BATCH LEARNING OF DISJOINT 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

Aynur Akkuş

September, 1996

ABSTRACT

BATCH LEARNING OF DISJOINT FEATURE INTERVALS

Aynur Akkuş
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Halil Altay Güvenir
September, 1996

This thesis presents several learning algorithms for multi-concept descriptions in the form of disjoint feature intervals, called *Feature Interval Learning* algorithms (FIL). These algorithms are batch supervised inductive learning algorithms, and use feature projections of the training instances for the representation of the classification knowledge induced. These projections can be generalized into disjoint feature intervals. Therefore, the concept description learned is a set of disjoint intervals separately for each feature. The classification of an unseen instance is based on the weighted majority voting among the local predictions of features. In order to handle noisy instances, several extensions are developed by placing weights to intervals rather than features. Empirical evaluation of the FIL algorithms is presented and compared with some other similar classification algorithms. Although the FIL algorithms achieve comparable accuracies with other algorithms, their average running times are much more less than the others.

This thesis also presents a new adaptation of the well-known k-NN classification algorithm to the feature projections approach, called k-NNFP for *k-Nearest Neighbor on Feature Projections*, based on a majority voting on individual classifications made by the projections of the training set on each feature and compares with the k-NN algorithm on some real-world and artificial datasets.

Keywords: machine learning, supervised learning, inductive learning, batch learning, feature projections, voting.

ÖZET

AYRIK ÖZNİTELİK BÖLÜNTÜLERİNİ TOPLU ÖĞRENME

Aynur Akkuş
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Halil Altay Güvenir
Eylül, 1996

Bu tezde öznitelik izdüşümlerine dayalı yeni öğrenme algoritmaları sunulmuştur. Öznitelik Bölüntülerini Öğrenme (FIL) olarak isimlendirilen bu algoritmalar toplu, denetimli ve tümevarımsal öğrenme yöntemlerini kullanırlar ve öğrenme örneklerinin öznitelik izdüşümlerini sınıflama bilgisini çıkarmak için kullanırlar. Bu izdüşümler ayrık öznitelik bölüntülerine genellenir. Böylece, öğrenilen kavram tanımları her öznitelik için ayrık öznitelik bölüntüleri şeklinde gösterilir. Daha önce görülmemiş bir örneğin sınıflandırması için her öznitelik tarafından bir ön sınıflandırma yapılır ve son sınıflama bu ön sınıflandırmaların ağırlıklı çoğunluk oylamasıyla belirlenir. Hatalı örnekleri tespit edebilmek için bölüntülere ağırlık verilerek bazı değişiklikler önerilmiştir. FIL algoritmalarının benzer sistemlerle uygulama sonuçları doğal ve yapay veri kümeleri üzerinde karşılaştırılmıştır. Bu algoritmaların doğruluk oranları daha öncekilere yakın olmasına rağmen ortalama çalışma süreleri çok daha azdır.

Bu tezde literatürde yaygın olarak bilinen k en yakın komşu sınıflandırma algoritması (k-NN) yeniden tanımlanmıştır ve k-NNFP, öznitelik izdüşümleri üzerinde k en yakın komşu sınıflandırması, olarak isimlendirilmiştir. k-NNFP algoritmasında sınıflandırma her öznitelikten gelecek olan tahminler arasından çoğunluk oylaması yapılarak belirlenir. k-NNFP ve k-NN algoritmalarının karşılaştırılması doğal ve yapay veri kümeleri üzerinde yapılmıştır.

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

ACKNOWLEDGMENTS

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

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

I cannot fully express my gratitude and thanks to Savaş Dayanık and my parents for their morale support and encouragement.

I would also like to thank to Bilge Say, Yücel Saygın, Gülşen Demiröz and Halime Büyükyıldız for their friendship and support.

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

CFP	: Classification by Feature Partitioning
COFI	: Classification by Overlapping Feature Intervals
C_i	: Label of the ith class
C4	: Decision tree algorithm
C4.5	: Decision tree algorithm
D_f	: Generalization distance for feature f in the COFI algorithm
D_{EH}	: Euclidean distance between example E and exemplar H
E	: An example
E_f	: fth feature value of the example E
f_j	: jth 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
GA-CFP	: Hybrid CFP Algorithm
H	: Hyperrectangle
H_f	: fth 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
HFP	: Homogeneous Feature Projections Feature Weighting Method
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
k	: no of neighbors in k-NN and k-NNFP

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
n	: Number of features in the dataset
NBC	: Naive Bayesian Classifier
NGE	: Nested-Generalized Exemplars
NN	: Nearest Neighbor Algorithm
$p(\mathbf{x}\|w_j)$	: 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 $\mathbf{x}$
$R(\alpha_i, \mathbf{x})$	: Conditional risk
SFA	: Single Feature Accuracy Feature Weighting Method
T2	: Agnostic PAC learning decision tree algorithm with at most two levels
$\mathbf{x}$	: Instance vector
$\mathbf{x}_i$	: Value vector of ith instance
w_f	: Weight of feature f
w_H	: Weight of exemplar H
1R	: System whose input is training examples and output is 1-rule
$\triangle$	: Weight adjustment rate of the CFP algorithm
$\lambda(\alpha_i, w_j)$	: Loss incurred for taking action α_i when the state of nature is w_j

Chapter 1

Introduction

Machine learning has played a central role in artificial intelligence since 1980's, especially in modeling behavior of human cognition and human thought processes for problem solving strategies. The studies in machine learning suggest computational algorithms and analyses of such algorithms that suggest explanations for capabilities and limitations of human cognition. Learning can be described as increasing the knowledge or skills in accomplishing certain tasks [13]. The learner applies inferences in order to construct an appropriate representation of some relevant reality.

One of the fundamental research problems in machine learning is how to learn from examples since it it usually possible to obtain a set of examples to learn from. From a set of training examples, each labeled with its correct class name, a machine learns by forming or selecting a generalization of the training examples. This process, also known as supervised learning, is useful for real-world classification tasks, e.g. disease diagnosis, and problem solving tasks in which control decisions depend on classification. Inductive learning refers to learning from examples in which knowledge is acquired by drawing inductive inference from the examples given. Acquiring knowledge involves operations of generalizing, specializing, transforming, correcting and refining knowledge representations [42, 43].

Many of the tasks to which machine learning techniques are applied are tasks that humans can perform quite well. However, humans often cannot tell how they solve these tasks. Inductive supervised learning is able to exploit the human ability to assign labels to given instances without requiring humans to explicitly formulate rules that do the same. These training instances are then analyzed by inductive supervised algorithms to learn specific tasks.

There are several different methods by which a human (or a machine) can acquire knowledge [43]:

- rote learning (learning by being programmed)
- learning from instruction (learning by being told)
- learning from teacher provided examples (concept acquisition)
- learning by observing the environment and making discoveries (learning from observation and discovery)

In this thesis, we will concern with concept acquisition. Concept acquisition can be defined as the task of learning a description of a given concept from a set of examples and counterexamples of that concept [13, 43]. Examples are represented usually by input vectors of feature values and their corresponding class labels. Concept descriptions are then learned as relations among the given set of feature values and the class labels.

The ability to classify is another important facet of intelligence. The task of a classification algorithm is to predict correctly the class of an unseen test example from a set of labeled training examples or classification knowledge learned by a concept acquisition algorithm. Many supervised learning algorithms have been developed to perform classification [5, 10, 28, 52, 58]. Classification systems require only a minimal domain theory and are based on training instances to learn an appropriate classification function.

One of the central problems in classifying objects is distinguishing features that are relevant to the target concept from that are irrelevant. Many researchers have addressed the issue of feature weighting in order to reduce the impact of irrelevant features and to increase the impact of more relevant features in classification tasks, by investigating feature weighting [2], and feature subset selection [38, 61]. Some classification systems give equal importance to all features. However, in real life, the relevance of features may not all be the same. The algorithms which assign equal weights to all features are more sensitive to the presence of irrelevant features. In order to prevent the intrusive effect of irrelevant features, feature subset selection approaches are utilized in which the space of subsets of feature sets are considered to determine the relevant and irrelevant features. As a simple example, the learning algorithm is run on the training data with different subsets of features, using cross-validation to estimate its accuracy with each subset. These estimates are used as an evaluation metric for directing search through the space of feature sets [6, 29, 38, 61]. On the other hand, the disadvantage of using feature selection method is that it treats features as completely relevant or irrelevant. In reality, the degree of relevance may not be just 0 or 1, but any value between them.

Knowledge representation in exemplar-based learning models are either representative instances [2, 5], or hyperrectangles [58, 59]. For example, instance-based learning model retains examples in memory as points, and never changes them. The only decisions that are made are what points to store and how to measure similarity. Several variants of this model have been developed [2, 3, 4, 5]. Nested generalized-exemplars model represents the learned knowledge as hyperrectangles [58, 59]. This model changes the point storage model of the instance-based learning and retains examples in the memory as axis-parallel hyperrectangles.

The Classification by Feature Partitioning [27, 28, 65], and Classification with Overlapping Feature Intervals [67] algorithms are also exemplar-based learning algorithms based on generalized feature values. They are incremental inductive supervised learning algorithms. Their basic knowledge representation is based on feature projections. Classification knowledge in these algorithms is represented as sets of disjoint and overlapping feature intervals, respectively. The classification of an unseen test example is determined through a weighted voting scheme on classifications based on the individual feature predictions. Feature projections for knowledge representation allows faster classification than other exemplar-based learning models since these projections can be organized for faster classification. Another important advantage of this representation is that it allows easy handling of missing feature values by simply ignoring them. The major drawback of this knowledge representation is that descriptions involving a conjunction between two or more features cannot be represented. However, the reported results show that both techniques are successful by processing each feature separately [27, 28, 65, 67]. This thesis will investigate that whether it is possible to obtain more accurate concept descriptions in the form of disjoint feature intervals when they are learned in the batch (non-incremental) mode.

As a preliminary work to this thesis, we have studied classification of objects on feature projections in a batch mode [7]. Classification in this method is based on a majority voting on individual classifications made by the projections of the training set on each feature. We have applied the k-nearest neighbor algorithm to determine the classifications made on individual feature projections. We called the resulting algorithm k-NNFP, for *k-Nearest Neighbor on Feature Projections*. The nearest neighbor (NN) algorithm stores all training instances in memory as points and classifies an unseen instance as the class of the nearest neighbor in the n-dimensional Euclidean space where n is the number of features. The extended form of the NN algorithm to reduce the effect of the noisy instances is the k-NN algorithm in which classification is based on a majority voting among k nearest neighbors. The most important characteristic of the k-NNFP algorithm is that the training instances are stored as their projections on each feature dimension. This allows the classification of a new instance to be made much faster than the k-NN algorithm. The voting mechanism reduces the intrusive effect of possible irrelevant features in classification. Furthermore, the classification accuracy of the k-NNFP algorithm increases when the value of k is increased, which indicates that the process of classification can incorporate the learned classification knowledge better when k increases.

First, we treated all features as equivalent in the k-NNFP algorithm. However, all features need not have equal relevance. In order to determine the relevances of features, the best method is to assign them weights. In this thesis, we propose two methods for learning feature weights for the learning algorithms whose knowledge representation is feature projections. The first method is based on homogeneities of feature projections, called homogeneous feature projections, for which the number of consequent values of feature projections of a same class supports an evidence for increasing the probability of correct classification in the learning algorithm that uses feature projections as the basis of learning. The second method is based on the accuracies of individual features, called single feature accuracy. In this approach, the learning algorithm is run on the basis of a single feature, once for each feature. The resulting accuracy is taken as the weight of that feature since it is a measure of contribution to classification for that feature. The first empirical evaluation of these feature weighting methods on real world datasets will be investigated in the k-NNFP algorithm in Section 3.4. These methods can be also applied to other learning algorithms which use feature weights.

In this thesis, we focused on the problem of learning multi-concept descriptions in the form of disjoint feature intervals following a batch learning strategy. We designed and implemented several batch algorithms for learning of disjoint feature intervals. The resulting algorithms are called *Feature Intervals Learning* algorithms (FIL). These algorithms are batch inductive supervised learning algorithms. Several modifications are made to the initial FIL algorithm, FI1, to investigate whether improvement for this method is possible or not. Although the FIL algorithms achieve comparable accuracies with the earlier classification algorithms, the average running times of the FIL algorithms are much less than those.

The FIL algorithms learn the projections of the concept descriptions over each feature dimension from a set of training examples. The knowledge representation of the FIL algorithms is based on feature projections. The projections of training instances are stored in memory, separately in each feature dimension. Concepts are represented as disjoint intervals for each feature. In the basic FIL algorithm, 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 finally the class label is the associated class of the interval.

In the FIL algorithms, each feature makes its local prediction by simply searching through the feature intervals containing that feature value of the test instance. The final prediction is based on the weighted majority voting among local predictions of features. The voting mechanism reduces the negative effect of possible irrelevant features in classification. Since FIL algorithms treat each feature separately, they do not use any similarity metric among instances for prediction unlike other exemplar-based models that are similarity-based algorithms. This allows the classification of a new instance to be made much faster than similarity-based classification algorithms.

Since induction of multi-concept descriptions from classified examples have large number of applications to real-world problems, we will evaluate FIL algorithms on some real-world datasets from the UCI-Repository [47]. 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 illnesses.

In summary, the primary contributions of this thesis can be listed as follows:

- We formalized the concept of feature projections for knowledge representation in inductive supervised learning algorithms.
- We applied this representation to classical NN algorithm, compared k-NN and k-NNFP (the k-NN that uses feature projections). We should note that the disadvantage of this representation does not affect the classification of real-world datasets.
- We presented several batch learning methods of disjoint feature intervals for assigning weights to features and intervals. We also presented two feature weight learning methods.
- We started the construction of two new medical datasets as an application area, and a test bed for ML algorithms.

This thesis presents and evaluates several batch learning methods in the form of disjoint feature intervals that use feature projections for knowledge representation. In the next chapter, a summary of the previous concept learning models are presented. In Chapter 3, feature projections for knowledge representation are discussed and some prior research is explained in detail. The details of the FIL algorithms are described in Chapter 4. The construction of feature intervals on a feature dimension and classification process is illustrated through examples, and several extensions of basic FIL algorithm are described. Complexity analysis and empirical evaluation of FIL algorithms are studied in Chapter 5. Performance of the FIL algorithms on artificially generated data sets and comparisons with other similar techniques on real-world data sets are also presented. The final chapter presents a summary of the results obtained from the experiments in this thesis. Also an overview of possible extensions to the work presented here is given as future work.

Chapter 2

Concept Learning Models

The symbolic empirical learning has been the most active research area in machine learning for developing concept descriptions from concept examples. These methods use empirical induction which is falsity-preserving rather than truth-preserving inference. Therefore the results of these methods are generally hypotheses which need to be validated by further experiments.

Inductive leaning is the process of acquiring knowledge by drawing inductive inferences from teacher or environment-provided facts by generalizing, specializing, transforming, correcting and refining knowledge representations [43]. There are two major types of inductive learning: *learning from examples* (concept acquisition) and *learning from observation* (descriptive learning). In thesis, we will concern ourselves with concept acquisition rather than descriptive generalization, which is the process of determining a general concept description (a law, a theory) characterizing a collection of observations. In concept acquisition, observational statements are characterizations of some objects preclassified by a teacher into one or more classes (concepts). Induced concept description can be viewed as a concept recognition rule, in that, if an object satisfies this rule, then it belongs to the given concept [43].

A *characteristic description* of a class of objects (conjunctive generalization) is typically a conjunction of some simple properties common to all objects in the class. Such descriptions are intended to discriminate the given class from all other possible classes. On the other hand, a *discriminant description* specifies one or more ways to distinguish the given class from a fixed number of other classes.

Given a set of instances which are described in terms of feature values from a predefined range, the task of concept acquisition is to induce general concept descriptions from those instances. Concept descriptions are learned as a relation among the given set of feature values and the class labels. The two types of concept learning are *single concept learning* and *multiple-concept learning*.

In single concept learning one can distinguish two cases:

1. Learning from "positive" instances only.
2. Learning from "positive" and "negative" examples (examples and counterexamples of the concept).

In *multiple-concept learning* one can also distinguish two cases:

1. Instances do not belong to more than one class, that is, classifications of instances are mutually disjoint.
2. Instances may belong to more than one class, that is, classifications of instances are possibly overlapping.

For concept learning tasks, one of the widely used representation technique is the *exemplar-based* representation. Either representative instances or generalizations of instances form concept descriptions [5, 58]. Another useful knowledge representation technique for concept learning is decision trees [52]. Statistical concept learning algorithms also use training instances to induce concept descriptions based on certain probabilistic approaches [21]. In the following sections, these concept learning models are presented.

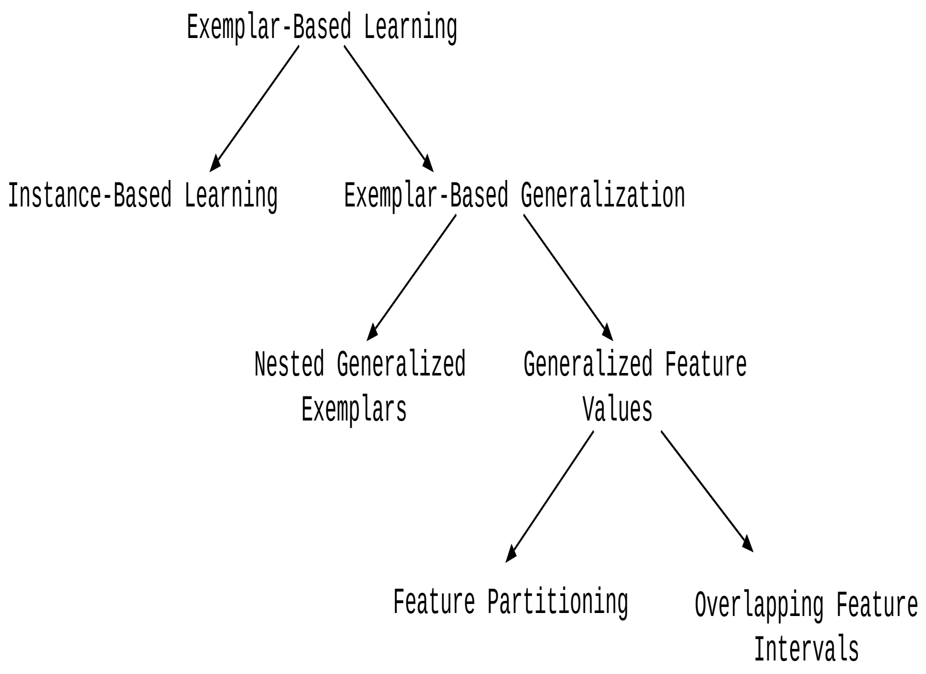

Exemplar-Based Learning
Instance-Based Learning
Exemplar-Based Generalization
Nested Generalized Exemplars
Generalized Feature Values
Feature Partitioning
Overlapping Feature Intervals

Figure 2.1. Classification of exemplar-based learning models.

2.1 Exemplar-Based Learning

Exemplar-based learning was originally proposed as a model of human learning by Medin and Schaffer [41]. 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 [58, 59], or generalized values [27, 28, 67]. Unlike explanation-based generalization (EBG) [18, 45], 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 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 [58]. This model changes the point storage model of the instance-based learning and retains examples in the memory as axis-parallel hyperrectangles. Generalized Feature Values learning models can be classified as exemplar-based generalization, such as NGE. The examples of GFV learning models are the Classification by Feature Partitioning (CFP), and the Classification by Overlapping Feature Intervals (COFI). In the CFP algorithm, examples are stored as disjoint intervals on each feature dimension. In the COFI algorithm, concept descriptions are represented in the form of overlapping feature intervals. In this thesis, we will study several batch learning methods whose knowledge representation is in the form of disjoint feature intervals that can be also categorized as GFV method. In the following sections, we will describe IBL, NGE, and GFV methods briefly. GFV methods that use feature projections for knowledge representation will be discussed in detail in Chapter 3 since this knowledge representation motivated us to develop this thesis.

2.1.1 Instance-Based Learning (IBL)

Instance-based learning algorithms represent concept descriptions as a set of stored instances, called exemplars, and with some information concerning their past performances during classification [5, 8]. These algorithms extend the classical nearest neighbor algorithm, which has large storage requirements [16, 17]. All examples are represented as points on the n-dimensional Euclidean space, where n is the number of features. The concept descriptions can change 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

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

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

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.

Extensions of these three algorithms [1, 3] are developed to remove some limitations which occur because of certain assumptions. For example, concepts are often assumed to

- be defined with respect to the same set of relevant features,
- be disjoint in instance space, and
- have uniform instance distributions.

To study the effect of relevances of features in IBL algorithms, IB4 has been proposed by Aha [3]. In this study, feature weights are learned being dependent on 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 (NGE)

Nested-generalized exemplar (NGE) theory is a variation of exemplar-based learning [58]. 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 implements NGE in a program called EACH (Exemplar-Aided Constructor of Hyperrectangles) [59]. 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 n-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 [58]. 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) 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 E and H, where E is a new data point and H is the hyperrectangle, 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 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.

The original NGE was designed for continues features only. Symbolic features require a modification of the distance and area computations for NGE.

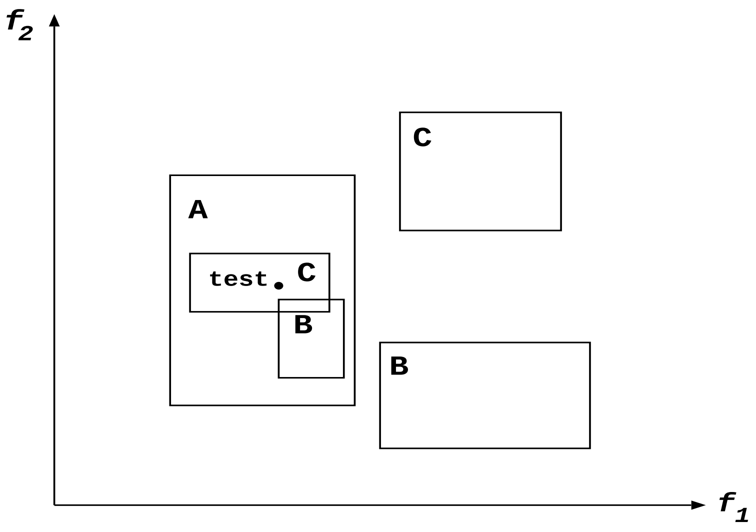


Figure 2.2. An example concept description of the EACH algorithm in a domain with two features.

In Figure 2.2, an example concept description of EACH algorithm is presented for 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. It is seen that rectangle A contains two rectangles, B and C, in its region. Therefore, B and C are *exceptions* in 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 has smaller, so the prediction will be the class value C for this test instance.

2.1.3 Generalized Feature Values

The previously presented techniques categorized as generalized feature values under exemplar-based generalization are the CFP [27, 28, 65], COFI [67], and k-NNFP [7] algorithms. Briefly, the CFP and COFI algorithms are incremental algorithms based on feature partitioning and overlapping feature intervals, respectively. They use feature projections as the basis of learning. Classification of unseen instances are based on voting among the individually predictions of features. The discussion of the CFP and COFI algorithms are presented in Chapter 3 in more detail (Section 3.1 and 3.2).

2.2 Decision Trees

Decision trees are one of the most well known and widely used approaches for learning from examples. This method was developed initially by Hunt, Marin and Stone [31], and later modified by Quinlan [49, 50]. Quinlan's ID3 [52] and C4.5 [55] are the most popular algorithms in decision tree induction. Initially, ID3 algorithm has applied to deterministic domains such as chess and games [49, 50]. Later, ID3 algorithm has extended to cope with noisy and uncertain instances rather than being deterministic [52].

Decision tree algorithms represents concept descriptions in the form of tree structure. Decision tree algorithms begin with a set of instances and create a tree data structure that can be used to classify new instances. Each instance is described by a set of feature values, which can have either continuous or symbolic (nominal) values, with the corresponding classification. Each internal node of a decision tree contains a test which indicates which branch to follow from that node. The leaf nodes contain class labels instead of tests. A new test instance is classified by using the class label stored at the leaf node.

Decision tree methods use divide and conquer approach. Each internal node must contain a test that will partition the training instances. The most important decision criteria in decision tree induction is how to decide the best test. ID3, and its successor C4.5 use information-theoretic metrics to evaluate the goodness of a test; in particular they choose the test that extracts the maximum amount of information from a set of instances, given the constraint that only one feature will be tested.

The recursive partitioning method of constructing decision trees continues to subdivide the set of training instances until each subset in the partition contains instances of a single class, or until no tests offer any further improvement. The result 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*. There are basically two ways in which the recursive partitioning method can be modified to produce simpler trees: deciding not to divide a set of training instances any further, or removing retrospectively some of the structure built up by recursive partitioning [55].

The former approach, sometimes called *stopping* or *prepruning*, has the advantage that time is not wasted in assembling structures that are not used in the final simplified tree. The typical approach is to look at the best way of splitting a subset and to assess the split from the point of view of statistical significance, information gain, error reduction. If this assessment falls below some threshold then the division is rejected.

Later, a simple decision tree approach, called 1R system, is proposed by Holte [30]. 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* [30].

The input of the 1R algorithm is a set of training instances. The output is concept descriptions in the form of 1-rule. The 1R system can be treated as a special case of generalized feature values methods. These methods consider all features information whereas the 1R system uses only one feature. 1R tries to partition feature values into several disjoint feature intervals. Since each feature is considered separately in 1R system, missing feature values can be simply ignored instead of ignoring the instance containing missing value. The FIL algorithms presented in this thesis also partition feature dimensions into disjoint intervals. However, the FIL algorithms make final predictions based on majority voting on individual classifications of all features rather than one feature as in 1R system. During the training phase of the 1R system, disjoint feature intervals are constructed on each feature dimension. Then, one of the concept descriptions on a feature is chosen as final concept descriptions, 1-rules, by selecting the one that makes the smallest error on the training dataset.

Holte used sixteen datasets to compare 1R and C4 [52], and fourteen of the datasets were selected from the collection of UCI-Repository [47] [30]. The main result of comparing 1R and C4 was an insight into the tradeoff between simplicity and accuracy. 1R rules are only a little less accurate (about 3 percentage points) than C4's pruned decision trees on almost all of the datasets. Decision trees formed by C4 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.

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.

2.3 Statistical Concept Learning

Statistical concept learning has been extensively studied for induction problems [21, 25, 69]. The main goal is to determine the classification of a given instance based on parametric or nonparametric techniques. The decision-making processes of humans are somewhat related to the recognition of patterns. For example the next move in chess game is based upon the present pattern on the board, and buying or selling stocks is decided by a complex pattern of information [25]. The goal of the pattern recognition is to clarify these complicated mechanisms of decision-making processes and to automate these functions using computers. Several pattern recognition methods, either parametric or nonparametric, have been presented in the literature [20, 21, 25, 69].

Bayesian classifier originating from work in pattern recognition is a probabilistic approach to inductive learning. This method estimates the (posterior) probability that an instance belongs to a class, given the observed feature values for the instance. The classification is determined by the highest estimated posterior probability [21, 25]. Bayesian classifiers assume that features are statistically dependent. On the other hand, Naive Bayesian classifier is one of the most common parametric classifiers assuming independence of features.

When no parametric structure can be assumed for the density functions, nonparametric techniques, for instance nearest neighbor method, must be used for classifications [21, 25]. The nearest neighbor method is one of the simplest methods conceptually, and is commonly cited as a basis of comparison with other methods. It is often used in case-based reasoning [62].

This section is devoted to statistical concept learning methods because they have similarities to the FIL algorithms developed in this thesis First, Bayes Decision Theory and Naive Bayesian Classifiers will be explained. Then, nearest neighbor methods with some variants will be discussed. Finally, a new version of k nearest neighbor algorithm, k-NNFP, based on feature projections will be briefly mentioned, and discussed in detail in Chapter 3 by comparing k nearest nearest neighbor techniques. In Chapter 5, the FIL algorithms will be compared with these statistical methods.

2.3.1 Bayes Decision Theory - Naive Bayesian Classifier (NBC)

The goal of the Bayesian classification is to determine the *a posteriori* probabilities $P(C_j|\mathbf{x})$ where C_j is the class and $\mathbf{x}$ is the instance to be classified. An instance $\mathbf{x}$=$< x_1, x_2, ...x_n >$ is a vector of feature values where n is the number of features. The *a priori* probability $P(C_j)$ and the conditional densities $P(\mathbf{x}|C_j)$ allows the use of Bayes rule to compute $P(C_j|\mathbf{x})$.

Let $\Omega = \{C_1, C_2, .., C_k\}$ be the finite set of k states of nature. Here each C_j corresponds to a class in our terminology. Let the feature vector $\mathbf{x}$ be a vector-valued random variable, and let $p(\mathbf{x}|C_j)$ be the state-conditional probability density function for $\mathbf{x}$, that is, the probability density function for $\mathbf{x}$ conditioned on C_j being the state of nature. Finally, let $P(C_j)$ be the *a priori* probability that nature is in the state C_j. That is, $P(C_j)$ is the proportion of all instances of class j in the training set. Then the *a posteriori* probability $P(C_j|\mathbf{x})$ can be computed from $p(\mathbf{x}|C_j)$ by Bayes rule [21]:

$$P(C_j|\mathbf{x}) = \frac{p(\mathbf{x}|C_j)P(C_j)}{p(\mathbf{x})} \tag{2.5}$$

where

$$p(\mathbf{x}) = \sum_{j=1}^{k} p(\mathbf{x}|C_j)P(C_j). \tag{2.6}$$

Let $A = \{\alpha_1, \alpha_2, .., \alpha_a\}$ be the finite set of a possible actions. Let $\lambda(\alpha_i, C_j)$ be the loss incurred for taking action α_i when the state of nature is C_j. Since $P(C_j|\mathbf{x})$ is the probability that the true state of nature is C_j, the expected loss associated with taking action α_i is

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{k} \lambda(\alpha_i|C_j)P(C_j|\mathbf{x}). \tag{2.7}$$

In decision theoretic terminology, an expected loss is called *risk*, and $R(\alpha_i|\mathbf{x})$ is known as the *conditional risk*. Whenever we encounter a particular observation $\mathbf{x}$, we can minimize our expected loss by selecting the action that minimizes the conditional risk. Now, the problem is to find a Bayes decision rule against $P(C_j)$ that minimizes the overall risk. A decision rule is a function $\alpha(\mathbf{x})$ that tells us which action to take for every possible observation. That is, for every $\mathbf{x}$, the decision function $\alpha(\mathbf{x})$ assumes one of the a values $\alpha_1, \alpha_2, .., \alpha_a$. The overall risk R is the expected loss associated with a given decision rule. To minimize the overall risk, we compute the conditional risk for $i = 1, .., a$ and select the action α_i for which $R(\alpha_i|\mathbf{x})$ is minimum. The resulting minimum overall risk is called the *Bayes risk* and is the best performance that can be achieved.

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{k} \lambda(\alpha_i|C_j)P(C_j|\mathbf{x}) \tag{2.8}$$

The probability of error is the key parameter in pattern recognition. There are many ways to estimate error for Bayesian classifiers. One of them is minimizing it. For example, if action α_i is taken and the true state of nature is C_j, then decision is correct if $i = j$, and in error if $i \neq j$. A loss function for this case, called *zero-one loss function* is:

$$\alpha(i,j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \end{cases} \tag{2.9}$$

The conditional risk becomes

$$R(\alpha_i|\mathbf{x}) = \sum_{j \neq i} P(C_j|\mathbf{x}) \tag{2.10}$$

$$R(\alpha_i|\mathbf{x}) = 1 - P(C_i|\mathbf{x}) \tag{2.11}$$

Note that $P(C_i|\mathbf{x})$ is the conditional probability that action α_i is correct. To minimize the average probability of error, one should maximize the a posteriori probability $P(C_j|\mathbf{x})$. For minimum error rate:

Decide C_i if $P(C_i|\mathbf{x}) > P(C_j|\mathbf{x})$ for all $j \neq i$.

In summary, a Bayesian classifier classifies a new instance by applying Bayes' rule to determine the probability of each class given the instance,

$$P(C_j|\mathbf{x}) = \frac{p(\mathbf{x}|C_j)P(C_j)}{\sum_i p(\mathbf{x}|C_i)P(C_i)} \tag{2.12}$$

The denominator sums over all classes and where $P(\mathbf{x}|C_j)$ is the probability of the instance $\mathbf{x}$ given the class C_j. After calculating these quantities for each class, the algorithm assigns the instance to the class with the highest probability. In order to make this expression operational, one must specify how to compute $P(\mathbf{x}|C_j)$. The Naive Bayesian Classifier (NBC) assumes independence of features within each class, allowing the following equality

$$P(\mathbf{x}|C_j) = \prod_{f=1}^{n} P(x_f|C_j). \tag{2.13}$$

An analysis of Bayesian classifier has been presented [36]. Also a method, called Selective Bayesian Classifier, has been proposed [37] to overcome the limitation of the Bayesian classifier for sensitivity to correlated features. Since NBC considers each feature independently, this will form a basis for comparison with the FIL algorithms. The experimental results of these comparisons will be presented in Chapter 5.

2.3.2 Nearest Neighbor Classifiers (NN)

One of the most common classification techniques is the nearest neighbor (NN) algorithm. In the literature, nearest neighbor algorithms for learning from examples have been studied extensively [17, 21]. Aha et al. have demonstrated that instance-based learning and nearest neighbor methods often work as well as other sophisticated machine learning techniques [5].

The NN classification algorithm is based on the assumption that examples which are closer in the instance space are of the same class. An example is represented as a vector of feature values plus class label. 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 of the class of the nearest neighbor among all stored training instances. Although several distance metrics have been proposed for NN algorithms [60], the most common metric is the Euclidean distance metric. Instances are represented as a vector of feature values plus class label. The Euclidean distance between two instances $x =< x_1, x_2, ..., x_n, C_x >$ and $y =< y_1, y_2, ...y_n, C_y >$ on an n dimensional space is computed as:

$$dist(x,y) = \sqrt{\sum_{f=1}^{n} diff(f,x,y)^2} \tag{2.14}$$

$$diff(f,x,y) = \begin{cases} |x_f - y_f| & \text{if } f \text{ is linear} \\ 0 & \text{if f is nominal} \quad \text{and } x_f = y_f \\ 1 & \text{if f is nominal} \quad \text{and } x_f \neq y_f \end{cases} \tag{2.15}$$

Here $diff(f,x,y)$ 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.

Although several techniques have been developed for handling unknown (missing) feature values [54, 55], the most common approach is to set them to the mean value of the values on corresponding feature.

Stanfill and Waltz introduced the Value Difference Metric (VDM) to define the similarity for symbolic-valued (nominal) features and empirically demonstrated its benefits [62]. The VDM computes a distance for each pair of the different values a symbolic feature can assume. It essentially compares the relative frequencies of each pair of symbolic 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 present a nearest neighbor algorithm that uses a modification of VDM, called MVDM (Modified Value Difference Metric) [15]. 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 [56].

A generalization of the nearest neighbor algorithm, k-NN, classifies a new instance by a majority voting among its k (≥ 1) nearest neighbors using some distance metrics in order to prevent the intrusive effect of noisy training instances. This 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. Kelly and Davis introduced WKNN, the *weighted* k-NN algorithm, and GA-WKNN, a *genetic algorithm* that learns feature weights for WKNN algorithm [33]. Assigning variable weights to the features of the instances before applying the k-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 important features becomes more critical than similarity with respect to irrelevant features. The study for weighting features in k-NN algorithm has shown that for the best performance the votes of the k nearest neighbors of a test example should be weighted in inverse proportion to their distances from the test example [70].

An experimental comparison of the NN and NGE (*Nested Generalized Exemplars*, a Nearest-Hyperrectangle algorithm) has been presented by Wettschereck and Dietterich [71]. NGE and several extensions of it are found to give predictions that are substantially inferior to those given by k-NN in a variety of domains. 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 [48]. 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.

2.3.3 NN Classifier on Feature Projections (NNFP)

Another statistical approach is a new version of the k-NN classification algorithm proposed in this thesis, which uses feature projections of training instances for classification knowledge [7]. The classification of an unseen instance is based on a majority voting on individual classifications made by the projections of the training set on each feature. We have applied the k-nearest neighbor algorithm to determine the classifications made on individual feature projections. We called the resulting algorithm k-NNFP, for k-Nearest Neighbor on Feature Projections. The classification knowledge is represented in the form of projections of the training data on each feature dimension. This allows the classification of a new instance to be made much faster than k-NN algorithm. The voting mechanism reduces the intrusive effect of possible irrelevant features in classification. The k-NNFP algorithm is discussed in detail in Section 3.3.

Chapter 3

Feature Projections for Knowledge Representation

In this chapter, feature projections for knowledge representation are discussed in detail. Given a set of training instances with correct class labels, knowledge for representation of a concept description (or classification) is maintained as the projections of the training set on each feature dimension separately. The most important advantage of this representation is that the projections of the feature values can be sorted for each feature, and this 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. The rationale behind this knowledge representation is that humans maintain knowledge in this form, especially in medical domains. An example for this approach is presented, called CRiteria Learning System [66]. It aims to learn decision rules in the form of criteria tables as humans do. One of the shortcomings of feature projections is that descriptions involving a conjunction between two or more features can not be represented.

This chapter discusses the CFP, COFI, and k-NNFP algorithms that use feature projections for knowledge representation. Briefly, the CFP and COFI algorithms are based on feature partitioning and overlapping feature intervals, respectively. The most important property of these algorithms is that they both consider each feature separately in an incremental manner. The reported results show that both techniques are successful by processing each feature separately [27, 28, 65, 67]. The encouraging results of the CFP and COFI algorithms motivated us for further investigation of feature projections as a form of knowledge representation from a different point of view. We think that more accurate results can be obtained from these techniques if a batch learning strategy is followed. After the discussion of the CFP and COFI algorithms, a new version of the classical k-NN algorithm which treats instances as feature projections rather than points, called k-NNFP (k Nearest Neighbor on Feature Projections) is presented. Next, an extension to it by weighting features for weighted-voting is presented.

3.1 Classification by Feature Partitioning (CFP)

The CFP algorithm is a method for learning from examples that uses feature projections for knowledge representation [27, 28, 65]. It is an incremental supervised inductive learning algorithm where instances are stored by their feature projections over each feature dimension. An instance is represented as a vector of feature values plus a label that encodes the class of the instance. In the training phase, disjoint feature intervals of concept definitions are constructed by generalization and specialization. An interval is a basic unit of knowledge representation in this algorithm. 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 through generalization with other neighboring points in the same feature dimension. In order to avoid overgeneralization, a parameter, called generalization limit (D_f), is given. Before generalizing an interval on a feature dimension f to cover a new point, the distance between interval and the new point must be less than D_f. Otherwise, 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, 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.

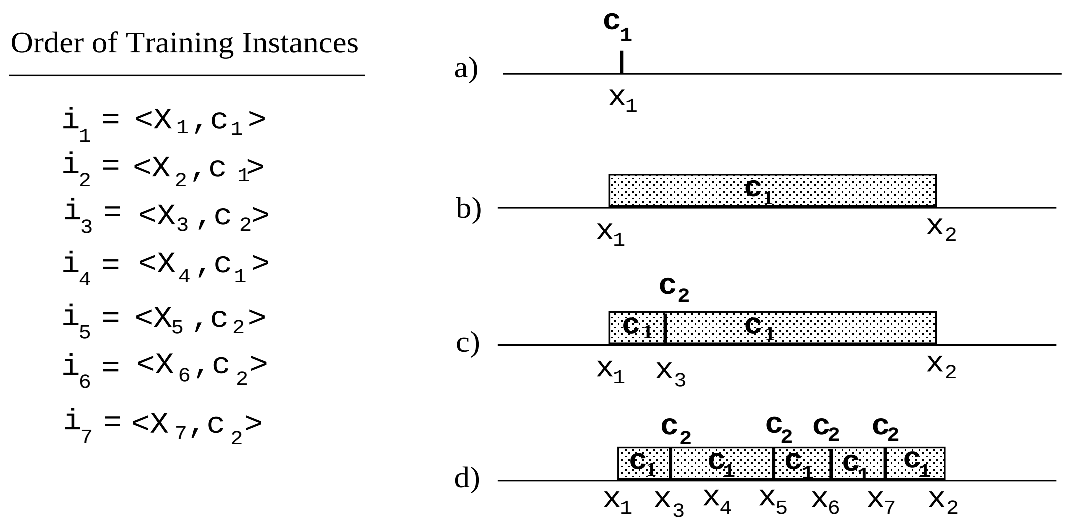


Figure 3.1. Construction of intervals in the CFP algorithm: (a) after i_1 is processed, (b) after i_2 is processed, (c) after i_3 is processed, (d) after all training instances are processed.

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 forms a point interval at the feature value x_1 on this feature dimension. After the second instance, 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 one 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 subpartitioned 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 weighted majority voting among local predictions of features.

In the CFP algorithm, feature intervals are constructed as disjoint set 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 handle noisy values [64, 65] and determine the domain dependent parameters (D_f and $\triangle$) of the CFP algorithm [27].

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 [65]. 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

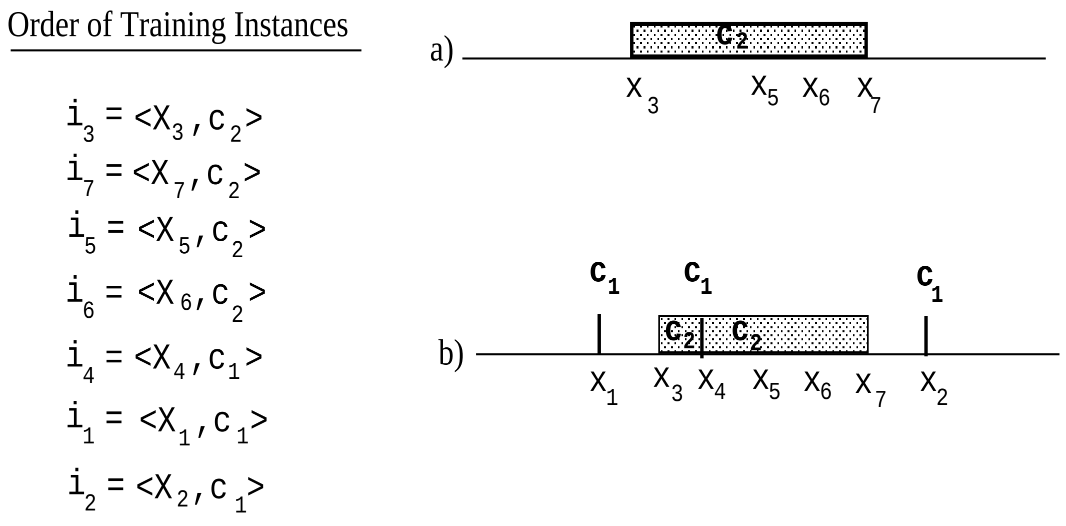


Figure 3.2. Construction of intervals in the CFP algorithm by changing the order of the training instances. Note that here the same set of instances in Figure 3.1., but in a different order, is used as the training set: (a) after i_3,i_7, i_5 and i_6 are processed, (b) after all instances are processed.

algorithm with the CFP algorithm has been developed [27]. 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. 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 computations than the 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

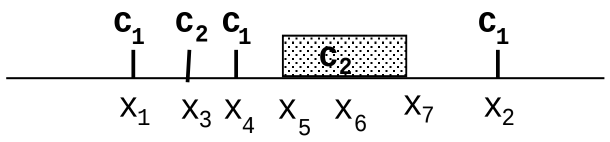


Figure 3.3. Construction of the intervals in the FIL algorithms with using the same dataset as used in Figure 3.1 and Figure 3.2.

it affects the resulting concept descriptions considerably. The different concept descriptions can classify a test instance as different classes. For example, the test instance $< x_8, C_1 >$ 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 offer a solution to this problem, by constructing intervals in a batch mode, that is, seeing all the training instances at once, and then processing them. Therefore, they construct intervals as independent of the order of training instances. Since all training instances are known initially, all feature values are sorted on each feature dimension in the form of point intervals. Then, neighboring same class points are generalized to form range intervals. Feature values at which there exist more than one class remain as point intervals. The concept description learned by the FIL algorithms from the same set of training instances is shown in Figure 3.3, independent of the order of the training instances. In addition to sensitivity to the order of training instances, the CFP algorithm overgeneralizes intervals as in 3.1c. In this case, intervals of concept C_1 are formed between point intervals of concept C_2. However, one might expect that the range $[x_5, x_7]$ should belong to class C_2.

3.2 Classification with Overlapping Feature Intervals (COFI)

The COFI algorithm is another exemplar-based concept learning algorithm that uses feature projections to generalize knowledge. It is an inductive supervised learning algorithm. Classification knowledge learned is maintained

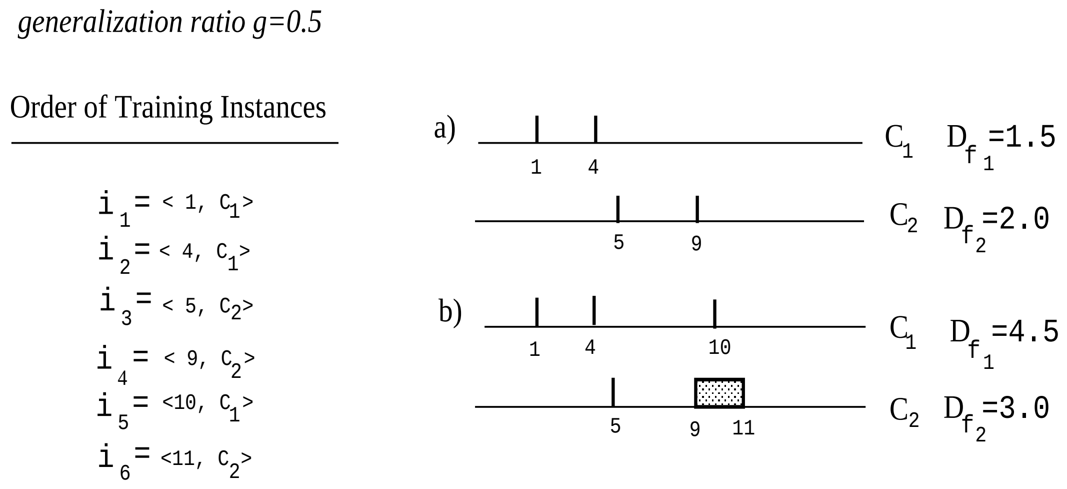


Figure 3.4. An example of construction of intervals in the COFI algorithm: (a) after i_1, i_2, i_3 and i_4 are processed, (b) after i_5 and i_6 are processed.

in the form of overlapping feature intervals. 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.

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. Basic unit of the representation is *interval* as in the CFP algorithm. 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 found as follows:

$$D_f = (current\_max_f - current\_min_f) * g. \qquad (3.1)$$

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 ration in the range $[0, 1]$. They 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 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, 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 last training instances' value 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

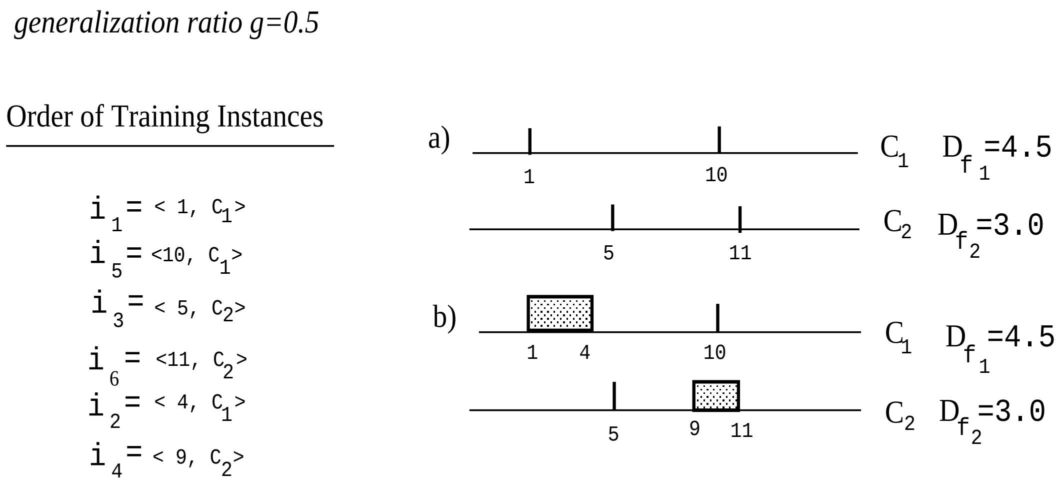


Figure 3.5. An example of construction of intervals in the COFI algorithm using the same set of training instances as in Figure 3.6, but in a different order: a) after i_1, i_5, i_3, and i_6 are processed, b) after i_2 and i_4 are processed.

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 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.

The FIL algorithms construct disjoint feature intervals from the same training instances as independent of the order of the training instances, as seen from the Figure 3.6. Since all feature values are known initially, the intervals

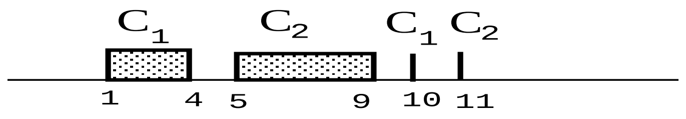


Figure 3.6. An example of construction of intervals in the FIL Algorithms using the same set of training instances as in Figure 3.4 and Figure 3.5.

constructed during the training process are unique whatever the order of the training instances is.

3.3 K Nearest Neighbor Classification on Feature Projections (k-NNFP)

In this section, a new approach to classification is presented, which is based on a majority voting on individual classifications made by the projections of the training set on each feature [7]. We have applied the k-nearest neighbor algorithm to determine the classifications made on individual feature projections. We called the resulting algorithm k-NNFP, for k-Nearest Neighbor on Feature Projections.

The classification knowledge is represented in the form of projections of the training data on each feature dimension. The classification of an instance is based on a voting taken on the classifications made on the basis of individual feature projections.

In Chapter 2, a brief introduction to k-NN algorithm and its several extensions were given. In the next subsection, the k-NNFP algorithm is described. Section 3.3.2 presents the complexity analysis and empirical evaluation of the k-NNFP and k-NN algorithms. Finally, Section 3.2.3 presents a summary of the k-NNFP algorithm and its applicability.

3.3.1 The k-NNFP Algorithm

This section presents the k-NNFP algorithm, a new classification based on feature projections using k nearest neighbor algorithm. First, the description of the algorithm is given. Then the algorithm is explained through an example dataset. Later, the behavior of the algorithm on datasets with irrelevant features will be given.

3.3.1.1 Description of the k-NNFP Algorithm

The implementation of the algorithm given here is non-incremental, namely, all training instances are taken and processed at once. An instance **x** is represented as **x**=$< x_1, x_2, ..x_n, C_x >$ where x_is $(1 \leq in)$ are the feature values and C_x is the corresponding class label. An important characteristic of this algorithm is that instances are stored as their projections on each feature dimension. 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, we use the k-NN algorithm on that single dimension. 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. Also note that, since each feature is processed separately,

```
classify($t$,$k$)
/* $t$: test instance, $k$: number of neighbors */
begin
    for each class $c$
        vote[$c$] = 0
    for each feature $f$
        /* put $k$ nearest neighbors of test instance $t$
                on feature $f$ into $Bag$ */
        $Bag$ = kBag($f, t, k$)
        for each class $c$
            vote[$c$] = vote[$c$] + count($c, Bag$);
    $prediction$ = UNDETERMINED
    for each class $c$
        if vote[$c$] > vote[$prediction$] then
            $prediction = c$
    return $prediction$
end.
```

Figure 3.7. Classification in the k-NNFP algorithm.

no normalization of feature values is needed.

The k-NNFP algorithm is outlined in Figure 3.7. All the projections of training instances on linear features are stored in memory as sorted values. In Figure 3.7, the votes of a feature is computed by the function $kBag(f, t, k)$, which returns a bag of size k containing the classes of the k nearest training instances to the instance t on feature f. The distance between the values on a feature dimension is computed using $diff(f, x, y)$ metric as follows:

$$diff(f,x,y) = \begin{cases} |x_f - y_f| & \text{if } f \text{ is linear} \\ 0 & \text{if } f \text{ is nominal and } x_f = y_f \\ 1 & \text{if } f \text{ is nominal and } x_f \neq y_f \end{cases} \tag{3.2}$$

Note that the bag returned by $kBag(f, t, k)$ does not contain any *UNDETERMINED* class as long as there are at least k training instances whose f values are known. Then, the number of votes for each class is incremented by the number of votes that a feature gives to that class, which is determined by the *count* function. The value of the function $count(c,\ Bag)$ is the number of occurrences of class c in bag Bag.

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.

3.3.1.2 An Example

In order to describe the classification in the k-NNFP algorithm, consider the sample training dataset in Figure 3.8. In this dataset, the feature f_0 is the only relevant feature, and f_1 is an irrelevant feature. There are three instances of each class A, B, and C in the training set. Let the test instance ($< 5, 5 >$) be of class B.

For the test instance in Figure 3.8, the k-NN classification, $kBag$ values and final prediction for the k-NNFP algorithm are given in Table 3.1. As seen in Table 3.1, the k-NN algorithm will classify the test instance as C if $k = 1$, as C or A if $k = 2$, as C, A or B if $k = 3$, and as C if $k = 4$. On the other hand, the k-NNFP algorithm will classify the test instance correctly if $k > 1$. This example shows that the k-NNFP algorithm will be unaffected in the presence of irrelevant features.

3.3.1.3 Handling Irrelevant Features

The conclusion about the irrelevant features from the previous example can be generalized. For an irrelevant feature f, the number of occurrences of a class C in a bag returned by $kBag(f, t, k)$ is proportional to the number of instances of class C in the training set. If there are equal number of instances of each class in the training set, than the votes of an irrelevant feature will be equal for each class, and the final prediction will be determined by the votes of the relevant features. If the training instances are not equally distributed

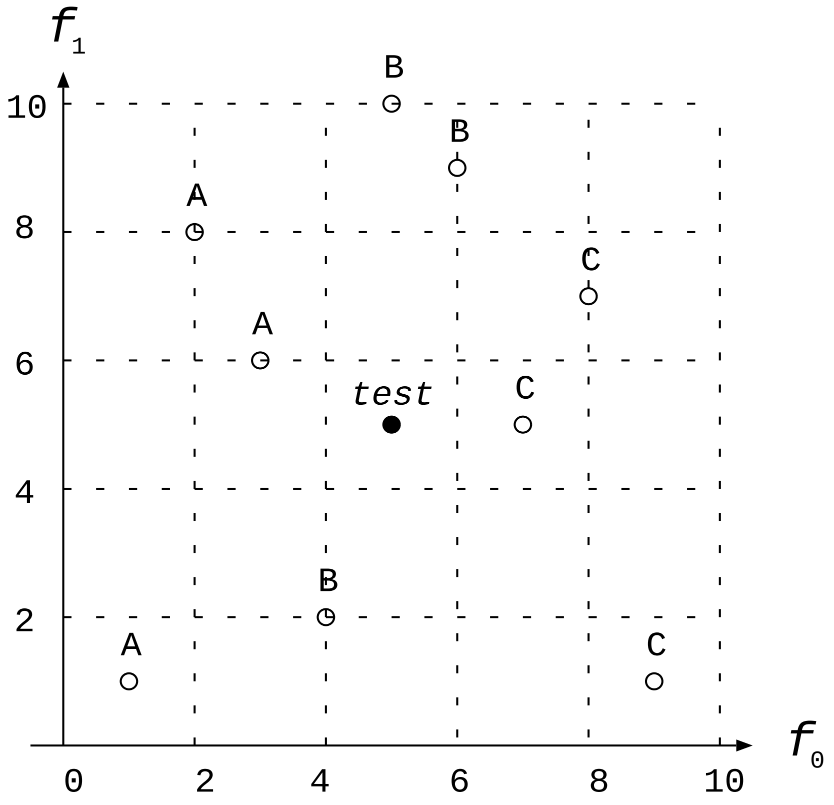


Figure 3.8. A sample training dataset and a test instance.

among the classes, then the votes of an irrelevant feature will be for the most frequently occurring class.

Table 3.1. For the test instance ($< 5, 5 >$) in Figure 2 the k-NN classification, kBag values and final prediction of the k-NNFP algorithm.

		k-NNFP			
k	k-NN	f_0	f_1	Sum of Votes	Prediction
1	[C]	[B]	[C]	[B,C]	B or C
2	[C,A]	[B,B]	[C,A]	[A,B,B,C]	B
3	[C,A,B]	[B,B,B]	[C,A,C]	[A,B,B,B,C,C]	B
4	[C,A,B,C]	[B,B,B,A]	[C,A,C,B]	[A,A,B,B,B,B,C,C]	B

3.3.1.4 Handling Missing Feature Values

The k-NNFP algorithm handles unknown (missing) feature values by simply not taking them into account. During batch training and classification, the features containing missing values are simply ignored. This is a natural approach because in real life if nothing is known about a feature, it is usually ignored. If all class dimensions give no prediction, then no prediction can be made and the resulting prediction for the class is UNDETERMINED. This is an unexpected case since at least one feature value should be known.

3.3.2 Evaluation of the k-NNFP Algorithm

Several measures of performance are possible. One performance measure of a classification algorithm is its classification accuracy. For supervised concept learning tasks, the most commonly used classification accuracy metric is the percentage of correctly classified instances over all test instances for a given dataset. The other performance measures are time and space complexities. In this section, the training and classification complexities of the k-NNFP and the k-NN algorithms are given. Next, an empirical evaluation of the algorithm is presented along with its comparison with the k-NN algorithm in terms of classification accuracy for increasing values of k and running time.

3.3.2.1 Complexity Analyses

Since all the training instances are stored in the memory in both k-NN and k-NNFP algorithms, the space required for training with m instances on a domain with n features is proportional to $m \cdot n$. That is, the space complexities of these algorithms are $O(m \cdot n)$.

In the training, all instances are stored on each feature dimension as their feature projections. And then they are sorted once at the end. Since the sorting of m feature values has the time complexity of $O(m \log m)$ For a dataset containing m instances and n features the training time complexity of the k-NNFP is $O(n \cdot m \cdot \log m)$. On the other hand, the k-NN algorithm has the time complexity of $O(m \cdot n)$ for storing all instances in memory.

The $kBag(f, t, k)$ function, to determine the votes of a feature, first finds the nearest neighbor of t on f and then next $k - 1$ neighbors around the nearest neighbor. The time complexity of this process is $O(\log m + k)$. The final classification requires the votes of each of n features. Therefore, the average classification time complexity of the k-NNFP algorithm is $O(n \cdot (k + \log m))$.

On the other hand, in the k-NN algorithm, the classification of a test instance requires the computation of its distance to m training instance on n dimensions. Time complexity of computing the distance between two instances is $O(n)$. So, computing the distance to m training instances is $O(m \cdot n)$. Sorting m instances according to their distances is $O(m \log m)$. Therefore, the classification time complexity of a single instance in the k-NN algorithm is $O(m(n + \log m))$, assuming $m >> k$.

3.3.2.2 Empirical Evaluation

Here, an empirical evaluation of the k-NNFP algorithm on both real-world data sets and artificially generated datasets is presented in order to show the effect of irrelevant features on the classification accuracy. The results will be compared with that of the k-NN algorithm.

Experiments with Real-World Datasets

The k-NNFP and k-NN algorithms are evaluated on some real-world datasets which are widely used in the machine learning field, therefore comparisons will be possible with other similar methods in future. The real-world datasets are selected from the UCI-Repository [47]. An overview of the datasets is given in Appendix A, and they are briefly explained.

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 this study, we chose the 5-fold cross-validation technique. That is, the whole dataset is partitioned into 5 subsets. The four of the subsets is used as the training set, and the fifth is used as the test set, and this process is repeated 5 times once for each subset being the test set. Therefore, each instance appears once in the test set, and four times in the training set. Classification accuracy is the average of these 5 runs.

Table 3.2. Accuracy (%) and average running time (msec) of the k-NNFP algorithm on real-world datasets.

Data Set:	bcancerw	cleveland	glass	hungarian	ionosphere	iris	liver	musk	wine
k=1	94.00	67.62	57.00	70.04	88.04	90.00	50.44	69.54	79.7
k=2	94.56	72.28	62.14	70.70	88.02	92.00	53.92	71.40	90.4
k=3	94.88	72.94	61.18	75.84	88.02	91.34	55.68	70.76	90.96
k=4	95.72	77.56	60.74	73.80	87.46	92.64	58.84	71.40	93.24
k=5	96.16	78.88	60.72	76.16	87.46	91.30	58.26	71.22	93.24
k=6	96.00	77.86	63.54	72.76	87.78	91.88	61.16	69.96	95.48
k=7	96.00	79.52	62.58	74.80	87.74	92.00	61.46	70.36	95.48
k=8	96.14	79.18	63.98	73.76	86.90	92.66	61.76	69.96	96.04
k=9	96.14	78.52	63.04	75.80	87.44	92.00	62.04	69.52	96.62
k=10	96.14	78.86	64.90	72.76	87.46	94.02	62.90	69.10	96.62
Avg. Time	340	740	94	266	477	40	1022	2654	282

Table 3.3. Accuracy (%) and average running time (msec) of the k-NN algorithm on real-world datasets.

Data Set:	bcancerw	cleveland	glass	hungarian	ionosphere	iris	liver	musk	wine
k=1	95.00	80.52	68.66	75.50	84.62	93.98	63.48	73.10	94.40
k=2	93.84	80.20	67.70	79.54	88.06	94.00	60.58	77.54	94.42
k=3	96.28	82.50	66.76	81.58	83.78	94.68	66.66	70.18	96.60
k=4	95.72	82.84	68.14	80.92	85.20	94.00	62.60	74.16	94.38
k=5	96.58	83.80	66.30	82.26	83.20	94.66	64.92	67.88	96.04
k=6	96.56	82.82	67.24	83.64	83.76	95.32	61.46	69.14	96.08
k=7	96.26	82.50	65.36	83.28	82.34	94.66	64.64	65.58	96.04
k=8	95.86	82.16	65.36	83.62	84.06	94.66	64.36	67.86	95.48
k=9	95.56	82.82	65.34	82.94	82.62	94.66	67.54	65.16	96.04
k=10	95.70	81.48	63.96	83.96	84.06	94.66	63.20	67.86	96.06
Avg. Time	3216	7786	318	695	2335	105	2060	18520	615

Table 3.4. The average time (in msec) required to train with 80% and test with the 20% of the artificial datasets for increasing number of features.

	Number of features					
	4	6	8	10	12	14
k-NNFP	85	212	285	367	517	556
k-NN	365	937	1257	1335	1472	1720

The accuracy of the k-NNFP in Table 3.2 and k-NN in Table 3.3 were obtained for the specified datasets for $k = 1, 2, \ldots 10$. These experiments show that the classification accuracy of the k-NNFP algorithm usually increases when the value of k increases. This suggests that the k-NNFP algorithm can exploit the knowledge represented in the form of feature projections for higher values of k. On the other hand, increase in the value of k does not result in a parallel increase in the accuracy of the k-NN algorithm. Langley and Sage's works on NN classifiers suggest that many of the UCI datasets have few irrelevant features, if any. Our experimental results also support this claim.

Experiments on Artificial Data

As illustrated through an example in subsection 3.3.1, the k-NNFP algorithm is, in general, unaffected from the presence of irrelevant features in the dataset. Experiments with artificial datasets have important roles to play in the study of irrelevant features. Hence, in order to empirically prove this claim, we have generated six datasets with increasing number of irrelevant features from zero to ten. Each of the datasets contain four relevant features, three classes with 100 instances each. A class is represented by a hyperrectangle in four (relevant) dimensional space, the values for irrelevant features are randomly generated. We have conducted 5-fold cross-validation experiments on these six datasets, and compared the results of k-NNFP and k-NN algorithms. The accuracy results are plotted in Figure 3.9.

As seen from these results, the decrease in the accuracy of the k-NNFP algorithm when the number of irrelevant features increase is much less than that of the k-NN algorithm. Also we observed that the accuracy of the k-NNFP algorithm increases parallel to the increase in the value of k, whereas the accuracy of the k-NN algorithm is not correlated with increase in the value of k.

The time required to train the k-NNFP and the k-NN algorithms with the 80% of the data and test with the remaining 20% for these datasets are given in Table 3.4. The comparison of the running times in this table agrees with the time complexity analysis of these algorithms given in Section 3.3.2.1.

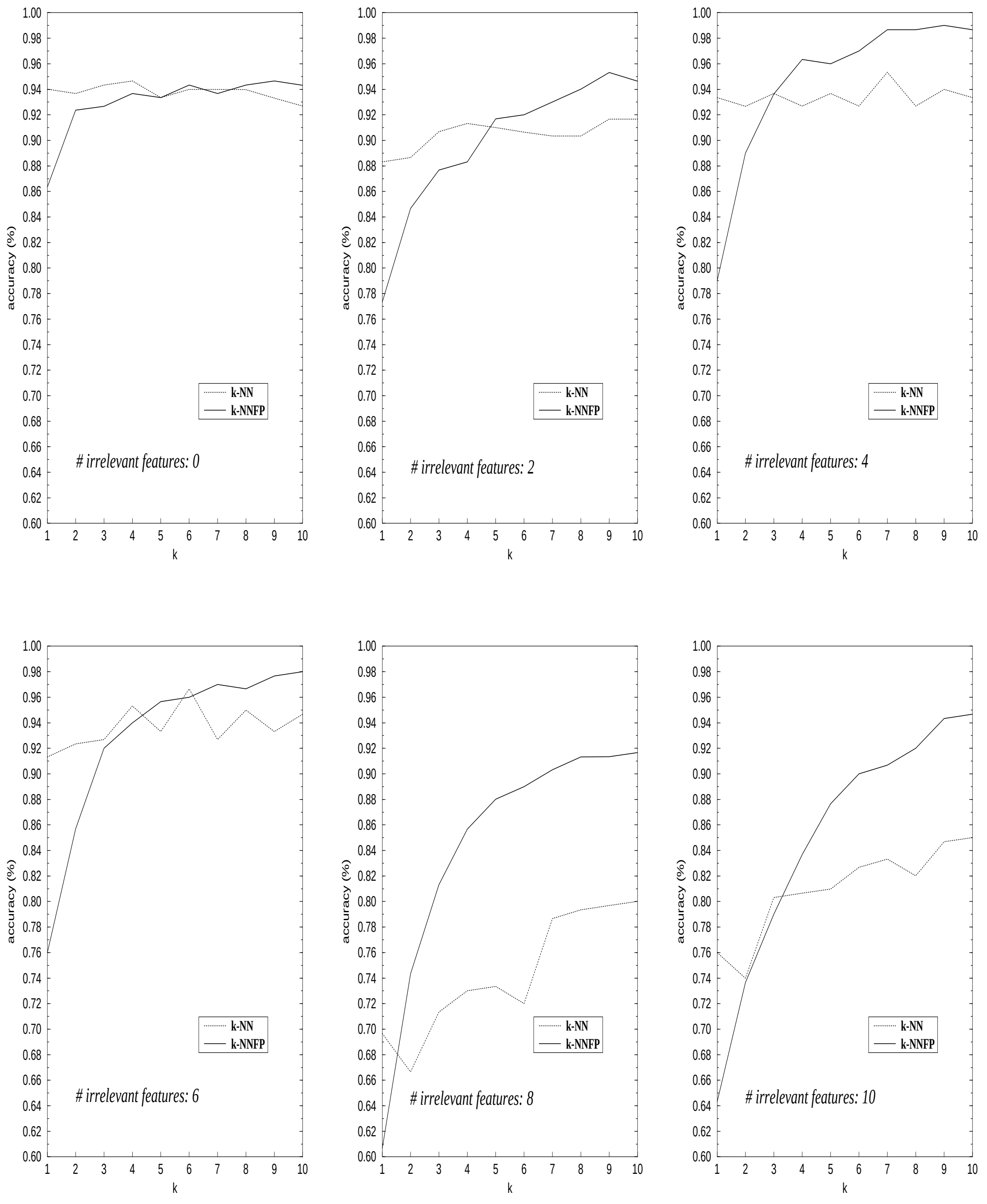


Figure 3.9. Comparison of k-NN and k-NNFP on artificial datasets for increasing value of k. In all datasets there are 4 relevant features, 3 classes and 100 instances for each class. The accuracy results are obtained by 5 way cross-validation.

3.3.3 Discussion

A new form of classification method, called k-NNFP, has been presented. This algorithm has been compared with the k-NN algorithm in terms of classification accuracy and time complexity on both real-world and artificially generated datasets.

In the k-NNFP algorithm, the classification knowledge is represented in the form of sets of projections of the training data separately on each feature dimension. The classification of an instance is based on a majority voting taken on the classifications made on the basis of individual feature projections. Since each feature is processed separately, there is no need for normalization of feature values. Also, for the same reason, the algorithm can simply ignore any missing feature values that may appear both in training and test instances. The effect of the missing and noisy feature values on the prediction accuracy of the k-NNFP algorithm will be investigated as a future work. As another direction for future work, we plan to integrate a feature weight learning algorithm to k-NNFP.

The k-NNFP algorithm is based on the assumption that each feature can contribute the classification process and the majority voting provides a correct classification when data contain many irrelevant features. The k-NNFP algorithm can provide better classification accuracy than k-NN algorithm when a dataset contains many irrelevant features with respect to relevant ones. This claim has been justified on artificially generated datasets. On real-world datasets, 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 k-NNFP algorithm treats feature values independently, whereas the k-NN algorithm treats all instances as points in n-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.

3.4 Weighting Features in k Nearest Neighbor Classification on Feature Projections (k-NNFP)

We propose two methods for learning feature weights to improve the classification accuracy of the k-NNFP algorithm. The classification of unseen examples are made on the basis of feature projections by a majority voting among the k (≥ 1) predictions of each feature separately. We have treated all features as equivalent in this algorithm (Section 3.3). However, all features may not have equal relevance, even some features may be completely irrelevant. In order to determine features' relevances, the best method is to assign them weights. The first method is based on the homogeneity of feature projections for which the number of consequent values of feature projections of a same class supports an evidence for increasing the probability of correct classification in the k-NNFP algorithm. We called this method HFP (Homogeneous Feature Projections). The second method is based on the individual accuracies of features. We called this method SFA (Single Feature Accuracy). In this approach, the k-NNFP algorithm is run on the basis of a single feature, once for each feature. The resulting accuracy is taken as the weight of that feature since it is a measure of contribution to classification for that feature. Empirical evaluation of these feature weighting methods in the k-NNFP algorithm on real world datasets is given.

These feature weighting methods aim to investigate the effect of weight assigning to features in k-NNFP algorithm. In these methods, no domain-specific knowledge is used. These methods can be categorized according to Wettshereck and Aha's five-dimensional framework's first dimension [72] as ignorant and feedback, respectively, since homogeneity of feature projections weight setting does not use any feedback from the k-NNFP algorithm whereas the second one uses feedback from k-NNFP algorithm. These methods modify the voting mechanism of k-NNFP algorithm by incrementing the vote of the predicted class by using the feature weight. These feature weighting methods can be easily incorporated into other classification algorithms that use feature weights.

In this study, we aim to investigate the importance of features' contribution to final classification since to assign higher weights to more relevant features increase the reliability of voting. This study focused on the empirical evaluations of feature weighting methods proposed on real-world datasets.

Comparison of similar algorithms highlights dissimilarities that can explain observed performance differences. Our experimental results show that weighting features in the k-NNFP algorithm improves the accuracy effectively in some real-world datasets, especially for smaller k values. An explanation of observed performance differences is presented in the third subsection.

In the next subsection, the k-NNFP algorithm is given with its weighted version, briefly. In the subsequent subsection, a detailed descriptions of feature weighting methods studied are given. The third subsection presents the empirical comparison of these methods on real-world datasets taken from the UCI-Repository [47]. The last subsection presents a summary of these feature weighting methods.

3.4.1 The Weighted k-NNFP Algorithm

In Section 3.3, the k-NNFP algorithm was introduced for classification based on feature projections using k nearest neighbor algorithm. Since all feature values are treated separately, there is no need for normalization of feature values. In the learning phase, each training instance is stored 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. The k-NNFP algorithm stores the feature projections of training instances in a sorted order. Therefore, the classification of a new instance requires a simple search of the nearest training instance values on each feature. The classification of an instance is based on a majority voting taken on the classifications made on the basis of individual feature projections. In general, with the majority voting for final classification, the effect of irrelevant features may be reduced. On the other hand, each feature can contribute to the classification by its relevance. So, if we place weights on features before voting, this can provide more accurate result for final class by reflecting each feature's relevance in the classification. The weighted k-NNFP algorithm is outlined in Figure 3.7. This algorithm

```
classify($t$, $k$)
/* $t$: test instance, $k$: number of neighbors */
begin
    for each class $c$
        vote[$c$] = 0
    for each feature $f$
        /* put $k$ nearest neighbors of test instance $t$
                on feature $f$ into $Bag$ */
        $Bag$ = kBag($f, t, k$)
        /* each feature contributes proportional ot its weight */
        for each class $c$
            vote[$c$] = vote[$c$] + weight[$f$] * count($c, Bag$);
    $prediction$ = UNDETERMINED
    for each class $c$
        if vote[$c$] > vote[$prediction$] then
            $prediction = c$
    return $prediction$
end.
```

Figure 3.10. Classification in the weighted k-NNFP algorithm.

was explained in Section 3.2.1.1. Here, the number of votes for each class is incremented by multiplying the weight of that feature by number of votes that a feature gives to that class, which is determined by the *count* function. The value of *count*(c, Bag) is the number of occurrences of class c in bag Bag.

3.4.2 Some Methods for Learning Feature Weights

Two feature weighting methods are proposed for k-NNFP algorithm to see the effect of irrelevant, and relevant features with relative relevancies. Firstly, the homogenity of feature projections method is discussed. Next, the second method which is based on the single feature accuracy is presented.

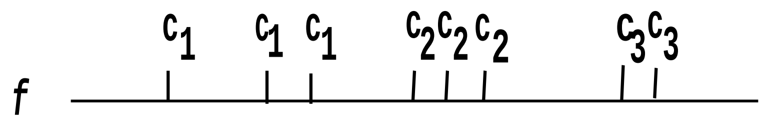


Figure 3.11. Homogeneous distribution on a feature dimension

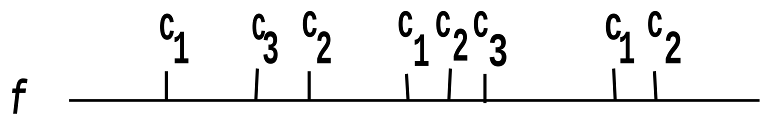


Figure 3.12. Heterogeneous distribution on a feature dimension

3.4.2.1 Weight Learning Based on Homogenity of Projections

The basic motivation for this method comes from the k-NNFP algorithm itself. The assumption of the k-NNFP algorithm is that closer values on a feature dimension are of the same class, distribution of trainig instances on a feature dimension is homogenious. That is, the projections of all training instances of the same class are grouped together. Figure 3.11 and Figure 3.12 illustrates homogeneous and heterogeneous feature projections, respectively. In homogeneous feature projections, the total number of consequent values of a same class can give a measure for its relevancy for classification prediction. In k-NNFP algorithm, all seen feature values are stored in memory as sorted. We can determine the weight of a feature as follows: Initially, a count is set to 0, then for all sorted feature values, if the consequent feature value's class is same as the previous one, then count is incremented. Therefore, feature weight can be found by dividing that count by the total number of distinct feature values on that feature. This can be summarized as follows:

$$w_f = \frac{\sum_{v=1}^{V_f} \alpha(f,v)}{V_f} \tag{3.3}$$

$$\alpha(f,v) = \begin{cases} 1 & \text{if } C_{v,f} = C_{v+1,f} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

All feature weights are computed using this formula. Here $C_{v,f}$ denotes the class label of v^{th} value on feature dimension f, and V_f denotes the number of distinct values on feature dimension f. This equation always gives a value for a feature between 0 and 1, so it can be the probability of correct classification for that feature. These are incorporated with feature weights to allow that more important features contribute to classification process more effectively.

3.4.2.2 Weight Learning Based on Single Feature Accuracy

The second method is motivated from the work of Holte since each feature is processed individually in k-NNFP algorithm [30]. We called this method SFA (Single Feature Accuracy) since feature weights are learned from the accuracy of the k-NNFP algorithm of each feature individually. Holte reports the results of experiments measuring the performance of very simple rules on the datasets commonly used in machine learning research. The specific kind of rules studied is called *1-rules*, which classify an object on the basis of a single feature. This study motivated us to examine the classification accuracy of the k-NNFP algorithm on the basis of a single feature. Therefore, those accuracies can be used as the weight of that feature since those accuracies reflect how much each feature can contribute to the final classification. However, a totally irrelevant feature will have about $1/No\ of\ Classes$ accuracy, called *random accuracy.* In order to avoid random correct classification, we subtract the random accuracy of a feature from the individual accuracies.

3.4.3 Experiments on Real-World Datasets

An empirical evaluation of two feature weighting methods, HFP and SFA, is presented here along with their comparisons with unweighted version of the k-NNFP algorithm by 5-fold cross-validation.. The weighted versions of the k-NNFP algorithm are evaluated on some real-world datasets selected from the collection of datasets provided in the UCI-Repository [47]. The characteristics of these datasets are shown in Appendix A.

The accuracy results of k-NNFP and its two weighted versions are given in Table 3.5. In this table, the first row of each k value presents the k-NNFP

Table 3.5. Accuracies (%) of the k-NNFP (N) and its weighted versions using homogeneneous feature projections (HFP) and single feature accuracy (SFA) feature weighting methods.

	Data Set:	bcancerw	cleveland	glass	hungarian	ionosphere	iris	liver	wine
N	k=1	94.00	67.62	57.00	**70.04**	88.04	**90.00**	50.44	79.70
HFP		**94.28**	67.62	**57.92**	68.70	88.32	89.98	50.42	**87.58**
SFA		**94.28**	**79.60**	57.00	61.52	**88.60**	89.98	**58.26**	87.00
N	k=3	94.88	72.94	61.18	75.84	**88.02**	91.34	55.68	90.96
HFP		**95.02**	72.92	62.14	**77.88**	**88.02**	94.02	56.52	94.36
SFA		**95.02**	**77.24**	**62.58**	77.18	**88.02**	**94.68**	**60.58**	**94.9**
N	k=5	**96.16**	78.88	60.72	76.16	87.46	91.30	58.26	93.24
HFP		96.02	80.18	59.78	**77.86**	87.18	93.32	57.96	94.38
SFA		**96.16**	**80.50**	**65.84**	74.78	**87.74**	**94.00**	**63.50**	**94.90**
N	k=7	**96.00**	79.52	62.58	74.80	87.74	92.00	61.46	95.48
HFP		94.96	79.20	63.06	**76.86**	**87.80**	93.34	61.44	**95.52**
SFA		95.86	**81.50**	**66.76**	74.78	86.90	**94.00**	**64.64**	95.50
N	k=9	96.14	78.52	63.04	75.80	87.44	92.00	62.04	96.62
HFP		**96.28**	79.18	63.06	**78.20**	**87.74**	94.02	62.32	96.62
SFA		**96.28**	**81.52**	**66.30**	72.74	**87.74**	**94.68**	**64.94**	**97.20**

algorithm results, the second row is the results of the HFP weight learning, and finally the third row presents the results of SFA feature weighting method.

These experiments showed that none of the weight learning algorithms improved the k-NNFP algorithm on the bcancerw and ionosphere datasets significantly. This should be because all the features on these datasets are equally relevant. On the cleveland, liver, iris and glass (except $k = 1$) datasets, the weights learned by the individual accuracies always performed significantly better than the others. The HFP weight learning method performed better than the other on the hungarian dataset, except $k = 1$. There were no significant difference between the two weight learning algorithms on the wine dataset.

Our findings emphasize that weighted versions do not improve the k-NNFP algorithm effectively in most of the real-world datasets. Langley & Sage concluded from their experiments with feature selection that a number of data sets in the UCI repository contain few or no irrelevant features [38].

3.4.4 Discussion

A version of the well-known k-NN algorithm, that stores the classification knowledge as the projections of the training instances on the features, called k-NNFP algorithm, had been shown to be successful (Section 3.3). We have presented two methods for determining the relative weights of features for use in the k-NNFP algorithm. The HFP method assigns a higher weight to features on which the projections of instances of the same class are located close to each other, resulting in a homogeneous distribution. The SFA method, on the other hand, assigns a weight as the classification accuracy that would have been obtained if only that feature were used in the classification.

Our experiments revealed that these weighting methods assign low weights to completely irrelevant features, and high weights to relevant ones. Further, among these two weight learning algorithms, the one that is based on the individual accuracies learned weights that helped k-NNFP achieve higher accuracies. The reason for this success is due to the feedback received from the classification algorithm. We conclude that this weight learning method could be successful for other classification algorithms that use feature weights. As a further work we plan to investigate these weight learning methods on artificial datasets.

3.5 Summary

In this chapter, feature projections for knowledge representation have been presented. The most important advantage of this representation is that sorted feature values reduces the time for computation of similarity to all training instances for NN like techniques. In addition, since each feature is considered separately, handling of missing feature values by simply ignoring them is natural. Furthermore, this representation is plausible. The major drawback of feature projections knowledge scheme is that descriptions involving conjunction between two or more features cannot be represented. However, prior research on this representation, by the CFP and COFI algorithms, has shown that they are successful in classification of real-world tasks.

The next chapter will introduce several batch learning methods for classification where knowledge is represented in the form of disjoint feature intervals. This is one of the primary contributions of this thesis.

Chapter 4

Batch Learning of Disjoint Feature Intervals

This chapter is devoted to batch Feature Intervals Learning (FIL) algorithms. We have seen in the previous chapter that feature projections for knowledge representation have become successful with the advantage of lower time requirement of classification task and natural handling of missing feature values; despite its limited representation power. The CFP and COFI algorithms presented in Chapter 3 are incremental supervised inductive learning algorithms (Section 3.1 and Section 3.2). Hence, the classification knowledge learned by these algorithms is sensitive to presentation order of training instances. In Section 3.3, we have presented a new classification algorithm k-NNFP that classifies unseen instances on the basis of feature projections in a batch mode. That is a variation of classical k-NN algorithm. This chapter is, therefore, devoted to developing batch learning of feature intervals and several modifications that can improve their performance. Basic characteristics of the FIL algorithms are that they are batch supervised inductive learning algorithms, based on feature intervals for knowledge representation. Although classification is much faster in the k-NNFP algorithm, its storage requirements are quite high. The algorithms discussed in this chapter attempt to find more compact representations of the training data by constructing feature intervals that represent a collection of feature values that belong to the same class. More compact representations lead to faster classifications and may increase the ability of the user to understand decisions made by the classifier.

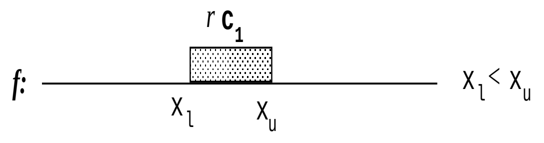


Figure 4.1. An example for an interval.

The FIL algorithms described here are the FI1, FI2, FI3, and FI4 algorithms with slight differences. First, we will explain the training and classification process in the basic FIL algorithm, FI1, through examples and then present the details of the algorithms. This is followed by discussion of modified algorithms (FI2, FI3, FI4). Finally, general characteristics of the FIL algorithms are discussed classifying them according to some important dimensions in machine learning.

4.1 Basic Definitions

First, we will give some necessary definitions before explaining the FIL algorithms.

Definition. An **interval** is a range of values of a feature dimension, such that all the training instances whose values for that feature fall into that range have the same class label.

Figure 4.1 shows an example for a feature interval. This interval represented as $< [x_l,\ x_u],\ r,\ C_1 >$ on feature f indicates that, in the training set there are r instances whose f values lie in the range $[x_l,\ x_u]$ and their class label is C_1.

Definition. A **point interval** is an interval whose lower and upper bounds are the same.

An example for a point interval is given in Figure 4.2. Here, there are training instances whose f values are x and their class label is C_1. Other neighboring feature values belong to different classes from C_1. There may be more than one point interval at a same feature value.

Definition. A **range interval** is an interval whose lower and upper bounds

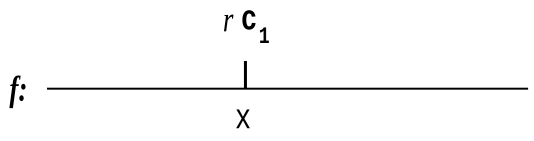


Figure 4.2. An example for a point interval.

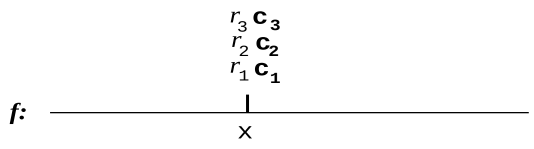


Figure 4.3. An example for a multi-class point.

are not equal ($x_l \neq x_u$).

Figure 4.1 also illustrates a range interval. Range intervals contain several feature values belonging to a same class label.

Definition. A **single-class point** is a value on a feature dimension that belongs to the single class label.

For example, x in Figure 4.2 is a single-class point on feature f. Neighboring same single-class points are extended into intervals. But, point intervals may be constructed at single-class points if the neighboring feature values belong to different class labels.

Definition. A **multi-class point** is a value on a feature that belongs to more than one class labels.

Figure 4.3 illustrates an example for a multi-class point. That is, there are r_1 training instances of class C_1, r_2 training instances of class C_2, and r_3 training instances of class C_3 whose f values are x_1. These can be represented in three point intervals:

$< [x_1,\ x_1],\ r_1,\ C_1 >$,

$< [x_1,\ x_1],\ r_2,\ C_2 >$,

$< [x_1,\ x_1],\ r_3,\ C_3 >$.

4.2 Description of the FIL Algorithms

In this section, the training and classification processes of the FIL algorithms will be explained through examples. Then, the details of these algorithms will be presented. Also voting-based classification process will be illustrated through examples.

4.2.1 The FI1 Algorithm

In the training phase of the FI1 algorithm (basic FIL), learning task is performed by constructing disjoint feature intervals in a batch mode. All training instances are taken and processed at once. Feature intervals on each feature dimension are constructed through generalization. Concept descriptions learned are represented in the form of sets of disjoint feature intervals. For the classification task, each feature determines its own prediction (preclassification) using only its local knowledge by searching the interval covering test example's value for that feature. The classification of an instance is based on a majority voting taken among the preclassifications made by each feature. The FIL algorithms can handle both continuous (linear) and nominal valued features.

4.2.1.1 Training in the FI1 Algorithm

The input to the FI1 algorithm is a training set that contains examples represented as vectors of feature values plus the corresponding class label. An example is represented as **x**$=< x_1, x_2, .., x_n, C_x >$ where $x_1, x_2, .., x_n$ are the corresponding feature values of features $f_1, f_2, .., f_n$, and C_x is the associated class label of the example x where $1 \leq C \leq k$, here k is the total number of the classes. Therefore, the dimension of the example vector i is $n+1$ where n is the number of features.

Since the FI1 algorithm learns in a batch mode, it takes all training examples and processes them at once. In the FI1 algorithm, the basic unit of the knowledge representation is an interval with four parameters:

$$< [\textit{lower bound},\ \textit{upper bound}],\ \textit{representativeness count},\ \textit{class label} >$$

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. Finally, the class label is the associated class of the interval. In other words, learned classification knowledge is represented as the set of feature intervals by generalizing neighboring same single-class points into intervals. Feature intervals are disjoint. However, multi-class points remain as point intervals as in Figure 4.3. In that case, a set of point intervals (upper and lower bounds are equal) are constructed for multi-class points. Otherwise, disjoint feature intervals are *single-class* intervals.

Let us give an example to illustrate the training process of the FI1 algorithm. Here, training instances are represented also as vectors of feature values and the associated class as shown in Figure 4.4. Training set has 18 examples described with three linear features. There are there different classes in this sample training set (C_1, C_2 and C_3). First, feature projections on each feature dimension are displayed in Figure 4.4 for this sample training set. This corresponds to the process of presenting all training instances initially and storing them in memory as sorted (if they are linear features) on each feature dimension. That is the only information kept in the memory to construct feature intervals.

Then, from this knowledge, initial point intervals are constructed with equal lower and upper values. This is the same as feature projections shown in Figure 4.4, except additional information such as lower, upper bound values, representativeness count and associated class label. Since all features are linear, their intervals are generalized. Generalization process combines neighboring point intervals into a single interval if they are of the same class. The resulting concept descriptions in the form of feature intervals is given in Figure 4.5. For example, the feature projections on the first feature dimension forms the following set of feature intervals on f_1 dimension:

TRAINING SET

$i_1 = < 1,10, 7, C_1>$	$i_{10} = <10,17,16, C_2>$
$i_2 = < 3,12, 7, C_1>$	$i_{11} = <10,17,18, C_2>$
$i_3 = < 4,10,10, C_1>$	$i_{12} = < 4,17, 4, C_3>$
$i_4 = < 4,12,10, C_1>$	$i_{13} = < 4,17, 4, C_3>$
$i_5 = < 4,15,10, C_1>$	$i_{14} = < 4,17, 1, C_3>$
$i_6 = < 4, 7, 7, C_2>$	$i_{15} = < 4,17, 4, C_3>$
$i_7 = < 6,17, 7, C_2>$	$i_{16} = < 6,17, 4, C_3>$
$i_8 = < 6, 9,13, C_2>$	$i_{17} = < 6,19, 1, C_3>$
$i_9 = < 8,17,15, C_2>$	$i_{18} = < 9,19, 4, C_3>$

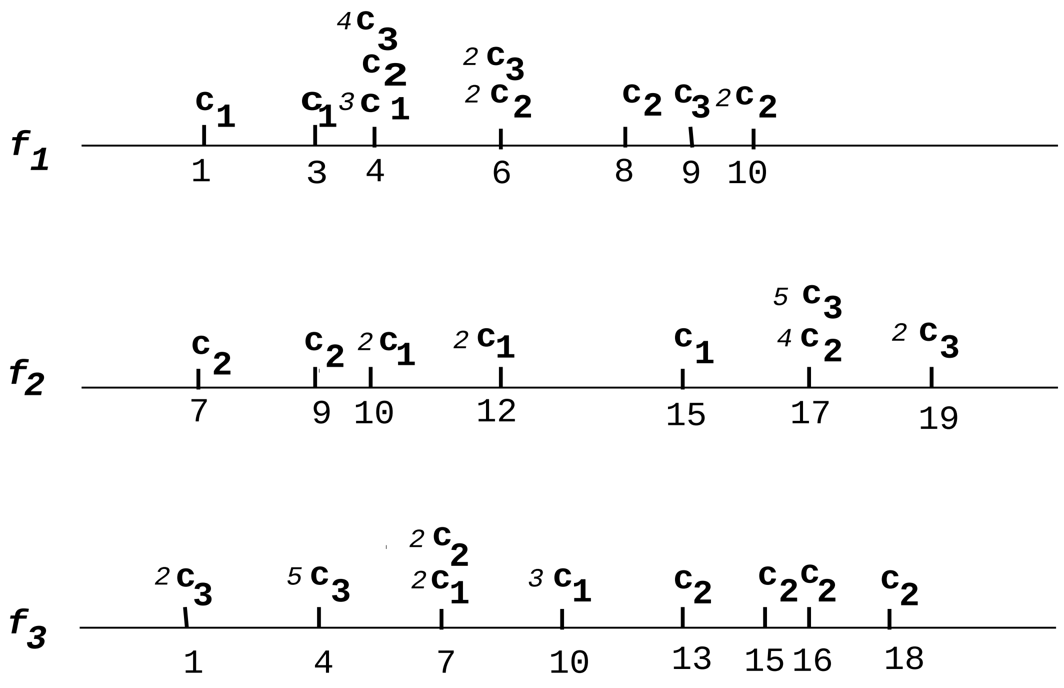


Figure 4.4. A Sample Training Set and Feature Projections on Each Feature Dimension

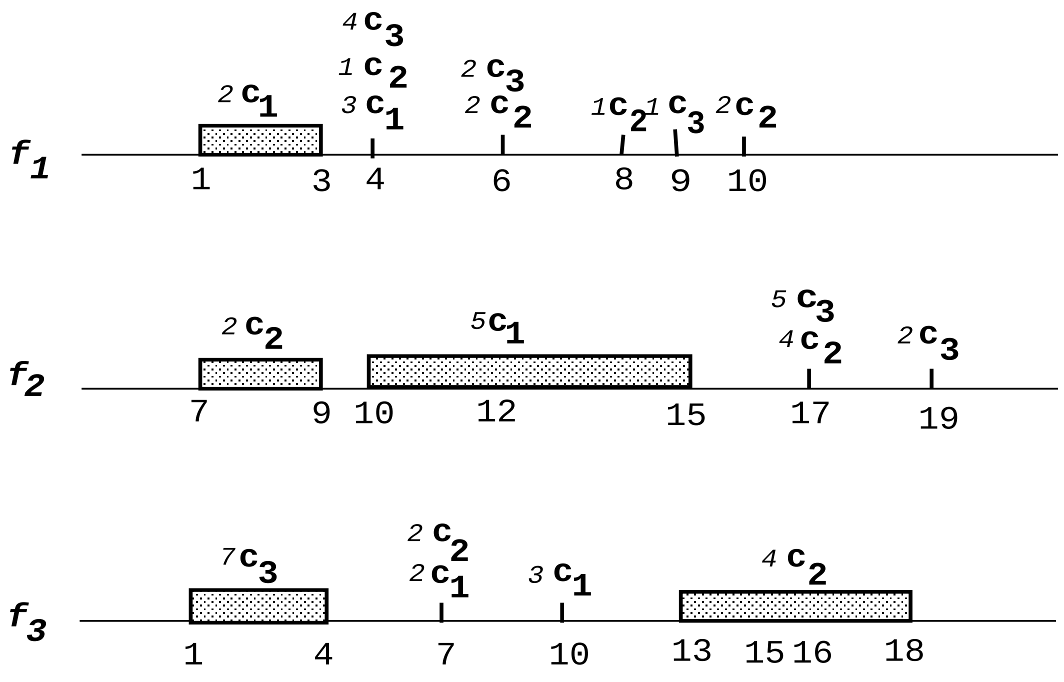


Figure 4.5. Construction of feature intervals in the FI1 algorithm.

$$\{ \quad <[1,\ 3],\ 2,\ C_1> \quad <[4,\ 4],\ 3,\ C_1> \quad <[4,\ 4],\ 1,\ C_2>$$
$$<[4,\ 4],\ 4,\ C_3> \quad <[6,\ 6],\ 2,\ C_2> \quad <[6,\ 6],\ 3,\ C_3>$$
$$<[8,\ 8],\ 1,\ C_2> \quad <[9,\ 9],\ 1,\ C_3> \quad <[10,\ 10],\ 2,\ C_2> \quad \}$$

The only range interval constructed on f_1 is the first interval since only it contains neighboring single-class points that belong to the same class label whereas the other ones are either multi-class points or neighboring single-class points that do not belong to the same class. Multi-class points remain as point intervals allowing more than one interval at the same feature value as shown in Figure 4.5. Nominal features have only point (possibly multi-class) intervals. This is because nominal values cannot be generalized. Figure 4.6 summarizes the training process of the FI1 algorithm.

4.2.1.2 Classification in the FI1 Algorithm

The output of the training process of the FI1 algorithm is the concept descriptions learned in the form of feature intervals. In the FI1 algorithm, the

```
train($Training\ Set$)
begin
        sortTrainingData($Training\ Set$) /* on each feature dimension */
        construct-intervals($Training\ Set$)
end.


construct-intervals($Training\ Set$)
begin
        for each feature $f$
            for each training instance $i$
                initialize-point-intervals($f, i$)
            if $f$ is linear then
                generalize-point-intervals($f$)
            /* if $f$ is a nominal feature, no generalization is done */
end.


generalize-point-intervals($f$)
begin
        for each consecutive interval pair
            if their classes are same and they are single-class intervals then
              join them into a range interval
              /* update lower, upper and representativeness values */
end.
```

Figure 4.6. Training process in the FI1 algorithm.

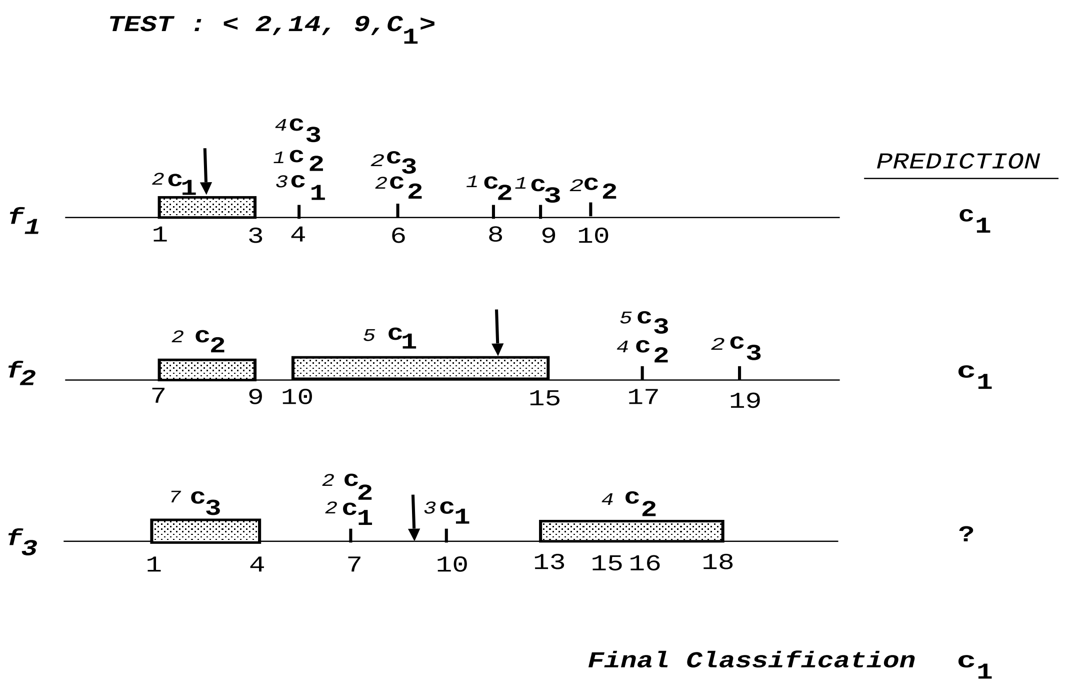


Figure 4.7. An example for classification in the FI1 algorithm.

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 have different levels of relevances. Assuming equal relevance is a special case of weighted-voting, i.e., 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 (=1). 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. If all feature dimensions give no predictions, then no classification can be made and the resulting decision for the class will be `UNDETERMINED`. Nevertheless, this case is quite unlikely to occur in real-world datasets.

In order to determine the final classification, the local vote of each feature are summed up. The class which receives the the maximum vote is the classification for the test instance. This can be summarized as follows.

$$classification(test) = c \text{ } such \text{ } that \text{ } v_c > v_i \text{ for each } i \neq c.$$

Let us illustrate the classification process of the FI1 algorithm by classifying the test instance $< 2, 14, 9, C_1 >$ according to the concept descriptions learned by the FI1 algorithm in the training phase as shown in Figure 4.5. Each feature value of this test instance is indicated in Figure 4.7 by arrows on each feature dimension. Each feature makes a preclassification for this instance. In the first dimension, the first feature value, 2, falls into the first interval with class C_1, $< [1, 3], 2, C_1 >$. Therefore, it predicts that the class of the test instance should belong to the class C_1. The result of preclassification of the second feature is again class C_1 since the second feature value, 14, falls into the interval $< 10, 15, 6, C_1 >$. The third feature makes no prediction since the third feature value, 9, is not contained by any interval. The vote vector for this test instance becomes $< 2, 0, 0 >$. Here, 2 votes for class C_1 and no votes for classes C_2, C_3. The class which receives the maximum vote, C_1 in this case, is determined as the final class prediction. Since the actual class value of the test instance is also C_1, the final prediction is a correct classification. It should be noted that, for this example, equal feature weights are assumed. The classification process of the FI1 algorithm is outlined in Figure 4.8. Some experiments will be performed to investigate the effect of weighting features in voting mechanism in Chapter 5.

```
classify($test$)
begin
        for each class $c$
             vote[$c$] = 0

        for each feature $f$
             $interval$ = search-interval($f$, $test_f$)
             /* each feature contributes proportional to its weight */
             if class of $interval \neq$ UNDETERMINED then
                vote[class of $interval$] = vote[class of $interval$] + weight[$f$];

        $prediction$ = first class
        for each class $c$
             if vote[$c$] > vote[$prediction$] then
                $prediction = c$

        if vote[$prediction$] = 0 then
             $prediction$ = UNDETERMINED /* all features make no prediction */

        return $prediction$
end.


search-interval($f$, $value$)
begin
        if $value$ on $f$ is a single-class point then
           return $interval$ on that point
        else if $value$ on $f$ is multi-class point then
           return $interval$ with the highest representativeness count
        else if $value$ on $f$ is contained in a range interval then
           return $interval$ on that $value$
else /* no interval exists for that value */
        return UNDETERMINED
end.
```

Figure 4.8. Classification process in the FI1 algorithm.

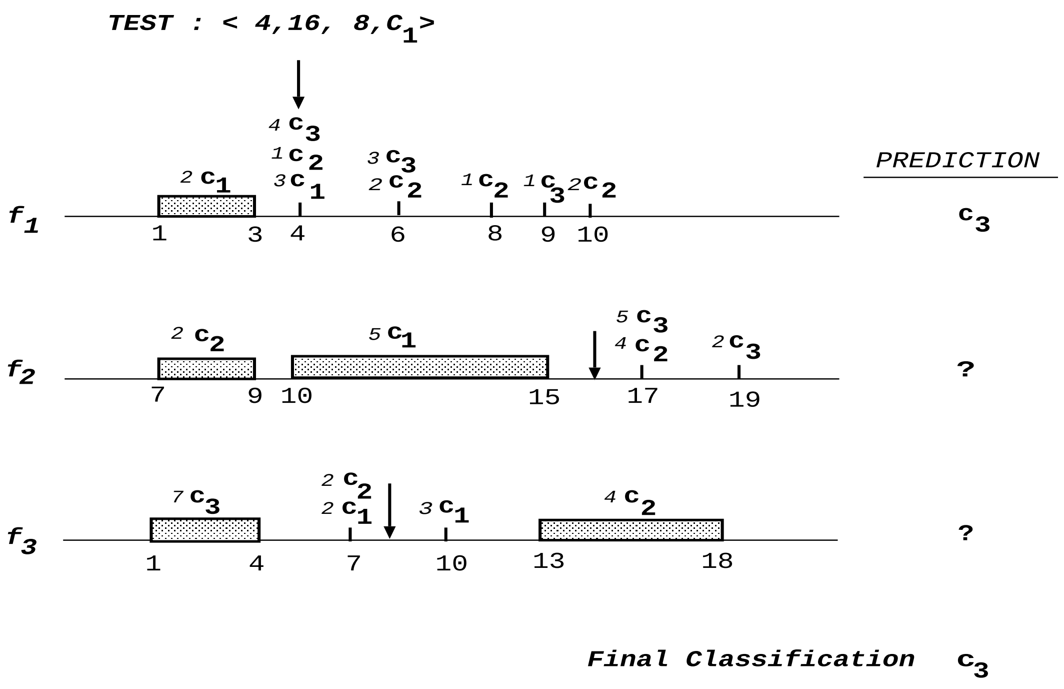


Figure 4.9. An Example for an incorrect classification in the FI1 algorithm that leads to the FI2 Algorithm.

4.2.2 The FI2 Algorithm

Figure 4.9 illustrates classification of another test example $< 4,\ 16,\ 8,\ C_1 >$. In this case, features f_2 and f_3 make no predictions since projections on these features are not contained by any interval. The first feature value falls into a multi-class point of class C_3. The FI1 algorithm determines the local prediction of the first feature according to the class that has the maximum representativeness count. Hence, C_3 will be predicted without considering the distribution of classes. This leads to a slight modification in the FI1 algorithm, called FI2. Basic unit of knowledge representation in the FI2 algorithm is also interval with a slight difference: it uses *relative representativeness count* which is the ratio of the representativeness count to the total number of training instances of the corresponding class rather than absolute representativeness count.

In this sample training set, there are three training instances of C_1 class at this feature value, 4, whereas there are four instances of C_3 class. So, the relative representativeness counts of intervals with class C_1 and C_2 are 3/5 and

4/7, respectively. The relative representativeness count of C_1 is greater than that of C_2. If preclassification on a feature dimension is made according to the relative representativeness count of multi-class points, this may be more fair, without always giving a chance to the classes that appear more frequently in the training set. Therefore, the concept of relative representativeness count introduces a modification to classification process of the FI1 algorithm. After training, only representativeness counts are divided by total number of corresponding classes. This is a kind of normalization of class distributions and required for datasets with unequally distributed classes.

The training process of the FI2 algorithm is identical to FI1 except that after construction of intervals, each feature maintains relative representativeness count rather than representativeness count, as outlined in Figure 4.10. The difference in the classification process appears in the preclassification of test values at multi-class points. The class of the interval which has the maximum relative representativeness count is chosen as the prediction. This difference in the classification process is summarized in Figure 4.11.

4.2.3 The FI3 Algorithm

Since learning is achieved in the batch manner, all training instances are known before the construction of feature intervals in both FI1 and FI2 algorithms. Once they are constructed, the intervals having less representativeness count than the one with maximum in the FI1 and relative representativeness in the FI2 algorithms are not used in the classification process. This raises the following question: *Why do we store them?* This motivated us to investigate a method to store a single point interval in multi-class points.

For this purpose, we tried to eliminate less likely contributing intervals to classification. The interval having the maximum representativeness count is chosen as the class of the interval on that multi-class point. The elimination of the intervals with lower representativeness counts leads to the *pruning* of presumably noisy intervals. However, one should be careful in this pruning. For example, consider a multi-class point at value v with intervals

```
generalize-point-intervals($f$)
begin
    for each consecutive interval pair
        /* update lower, upper and representativeness values */
        if their classes are the same and they are point intervals then
            join them into a range interval

    /* normalization of class distributions among intervals */
    for each interval
        relative represent value = $\frac{represent\ value\ of\ interval}{total\ no\ of\ class\ of\ interval}$
end.
```

Figure 4.10. Generalization of intervals in the FI2 algorithm.

```
search-interval($f$, $value$)
begin
    if $value$ on $f$ is a single-class point then
        return $interval$ on that point
    else if $value$ on $f$ is multi-class point then
        return $interval$ with the highest relative representativeness count
    else if $value$ on $f$ is contained in a range interval then
        return $interval$ on that $value$
    else /* no interval exists for that value */
        return UNDETERMINED
end.
```

Figure 4.11. Preclassification process in the FI2 algorithm.

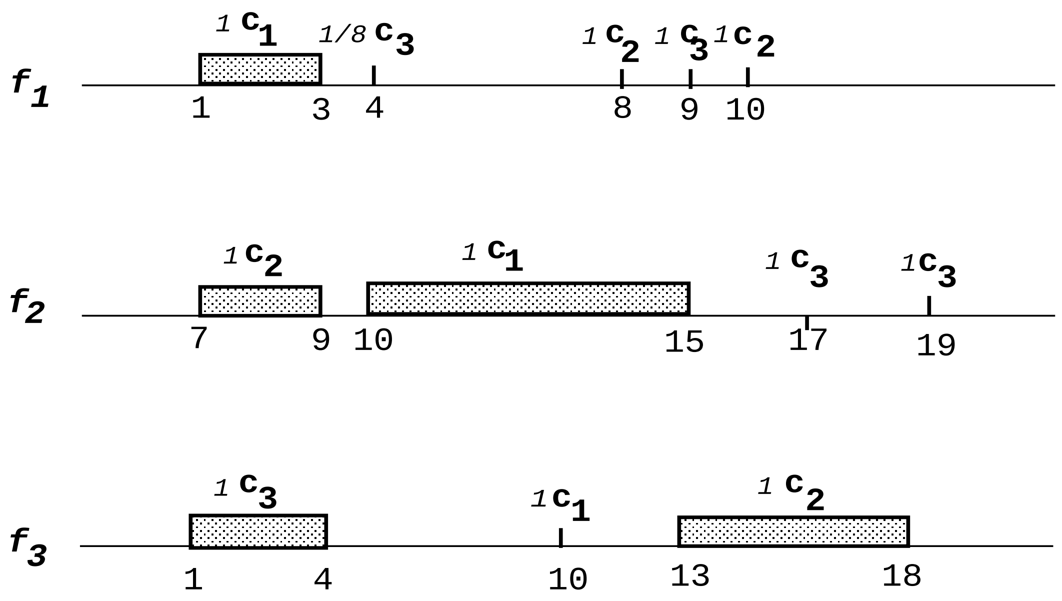


Figure 4.12. Construction of feature intervals in the FI3 algorithm.

$< [v, v], 50, C_1 >$,

$< [v, v], 49, C_2 >$,

$< [v, v], 2, C_3 >$.

If we simply remove the last two intervals, we loose the information that at this value v, C_1 and C_2 classes are equally possible. In order to establish a balance between intervals with high representativeness counts, we designed a new method for placing weights to intervals rather than features. To determine the weight of a new point interval, two point intervals having maximum representativeness counts are found. Then, the weight of the interval is set to be the difference between two maximum representativeness counts divided by the total number of representativeness counts of multi-class points at that feature value. We called this algorithm as FI3. An interval in the FI3 algorithm is represented as follows:

$<$ [*lower bound*, *upper bound*], *weight of interval*, *class label* $>$

Here, weight of the interval represents the vote of the interval when it contributes to classification. All other information is the same as in the FI1 algorithm.

The concept descriptions of the sample training set given in Figure 4.4 learned by the FI3 algorithm is presented in Figure 4.12. Note that the storage requirement of the FI3 algorithm is smaller than the FI1 and FI2 algorithms if there are many multi-class points.

In the preclassification of feature f, its vote is for the class of interval as the weight of interval containing the feature value of the test instance on the feature dimension f. The classification example in Figure 4.13 illustrates the behavior of the FI3 algorithm. In this example, the class C_3 will take 1 vote from feature f_3 since f_3th value falls into a range interval and range interval votes are set to be 1. The first feature votes 1/8 vote for C_3 since first feature value falls into a multi-class interval. The class C_2 will take 1 vote from the range interval on feature f_2. The vote vector becomes $< 0, 1, 9/8 >$. Final classification is the class C_3. Since the actual class of the test example is C_3, the test instance will be correctly classified by the constructed intervals as shown in Figure 4.13.

The differences in the training and classification algorithms are listed in Figure 4.14 and Figure 4.15. In the training, the weights of feature intervals are learned in addition to their constructions. In the classification, these weights are used for feature votes.

4.2.4 The FI4 Algorithm

In the FI3 algorithm, initial single-class point intervals will have the maximum weight (=1). However, these can be noisy intervals as well. To decrease the effect of such intervals, normalization of these interval weights are required. This is done by dividing these weights to the total number of their classes in the training dataset. Figure 4.17 illustrates this by an example. The test instance $< 8,\ 18,\ 3,\ C_3 >$ will be tested according to the knowledge learned by the FI3 algorithm. The value for f_1 falls into the interval $<\ [8,\ 8],\ 1/6\ ,\ C_2\ >$. The weight of this interval becomes 1/6 since there is only one training instance whose f_1th value is 8, but totally there are 6 training instances of class C_2

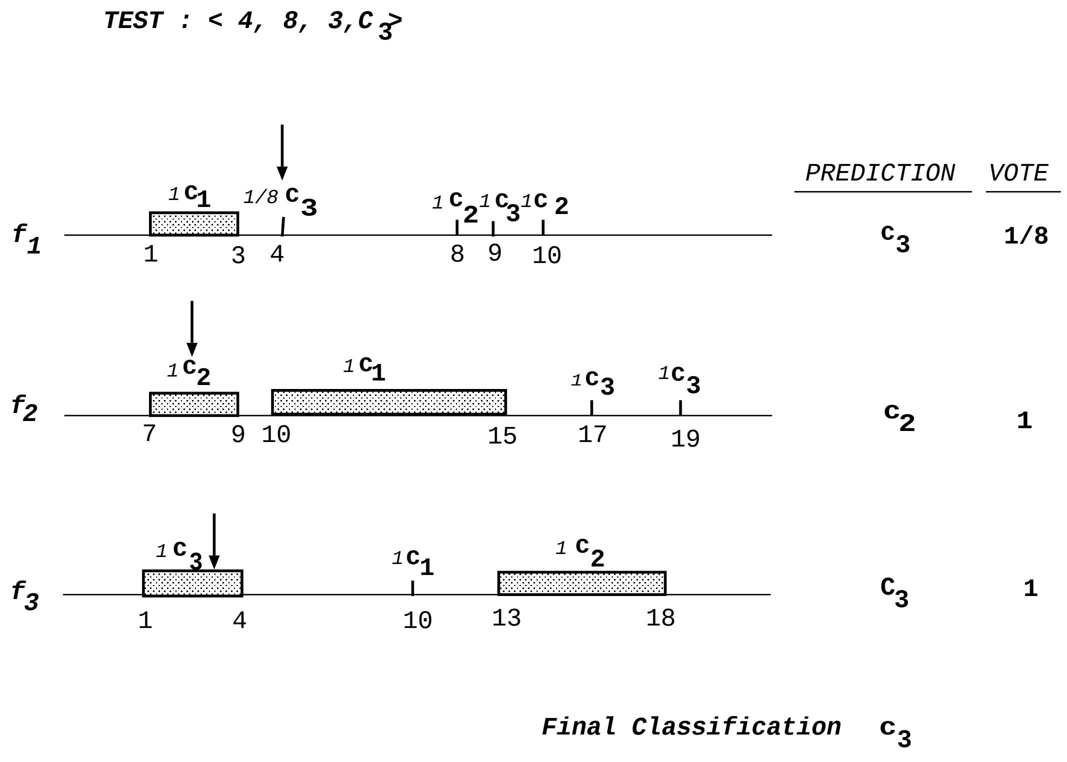


Figure 4.13. An example for classification in the FI3 algorithm.

in the whole dataset. So, the first feature predicts C_2 with weight 1/6, the second feature makes no prediction and the third one predicts C_3 with weight 1, because all feature values that belong to the class C_3 on feature f_3 fall into the same interval.

The training process of the FI4 algorithm is identical to the FI3 algorithm except normalization of feature interval weights according to class distributions in the training set. The normalization process is outlined in Figure 4.16. The classification task is performed as in the FI3 algorithm using more reliable feature interval weights.

4.3 Characteristics of FIL Algorithms

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

```
construct-intervals($Training\ Set$)
begin
    for each feature $f$
        for each training instance $i$
            initialize-point-intervals($f, i$)
        if $f$ is linear then
            generalize-point-intervals($f$)
        /* if $f$ is a nominal feature, no generalization is done */
        compute-interval-weights($f$)
end.
```

```
compute-interval-weights($f$)
begin
    for each interval
        if range interval or single-class point then
          $weight\ of\ interval\ =\ 1$
        else
          find the interval having maximum repr. count
          $weightof\ interval\ = \frac{difference\ between\ two\ max.\ representativeness\ counts}{total\ repr.\ count}$
end.
```

Figure 4.14. Training process in the FI3 algorithm.

```
classify($test$)
begin
    for each class $c$
        vote[$c$] = 0

    for each feature $f$
        $interval$ = search-interval(f, $test_f$)
        /* each interval contributes proportional to its weight */
        if class of interval $\neq$ UNDETERMINED then
           vote[class of $interval$] =
                vote[class of $interval$] + weight of $interval$

        $prediction$ = first class
        for each class $c$
           if vote[$c$] > vote[$prediction$] then
                $prediction$ = $c$

        if vote[$prediction$] = 0 then
           $prediction$ = UNDETERMINED /* no final prediction */

        return $prediction$
    end.
```

Figure 4.15. Classification process in the FI3 algorithm.

```
compute-interval-weights($f$)
begin
    for each $interval$
        if range interval or single-class point then
           weight of $interval$ = 1
        else
           find the $interval$ having maximum repr. count
           weight of $interval$ = $\frac{difference\ between\ two\ max.\ representativeness\ counts}{total\ repr.\ count}$

    /* interval weights are normalized according to class distributions */
    for each $interval$
        divide weight of $interval$ by
           total no of class of $interval$ in training set
end.
```

Figure 4.16. Normalization of interval weights in the FI4 algorithm.

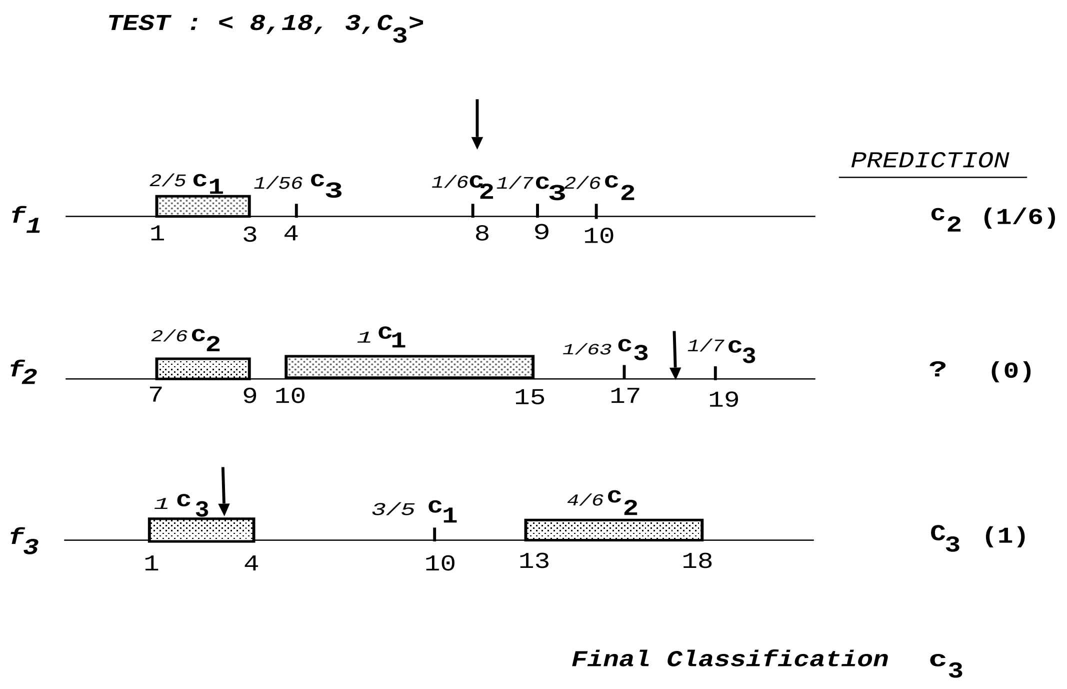


Figure 4.17. An example of classification in the FI4 algorithm.

4.3.1 Knowledge Representation

Knowledge representation is one of the most important dimensions in classifying machine learning techniques. Many learning systems acquire knowledge in the form of *rules*. Another way to represent what is learned is with decision trees as in the ID3 and C4.5 algorithms [55]. On the other hand, knowledge representation in exemplar-based learning models is sets of representative instances [1, 2, 5] or hyperrectangles which represent generalizations [58, 59].

In Chapter 3, we presented a new knowledge representation in the form of feature projections. Generalization and specialization are made on the basis of feature projections. This introduces 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 [65, 67]. The CFP and COFI algorithms use this representation to learn concept descriptions in the form of disjoint feature intervals and overlapping feature intervals in an incremental manner [27, 28, 65, 67]. The k-NNFP algorithm also uses this representation in order to classify test instances on the basis of feature projections [7].

The FIL algorithms also acquire concept descriptions by using feature projections for knowledge representation. Learned concept descriptions are stored in memory in the form of *disjoint feature intervals*. These intervals are disjoint (single-class) covering only single-class neighboring point. The multi-class points are represented a set of point intervals. Each interval contains upper and lower bounds, representativeness count that is the number of examples that interval represents, and the associated class label of the interval. The number of intervals on a feature dimension depends on the training set, and they are unique for the same training set being independent of presentation order of training instances. At the worst case, if all examples have different feature values, the feature may be either nominal or linear, then the number of intervals is equal to $m \cdot n$ where m is the number of instances, and n is the number of features.

4.3.2 Inductive Learning

Inductive learning can be described as learning from facts that are provided by a teacher or an environment by drawing inductive inference. Acquiring knowledge involves operations of generalizing, specializing, transforming, correcting and refining knowledge representations [43]. Learning a concept usually means to learn its description, i.e., a relation between the name of the concept and a given set of features by making some inferences.

The FIL algorithms perform the learning task from a set of training examples and make generalizations on the feature projections to construct the concept descriptions in the form of disjoint feature intervals.

4.3.3 Supervised Learning

Supervised learning has been the most widely studied learning paradigm in inductive learning systems, pattern classification and system identification [13]. In this learning paradigm, the learner is asked to associate pairs of items. For example, in pattern classification or concept acquisition, the first item is an instance of some pattern or concept and the second item is the name of the concept. In system identification, the learner must reproduce the input-output behavior of some unknown system. Here, the first item of each pair is an input and the second item is the corresponding output. In machine learning, from a set of training examples, each labeled with its correct class name, a machine learns by forming or selecting a generalization of the training examples.*Unsupervised learning* techniques try to estimate the class distributions successively from unlabeled training instances.

The FIL algorithms learn from examples provided, that is, the supervised learning paradigm is followed. Here, the first item is the feature values of an instance and the second item is the class of that instance.

4.3.4 Batch Learning

Quinlan has pointed out two alternative learning strategies as *incremental* and *batch (non-incremental)* [52]. Incremental learning aims to improve an internal model with each example it processes. Researchers who explore the incremental approach are typically concerned with developing plausible models of human learning, with agents that must interact with a dynamic environment, or with the efficiency of the learning mechanisms. On the other hand, batch learning attempts to construct concept descriptions after seeing all training instances to maximize the performance of the learning system. In contrast to incremental learning, researchers who employ batch learning strategy are concerned with automating the process of knowledge acquisition for higher performance.

A batch learning strategy usually assumes random access to the examples in the training set. A learning system which follows this strategy searches for patterns and regularities in the training set in order to induce concept descriptions. They may examine and re-examine the training set many times before settling on a successful model. The most important advantage of this approach is that it is not sensitive to the order of the training examples.

Despite the differences in motivation, researchers in both paradigms have much to learn from each other. Incremental and batch systems often use the same basic learning operators and produce similar results. In many cases, one can create incremental variations of non-incremental algorithms. Presumably, many incremental learning methods also have non-incremental counterparts.

Batch learning strategy is employed in the FIL algorithms. Before training, all instances are presented as input to the algorithms. In the FIL algorithms, concept descriptions are represented in the form of disjoint feature intervals. The construction of intervals is unique for that training set, that is, they are independent of presentation order of training instances.

4.3.5 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 [18]. In EBG, 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.

In contrast, exemplar-based learning does not construct explanations. Instead, it incorporates new examples into its experience by modifying its existing concept representation in the memory. Because it does not convert examples into another representation form, it does not need a domain theory to explain what conversions are legal. A consequence 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 FIL algorithms are also exemplar-based learning algorithms, based on *generalized feature values*. Although they do not use parameters used as in the CFP and COFI algorithms, weights of features in the FI1 and FI2 algorithms are given externally. In the FI3 and FI4 algorithms, there is no need for feature weights. Therefore, the FI3 and FI4 algorithms do not require any parameter to be provided externally.

4.3.6 Multi-concept Learning

Many early concept learning algorithms have been developed for exactly one concept. Later, many learning algorithms have been developed that induce multi-concept descriptions from examples. The FIL algorithms have been designed for learning multi-concept descriptions as well.

4.3.7 Properties of Feature Values

The features in a dataset may have nominal (categorical), or continuous (numerical) values. The term *continuous* is used in literature to refer to features taking on numerical values (integer or real), in general a feature with a linearly ordered set of attribute values. The FIL algorithms can handle both linear and nominal features. Linear features may take on values from $-\infty$ to ∞ and they are continuous. Nominal features take on discrete feature values, for example, color attribute of 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 the generalization process. Generalization is applied only to linear features.

```
; Information about the IRIS dataset
Features l l l l
Classes  0 1 2
```

Figure 4.18. An example for the information provided to the FIL algorithms.

4.3.8 Handling Missing (Unknown) Feature Values

One of the most important advantages of the FIL algorithms is the handling of missing feature values. There is no need to fill in missing values in the FIL algorithms. This affects neither the construction of concept descriptions nor the voting mechanism. 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 User Interface

We have designed and implemented user interfaces for the FIL algorithms. These implementations have been done by using Motif user-interface toolkit. The FIL algorithms have been implemented in C language in Unix environment. The user can select a dataset from the 'Open' menu item. Then, with an initial training ratio training and testing sets are formed. User can enter the training ratio from the menu item 'Train Ratio' as well. Figure 4.18 presents an example for the information given to the FIL algorithms about the dataset, *iris* in this example with number and types of features and number and names of classes.

The feature intervals constructed during training phase of the algorithms are displayed on each feature dimension assigning a different color to each class label on the screen. Usage of colors provides users to better understand predictions made by individual features. User can see the classification of a single test instance by performing classification task step by step with "NEXT" button. Also, all test instances can be classified at once with "ALL" button. It is also possible to see the previous examples and their classifications with "PREVIOUS" button. Classification accuracy and no of correct classifications after

f:1	s:1	l:4.3	u:4.8	c:0	r:13
f:1	s:2	l:4.9	u:4.9	c:0	r:2
f:1	s:3	l:4.9	u:4.9	c:1	r:1
f:1	s:4	l:4.9	u:4.9	c:2	r:1
f:1	s:5	l:5	u:5	c:0	r:8
f:1	s:6	l:5	u:5	c:1	r:2
f:1	s:7	l:5.1	u:5.1	c:0	r:7
f:1	s:8	l:5.2	u:5.2	c:0	r:3
f:1	s:9	l:5.2	u:5.2	c:1	r:1
f:1	s:10	l:5.4	u:5.4	c:0	r:5
f:1	s:11	l:5.4	u:5.4	c:1	r:1
f:1	s:12	l:5.5	u:5.5	c:1	r:3
f:1	s:13	l:5.6	u:5.6	c:1	r:3
f:1	s:14	l:5.6	u:5.6	c:2	r:1
f:1	s:15	l:5.7	u:5.7	c:0	r:1
f:1	s:16	l:5.7	u:5.7	c:1	r:3
f:1	s:17	l:5.8	u:5.8	c:0	r:1
f:1	s:18	l:5.8	u:5.8	c:1	r:3
f:1	s:19	l:5.8	u:5.8	c:2	r:3
f:1	s:20	l:5.9	u:5.9	c:1	r:1
f:1	s:21	l:6	u:6	c:1	r:3
f:1	s:22	l:6	u:6	c:2	r:1
f:1	s:23	l:6.1	u:6.1	c:1	r:3
f:1	s:24	l:6.1	u:6.1	c:2	r:1
f:1	s:25	l:6.2	u:6.2	c:1	r:2
f:1	s:26	l:6.2	u:6.2	c:2	r:2
f:1	s:27	l:6.3	u:6.3	c:1	r:3
f:1	s:28	l:6.3	u:6.3	c:2	r:6
f:1	s:29	l:6.4	u:6.4	c:1	r:1
f:1	s:30	l:6.4	u:6.4	c:2	r:4
f:1	s:31	l:6.5	u:6.5	c:1	r:1
f:1	s:32	l:6.5	u:6.5	c:2	r:4
f:1	s:33	l:6.6	u:6.6	c:1	r:2
f:1	s:34	l:6.7	u:6.7	c:1	r:3
f:1	s:35	l:6.7	u:6.7	c:2	r:3
f:1	s:36	l:6.8	u:6.8	c:1	r:1
f:1	s:37	l:6.8	u:6.8	c:2	r:2
f:1	s:38	l:6.9	u:6.9	c:1	r:1
f:1	s:39	l:6.9	u:6.9	c:2	r:3
f:1	s:40	l:7.1	u:7.9	c:2	r:11

Figure 4.19. Intervals of iris domain on the first feature.

f:2	s:1	l:2	u:2	c:1	r:1
f:2	s:2	l:2.2	u:2.2	c:1	r:2
f:2	s:3	l:2.2	u:2.2	c:2	r:1
f:2	s:4	l:2.3	u:2.3	c:0	r:1
f:2	s:5	l:2.3	u:2.3	c:1	r:3
f:2	s:6	l:2.4	u:2.4	c:1	r:1
f:2	s:7	l:2.5	u:2.5	c:1	r:2
f:2	s:8	l:2.5	u:2.5	c:2	r:3
f:2	s:9	l:2.6	u:2.6	c:1	r:3
f:2	s:10	l:2.6	u:2.6	c:2	r:2
f:2	s:11	l:2.7	u:2.7	c:1	r:5
f:2	s:12	l:2.7	u:2.7	c:2	r:3
f:2	s:13	l:2.8	u:2.8	c:1	r:4
f:2	s:14	l:2.8	u:2.8	c:2	r:8
f:2	s:15	l:2.9	u:2.9	c:0	r:1
f:2	s:16	l:2.9	u:2.9	c:1	r:5
f:2	s:17	l:2.9	u:2.9	c:2	r:1
f:2	s:18	l:3	u:3	c:0	r:4
f:2	s:19	l:3	u:3	c:1	r:6
f:2	s:20	l:3	u:3	c:2	r:9
f:2	s:21	l:3.1	u:3.1	c:0	r:3
f:2	s:22	l:3.1	u:3.1	c:1	r:3
f:2	s:23	l:3.1	u:3.1	c:2	r:3
f:2	s:24	l:3.2	u:3.2	c:0	r:4
f:2	s:25	l:3.2	u:3.2	c:1	r:2
f:2	s:26	l:3.2	u:3.2	c:2	r:5
f:2	s:27	l:3.3	u:3.3	c:0	r:2
f:2	s:28	l:3.3	u:3.3	c:1	r:1
f:2	s:29	l:3.3	u:3.3	c:2	r:2
f:2	s:30	l:3.4	u:3.4	c:0	r:9
f:2	s:31	l:3.4	u:3.4	c:2	r:2
f:2	s:32	l:3.5	u:3.5	c:0	r:5
f:2	s:33	l:3.6	u:3.6	c:0	r:2
f:2	s:34	l:3.6	u:3.6	c:2	r:1
f:2	s:35	l:3.7	u:3.7	c:0	r:2
f:2	s:36	l:3.8	u:3.8	c:0	r:2
f:2	s:37	l:3.8	u:3.8	c:2	r:2
f:2	s:38	l:3.9	u:4.4	c:0	r:5

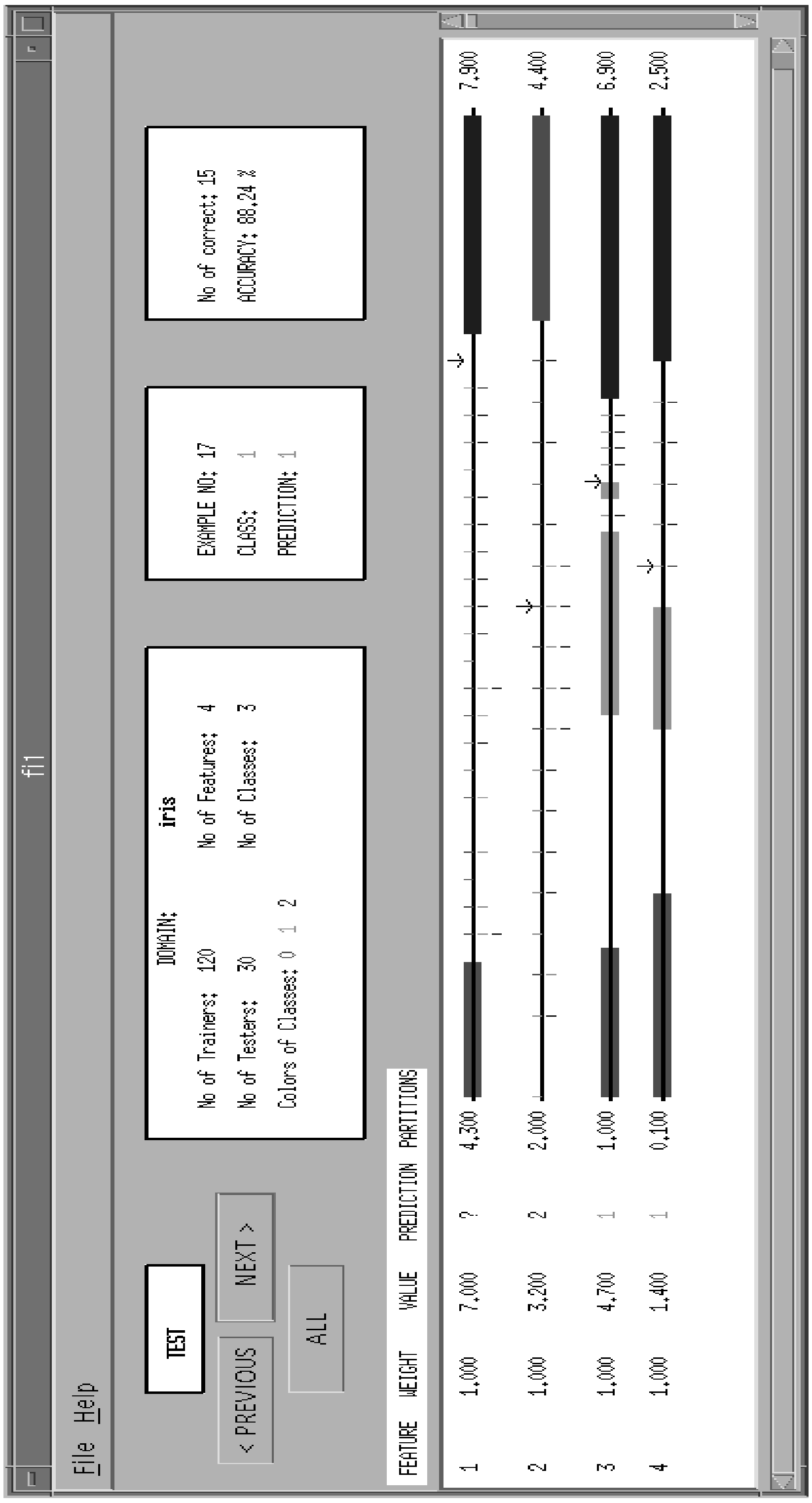
Figure 4.20. Intervals of iris domain on the second feature.

f:3	s:1	l:1	u:1.9	c:0	r:40
f:3	s:2	l:3.3	u:4.4	c:1	r:21
f:3	s:3	l:4.5	u:4.5	c:1	r:5
f:3	s:4	l:4.5	u:4.5	c:2	r:1
f:3	s:5	l:4.6	u:4.7	c:1	r:6
f:3	s:6	l:4.8	u:4.8	c:1	r:2
f:3	s:7	l:4.8	u:4.8	c:2	r:1
f:3	s:8	l:4.9	u:4.9	c:1	r:2
f:3	s:9	l:4.9	u:4.9	c:2	r:2
f:3	s:10	l:5	u:5	c:1	r:1
f:3	s:11	l:5	u:5	c:2	r:2
f:3	s:12	l:5.1	u:5.1	c:1	r:1
f:3	s:13	l:5.1	u:5.1	c:2	r:6
f:3	s:14	l:5.2	u:6.9	c:2	r:30

Figure 4.21. Intervals of iris domain on the third feature.

f:4	s:1	l:0.1	u:0.6	c:0	r:40
f:4	s:2	l:1	u:1.3	c:1	r:20
f:4	s:3	l:1.4	u:1.4	c:1	r:6
f:4	s:4	l:1.4	u:1.4	c:2	r:1
f:4	s:5	l:1.5	u:1.5	c:1	r:8
f:4	s:6	l:1.5	u:1.5	c:2	r:2
f:4	s:7	l:1.6	u:1.6	c:1	r:2
f:4	s:8	l:1.6	u:1.6	c:2	r:1
f:4	s:9	l:1.7	u:1.7	c:1	r:1
f:4	s:10	l:1.7	u:1.7	c:2	r:1
f:4	s:11	l:1.8	u:1.8	c:1	r:1
f:4	s:12	l:1.8	u:1.8	c:2	r:7
f:4	s:13	l:1.9	u:2.5	c:2	r:30

Figure 4.22. Intervals of iris domain on the fourth feature.



DOMAIN: **iris**

No of Trainers: 120 No of Features: 4

No of Testers: 30 No of Classes: 3

Colors of Classes: 0 1 2

EXAMPLE NO: 17

CLASS: 1

PREDICTION: 1

No of correct: 15

ACCURACY: 88.24 %

FEATURE	WEIGHT	VALUE	PREDICTION	PARTITIONS	
1	1.000	7.000	?	4.300	7.900
2	1.000	3.200	2	2.000	4.400
3	1.000	4.700	1	1.000	6.900
4	1.000	1.400	1	0.100	2.500

Figure 4.23. Feature intervals constructed by the FI1 algorithm for the iris dataset.

classifying each test instance are displayed along with the current test example feature values and associated class label. Prediction of each feature with its associated weight: feature weights in the FI1 and FI2 algorithms and interval weights in the FI3 and FI4 algorithms are also displayed. The constructed intervals can be saved into a text file from the menu with corresponding lower and upper bounds, associated class and representativeness and relative representativeness counts (in FI1 and FI2, respectively) or associated weights (in FI3 and FI4). The disjoint intervals of the iris dataset is as in the Figure 4.19, 4.20, 4.21, 4.22.

The concept descriptions for the iris dataset learned by the FI3 algorithm are presented in Figure 4.23.

4.5 Summary

In this chapter, details of the FIL algorithms has been presented. Their general characteristics are discussed considering important dimensions classifying machine learning techniques. Also, the user interface of all FIL algorithms are described.

In the FIL algorithms, a feature interval can be defined as *generalized values* that may cover several feature values. Intervals (single-class) are disjoint, however, at multi-class points, overlapping point intervals are constructed. Once the feature intervals are learned, a test example can be classified on each feature dimension by means of these intervals by a voting scheme.

The FIL algorithms assume that similar feature values have similar classifications. The voting mechanism in the FI1 and FI2 algorithms is based on a weighted-voting scheme, with prior knowledge. However, without prior weights, features will have equal relevances for classification decisions. On the other hand, although the FI3 and FI4 algorithms are based on weighted-voting scheme, these weights are set to intervals internally. Hence, the FI3 and FI4 algorithms require no user tuning of parameters such as generalization ratio or global feature weight-adjustment rate. Their primary goal is to maintain perfect consistency with the initial training set.

Chapter 5

Evaluation of the FIL Algorithms

In this chapter, both complexity analyses and empirical evaluations of the FIL algorithms are given. First, training and classification of a single instance time complexities are given. Next, the empirical evaluations are presented on some real-world datasets for comparison with some similar algorithms such as NBC, CFP, k-NN, and k-NNFP. Later, the experiments on artificially generated datasets are discussed. The goal of these experiments is to demonstrate performances of the FIL algorithms. Also, some experimental results are presented for the evaluation of the feature weighting methods proposed in this thesis. Experiments described in this chapter are designed to determine the behavior of the FIL algorithms on irrelevant features, noisy instances and missing feature values.

5.1 Complexity Analysis

In this section, the FIL algorithms are analyzed in terms of space and time complexities. Time complexity analyses are presented for training process and classification of single test instance.

Space Complexity Analysis: In the training phase of the FIL algorithms, disjoint feature intervals for concept descriptions are constructed on each feature dimension. The space required for training with m instances on a domain with n features is proportional to $m \cdot n$ at worst case. However, on the average, it should be less than $O(m \cdot n)$ since feature intervals may contain several feature values. If the average number of intervals constructed on a feature dimension is i, then the average space complexity of the FIL algorithms will be $O(i \cdot n)$. If feature intervals contain several values, the storage requirement of the FIL algorithms will be less than the k-NN and k-NNFP algorithms since k-NN stores all instances in memory as conjunctions of feature values and k-NNFP stores them as feature projections. Although the learned feature intervals will not be the same as the CFP algorithm since it learns in an incremental way, the storage requirement may be nearly the same. The NBC also stores all training instances to find the class distributions.

Time Complexity of Training: As mentioned before, all instances are stored on each feature dimension as their feature projections initially. Feature projections on a feature dimension are sorted with time complexity $O(m \cdot \log m)$. So, sorting all feature values has time complexity $O(m \cdot n \cdot \log m)$ for n features. Then disjoint feature intervals are constructed by examining these sorted feature projections on each feature dimension with time complexity $O(n \cdot m)$. Therefore, the training time complexity of the FIL algorithms is $O(n \cdot m \cdot \log m + n \cdot m) = O(n \cdot m \cdot \log m)$ for training a dataset with m instances described by n features.

Time Complexity of a Single Classification: During the preclassification, the *search-interval*$(f, value)$ searches the interval containing feature value of the test instance on the feature dimension f. by binary search to determine the prediction of that feature. The number of intervals on a feature dimension is at most equal to the number of training instances, m. Hence, the worst case time complexity of this search process is $O(\log m)$ for a feature. Since the final classification is based on the prediction of each feature, single instance classification time complexity of the FIL algorithms is $O(n \cdot \log m)$.

We have presented training and classification time complexities of the k-NNFP and k-NN algorithms in Section 3.3.2. The training complexity of the k-NNFP algorithm is nearly the same as the FIL algorithms whereas the training time complexity of the k-NN algorithm is $O(n \cdot m)$ for just storing instances in memory. Complexity analyses of FIL algorithms indicate that these algorithms classify unseen instances much more faster than the k-NN like algorithms.

5.2 Empirical Evaluation of the FIL Algorithms

In this section, empirical evaluations of the FIL algorithms on real-world datasets which are widely used in the field of machine learning from the UCI-Repository [47] and two new datasets constructed in this thesis. We will also evaluate the FIL algorithms on artificial datasets. The first section describes the methodologies used in the experiments. Next, the performance of the FIL algorithms on real-world datasets are presented. In the third section, some experiments are described on artificial datasets.

5.2.1 Testing Methodology

This section briefly describes the methodologies used in the machine learning experiments. The goal of experiments is to better understand behaviors of learning algorithms, hence their causes, as in other sciences. This will lead to empirical laws that can aid the process of theory formation and theory evaluation.

Improved performance is the major aim of learning algorithms [34]. 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.*

For supervised concept learning tasks, the most commonly used metric is the percentage of correctly classified 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 [23]. 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.*

Cross-Validation: In this technique, dataset is partitioned into k mutually disjoint subsets with the same cardinality. The $k-1$ of these sets are used as the training set, the remaining one is used as the test set. This process is repeated k times once for each subset being the test set. Classification accuracy is measured as the average accuracy on all the test sets. The union of the all test sets equals to the whole dataset. This is called as k-fold cross-validation.

Leave-one-out: This technique is a special case of k-fold cross-validation taking $k = m$. 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 [22, 35]. While leave-one-out is a preferred technique, for large datasets it may be computationally expensive [32].

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 FIL algorithms. In the following subsection, the performance of the FIL algorithms will be given in terms of classification accuracy. In this thesis, 5-fold cross-validation technique is used to report the classification accuracies of FIL algorithms and compare them with other methods. 5-fold cross-validation enables 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.

Table 5.1. Accuracy results (%) of the FIL algorithms on real-world datasets. SFA-FIx and HFP-FIx show the weighted versions of the FI1 and FI2 algorithms.

Dataset	FI1	SFA-FI1	HFP-FI1	FI2	SFA-FI2	HFP-FI2	FI3	FI4
arrhythmia	55.08	55.08	55.08	55.00	55.08	55.08	55.08	55.68
bcancerw	95.72	95.72	95.72	96.44	96.30	95.72	95.86	97.00
cleveland	78.22	80.18	78.86	80.52	83.14	81.26	78.86	80.50
dermatology	36.90	41.40	35.00	44.60	43.34	36.72	73.33	79.02
diabets	65.76	66.16	65.50	64.84	63.80	65.50	68.74	69.76
glass	49.88	49.92	43.86	49.50	53.22	48.96	56.98	45.36
horse	64.12	65.48	64.38	73.64	74.46	64.38	65.24	76.36
hungarian	69.02	70.04	69.36	80.94	81.92	68.15	68.00	75.48
ionosphere	87.18	87.16	87.16	84.90	85.46	87.16	87.74	88.88
iris	86.66	90.66	89.30	88.00	91.32	89.60	90.66	90.66
liver	54.78	57.40	54.22	53.92	56.52	54.92	57.98	59.72
musk	61.96	62.16	61.56	71.04	72.50	61.76	71.02	73.34
wine	82.54	88.16	88.14	87.62	89.90	87.94	91.6	89.92

5.2.2 Experiments with Real-World Datasets

For empirical evaluations of the FIL algorithms, some real-world datasets from the collection of UCI-Repository [47] and two new real-world datasets constructed in this thesis are used . These domains provide the FIL algorithms with opportunity of comparison with other similar learning algorithms. Also they demonstrate the applicability of the FIL algorithms to real-world problems. The real-world datasets are explained in Appendix A. These datasets are used for the comparison of the FIL algorithms to the NBC, CFP, k-NN and k-NNFP algorithms. The FIL algorithms use feature weights learned by the HFP and SFA methods in the experiments described here. The CFP algorithm was run for $D_f = 0.1$ and $\triangle = 0$.

Table 5.1 presents the results of experiments on these real-world datasets which are conducted by using 5-fold cross-validation evaluation technique for the FIL algorithms and SFA and HFP feature weighting methods. K is taken as 5 in these experiments since the k-NN and k-NNFP algorithms give almost the best accuracies for $k = 5$. The results of experiments of the NBC, CFP, k-NN and k-NNFP algorithms are summarized in Table 5.2. Both FI1 and FI2 algorithms achieve almost same accuracies. The FI3 and FI4 algorithms are

Table 5.2. Accuracy results (%) of the FI4, NBC, CFP, k-NNFP and k-NN algorithms on real-world datasets.

Dataset	FI4	NBC	CFP	k-NNFP	k-NN	Baseline (%)
arrhythmia	55.68	3.12	55.09	55.08	58.20	55
bcancerw	97.00	97.28	95.71	96.16	96.58	66
cleveland	80.50	80.52	74.24	78.88	83.80	54
dermatology	79.02	43.98	35.64	59.42	91.64	27
diabets	69.76	71.24	65.49	67.70	73.18	65
glass	45.36	52.34	52.28	60.72	66.30	36
horse	76.36	81.24	64.94	71.74	80.44	63
hungarian	75.48	79.90	71.04	76.16	82.26	64
ionosphere	88.88	87.74	87.47	87.46	83.20	64
iris	90.66	92.00	86.66	91.30	94.66	33
liver	59.72	60.30	56.23	58.26	64.92	58
musk	73.34	2.10	60.28	71.22	67.88	57
wine	89.92	95.50	86.44	93.24	96.04	40

superior to the FI1 and FI2 algorithms. It is seen from the tables that FIL algorithms achieve high accuracies as much as previous algorithms on many of these datasets. The k-NN algorithm gives maximum accuracy in almost all datasets. The FI4 algorithm usually outperforms the other FIL algorithms. The SFA-FI2 algorithm gives high accuracy as much as the k-NN algorithm for cleveland dataset. Also, the empirical evaluation of the CFP algorithm is presented in [64] and the k-NN and k-NNFP algorithm in [7].

Table 5.3 shows the average running times of the FIL algorithms across the NBC, CFP, k-NNFP, and k-NN algorithms. Since all FIL algorithms give almost equal average running times, they are represented in the table under the name *FIL*. It is seen that the running times of the FIL algorithms are relatively smaller than the other algorithms. This verifies the training and classification time complexities presented in Section 5.1. Although the classification accuracy differ about 5% points, the running time of the k-NN algorithm is much higher than the other algorithms.

Table 5.3. The Average Time (msec) required for the FIL, NBC, CFP, k-NN and k-NNFP algorithms on real-world datasets.

Dataset	FIL	NBC	CFP	k-NNFP	k-NN
arrhythmia	3,527	21,641	11,886	3,229	18,135
bcancerw	399	925	340	364	3,276
cleveland	221	214	292	217	772
dermatology	183	197	347	189	528
diabets	375	1,145	610	297	3,294
glass	130	134	118	105	318
horse	494	641	479	465	1,400
hungarian	287	146	348	255	631
ionosphere	596	882	1,232	522	2,339
iris	45	17	250	44	108
liver	129	148	267	114	607
musk	3,477	9,529	11,279	2,740	18,744
vehicle	586	4,787	818	2,012	14,441
wine	113	79	73	299	600

5.2.3 Experiments with Artificial Datasets

To cope with noisy and incomplete data is an important criteria for a learning system to be used in real-world applications [40]. One important point for a learning system is presence of irrelevant features [9]. Therefore, artificial datasets are important to study the effects of irrelevant features, noise in the domain, and missing feature values since artificial datasets allows to test the system in a more controlled way. In order to empirically demonstrate the behaviors of the FIL algorithms on artificial datasets, we conduct some experiments. Concept descriptions for these artificially generated datasets are represented in the form of possibly overlapping hyperrectangles. We will explain how we generated these datasets in each section with the descriptions of experiments. Section 5.2.3.1 describes and presents the results of experiments with increasing number of irrelevant features. Next, increasing noise level is studied for the FIL algorithms. Then, increasing ratio of missing feature values is tested. In these experiments, the CFP algorithm was run for $D_f = 0.1$ and $\triangle = 0$.

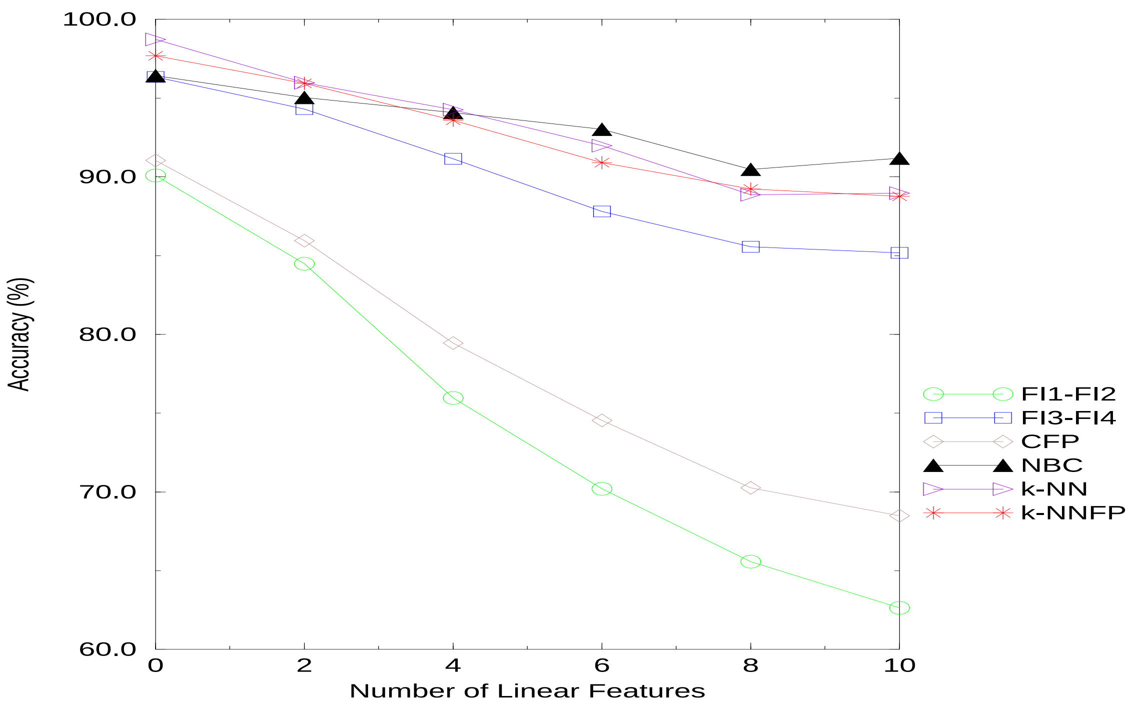


Figure 5.1. Accuracy results of the FIL, CFP, NBC, k-NN, k-NNFP algorithms on domains with irrelevant attributes.

5.2.3.1 Experiments with Increasing Number of Irrelevant Features

Real-world datasets may contain unequally relevant features. For example, medical domains usually contain more information than is actually required for distinguishing one disease from others. Most probably some of these features are not as relevant as the others [39].

The voting mechanism used in the FIL algorithms allows correct classifications in the presence of irrelevant features to a certain extent.

To investigate the behaviors of the FIL algorithms in the presence of irrelevant features, we conducted a series of experiments. We generated six datasets with increasing number of irrelevant features from zero to ten. Each instance is described by four relevant features and a number of irrelevant ones. Concept descriptions are represented by hyperrectangles in four (relevant) dimensional space, the values for irrelevant features are randomly generated. These artificial datasets are also used for the evaluation of k-NNFP and k-NN algorithms in Section 3.2.2.2. We ran these algorithms 50 times on these six datasets generated randomly each time. We have compared the average results of the

FIL algorithms on these artificial datasets with the average results of the CFP, NBC, k-NNFP and k-NN.

The accuracy results of artificial datasets with increasing number of irrelevant features are plotted in Figure 5.1. The FI1 and FI2 algorithms give about the same accuracy results for these experiments. Similarly, the FI3 and FI4's behavior is almost the same in these experiments. Note that the feature values range from 0 to 10 as continues. However, in order to have some multi-class points, values are generated between 0 and 100 as integers and divided by 10 (i.e., 85 /10 = 8.5). As seen from the table, assigning weights to intervals outperforms the FI1 and FI2 algorithms in which features have equal relevance. The NBC algorithm achieves the greatest accuracy in the presence of irrelevant features. The performance of the CFP algorithm is worse than the FI3 and FI4 algorithms. The k-NNFP and k-NN algorithms' behavior on these datasets is almost the same.

5.2.3.2 Experiments with Increasing Noise Level

In this section, noise tolerance of the FIL algorithms are investigated. There are two major types of noise that can be found in real-world datasets: feature (attribute) noise, and classification noise [3, 11, 14, 24, 63]. Feature noise can be defined as incorrect feature value information. Classification noise involves corruption of the class label of an instance.

Quinlan demonstrated that feature noise, occurring simultaneously in all features describing the instances, can result in faster degradation in classification accuracy than might noise only in the class label [51]. Therefore, we studied the feature noise in our experiments with artificial domains, where feature values only in the training set are replaced with a randomly selected value in the feature domain with a fixed probability, called *noise ratio*.

The artificial dataset with four relevant features and no irrelevant features used in the experiments with increasing irrelevant features is used in this section in order to study the effect of increasing noise level.

Figure 5.2 presents achieved accuracy of the FIL algorithms for comparison

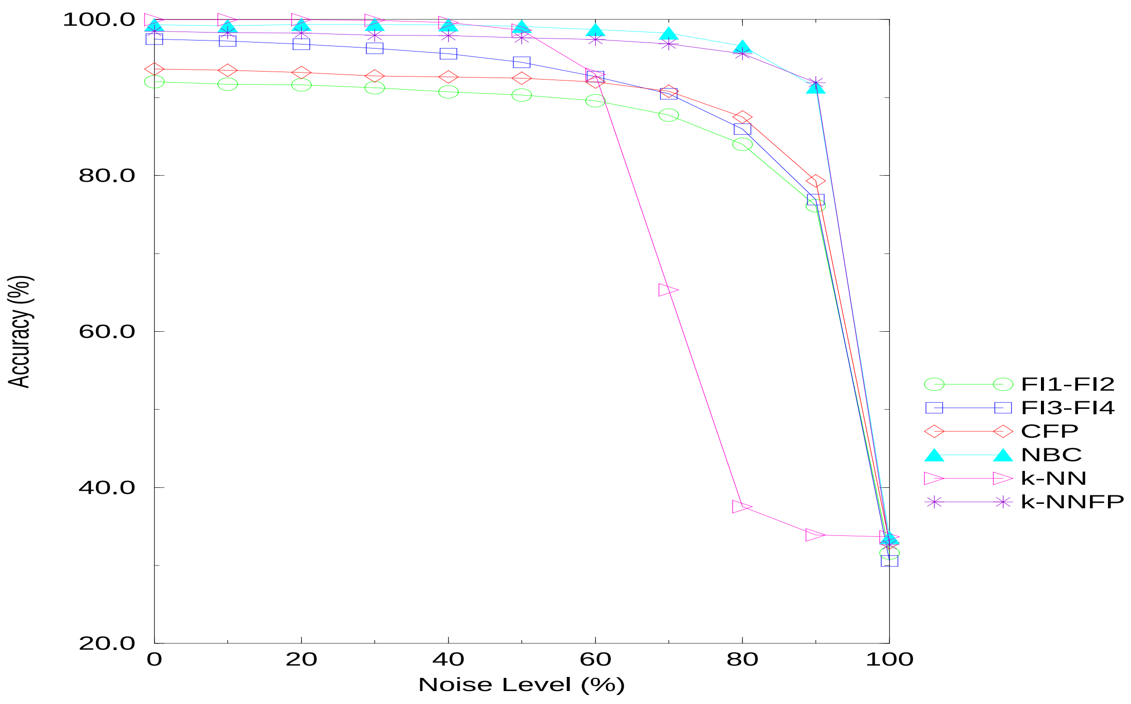


Figure 5.2. Accuracy results of the FIL, CFP, NBC, k-NN, k-NNFP algorithms on domains with increasing noise level.

to the CFP, NBC, k-NNFP and k-NN algorithms. The results are the averages of the 50 runs of artificial datasets in which noisy feature values are randomly replaced with other possible values on that feature dimension. In this thesis, to handle noisy feature values, we introduced the FI3 and FI4 algorithms that construct disjoint feature intervals by weighting them for classification. The results of the experiments indicate that both FI3 and FI4 algorithms are successful than the FI1 and FI2 algorithms. Actually, there is no significant difference among all the algorithms on noisy domains up to 60% noise level, the accuracy of the k-NN algorithm sharply decreased after this point. Other algorithms are robust up to 80% noise level.

5.2.3.3 Experiments with Increasing Ratio of Missing Values

Most of the real-world data sets contain missing attribute values. In the literature, some methods are proposed to handle instances containing missing feature values [26, 52, 53, 54, 55]. These methods can be summarized as:

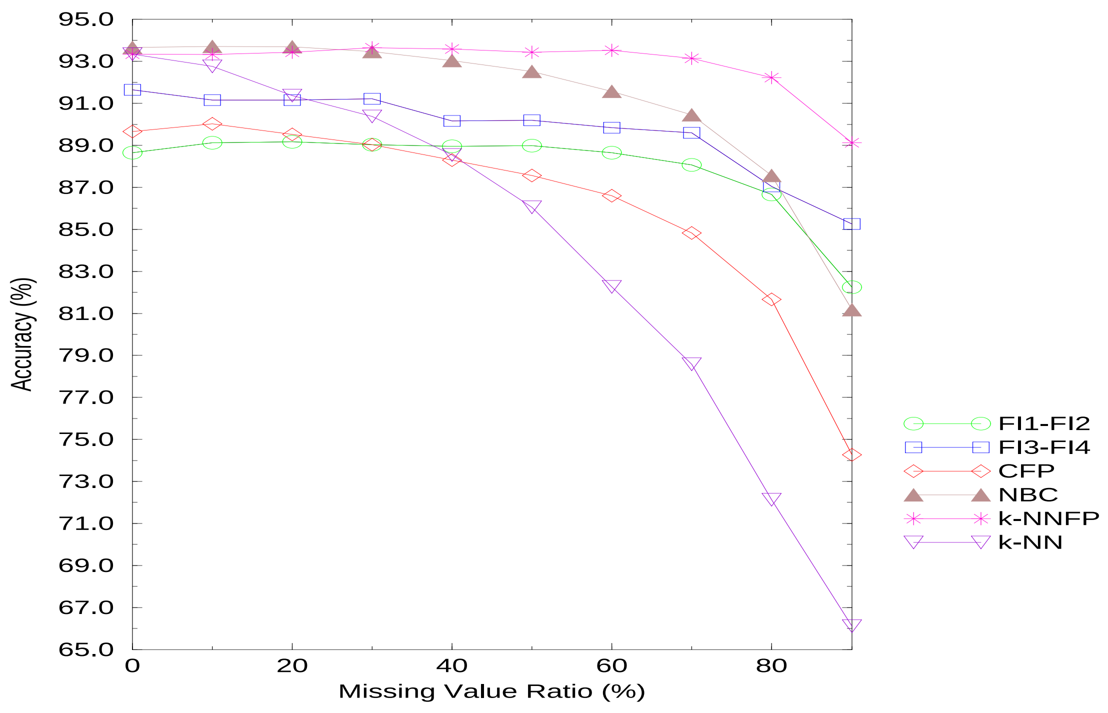


Figure 5.3. Accuracy results of the FIL, CFP, NBC, k-NN and k-NNFP algorithms on domains with increasing ratio of missing feature values.

- Ignoring examples which have unknown feature value.
- Assuming an additional special value for unknown attribute values. This can lead to an anomalous situation.
- 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.

The method employed in the FIL algorithms for handling feature values is similar to the first method mentioned above. However, it states that an incomplete instance is ignored whereas the FIL algorithms simply ignores missing feature values since they process each feature separately. Similarly, the CFP, NBC, and k-NNFP handles missing values. On the other hand, the k-NN algorithm tries to determine the value of an unknown attribute value using probability distribution of the known values of a feature. One advantage of simply ignoring missing feature values is that it allows reduction in training and classification time.

Figure 5.3 presents the accuracies obtained from the experiments with different amounts of unknown (*missing*) attribute values. The k-NNFP algorithm achieved significantly better accuracy than the others. The most affected algorithm by the presence of missing feature values is the k-NN algorithm, as expected because it tries to fill in missing values. Up to 70% missing value, FIL algorithms achieve the same accuracy. Therefore, the FIL algorithms are robust to the missing feature values.

5.3 Summary

The results from the experiments in this chapter support the following conclusions.

- faster classification times with feature projections knowledge representation
- weighted-voting in the FIL algorithms is more tolerant to the presence of irrelevant features
- feature projections knowledge representation is quite successful in handling missing feature values.
- k-NNFP and FIL algorithms are robust to the missing feature values (up to 70%).

Chapter 6

Conclusions and Future Work

In this thesis, a new classification algorithm, called k-NNFP has been presented. In this algorithm, the classification knowledge is represented in the form of sets of feature projections of the training data separately on each feature dimension. The classification of an unseen instance is based on a majority voting taken on the classifications made on the basis of individual feature projections.

We have compared the k-NNFP algorithm with the k-NN algorithm in terms of classification accuracy and running time on both real-world and artificial datasets. On real-world datasets, the k-NNFP algorithm achieves comparable accuracy with the k-NN algorithm. On the other hand, the average running time is much less than that of the k-NN algorithm. The majority voting in the classification process of the k-NNFP algorithm reduces the intrusive effect of the irrelevant features. This claim has been justified on artificial datasets.

We treated all features as equivalent in the k-NNFP algorithm. However, all features may not have equal relevance in real-world applications, even some features may be completely irrelevant. In order to determine features' relevances, two feature weight learning algorithm have been proposed for the learning algorithms that use feature weights. The first method, called HFP, assigns high weight values to features on which the projections of instances of the same class are located close to each other, resulting in homogeneous distribution.

The SFA method assigns a weight as the classification accuracy that would have been obtained if only that feature were used in the classification. These techniques have been evaluated on the weighted k-NNFP algorithm. The SFA method learned weights that helped the k-NNFP algorithm achieve higher accuracies. The reason for this success is due to the feedback received from the k-NNFP algorithm.

In this thesis, we have also developed several batch learning algorithms called FIL algorithms for Feature Interval Learning. These algorithms use feature projections of the training instances for the representation of the classification knowledge induced. These are FI1, FI2, FI3, FI4 algorithms with slight differences. Linear feature projections are generalized into disjoint intervals during the training phase. The classification of an unseen instance is based on a majority voting among individual predictions of features. Feature projections knowledge representation in these algorithms provide them with much faster classification. In fact, majority voting reduces the intrusive effect of irrelevant features or noisy feature values.

The FIL algorithms have been compared with the NBC, CFP, k-NN and k-NNFP algorithms empirically. The FI3 and FI4 algorithms are found to be superior to the FI1 and FI2 algorithms. In addition, the FI1 and FI2 algorithms gives better accuracies when the SFA feature weighting method is integrated. Although the FIL algorithms achieve comparable accuracies with other algorithms about 5% less than the k-NN algorithm, their average running times are much more less than the k-NN algorithm.

Feature projections for knowledge representation has the following advantages for the learning algorithms:

- plausible
- no need for normalization of feature values
- simply ignoring missing feature values
- faster classification times

The major disadvantage of this representation is that concept descriptions involving a conjunction between two or more features cannot be represented. Actually, the whole is more than sum of its components. Therefore, the FIL algorithms are applicable to concepts where each feature, independent of other features, can contribute to the classification of the concept. In fact, this is the nature of the most real-world datasets. They are not applicable to domains where all of the concept descriptions overlap, or domains in which concept descriptions are nested.

As a future work, we plan to investigate the HFP and SFA feature weight learning algorithms on artificial datasets. For overlapping concept descriptions, batch learning algorithms whose knowledge representation is in the form of overlapping feature intervals can be developed. Another research direction is to investigate learning concept dependent feature weights for the learning algorithms that use feature projections for knowledge representation. Moreover, feature weights are learned using genetic 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*, pp: 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*, pp: 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 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. pp. 12-19, 1996.

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

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

[10] D. Angluin, Queries and Concept Learnings, *Machine Learning*, 2(2):312-342, 1988.

[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.), pp. 21-29, 1995.

[13] J.G. Carbonell, editor, *Machine Learning: Paradigms and Methods*, The MIT Press, 1990.

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

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

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

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

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

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

[20] P.J. Denning, The Science of Computing, *American Scientist*, volume: 77, pp: 216-219, 1989.

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

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

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

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

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

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

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

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

[29] 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. pp. 293-301, 1994.

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

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

[32] L. Kanal and Chandrasekaran, On Dimensionality and Sample Size In Statistical Pattern Classification, *Pattern Recognition*, pp: 225-234, 1971.

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

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

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

[36] P. Langley, An Analysis of Bayesian Classifiers, In *Proceedings of the tenth National Conference on Artificial Intelligence*, San Jose: AAAI Press, 1992.

[37] P. Langley, Induction of Selective Bayesian Classifiers, In *Proceedings of the tenth Conference on Uncertainty in Artificial Intelligence*, Seattle, WA: Morgan Kaufman, 1994.

[38] P. Langley, and S. Sage, Oblivious decision trees and abstract cases, in "Working Notes of the AAAI-94 Workshop on Cased-Based Reasoning", AAAI Press, Seattle, pp. 113-117, 1994.

[39] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm, *Machine Learning*, 1:47-80, 1986.

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

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

[42] R.S. Michalski and R.L. Chilausky, Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two methods of Knowledge Acquisition In the Context of Developing an Expert System

for Soybean Disease Diagnosis, *International Journal of policy Analysis and Information Systems*, 4, 1980.

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

[44] T.M. Mitchell, An Analysis of Generalization As a Search Problem, In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, vol:1, pp: 577-582, 1979.

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

[46] T.M. Mitchell, The Need for Biases in Learning Generalizations, In J.W. Shavlik and T.G. Ditterich, editors, *Readings in Machine Learning*, pp: 184-191. Morgan Kaufman, San Mateo, CA, 1990.

[47] 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.

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

[49] J.R. Quinlan, Discovering Rules From Large Collections of Examples: A Case Study, In D. Michie (Ed.),*Expert Systems in the Microelectronic Age*, Edinburgh: edinburgh University Press, 1979.

[50] J.R. Quinlan, Learning Efficient Classification Procedures and Their Application to Chess and Games, In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning, An Artificial Intelligence Approach*, Los Altos: Morgan Kaufmann, 1983.

[51] 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.

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

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

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

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

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

[57] L. Rendell, A General Framework for Induction and a Study of Selective Induction. *Machine Learning*, 1:177-226, 1986.

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

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

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

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

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

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

[64] 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.

[65] 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.

[66] 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.

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

[68] L. G. Valiant, A theory of learnable. *Communications of the ACM*, 27:1134-1142, 1984.

[69] S.M. Weiss and I. Kapouleas, Am Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods, In *Proceedings of the Eleventh International Joint conference on Artificial Intelligence*, San Mateo, CA:Morgan Kaufmann, 1990.

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

[71] D. Wettschereck and T. G. Dietterich, An Experimental Comparison of the Nearest Neighbor and Nearest-hyperrectangle Algorithms, *Machine Learning*, **9**: 5-28, 1995.

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

Appendix 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	352	279	279	16	0.33	55
bcancerw	699	10	10	2	0.25	66
cleveland	303	13	6	2	0	54
dermatology	157	34	34	6	0.07	27
diabets	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
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 arrhythmia dataset. The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. Class 01 refers to 'normal' ECG classes 02 to 15 refers to different classes of arrhythmia and class 16 refers to the rest of unclassified ones. Currently, there are 352 instances which are described by 279 feature values. There are several missing feature values. Class distribution of this datasets is very unfair as seen from Table A.1. Class 01 (normal) is the most frequent one. 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 second dataset constructed in this thesis current contains 157 instances described by 34 feature values to distinguish dermatological illnesses from histopathological descriptions for 6 classes (illnesses). These classes are *1-Psoriaris, 2-Dermatit, 3-L. Planus, 4-Posea, 5-Kr.Dermatit, 6-P.Rubrapilaris.* One of the features (age) take values between 0 and 100, while other 35 features take values 0, 1, 2, 3.

Diabets: 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).

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 [23] 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. 276 of the instances are used in training the remaining 69 are used in testing.

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.

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.