. A line starting with a ";" indicates a comment line, a line starting with "Features" tells the number and types —either linear (l) or nominal (n)— of features, and a line starting with "Classes" tells all the possible class names that may

appear in the whole dataset. The information file can also have an additional line starting with "Weights", in which the weights of the features are given. The file of training instances has a ".train" extension and the instances in this file is used to construct concept descriptions. The file of test instances has a ".test" extension and the instances in this file are given as unseen instances to the VFI classifiers and their actual class label is compared with the class label predicted by the VFI classifiers.

The VFI algorithms can be run from the command line as well as using the user interface that we have designed and implemented by using the Motif user-interface toolkit. The user can select a dataset from the "Open" menu item. Then, with an initial training ratio training and testing datasets are formed from the dataset file having an extension ".data". The user can also change the default training ratio by selecting the menu item "Train Ratio". The lowest and highest points of each class on each feature dimension are displayed on each feature dimension assigning a different color to each class label on the screen. Usage of colors provides users to better visualize the predictions made by individual features. User can proceed one by one on the test instances by performing classification task with the "NEXT" button. Also, the user can choose to classify all test instances at once with the "ALL" button. It is also possible to see the previous test instances and their classifications with the "PREVIOUS" button. On each feature dimension, the point where each corresponding feature value of the current test instance falls is shown. The individual votes of each feature and total votes given to each class along with the final prediction are shown for each test instance. Classification accuracy and no of correct classifications after classifying each test instance are updated. The constructed intervals can be saved into a text file from the menu with the corresponding lower bounds and class counts for each class. In order to illustrate how our user interface looks like, two example on the Dermatology and Arrhythmia datasets are shown in Figures 4.24 and 4.25 respectively.

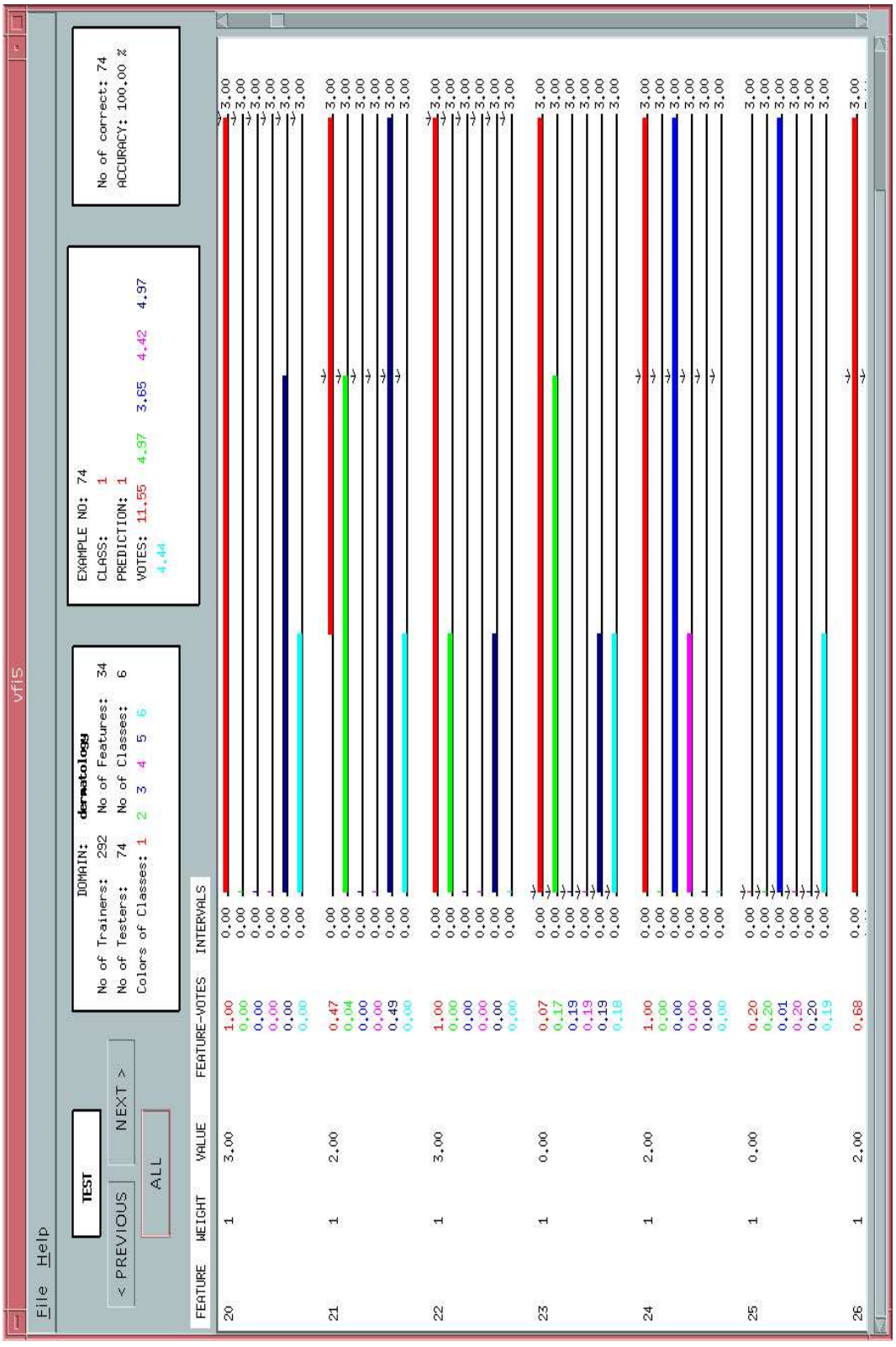

Figure 4.24. The visualization of the feature intervals constructed by the VFI algorithms for the Dermatology dataset by our user interface.

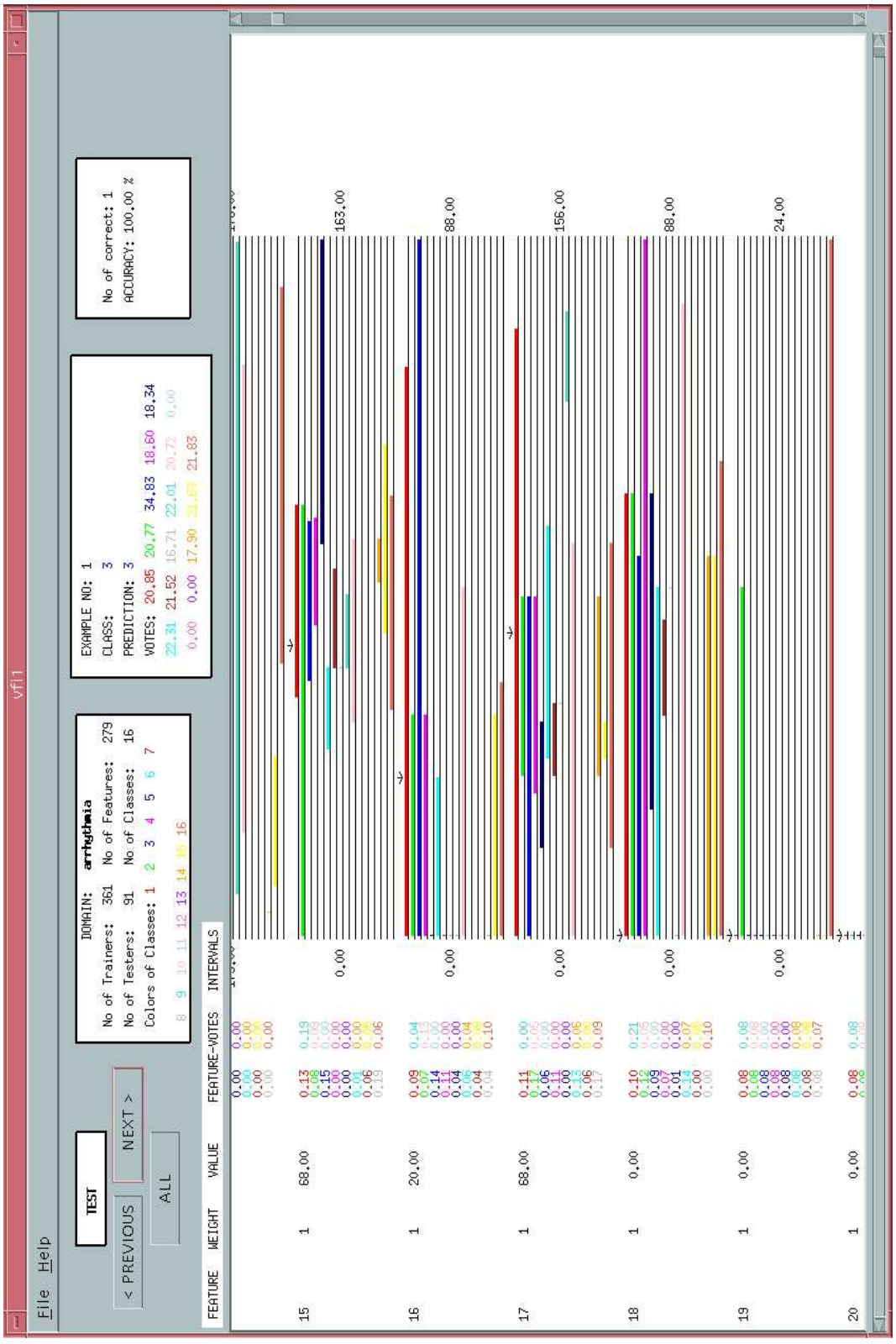

Figure 4.25. The visualization of the feature intervals constructed by the VFI algorithms for the Arrhythmia dataset by our user interface.

4.5 Summary

This chapter introduced several versions of the VFI classification algorithms. VFI algorithms use the feature projection based knowledge representation scheme and a voting scheme in classification similar to those used in the CFP, COFI, k-NNFP, and FIL algorithms described in Chapter 3. Learning in the VFI algorithms is achieved by constructing feature intervals on each feature dimension and classification is performed by a voting scheme. VFI algorithms consider each feature separately as in the case of Naive Bayesian Classifier as well as all other feature projection based methods.

Since each feature is processed separately, the missing feature values that may appear both in the training and test instances are simply ignored. In other classification algorithms, such as decision tree inductive learning algorithms, the missing values cause problems [57]. This problem has been overcome by simply omitting the feature with the missing value in the VFI algorithms. This separate consideration of features enable fast training and classification times, which will be analyzed in the following chapter. Another advantage of the VFI algorithms is that they can make a general classification returning a probability distribution over all classes instead of a *categorical classification* [45]. Also note that the VFI algorithms as well as other feature projection based methods in particular, are applicable to concepts where each feature can be used in the classification of the concept independently. One might think that this requirement may limit the applicability of the VFI algorithms, since in some domains the features might be dependent on each other. Holte has pointed out that the most datasets in the UCI repository are such that, for classification, their features can be considered independently of each other [36]. Also Kononenko claimed that in the data used by human experts there are no strong dependencies between features because features are properly isolated and defined [44].

The versions of the VFI algorithms described in this chapter assume that all features are equally relevant and thus should have equal voting power in classification. But this might not be the case in real-world datasets, therefore in Chapter 6, I will explain how we integrated a feature weight learning method

which assigns the optimum voting power to the features in order to improve the performance of the VFI classifiers.

Chapter 5

Evaluation of the VFI Algorithms

In this chapter, both complexity analyses and empirical evaluations of the VFI algorithms are given. First, training and classification time complexities are computed. Next, the empirical evaluations are presented on some real-world datasets for comparison with several other classification algorithms described in this thesis. Later, the experiments on artificially generated datasets are discussed. Experiments on artificially generated datasets are designed to determine the behavior of the VFI algorithms on irrelevant features, noisy instances and missing feature values.

5.1 Complexity Analysis

In this section, the VFI algorithms are analyzed in terms of space and time complexities. Time complexity analyses are presented for training process and classification of single test instance. In this section, m represents the number of training instances, d the number of features, and k the number of classes.

5.1.1 Space Complexity Analysis

The VFI1 algorithm [22] represents a concept description by feature intervals on each feature dimension. Each linear feature dimension has at most $2k + 1$ intervals. The maximum number of intervals that a linear feature can have is $2k+1$ and occurs when all end points are distinct. If all end points are distinct, there are $2k$ end points because each class has one lowest and one highest point, and $2k$ end points makes up $2k+1$ intervals. Each interval requires $k+1$ memory units, one for the lower bound of the interval and k for the votes of each class. So each linear feature dimension requires $(2k + 1) \cdot (k + 1)$ space, and the total space requirement of the VFI1 algorithm is $d \cdot (2k+1) \cdot (k+1)$ which is $O(d \cdot k^2)$. The other versions of VFI have different number of intervals than VFI1 does as shown in Table 5.1. VFI2 has at most $2k$ intervals because there are $2k-1$ mid points of $2k$ end points, therefore VFI2 requires $d \cdot (2k) \cdot (k + 1)$ memory units. VFI3 has at most $2k + 1$ intervals as in VFI1, since VFI3 is exactly the same in determining the boundaries of intervals. On the other hand, VFI4 might have point intervals in addition to the range intervals constructed as in VFI3. However, when a point interval is constructed from an end point this means that the lowest and highest points of a class are equal. If there exists a point interval on a linear feature dimension, the maximum number of distinct end points decreases by 1 while the number of intervals increases by 1. If the lowest and highest points of each class are equal which will cause k point intervals (maximum number of point intervals on a linear feature) and $k + 1$ range intervals to be constructed. Hence, the total number of intervals constructed by VFI4 in the extreme case is $2k + 1$. Since the maximum number of intervals in both VFI3 and VFI4 is same as that of VFI1, the memory required is $d \cdot (2k + 1) \cdot (k + 1)$. Lastly, VFI5 has at most $4k + 1$ intervals because it keeps an extra point interval for each $2k$ end points in addition to the range intervals between these end points. VFI5 requires $d \cdot (4k + 1) \cdot (k + 1)$ memory units. Nevertheless, the asymptotic space requirement of all is $O(d \cdot k^2)$.

The above complexity analysis assumed that all features are linear, but there might be nominal features in the domain. Since point intervals are constructed from all distinct values of a nominal feature, the number of intervals is equal to the number of distinct values that nominal feature can take.

Table 5.1. The maximum number of intervals on a linear feature dimension for all VFI classifiers.

Classifier	Maximum Number of Intervals
VFI1	$2k+1$
VFI2	$2k$
VFI3	$2k+1$
VFI4	$2k+1$
VFI5	$4k+1$

The space complexities of other classification algorithms are also given. The space requirement of the FIL algorithms is $O(i \cdot d)$, where i is the average number of intervals constructed on a feature dimension [7]. Since the maximum value i can take is m, this space complexity is at worst $O(m \cdot d)$ but is usually less than $O(m \cdot d)$. The space requirement of the COFI algorithm and the CFP algorithm is $O(i \cdot d)$, where i is the number of overlapping feature intervals in the COFI algorithm and disjoint segments in the CFP algorithm [73, 71]. Similarly, their space complexity is at worst $O(m \cdot d)$ but is usually less than $O(m \cdot d)$.

The space complexity of both 1-NN and 1-NNFP algorithms is $O(m \cdot d)$, since all the training instances are stored in memory. In NBCN, a mean and variance for the training instances of each class on each linear feature is kept in memory, which requires $O(d \cdot k)$ memory units assuming that all features are linear. However, a nominal feature requires $O(k \cdot v)$ memory units because the frequency of each class is kept for each distinct value of that nominal feature, where v is the number of distinct values of that nominal feature.

5.1.2 Time Complexity of Training

In the training phase of the VFI1 algorithm, the end points on each feature dimension are found and sorted. Since there are $2k$ end points and d features, this requires $O(d \cdot k \cdot \lg k)$ time. After sorting the end points, for each training instance the corresponding interval on each feature dimension is searched and the counts of corresponding classes are incremented. Since there are m training

instances and at most $2k+1$ intervals on each feature dimension, this requires $m \cdot d \cdot (2k+1)$ time units at worst. The total time requirement becomes $O(d \cdot k \cdot \lg k + m \cdot d \cdot k) = O(m \cdot d \cdot k)$. Hence, the training time of VFI increases with the number of features and classes, and the size of the dataset. The other versions of VFI might have different maximum number of intervals than VFI1 does as shown in Table 5.1. However, this does not change the asymptotic upper bound time complexity of $O(m \cdot d \cdot k)$. VFI4 checks all the end points to find out whether to construct point intervals, but this check also does not change the asymptotic training time complexity.

The training time complexity of the FIL algorithms is $O(d \cdot m \cdot \lg m)$ [7]. The training time complexity of the COFI algorithm is $O(m^2 \cdot d)$ at worst [73]. The training time complexity of the 1-NNFP algorithm is $O(d \cdot m \cdot \lg m)$ [8]. The training time complexity of the well-known 1-NN algorithm is $O(m \cdot d)$ because of the normalization of all feature values into a same range. Since all the training instances of each class are processed to compute the mean and variance on each feature, the training time complexity of the NBCN algorithm is $O(m \cdot d \cdot k)$.

5.1.3 Time Complexity of a Single Classification

In the classification phase of the VFI1 algorithm, for each feature, the interval that the corresponding feature value of the test example falls into, is searched and the individual votes of each feature is summed up to get the total votes. Since there are at most $2k+1$ intervals on each feature dimension, the classification phase takes at worst case $d \cdot (2k+1)$ time units which is $O(d \cdot k)$. Since there are at most $2k$ intervals on each feature in the VFI2 algorithm, the classification in VFI2 requires $d \cdot 2k$ time units which is asymptotically again $O(d \cdot k)$. Although the types of the interval boundaries are considered during classification in both the VFI3 and VFI4 algorithms, this does not affect the asymptotic time complexity. Therefore, the classification time complexity of both VFI3 and VFI4 is also $O(d \cdot k)$. The VFI5 algorithm has $4k+1$ intervals at most and the classification requires $d \cdot (4k+1)$ time units which is again $O(d \cdot k)$. Hence, a single classification time of all VFI classifiers increases with

the number of features and classes.

The classification time complexity of a single test instance in the FIL algorithms is $O(d \cdot \lg m)$ [7]. The classification time of the COFI algorithm is $O(m \cdot d)$ at worst [73]. The classification time complexity in the 1-NNFP is $O(d \lg m))$ because the time complexity to find the nearest neighbor among sorted values of each feature dimension is $O(\lg m)$ [8]. On the other hand, in the well-known 1-NN algorithm, the classification of an instance requires the computation of its distance to m training instances on d dimensions. Time complexity of computing the distance between two instances is $O(d)$, so computing the distance to m training instances is $O(m \cdot d)$. To find the nearest neighbor, which has the minimum distance to the new instance, among m instances is $O(m)$. Therefore, the classification time complexity of a single instance in the 1-NN algorithm is $O(m \cdot d$. Since the conditional probability that a given test instance belongs to a class given a feature value is directly computed from the corresponding mean and variance of the normal distribution of training instances of that class on that feature, the classification time complexity of a single test instance in NBCN is $O(d \cdot k)$.

5.2 Empirical Evaluation of the VFI Classifiers on Real-World Datasets

Empirical evaluation is clearly essential to the process of designing and implementing new algorithms. In this section empirical evaluation of the VFI algorithms compared with C4.5[1], NBCN, CFP[2], COFI[3], 1-NN, 1-NNFP, and FI4[4] algorithms on real-world datasets which are widely used by machine learning researchers and provided by the machine learning group at the University of California at Irvine [51]. Since experimental science is concerned with data that occurs in real world, machine learning research on classification algorithms

[1] In all the experiments, C4.5 was run with default settings and pruned results are reported.

[2] In all the experiments, CFP was run with $D_f = 0.1$ and $\triangle = 0$.

[3] In all the experiments, COFI was run with $g = 0.1$.

[4] FI4 is chosen to represent the FIL algorithms.

are usually compared on these commonly used datasets. An overview of the datasets is shown in Table A.1.

The VFI classifiers are also applied on two medical datasets compiled during this thesis [23, 33]. In several medical domains the inductive learning systems were actually applied, for example, two classification systems are used in localization of primary tumor, prognostics of recurrence of breast cancer, diagnosis of thyroid diseases, and rheumatology [44]. The domain for one of our datasets, which is called as *Dermatology* in this thesis, is for *Differential Diagnosis of Erythemato-Squamous Diseases* (see Appendix for more information). The problem in the other domain is to distinguish between the presence and type of *cardiac arrhythmia* and to classify it in one of the 16 groups (see Appendix) and the compiled dataset is called *Arrhythmia* in this thesis. The datasets consist of a set of descriptions of patients with known diagnoses predicted by a medical expert. After a concept description is learned by the learning systems, a diagnosis for a new patient is predicted using the learned description.

We will also evaluate the VFI algorithms on artificial datasets in order to observe the effect of irrelevant features, noise, and unknown values on the classification accuracy. The next section describes the methodologies used in the experiments. Section 5.2.2 presents the performance of the VFI algorithms on real-world datasets. In Section 5.2.3, some experiments are described on artificial datasets.

5.2.1 Testing Methodology

This section briefly describes the methodologies used in the empirical evaluations of machine learning algorithms. Improved performance is the major aim of learning algorithms [41]. These various performance measures are the natural dependent variables for machine learning experiments, just as they are for studies of human learning. The accuracy and efficiency of an algorithm can be measured by various performance measures. There are three important measures of evaluation for a learning algorithm: *accuracy*, *time*, and *space complexities*.

```
cross–validation (Classifier, DataSet, N)
begin
  divide the DataSet into N folds
  repeat N times
    TestSet = a fold that hasn't been used for test yet
    TrainSet = DataSet − TestSet
    accuracy = Classifier(TrainSet,TestSet)
  return average of N accuracies
end.
```

Figure 5.1. The algorithm for N–fold cross–validation.

For supervised concept learning (classification) tasks, the most commonly used metric is the percentage of correctly classified test instances over all test instances. This metric cannot be used for unsupervised learning tasks like conceptual clustering, but this measure can be generalized as the average ability to predict attribute values [26]. Accuracy of an algorithm is a measure of correct classifications on a test set of unseen instances. There are several ways of measuring the accuracy of an algorithm, in the literature the common techniques are *cross-validation*, *leave-one-out* and *average of randomized runs*.

N-fold Cross-Validation: In Figure 5.1, the algorithm for N-fold Cross-Validation is shown. In this technique, a dataset is partitioned into N mutually disjoint subsets with the same cardinality or in a way that the cardinalities differ at most by 1. The $N-1$ of these sets are used as the training set, and the remaining one is used as the test set. This process is repeated for N times, once for each subset being the test set. Classification accuracy is measured as the average accuracy on all test sets. The union of the all test sets is equal to the whole dataset. Cross-validation ensures that the training and test sets are disjoint.

Leave-one-out: This technique is a special case of N-fold cross-validation where $N = m$. That is, for a dataset containing m instances, training set contains $m-1$ instances whereas test set contains only 1 instance. Then, this is repeated for all instances being test instance each time leading to m-fold

cross-validation. It is an elegant and straightforward technique for estimating classifier error rates. Evidence for the superiority of the leave-one-out approach is documented in the literature [25, 46]. While leave-one-out can be preferred for small datasets, it is computationally expensive for large datasets [39].

Average of Randomized Runs: In this method, the algorithm is tested over randomly selected training and testing sets. The important point is that training and test sets must be disjoint. The test is repeated for a fixed number of times. The classification accuracy is determined as the average accuracy across all trials.

In the previous section, we have computed the time and space complexities of the VFI algorithms. In the following subsection, the performance of the VFI algorithms will be given in terms of their classification accuracies. In order to measure the classification accuracy of an algorithm on a dataset, first the dataset is shuffled with a random seed such that the classes are equally distributed and 10-fold cross-validation is applied on this shuffled dataset 10 times, each time using a different seed. Then the average of these 10 10-fold cross-validation accuracies makes up the classification accuracy that we report as the accuracy measurement of an algorithm in this thesis. Our cross-validation program provides the same disjoint training and testing sets each time for each algorithm in order to compare the results under same conditions. Disjoint training and testing sets make sure that unseen test instances are classified to measure the accuracy of algorithms. Repeating the cross-validation several times on different shuffles of the dataset enables the performance measurement to be more robust.

5.2.2 Experiments on Real-World Datasets

In order to evaluate the VFI algorithms empirically, we have performed some experiments on real-world datasets from the collection of UCI-Repository [51] and two new real-world datasets compiled during this thesis. These domains help us to compare the VFI algorithms with other classification algorithms as well as demonstrating the applicability of the VFI algorithms to real-world problems. Detailed information about these real-world datasets are given in

Table 5.2. Classification accuracy (%) of feature projection based methods —CFP, COFI, 1-NNFP, FI4, VFI1, VFI2, VFI3, VFI4, VFI5— obtained by averaging 10 10-fold cross-validation results on eighteen real-world datasets.

Inducer:	CFP	COFI	1-NNFP	FI4	VFI1	VFI2	VFI3	VFI4	VFI5
Arrhythmia	54.23	55.15	50.98	57.79	52.15	48.87	52.54	45.8	**61.49**
Bcancerw	95.64	56.08	95.01	**97.17**	96.2	95.67	88.48	88.48	95.08
Cleveland	74.08	82.32	68.24	79.07	82.09	**83.35**	81.63	81.63	81.86
Dermatology	50.24	94.56	47.18	59.46	95.98	96.14	93.34	96.58	**96.64**
Diabetes	66.12	63.94	66.17	**68.09**	56.55	66.64	64.33	64.33	54.73
Glass	54.54	50.74	53.79	42.52	57.3	55.51	55.38	54.95	**58.81**
Horse	66.47	77.48	68.02	75.33	**78.13**	76.8	77.99	77.99	78.05
Hungarian	68.75	83.57	71.88	76.78	83.43	84.31	82.85	82.85	**85.23**
Ionosphere	87.29	64.12	87.14	88.37	84.55	85.92	90.48	**90.57**	81.07
Iris	89.4	91.33	87.0	92.13	95.93	94.6	93.8	93.8	**96.0**
Liver	58.17	52.62	54.31	**61.42**	59.75	56.58	58.18	58.18	59.24
Musk	72.62	57.32	76.49	**81.94**	75.48	72.06	75.29	75.29	76.97
New-thyroid	87.61	92.44	89.44	87.22	93.75	**94.94**	93.56	93.56	92.63
Page-blocks	90.06	**91.22**	90.86	90.64	87.18	86.41	86.75	86.75	88.02
Segmentation	77.27	**83.72**	76.18	78.53	77.4	76.9	77.68	77.68	77.03
Sonar	68.02	65.07	63.7	65.62	59.64	**68.06**	57.98	57.98	58.75
Vehicle	56.73	36.78	51.36	58.74	52.91	**59.16**	53.72	53.72	57.39
Wine	87.97	91.5	85.05	89.15	**97.13**	95.38	96.34	96.34	96.4
Average:	72.51	71.66	71.27	74.99	76.97	77.63	76.69	76.47	77.52

Appendix A.

Since the motivation of developing the VFI classifiers comes from other feature projection based methods, we first compare the classification accuracies of the VFI algorithms with those of other feature projection based methods described in Chapter 3. The classification accuracies of the VFI classifiers compared with those of the CFP, the COFI, the 1-NNFP, and the FI4 classifiers obtained by averaging 10 10-fold cross-validation results on eighteen real-world datasets are given in Table 5.2. The highest classification accuracy for each dataset is shown in bold. The results show that it is usually one of the VFI classifiers that has the highest accuracy among all other feature projection based methods. These experiments empirically show that VFI classifiers are the best performing feature projection based technique in terms of classification accuracy. Each time a different version might achieve the highest accuracy, that's why we present all the versions in this thesis. Although the accuracies of different versions of VFI on a given dataset are usually close to each other, it might differ on a few datasets. At the bottom of the table, the average of the accuracies of each classifier on all datasets is also shown by "Average",

Table 5.3. Classification accuracy (%) of VFI1, VFI2, VFI3, VFI4, VFI5, NBCN, 1-NN, and C4.5 obtained by averaging 10 10-fold cross-validation results on eighteen real-world datasets.

Inducer:	VFI1	VFI2	VFI3	VFI4	VFI5	1-NN	NBCN	C4.5
Arrhythmia	52.15	48.87	52.54	45.8	61.49	53.93	50.78	66.99
Bcancerw	96.2	95.67	88.48	88.48	95.08	95.28	96.01	94.99
Cleveland	82.09	83.35	81.63	81.63	81.86	77.34	83.53	74.85
Dermatology	95.98	96.14	93.34	96.58	96.64	95.63	87.47	94.68
Diabetes	56.55	66.64	64.33	64.33	54.73	70.46	75.43	74.15
Glass	57.3	55.51	55.38	54.95	58.81	69.63	46.33	69.25
Horse	78.13	76.8	77.99	77.99	78.05	79.09	78.04	85.42
Hungarian	83.43	84.31	82.85	82.85	85.23	76.52	84.13	79.29
Ionosphere	84.55	85.92	90.48	90.57	81.07	86.79	87.46	89.74
Iris	95.93	94.6	93.8	93.8	96.0	95.26	95.39	93.72
Liver	59.75	56.58	58.18	58.18	59.24	62.51	56.14	66.37
Musk	75.48	72.06	75.29	75.29	76.97	85.39	72.64	82.91
New-thyroid	93.75	94.94	93.56	93.56	92.63	96.85	96.35	92.73
Page-blocks	87.18	86.41	86.75	86.75	88.02	96.04	90.16	96.93
Segmentation	77.4	76.9	77.68	77.68	77.03	97.22	79.72	96.99
Sonar	59.64	68.06	57.98	57.98	58.75	86.54	67.55	73.45
Vehicle	52.91	59.16	53.72	53.72	57.39	69.74	45.47	72.7
Wine	97.13	95.38	96.34	96.34	96.4	95.08	97.46	93.78
Average:	76.97	77.63	76.69	76.47	77.52	82.74	77.23	83.27

and these average accuracies also show that VFI classifiers achieve better than other feature projection based methods on the average.

Next, the classification accuracies of VFI classifiers compared with those of the NBCN, the 1-NN, and the C4.5 algorithms obtained by averaging 10 10-fold cross-validation results on eighteen real-world datasets are given in Table 5.3. The experiments show that the highest average accuracy results are those of C4.5 followed by 1-NN. VFI classifiers are outperformed by C4.5 and 1-NN significantly on only one-third of all the datasets. On seven of the datasets, which is nearly one-third of all the datasets, VFI classifiers achieve better than both C4.5 and 1-NN. C4.5 and 1-NN are the state-of-the-art classification algorithms in machine learning and differ from all the other algorithms in this thesis that they do not consider each feature separately. The separate consideration of features is common in all the feature projection based methods and the Naive Bayesian Classifier, which is also a classical classifier. When we

Table 5.4. Average training running times (msec.) of CFP, COFI, 1-NNFP, FI4, VFI1, NBCN, and 1-NN on a SUN Sparc 20/61 workstation. Training is done with 9/10 instances of the whole dataset.

Inducer:	CFP	COFI	1-NNFP	FI4	VFI1	NBCN	1-NN
Arrhythmia	7461.20	282.00	1453.80	2261.94	829.98	764.78	90.64
Bcancerw	173.00	43.27	146.02	154.93	19.74	11.84	5.15
Cleveland	111.93	21.17	82.32	69.73	11.01	60.63	2.75
Dermatology	358.73	39.84	272.13	202.07	53.50	57.25	8.58
Diabetes	274.94	48.69	77.47	113.96	18.67	11.06	5.00
Glass	82.89	14.84	19.30	31.04	11.73	12.73	1.19
Horse	171.00	29.26	342.68	331.09	18.71	105.25	5.45
Hungarian	88.89	19.47	195.62	195.53	9.98	44.06	2.36
Ionosphere	535.00	40.63	114.68	192.21	42.59	29.23	8.31
Iris	19.35	8.53	5.03	7.82	3.50	5.12	0.48
Liver	74.56	21.30	20.83	27.10	7.25	6.69	1.60
Musk	4762.64	174.86	861.63	1184.50	266.00	151.65	56.40
New-thyroid	31.54	12.54	9.28	48.06	5.01	5.74	0.80
Page-blocks	5349.40	338.94	1208.66	1100.16	244.55	64.34	93.39
Segmentation	12795.00	171.47	860.20	1291.16	200.00	61.44	55.26
Sonar	624.84	40.74	113.48	209.00	50.60	43.06	7.13
Vehicle	901.00	61.26	202.28	205.00	54.24	27.93	12.58
Wine	83.34	12.54	20.32	35.08	11.14	12.17	1.52
Average:	1883.29	76.74	333.65	425.58	103.23	81.94	19.92

compare the accuracies of VFI classifiers with those of NBCN, we observe that VFI classifiers outperform NBCN on most of the datasets. The average accuracies of NBCN and the VFI classifiers are approximately equal to each other which shows that VFI classifiers achieve comparably with NBCN.

Although NBCN and VFI classifiers lose in classification accuracy on some datasets compared to C4.5 and 1-NN, they provide much faster classification running times with only small increases in training times. To show this empirically, the average training and classification running times of all algorithms are shown in Table 5.4 and Table 5.5 respectively. In these tables, we show the FI4 algorithm as representative for the FIL algorithms and VFI1 as representative for the VFI algorithms since the training times of all the versions are almost equal. On the average, the fastest in training is the 1-NN because it is totally a lazy learner and it does not process the input examples. The COFI, NBCN, and VFI1 algorithms go beyond this lazy learning and learns some concept

Table 5.5. Average classification running times (msec.) of CFP, COFI, 1-NNFP, FI4, VFI1, NBCN, and 1-NN on a SUN Sparc 20/61 workstation. Classification is done with 1/10 instances of the whole dataset and 0 msec. means less than 0.1 msec.

Inducer:	CFP	COFI	1-NNFP	FI4	VFI1	NBCN	1-NN
Arrhythmia	356.00	193.06	101.00	41.31	122.70	689.68	5478.24
Bcancerw	4.10	4.14	1.78	1.54	1.83	6.56	773.05
Cleveland	2.76	2.06	1.03	0.87	1.01	2.99	156.38
Dermatology	6.91	9.37	3.93	2.20	4.90	33.15	514.28
Diabetes	11.71	4.33	4.99	2.37	1.76	6.35	832.23
Glass	3.56	1.97	1.62	0.66	0.82	6.26	63.19
Horse	3.51	3.03	2.00	1.36	1.71	4.35	327.00
Hungarian	2.34	1.94	1.00	0.87	0.96	2.58	144.73
Ionosphere	22.11	4.43	15.94	4.56	3.36	11.8	458.58
Iris	0.66	0.83	0.00	0.17	0.20	1.50	19.60
Liver	3.09	1.70	1.07	0.50	0.74	2.40	137.20
Musk	237.00	24.33	96.40	35.56	33.10	73.98	3561.00
New-thyroid	1.16	1.17	0.30	0.50	0.38	2.27	46.03
Page-blocks	297.08	46.29	85.16	24.59	22.59	125.35	55784.00
Segmentation	875.45	35.70	73.10	20.36	21.91	141.00	13688.30
Sonar	26.57	3.80	19.02	5.66	4.00	12.72	258.47
Vehicle	35.36	9.19	10.34	4.58	5.55	28.60	1667.03
Wine	2.81	1.46	2.08	0.81	0.87	3.74	54.51
Average:	104.12	19.38	23.38	8.25	12.68	64.18	4664.65

descriptions in less than 0.1 seconds. The 1-NNFP and FI4 algorithms are 3 and 4 times slower than VFI1 but they still learn their concept descriptions in less than half a second. The CFP algorithm is the slowest in learning its concept description. If we look at the individual training times, wee see that CFP is slower on datasets with either large number of instances or features than it is on small datasets and/or datasets with small number of features.

When a classifier such as 1-NN is totally lazy in learning, everything is left to classification process that causes the classifier to be enormously slow in classification, compared to all feature projection based learning methods and the NBCN algorithm. The average classification time of the 1-NN algorithm is 388 times more than the VFI1 algorithm as shown in Table 5.5. This empirical result is supported by the complexity analyses in Section 5.1.3 where the complexity of classifying a single instance in VFI1 is in $O(d \cdot k)$ whereas the complexity of classifying a single instance in 1-NN is in $O(m \cdot d$. All feature

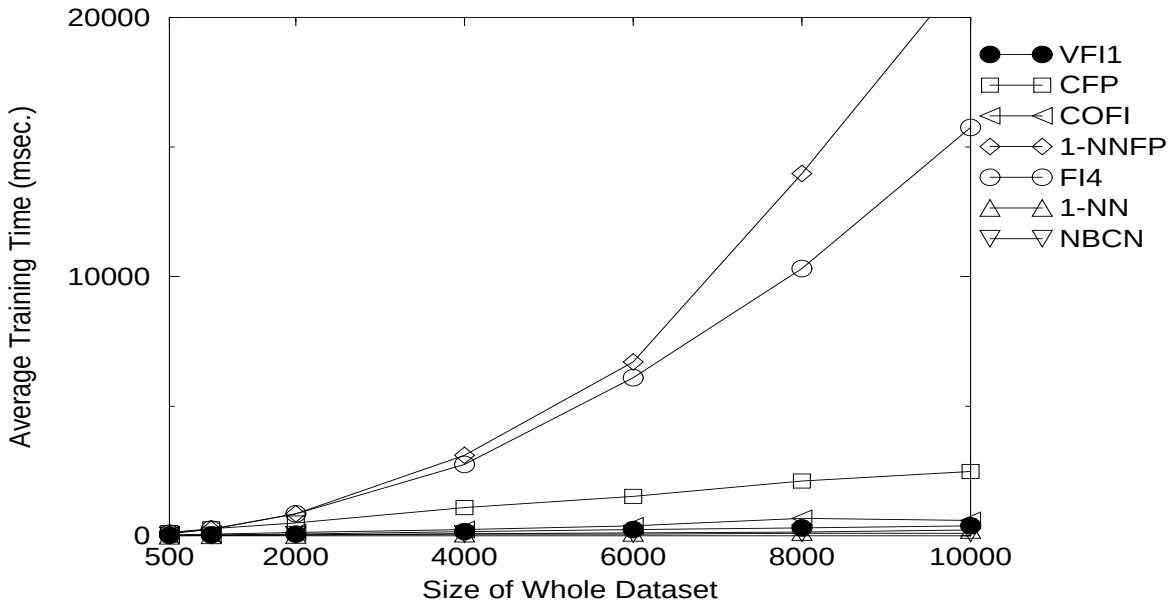

Figure 5.2. Average training time of all classifiers on datasets with increasing number of instances. 9/10 of the whole dataset is used in training.

projection based learning methods are very fast in classification which is a result of this knowledge representation scheme. However, the CFP algorithm is a little slower in classification than the others as in the case of training.

When we compare the average classification time of our VFI1 classifier with that of the NBCN classifier, we observe that VFI1 is 5 times faster than NBCN. Naive Bayesian Classifiers are fast classical classifiers, and Kononenko pointed out that induction of decision trees is relatively slow as compared to Naive Bayesian classifier [44]. Since VFI1 requires approximately equal training time and faster classification time compared to the Naive Bayesian classifier, VFI1 is also faster than decision tree inducers.

In order to see the effect of the size of the dataset in running times of the classifiers, I have run all the classifiers on datasets with increasing number of instances. The effect of the data size on the average training time is shown in Figure 5.2. The 1-NNFP and FI4 algorithms are the ones that suffer mostly with increasing data size. The COFI algorithm shows a smooth increase and all the others are not affected much. The VFI1 algorithm is used to represent the

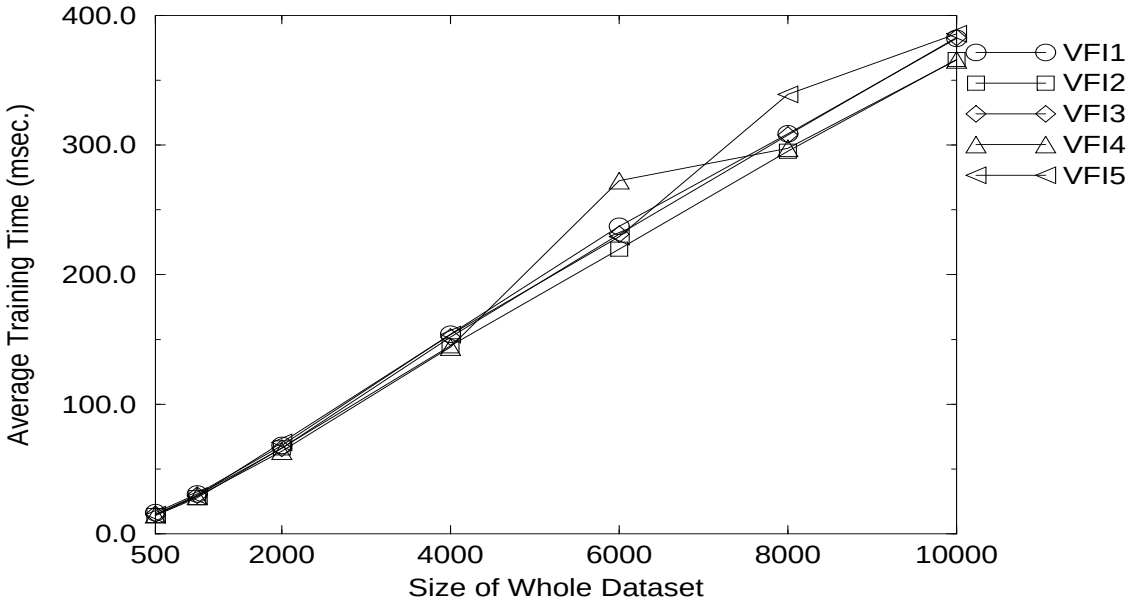

Figure 5.3. Average training time of all VFI versions on datasets with increasing number of instances. 9/10 of the whole dataset is used in training.

VFI algorithms, but a further comparison of the training times with increasing data size of all versions is shown in Figure 5.3.

The effect of data size on the average classification times shown in Figure 5.4 is more important to compare the VFI1 algorithm with the 1-NN algorithm. The 1-NN algorithm has such a sharp increase in classification time that the other algorithms are nearly unseen in the graph shown in Figure 5.4. Figure 5.5 shows the comparison of classification times of all other algorithms with increasing data size. This graph shows that the classification times of the VFI1, FI4, and 1-NNFP algorithms increase the least among all other algorithms. The NBCN algorithm is the mostly affected algorithm from the increase in the data size. The VFI1 algorithm is used to represent the VFI algorithms in the above graphs, but a further comparison of the classification times with increasing data size of all versions is shown in Figure 5.6.

In this section, we have empirically compared the VFI classifiers in terms of classification accuracy, average training times, and average classification times with several other classification algorithms on real-world datasets. The

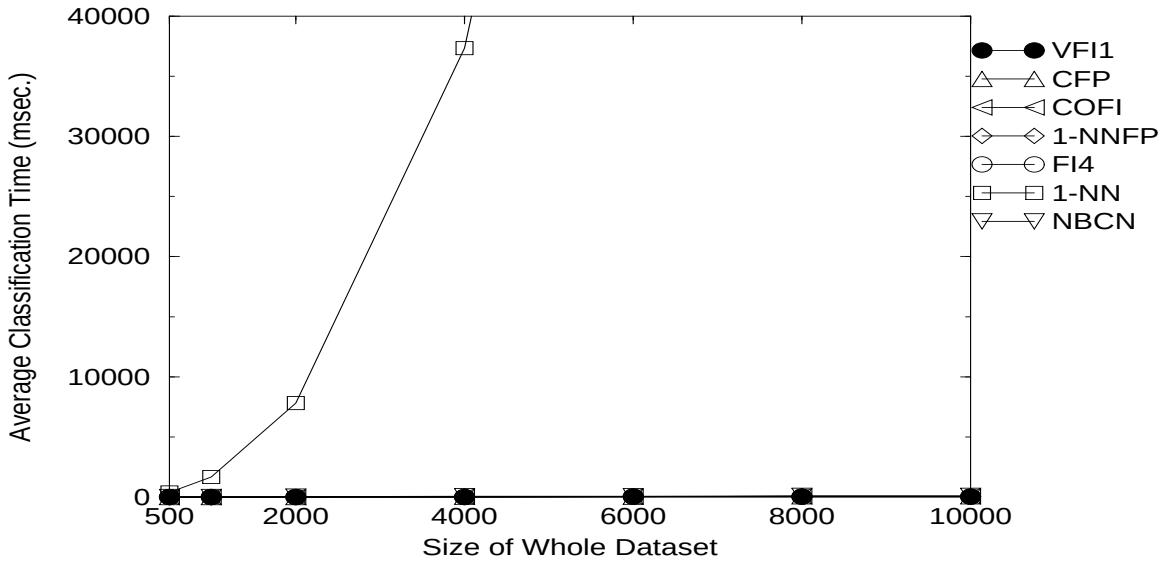

Figure 5.4. Average classification time of all classifiers on datasets with increasing number of instances. 1/10 of the whole dataset is used in classification.

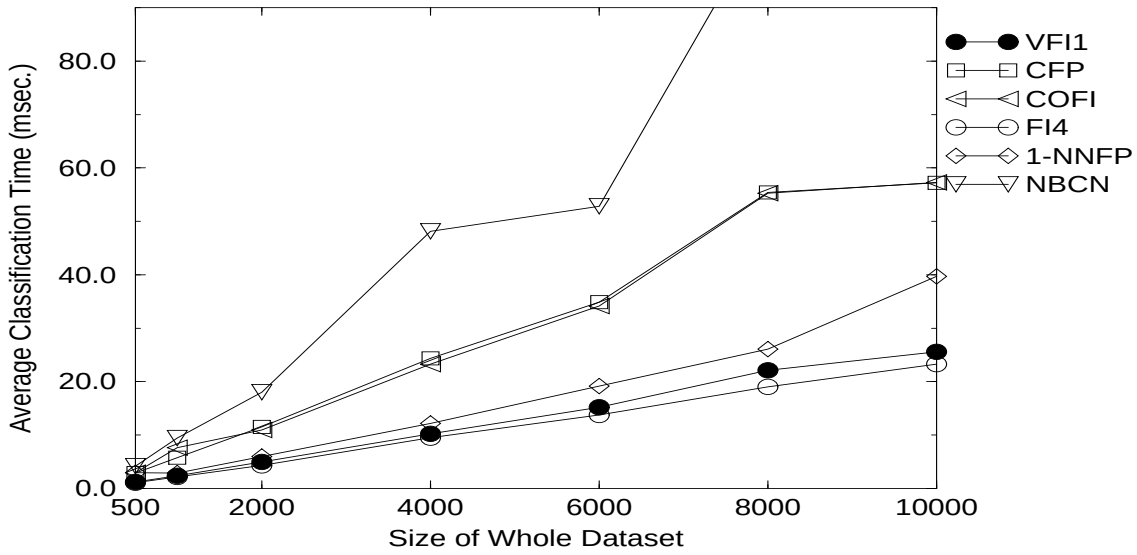

Figure 5.5. Average classification time of all classifiers except the 1-NN algorithm on datasets with increasing number of instances. 1/10 of the whole dataset is used in classification.

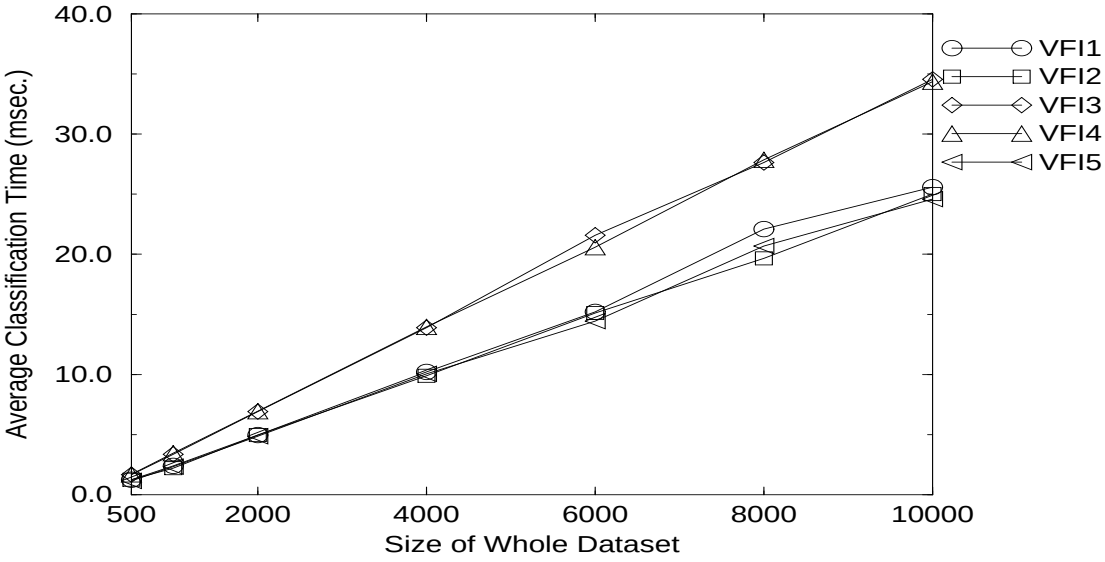

Figure 5.6. Average classification time of all VFI versions on datasets with increasing number of instances. 1/10 of the whole dataset is used in classification.

accuracy results have shown that VFI classifiers achieve the highest accuracies among all feature projection based methods. When compared to some well-known classification methods, VFI classifiers are outperformed by 1-NN and C4.5 on the average but usually perform better than NBCN. However, the classification in the VFI classifiers are shown to be much much faster than 1-NN and even faster than NBCN. Moreover, it is possible to improve the classification accuracy of the VFI classifiers significantly by learning feature weights, which will be discussed in Chapter 6.

5.2.3 Experiments on Artificial Datasets

Real-world domains might have irrelevant features, noisy instances, and unknown (missing) feature values. Learning even in the presence of irrelevant features is an important criteria for a learning system [10] as well as learning from noisy and/or incomplete data [47]. Therefore, we generated artificial datasets from a real-world dataset by adding irrelevant features, noise, and unknown values and empirically evaluated the VFI algorithms compared with

other classifiers. We used the real-world dataset Iris in our experiments and observed the change in the classification accuracy of the classifiers as we added either some irrelevant features, or some noisy values, or some unknown values to the Iris dataset. We chose the Iris dataset in our experiments because it is a commonly used dataset and nearly all of the classifiers are successful on this dataset. The original Iris dataset does not contain any unknown feature values.

Next section presents the results of experiments with increasing number of irrelevant features added to the Iris dataset. Section 5.2.3.2 presents the effect of increasing noise level for the VFI algorithms compared with other algorithms. Section 5.2.3.3 presents the effect of unknown values in both the training and testing dataset that includes new instances to be classified.

5.2.3.1 Experiments with Increasing Number of Irrelevant Features

Real-world datasets may contain irrelevant features or unequally relevant features. For example, medical doctors usually have this relevance information in their mind and distinguish diseases from each other by paying more attention to some more relevant features. Machine learning researchers have developed feature selection methods to cope with irrelevant features [6, 38, 67].

We investigated the effect of irrelevant features on the classification accuracy of the VFI classifiers, and compared with other classifiers. In these experiments, we added some irrelevant continuous features with randomly assigned values to the Iris dataset. The number of such artificially added irrelevant features is an even number between 0 and 10.

The classification accuracies of the VFI classifiers compared with other feature projection based learning methods with increasing number of irrelevant features are plotted in Figure 5.7. The x-axis shows the number of irrelevant features added to the Iris dataset. The y-axis shows the 10-fold cross-validation accuracy results obtained from the average of 50 runs of the classifiers on the generated datasets. The accuracy of VFI classifiers are not affected with the addition of irrelevant features, even when there are 10 irrelevant features. On

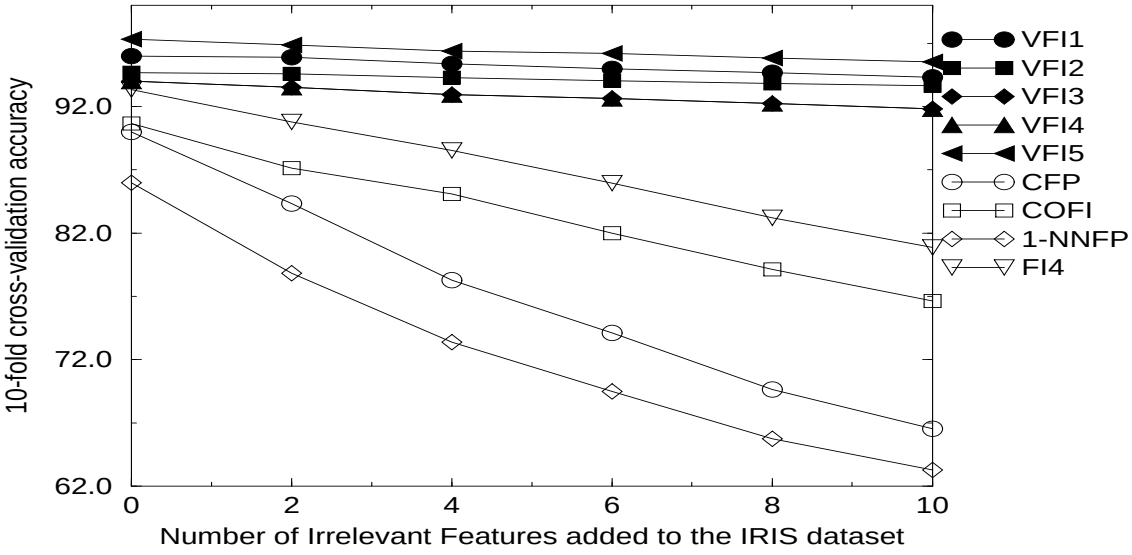

Figure 5.7. 10-fold cross-validation accuracy results of the VFI algorithms compared with that of CFP, COFI, 1-NNFP, and FI4 algorithms on Iris dataset with increasing number of irrelevant attributes.

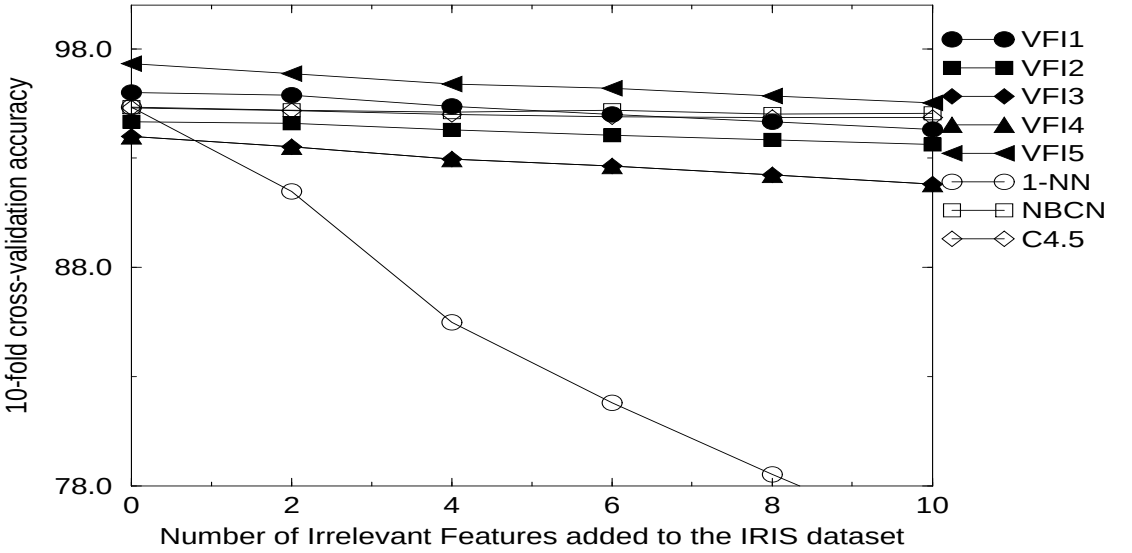

Figure 5.8. 10-fold cross-validation accuracy results of the VFI algorithms compared with that of 1-NN, C4.5, NBCN algorithms on Iris dataset with increasing number of irrelevant attributes.

the other hand, all the other feature projection based methods seem to be affected negatively with the addition of irrelevant features. The performance decrease slope of FI4 and COFI look nearly same whereas the performance decrease of 1-NNFP and CFP have sharper slopes.

Figure 5.8 shows the comparison of the VFI classifiers with other well-known classifiers described in Chapter 2 such as C4.5, NBCN, and 1-NN. As shown in the previous graph, VFI classifiers are not affected with the addition of irrelevant features. Similarly, the classification accuracies of both NBCN and C4.5 have not also changed. However, the 1-NN algorithm is negatively affected with the addition of irrelevant features.

5.2.3.2 Experiments with Increasing Noise Level

This section investigates the effect of noise in the datasets on the VFI algorithms compared to other algorithms. There are two major types of noise that can be found in real-world datasets [3, 11, 15, 27, 69]:

1. Feature (attribute) noise, defined as incorrect feature value.
2. Classification noise, defined as incorrect class label of an instance.

Quinlan demonstrated that feature noise, occurring simultaneously in all features describing the instances, can result in faster decrease in classification accuracy than noise only in the class label does [54]. Therefore, we studied the feature noise in our experiments. Feature values of the Iris dataset only in the training set are replaced with random values in the feature domain with an increasing noise probability. We experiment with noise probabilities from 0.05 to 0.5.

The classification accuracies of the VFI classifiers compared with other feature projection based methods with increasing level of noise in feature values are plotted in Figure 5.9. The x-axis shows the probability of noise (level of noise) added to the Iris dataset. The y-axis indicates the 10-fold cross-validation accuracy results obtained from the average of 50 runs of the classifiers on the generated datasets. The VFI classifiers seem to be negatively

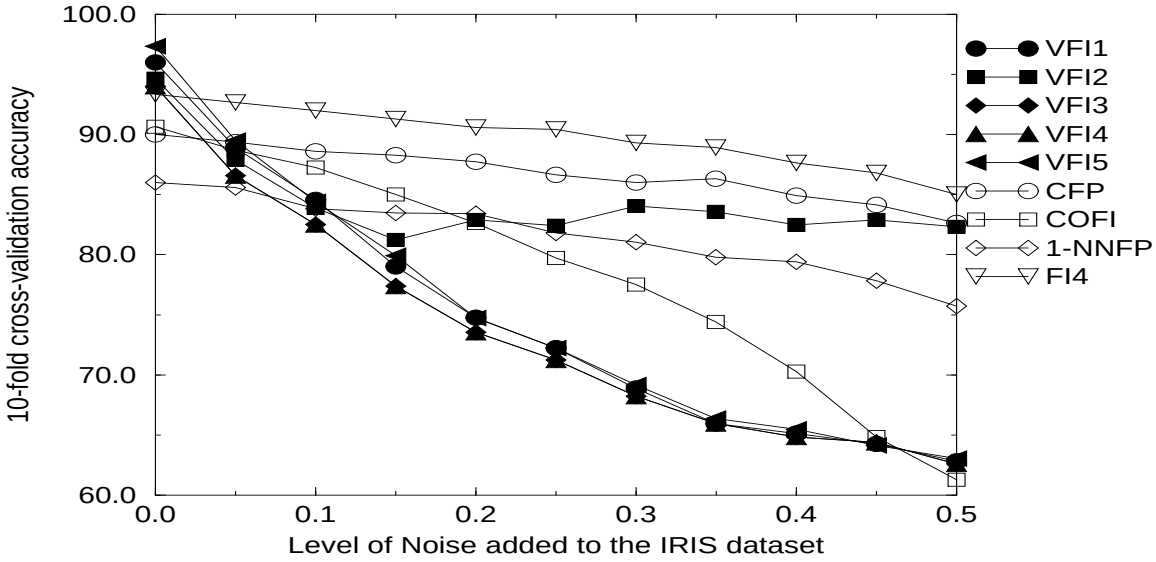

Figure 5.9. 10-fold cross-validation accuracy results of the VFI algorithms compared with that of CFP, COFI, 1-NNFP, and FI4 algorithms on Iris dataset with increasing level of noise.

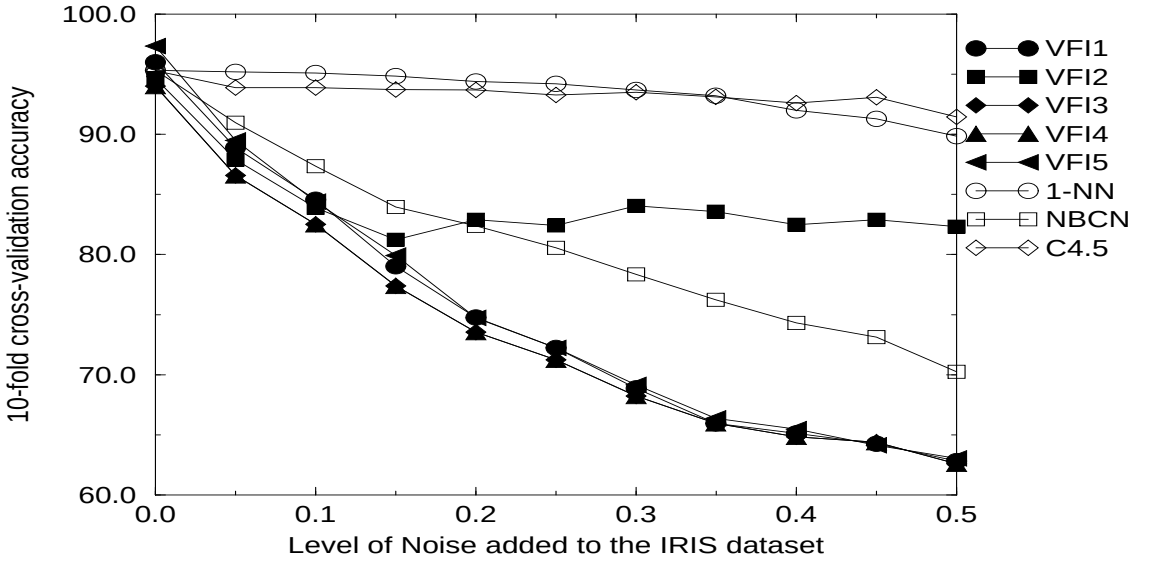

Figure 5.10. 10-fold cross-validation accuracy results of the VFI algorithms compared with that of 1-NN, C4.5, NBCN algorithms on Iris dataset with increasing level of noise.

affected with the addition of noisy values. The accuracies of almost all the versions significantly decrease with increasing level of noise except VFI2, which is affected much less than the other versions in the presence of noise. This is a superiority of VFI2 over other versions of VFI classifiers. The accuracy of COFI algorithm also decreases with increasing level of noise. The feature projection based methods that are not affected significantly from the addition of noise are CFP, 1-NNFP, and FI4 algorithms.

Figure 5.10 shows the comparison of the VFI classifiers with other well-known classifiers such as C4.5, NBCN, and 1-NN with increasing level of noise. This graph shows that the 1-NN and C4.5 algorithms are not affected with the addition of noise to the Iris dataset. On the other hand, NBCN is negatively affected with the addition of noisy values and performs more poorly than VFI2 in the presence of noise.

5.2.3.3 Experiments with Increasing Level of Missing Values

Most of the real-world datasets contain missing (unknown) feature values and the percentage of missing values are shown in Table A.1. In order to cope with instances that contain missing values, several methods have been proposed [30, 55, 56, 57, 58]. These methods can be summarized as:

- Ignoring instances which have unknown feature values.
- Assuming an additional special value for unknown attribute values.
- Using probability theory by utilizing information provided by context.
- Generating additional instances for all possible values of the unknown attribute.
- Exploring all branches (on decision trees) remembering that some branches are more probable than others.

VFI algorithms follow a natural and plausible approach for handling unknown feature values, they simply ignore only the feature with the unknown

value instead of ignoring the whole instance. What makes this approach possible is the separate consideration of features in both the training and classification phase of the VFI classifiers. Handling unknown feature values is the same as in the other feature projection based learning methods as well as the Naive Bayesian Classifier since they all treat each feature separately. Simply ignoring unknown feature values allows reduction in training and classification time. On the other hand, the 1-NN algorithm tries to determine the value of an unknown feature value using probability distribution of the known values of that feature. For the decision tree induction algorithms, when the value for a feature of an instance is unknown, the test outcome at the decision node testing that feature for that instance will be unknown. C4.5 divides such an instance into probabilistic fragments allowing a single instance to follow multiple paths in the tree. This applies both when the training instances are divided during the construction of a tree and when the tree is used to classify new instances.

Since unknown values might both appear in training data and testing data that contains new instance to be classified, experiments to investigate the effect of increasing level of unknown feature values in both training and testing datasets are performed.

First, I will present the effect of increasing level of unknown feature values in the training dataset on the classification accuracy of several classifiers as well as the VFI classifiers. To generate training datasets with unknown feature values from the Iris dataset, we replace randomly selected feature values to unknown with an increasing unknown probability. In our experiments, the unknown probability varies from 0.0 to 0.5.

The classification accuracies of the VFI classifiers compared with other feature projection based learning methods with increasing percentage of unknown values in feature values of the training dataset are plotted in Figure 5.11. The x-axis shows the probability of unknown values (level of unknown values) added to the Iris training dataset. The y-axis summarizes the 10-fold cross-validation accuracy results obtained from the average of 50 runs of the classifiers on the generated datasets. The accuracy of VFI classifiers are not affected with the addition of unknown feature values, even when the 50% percentage of the dataset is full of unknown values. VFI5 seems to be the version that loses the most in

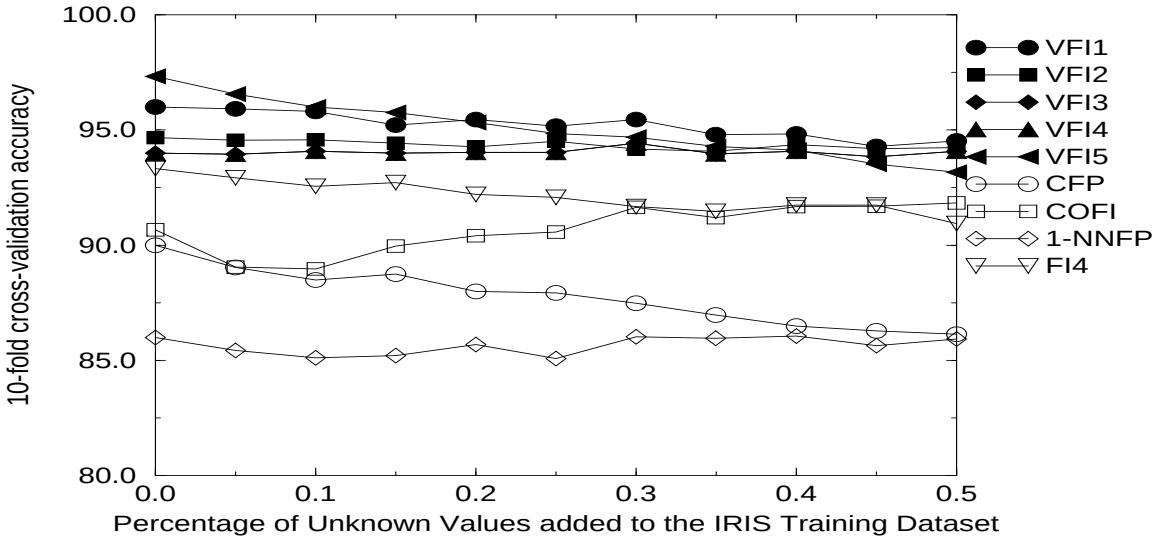

Figure 5.11. 10-fold cross-validation accuracy results of the VFI algorithms compared with those of CFP, COFI, 1-NNFP, and FI4 algorithms on Iris dataset with increasing percentage of unknown values in training dataset.

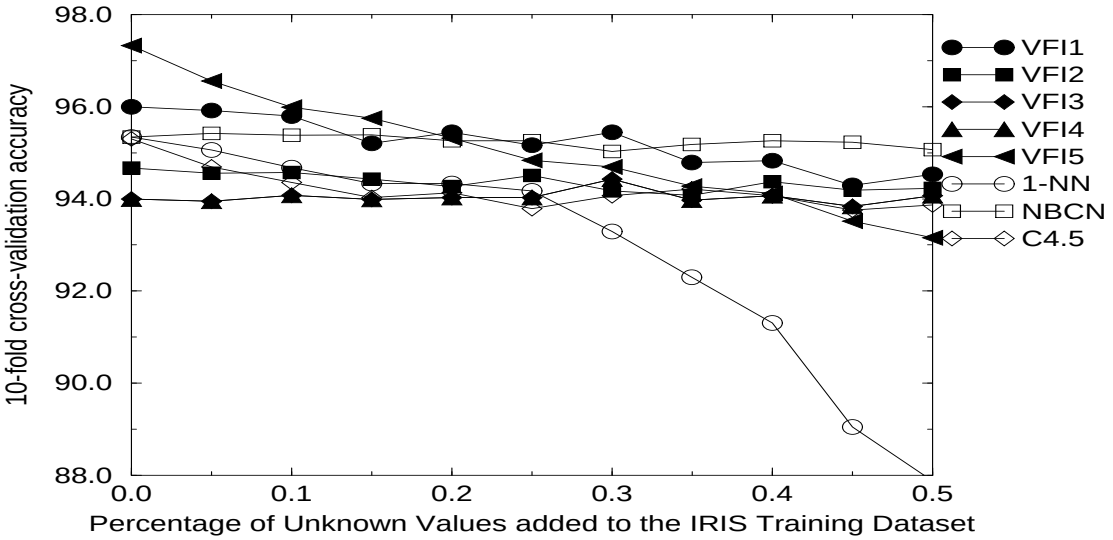

Figure 5.12. 10-fold cross-validation accuracy results of the VFI algorithms compared with those of 1-NN, NBCN, and C4.5 on Iris dataset with increasing percentage of unknown values in training dataset.

accuracy among five versions, but the difference is insignificant. Similarly, the other feature projection based methods do not lose accuracy with the addition of unknown feature values to the training dataset. However, it is interesting that COFI gains classification accuracy with the addition of unknown values.

Figure 5.12 shows the comparison of the VFI classifiers with other well-known classifiers such as C4.5, NBCN, and 1-NN. As shown in the previous graph, VFI classifiers are not affected with the addition of unknown feature values. Similarly, the accuracy of NBCN does not change with the addition of unknown values to the training dataset. Remember that NBCN simply ignores the feature having unknown value as in the other feature projection based methods. The plots of C4.5 is nearly lost in the graph, but it starts at the same point as that of NBCN and stops a little below NBCN. This shows that C4.5 is affected a little more than NBCN but the decrease in its classification accuracy is not that large. The significant degrade in accuracy is observed in 1-NN, which determines the value of an unknown feature value using probability distribution of the known values of that feature. These results show that the method of handling unknown values of all the feature projection based methods and NBCN are superior to that of 1-NN, which is not suitable for simply ignoring only the feature with unknown value.

We also investigated the effect of increasing level of unknown feature values that might exist in the new instances to be classified on the classification accuracy of several classifiers. To generate test datasets including new instances with unknown feature values from the Iris dataset, randomly selected feature values are replaced by unknown. In the experiments, the probability of replacing feature values with an unknown value ranges from 0.0 to 0.5.

Figure 5.13 shows the classification accuracies of the VFI classifiers compared with other feature projection based learning methods with increasing level of unknown values in feature values in the test dataset. The x-axis shows the probability of unknown values (level of unknown values) added to the Iris test dataset. The y-axis shows the 10-fold cross-validation accuracy results obtained from the average of 50 runs of the classifiers on the generated datasets. The accuracy of VFI classifiers decrease only a little with the addition of unknown feature values to the test dataset. VFI5 seems to be the version that

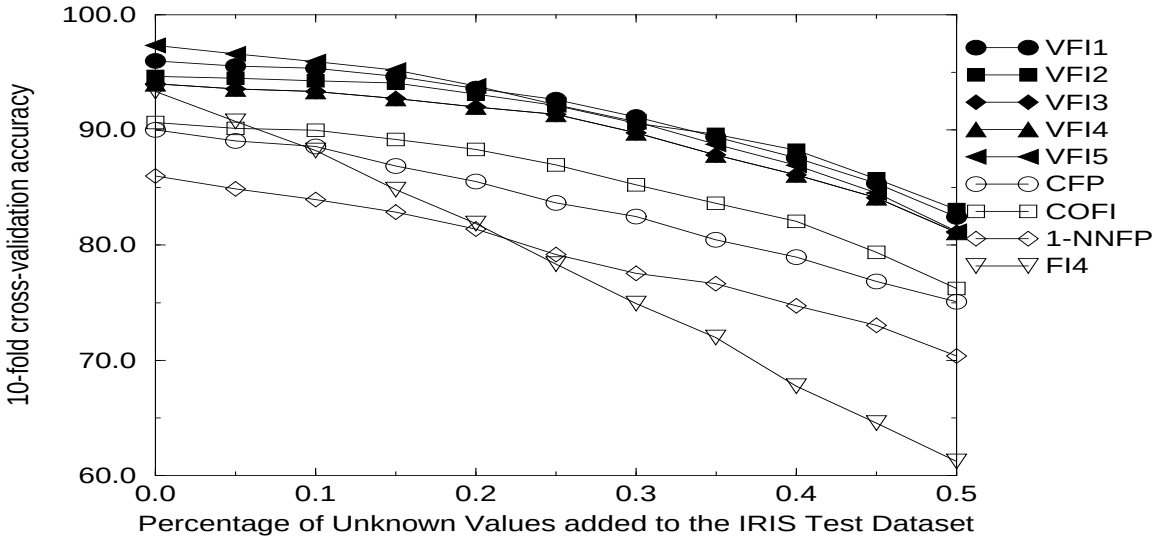

Figure 5.13. 10-fold cross-validation accuracy results of the VFI algorithms compared with those of CFP, COFI, 1-NNFP, and FI4 algorithms on Iris dataset with increasing percentage of unknown values in test data.

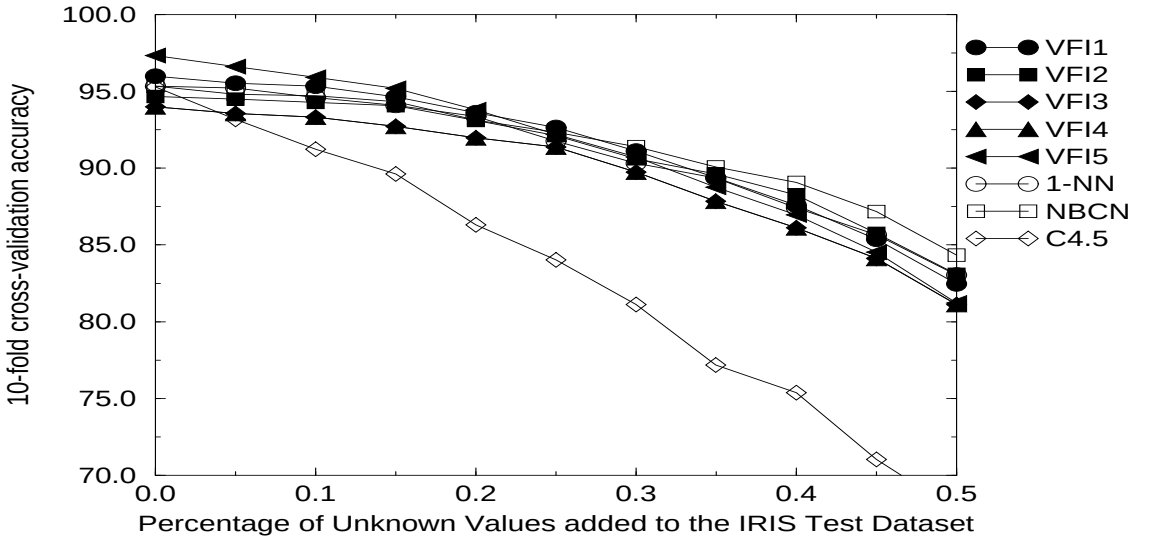

Figure 5.14. 10-fold cross-validation accuracy results of the VFI algorithms compared with those of 1-NN, NBCN, and C4.5 algorithms on Iris dataset with increasing level of unknown values in test data.

loses the most in accuracy among five versions, but the difference is insignificant. Similarly, the other feature projection based methods lose little accuracy with the addition of unknown feature values to the test dataset with the only exception of the FI4 algorithm. The FI4 algorithm experiences a sharp degrade with the addition of unknown values to the test dataset.

Figure 5.14 shows the comparison of the VFI classifiers with other well-known classifiers such as C4.5, NBCN, and 1-NN. As shown in the previous graph, the accuracy of th VFI classifiers decrease a little with the addition of the unknown values to the test instances. Similarly, the 1-NN and NBCN algorithms lose some accuracy with this addition. The algorithm which is affected most negatively is the C4.5 algorithm, which experiences a sharp degrade with the addition of unknown values to the test instances. This shows that the method of handling unknown values in the test instances of C4.5 is not successful. On the other hand, simply ignoring that unknown value as done in all feature projection based methods and NBCN is more successful.

5.3 Discussion

The experimental results on real-world datasets show that VFI classifiers are almost always achieve the highest classification accuracies among all the other feature projection based methods. Another result from empirical evaluation on real-world datasets is that VFI performs comparably on the average and even better on most datasets than NBCN, but the well-known algorithms such as 1-NN and C4.5 often achieve higher accuracies than VFI classifiers. However, VFI algorithms have a significant speed advantage in classification over 1-NN and also classifies a little faster than NBCN. Moreover, the classification accuracy of the VFI classifiers can be improved by learning feature weights described in the next chapter.

The experiments on artificially generated datasets have shown that the VFI classifiers are very robust to the presence of irrelevant features whereas all other feature projection based methods and the nearest neighbor algorithm lose classification accuracy with the addition of irrelevant features. However,

the VFI algorithms are affected negatively in the presence of noisy feature values. These experiments on noisy datasets have also shown that VFI2 is the noise-tolerant version of the VFI classifiers. Lastly, the effect of unknown values that might exist both in training and test datasets is investigated. The experimental results have shown that VFI classifiers as well as the other feature projection based methods are not affected with the addition of the unknown values to the training instances whereas the classification accuracy of the 1-NN algorithm decreases in the presence of unknown values. This has shown that the method of handling unknown feature values in the training dataset used by the feature projection based methods is better than that of the 1-NN algorithm. When unknown values are present in the new instances to be classified, the classification accuracy of VFI classifiers decrease a little whereas the C4.5 algorithm experiences a sharp degrade in accuracy. This has shown that the method of handling unknown feature values in the new instances used by the feature projection based learning methods is better than that of C4.5.

Chapter 6

Learning Feature Weights

All the classification algorithms mentioned up to this point assume that the features representing a domain are equally relevant. But in many real-world problems the features might have different degrees of relevance ranging from being totally relevant to being totally irrelevant in representing the concept. The k-NN algorithm presented in Chapter 2, the k-NNFP, CFP, COFI, and FIL algorithms presented in Chapter 3 and the versions of the VFI algorithms presented in Chapter 4 assumed that all the features are equally relevant. However, they are all suitable to use the feature relevance weight information in addition to the training examples. The aim of using feature weights is to reduce the impact of irrelevant and weakly relevant features and to increase the impact of the strongly relevant features in learning the concept description of a given domain.

Feature weight learning methods are organized along 5 dimensions [76]. The first dimension is called as *bias*, which refers to whether the weight learning algorithm receives feedback from the learning algorithm (i.e., the classifier) or not. This bias dimension is analyzed in a separate paper by Wettschereck and Aha [75]. *Performance* bias weight learning methods use performance feedback from the classifier during learning. They have an advantage: their search for feature weight settings is guided by how well those settings perform. *Preset* bias methods do not use feedback from the classifier to assign weight settings. Instead, they use a pre-existing model's bias. The distinction

between performance and preset bias may also be described as *wrapper* and *filter* models [38]. One group of performance bias methods, called *incremental hill-climbers* [75], modify feature weights incrementally to increase similarity between a test instance and nearby training instances in the same class, and to decrease its similarity with nearby training instances in other classes. IB4 [3], EACH's weighting method [63], and CFP's weighting method [32] are examples of incremental hill-climbers. The other group of performance bias methods, called *continuous optimizers* [75], iteratively update feature weights using only training instances. GA-WKNN [40], GA-CFP [31], and $k - NN_{VSM}$ [74] are examples of continuous optimizers.

The second dimension of the framework of weighting methods is the size of *weight space*, which is used to distinguish *feature weighting* from *feature selection* algorithms. In fact feature selection algorithms are a proper subset of feature weighting algorithms that employ binary weights (i.e., 0 or 1), meaning that feature is either totally relevant (1) or totally irrelevant (0). The third dimension is the *representation*, which is used to distinguish algorithms that use the given representation from those that transform the given representation into one that might yield better performance. The fourth dimension is the *generality*, which refers to whether the learning algorithm learns settings for a single set of weights that are employed globally (i.e., over the entire instance space) or assume weights that differ among local regions of the instance space. The last dimension is *knowledge*, which distinguishes knowledge-poor weight learning algorithms from others that employ domain specific knowledge to set feature weights. These five dimensions are used to make up a framework for feature weight learning methods [76]. Several variants of k-NN using feature weights in the distance function have been proposed, which are classified according to this framework. The feature projection based techniques can use all of the weight learning methods that are used for the k-NN algorithm.

In this thesis, we introduce a performance bias weight learning method according to the bias dimension of the framework introduced by Wettschereck et al. [76]. This new weight learning method uses genetic algorithms to learn feature weights, and thus is a continuous optimizer. According to the weight

space dimension, it is a feature weighting algorithm since the weights of features are not limited to only binary weights. If we classify according to the representation dimension, it is a weight learning method that uses the given representation. According to the generality dimension, it is a global weight learning algorithm because the weights are learned for the entire instance space. Learning weights by genetic algorithms is a knowledge-poor method since no domain specific knowledge is used. Four genetic algorithms are designed to be used for any given classifier and used for optimizing the classification accuracy for the Nearest Neighbor [21] and the VFI1 classifier.

Some other performance bias methods also using genetic algorithms are GA-WKNN [40] and GA-CFP [31]. GA-WKNN uses many genetic operators to learn the feature weights for the Weighted Nearest Neighbor Algorithm (WKNN), where k was chosen as 3. GA-CFP uses genetic algorithms to learn feature weights and some other parameters for the CFP classifier [32].

The next section gives a brief introduction to genetic algorithms, Section 6.2 describes how genetic algorithms are used in weight learning for classifiers. Then the experimental results for 1-NN and VFI classifiers using weight learning genetic algorithms are given in Section 6.3.

6.1 Genetic Algorithms

Genetic algorithms are search and optimization algorithms based on natural selection and natural genetics. They have been introduced by John Holland [35]. Genetic algorithms combine survival of the fittest (best) among a population of strings (chromosomes) with a structured yet randomized information exchange to form a search algorithm. In every generation, a new set of artificial creatures represented by chromosomes is created from the previous population of creatures selected according to the survival of the fittest principal. Although the genetic algorithms are randomized search algorithms, they efficiently exploit historical information to reach new search points with expected improved performance.

```
genetic-algorithm ():
begin
    step = 0
    initialize population P(step)
    /* each chromosome is a coding of the parameter set */
    evaluate (P(step))
    repeat until termination condition
        step = step + 1
        reproduction (P(step))
        crossover (P(step))
        mutation (P(step))
        elitism (P(step))
        evaluate (P(step))
end.
```

Figure 6.1. The algorithm for a genetic algorithm.

Genetic algorithms can be used to optimize some parameters to be used in a system. The search for the optimum parameters start with a randomly generated population of chromosomes which is a coding of the parameter set and continues by generating new populations from the old ones. Genetic algorithms (GAs) are different from more traditional optimization and search procedures in four ways [20]:

- GAs work with a coding of the parameter set, not the parameters themselves.
- GAs search from a population of points, not a single point.
- GAs use a payoff (objective function) information, not derivatives or other auxiliary knowledge.
- GAs use probabilistic transition rules, not deterministic rules.

A general outline of a genetic algorithm is given in Figure 6.1. The algorithm starts with an initial population of chromosomes each of which is a coding for a setting of the parameter set. As mentioned earlier GAs search

from a population of points, so there must be a set of operations that take this initial population and generate successive populations that improve over time. After the current population is evaluated according to the given performance measure, the genetic operators are applied to the individuals of the current population. The first three operators —reproduction, crossover, and mutation— are the main operators of a genetic algorithm that is used in many practical problems.

Reproduction is a process where individual chromosomes are copied to the next generation according to their objective function (fitness function) values. We can think of the objective function as some measure of profit, utility, or goodness that we want to maximize. Copying chromosomes according to their fitness values means that chromosomes with a higher value have a higher probability of contributing one or more offspring in the next generation. Reproduction operator is an artificial version of *natural selection*, a Darwinian survival of the fittest among string creatures. One of the algorithmic forms of the reproduction operator is a biased roulette wheel where each current chromosome in the population has a roulette wheel slot sized in proportion to its fitness. A simple spin of the weighted roulette wheel yields the reproduction candidate offspring. This enables the more highly fit chromosomes to have a higher number of offsprings in the succeeding generation. Once a chromosome has been selected for reproduction, an exact copy of that chromosome is created. Then this chromosome enters into a mating pool for crossover operator.

After reproduction, the newly produced chromosomes in the mating pool are mated at random with each other. Each selected pair of chromosomes undergoes a crossover operation. There are several crossover operations, four of which are used in this thesis are as follows:

1. **One-Point Crossover (1PCO):**
   One-point crossover is the simplest crossover operator, where two offsprings are produced from two parent chromosomes. Given two parent chromosomes of length l, an integer position k (crossover site) along the chromosome is selected uniformly at random between 2 and $l-1$ inclusive. Two new chromosomes are created by swapping all genes between

positions $k+1$ and l. The algorithm for 1PCO is given in Figure 6.2.

2. **Two-Point Crossover (2PCO):**
   Two-point crossover produces two offsprings from two parent chromosomes. There are two crossover sites instead of one as in the case of one-point crossover. First, an integer position k_1 (first crossover site) is selected randomly between 2 and $l-1$ (from the range $[2\ ..\ l-1]$). Then another integer position k_2 (second crossover site) is selected randomly between k_1+1 and l (from the range $[k_1+1\ ..\ l]$). Two new chromosomes are created by swapping all genes between positions k_1 and k_2 (in the range $[k_1\ ..\ k_2)$). Two-Point Crossover is a special case of *multiple-point crossover*, where we can have more than two crossover sites. The algorithm for 2PCO is given in Figure 6.2.

3. **Uniform Crossover (UCO):**
   Uniform crossover produces two offsprings from two parent chromosomes. The corresponding genes of parent chromosomes are swapped with some probability, p_u. Two probabilities which sum up to 1 are symmetric and $p_u = 0.5$ causes the maximum exchange of genes between parent chromosomes. The algorithm for UCO is also given in Figure 6.2.

4. **Continuous Uniform Crossover (CUCO):**
   Continuous Uniform Crossover is a new crossover operator developed and used in the weight learning genetic algorithm. Given two chromosomes $x =< x_1, x_2, \ldots, x_n >$ and $y =< y_1, y_2, \ldots, y_n >$ such that n is the number of genes in a chromosome and is equal the number of features when a chromosome is the encoding of a feature weight vector, the offsprings are defined as
   $x' =< x'_1, x'_2, \ldots, x'_n >$ and $y' =< y'_1, y'_2, \ldots, y'_n >$, where

$$x'_i = s \times x_i + (1-s) \times y_i \tag{6.1}$$

$$y'_i = s \times y_i + (1-s) \times x_i \tag{6.2}$$

   Here s, called *stride*, is constant through a single crossover operation. Given that $\sum_{i=1}^{n} x_i = 1$ and $\sum_{i=1}^{n} y_i = 1$,

```
crossover (Population): /* 1-point crossover */
begin
    for each chromosome pair c and c + 1 in Population
        if crossover is possible /* with crossover probability, p_c */
            cross_point = a random number in [2 .. NoGenes)
            switch alleles up to cross_point of c and c + 1

end.
```

```
crossover (Population): /* 2-point crossover */
begin
    for each chromosome pair c and c + 1 in Population
        if crossover is possible /* crossover probability, p_c */
            cross_point_1 = a random number in [2 .. NoGenes)
            cross_point_2 = a random number in (cross_point_1 .. NoGenes]
            switch alleles of c and c + 1 between cross_point_1 and cross_point_2

end.
```

```
crossover (Population): /* uniform crossover */
begin
    for each chromosome pair c and c + 1 in Population
        if crossover is possible /* with crossover probability, p_c */
            for each allele pair c[i] and c + 1[i]
                if switch is possible /* switch probability, p_u */
                    switch c[i] and c + 1[i]

end.
```

Figure 6.2. Algorithms for One-Point Crossover, Two-Point Crossover, and Uniform Crossover.

$$\sum_{i=1}^{n} x'_i = s \times \sum_{i=1}^{n} x_i + (1-s) \times \sum_{i=1}^{n} y_i = s + (1-s) = 1$$

and the same equality holds for $\sum_{i=1}^{n} y'_i$. So it is guaranteed that the sum of the alleles of an offspring is still 1 given that the sum of the parents' alleles is 1.

Since x'_1 and y'_1 represent the weight of individual features, $0 \leq x'_1 \leq 1$ and $0 \leq y'_1 \leq 1$ must hold. Therefore, the choice of the *stride* s should be restricted. Since, for any value of s, the sum of the alleles is 1, it will be guaranteed that each allele would be less than 1 as long as each allele is ensured to be greater than 0. In order to have $x'_i \geq 0$ and $y'_i \geq 0$ for all i $(1 \leq i \leq n)$,

$$\frac{-x_i}{y_i - x_i} \leq s \leq \frac{y_i}{y_i - x_i}$$

where $y_i \geq x_i$. Each allele pair (x_i, y_i) brings an upper bound, call it $upper_i$ and a lower bound, call it $lower_i$, on s where

$$upper_i = \frac{y_i}{y_i - x_i} \geq 1 \quad \text{and} \quad lower_i = \frac{-x_i}{y_i - x_i} \leq 0$$

The stride s should be in the range $[lower \,..\, upper]$ to preserve the legality of the offsprings where $lower$ and $upper$ are chosen from n $upper_i$ and $lower_i$ bounds such that

$$upper = min_i(\frac{y_i}{y_i - x_i}) \quad \text{and} \quad lower = max_i(\frac{-x_i}{y_i - x_i})$$

Two strides, s_1 and s_2, are symmetric when $s_1 + s_2 = 1$. Here symmetry means that continuous uniform crossover on two given parents produces same pair of offsprings in both use s_1 and s_2. For example, the strides $s_1 = 0.5$ and $s_2 = 0.5$, $s_1 = -2$ and $s_2 = 3$ are symmetric. There is always a symmetric stride value for each value of strides. Further, the symmetric stride of a given stride greater (less) than 0.5 is less (greater) than 0.5. So we can discard the stride values less than 0.5 in the global range for s, since there is a stride in the range $[0.5 \,..\, upper]$ symmetric to the stride in the range $[lower \,..\, 0.5]$.

If we think of some special values of s, we see that when $s = 1$ or $s = 0$, the offsprings are same as their parents. When $s = 0.5$, the alleles of

the offsprings are the average of the alleles of their parents. When $s > 1$ or $s < 0$, the alleles of the offsprings are greater than the maximum of the corresponding allele pair of their parents and less than the minimum of the corresponding allele pair of their parents. Hence, the stride being greater than 1 or less than 0 enables the crossover operation to try the outer values of the alleles of the parents. Because of the symmetry we have discarded the strides less than 0.5, having a range $[0.5 \,..\, upper]$ for s. When $upper$ value is greater than 1, we restrict s to be in the range $[1 \,..\, upper]$ in order to try the outer values of the parent alleles. But when upper bound is equal to 1, s is restricted to be in the range $[0.5 \,..\, 1]$.

Let us see an example CUCO operation. Let the parent chromosomes be $x =< 0.4, 0.6 >$, and $y =< 0.5, 0.5 >$. The first allele pair ($x_1 = 0.4$, $y_1 = 0.5$) requires $-4 \leq s \leq 5$, and the second allele pair ($y_2 = 0.6$, $x_2 = 0.5$) requires $-5 \leq s \leq 6$. From the lower bounds we choose the maximum (i.e. -4) and from the upper bounds we choose the minimum (i.e. 5). We get the requirement for s to be in the range $[-4 \,..\, 5]$. By using the symmetry property, s is restricted to be in the range $[0.5 \,..\, 5]$. Since $upper > 1$, s is randomly selected from the range $[1 \,..\, 5]$.

Let the randomly chosen value of s be 3:

$$x'_1 = 3 \times x_1 + (1 - 3) \times y_1 = 1.2 - 1.0 = 0.2 \tag{6.3}$$

$$x'_2 = 3 \times x_2 + (1 - 3) \times y_2 = 1.8 - 1.0 = 0.8 \tag{6.4}$$

Hence, one of the offsprings is $x' =< 0.2, 0.8 >$ where $\sum_{i=1}^{2} x'_i = 1$. The other offspring alleles are as follows:

$$y'_1 = 3 \times y_1 + (1 - 3) \times x_1 = 1.5 - 0.8 = 0.7 \tag{6.5}$$

$$y'_2 = 3 \times y_2 + (1 - 3) \times x_2 = 1.5 - 1.2 = 0.3 \tag{6.6}$$

Hence, the other offspring is $y' =< 0.7, 0.3 >$ where $\sum_{i=1}^{2} y'_i = 1$. Note that the symmetric stride $s = -2$ would give the offsprings
$x' =< 0.7, 0.3 >$ and $y' =< 0.2, 0.8 >$ which are the same as the offsprings produced by using $s = 3$.

The difference of CUCO from other crossover operators is its ability to preserve the legality of the chromosomes after the crossover operation.

The "legality" here means that the sum of the alleles of each chromosome is 1. Hence, given two parent chromosomes which have the sum of their alleles 1, the sum of the alleles of each offsprings is also 1 without any other operation. On the other hand, all other crossover operators defined above require a normalization to make the sum of the alleles of each offspring 1.

Reproduction according to fitness combined with crossover is enough for the process of a genetic algorithm. Mutation plays a secondary role in the operation of genetic algorithms, but mutation operator protects against the loss of important genetic material. It is usually in the form of a small change in the alleles of an offspring. In our genetic algorithms, the mutation is performed by switching two randomly selected neighboring alleles, which does not cause a change in the sum of the alleles of a chromosome. The probability of mutation (p_m) of an offspring is very low in general.

Other than the three main operations of a genetic algorithm, there is another operator in Figure 6.1 called the elitism operation, which always copies the fittest chromosome in the current population to the next population. This provides the best chromosome not to be lost once it is found.

Genetic algorithms start with a randomly generated population of chromosomes representing the parameter set. They generate a new population from the current one by using the genetic operators such as reproduction, crossover, mutation, and elitism. They either stop when the fitness of a chromosome in the current population reached the best possible fitness or when the maximum number of generations set by the user is reached. The aim of this search is to find the best parameter set optimizing a given system.

In the next section, the use of genetic algorithms in learning feature weights for any given classifier is briefly explained. Then, the experimental results are given for the weighted nearest neighbor and the VFI1 classifiers.

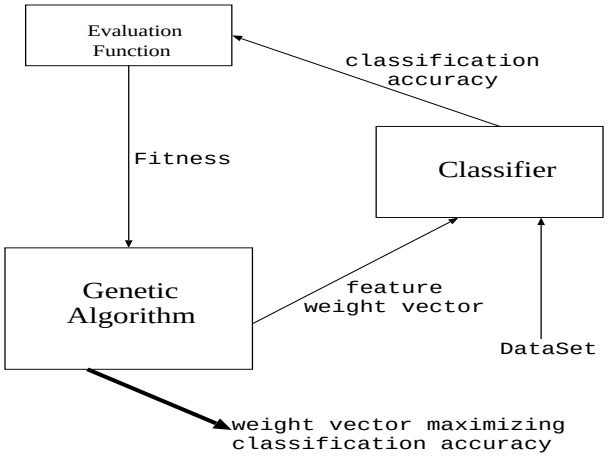

Figure 6.3. The GA-*Classifier* Feature Weighting Algorithm.

6.2 Weight Learning Genetic Algorithms

Genetic algorithms can be used to maximize the classification accuracy of a given classifier, which can improve its performance when feature weights are used to assign different degrees of relevance to the features. The implementation of GA-*Classifier* is generic and can be used for any classifier that can make use of feature weights.

Figure 6.3 shows how our weight learning genetic algorithm (GA-*Classifier*) works to maximize the performance of the *Classifier*. The genetic algorithm starts with an initial population of chromosomes each representing a feature weights setting. Feature weights are represented by real numbers between 0 and 1 inclusive such that the sum of all feature weights is 1. Since we are searching for the optimum set of weights, a chromosome is the coding of these feature weights, i.e. each gene corresponds to a feature weight. Hence, the length of a chromosome is equal to the number of features of a given domain.

In order to continue, GA requires to know how fit each chromosome is. The Classifier takes the dataset and the feature weights encoded in the chromosome as input, and its classification accuracy using these weights is used as the evaluation of that chromosome. The classification accuracy is measured by 5-fold cross-validation (see Figure 5.1) and the fitness of that weight vector is the

cube of this classification accuracy. Taking the cube of the performance measure aims to enlarge the difference of chromosomes having very close evaluation function values. Each chromosome in the current population is evaluated by the Classifier and the genetic algorithm produces a new population by using these fitness values for the roulette wheel reproduction operation. Other genetic operators used are crossover, mutation, and elitism. When a chromosome is selected for mutation, two randomly chosen genes of that chromosome are swapped. This kind of mutation preserves the legality of the chromosomes. The GA stops either when the classification accuracy of %100 is reached or after a predetermined number of generations are generated. At the end, the feature weight vector maximizing the classification accuracy of the Classifier is learned.

6.3 Experiments

The GA-*Classifier* can be used to learn feature weights for any classifier that can embed feature weights in its learning process. We did experiments for the Weighted Nearest Neighbor (WNN) classifier and Weighted VFI (WVFI) classifiers. In each run of the genetic algorithms a population of size 100 was used. The chromosomes are feature weight vectors such that the sum of the weights is 1. The genetic algorithms terminated after 200 generations. The probability of crossover (p_c) was set as 0.8.

6.3.1 Weighted Nearest Neighbor Classifier

When we put the WNN classifier in the place of Classifier in Figure 6.3, we get a GA-WNN classifier which learns its weights by a genetic algorithm. This genetic algorithm aims to learn the optimum weights that would maximize the classification accuracy of the WNN classifier. Besides learning weights, we wanted to compare each of the four crossover operators by comparing the performance of the following WNN classifiers:

Table 6.1. Classification accuracy(%) of NN, 1PCO-WNN, 2PCO-WNN, UCO-WNN, and CUCO-WNN obtained by 5 way cross-validation on four real-world datasets.

Data Set:	Iris	Glass	Wine	Liver
NN	93.98	68.66	94.40	63.48
1PCO-WNN	95.34	85.96	99.44	71.90
2PCO-WNN	96.00	84.10	99.44	69.30
UCO-WNN	95.34	85.50	100.00	68.42
CUCO-WNN	97.34	86.86	98.86	72.20

1. WNN learning feature weights using a genetic algorithm which uses one-point crossover (1PCO-WNN)

2. WNN learning feature weights using a genetic algorithm which uses two-point crossover (2PCO-WNN)

3. WNN learning feature weights using a genetic algorithm which uses uniform crossover (UCO-WNN)

4. WNN learning feature weights using a genetic algorithm which uses continuous uniform crossover (CUCO-WNN)

When this work [21] has done, the probability of mutation was set as 0 in order to observe the capabilities of crossovers without mutation. The fitness of a chromosome is determined by 5-fold cross-validation. The experiments have been done on four real-world datasets (see Appendix for more information about the datasets). The classification accuracies of NN and the four weighted versions are shown in Table 6.1.

The accuracies in Table 6.1 show that in all of the four datasets used, weighted versions of the nearest neighbor algorithm outperforms unweighted version of the nearest neighbor algorithm. These results indicate that assigning different weights to features in all these domains improves the classification accuracy of the NN classifier. Another important observation from the experiments is that CUCO-WNN generally has higher accuracies than other three weighted nearest neighbor algorithms.

In the Iris domain CUCO-WNN has the highest classification accuracy, and UCO-WNN and 1PCO-WNN have the worst accuracy. In the Glass domain CUCO-WNN again has the highest classification accuracy, and 2PCO-WNN has the worst accuracy. The accuracy improvement gained by assigning weights to features is very significant in the Glass domain where the smallest improvement is 15.44% with the learned weights $\langle 0.375, 0.083, 0.12, 0.007, 0.018, 0.207, 0.186, 0, 0.004\rangle$ respectively. In the Liver domain, CUCO-WNN again has the highest classification accuracy, and UCO-WNN has the lowest classification accuracy. Only in the Wine domain we observed that UCO-WNN has the highest classification accuracy, 1PCO-WNN and 2PCO-WNN follow it, and CUCO-WNN has the lowest classification accuracy; however the differences in the accuracies are insignificant. In Figure 6.4 the comparison of the four algorithms on four real-world datasets for increasing number of generations is shown.

Experiments have shown that CUCO-WNN generally outperforms other three weighted nearest neighbor algorithms because CUCO-WNN learns the best feature weights by which the highest classification accuracy is obtained. However, what we also have observed is that the classification accuracies of these four weighted nearest neighbor algorithms are close to each other.

6.3.2 Weighted Voting Feature Intervals Classifiers

When we put one of the VFI classifiers in the place of Classifier in Figure 6.3, we get a GA-VFI classifier which learns feature weights by a genetic algorithm. This genetic algorithm aims to learn the optimum weights that would maximize the classification accuracy of the VFI classifier.

In this section, the comparisons of the unweighted VFI1 classifier, where all the features have equal voting power with a weighted GA-VFI1 classifier where each feature might have a different voting power are given. The aim of the experiments were to show how weights can improve the performance of the VFI classifiers and not to compare the crossover operators at the same time as done in the previous section. Therefore, we use only the continuous crossover operator in our genetic algorithm that is used to learn the optimum weights

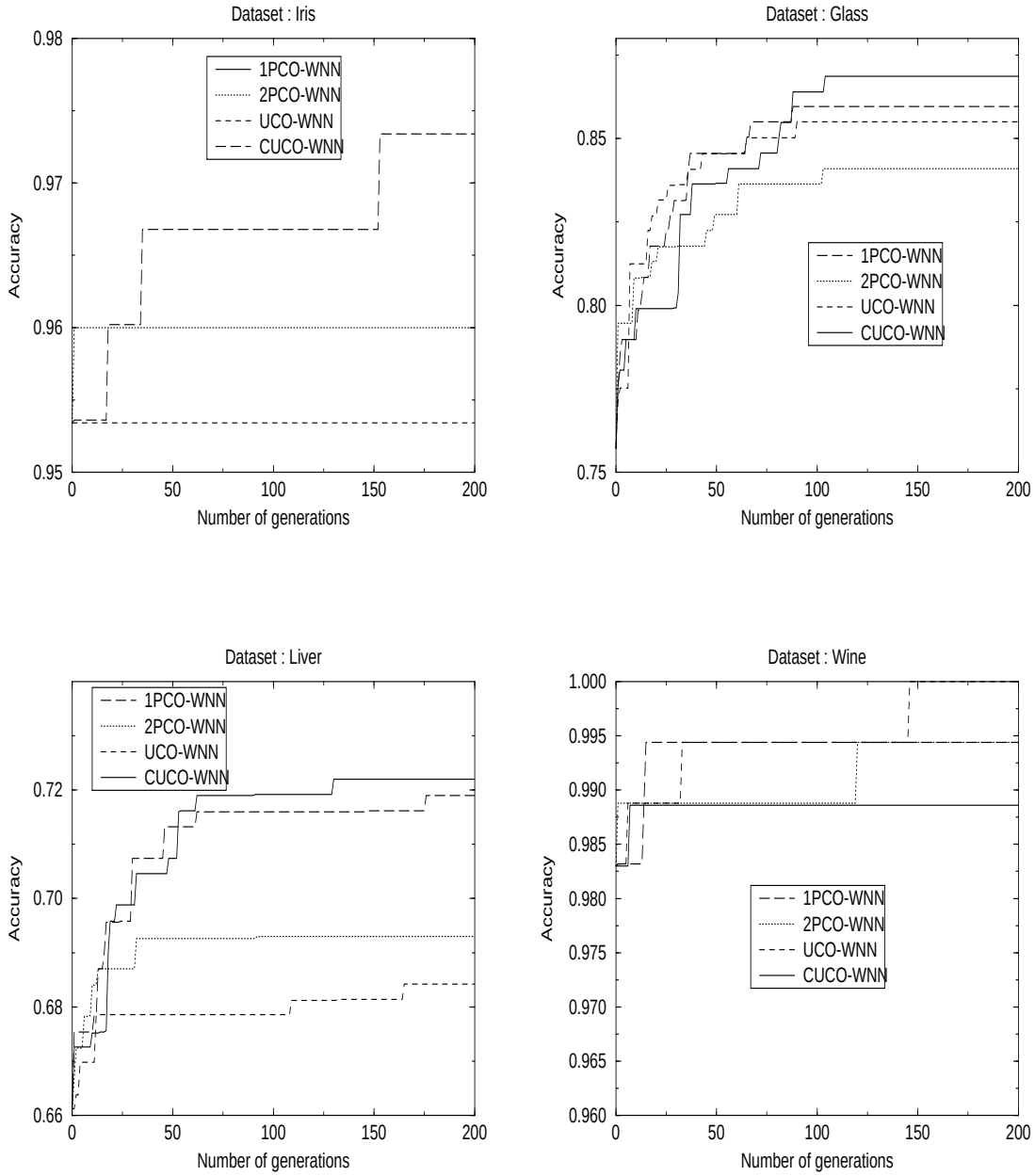

Figure 6.4. Comparison of 1PCO-WNN, 2PCO-WNN, UCO-WNN, and CUCO-WNN on real-world datasets for increasing number of generations. The accuracy results are obtained by 5-fold cross-validation.

Table 6.2. Classification accuracy(%) of VFI1, CUCO-WVFI1 obtained by 5-fold cross-validation on six real-world datasets.

Data Set:	Diabetes	Glass	Iris	Liver	Sonar	Wine
VFI1	56.51	57.48	95.33	55.65	52.86	95.52
CUCO-WVFI1	66.93	63.56	96.00	63.19	71.23	98.89
Improvement:	10.42	6.08	0.67	7.54	18.37	3.37

for VFI1. Unlike CUCO-WNN, the probability of mutation was set to 0.001 in these experiments. The resulting weighted classifier is called CUCO-WVFI1 and the 5-fold cross-validation results of both VFI1 and CUCO-VFI1 on six datasets is shown in Table 6.2.

The classification accuracies in Table 6.2 show that in all of the six datasets used, weighted version of the VFI1 algorithm outperforms unweighted version of the VFI1 algorithm. These results indicate that assigning different weights, that is different voting powers to features in all these domains improves the classification accuracy of the VFI1 classifier. For example, the Diabetes and Sonar datasets were the datasets on which VFI classifiers achieve lower accuracies than other well-known classifiers as reported in Chapter 5 and CUCO-WVFI1 (weighted version of VFI1) performs significantly better than VFI1. The improvement in accuracy means that all the features are not equally relevant in these domains. For example, the weights learned for the Diabetes dataset are $\langle 0.008, 0.375, 0.198, 0, 0.0330.340, 0.032, 0.014 \rangle$ where fourth feature has found totally irrelevant, first and last features are nearly irrelevant, and second and third features have found to be the most relevant features by the weight learning genetic algorithm.

6.4 Summary and Discussion

We have presented a feature weight learning method using genetic algorithms, which learns the optimum feature weights for any classifier that can get weights outside in order to maximize the classification accuracy of that classifier. The experiments have shown that learning feature weights for both the Nearest

Neighbor and the VFI algorithms have improved their classification accuracies on some real-world datasets.

We have also developed a new crossover operator (CUCO) to be used in these weight learning genetic algorithms and compared it with three other common crossover operators by using them in GA-WNN. Experiments have shown that CUCO is generally better than the other crossover operators. However, the classification accuracies of GA-WNN's using different crossover operators are close to each other.

Genetic algorithms are appropriate for feature wieght learning tasks, but they are slow. They get slower on datasets with large number of instances and features. For example, the time for GA-WVFI1 to proceed one step is approximately 4 minutes on the Iris dataset, whereas this time increases to 40 minutes on the Arrhythmia dataset with 279 features.

Chapter 7

Visualization of the Learned Concepts

The explanation ability of a classification process is as much important as its accuracy. We have shown the empirical evaluation of VFI classifiers in Chapter 5 on several real-world datasets including the newly constructed Dermatology and Arrhythmia datasets. But the high classification performance is not enough for a classification system, it should also convey some comprehensible information to humans. For this purpose, we tried to visualize the concept description learned by VFI classifiers. Since each feature votes for each class during classification, these votes make up the concept description and gives information about the relation between the values of each feature and the class label observed at that value.

The concept description learned by VFI1, VFI2, VFI3, VFI4, and VFI5 for the Dermatology dataset is shown in Figures 7.1, 7.2, 7.3, 7.4, and 7.5, respectively. For space efficiency, only a few interesting features are shown in all figures. At the top of Figure 7.1, a general information about the dataset is given. Then the intervals with their votes for each class are displayed, where class numbers in rectangular brackets are used for the class names of the domain (see Appendix A). To the right of some of the votes a (+) or (–) or nothing

meaning (o) is given, which results from the following mapping of the real-valued votes to discrete evaluations:

$$s(vote) = \begin{cases} \text{if } vote = highest \text{ and } vote - next > d, \text{then} & (+) \\ \text{else if } vote < \frac{d}{2}, \text{then} & (-) \\ \text{else} & (o) \end{cases} \quad (7.1)$$

where $next$ is the next highest vote after $vote$ in that interval and $d = \frac{1}{No\_Classes}$. The aim of this mapping was to see the ability of the features in distinguishing between classes. When a feature value makes a class (+), it means that the instance is certainly of this class. Note that, at most one class can get a (+) evaluation. A (–) class means that the instance is certainly not of this class according to feature 1 (*erythema*) and a (o) means that this feature can not say anything about the class. Unlike (+) category, more than one class can get (–) and (o).

In Figure 7.1, for the first feature, four intervals and the boundary points of these intervals are shown with their votes. Since we know that features of the Dermatology dataset take values 0, 1, 2, or 3, there will be no instance with value less than 0 for the first feature. But since we wanted our visualization to be general, those intervals are also shown to the user. When we look at the upper end point of the first interval, which is 0, we see the votes $< 0.11, 0.04, 0.15, 0.10, 0.55, 0.05 >$ for each corresponding class. This shows that feature 1 (*erythema*) votes nearly half for class 5 (*cronic dermatitis*), one-tenth for class 1 (*psoriasis*) and 4 (*pityriasis rosea*), and votes few for other classes. If there were a threshold of 0.15 for votes, similar to the case of Turkish Parliament Elections Voting Scheme, only class 3 and 5 would be over this threshold. But in the voting scheme of VFI classifiers, there are no thresholds and every single vote participates in the overall voting process.

Being the designers of these classifiers, these real-valued votes were understandable for us. But thinking of the human experts (the doctors) who collected these data for us, we thought we should transform this representation into a discrete language consisting of (+): positive, (o): neutral, (–): negative. When the value of feature 1 is equal to 0, class 5 gets a (+) in the new representation, class 2 and 6 (–), and other classes (o). Note that the distinguishing labels (+) and (–) are shown whereas the (o) labels are omitted in Figure 7.1.

```
               Domain: Dermatology,
   No_Features: 34    No_Classes: 6    No_Trainers: 366
Weights: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      classCount[1]=112  classCount[2]=61  classCount[3]=72
      classCount[4]=49   classCount[5]=52  classCount[6]=20

Intervals of Dermatology domain:

Feature 1

   VOTES FOR CLASSES:        [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:   0.15       (-)      0.23       (-)      0.63(+)    (-)
If    value =  0    then:   0.11      0.04(-)   0.15      0.10      0.55(+)   0.05(-)
If  0 < value < 1   then:   0.06(-)   0.07(-)   0.10      0.20      0.47(+)   0.09
If    value =  1    then:   0.11      0.12      0.14      0.19      0.31      0.14
If  1 < value < 3   then:   0.16      0.16      0.17      0.17      0.14      0.18
If    value =  3    then:   0.23      0.22      0.17      0.13      0.10      0.15
If    3  < value    then:   0.29      0.29      0.16      0.10      0.05(-)   0.12

Feature 6

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:   0.20      0.20      0.01(-)   0.20      0.20      0.20
If    value =  0    then:   0.18      0.18      0.12      0.18      0.18      0.18
If  0 < value < 3   then:   0.15      0.15      0.24      0.15      0.15      0.15
If    value =  3    then:   0.08(-)   0.08(-)   0.62(+)   0.08(-)   0.08(-)   0.08(-)
If    3  < value    then:    (-)       (-)      1.00(+)    (-)       (-)       (-)

Feature 7

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:   0.20      0.21      0.21      0.21      0.17       (-)
If    value =  0    then:   0.20      0.20      0.20      0.20      0.18      0.01(-)
If  0 < value < 1   then:   0.20      0.20      0.20      0.20      0.19      0.02(-)
If    value =  1    then:   0.11      0.11      0.10      0.10      0.19      0.39(+)
If  1 < value < 2   then:   0.03(-)   0.02(-)    (-)       (-)      0.19      0.76(+)
If    value =  2    then:   0.02(-)   0.01(-)    (-)       (-)      0.12      0.85(+)
If  2 < value < 3   then:   0.01(-)    (-)       (-)       (-)      0.06(-)   0.93(+)
If    value =  3    then:    (-)       (-)       (-)       (-)      0.03(-)   0.97(+)
If    3  < value    then:    (-)       (-)       (-)       (-)       (-)      1.00(+)
```

Figure 7.1. Concept Description Learned by VFI1 including only a few features.

This means that the value of 0 for feature 1 positively distinguishes class 5 from other classes (i.e. according to feature 1 with value zero, this patient has diagnosis 5), negatively distinguishes class 1 and 2 (i.e. this patient can not have diagnosis 1 or 2), and says neither "yes" nor "no" for the other classes. Not all the intervals distinguish much between classes: for example when the feature 1 has a value between 1 and 3 ($1 < value < 3$), all the classes are neutral (o); that is, this range of values for feature 1 does not distinguish any class from the others. In Figure 7.1, the $value = 0$ of feature 6 does not distinguish any class, similarly the next interval $0 < value < 3$ does. This range of values correspond to the values 0, 1, and 2 for feature 6 and what VFI1 learns from the

```
                Domain: Dermatology,
   No_Features: 34    No_Classes: 6    No_Trainers: 366
Weights: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      classCount[1]=112  classCount[2]=61  classCount[3]=72
      classCount[4]=49   classCount[5]=52  classCount[6]=20

Intervals of Dermatology domain:

Feature 6

         VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If   value <  1.5     then:    0.20      0.20      0.01(-)   0.20      0.20      0.20
If   value =  1.5     then:    0.10      0.10      0.51(+)   0.10      0.10      0.10
If   1.5  < value     then:     (-)       (-)      1.00(+)    (-)       (-)       (-)

Feature 11

         VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If     value =  0     then:    0.14      0.18      0.19      0.19      0.19      0.10
If     value =  1     then:    0.34      0.06(-)   0.02(-)    (-)       (-)      0.59(+)

Feature 15

         VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0.5    then:    0.20      0.20      0.20      0.20       (-)      0.20
If    value =  0.5    then:    0.10      0.10      0.10      0.10      0.50(+)   0.10
If 0.5 < value < 1.5  then:     (-)       (-)       (-)       (-)      1.00(+)    (-)
If    value =  1.5    then:     (-)       (-)      0.03(-)    (-)      0.97(+)    (-)
If 1.5 < value < 2.5  then:     (-)       (-)      0.06(-)    (-)      0.94(+)    (-)
If    value =  2.5    then:     (-)       (-)      0.03(-)    (-)      0.97(+)    (-)
If    2.5  < value    then:     (-)       (-)       (-)       (-)      1.00(+)    (-)
```

Figure 7.2. Concept Description Learned by VFI2 including only a few features.

training instances is that all the classes are possible with these values. VFI1 has learned that when feature 6 has $value = 3$, class 3 can be distinguished positively from the other classes. In Figure 7.1, the concept learned on feature 7 is also shown. Feature 7 significantly distinguishes class 6 at nonzero values.

In Figure 7.2, I have included features 6, 11, and 15 in the concept description learned by VFI2 for the Dermatology dataset. Since the possible values of all the features in the Dermatology dataset can take are 0, 1, 2, 3 and VFI2 constructs intervals with boundaries 0.5, 1.5, 2.5, the visualization of VFI2 is more meaningful for the Dermatology dataset. For example, the interval $0.5 < value < 1.5$ constructed on feature 15 represents the value 2 in the Dermatology dataset because this dataset is compiled such that there are no other values in this range. Feature 11 (*family history*) is a nominal feature and all the intervals constructed are point intervals on this feature. For example, when feature 11 is equal to 1 meaning that some kind of Dermatology disease

has been observed in the patient's family, class 6 is highly confirmed by this feature.

In order to compare the descriptions produced by VFI2 and VFI1, feature 6 is also included in Figure 7.2. Feature 6 confirms class 3 and totally rejects all the other classes when $value > 1.5$; that is, when the value of feature 6 is 2 or 3. When the values are 0 or 1 ($value < 1.5$), feature 6 rejects class 3 and votes equally for the rest of the classes. However, with the concept learned by VFI1 on feature 6 it can only distinguish class 3 positively when the value is 3, for the other values such as 0, 1, and 2 none of the classes are distinguishable. This is because VFI1 loses some information by overgeneralizing the values into a range $0 < value < 3$, which also occurs in VFI3 and later solved by VFI4 (see Section 4.2.4). This will be explained in more detail soon by comparing the concept descriptions of VFI3 and VFI4.

Figure 7.3 shows the concept description learned by VFI3 only on features 6, 10, and 34. Let us look at feature 34, which is the age of the patients. when the age of the patient is between 7 and 16, class 6 (*pityriasis rubra pilaris*) is highly confirmed and when the age is larger than 22 class 6 is rejected. This agrees with what the doctor says about the age feature of a patient. Feature 10 is distinguishing for class 1 whereas feature 6 is totally unuseful to distinguish between classes because every class has equal votes for every range of values. The distribution of the classes on feature 6 in the Dermatology dataset is as follows: the instances of every class other than 3 always have value equal to 0 for feature 6 and almost all of the instances of class 3 have a nonzero value for feature 6 except one or two instances with value 0. Thus, the lowest and highest points on feature 6 of class 3 are 0 and 3 respectively, and both the lowest and highest points of every other class are 0. Thus, the boundaries of the intervals constructed by VFI3 are 0 and 3, since they are the only distinct end points on feature 6. This causes an overgeneralization of values on feature 6 and the information that the instances of every class other than class 3 have value equal to 0 for feature 6 is lost in VFI3. These kind of situations were what motivated us to develop VFI4, which realizes such end points that are both lowest and highest points and constructs point intervals from these points.

The concept descriptions learned by VFI4 on features 6, 10, 13, and 25 is

```
                Domain: Dermatology,
   No_Features: 34    No_Classes: 6    No_Trainers: 366
Weights: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      classCount[1]=112  classCount[2]=61  classCount[3]=72
      classCount[4]=49   classCount[5]=52  classCount[6]=20

Intervals of Dermatology domain:

Feature 6

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If    value =  0    then:    0.17      0.17      0.17      0.17      0.17      0.17
If  0 < value < 3   then:    0.17      0.17      0.17      0.17      0.17      0.17
If    value =  3    then:    0.17      0.17      0.17      0.17      0.17      0.17
If    3  < value    then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 10

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If    value =  0    then:   0.06(-)    0.19      0.20      0.20      0.20      0.15
If  0 < value < 1   then:   0.06(-)    0.19      0.20      0.20      0.20      0.15
If    value =  1    then:    0.29      0.14      0.10      0.10      0.10      0.27
If  1 < value < 2   then:    0.51      0.09       (-)       (-)       (-)      0.39
If    value =  2    then:   0.76(+)   0.05(-)     (-)       (-)       (-)      0.20
If  2 < value < 3   then:   1.00(+)     (-)       (-)       (-)       (-)       (-)
If    value =  3    then:   1.00(+)     (-)       (-)       (-)       (-)       (-)
If    3  < value    then:     (-)       (-)       (-)       (-)       (-)       (-)


Feature 34 (age)

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If    value =  0    then:   1.00(+)     (-)       (-)       (-)       (-)       (-)
If  0 < value < 7   then:   1.00(+)     (-)       (-)       (-)       (-)       (-)
If    value =  7    then:    0.51       (-)       (-)       (-)       (-)      0.49
If  7 < value < 8   then:   0.01(-)     (-)       (-)       (-)       (-)     0.99(+)
If    value =  8    then:   0.04(-)     (-)       (-)       (-)      0.03(-)  0.93(+)
If  8 < value < 10  then:   0.06(-)     (-)       (-)       (-)      0.06(-)  0.88(+)
If    value =  10   then:   0.04(-)    0.09       (-)       (-)      0.03(-)  0.84(+)
If  10 < value < 12 then:   0.02(-)    0.18       (-)       (-)       (-)     0.80(+)
If    value =  12   then:   0.03(-)    0.14       (-)      0.06(-)   0.02(-)  0.75(+)
If  12 < value < 16 then:   0.04(-)    0.10       (-)       0.12     0.04(-)  0.70(+)
If    value =  16   then:    0.09      0.15      0.03(-)    0.17      0.14     0.41(+)
If  16 < value < 22 then:    0.15      0.20      0.07(-)    0.22      0.25      0.11
If    value =  22   then:    0.18      0.20       0.15      0.21      0.21     0.05(-)
If  22 < value < 65 then:    0.20      0.19       0.24      0.20      0.17       (-)
If    value =  65   then:    0.20      0.18       0.12      0.21      0.29       (-)
If  65 < value < 70 then:    0.19      0.18        (-)      0.22     0.41(+)     (-)
If    value =  70   then:   0.60(+)    0.09        (-)      0.11      0.21       (-)
If  70 < value < 75 then:   1.00(+)     (-)        (-)       (-)       (-)       (-)
If    value =  75   then:   1.00(+)     (-)        (-)       (-)       (-)       (-)
If    75  < value   then:     (-)       (-)        (-)       (-)       (-)       (-)
```

Figure 7.3. Concept Description Learned by VFI3 including only a few features.

```
               Domain: Dermatology,
   No_Features: 34    No_Classes: 6    No_Trainers: 366
Weights: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      classCount[1]=112  classCount[2]=61  classCount[3]=72
      classCount[4]=49   classCount[5]=52  classCount[6]=20

Intervals of Dermatology domain:

Feature 6

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  0   then:    0.20      0.20      0.01(-)   0.20      0.20      0.20
If  0 < value < 3   then:     (-)       (-)       1.00(+)   (-)       (-)       (-)
If    value =  3    then:     (-)       (-)       1.00(+)   (-)       (-)       (-)
If    3  < value    then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 10

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If  (*) value =  0  then:    0.04(-)   0.19      0.20      0.21      0.21      0.15
If  0 < value < 1   then:    0.50(+)   0.12      0.14      (-)       (-)       0.24
If    value =  1    then:    0.51(+)   0.11      0.07(-)   (-)       (-)       0.32
If  1 < value < 2   then:    0.51      0.09      (-)       (-)       (-)       0.39
If    value =  2    then:    0.76(+)   0.05(-)   (-)       (-)       (-)       0.20
If  2 < value < 3   then:    1.00(+)   (-)       (-)       (-)       (-)       (-)
If    value =  3    then:    1.00(+)   (-)       (-)       (-)       (-)       (-)
If    3  < value    then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 13

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  0   then:    0.18      0.12      0.16      0.18      0.17      0.19
If  0 < value < 1   then:    0.03(-)   0.39      0.17      0.18      0.23      (-)
If    value =  1    then:    0.04(-)   0.54(+)   0.21      0.09      0.12      (-)
If  1 < value < 2   then:    0.05(-)   0.69(+)   0.25      (-)       (-)       (-)
If    value =  2    then:    0.05(-)   0.69(+)   0.25      (-)       (-)       (-)
If    2  < value    then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 25

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If  (*) value =  0  then:    0.20      0.20      0.01(-)   0.20      0.20      0.19
If  0 < value < 1   then:     (-)       (-)       0.62(+)   (-)       (-)       0.38
If    value =  1    then:     (-)       (-)       0.81(+)   (-)       (-)       0.19
If  1 < value < 3   then:     (-)       (-)       1.00(+)   (-)       (-)       (-)
If    value =  3    then:     (-)       (-)       1.00(+)   (-)       (-)       (-)
If    3  < value    then:     (-)       (-)       (-)       (-)       (-)       (-)
```

Figure 7.4. Concept Description Learned by VFI4 including only a few features.

shown in Figure 7.4. The point intervals constructed on linear features are marked by a (*) like $value = 0$ point interval on feature 6. On this point interval, class 3 is rejected and all other classes are equally voted by feature 6. On all nonzero values of feature 6, class 3 is confirmed and all other classes are rejected by feature 6. This concept is the correct inductive result drawn from the training instances on feature 6, since training instances of classes other than 3 always have value equal to 0 for feature 6 and almost all of the instances of class 3 have nonzero value for feature 6. This situation might frequently happen in a dataset with continuous features getting values from a small set of values such as the Dermatology dataset. VFI4 constructs point intervals also for zero values of feature 10, 13, and 25 as shown in Figure 7.4. Feature 10 is a distinguishing feature for class 1 whereas feature 25 distinguishes class 3 among other classes.

Lastly, Figure 7.5 shows the concept description learned by VFI5 on features 1, 6, 15, and 20. Since boundaries of each interval are point intervals in VFI5, the point intervals between each range interval are shown. For example, every possible value is a point interval on feature 20, so there is no possible value that an instance can take for feature 20 in the range intervals. Therefore, it is more meaningful to visualize the concept description learned for Dermatology dataset by VFI5 than that of the other versions. Looking at these point intervals, we say that nonzero values confirm class 1 and reject all the others and zero value reject class 3 and is indifferent for all other classes. The situation that was lost in VFI1 and VFI3 on feature 6 is also now observable with the point intervals on values 0 and 3. The nonzero values of feature 15 confirm class 5 and rejects all other classes, whereas zero value of feature 15 rejects class 5.

The concept descriptions are learned by classifiers in order to be used in classification of a new instance. The performance of a classifier is measured by the ratio of the number of correctly classified test instances over total number of test instances. What is as much important as the classification accuracy is the explanation ability of the classification process. Does the classifier work like a black box or can it explain why and how it came up with the resulting classification? VFI classifiers can explain why and how the new instance is

```
               Domain: Dermatology,
   No_Features: 34    No_Classes: 6    No_Trainers: 366
Weights: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      classCount[1]=112  classCount[2]=61  classCount[3]=72
      classCount[4]=49   classCount[5]=52  classCount[6]=20

Intervals of Dermatology domain:

Feature 1

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  0   then:    0.15       (-)      0.23       (-)      0.63(+)    (-)
If  0 < value < 1   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  1   then:    0.06(-)   0.08(-)   0.09      0.21      0.46(+)   0.10
If  1 < value < 3   then:    0.16      0.15      0.19      0.18      0.12      0.21
If (*) value =  3   then:    0.29      0.29      0.16      0.10      0.05(-)   0.12
If  3 < value < 0   then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 6

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  0   then:    0.20      0.20      0.01(-)   0.20      0.20      0.20
If  0 < value < 3   then:     (-)       (-)      1.00(+)    (-)       (-)       (-)
If (*) value =  3   then:     (-)       (-)      1.00(+)    (-)       (-)       (-)
If  3 < value < 0   then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 15

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  0   then:    0.20      0.20      0.20      0.20       (-)      0.20
If  0 < value < 1   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  1   then:     (-)       (-)       (-)       (-)      1.00(+)    (-)
If  1 < value < 2   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  2   then:     (-)       (-)      0.06(-)    (-)      0.94(+)    (-)
If  2 < value < 3   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  3   then:     (-)       (-)       (-)       (-)      1.00(+)    (-)
If  3 < value < 0   then:     (-)       (-)       (-)       (-)       (-)       (-)

Feature 20

       VOTES FOR CLASSES:     [1]       [2]       [3]       [4]       [5]       [6]
If    value <  0    then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  0   then:    0.01(-)   0.21      0.21      0.21      0.19      0.18
If  0 < value < 1   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  1   then:    0.49       (-)       (-)       (-)      0.14      0.37
If  1 < value < 2   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  2   then:    0.97(+)    (-)       (-)       (-)      0.03(-)    (-)
If  2 < value < 3   then:     (-)       (-)       (-)       (-)       (-)       (-)
If (*) value =  3   then:    1.00(+)    (-)       (-)       (-)       (-)       (-)
If  3 < value < 0   then:     (-)       (-)       (-)       (-)       (-)       (-)
```

Figure 7.5. Concept Description Learned by VFI5 including only a few features.

classified as the predicted class in terms of the individual votes of each feature given for that class. Looking at these individual votes of each feature, with what level of confidence that feature confirms (high votes) or rejects (low votes) the final prediction is obvious.

An example classification of a new instance (patient) drawn from the Dermatology domain is given in Figure 7.6. When these comparisons were done we have used 250 training instances to learn the concept descriptions. In Figure 7.6, first the feature values of the instance (properties of the patient such as the age of this patient is 34) and then the individual votes of each feature distributed among classes is shown. These votes are then summed up to get the total vote vector, from which the class with the highest vote is predicted as the class of the new instance. The VFI1 classifier predicts class 1 for this instance, which was the same as the human expert's diagnosis. This is a very confident prediction for VFI1, because the next highest vote is nearly the half of the vote received by the predicted class. The individual votes for class 1 are either (+) or neutral except feature 14, moreover the (+) votes almost always appear for class 1 and there is only one (+) received by class 3 from one feature. This table of votes shown in Figure 7.6 is a very good explanation for the classification performed in the sense that everything is open to the user. For example, feature 20 (*clubbing of the rete ridges*) gives a vote of 0.98 for class 1 (note that votes are normalized such that the sum of votes for each class is 1.0), meaning that feature 20 says that this instance must be of class 1 and reflects its individual confirmation in the total vote. At the same time, feature 20 rejects all other classes (all other classes are (–)), meaning that this instance can not be of those classes other than class 1. Feature 34 (age) with value equal to 34 is negative for *pityriasis rubra pilaris* (class 6) and neutral for all other classes. Looking only at this feature does not say anything about the class of the instance, but still it does not reject the first class.

The classification of the VFI1 classifier may not be that much confident for all the time. Let us look at another example classification in Figure 7.7. The feature values, the individual votes of features, and the total votes are shown in the figure. The instance is predicted as class 2, which is the actual class predicted by the human expert. But the next highest vote, received by

```
Feature values of test instance 1:
F[1]: 2 F[2]: 3 F[3]: 3 F[4]: 3 F[5]: 3 F[6]: 0 F[7]: 0 F[8]: 0
F[9]: 3 F[10]:3 F[11]:0 F[12]:0 F[13]:0 F[14]:0 F[15]:0 F[16]:0
F[17]:3 F[18]:2 F[19]:2 F[20]:3 F[21]:3 F[22]:3 F[23]:1 F[24]:3
F[25]:0 F[26]:0 F[27]:0 F[28]:0 F[29]:0 F[30]:0 F[31]:0 F[32]:1
F[33]:0 F[34]:34
```

Classes:	[1]	[2]	[3]	[4]	[5]	[6]
Votes of Feature[1]:	0.16	0.16	0.17	0.18	0.15	0.18
Votes of Feature[2]:	0.36	0.27	0.15	0.07(-)	0.05(-)	0.10
Votes of Feature[3]:	0.34	0.05(-)	0.38	0.07(-)	0.10	0.05(-)
Votes of Feature[4]:	0.13	0.21	0.35	0.03(-)	0.28	0.00(-)
Votes of Feature[5]:	0.21	0.01(-)	0.52(+)	0.26	(-)	(-)
Votes of Feature[6]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[7]:	0.20	0.21	0.21	0.21	0.17	0.01(-)
Votes of Feature[8]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[9]:	0.47	0.01(-)	(-)	(-)	0.01(-)	0.51
Votes of Feature[10]:	0.87(+)	0.03(-)	(-)	(-)	(-)	0.10
Votes of Feature[11]:	0.15	0.19	0.20	0.20	0.20	0.07(-)
Votes of Feature[12]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[13]:	0.17	0.14	0.17	0.17	0.17	0.18
Votes of Feature[14]:	0.07(-)	0.10	0.21	0.20	0.21	0.21
Votes of Feature[15]:	0.20	0.20	0.19	0.20	0.01(-)	0.20
Votes of Feature[16]:	0.47	0.04(-)	0.01(-)	0.01(-)	0.36	0.10
Votes of Feature[17]:	0.20	0.16	0.25	0.06(-)	0.26	0.06(-)
Votes of Feature[18]:	0.28	0.02(-)	0.06(-)	0.13	0.29	0.22
Votes of Feature[19]:	0.24	0.14	0.17	0.10	0.13	0.22
Votes of Feature[20]:	0.98(+)	(-)	(-)	(-)	0.02(-)	(-)
Votes of Feature[21]:	0.62(+)	0.01(-)	(-)	(-)	0.37	0.00(-)
Votes of Feature[22]:	0.63(+)	0.07(-)	0.07(-)	0.07(-)	0.07(-)	0.07(-)
Votes of Feature[23]:	0.51(+)	0.12	0.09	0.09	0.10	0.09
Votes of Feature[24]:	0.48(+)	0.08(-)	0.22	0.08(-)	0.08(-)	0.08(-)
Votes of Feature[25]:	0.17	0.17	0.14	0.17	0.17	0.17
Votes of Feature[26]:	0.10	0.19	0.17	0.15	0.19	0.19
Votes of Feature[27]:	0.20	0.20	0.01(-)	0.20	0.20	0.20
Votes of Feature[28]:	0.37	0.02(-)	0.14	0.01(-)	0.31	0.16
Votes of Feature[29]:	0.18	0.18	0.12	0.18	0.18	0.18
Votes of Feature[30]:	0.19	0.19	0.19	0.19	0.19	0.05(-)
Votes of Feature[31]:	0.20	0.20	0.20	0.20	0.20	0.02(-)
Votes of Feature[32]:	0.14	0.18	0.10	0.18	0.21	0.18
Votes of Feature[33]:	0.20	0.20	0.00(-)	0.20	0.20	0.20
Votes of Feature[34]:	0.20	0.21	0.22	0.19	0.17	(-)
Total Votes:	10.22	4.47	5.08	4.33	5.57	4.32

```
Prediction: 1 actual class : 1
```

Figure 7.6. A correct classification of a given test instance (patient) drawn from the Dermatology domain by the VFI1 classifier.

```
Feature values of test instance 9:
F[1]: 2 F[2]: 2 F[3]: 2 F[4]: 1 F[5]: 0 F[6]: 0 F[7]: 0 F[8]: 0
F[9]: 0 F[10]:0 F[11]:0 F[12]:0 F[13]:0 F[14]:1 F[15]:0 F[16]:1
F[17]:2 F[18]:0 F[19]:0 F[20]:0 F[21]:0 F[22]:0 F[23]:0 F[24]:0
F[25]:0 F[26]:0 F[27]:0 F[28]:2 F[29]:0 F[30]:0 F[31]:0 F[32]:1
F[33]:0 F[34]:34
```

Classes:	[1]	[2]	[3]	[4]	[5]	[6]
Votes of Feature[1]:	0.16	0.16	0.17	0.18	0.15	0.18
Votes of Feature[2]:	0.17	0.19	0.16	0.17	0.12	0.19
Votes of Feature[3]:	0.23	0.13	0.22	0.17	0.12	0.14
Votes of Feature[4]:	0.17	0.18	0.12	0.18	0.13	0.21
Votes of Feature[5]:	0.16	0.19	0.12	0.13	0.20	0.20
Votes of Feature[6]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[7]:	0.19	0.19	0.19	0.19	0.18	0.06(-)
Votes of Feature[8]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[9]:	0.07(-)	0.22	0.23	0.23	0.22	0.04(-)
Votes of Feature[10]:	0.06(-)	0.19	0.20	0.20	0.20	0.15
Votes of Feature[11]:	0.14	0.19	0.20	0.20	0.20	0.07(-)
Votes of Feature[12]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[13]:	0.17	0.13	0.17	0.18	0.17	0.18
Votes of Feature[14]:	0.32	0.28	0.10	0.11	0.10	0.10
Votes of Feature[15]:	0.20	0.20	0.19	0.20	0.01(-)	0.20
Votes of Feature[16]:	0.21	0.11	0.10	0.10	0.25	0.23
Votes of Feature[17]:	0.17	0.17	0.17	0.16	0.16	0.17
Votes of Feature[18]:	0.14	0.22	0.20	0.18	0.13	0.12
Votes of Feature[19]:	0.04(-)	0.22	0.16	0.24	0.28	0.07(-)
Votes of Feature[20]:	0.02(-)	0.20	0.20	0.20	0.19	0.19
Votes of Feature[21]:	0.01(-)	0.22	0.24	0.24	0.04(-)	0.24
Votes of Feature[22]:	0.13	0.17	0.17	0.17	0.17	0.17
Votes of Feature[23]:	0.09	0.17	0.18	0.18	0.18	0.18
Votes of Feature[24]:	0.16	0.17	0.17	0.17	0.17	0.17
Votes of Feature[25]:	0.17	0.17	0.14	0.17	0.17	0.17
Votes of Feature[26]:	0.09	0.20	0.16	0.16	0.20	0.20
Votes of Feature[27]:	0.20	0.20	0.01(-)	0.20	0.20	0.20
Votes of Feature[28]:	0.00(-)	0.28	0.19	0.30	0.06(-)	0.18
Votes of Feature[29]:	0.18	0.18	0.12	0.18	0.18	0.18
Votes of Feature[30]:	0.19	0.19	0.19	0.19	0.19	0.04(-)
Votes of Feature[31]:	0.20	0.20	0.20	0.20	0.20	0.01(-)
Votes of Feature[32]:	0.14	0.19	0.09	0.18	0.22	0.17
Votes of Feature[33]:	0.20	0.20	0.00(-)	0.20	0.20	0.20
Votes of Feature[34]:	0.21	0.21	0.22	0.19	0.16	(-)
Total Votes:	5.12	6.44	5.38	6.26	5.67	5.13

```
Prediction: 2 actual class : 2
```

Figure 7.7. Another correct (not that confident as the previous classification) classification of a given test instance (patient) drawn from the Dermatology domain by the VFI1 classifier.

class 4, is not much different than the vote of class 2. Thus, this prediction is in fact not that much confident because the classifier chooses one class rather than the other depending on a very slight difference in the votes. If we look at the individual votes of each feature, we see that there is no class receiving a (+) from any feature; that is, no feature can be exactly sure about which class to predict. There are some (–) classes and mostly the features are neutral about the classes. When we compare the feature votes of class 2 with that of class 4, we do not find votes much different than each other except the votes of especially feature 14. Since the votes of this feature support class 2 rather than class 4, it affects the final prediction to be class 2. The difference between votes for these two classes is the highest in feature 14, so feature 14 with value 1 seems to be the most important feature in distinguishing between class 2 and class 4. Our human expert admitted that she also encounters the same problem of distinguishing between class 2 and class 4 as encountered by the VFI1 classifier. In this classification (Figure 7.7), VFI1 classified the instance correctly, but the next instance will be misclassified by the VFI1 classifier.

The classification information of a test instance misclassified by VFI1 is shown in Figure 7.8. The feature values of the instance are shown at first and then the individual feature votes are displayed. The prediction of the classifier is class 4 (*pityriasis rosea*) whereas the actual prediction of the human expert for this instance was class 2 (*seboreic dermatitis*). Class 4 received the highest vote, but class 2 received a vote very close to the vote of class 4. Thus, this prediction is in fact not much confident because the classifier chooses one class rather than the other depending on a very slight difference in the votes. If we look at the individual votes of each feature, we see that there is no class receiving a (+) from any feature; that is, no feature can be exactly sure about which class to predict. There are some (–) classes and mostly the features are neutral (o) about the classes. When we compare the feature votes of class 2 with that of class 4, we do not find votes much different than each other except the votes of feature 4 (*itching*) which votes 0.08 for class 2 and 0.29 for class 4 and feature 14 (*PNL infiltrate*), which votes 0.28 for class 2 and 0.11 for class 4, which is the predicted class. This means that only features 4 and 14 can be more useful to differentiate between class 4 and 2 than other features do but in different directions. If feature 14 had been given the opportunity to have

```
Feature values of test instance 3:
F[1]: 2 F[2]: 1 F[3]: 1 F[4]: 0 F[5]: 0 F[6]: 0 F[7]: 0 F[8]: 0
F[9]: 1 F[10]:0 F[11]:0 F[12]:0 F[13]:0 F[14]:1 F[15]:0 F[16]:1
F[17]:1 F[18]:0 F[19]:1 F[20]:0 F[21]:0 F[22]:0 F[23]:0 F[24]:0
F[25]:0 F[26]:0 F[27]:0 F[28]:2 F[29]:0 F[30]:0 F[31]:0 F[32]:2
F[33]:0 F[34]:35
```

Classes:	[1]	[2]	[3]	[4]	[5]	[6]
Votes of Feature[1]:	0.16	0.16	0.17	0.18	0.15	0.18
Votes of Feature[2]:	0.09	0.09	0.17	0.20	0.30	0.14
Votes of Feature[3]:	0.11	0.19	0.11	0.20	0.20	0.20
Votes of Feature[4]:	0.20	0.08	0.03(-)	0.29	0.10	0.30
Votes of Feature[5]:	0.16	0.19	0.12	0.13	0.20	0.20
Votes of Feature[6]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[7]:	0.19	0.19	0.19	0.19	0.18	0.06(-)
Votes of Feature[8]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[9]:	0.26	0.12	0.11	0.11	0.12	0.29
Votes of Feature[10]:	0.06(-)	0.19	0.20	0.20	0.20	0.15
Votes of Feature[11]:	0.14	0.19	0.20	0.20	0.20	0.07(-)
Votes of Feature[12]:	0.17	0.17	0.13	0.17	0.17	0.17
Votes of Feature[13]:	0.17	0.13	0.17	0.18	0.17	0.18
Votes of Feature[14]:	0.32	0.28	0.10	0.11	0.10	0.10
Votes of Feature[15]:	0.20	0.20	0.19	0.20	0.01(-)	0.20
Votes of Feature[16]:	0.21	0.11	0.10	0.10	0.25	0.23
Votes of Feature[17]:	0.13	0.15	0.10	0.23	0.13	0.26
Votes of Feature[18]:	0.14	0.22	0.20	0.18	0.13	0.12
Votes of Feature[19]:	0.13	0.15	0.17	0.20	0.16	0.18
Votes of Feature[20]:	0.02(-)	0.20	0.20	0.20	0.19	0.19
Votes of Feature[21]:	0.01(-)	0.22	0.24	0.24	0.04(-)	0.24
Votes of Feature[22]:	0.13	0.17	0.17	0.17	0.17	0.17
Votes of Feature[23]:	0.09	0.17	0.18	0.18	0.18	0.18
Votes of Feature[24]:	0.16	0.17	0.17	0.17	0.17	0.17
Votes of Feature[25]:	0.17	0.17	0.14	0.17	0.17	0.17
Votes of Feature[26]:	0.09	0.20	0.16	0.16	0.20	0.20
Votes of Feature[27]:	0.20	0.20	0.01(-)	0.20	0.20	0.20
Votes of Feature[28]:	0.00(-)	0.28	0.19	0.30	0.06(-)	0.18
Votes of Feature[29]:	0.18	0.18	0.12	0.18	0.18	0.18
Votes of Feature[30]:	0.19	0.19	0.19	0.19	0.19	0.04(-)
Votes of Feature[31]:	0.20	0.20	0.20	0.20	0.20	0.01(-)
Votes of Feature[32]:	0.18	0.16	0.17	0.17	0.15	0.17
Votes of Feature[33]:	0.20	0.20	0.00(-)	0.20	0.20	0.20
Votes of Feature[34]:	0.21	0.21	0.22	0.19	0.16	0.00(-)
Total Votes:	5.23	6.09	5.08	6.34	5.57	5.69

```
Prediction: 4 actual class : 2
```

Figure 7.8. An incorrect classification of a given test instance (patient) drawn from the Dermatology domain by the VFI1 classifier.

more voting power than the other features, it might have changed the final prediction. The lesson we can draw from this misclassification is that feature 14 with value 1 is important to differentiate between class 2 and 4.

To differentiate between class 2 and 4 is also a hard task for the human expert. We showed her the feature values of the instance and asked her to make the classification again looking at these feature values. She said that to make a diagnosis for this instance is really tough and problematic. She guessed this instance is either of class 2 or class 4, waited for a while, and after looking at feature 26 (*disappearance of the granular layer*), which does not appear in this patient, she said it seems more like class 2 but may also be class 4. But the votes of feature 26 for both classes are nearly the same, which means that although the doctor says that the lower values of feature 26 is a sign for class 4, the up-to-now training instances only slightly show this. As a result, it is not surprising for VFI classifiers to make misclassification in class 2 and class 4 instances in the Dermatology dataset because it is also difficult for the doctor to differentiate between them. For example, with this instance, the doctor could not pick up good differentiating features confirming class 2. The close total votes received by these two classes also show this difficulty. One advantage of VFI classifiers is that one can see the probability of each class as well as the predicted class, such as the total votes in Figure 7.8 tell that the prediction is class 4 but the next possible class is 2 with very similar probability.

The explanations generated by VFI classifiers give valuable information about the classifications such as the next possible class as well as the predicted class, the features confirming which classes and how much they confirm, the features rejecting which classes. This kind of information might help the human expert in making new classifications especially if the human expert is not experienced enough. Although the human expert collecting the data for us is very experienced in this field, our classifier corrected two of her misclassifications, that made her change her previous classification. What I say in this situation is that "VFI classifiers saved one patient". The classification of one of the instances is shown in Figure 7.9. The concept description used to classify this instance was learned by 365 training instances. The VFI1 classifier predicts class 1 with high confidence (total vote = 8.33), but the actual class

```
Feature values of test instance 1:
F[1]: 2 F[2]: 2 F[3]: 2 F[4]: 3 F[5]: 2 F[6]: 0 F[7]: 0 F[8]: 0
F[9]: 1 F[10]:1 F[11]:0 F[12]:0 F[13]:0 F[14]:1 F[15]:0 F[16]:0
F[17]:1 F[18]:1 F[19]:1 F[20]:1 F[21]:1 F[22]:1 F[23]:1 F[24]:1
F[25]:0 F[26]:2 F[27]:0 F[28]:0 F[29]:0 F[30]:0 F[31]:0 F[32]:3
F[33]:0 F[34]:40
```

Classes:	[1]	[2]	[3]	[4]	[5]	[6]
Votes of Feature[1]:	0.16	0.16	0.17	0.17	0.14	0.18
Votes of Feature[2]:	0.20	0.21	0.16	0.15	0.09	0.19
Votes of Feature[3]:	0.26	0.12	0.25	0.15	0.10	0.12
Votes of Feature[4]:	0.11	0.18	0.38	0.03(-)	0.29	0.01(-)
Votes of Feature[5]:	0.19	0.07(-)	0.34	0.27	0.06(-)	0.06(-)
Votes of Feature[6]:	0.16	0.16	0.19	0.16	0.16	0.16
Votes of Feature[7]:	0.20	0.20	0.20	0.20	0.18	0.02(-)
Votes of Feature[8]:	0.16	0.16	0.21	0.16	0.16	0.16
Votes of Feature[9]:	0.28	0.11	0.08	0.08(-)	0.10	0.36
Votes of Feature[10]:	0.39	0.12	0.10	0.08(-)	0.08(-)	0.23
Votes of Feature[11]:	0.14	0.18	0.19	0.19	0.19	0.10
Votes of Feature[12]:	0.16	0.16	0.21	0.16	0.16	0.16
Votes of Feature[13]:	0.17	0.15	0.16	0.17	0.17	0.17
Votes of Feature[14]:	0.32	0.34	0.07(-)	0.10	0.07(-)	0.11
Votes of Feature[15]:	0.20	0.20	0.19	0.20	0.02(-)	0.20
Votes of Feature[16]:	0.21	0.11	0.11	0.14	0.23	0.20
Votes of Feature[17]:	0.15	0.17	0.14	0.21	0.12	0.21
Votes of Feature[18]:	0.16	0.15	0.16	0.16	0.17	0.21
Votes of Feature[19]:	0.12	0.16	0.17	0.20	0.15	0.20
Votes of Feature[20]:	0.47(+)	0.08	0.08	0.08	0.12	0.15
Votes of Feature[21]:	0.27	0.13	0.09	0.09	0.31	0.13
Votes of Feature[22]:	0.61(+)	0.08(-)	0.07(-)	0.07(-)	0.08(-)	0.07(-)
Votes of Feature[23]:	0.45(+)	0.18	0.08(-)	0.08(-)	0.09	0.12
Votes of Feature[24]:	0.60(+)	0.08(-)	0.08(-)	0.09	0.08(-)	0.08(-)
Votes of Feature[25]:	0.19	0.19	0.03(-)	0.19	0.19	0.19
Votes of Feature[26]:	0.50(+)	0.07(-)	0.17	0.12	0.07(-)	0.07(-)
Votes of Feature[27]:	0.20	0.20	0.01(-)	0.20	0.20	0.20
Votes of Feature[28]:	0.15	0.17	0.16	0.18	0.16	0.18
Votes of Feature[29]:	0.20	0.20	0.01(-)	0.20	0.20	0.20
Votes of Feature[30]:	0.19	0.19	0.19	0.19	0.19	0.06(-)
Votes of Feature[31]:	0.20	0.19	0.20	0.20	0.20	0.02(-)
Votes of Feature[32]:	0.16	0.15	0.26	0.13	0.19	0.11
Votes of Feature[33]:	0.20	0.20	0.00(-)	0.20	0.20	0.20
Votes of Feature[34]:	0.20	0.19	0.23	0.19	0.17	0.01(-)
Total Votes:	8.33	5.42	5.15	5.17	5.10	4.83

```
Prediction: 1 actual class : 2
```

Figure 7.9. A misclassification of an instance drawn from the Dermatology domain done by the human expert and corrected by the VFI1 classifier.

told by the doctor was class 2, which received a total vote of 5.42. There are features confirming class 1 but there are no features confirming class 2. Moreover, there are features rejecting class 2. These individual votes make up a significant difference in the votes of these two classes. When we showed the human expert this classification results pointing the total vote received by class 1, she changed her mind and approved the classification of VFI1. This was a very important result achieved by the VFI classifiers, since the computer corrected the human expert's fault by learning from the previous patients with known diagnoses.

In this chapter, we have shown that VFI classifiers do not work like black boxes and can explain why and how it came up with the resulting classification in a comprehensible way to human. The human expert agrees with the information visualized in the concept descriptions learned by VFI classifiers. The classification explanations do not only display only the prediction but also how certain that prediction is compared to other classes.

Chapter 8

Conclusions and Future Work

We have presented several new multi-concept learning algorithms called Voting Feature Intervals (VFI) algorithms. The VFI classification algorithms are non-incremental supervised inductive learning algorithms that learn the concept descriptions in the form of sets of feature intervals on each feature dimension from a set of preclassified examples provided by a teacher. A feature interval represents a set of classes with its individual votes and the classification of a new instance is determined by the sum of these individual votes distributed by each feature. The features might have equal voting power or some relevant features might have been given a higher voting power than some other irrelevant features. The relevance information of features can be learned by a feature weight learning method which is also developed and applied to the VFI classifiers in this thesis.

Representing a concept separately on each feature dimension allows faster classification than the nearest neighbor and the decision tree induction algorithms. The classification in the VFI classifiers has been shown to be much much faster than that of the well-known 1-NN algorithm. Moreover, the classification in the VFI classifiers are also faster than that of the NBCN classifier on the average. This separate representation also enables a natural and effective method of handling missing (unknown) feature values for which a value should be provided to replace in both the nearest neighbor and the decision tree induction algorithms. The experiments on artificially generated datasets

containing missing feature values have shown that the method of simply ignoring only the feature with the unknown value used in all feature projection based methods and the Naive Bayesian classifier results in higher accuracies than those used by the 1-NN and C4.5 algorithms. Another advantage of the separate knowledge representation of the concept description is that the normalization of feature values to a same range for all the features is not required as required in the case of the nearest neighbor algorithm.

The knowledge representation scheme based on feature projections has been used in several other learning methods [32, 73, 7, 8] which have generalized the feature projections of the training instances in different ways. The experiments on real-world datasets have shown that the VFI classifiers achieve the highest classification accuracies among all these feature projection based methods. On the other hand, the VFI classifiers have not always achieved higher accuracies than the well-known 1-NN and C4.5 algorithms. However, the VFI classifiers have been usually more successful than the Naive Bayesian classifier assuming normal distribution for linear features (NBCN), which is also a very classical classifier.

In these performance comparisons, we have used the unweighted VFI classifiers where each feature has equal voting power. We have also developed weight learning genetic algorithms that learn the optimum feature relevance weights to maximize the classification accuracy of the given classifier. The weight learning experiments on some real-world datasets have shown that it is possible to have significant increase in the classification accuracy of the VFI classifiers. However, one deficiency of genetic algorithms is that they are slow, therefore we could not have applied them to learn weights for large datasets. As a future work, other feature weight learning methods [76] might be used to learn feature weights for the VFI classifiers.

We have proposed and developed a new crossover operator called continuous uniform crossover (CUCO) to be used in these weight learning genetic algorithms and compared it with three common crossover operators by using them in genetic algorithms that learn weights for the nearest neighbor algorithm [21]. Experiments have shown that CUCO is generally better than the other crossover operators. Nevertheless, the classification accuracies of the weighted

nearest neighbor algorithm using different crossover operators are close to each other.

The effect of the presence of irrelevant features in the datasets have been investigated and the experiments have shown that the classification accuracy of the VFI classifiers are not affected much with the addition of irrelevant features. On the other hand, the performance of other feature projection based methods and the nearest neighbor algorithm degrade in the presence of irrelevant features. However, the VFI classifiers are affected negatively when the datasets contain noisy feature values. The negative effect of noisy feature values on the classification performance of the VFI classifiers might be investigated and noise-tolerant versions might be developed for further research.

What is as much important as how accurate and/or fast a classifier performs is the understandability of both the concept description learned and the classification process. For this purpose, we have visualized the concept description in the form of sets of intervals on each feature dimension where an interval either confirms, or rejects, or does none of these two for some class in the domain on each feature. Our human expert has agreed with the information visualized for the Dermatology dataset. Another useful understandability property of the VFI classifiers is the explanation ability of the VFI classifiers in classification. VFI classifiers do not work like a black box and can explain why and how they came up with the resulting classification in a comprehensible way to human. The explanations generated by VFI classifiers give valuable information about the classifications such as the next possible class as well as the predicted class, the features confirming which classes and how much they are confident with their confirmation, and the features rejecting which classes. This kind of information might help the human expert in making new classifications especially if the human expert is not experienced enough.

Other than those specified above, we have another direction for future work. The individual voting of features is common in all feature projection based learning methods where some of them use single-class voting such as the CFP and the FIL algorithms [7, 32] and some others use multi-class voting such as the COFI and the VFI algorithms [22, 73]. The sum of the votes distributed to classes in the unweighted VFI algorithm is equal to 1 and the votes are positive

real values. Some features give a vote 0 or some value which we visualized as (–) meaning a rejection and some features give such a vote that is visualized by (+) meaning a confirmation in Chapter 7. As one future research direction, the votes we visualized as (–) might get a negative vote from that feature, which will provide that feature with more rejection power. As another research direction, these (+) and (–) evaluations might be used to learn feature weights that differ among intervals of that feature for the VFI classifiers; that is, a feature would have different weights depending on its intervals. Thus, each feature-interval pair might have a weight related to the (+) or (–) evaluation of the interval. This is meaningful because the intervals that have (+) and/or (–) evaluations for classes are significantly differentiating one class from another, thus those intervals are more informative than the intervals that distribute equal votes to all classes.

Another further research might be carried on the point intervals constructed on nominal feature dimensions. Since the values of nominal features have no relation with each other, unlike the linear ordering relation between the values of a linear feature, all intervals of a nominal feature are point intervals. A point interval is defined on a singleton set of values in the VFI algorithms described in this thesis. Instead of constructing point intervals from all distinct values that a nominal feature can take on, some —two or more— point intervals might be combined into a multi-point interval. Then, this multi-point interval would be defined on a subset of values instead of a singleton set of values. This combination can be determined by the class distributions of the point intervals to be combined. For example, it might be meaningful and efficient to combine two point intervals which have very similar class distributions. However, the combination of point intervals would not be correct if the nominal feature is a boolean feature.

The main advantages of the VFI classifiers, which learns the concept descriptions in the form of sets of intervals separately on each feature and uses a voting scheme in classification, can be summarized as follows:

- highest classification accuracies among all other feature projection based methods such as the CFP, COFI, 1-NNFP, and FIL algorithms

- faster classification than other well-known classifiers
- visualization of the learned concept description and the explanation ability of the classification in a comprehensible way to humans
- separate consideration of features yields a natural and effective way of handling unknown feature values, which is a common advantage of all feature projection based methods
- separate consideration of features does not require any normalization of feature values to a same range
- robust to the presence of irrelevant features in the domain
- allows incorporation of feature weights from external sources

The major disadvantage of this representation is that concept descriptions involving a conjunction between two or more features can not be represented. The feature projection based algorithms are not applicable to domains where all of the concept descriptions overlap, or domains in which concept descriptions are nested. Instead, they are applicable to concepts where each feature, independent of other features, can contribute to the classification of an instance. In fact, this is the nature of the most real-world datasets. Holte has pointed out that the most datasets in the UCI repository are such that, for classification, their features can be considered independently of each other [36]. Also Kononenko claimed that in the data used by human experts there are no strong dependencies between features because features are properly isolated and defined [44].

This thesis has completed the work on feature projection based learning algorithms. The CFP, COFI, 1-NNFP, FIL, and VFI classification algorithms all learn the concept descriptions separately on each feature by generalizing the feature projections of the training examples and use a voting scheme where each feature participates in the classification by its individual vote. This thesis wraps up all these feature projection based learning algorithms into a unifying formalism. In this formalism, the algorithms are categorized into Single-Class vs Multi-Class according to whether the basic unit of representation carries

classification information for a single class or for all classes. Another dimension according to which the algorithms are categorized is whether the training examples are processed in an incremental or non-incremental manner. Thus, this thesis has presented a wide comparison between all feature projection based classification learning algorithms.

Bibliography

[1] D.W. Aha, Incremental, Instance–Based Learning of Independent and Graded Concept Descriptions, In *Proceedings of the Sixth International Workshop Machine Learning*, 387–391, Ithaca, NY: Morgan Kaufmann, 1989.

[2] D.W. Aha, A Study of instance–based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine, 1990.

[3] D.W. Aha, Tolerating Noisy, Irrelevant and Novel Attributes in Instance–Based Learning Algorithms, *International Journal of Man–Machine Studies*, 36:267–287, 1992.

[4] D.W. Aha and D. Kibler, Noise–Tolerant Instance–Based Learning Algorithms, In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 794–799, Detroit, MI: Morgan Kauffman, 1989.

[5] D.W. Aha, D. Kibler and M.K. Albert, Instance–Based Learning Algorithms, *Machine Learning*, 6:37–66, 1991.

[6] D. W. Aha and R. L. Bankert, Feature selection for case–based classification of cloud types: An empirical comparison. In D. Aha (Ed.) *Case–Based Reasoning: Papers from the 1994 Workshop* (*TR WS–94–01*) Menlo Park, CA: AAAI Press, 1994.

[7] A. Akkuş. Batch Learning of Disjoint Feature Intervals. Bilkent University, Dept. of Computer Engineering and Information Science, MSc. Thesis, 1996.

[8] A. Akkuş and H. A. Güvenir, k Nearest Neighbor Classification on Feature Projections, In *Proceedings of the* 13^{th} *International Conference on Machine Learning.* Lorenza Saitta (Ed.), Bari, Italy: Morgan Kaufmann, 12–19, 1996.

[9] M.K. Albert and D.W. Aha, Analyses of Instance–Based Learning Algorithms, In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 553–558, 1991.

[10] H. Almuallim and T.G. Dietterich, Learning with Many Irrelevant Features, In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 547–552, 1991.

[11] D. Angluin and P. Laird, Learning from Noisy Examples, *Machine Learning*, 2:343–370, 1988.

[12] P. Auer, R. C. Holte andW. Maass, Theory and Applications of Agnostic PAC–Learning with Small Desicion Trees, In *Proceedings of the* 12^{th} *International Conference on Machine Learning.* A. Prieditis and S. Russell (Ed.), 21–29, 1995.

[13] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Belmont, CA: Wadsworth, 1984.

[14] B. Cestnik, I. Kononenko, and I. Bratko, ASSISTANT 86: A knowledge–elicitation tool for sophisticated users, in *Progress in Machine Learning*, Wilmslow, UK: Sigma Press, 1987.

[15] P. Clark and T. Niblett, Induction in Noisy Domains, In I. Bratko and N.Lavrac (Eds.), *Progress in Machine Learning*, 11–30, Wilmslow, England:Sigma Press, 1987.

[16] S. Cost, S. Salzberg, A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features, *Machine Learning*, 10(1):57–58, 1993.

[17] T.M. Cover and P.E. Hart, Nearest Neighbor Pattern Classification, *IEEE Transactions on Information Theory*: 13:21–27, 1967.

[18] B. V. Dasarathy, *Nearest Neighbor (NN) Norms, NN Pattern Classification Techniques.* IEEE Computer Society Press, 1990.

[19] G. Dejong and R. Mooney, Explanation–Based Learning: An Alternative View, *Machine Learning*, 1:145–176, 1986.

[20] G. Dejong, Learning with Genetic Algorithms: An Overview, *Machine Learning*, 3:121–128, 1988.

[21] G. Demiröz, and H. A. Güvenir. Genetic Algorithms to Learn Feature Weights for the Nearest Neighbor Algorithm. In *Proceedings of the* 6^{th} *Belgian–Dutch Conference on Machine Learning* (*BENELEARN–96*), 117–126, 1996.

[22] G. Demiröz, and H. A. Güvenir. Classification by Voting Feature Intervals. In *Proceedings of Ninth European Conference on Machine Learning* (*ECML-97*), Springer–Verlag, LNAI 1224, 85–92, 1997.

[23] G. Demiröz, H. A. Güvenir, and Nilsel İlter. Differential Diagnosis of Erythemato–Squamous Diseases using Voting Feature Intervals. In *Proceedings of the Sixth Turkish Symposium on Artificial Intelligence and Neural Networks* (*TAINN'97*), 190–194, 1997.

[24] R.D. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, New York: Wiley, 1973.

[25] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*, In SIAM, Philadelphia, Pa., 1982.

[26] D.H. Fisher, Knowledge Acquisition Via Incremental Conceptual Clustering, *Machine Learning*, 2:139–172, 1987.

[27] J.M. Fitzpatrick and J.J. Grefenstette, Genetic Algorithms in Noisy Environments, *Machine Learning*, 3:101–120, 1988.

[28] J. H. Friedman, A recursive partitioning decision rule for non–parametric classification, *IEEE Transactions on Computers*, 404–408, 1977.

[29] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, San Diego, 1990.

[30] J.W. Grzymala–Busse, On the Unknown Attribute Values in Learning from Examples, In *Proceedings of Sixth International Symposium Methodologies for Intelligent Systems*, 368–377, October 1991.

[31] H.A. Güvenir and İ. Şirin, A Genetic Algorithm for Classification by Feature Partitioning, In *Proceedings of the fifth International Conference on Genetic Algorithms*, 543–548, 1993.

[32] H.A. Güvenir and İ. Şirin, Classification by Feature Partitioning, *Machine Learning*, **23**:47–67, 1996.

[33] H.A. Güvenir, B. Acar, G. Demiröz, and A. Çekin. A Supervised Machine Learning Algorithm for Arrhythmia Analysis, to appear in *Computers in Cardiology*, Lund, Sweden, 1997.

[34] D. Heath, S. Kasif, and S. Salzberg, Learning Oblique Decision Trees, *Proceedings of 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1002–1007, 1993.

[35] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[36] R.C. Holte, Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, *Machine Learning*, 11:63–91, 1993.

[37] E. Hunt, J.Marin and P. Stone, *Experiments in Induction*, New York, Academic Press, 1966.

[38] G. H. John, R. Kohavi and K. Pfleger, Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*. New Brunswick, NJ: Morgan Kaufmann, 293–301, 1994.

[39] L. Kanal and Chandrasekaran, On Dimensionality and Sample Size In Statistical Pattern Classification, *Pattern Recognition*, 225–234, 1971.

[40] J.D. Kelly and L. Davis, A Hybrid Genetic Algorithm for Classification, In *Proceedings of the twelfth International Joint Conference on Artificial Intelligence*, 645–650, 1991.

[41] D. Kibler and P. Langley, Machine Learning as an Experimental Science, In J.W. Shavlik and T.G. Ditterich, editors, *Readings in Machine Learning*, 38–43. Morgan Kaufman, San Mateo, CA, 1990.

[42] R. Kohavi, B. Becker, and D. Sommerfield. Improving Simple Bayes. In Poster Papers of *Ninth European Conference on Machine Learning* (*ECML-97*), 78–87, 1997.

[43] R. Kohavi, D. Sommerfield, and J. Dougherty. Data Mining using MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, 234–245, 1996. http://www.sgi.com/Technology/mlc.

[44] Kononenko, I. (1993). Inductive and Bayesian Learning in Medical Diagnosis. *Applied Artificial Intelligence*, Vol. 7, 317–337.

[45] Kononenko, I. & Bratko, I. (1991). Information–Based Evaluation Criterion for Classifier's Performance. *Machine Learning*, Vol. 6, 67–80.

[46] P. Lachenbruch and M. Mickey, Estimation of Error Rates in Discriminant Analysis, *Technometrics*, 1–111, 1968.

[47] H. Lounis and G. Bisson, Evaluation of Learning Systems: An Artificial Data–Based Approach, In *Proceedings of European Working Session on Learning*, 463–481, 1991.

[48] D. Medin and M. Schaffer, Context Theory of Classification Learning, *Psychological Review*, 85:3, 207–238, 1978.

[49] R.S. Michalski, J.G. Carbonell and T.M. Mitchell, *Machine Learning, An Artificial Intelligence Approach*, Los Altos: Morgan Kaufmann, 1983.

[50] T.M. Mitchell, R. Keller and S. Kedar–Cabelli, Explanation–Based Generalization: A Unifying View, *Machine Learning*, 1:47–80, 1986.

[51] P. Murphy, UCI Repository of machine learning databases – Maintained at the Department of Information and Computer Science, University of California, Irvine, Anonymous FTP from ics.uci.edu in the directory pub/machine–learning–databases, 1995.

[52] S.K. Murthy, S. Kasif, S.Salzberg, and R. Beigel, OC1: Randomized Induction of Oblique Decision Trees, In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (*AAAI-93*), 322–327, 1993.

[53] S. Okamoto and K. Satoh, An Average–Case Analysis of k–Nearest Neighbor Classifier. In *Proceedings of the First International Conference on Case–Based Reasoning*, 243–264, 1995.

[54] J.R. Quinlan, The Effect of Noise on concept Learning, In R.S. Michalski, J.G. Carbonell and T.M. Mitchell, *Machine Learning Volume II: An Artificial Intelligence Approach*, Los Altos: Morgan Kaufmann, 1983.

[55] J.R. Quinlan, Induction of Decision Trees, *Machine Learning*, 1:81–106, 1986.

[56] J.R. Quinlan, Decision Trees as Probabilistic Classifiers, In *Proceedings of Fourth International Workshop on Machine Learning*, 31–37, June 1987.

[57] J.R. Quinlan, Unknown Attribute Values in Induction, In A. Segre (Ed.), In *Proceedings of the 16th International Workshop on Machine Learning*, 164–168, San Mateo, CA:Morgan Kaufmann, 1989.

[58] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, California, 1993.

[59] J. Rachlin, S. Kasif, S. Salzberg and D.W. Aha, Towards a Better Understanding of Memory–Based Reasoning Systems, *International Machine Learning Conference*, 1994.

[60] L. Rendell, A new basis for state–space learning systems and a successful implementation, *Artificial Intelligence* 20:369–392, 1983.

[61] D. Rumelhart, G. Hinton, and R. Williams, Learning Representations by back–propagating errors. *Nature*, .

[62] S. Salzberg, *Learning with Generalized Exemplars*, Kluwer Academic Publishers, Massachusetts, 1990.

[63] S. Salzberg, A Nearest Hyperrectangle Learning Method, *Machine Learning*, 6:251–276, 1991.

[64] S. Salzberg, Distance Metrics for Instance–Based Learning, *ISMIS'91 6th International Symposium, Methodologies for Intelligent Systems*, 399–408, 1991.

[65] J. W. Shavlik and T. G. Dietterich, *Readings in Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1990.

[66] H. A. Simon, Why should machines learn? In *Machine learning: an artificial intelligence approach* (Vol. I), San Mateo, CA: Morgan Kaufmann, 1983.

[67] D. B. Skalak, Prototype and feature selection by sampling and random mutation hill–climbing algorithms, In *Proceedings of the* 11^th *International Conference on Machine Learning*, New Brunswick, NJ: Morgan Kaufmann, 293–301, 1994.

[68] G. Stanfill and D. Waltz, Toward Memory–Based Reasoning, *Communications of the ACM* 29:1213–1228, 1986.

[69] J.C. Schlimmer and R.H. Granger, Incremental Learning from Noisy Data, *Machine Learning*, 1:317–354, 1986.

[70] İ. Şirin and H.A. Güvenir, Empirical Evaluation of the CFP Algorithm, In *Proceedings of the Sixth Australian Joint Conference on Artificial Intelligence*, pages 311–315, 1993.

[71] İ. Şirin and H.A. Güvenir, *An Algorithm for Classification by Feature Partitioning*, Technical Report CIS–9301, Bilkent University, Dept. of Computer Engineering and Information Science, Ankara, 1993.

[72] A. K. Spackman, Learning Categorical Decision Criteria in Biomedical Domains, In *Proceedings of the Fifth International Conference on Machine Learning*, University of Michigan, Ann Arbor, 1988.

[73] H. G. Ünsal, Classification with Overlapping Feature Intervals, Bilkent University, Dept. of Computer Engineering and Information Science, MSc. Thesis, 1995.

[74] D. Wettschereck, A study of Distance–Based Machine Learning Algorithms, PhD Thesis, Oregon State University, 1994.

[75] D. Wettschereck and D. Aha, Weighting Features, In *Proceedings of the First International Conference on Case–Based Reasoning*,Lisbon, Portugal: Springer–Verlag, 1995.

[76] D. Wettschereck, D. Aha, and T. Mohri, A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. To appear in *Artificial Intelligence Review*, 1997.

A

Real-World Datasets

Table A.1. Comparison on some real-world datasets.

Dataset	Size	# of Features	# of Linear Features	# of Classes	Unknown Values (%)	Baseline Accuracy (%)
Arrhythmia	452	279	279	16	0.33	54
Bcancerw	699	10	10	2	0.25	66
Cleveland	303	13	6	2	0	54
Dermatology	309	34	34	6	0.07	31
Diabetes	768	8	8	2	0	65
Glass	214	9	9	6	0	36
Horse	368	22	7	2	24	63
Hungarian	294	13	6	2	0	64
Ionosphere	351	34	34	2	0	64
Iris	150	4	4	3	0	33
Liver	345	6	6	2	0	58
Musk	476	166	166	2	0	57
New-thyroid	215	5	5	3	0	70
Page-blocks	5473	10	10	5	0	90
Segmentation	2310	19	19	7	0	14
Sonar	208	60	60	2	0	53
Vehicle	846	18	18	4	0	26
Wine	178	13	13	2	0	40

Table A.1 summarizes some properties of the datasets to be used in the experiments. In this table, name of the real-world datasets are shown with the

size of the dataset, number of features, number of linear features, number of classes, percentage of the unknown attribute values, and the baseline accuracy. The baseline accuracy of a dataset is the accuracy that will be obtained by predicting the class of any test instance as the class of the most frequently occurring class.

Arrhythmia: In this thesis, we construct two real-world datasets. One of them is the Arrhythmia dataset. The aim is to distinguish between the presence and types of cardiac Arrhythmia and to classify it in one of the 16 groups. Currently, there are 452 patient records which are described by 279 feature values. Class 01 refers to *normal* ECG, class 02 to *Ischemic changes (Coronary Artery Disease)*, class 03 to *Old Anterior Myocardial Infarction*, class 04 to *Old Inferior Myocardial Infarction*, class 05 to *Sinus tachycardy*, class 06 to *Sinus bradycardy*, class 07 to *Ventricular Premature Contraction (PVC)*, class 08 to *Supraventricular Premature Contraction (PVC)*, class 09 to *Left bundle branch block*, class 10 to *Right bundle branch block*, class 11 to *1. degree AtrioVentricular block*, class 12 to *2. degree AtrioVentricular block*, class 13 to *3. degree AtrioVentricular block*, class 14 to *Left ventricule hypertrophy*, class 15 to *Atrial Fibrillation or Flutter*, and class 16 refers to the rest. The first 9 features are *Age* (f_1) given in years, *Sex* (f_2) which is either male or female, *Height* (f_3) given in centimeters, *Weight* (f_4) given in kilograms, *QRS interval* (f_5) which is the average QRS duration in msec., *P-R interval* (f_6) which is the average duration between onset of P and Q waves in msec., *Q-T interval* (f_7) which is the average duration between onset of Q and offset of T waves in msec., *T interval* (f_8) which is the average duration of T wave in msec., *P interval*(f_9) which is the average duration of P wave in msec. The features from f_{10} to f_{14} are the vector angles in degrees on front plane of *QRS* (f_{10}), *T* (f_{11}), *P* (f_{12}), *QRST* (f_{13}), and *J* (f_{14}) respectively. The feature f_{15} is *heart rate* which is the number of heart beats per minute. The following 11 features are measured from the **DI** channel: *Average width of Q wave* measured in msec. (f_{16}), *Average width of R wave* measured in msec. (f_{17}), *Average width of S wave* measured in msec. (f_{18}), *Average width of R' wave* measured in msec. (f_{19}), *Average width of S' wave* measured in msec. (f_{20}), *Number of intrinsic deflections* (f_{21}), *Existence of ragged R wave* (f_{22}) which is a boolean feature, *Existence of diphasic derivation of R wave* (f_{23})

which is a boolean feature, *Existence of ragged P wave* (f_{24}) which is a boolean feature, *Existence of diphasic derivation of P wave* (f_{25}) which is a boolean feature, *Existence of ragged T wave* (f_{26}) which is a boolean feature, *Existence of diphasic derivation of T wave* (f_{25}) which is a boolean feature. The above 11 features measured for the **DI** channel are all measured for the **DII** (from feature f_{28} to f_{39}), **DIII** (from feature f_{40} to f_{51}), **AVR** (from feature f_{52} to f_{63}), **AVL** (from feature f_{64} to f_{75}), **AVF** (from feature f_{76} to f_{87}), **V1** (from feature f_{88} to f_{99}), **V2** (from feature f_{100} to f_{11}), **V3** (from feature f_{112} to f_{123}), **V4** (from feature f_{124} to f_{135}), **V5** (from feature f_{136} to f_{147}), and **V6** (from feature f_{148} to f_{159}) channels. The following 9 features are measured from the **DI** channel: *Amplitude of JJ wave* (f_{160}) measured in $\times 0.1$ milivolts, *Amplitude of Q wave* (f_{161}) measured in $\times 0.1$ milivolts, *Amplitude of R wave* (f_{162}) measured in $\times 0.1$ milivolts, *Amplitude of S wave* (f_{163}) measured in $\times 0.1$ milivolts, *Amplitude of R' wave* (f_{164}) measured in $\times 0.1$ milivolts, *Amplitude of S' wave* (f_{165}) measured in $\times 0.1$ milivolts, *Amplitude of P wave* (f_{166}) measured in $\times 0.1$ milivolts, *Amplitude of T wave* (f_{167}) measured in $\times 0.1$ milivolts, *QRSA* (f_{168}) which is the sum of the areas of all segments divided by 10, *QRSTA* (f_{169}) which is equal to $QRSA + 0.5 \times$ width of Twave $\times\, 0.1 \times$ height of Twave. The above 9 features measured for the **DI** channel are all measured for the **DII** (from feature f_{170} to f_{179}), **DIII** (from feature f_{180} to f_{189}), **AVR** (from feature f_{190} to f_{199}), **AVL** (from feature f_{200} to f_{209}), **AVF** (from feature f_{210} to f_{219}), **V1** (from feature f_{220} to f_{229}), **V2** (from feature f_{230} to f_{239}), **V3** (from feature f_{240} to f_{249}), **V4** (from feature f_{250} to f_{259}), **V5** (from feature f_{260} to f_{269}), and **V6** (from feature f_{270} to f_{279}) channels.

There are several missing feature values. Class distribution of this dataset is very unfair and instances of classes 11, 12, and 13 do not exist in the current dataset. Class 01 (normal) is the most frequent one. Although the ECG of some patients show the characteristics of more than one Arrhythmia, in constructing the dataset it is assumed that no patient has more than one cardiac Arrhythmia.

Breast Cancer: Breast Cancer data set contains 273 patient records. All the patients underwent a surgery to remove tumors, all of them were followed up five years later. The objective here is to predict whether or not breast

cancer would recur during that five year period. The recurrence rate is about 30 %, and hence such prognosis is important for determining post-operational treatment. The data set contains nine variables that were measured, including both numeric and binary values. The prediction is binary: either the patient did suffer a recurrence of cancer or not.

Cleveland and Hungarian Data: Both datasets are about the heart disease diagnosis. Each dataset is described with same features. Cleveland data was collected from the Cleveland Clinic Foundation and Hungarian data was collected from the Hungarian Institute of Cardiology.

These databases contain 76 attributes originally, but in ML field 13 of them is used. All attributes are numeric valued and 6 of them have nominal values. The class is determined according to the presence of heart disease, that is, this is binary classification problem. There are no missing values in these datasets for the features that we have used.

Dermatology: The differential diagnosis of erythemato-squamous diseases is a real problem in Dermatology. They all share the clinical features of erythema and scaling, with very little differences. The diseases in this group are *psoriasis* (C_1), *seboreic dermatitis* (C_2), *lichen planus* (C_3), *pityriasis rosea* (C_4), *cronic dermatitis* (C_5), and *pityriasis rubra pilaris* (C_6). Usually a biopsy is necessary for the diagnosis but unfortunately these diseases share many histopathological features as well. Another difficulty for the differential diagnosis is that a disease may show the features of another disease at the beginning stage and may have the characteristic features at the following stages. Patients were first evaluated clinically with 12 features which are *erythema* (f_1), *scaling* (f_2), *definite borders* (f_3), *itching* (f_4), *koebner phenomenon* (f_5), *polygonal papules* (f_6), *follicular papules* (f_7), *oral mucosal involvement* (f_8), *knee and elbow involvement* (f_9), *scalp involvement* (f_{10}), *family history* (f_{11}), and *age* (f_{34}). Afterwards, skin samples were taken for the evaluation of 22 histopathological features which are *melanin incontinence*(f_{12}), *eosinophils in the infiltrate* (f_{13}), *PNL infiltrate* (f_{14}), *fibrosis of the papillary dermis* (f_{15}), *exocytosis* (f_{16}), *acanthosis* (f_{17}), *hyperkeratosis* (f_{18}), *parakeratosis* (f_{19}), *clubbing of the rete ridges* (f_{20}), *elongation of the rete ridges* (f_{21}), *thinning of the suprapapillary epidermis* (f_{22}), *spongiform pustule* (f_{23}), *munro microabcess* (f_{24}), *focal*

hypergranulosis (f_{25}), *disappearance of the granular layer* (f_{26}), *vacuolisation and damage of basal layer* (f_{27}), *spongiosis* (f_{28}), *saw-tooth appearance of retes* (f_{29}), *follicular horn plug* (f_{30}), *perifollicular parakeratosis* (f_{31}), *inflammatory monoluclear inflitrate* (f_{32}), and *band-like infiltrate* (f_{33}). The values of the histopathological features are determined by an analysis of the samples under a microscope.

In the dataset constructed for this domain, the family history feature has the value 1 if any of these diseases has been observed in the family, and 0 otherwise. The age feature simply represents the age of the patient. Every other feature (clinical and histopathological) was given a degree in the range of 0 to 3. Here, 0 indicates that the feature was not present, 3 indicates the largest amount possible, and 1, 2 indicate the relative intermediate values.

Diabetes: This data set contains diabetes diseases collected from National Institute of Diabetes and Digestive and Kidney Diseases. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The population lives near Phoenix, Arizona, USA. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The data set contains records of 768 patients with 8 features.

Glass Data: This dataset consists of attributes of glass samples taken from the scan of an accident. The Glass dataset contains 214 instances of which belongs to one of six classes. In this dataset there are 9 features. All feature values are continuous.

Horse Data: In this dataset there are 368 instances. Number of attributes is 22 and the number of classes is 2. Seven of these features are linear and fifteen of them are nominal. The 24% of the feature values is missing (unknown). The features V3, V25, V26, V27, and V28 are deleted from the original Horse-colic (called Horse in this thesis) dataset and feature V24 is used as the class.

Ionosphere Data: The radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. *Good* radar returns are those showing evidence of some type of structure in the ionosphere. *Bad* returns are those that do not; their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

Iris Flowers: Iris flowers dataset from Fisher [26] consists of four integer valued continuous features and a particular species of iris flower. There are three different classes: *iris virginica, iris setosa, iris versicolor*. The four attributes measured were sepal length, sepal width, petal length and petal width. The dataset contains 150 instances, 50 instances of each three classes.

Liver: This data set contains 345 instances and collected by BUPA Medical Research Ltd. Each instance constitutes the record of a single male individual. There are 6 attributes and the first 5 variables are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption. The last attribute presents drinks number of half-pint equivalents of alcoholic beverages drunk per day. The purpose of this data set is to determine whether patient has liver disorders or not.

Musk: This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, the low-energy conformations of the molecules were generated and then filtered to remove highly similar conformations. This left 476 conformations. Then, a feature vector was extracted that describes each conformation.

This many-to-one relationship between feature vectors and molecules is called the "multiple instance problem". When learning a classifier for this data, the classifier should classify a molecule as musk if ANY of its conformations is classified as a musk. A molecule should be classified as non-musk if NONE of its conformations is classified as a musk.

Page-blocks: The problem consists in classifying all the blocks of the page layout of a document that has been detected by a segmentation process. This is an essential step in document analysis in order to separate text from graphic areas. Indeed, the five classes are: *text* (1), *horizontal line* (2), *picture* (3), *vertical line* (4) and *graphic* (5). It is compiled by Donato Malerba at the University of Bari. This dataset is one of the largest datasets in the UCI Repository. This dataset is abbreviated as page in this thesis.

Segmentation: This is an *image segmentation* data compiled by the Vision Group at the University of Massachusetts.The instances were drawn randomly from a database of 7 outdoor images. The images were handsegmented to create a classification for every pixel. Each instance is a 3x3 region. The classes are *brickface*, *sky*, *foliage*, *cement*, *window*, *path*, *grass*.

Wine Data: This dataset is about recognizing wine types. This data is provided by Pharmaceutical and Food analysis and technologies. The classes are separable. In a classification context, this is a well-posed problem with "well behaved" class structures. This dataset is the result of the chemical analysis of wines grown in the same region in Italy but derived from three different cultures. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The dataset contains 178 instances. All features are linear.