University of Miami
Scholarly Repository

Open Access Theses

Electronic Theses and Dissertations

2007-01-01

Treatment of Instance-Based Classifiers Containing Ambiguous Attributes and Class Labels

Hans Mullinnix Holland
University of Miami, cdholland@gmail.com

Follow this and additional works at: https://scholarlyrepository.miami.edu/oa_theses

Recommended Citation
Holland, Hans Mullinnix, "Treatment of Instance-Based Classifiers Containing Ambiguous Attributes and Class Labels" (2007). *Open Access Theses*. 84.
https://scholarlyrepository.miami.edu/oa_theses/84

This Open access is brought to you for free and open access by the Electronic Theses and Dissertations at Scholarly Repository. It has been accepted for inclusion in Open Access Theses by an authorized administrator of Scholarly Repository. For more information, please contact repository.library@miami.edu.

Library Rights Statement

In presenting the Thesis, *Treatment of Instance-Based Classifiers Containing Ambiguous Attributes and Class Labels*, in partial fulfillment of the requirements for an advanced degree at the University of Miami , I agree that the Library shall make it freely available for inspection. I further agree that permission for copying, as provided for by the Copyright Law of the United States (Title 17, U.S. Code), of this Thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this Thesis for financial gain shall not be allowed without my written permission.

I hereby grant permission to the University of Miami Libraries to use my Thesis for scholarly purposes.

Hans M. Holland

Date

UNIVERSITY OF MIAMI

TREATMENT OF INSTANCE-BASED CLASSIFIERS CONTAINING AMBIGUOUS ATTRIBUTES AND CLASS LABELS

By

Hans M. Holland

A THESIS

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Master of Science

Coral Gables, Florida

December 2007

UNIVERSITY OF MIAMI

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science

TREATMENT OF INSTANCE-BASED CLASSIFIERS CONTAINING AMBIGUOUS ATTRIBUTES AND CLASS LABELS

Hans M. Holland

Approved:

Dr. Miroslav Kubat Professor of Electrical and Computer Engineering	Dr. Terri A. Scandura Dean of the Graduate School
Dr. Mei-Ling Shyu Professor of Electrical and Computer Engineering	Dr. Peter Tarjan Professor of Biomedical Engineering

HOLLAND, HANS M. (M.S., Electrical and Computer Engineering)
Treatment of Instance-Based Classifiers Containing Ambiguous Attribute and Class Labels (December 2007)

Abstract of a thesis at the University of Miami.

Thesis supervised by Professor Miroslav Kubat.
No. of pages in text. (78)

The importance of attribute vector ambiguity has been largely overlooked by the machine learning community. A pattern recognition problem can be solved in many ways within the scope of machine learning. Neural Networks, Decision Tree Algorithms such as C4.5, Bayesian Classifiers, and Instance Based Learning are the main algorithms. All listed solutions fail to address ambiguity in the attribute vector. The research reported shows, ignoring this ambiguity leads to problems of classifier scalability and issues with instance collection and aggregation. The Algorithm presented accounts for both ambiguity of the attribute vector and class label thus solving both issues of scalability and instance collection. The research also shows that when applied to sanitized data sets, suitable for traditional instance based learning, the presented algorithm performs equally as well.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Dr. Miroslav Kubat for his patience and expert guidance and aid while I undertook this journey. Through his work as a professor and researcher I have come to respect and enjoy the path being cut through science by people like him. He opened my eyes to a world of research I never thought possible and encouraged me to be better than I thought I could.

I have to thank my family because without their love and support none of this would have been possible.

I would also like to thank my wife Lauren Holland for her love and support in those dark times of frustration and long nights. There are few people that could have put up with what I had to do…

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

Figure **Page**

CHAPTER 1

Introduction

1.1 Ambiguity is all around us

In many classification problems knowledge about a particular outcome is not known with absolute certainty. An argument can be made that in nature nothing can be known with absolute certainty, simply based on the fact that the world around us has so many variables. True reality is a very complex system where it is impractical, if not impossible, to account for all possible cases. Within Machine Learning, like other science disciplines, the focus is to model the real world in order to explain natural phenomena. As scientists and engineers we are charged with explaining how our environment works in a way that can aid man-kind. To this end we look for simple solutions to complex problems that can stand the rigors of scientific experimentation.

This thesis presented brings to light an often overlooked and neglected situation concerning ambiguously defined attributes for an instance based classifier. Ambiguity naturally occurs in real world data domains but is often sanitized for the sake of simplicity. Within the Machine Learning community there is a sense of inertia considering data domains. It has been the culture that domains have the ambiguity removed at time of creation. It is the contention of this paper that a better approach be found concerning this data. Presented here is a simple solution for accounting for ambiguity without the harmful effects of removing or changing the domain's data or underlying data structure.

Webster's Dictionary defines ambiguity as

```
"ambiguity n 1: an expression whose meaning cannot
 be determined from its context
2:  unclearness by virtue of having more than one meaning"
```

For purposes of this thesis we will extend the definition of ambiguity on the second Webster's definition. Ambiguity will be defined as the partial absence of specific knowledge about a topic. Complete absence of specific knowledge we define as ignorance. The ambiguity we are looking to handle is specifically concerns the input or output for pattern recognition problems. In the attempt to quantify ambiguity, an ambiguity measure is presented as well as an updated similarity function for instance based classification. Since ambiguity can exist in the class label as well as the attribute, a class label voting mechanism is necessary. In order to measure the performance of this new approach, a performance metric is suggested dealing with the ambiguity.

Within the area of Machine Learning we are restricting ourselves to systems that produce classification labels via an input vector or instance. In a classification, system ambiguity can occur in two places: 1. input vectors (attribute labels), 2. classification labels. Ambiguity in attribute values has been largely overlooked by the Machine Learning community. In many cases missing or multiple values for a particular input test have been excluded or "sanitized" by picking the most likely value removing any inherent ambiguity from the domain. Ambiguity within the classification labels can be expressed by assigning more than one label to a test. More commonly, the case may be that there is more than one expert responding to a single set of inputs, thus giving more than one accurate answer with varying results. This type of ambiguity is very prevalent in many areas of Machine Learning. The Dempster-Shafer knowledge theory does a good job of defining the knowledge that can be known about a particular outcome [15]. This, along with Vannoorenberghes bagging and boosting methodologies, presents a strong solution using the induction of Belief Decision Trees [18]. In this approach the knowledge is expressed as a weighting factor that can express a spectrum in knowledge from full knowledge to total ignorance on the frame of discernment [18]. The drawback to this approach is that it currently only applies to classification labels and not input vectors as we address here.

The approach taken in this thesis aims at defining and accounting for input vector or attribute label ambiguity, in a robust, manageable way. Ultimately the algorithm presented can be included within commercial classification systems similar to the universal adoption of the C4.5 algorithm [14] designed by J. Ross Quinlan.

Ambiguity can arise not just in the classification label but also in the input vectors of the system as well. This ambiguity is a very important part of the data set. For instance-based classifiers dealing with missing values has always been a difficult task. If the attribute in question is ignored then that can possibly unfairly weight the rest of the vector. Ignoring missing values can also produce over-fitting or under-fitting for systems with few training examples. A missing value is significant in the fact that it is ”missing” and can show total ignorance about that particular attribute, instead of ”nothing.” This can not be overlooked as insignificant in the analysis of expert systems. By accounting for ambiguity in the input vectors of such a system, you can easily bring to light new data sets that more accurately model a real world system.

1.2 Statement of Problem

The problem statement for this thesis is the treatment of ambiguity present in the input vector of an expert system, of the type instance-based classifier. In this frame of reference it is assumed that a missing value reflects complete ambiguity about an input vector. With no affinity toward one attribute label, the value is simply the set of all possible values for the attribute. For discrete data this is manageable, but in the case of a continuous attribute special considerations are made as discussed later. In order to keep the discussion focussed, discrete valued attribute labels are handled with the k-Nearest Neighbor classifier methodology.

Missing values have been dealt with for many algorithms, but for the nearest neighbor classifier the accepted rule was 1. to throw out the instance from the sample, 2. simply count the attributes being the farthest from the compared data point, or 3. replace the unknown with the most common value present in the data set. Sometimes the fact that a particular attribute has a missing value can be significant, because it can mean there is a lack of knowledge for that set of inputs. If a situation was being modeled that each successive input was dependent on its predecessor then the value of the attribute, missing or otherwise, is very significant in the outcome of the classification. By removing or artificially setting the values, as is common with one of the mentioned "unknown value" approaches, erroneous generalizations can be drawn from the outcome.

The ambiguous attribute/classification label problem grew out of a real world problem dealing with the evaluation of Computer Network Intrusion Detection System (IDS) evaluations. In this problem we were looking for a quick and easy way of evaluating a series of IDSs and determining which one provides the best solution. In brief, an IDS is used to determine if an event taking place in a Computer Network system is considered a malicious attack or normal operation. These systems are used to safeguard our computer networks against potential attacks. We are interested in finding the system with the fewest false positive classifications of possible network attacks. We can collect information about the attacks and classify as a potential attack. For this type of domain the data can be ambiguous due to the crossover that exists in the types of attacks that are possible. A presentation and explanation of the data domain is presented in later chapters.

In order to accurately state the problem being addressed by this thesis, the following definitions and assumptions need to be made.

1. Within the field of Machine Learning we are talking about the subset applied directly to instance-based classifiers, and most specifically to the k-Nearest Neighbor Classifier (k-NN).

2. Ambiguity is a term meaning some level of knowledge about an outcome of some task. This ambiguity can be a range from total ignorance to complete knowledge. The task can be reading an instrument or the outcome of an expert system such as a decision tree or other classification system.

3. For the scope of this thesis we are focusing on the ambiguity that can arise in the input vector and classification label for instance-based classification with k-NN. The implementation of the algorithm discussed applies to instance-based classifiers specifically but can be generalized and applied to many other expert systems where similarity calculations are made.

There is a need to account for the significance ambiguity can play in the analysis of expert systems by accounting for it in the input vectors of said classification systems.

1.3 Accuracy and ambiguity

We assume the same ambiguity can also be observed in the class labels. For instance, out of classes $C_1 \ldots C_4$, the teacher knows that the example is not from C_3 and not from C_4, but does not know if it is C_1 or C_2.

The fact that class labels are ambiguous makes it necessary to redefine the performance criterion accordingly. Let N be the number of testing examples, let $C_{i,known}$ be the set of class labels of the i-th testing example, and let $C_{i,classif}$ be the set of class labels given to the i-th testing example by the classifier. We evaluate the classification performance of this classifier by the following formula:

$$AmbA = \frac{1}{N}\sum_{i=1}^{N} Acc_i; \quad \text{where} \tag{1}$$

$$Acc_i = \frac{|C_{i,known} \bigcap C_{i,classif}|}{|C_{i,known} \bigcup C_{i,classif}|} \tag{2}$$

Performance of the system and the approaches presented are also evaluated using Micro- and Macro F1, where F1 is the average of the Micro- and Macro- Precision and Recall Calculations. We want to explore what modifications are needed if the k-Nearest Neighbor classification (k-NN) is to be used in such ”ambiguous” domains. Three aspects of classical k-NN classifiers need to be modified if the paradigm is to be applied to ambiguously described data. First of them is the way similarity between two examples is evaluated, the second relates to the voting mechanism, and thirdly the manner in which performance is measured is modified in order to accurately analyze the findings.

1.4 Organization of the Thesis

In this thesis an algorithm dealing with the ambiguity in the attribute labels and classification labels of an instance-based classification system is presented. Background research into the workings of instance-based classifications and dealings with ambiguity by other types of systems is presented as a look at the current state-of-the-art.

The organization of this thesis is as follows: previous research is presented in chapter 2 followed by the formulation of the issue based on said research. In the sections following our algorithm dealing with ambiguity is presented for consideration followed by an in depth explanation of experiments and results obtained by running an implementation of the outlined algorithm. Finally an analysis of the results is presented along with a list of areas of improvements and future work that can be explored.

CHAPTER 2

Previous Research

Classification problems, such as pattern recognition problems, have been a big part of Machine Learning. There are a variety of approaches including, but not limited to, Neural Networks, Decision Tree Algorithms, Bayesian Classifiers, and Instance-Based Learning.

A training instance is defined by a formula that represents the combination of an attribute vector **x** and a class label **C**. In a training set each attribute vector has a class label assigned to it, while testing instances are only defined by the attribute vector of the unseen case. A pattern recognition algorithm looks at the unseen testing attribute vector and attempts to assign a known class label to it. In both cases of training and testing many instances comprise to make a training data set or testing data set, respectively. The table gives an example of a non-ambiguous attribute vector.

Table 1. Ambiguous Attribute Vector

Example **x**	
Attribute Ident.	Value Label
x_1	YELLOW
x_2	LARGE
x_3	SQUARE

As we take a quick survey of possible solutions to the pattern recognition problem we find Neural Networks topping the list. This approach is concerned with two things ”Learning” and ”Generalization” [10]. The principal characteristics of Neural Networks are the ability to ”learn by example and generalize.”

[10]. This type of approach is great for solving pattern recognition problems. In general, neural networks function by first presenting a training set so it can learn any available patterns. A network or algorithm is successful if it can accurately classify a presented pattern, not previously encountered during training. This act of correct classification is also considered successful generalization. This process is very similar in theory to that of other classification approaches, what sets neural networks apart is the fact that these networks are concerned with processing the data and not statistics about the data. Another way this approach differs is the fact that a network of artificial neurons is used to process the data. This is considered closer to how the human brain actually functions. A neural network is composed of many neurons interfaced together. This allows for faster, sometimes even non-linear, approaches to pattern recognition to be taken.

Concerning artificial Neural Networks we will start our detailed explanation with a definition of a Neural Network's building block, the neuron. For the sake of this discussion w neuron can be a processing element, node, or threshold logic unit [10]. A model of a neuron is presented.

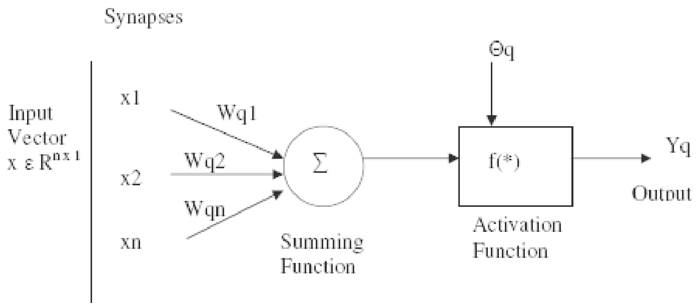

Figure 1. A Neuron

A Neural Network is an interconnected string of these artificial neurons. As can be seen from Figure 1 there are four parts that constitute artificial neurons:

1. Synaptic Weights Input to the synapses is a continuous values input vector xER, with each component described as x_j for j=1,2,...,n for the form x $= [x_1, x_2, \ldots, x_n]^T$. Each component, x_j, is an input to the jth synapses connected to the neuron q with weights w_{qj}.

2. Summing Function This very simply adds the synaptic weights adjusted input vector components together.

   $$u_q = x_1(w_{q1}) + x_2(w_{q2}) + \ldots + x_n(w_{qn}) \tag{3}$$

   Making u_q a linear combination of the inputs for synapses q

3. Activation Function The activation function can be linear, non-linear, continuous-valued, or bipolar. When the function is non-linear it acts to limit the neuron's output amplitude, typically normalizing to the range [0,1] or [-1,1]. The simplest activation function for demonstration purposes is a square wave.

4. Threshold Value The Θ is normally applied externally and acts to lower the cumulative input to the activation function. Theta being threshold and v_q being bias, raising the input. They are related by the following: $v_q = u_q +$ Θ.

Mathematically, the artificial neuron can be described by the output of the linear combiner:

$$u_q = \sum_{j=1}^{n} w_{qj} x_j = \vec{w}_q^T \vec{x} = \vec{x} w_q \tag{4}$$

Where x is described above as the input vector. $w_q = [w_{q1}, w_{q2} \ldots, w_{qn}]^T \in R^{nx1}$

The output of the Activation Function is

$$y_q = f(v_q) = f(u_q \Theta_q) \tag{5}$$

After combining equations 4 and 5 we are left with the simplified equation:

$$y_q = f(\sum_{j=1}^{n} w_{qj} x_j - \Theta_q) \tag{6}$$

Most common Neural Networks and artificial neuron discussed today date to a Hopfield Neuron. A Hopfield Artificial Neuron is described in the following layout [10].

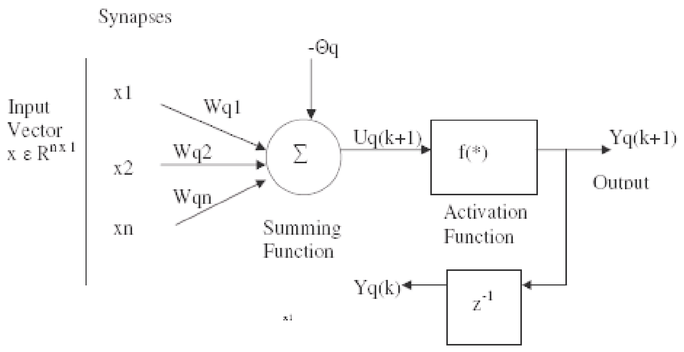

Figure 2. A Hopfield Artificial Neuron

The alteration in the neuron makes it possible to perform asynchronous parallel processing, creating fully interconnected counter-addressable memory with the primary function of retrieving stored patterns from the presentation of noisy or incomplete versions [10].

A decision tree is another way of solving pattern recognition problems. We will discuss briefly the approach taken by Quinlan when constructing the C4.5 induction decision tree algorithm. The induction method used is based on information gain. In general, a decision tree is made up of either a "leaf" node, indicating a class label, or a "decision" node that represents a test to be carried out on a single attribute [14]. Decision trees for classification follow a divide and conquer approach, eventually ending in a class label leaf node.

The construction of a decision tree is as follows:

1. Select an attribute to place at the root node

2. Split the data into subsets along the selected attribute. If all class labels in

a subset are the same then label the node a leaf and end the process.

3. Repeat steps 1 thru 2 for each subset.

Implicit in this algorithm is the mechanism for deciding which attribute to put in the first node, or root node. For the C4.5 algorithm Quinlan chose to use information gain theory to determine the attribute for the decision node. Information gain is a probability based approach. The measure of information is also known as entropy for a system.

$$Entropy(S) \equiv \sum_{i=1}^{c}(-p_i log_2 p_i) \tag{7}$$

where p_i is the proportion of S belonging to class i

So the gain of attribute A for a system S is given by:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Vales(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{8}$$

[12].

We calculate gain for all available attributes and select the one with the largest value to be assigned as the test for the decision node.

Decision trees can be graphically represented as follows:

Decision based rules can be inferred by following from a previously constructed decision tree. Through information gain a tree is developed as discussed above. The tree created represents the training data and is generalized for use on unseen cases. This generalization may include pruning of the tree for efficiency and accuracy. Pruning refers to the process of simplifying a decision tree by replacing decision nodes with leaf nodes. The purpose of pruning is to arrive at a simple tree that can be traversed quickly with a high degree of classification accuracy. C4.5 and later editions have built-in pruning algorithms.

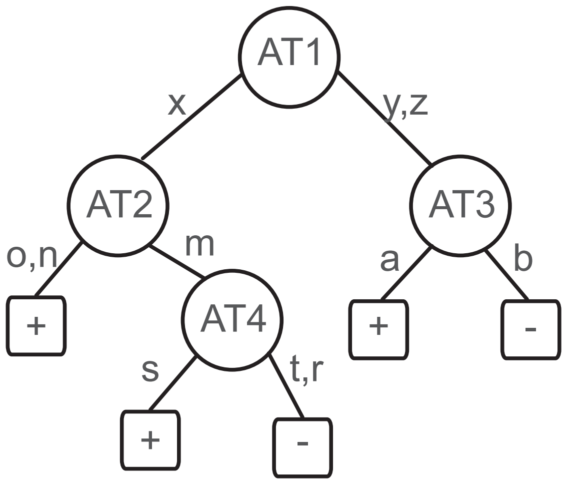

Figure 3. Decision Tree Example

There are two ways to prune: pre- and post-pruning techniques. Pre-pruning implies that when partitioning the training set, for attribute test evaluation, if there is not the coverage for an attribute then it gets replaced with the class label at the node. In C4.5 the amount of coverage used in this test is controlled by a command line input variable. On the other hand, the most common approach taken for post-pruning is called "Error Based Pruning" [14]. This approach is simply replacing a node of a tree with a leaf (class label). The leaf to choose is the one with most percentage of coverage from the training set, thus reducing the amount of error introduced due to misclassification. As can be seen, the differences with regards to post- and pre-pruning are in how the error estimates are estimated (pre-pruning) or calculated (post-pruning).

Decision trees can handle unknown attribute values. Many ideas have been suggested from examining probability distributions in order to tell the most likely candidate value, to excluding the case altogether. C4.5, as an example, ignores the cases with missing values when calculating gain for a particular attribute. When partitioning tests are determined, the C4.5 approach is to develop a probability for the test. If it is known, then the value is 1, if unknown, then a weaker statement is made. The probabilities are used as weights for partitioning purposes. C4.5 assumes that any unknown outcomes are ”distributed probabilistically according to the relative frequency of known outcomes.” [14]

Another probabilistic approach to classification is Bayesian Classification. The Bayesian approach to classification is purely statistical. For the purposes of simplifying calculations, we are going to assume a Nave Bayes Classifier Approach [12]. This approach assumes that attribute values are conditionally independent for a target value. This is a safe assumption since for the purposes of our issues we only care about the relationship attribute values have within the attribute itself, and not spanning attributes. The Nave Bayesian Classifier takes the form:

$$v_{NB} = argmax_{v_i \in V} P(v_j) \prod P_i(a_i|v_j) \tag{9}$$

In normal Bayesian classifiers all possible combinations of cases would need to be seen. With the simplification to the Nave approach little accuracy is lost and the number of possible calculations reduces tremendously. The number of terms used in the above formula is calculated by multiplying the number of distinct attributes by the number of distinct target values [12].

Each possible state of an attribute given a class label has a posterior probability associated with it. These are all calculated for the final steps in the algorithm. For a given attribute the probability of its values yields a given classification. This probability is independent from the probability of other values for the same attribute.

Different from both Bayesian and Decision Tree Classification, Instance based classification takes the approach to pattern recognition from a single instance. As the simplest algorithm to conceptualize and implement, instance based classification does not provide an explicit description of the target function when presented with training examples. Instance based classification algorithms such as k-Nearest Neighbor[3] and locally weighted regression methods[2] assume the data instances can be represented in Euclidean space[12]. When a new instance is presented, a distance calculation is made between the testing instance and every element of the training space. The training instance with the shortest overall distance from the testing example is used to classify the unseen instance. Data sets are generally cleaned or "sanitized" before they are presented. In the case of missing values the attribute is either dropped from the measure, or a most frequent value is substituted. There has been little work done to account for ambiguous attribute values.

Instance based learning in such algorithms as k-Nearest Neighbor is conceptually very easy to understand and implement. Case-based learning, as it is also termed, is simply storing and retrieving similar cases and comparing them to the query case. If the stored (training) case is the same or close in value to the query case then the stored case is used for classification [12]. The main focus of this thesis is on instance-based classification implemented via the k-Nearest Neighbor algorithm. A description of this algorithm follows:

Training Algorithm:

For each training example $(x, f(x))$, add the example to the list training.

Classification Algorithm:

A query instance x_q to be classified

- Let $x_1 \ldots x_k$ be the k instances of the training set that are closest to w_q
- Return

$$f(x_q) = argmax \sum_{i=1}^{k} \delta(v, f(x_i)) \tag{10}$$

where δ is a distance formula for each attribute. For discrete valued functions $\delta(a, b) = 1$ if $a = b$, otherwise $\delta(a, b) = 0$. [12]

In order to perform the tasks associated with a pattern recognition problem, the data set must first be defined by an attribute vector. These attribute vectors consist of values, either discrete or continuous data. The values can be represented as single items, sets of items, or ranges of values [9].

Neural Networks, Decision Tree Algorithms, Bayesian Classifiers, and Instance Based Classification are among the main topics described above. All of these algorithms have the same approach to training data sets. Single value attributes for attribute vectors have been the standard format. Only recently has the scope been widened to account for ambiguity in class labels when working with Decision Trees. The above mentioned approaches assume a sanitized attribute vector and, except for recent work, expect single valued class labels.

Statistically based pattern recognition solutions include decision tree algorithms such as C4.5 [14]. Algorithms such as the C4.5, approach the pattern recognition problem from the perspective of the entire training set. In general, the decision tree classification is a top down, greedy search approach [12]. Since a series of tests are performed to arrive at a class label, the question is asked "How to determine the test as a particular node?" Statistically speaking the attribute that composes a tree node test is the attribute that reflects the largest information gain for an arbitrary subset. Based on the ID3 and C4.5 algorithms there is little room for ambiguity in the attribute values. Progress has been made to account for ambiguity in the class labels. Decision trees based in part or whole on a class label represented by a belief function is called a belief decision tree. Belief Decision trees using the Dempster-Shafer Theory of Evidence seem to be the most promising attempt at a framework to account for ambiguity in class labels [18, 19]. However, in the experiments performed, ambiguity is restricted to the training sets and classification labels only. In most cases data sets provided to the decision tree algorithm are considered sanitized. Missing attribute values are removed in a variety of ways. The most common approach for decision trees is to replace the values with the most common value for a given data set [20].

The Dempster-Shafer approach is a generalization of the Bayesian approach to subjective probabilities [15]. When using the Bayesian theory approach, we are required to calculate the probability for all the questions of interest, whereas with using the Dempster-Shafer belief function, we can base degrees of belief (sometimes, but not always, expressed as probabilities) on the probabilities of a related question. This approach is based on two ideas: the first is the determination of a degree of belief from subject probabilities for a related question. Secondly, this approach is based on Dempster's approach to combining evidence or degrees of belief when they are constructed from independent pieces of evidence. The best way to explain this approach is to cite a simple example in order to convey the nuances.

> To illustrate the idea of obtaining degrees of belief for one question form subjective probabilities for another, suppose I have subjective probabilities for the reliability of my friend Betty. My probability that she is reliable is 0.9, and my probability that she is unreliable is 0.1. Suppose she tells me a limb fell on my car. This statement, which must be true if she is reliable, is not necessarily false if she is unreliable. So her testimony alone justifies a 0.9 degree of belief that a limb fell on my car, but a zero degree of belief (not a 0.1 degree of belief) that no limb fell on my car. This zero does not mean that I am sure no limb fell on my car, as a zero probability would; it merely means that Betty's testimony gives me no reason to believe that no limb fell on my car. The 0.9 and the zero together constitute a belief function. [15]

Another purely statistical approach is a Bayes Classifier. This approach typically requires initial calculations of many probabilities. Nave Bayes Classifiers are very practical for systems with many training examples. This algorithm works on the assumption that ”the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data.” [12] An extension to the Nave Bayes Classifier approach that is less restrictive than the assumed attribute conditional independence is Bayesian Belief Networks [21]. This approach states that conditional independence assumptions can apply to subsets of variables instead of just all of them. Bayesian Classifiers can be augmented to account for class label ambiguity, but again little has been done for attribute values ambiguity. One exception to this rule is the use of the Dempster-Shafer approach to represent attribute ambiguity.

Belief Decision Trees refer to decision trees that are constructed from uncertain data. This uncertainty is restricted to the construction phase of the decision tree, furthermore, to the classification labels for that tree. This all stems from the training set only. The adapted decision tree construction method makes use of the Transferable Belief Model [21] and extended to use the Dempster-Shafer Theory of belief functions [19] for representing unknown class labels. In the approach taken by Vannoorenberghe, the focus is on simplifying the k-class problem into a two class problem with the focus on ”one-against-all” [19] as to comparisons and class labels. This effectively decomposes the k-class problem into k 2-class problems.

The procedure for constructing a Belief Decision Tree (BDT) is as follows. In a traditional decision tree the attribute selection measure is a probabilistic information gain function. However, for BDTs this method must now account for the basic belief assignment for the described class label. The procedure for finding the adapted gain ratio for an attribute is presented as follows:

Notations and Assumptions

1. S: a given set of objects
2. I_j: an instance of the object
3. A = A1, A2, Ak: a set of k attributes
4. D(A_j): the domain of the attribute A_i E A
5. A(I_j): the value of the attribute A for the object I_j
6. S_v^A = {I_j:A(I_j) = v}: the subset of objects which value for attribute A ∈ A is v ∈ D(A_i)
7. $\Theta = C_1, C_2, ...C_n$: the frame of discernment involving the possible classes related to the classification problem.
8. C(I_j): the actual class of the object I_j
9. $m_g^\Theta\{I_j\}$[A](C): denotes the conditional basic belief assignment given to C u Θ relative to object I_j given by an agent g that accepts the A is true.

Construction of a Belief Decision Tree

1. For each object I_j in S, we have a basic belief assignment (bba) $m^\Theta\{I_j\}$ that represents out belief about the value C(I_j). Suppose we select randomly and with equal probability one object in S. What can be said about $m^\Theta\{S\}$, the bba concerning the actual class of that object selected in S? $m^\Theta\{S\}$ is the average of the bbas taken over the objects in the subset S:

$$m^\Theta\{S\}(C) = \frac{\sum_{I_j \in S} m_\Theta\{I_j\}(C)}{|S|} for C \subseteq \Theta \tag{11}$$

2. Apply the pignistic transformation to $m^{\Theta}\{S\}$ to get the average probability $BetP^{\Theta}\{S\}$ on each singular class of the randomly selected instance.

3. Perform the same computation for each subset S_v^A, we get $BetP^{\Theta}\{S_v^A\}$ for $v \in D(A)$, $A \in A$

4. Compute Info(S) and $Info_A(S)$ as done through traditional decision tree algorithms, but using the pignistic probabilities. We get:

$$Info(S) = -\sum_{i=1}^{n} BetP^{\Theta}\{S\}(C_i) log_2 BetP_{\Theta}\S\}(C_i) \tag{12}$$

$$Info_A(S) = \sum_{v \in D(A)} \frac{|S_v^A|}{|S|} Info(S_v^A) \tag{13}$$

$$= -\sum_{v \in D(A)} \frac{|S_v^A|}{|S|} \sum_{i=1}^{n} BetP_{\Theta}\{S_v^A\}(C_i) log_2 BetP^{\Theta}\{S_v^A\}(C_i) \tag{14}$$

Once computed, we get the information gain provided by the attribute A in the set of objects S such that:

$$Gain(S, A) = Info(S) Info_A(S) \tag{15}$$

5. Using the Split Info, compute the gain ratio relative to each attribute A:

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInfo(A)} \tag{16}$$

In the attribute test node the attribute having the highest gain ratio will be selected as the root of the following tree segment.

[21] It can be argued that missing attribute values are the prime case of total ambiguity for that selected attribute. It is the mission of this thesis to move forward to the next logical step, addressing cases of attribute ambiguity. In the approach taken in this thesis, we are treating attribute vectors and class labels of instance based classification for simplicity of experimentation.

Previous research has largely neglected attribute ambiguity and to that end was shown problems of scalability and issues of instance aggregation. In most traditional instance based classifier algorithms ambiguous data in the attribute vector or class label are accounted for by sanitizing the data set to remove them from training examples. Another approach is simply to treat the values as previously unseen possible values, a new class label previously defined. In the sanitation method, the obvious draw back is the fact that the training set loses instances. This is particularly pronounced when there are few records to work from, as viable instances are harder to create or assign the further away we move from modeling a presented system.

Secondly, when another value is created or added from a previously unseen state, the risk of weighting the training set toward one value can increase. In this treatment we are focused on instance based learning systems and the issues of scalability and instance selection. Presented for consideration is a simple solution and evaluation of behaviors observed by a range of experiments run on a variety of data sets. Presented is a treatment of ambiguous training and testing sets.

CHAPTER 3

Formulation of Problem

3.1 Ambiguity in Attribute Vectors

We assume the learner's input consists of a set of training examples in the form (**x**, C(**x**)) where **x** is an example described by a discrete attribute vector $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$ and C(**x**) is a class label. In the particular task addressed by this paper, the concrete attribute values are known only to a certain extent. For instance, such an example can be described by the following vector:

Table 2. Ambiguous Attribute Vector

Example **x**	
x_1	YELLOW or PURPLE
x_2	LARGE
x_3	SQUARE

This is to indicate that the example's color (x_1) is known to be either "YELLOW" or "PURPLE", size (x_2) is "LARGE", and shape (x_3) is "SQUARE". Note that while some attribute values are provided concretely, others are only known to be constrained to a subset of values (e.g., we know the color is not "BLUE", but we do not know if it is "YELLOW" or "PURPLE").

A mechanism to quantify the *amount* of ambiguity in the data is necessary in order to evaluate a potential domain. Suppose that, in a given example, an attribute is given n_A out of n_T different values. For instance, as in the above example, if the attribute can acquire values $[BLUE, YELLOW, PURPLE]$ and is known to be either $YELLOW$ or $PURPLE$, for the given example, then $n_T = 3$ and $n_A = 2$. The amount of ambiguity is then quantified as follows:

$$AAmb = \frac{n_A - 1}{n_T - 1} \tag{17}$$

This ensures that $AAmb = 0$ if the value is known precisely $(n_A - 1 = 0)$ and $AAmb = 1$ if the value is totally unknown. Note that each attribute must be able to acquire at least two different values so that $n_T > 1$. The total amount of attribute-value ambiguity in the data can be assessed either by taking the average value of $AAmb$ over all example-attribute pairs or over only those example-attribute pairs where the attribute value is ambiguous. In the following formula, m is the number of example-attribute pairs considered.

$$AAmb_{total} = \frac{1}{m} \sum_{j}^{m} \frac{n_{A(j)} - 1}{n_{T(j)} - 1} \times 100\% \tag{18}$$

We assume the same ambiguity can also be observed in the class labels. For instance, out of classes $C_1 \ldots C_4$, the teacher knows that the example is not from C_3 and not from C_4, but does not know if it is C_1 or C_2.

An example of the real world domain for the Intrusion Detection System is shown here as an instance of a domain with both attribute and classification label ambiguity. In this example the ambiguity is represented by multi-labels separated with a ";".

Table 3. Three training examples from the IDS Domain. Some attribute values are ambiguous.

Case	*Protocol*	*Event*	*OS*	*AttackType*	CLASS
1	UDP	App. Alter	WinXP; Win2k3	Exploit	False Pos.; Correct
2	ICMP	App. Alter	Unix Linux	DoS	False Neg.
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	TCP	Web	Win2k3	Recon.	Correct

The fact that class labels are ambiguous makes it necessary to redefine the performance criterion accordingly. As mentioned before, let N be the number of testing examples, let $C_{i,known}$ be the set of class labels of the i-th testing example, and let $C_{i,classif}$ be the set of class labels given to the i-th testing example by the classifier. We evaluate the classification performance of this classifier by the following formula:

$$AmbA = \frac{1}{N}\sum_{i=1}^{N} Acc_i; \qquad \text{where} \tag{19}$$

$$Acc_i = \frac{|C_{i,known} \bigcap C_{i,classif}|}{|C_{i,known} \bigcup C_{i,classif}|} \tag{20}$$

We want to explore what modifications are needed if the k-Nearest Neighbor classification (k-NN) is to be used in such ”ambiguous” domains. Two aspects of classical k-NN classifiers need to be modified if the paradigm is to be applied to ambiguously described data. First of those is the way similarity between two examples is evaluated, the second relates to the voting mechanism. Let us address each of them in turn.

3.2 Handling Ambiguity in k-Nearest Neighbor classifiers

Using instance based classification, there are two main issues we are concerned with. First, how to calculate a similarity metric and second, how to determine the appropriate class label when dealing with ambiguous data sets. It has been the shortcoming of previous research not to address these two problems for ambiguous examples. In the following section the issues of similarity are addressed by the definition of a similarity function that acts to normalize for the attribute values. The issue of class label voting and selection is addressed with the formulation of a Class Label Weighting approach. For completeness, we compare the results against a fuzzy approach to class label voting.

3.2.1 Vector-to-Vector Similarity

In domains with examples described by attribute vectors, mutual similarity of the two vectors is usually established by means of their geometric distance. The shorter the distance, the greater the similarity.

The simplest case of ambiguity is when the value is totally unknown, values are then replaced with a "question mark". If this is the case then a small modification to the traditional distance metric will need to be made. Presented are three such modifications.

Traditional Nearest Neighbor Approaches :

k-NNa: If $x_i = y_i$, then $s(x_i, y_i) = 0$; if $x_i \neq y_i$, then $s(x_i, y_i) = 1$; and if a question mark is encountered in either x_i or y_i, then $s(x_i, y_i) = 1$. These values are then summed over all M attributes:

$$S(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{M} s(x_i, y_i); \quad \text{where} \tag{21}$$

$$s(x_i, y_i) = \begin{cases} 1 & y_i \neq x_i \\ 0 & y_i = x_i \\ 1 & y_i = ?\ or\ x_i = ? \end{cases} \tag{22}$$

k-NNb: If the ambiguities are rare, we can afford to ignore the unknown attribute values altogether. For all the remaining attributes, $x_i = y_i$ implies $s(x_i, y_i) = 0$ and $x_i \neq y_i$ implies $s(x_i, y_i) = 1$. Denoting by $M1$ the number of unambiguous attributes, we use the following formula:

$$S(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{M1} s(x_i, y_i); \quad \text{where} \tag{23}$$

$$s(x_i, y_i) = \begin{cases} 1 & y_i \neq x_i \\ 0 & y_i = x_i \end{cases} \tag{24}$$

k-NNc: Yet another possibility is to replace each question mark with the most frequent value of the given attribute. The similarity-calculating formula then remains unchanged:

$$S(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{M} s(x_i, y_i); \quad \text{where} \tag{25}$$

$$s(x_i, y_i) = \begin{cases} 1 & y_i \neq x_i \\ 0 & y_i = x_i \end{cases} \tag{26}$$

Reflecting partially known attribute values :

Let us now proceed to a more complicated case, where a given attribute value is known only partially by which we mean the attribute is known to have one out of a subset of the legal values for this attribute. Note that this is different from total ignorance when the attribute value is replaced by a "question mark". Given the following example of a domain representing whether a baseball game should be played, the attributes map to the season (e.g Summer, Spring, Fall, Winter), air quality (e.g. Dry, Humid), and part of the day (e.g. Morning, Noon, Afternoon, and Night) with the class being either "Play", "Delay", or "No-Play":

Table 4. Table shows an example of a testing and training attribute vectors

Testing Example $\mathbf{x}$	
x_1	SUMMER or SPRING
x_2	HUMID
x_3	DAY

Training Example $\mathbf{y}$	
y_1	SPRING
y_2	DRY
y_3	DAY
CLASS LABEL	Play

In order to account for the x_1 attribute effectively, the distance metric needs to be modified in the following way:

Let $\mathbf{x}$ and $\mathbf{y}$ be attribute vectors for the testing and training set respectively.

$$S(\mathbf{x},\mathbf{y})_{kAmbigNN} = \sum_{i=1}^{n} s(x_i, y_i); \quad \text{where} \quad s(x_i, y_i) = \frac{|x_i \bigcap y_i|}{|x_i \bigcup y_i|} \tag{27}$$

Calculations of similarity would follow as:

$$S(\mathbf{x},\mathbf{y})_{kAmbigNN} = \frac{|SPRING|}{|SPRING, SUMMER|} + \frac{|\oslash|}{|DRY, HUMID|} + \frac{|DAY|}{|DAY|} \tag{28}$$

$$= \frac{1}{2} + \frac{0}{2} + \frac{1}{1} \Rightarrow 1.5 \tag{29}$$

Similarity calculations give a weight to the attributes being compared, thus defining a quantifiable difference between two attributes not based on ordinal properties. This similarity value for each attribute is represented by the position on the continuum of unrelated to identical values. The values for the similarity of a single attribute fall in the set of real numbers for $[0 \ldots 1]$. Total instance similarity is the summation across all attributes in the instance. By using this function, the data set is normalized and discrete differences in nominal data can be found in a robust way. This method of using the similarity function, outlined in Equation 27 for nearest neighbor classification, we call the k Ambiguous Nearest Neighbor Classifier (k-AmbigNN) approach.

Reflecting partially known attributes with continuous values :

The algorithm presented here can be extended to account for continuous valued attribute labels and ranges. Two changes need to be made in order to accommodate this: one to the similarity function and one to the way ambiguity is defined for the attribute. When working with discrete values, ambiguity is quite clear. As demonstrated, ambiguity is simply the multiplicity of discrete attribute values. Defining ambiguity on a continuous domain takes a bit more effort. The following graphs show the different types of ambiguities that can arise. As can be seen, ambiguity is represented as a range or a set of ranges. A single continuous value can be accounted for in the normal way.

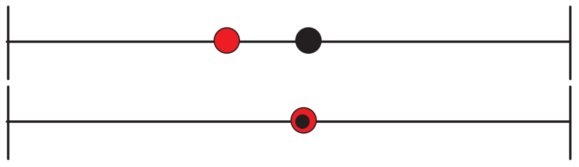

Figure 4. Two Data Points

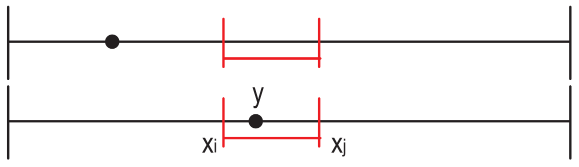

Figure 5. One Data Point One Data Range

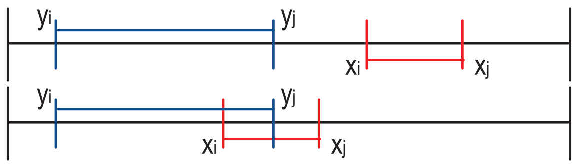

Figure 6. Two Data Ranges

Moving on to the second issue of the similarity function, we find that when accounting for continuous values the most efficient mechanism is to use a variation on the Euclidean distance measure. In Equation 27 we make a change as follows:

$$S(\mathbf{x},\mathbf{y})_{kAmbigNN,continuous} = \sum_{i=1}^{n} s(x_i, y_i); \quad \text{where} \quad s(x_i, y_i) = |x_i - y_i| \tag{30}$$

3.2.2 Voting Mechanism

In the example from Table 5, the 3-NN classifier has already identified the three nearest neighbors, some of which have more than one class label (because the true class is known only partially). Labeling the testing example with a union of these labels would not reflect the different degree of "focus" present in the individual neighbors. Thus the fact that the first neighbor suggests classes $[C_1, C_2]$ and the third neighbor suggests only C_2 seems to indicate there should be more confidence in the latter example. Illustrated here are two approaches used for class label voting that accounts for ambiguity in the class label through weighting methods.

Table 5. Ambiguous class labels from a k-NN classifier

Class Labels	
Nearest Neighbor 1	C_1 or C_2
Nearest Neighbor 2	C_1 or C_3 or C_4
Nearest Neighbor 3	C_2

The first approach assigns a weight to each class label by letting $N_{i,c} = 1$ if class c is found among the labels of the i-th nearest neighbor and let $N_{i,c} = 0$ if it is not. If $N_{i,total}$ is the number of labels in the i-th nearest neighbor, then, for k nearest neighbors, we calculate the weight of class c as follows:

$$W_{factor}(c) = \frac{1}{k} \sum_{i=1}^{k} \frac{N_{i,c}}{N_{i,total}} \tag{31}$$

Voting is done by selecting the class label with the highest W_{factor}. Example W_{factor} values for this voting mechanism using the data provided in Table 5 are presented in Table 6. For the case of the three nearest neighbors from Table 5, the class weights are calculated in Table 6. We define a threshold, θ, to aid in the selection process. The algorithm will choose classes whose weights exceed a user's threshold, θ. For example, if $\theta = 0.4$, class C_2 is chosen; if $\theta = 0.2$, classes $[C_1, C_2]$ are chosen.

Table 6. Voting for the class label based on the nearest neighbors listed

W$_{factor}$(C)		
$C_1 =$	$\frac{1}{3}\left(\frac{1}{2}+\frac{1}{3}+\frac{0}{1}\right)$	$\Rightarrow 0.278$
$C_2 =$	$\frac{1}{3}\left(\frac{1}{2}+\frac{0}{3}+\frac{1}{1}\right)$	$\Rightarrow 0.500$
$C_3 =$	$\frac{1}{3}\left(\frac{0}{2}+\frac{1}{3}+\frac{0}{1}\right)$	$\Rightarrow 0.111$
$C_4 =$	$\frac{1}{3}\left(\frac{0}{2}+\frac{1}{3}+\frac{0}{1}\right)$	$\Rightarrow 0.111$

The second approach is based on a fuzzy voting scheme and was implemented for sake of comparison with earlier work done in uncertainty processing. The main idea is to consider the degree of membership a training case has in a classification label. This process is defined in the following manner:

- n_c ... number of classes,
- k ... number of nearest neighbors,
- $1 \leq j \leq k$ ... j-th neighbor,
- $\mu_i(\mathbf{y}_j)$ ... membership of the neighbor, $\mathbf{y}_j$, in the i-th class,
- $\mu_i(\mathbf{x})$ ... membership of the training example, $\mathbf{x}$, in the i-th class,
- $\|\mathbf{x} - \mathbf{y}_j\|$ ... the distance between $\mathbf{x}$ and $\mathbf{y}_j$

- $1.0 \leq w \leq 2.0 \ldots$ a fuzzy parameter

The following formula, based on ideas developed by [11], represents the normalized membership for each of the class labels.

$$\mu_i(\mathbf{x}) = \frac{\sum_{j=1}^{k} \mu_i(\mathbf{y}_j) \cdot (\|\mathbf{x} - \mathbf{y}_j\|^{-\frac{2}{w-1}})}{\sum_{k=1}^{k} (\|\mathbf{x} - \mathbf{y}_j\|^{-\frac{2}{w-1}})}, \quad i = 1, 2, \ldots, n_c \tag{32}$$

Note that, we are using distance $\|\mathbf{x} - \mathbf{y}_j\|$ instead of similarity. Furthermore, note that the formula involves two parameters: w and θ_{fuzzy}, the latter being a threshold that controls the class selection.

CHAPTER 4

Experimental Procedure

The experiments that were run show the functionality of the algorithm presented. Ambiguous data sets were compiled and the algorithm tested against them in an effort to prove that our instance based classifier outperforms normal instance based classifiers on ambiguous domains. In order to run the experiments ambiguous data sets had to be defined. In the Description of Data Set section the breakdown of the data sets is discussed. The procedure for generating the ambiguous data is described in the preceding sections as well. The previously described algorithm will be referred to a k-Ambiguous Instance Based Classification (k-AmbigNN).

4.1 Description of Data Sets/Files

The data that is provided to the algorithm comes in two types. Meta-data that describes the training and testing set and the actual data set itself. The meta-data defines the name of the attributes and class labels as well as the type of data. It is important to know the data type for the attribute because the algorithm changes depending on the attribute or class label type.

The application takes three files as input for purposes of classification.

filestem.names

The names file contains meta-data describing the attributes and class labels of the instances in the data file. The names file follows the following format:

```
Class Label:Possible class labels
(class label types are defined the same way as attribute labels)
Attribute Name1:Attribute Type
Attribute Name2:Attribute Type
.
Attribute Namen:Attribute Type
```

Acceptable attribute types are:

1. Numerical : *Numerical data types are numbers that are defined by the set of Real numbers. Numerical data can have an order of magnitude and precision within limits of the system processing them.*

2. String : *A string data type can be used to define data that is not numerical in nature. "Good, Bad, Fair" are examples of string data.*

3. Range : *A range type normally represents a range of data that are discrete and ordinal such as the set of all integers or real numbers. An example of a range data type is 1-10 or a-b where the character "-" is a special character denoting the range.*

Names file format example:

```
PlayGame:true,false
Current_Weather:good,bad,fair
Humidity:numeric
Forecast:good,bad,fair
Duepoint:range
```

filestem.data and *filestem*.test

These two file types are defined in the same way. They are comma separated data files where each line is a new record or instance. The system reads each line and breaks them along the comma line to match up with the attribute types defined by the names file.

Data file format example:

```
good,28,fair,25-45,true
fair,45,fair,55-57,true
good,88,bad,25-45,true
fair,45,fair,62-87,false
...
```

The Test file format differs by not having the last column defining the class label for the input vector.

```
good,28,fair,25-45
fair,45,fair,55-57
good,88,bad,25-45
fair,45,fair,62-87
...
```

In the event that the instance vector is ambiguously defined, the multiple attribute labels or classification labels would be defined with a ";" as the separator for the label.

```
good;fair,28,fair,25-45
fair,45,fair,55-57;25-56
good,88,bad,25-45
fair,45,fair;good;bad,62-87
fair,,good,
...
```

Unknown labels are defined in one of two ways either by a hole in the data set defined by ",?," or the set of every value for the described attribute: "good;bad;fair".

4.2 Generation of Test Data

As previously mentioned, the ambiguously defined attribute label problem is supported by the real world domain dealing with Computer Network Intrusion Detection Systems and the evaluation of such systems. There are a total of three types of data domains employed in the following sets of experiments. The first data domain is the IDS domain. All three domains contain completely discrete data. The algorithm does provide for continuous data, as mentioned before, discrete domains were chosen for calculation simplicity and because the main IDS domains of interest are discrete. The second type of domains used are generated from the UCI Repository. In this category there are two sub categories:1. raw UCI domain (University) contributed by Lebowitz and 2. other UCI domains that were artificially made ambiguous. These UCI domains can contain missing values. The third general type of domain are completely synthetic and were generated for benchmark testing the boundaries of the algorithm and were also artificially made ambiguous. The attribute and label ambiguity went beyond simply increasing the number of missing values or by randomly adding attribute values to create attribute value sets. In an effort to maintain accuracy to real world situations, we introduced ambiguity to the attribute values by adding related values to make an attribute set. We found the most likely attribute values to fit each vectors situation with the use of a decision tree algorithm.

4.2.1 Network Intrusion Detection System Data

The data file was compiled from observations made on a concrete computer network as a result of a number of measurements. The collected data included the protocol of the incoming network request, the event that took place due to the request, analyzed attack type, type of operating system targeted in the attack, system used for intrusion detection, and possible classification labels: False Positive, False Negative, and Correctly classified.

Table 7. Illustration of IDS Data Definition

Protocol	*Event*	*AttackType*	*OS*	*System*	Class
UDP	Web	Recon.	Linux	NIDS	False Pos.
TCP	App. Alter	Exploit	Unix	SIV	False Neg.
ICMP	Launch App.	DoS	WinXP	LM	Correct
	Install		Win 98/Me		
	Term. Key Proc.		Win2k		
	Auto Launch		Win2k3		

Table 7 summarizes attributes that acquire ambiguous values. The ambiguity is due to the network administrator's limited ability to interpret the corresponding log files. Other attributes were unambiguous—in reality, there were 10 attributes and 126 examples. The amount of ambiguity according to Equation 18 is 30%.

4.2.2 Synthetic Data Set

The labels of the attributes are discrete. We choose this type of data set because it is very easy to create and minimizes the complexity of the calculations for both our approach and the traditional approaches compared against. We will work with the following synthetic domains.

Boolean Expression for Classification: These domains are constructed by first creating a large data set of binary attributes. In order to make a suitably large domain, we started with 10 binary attributes producing 1024 cases to use. Examples that satisfy the following equations are considered positive, while those that do not are negative. In the following we denote the logical operator "AND" with the $\wedge$, the logical operator "OR" with the $\vee$, and the logical operator "NEGATIVE" with the $\neg$.

$$Bool(\mathbf{x}) = (x_1 \vee x_6) \wedge (x_4 \vee x_{10}) \wedge (\neg x_2 \vee x_7) \tag{33}$$

$$Bool(\mathbf{x}) = (x_{10} \wedge x_1) \vee (x_4 \wedge x_6 \wedge x_8) \tag{34}$$

Attribute ambiguity is randomly introduced across the entire domain by randomly selecting (with replacement) attribute values and appending them with the most common value(s) for the domain. The amount of ambiguity is controlled by the Amount of Ambiguity (AAmb) for the data set as defined by Equation 17. Subdomains with AAmb values ranging from 10% to 50% are created for purposes of experimentation.

Decision Tree for Classification: The second domain was created from a decision tree. The decision tree presented in Figure 10 is used to create the domain. 10 multi-valued attributes are used for this domain. The decision tree is then pruned to remove a node. The pruned tree, presented in Figure 8, is used to re-classifying the data. Attribute ambiguity is introduced into this domain by randomly selecting an attribute, with replacement, and adding the most frequently occurring value to it. The amount of ambiguity in a domain is defined by the formulation of the AAmb term from Equation 17. Sub-domains with AAmb values ranging from 10% to 50% for both ambiguous and crisp class labels are created with for purpose of experimentation. We created two of these domains for experimentation shown in Figure 8 and Figure 10.

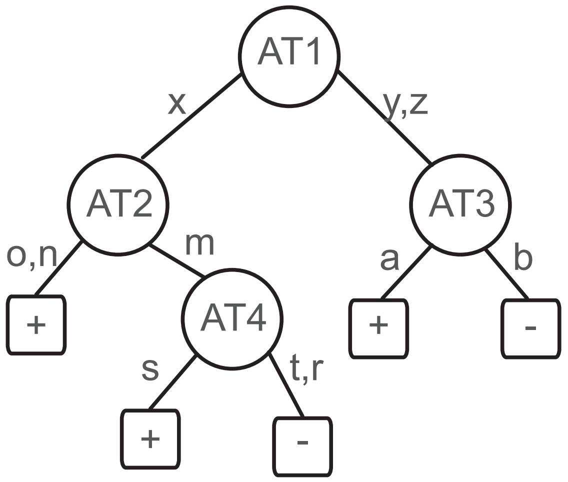

Figure 7. This decision tree was used to classify a synthetic domain.

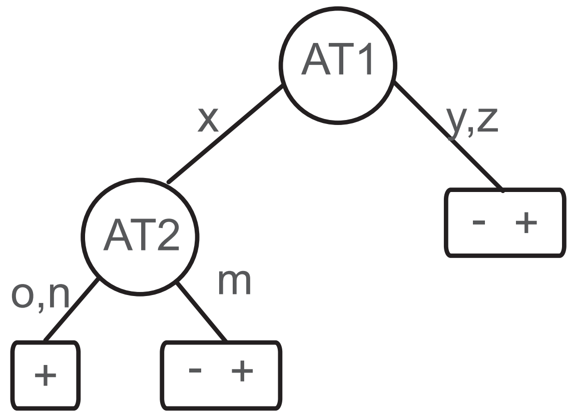

Figure 8. Instead of pruning, the leaf nodes were combined to generate an ambiguous class label. This synthetic tree was then used to re-classify its corresponding domain.

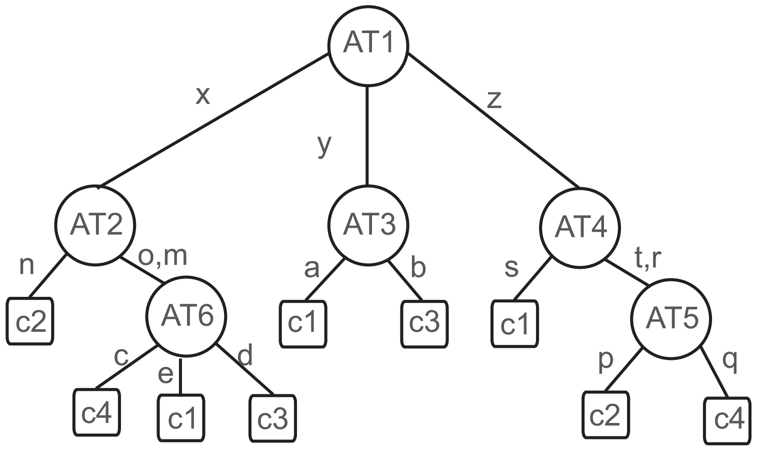

Figure 9. This decision tree was used to classify a more complex synthetic domain.

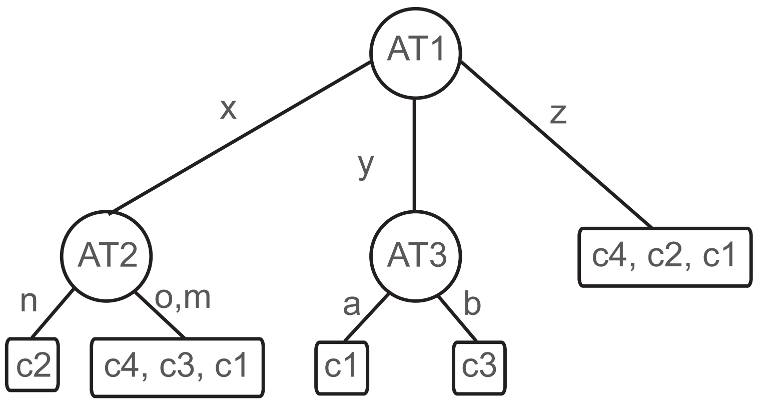

Figure 10. Instead of pruning, the leaf nodes were combined to generate an ambiguous class label. This synthetic tree was then used to re-classify its corresponding domain.

Adapted UCI Repository

Using the Balloon and Voting-Records data set from the UCI Machine Learning Repository, we are creating a third type of data set by introduced ambiguity. The ambiguity is introduced in two ways: first, class label ambiguity and second, attribute ambiguity. Class label ambiguity is achieved by determining decision trees that best fit the Balloon and the Voting data sets. A pruning process involving the change of a single path down the tree yielding multiple class labels is invoked. When the data set is exposed to the new decision tree for re-classification the potential of different or multiple class labels is high. This procedure can be executed repeatedly with differently pruned tree to build larger ambiguity into the class labels.

In order to introduce ambiguity into attribute values, an attribute was selected at random with replacement. For that attribute the most common value was deduced from the entire set and added to the existing value. Random selection with replacement was used so that a larger amount of ambiguity per attribute could be achieved. The overall ambiguity metric for the sample is controlled by the AAmb defined in Equation 17. Sub-domains with AAmb values ranging from 10% to 50% are created for experimentation.

4.2.3 Experimental Procedure

Whenever statistical significance tests are needed we used the 5x2 cross-validation with two-tailed t-test as recommended by Dietterich [4].

Performance Evaluation As mentioned above performance was evaluated by a metric named AmbA. Experiments using the Precision and Recall calculations were also performed using the Micro-, Macro-, and F1 approaches. In order to understand these calculations it is important first to understand what they are telling us. In general Precision is the measure of specificity. Another way to say this is Precision is the measure of relevance a result has to a domain. While in contrast Recall is simply the measure of sensitivity a result has to the overall domain. Finally the F1 score is the measure of accuracy. F1 can be viewed as a weighted average of the Precision and Recall calculations.

The Micro- and Macro- calculations give us two perspectives of the accuracy measures. Micro- is simply defined as calculating the statistical value for each class label as it is produced and then taking the average, while Macro- implies that the average is taken across the statistics generated for all decision.

Presented is a quick description of these calculations:
Let us define TP as "true positive", FP as "false positive", TN as "true negatives", and FN as "false negatives"

$$Precision : Pr = \frac{TP}{TP + FP} \tag{35}$$

$$Recall : Re = \frac{TP}{TP + FN} \tag{36}$$

$$F_1 = \frac{2xPrxRe}{Pr + Re} \tag{37}$$

Table 8. Micro and Macro Definitions

	$Precision$	$Recall$	F_1
Macro	$Pr^M = \frac{\sum_i^k Pr_i}{k}$	$Re^M = \frac{\sum_i^k Re_i}{k}$	$F_1^M = \frac{2xPr^M xRe^M}{Pr^M + Re^M}$
Micro	$Pr^\mu = \frac{\sum_{i=1}^k TP_i}{\sum_{i=1}^k TP_i + FP_i}$	$Re^\mu = \frac{\sum_{i=1}^k TP_i}{\sum_{i=1}^k TP_i + FN_i}$	$F_1^\mu = \frac{2xPr^\mu xRe^\mu}{Pr^\mu + Re^\mu}$

5x2 Cross-Validation Method The main tenant of the 5x2cv method is to perform a 2-fold cross validation 5 times. For each replication the domain is split randomly into equal sized pieces. The two learning algorithms are trained on both sets producing four error estimates or accuracy calculations. Using this data a two-tailed t-test was performed to determine significant performance differences in the two algorithms. For our algorithms our k-AmbigNN algorithm was systematically tested against each of the traditional approaches producing the data plots in the following results section.

Two major experiments were conducted as outlined: 1. evaluation on class label weighting factors for determining classification from ambiguously defined class labels, and 2. systematic evaluation of k-AmbigNN algorithm on synthetic and real world domains.

For the synthetic domains the k nearest neighbors are computed from the k-AmbigNN algorithm where $k = 1$ and $k = 5$. The selected k nearest neighbors are then subjected to the voting mechanisms (W_{factor} and k-fuzzyNN approach) for classification. These two tests are run on each synthetic sub-domain with increasing AAmb values producing data to be used in Accuracy plots (defined by Equation 19 vs. AAmb plot).

For the two adapted UCI domains the k nearest neighbors are computed for the k-AmbigNN and the k-NNa approach for values of $k = 1$ and $k = 5$. The selected k nearest neighbors are subjected to both voting mechanisms for classification. These two tests are run on each adapted sub-domain in order to produce accuracy data to be plotted against the AAmb value. k-NNa is the only traditional approach selected because these domains do not contain missing values.

4.2.4 Results

First, we wanted to learn how the performance is affected by the cut-off point, θ, used in the voting mechanism—recall that the testing example is assigned all class labels for which $W_{factor}(c) > \theta$. High values of θ lead to the "crisp" single-label case, or even to the situation where no class label exceeds θ. Conversely, we determined that very low values will render the labeling more ambiguous than necessary. The correct selection of θ seems to depend on the amount of class label ambiguity present. We measured the classification performance by the *AmbA* metric defined by Equation 19. For the synthetic data, Table 9 shows the values of *AAmb* for different values of θ. The results indicate that the optimum cut-off is somewhere between 0.3 and 0.5, and that its optimum value depends also on the number, k, of the nearest neighbors employed.

Table 9. AmbA-performance for different cut-off points and k-values as evaluated on synthetic domain with *AAmb = 30%*.

cut-off	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
k = 1	0.385	0.385	0.385	0.385	0.385	0	0	0
k = 3	0.285	0.285	0.366	0.366	0.366	0.112	0.112	0
k = 5	0.149	0.459	0.459	0.416	0.416	0.248	0.019	0.019
k = 9	0.037	0.267	0.466	0.503	0.398	0.286	0.019	0.019

For the balance of experiments a value of 0.3 was chosen for *theta*. This value seemed to perform the best across the broad stroke.

Figure 11 compares the performance of k-AmbigNN with that of k-NN and k-fuzzyNN in the intrusion detection domain for different values of k. To be able to use k-NN, we replaced ambiguous values with question marks. The individual points in the charts have been obtained as averages from the 5x2 cross-validation, a methodology recommended for the assessment of machine learning algorithms by Dietterich [4]. It is apparent that the k-AmbigNN consistently out-performs the other approaches over a whole range of k-values. Since k-NN systematically under-performs k-AmbigNN, we conclude that replacing partial ambiguity with total ambiguity is indeed unnecessary and harmful to the outcome. Interestingly, whereas k-NN's performance dropped with the growing value of k due to the separation in the data points, k-AmbigNN's performance grew with the increasing k. The margin between k-NN and k-AmbigNN is statistically significant according to the two-tailed t-test with a confidence level of 2%. The outcome of the fuzzy approach is a different story. The fuzzy approach does not seem to be appropriate here. It is surmised that our ambiguities are too simple for the fuzzy-set approach to fully unfold its strength.

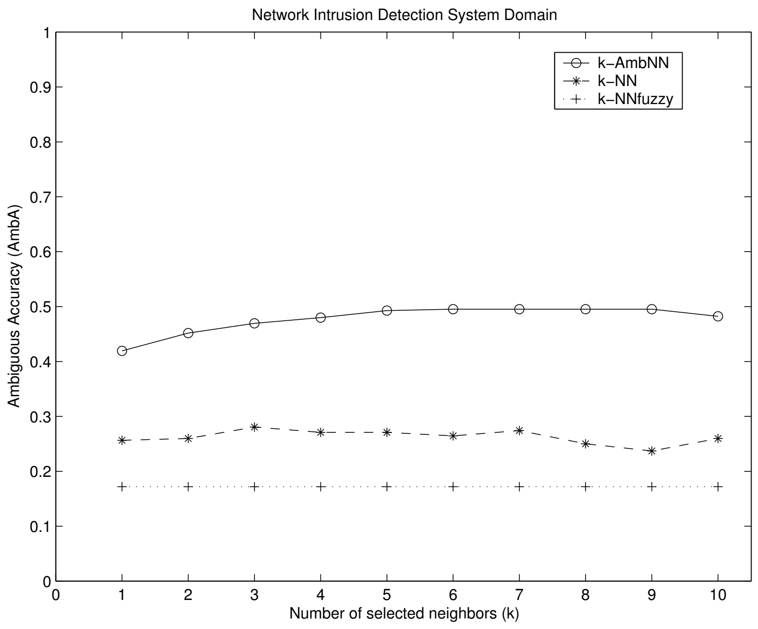

Figure 11. Comparing the performance of k-AmbigNN with that of k-NN on the IDS domain.

The figures following Figure 11 show the IDS domain evaluated using the k-AmbigNN and k-NN approached and Macro- and Micro Precision, Recall, and F_1 as the accuracy measures. Figure 14 and Figure 17 indicate the trend noticed before that the k-AmbigNN algorithm increases its performance as the k values grow. This assertion is supported experimentally in Figures 24, Figures 21, Figures 32, and Figures 29.

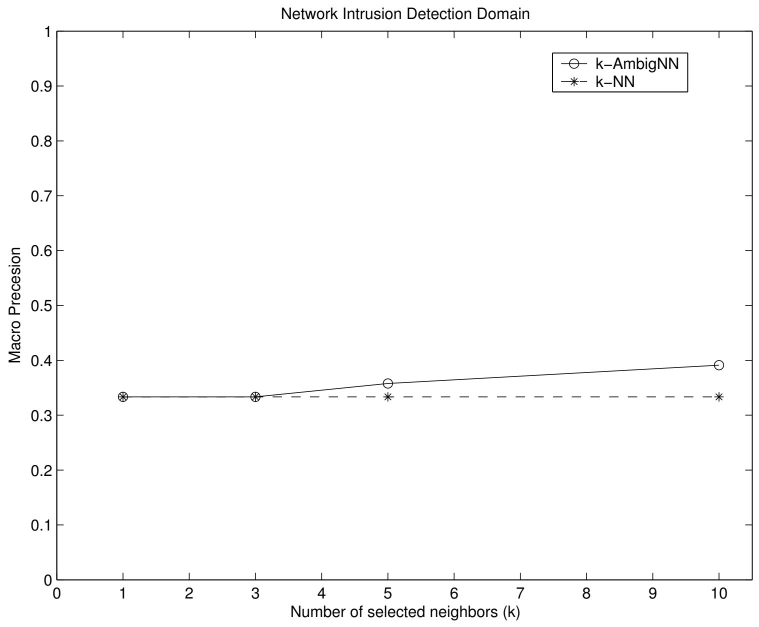

Figure 12. Macro Precision vs. number of neighbors for k-AmbigNN and k-NN.

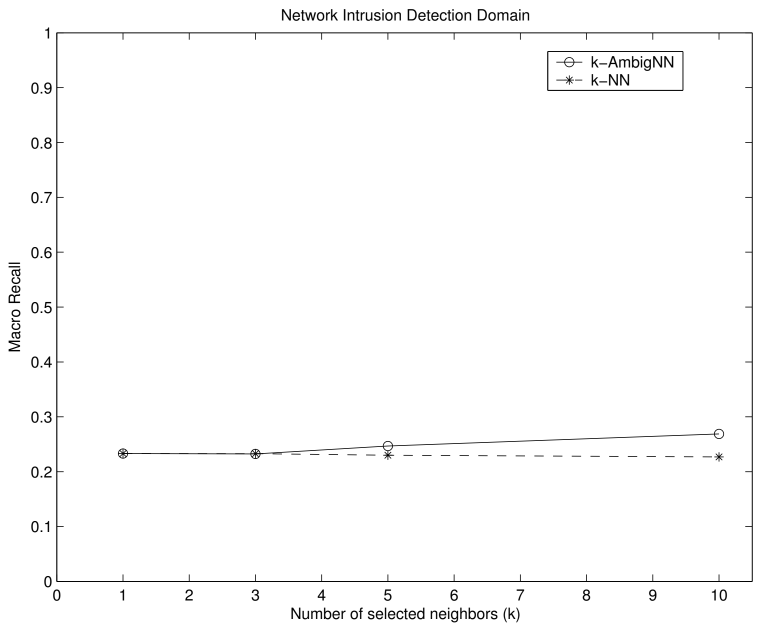

Figure 13. Macro Recall vs. number of neighbors for k-AmbigNN and k-NN.

Experimental results show in Figure 14 shows the Macro F_1 plotted versus the number of neighbors fro both the k-AmbigNN and the k-NN. This plot shows a trend toward greater accuracy as the number of neighbors are increased.

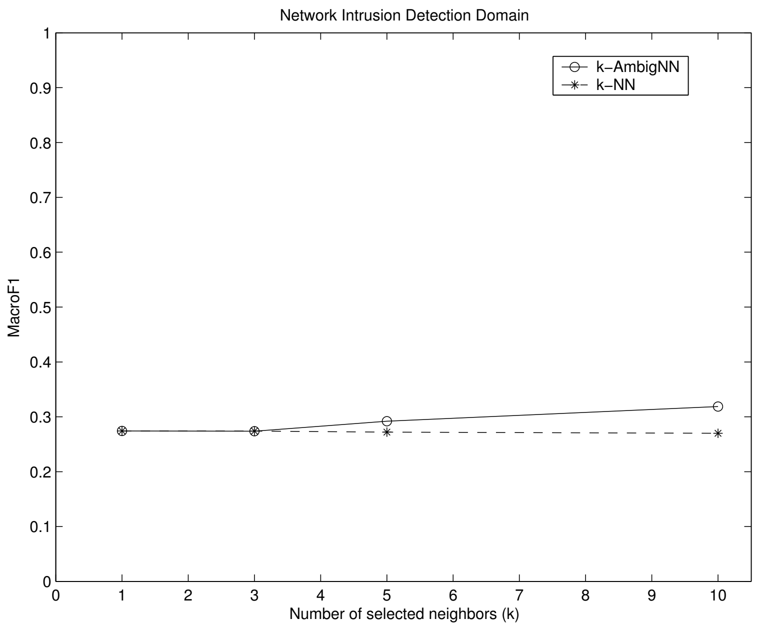

Figure 14. Macro F_1 vs. number of neighbors for k-AmbigNN and k-NN.

Figures 15 - 17 support the above findings with the micro precision, recall, and F_1 calculations. The findings on the IDS domain suggests that accounting for ambiguity in the manner presented here, the k-AmbigNN approach, has an edge over the traditional approaches.

Figure 18 corroborates all these observations by experiments with the `university` domain. In this sparser, but less noisy domain, even k-AmbigNN's performance tended to drop with growing k, although even here we seem to have a reason (at least for small k) to claim that k-AmbigNN is more robust regarding the increasing values of k. We cautiously suggest that the experiments with the first two domains indicate that k-AmbigNN offers not only higher performance, but also higher robustness with respect to k. The fuzzy voting mechanism seems to fail with this domain as well.

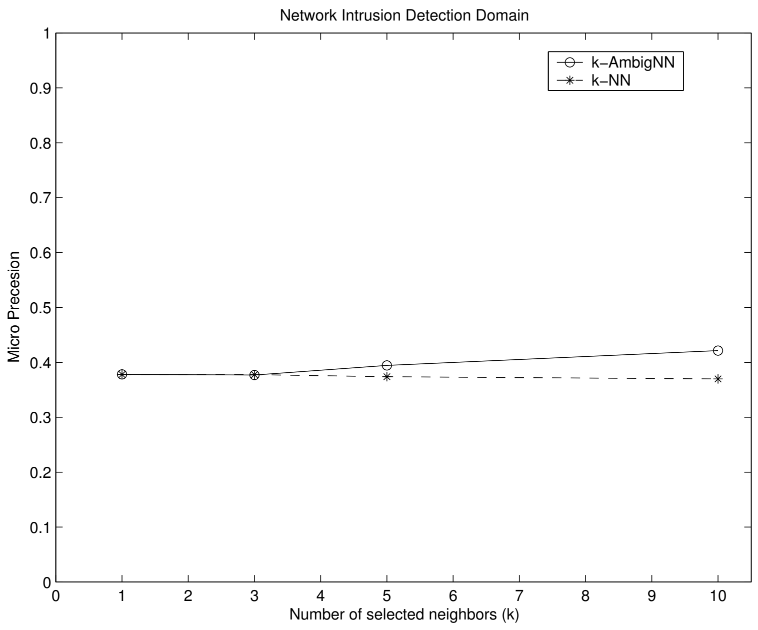

Figure 15. Micro Precision vs. number of neighbors for k-AmbigNN and k-NN.

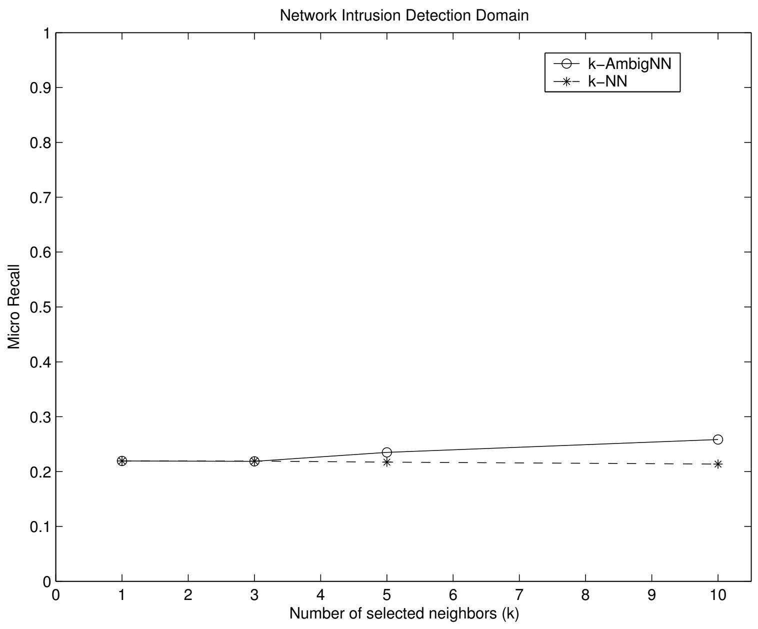

Figure 16. Micro Recall vs. number of neighbors for k-AmbigNN and k-NN.

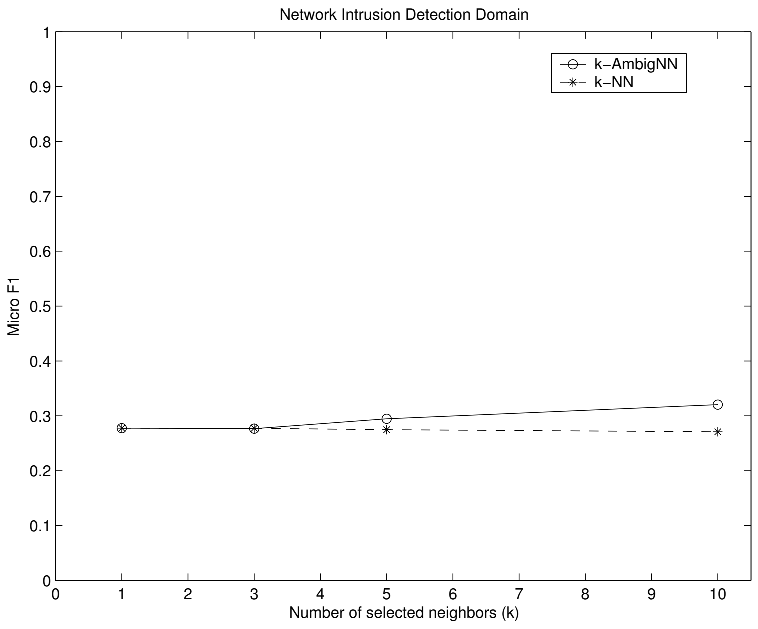

Figure 17. Micro F_1 vs. number of neighbors for k-AmbigNN and k-NN.

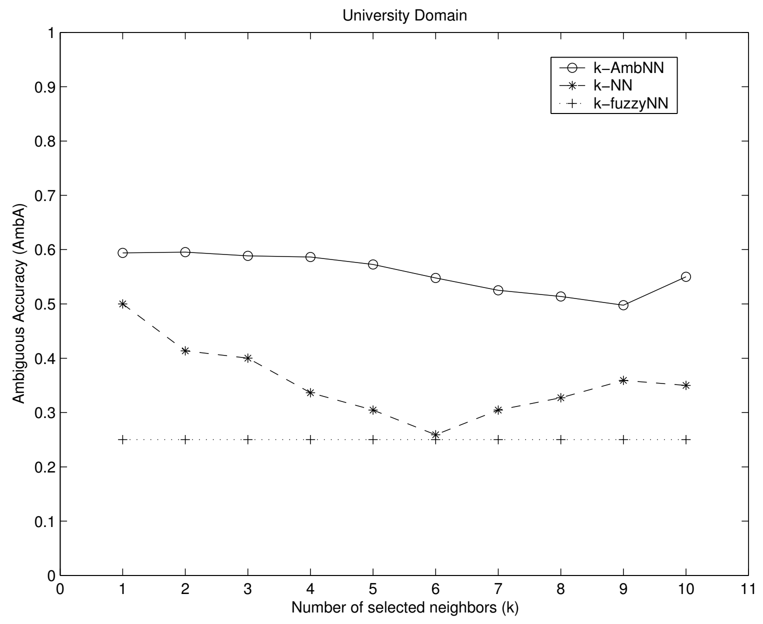

Figure 18. Comparing the performance of k-AmbigNN with that of k-NN on the `university` Domain.

Figures 19 - 21 show the results of the Macro tests performed on the UCI Universities domain. This domain is highly ambiguous across multiple variables. A clear trend is visible in the macro approaches. Precision seems to perform better. Figure 20 shows the Macro Recall for the UCI Universities domain where the first data points seem out of place. We believe this is a fluke of the data set being investigated. The important take away is the smooth trending of the k-AmbigNN approach seems to resist the changes in k better.

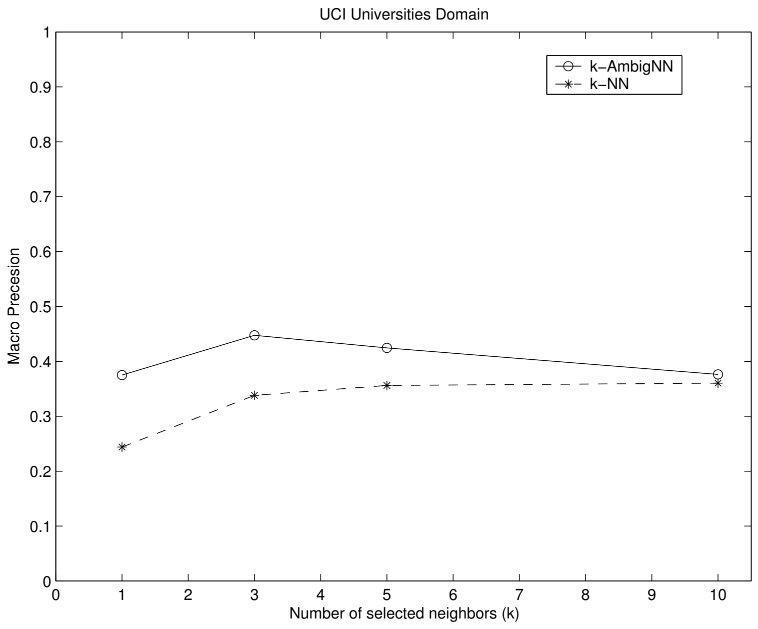

Figure 19. Macro Precision vs. number of neighbors for *k*-AmbigNN and *k*-NN.

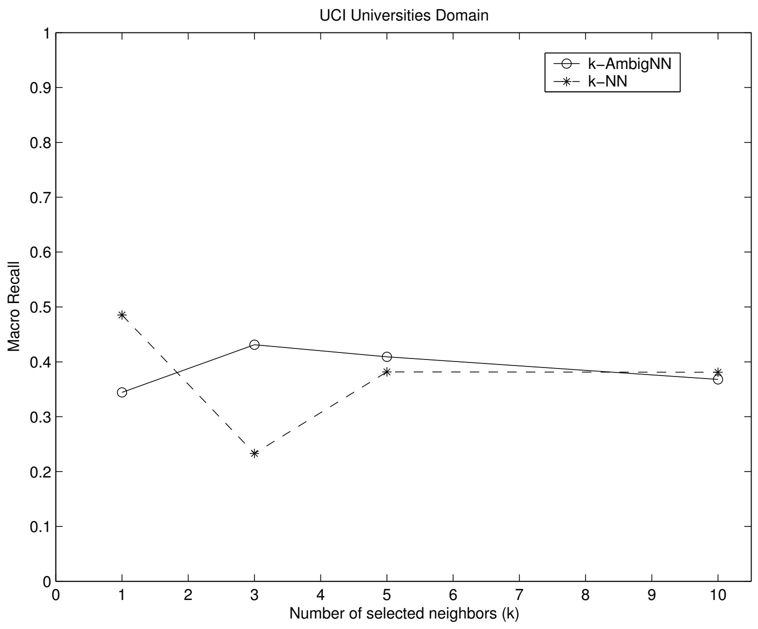

Figure 20. Macro Recall vs. number of neighbors for *k*-AmbigNN and *k*-NN.

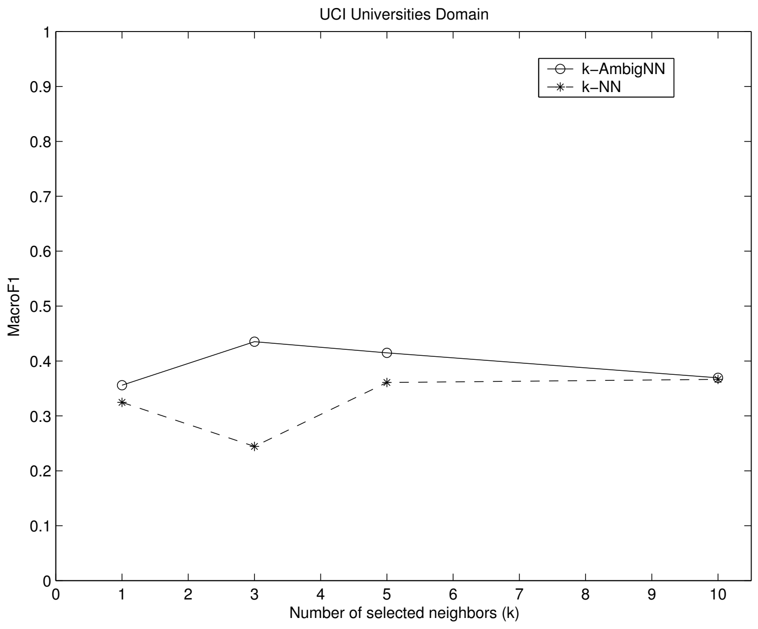

Figure 21. Macro F_1 vs. number of neighbors for k-AmbigNN and k-NN.

As was discussed earlier the same patterns follow in Figure 22 - Figure 24 for the micro experiments as they pertain to the UCI Universities domain. There is an expected minor trend down as the number of k grows. Again the resistance to k change is the main take away.

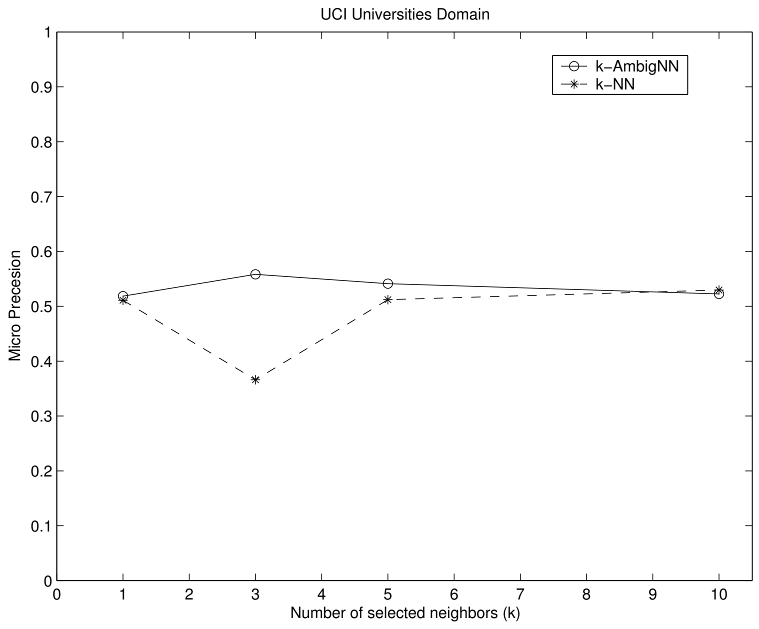

Figure 22. Micro Precision vs. number of neighbors for k-AmbigNN and k-NN.

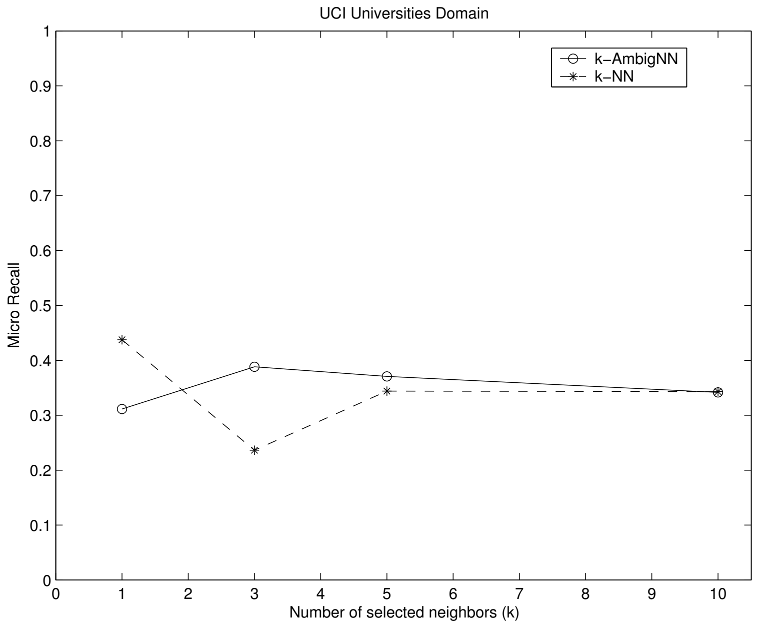

Figure 23. Micro Recall vs. number of neighbors for k-AmbigNN and k-NN.

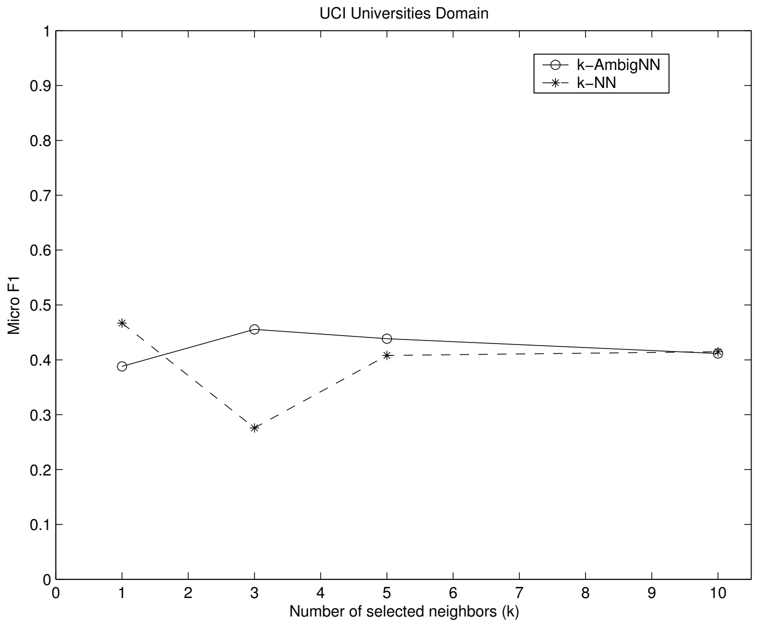

Figure 24. Micro F_1 vs. number of neighbors for k-AmbigNN and k-NN.

After the initial runs on "clean" domains with only inherent ambiguity, a modification was made. In the next group of experiments Figure 33 - Figure 42, an optimum value of k was chosen and programmatically ambiguity was introduced to the domains. This is achieved by an "induction" algorithm that estimates the classification accuracy (applying the 5-fold cross-validation approach to the training set) for different values of k and then selects the one that promises the highest performance. The same approach could in principle have been employed also for the choice of the θ-threshold used in the voting scheme but we used the fixed $\theta = 0.3$ not to complicate the matter.

Figure 25 and Figure 26 show the classification performance depended on the amount of ambiguities in the data for both decision tree generated domains. While k-fuzzyNN and k-NN appear relatively unperturbed by this parameter, k-AmbigNN's performance drops conspicuously with growing values of $AAmb$. The observation leads us to assume that k-AmbigNN is more appropriate where the domain ambiguity is limited.

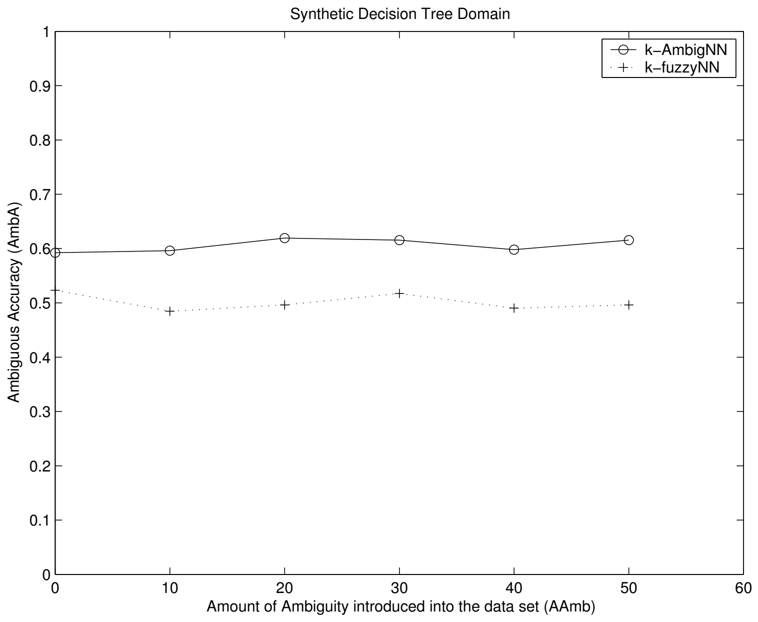

Figure 25. Comparison of the performance of k-AmbigNN, k-NN, k-fuzzyNN on the decision-tree-generated synthetic domains, as measured for different degrees of ambiguity.

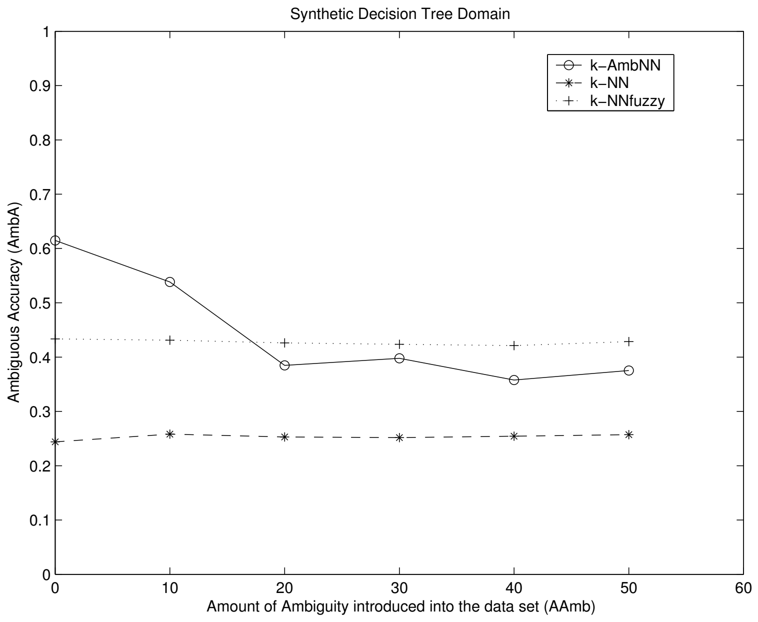

Figure 26. Comparison of the performance of k-AmbigNN, k-NN, k-fuzzyNN on the second decision-tree-generated synthetic domains, as measured for different degrees of ambiguity.

Macro and Micro Precision, Recall, and F1 were performed on the decision tree domain. We chose to experiment with the second decision tree domain because we felt it represented more realistic domains. The results can be found in Figures 27 - 32. As can be seen in the figures the *k*-AmbigNN approach generally performs as well as the traditional approach on the un-ambiguated domain. The general trend that the *k*-AmbigNN approach performs better with a higher number neighbors participating in the voting is supported. *k*-fuzzyNN was left off these figures due to the fact of very low results.

In Figures 27 - 29 the rather flat trend indicated that the number of neighbors mat not be the major contributor for the performance of the *k*-AmbigNN approach. In these experiments we kept the amount of ambiguity constant and varied the number of neighbors.

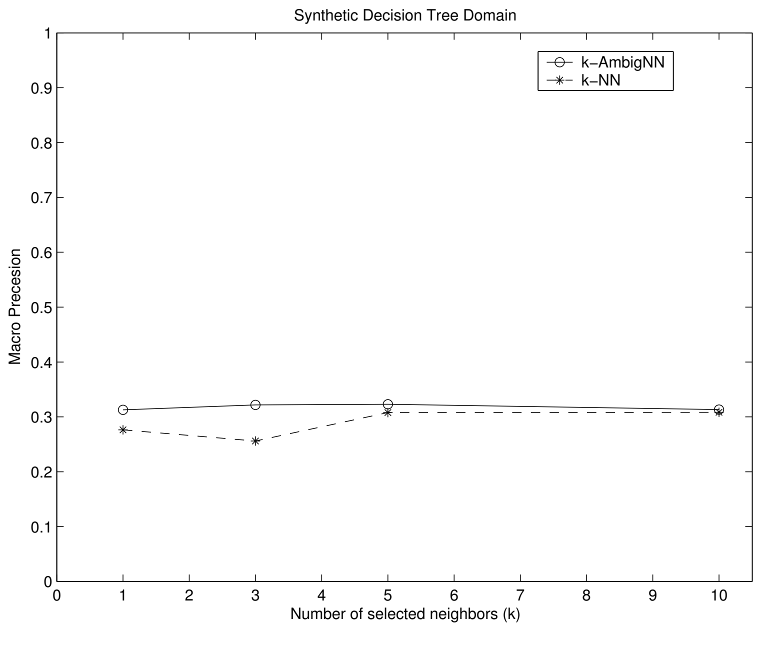

Figure 27. Macro Precision vs. number of neighbors for *k*-AmbigNN and *k*-NN calculated on the second decision tree domain.

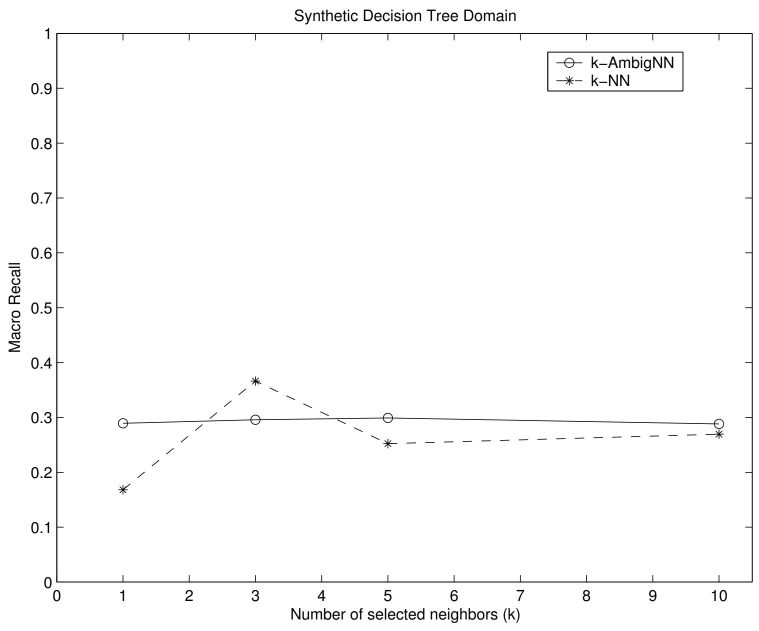

Figure 28. Macro Recall vs. number of neighbors for k-AmbigNN and k-NN calculated on the second decision tree domain.

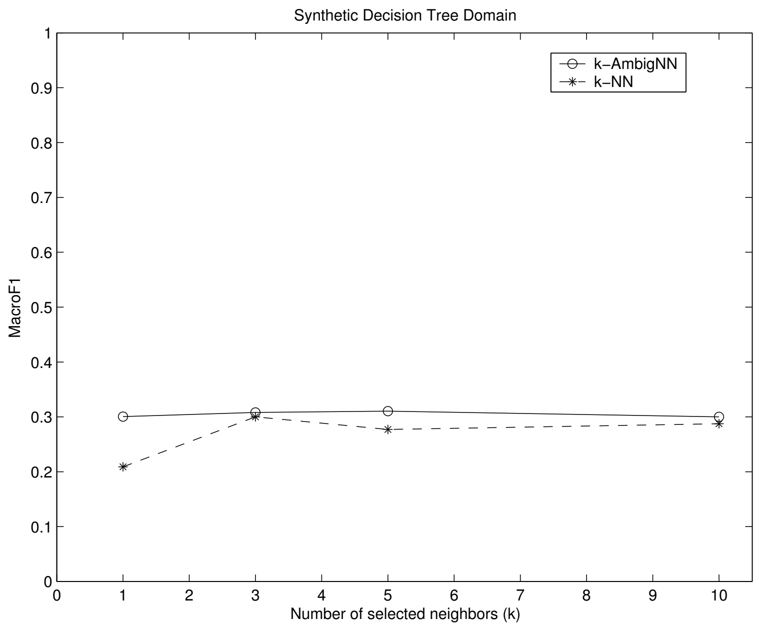

Figure 29. Macro F_1 vs. number of neighbors for k-AmbigNN and k-NN calculated on the second decision tree domain.

Figures 30 - 32 supports the claim we made about earlier macro experiments on the same synthetic decision tree domain.

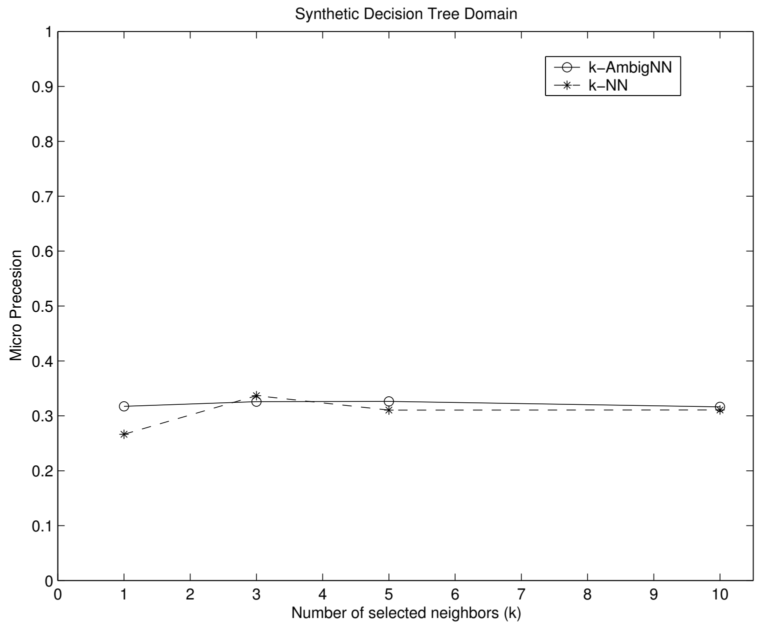

Figure 30. Micro Precision vs. number of neighbors for k-AmbigNN and k-NN calculated on the second decision tree domain.

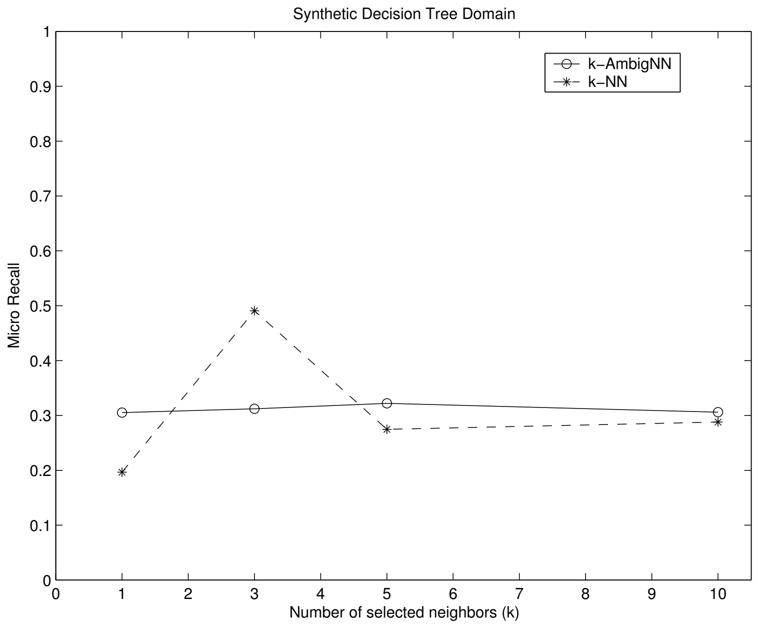

Figure 31. Micro Recall vs. number of neighbors for k-AmbigNN and k-NN calculated on the second decision tree domain.

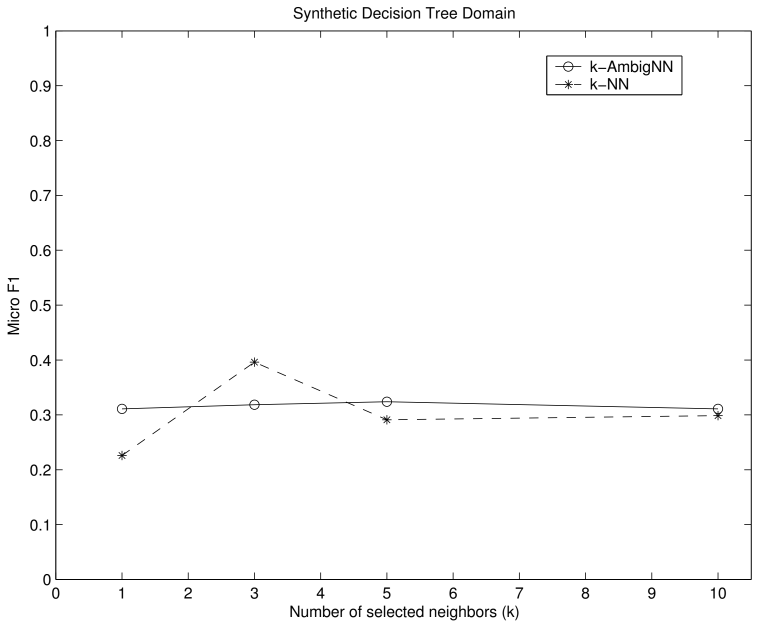

Figure 32. Micro F_1 vs. number of neighbors for k-AmbigNN and k-NN calculated on the second decision tree domain.

The next set of graphs show the two Boolean domains and indicate that the k-AmbigNN approach does outperform the more traditional approach. It is important to note that in these synthetic Boolean domains the attribute values and class labels are binary. An interesting pattern starts to emerge from the results indicating a convergence point for the algorithms. As the amount of ambiguity increases all the algorithms converge to a similar performance value. It is surmised that this converging pattern is due to the fact that so much ambiguity exists in the domain that it is just as likely to be all as none, effectively making "question marks" out of each domain.

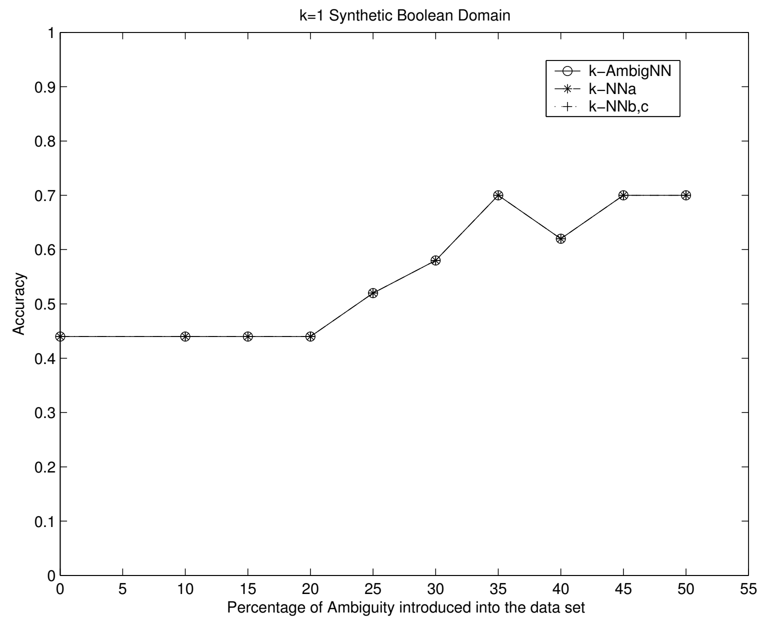

Figure 33. Performance of k-AmbigNN approaches on synthetically introduced ambiguity on synthetic Boolean domain.

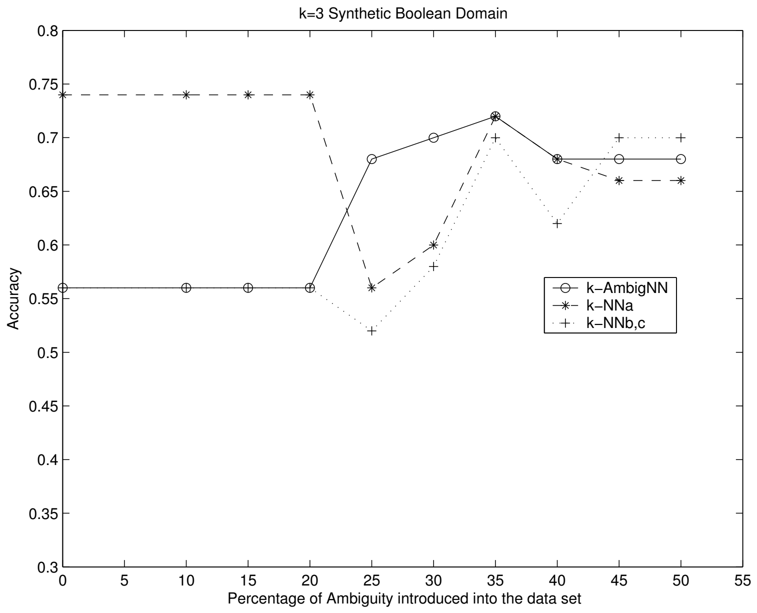

Figure 34. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on synthetic Boolean domain.

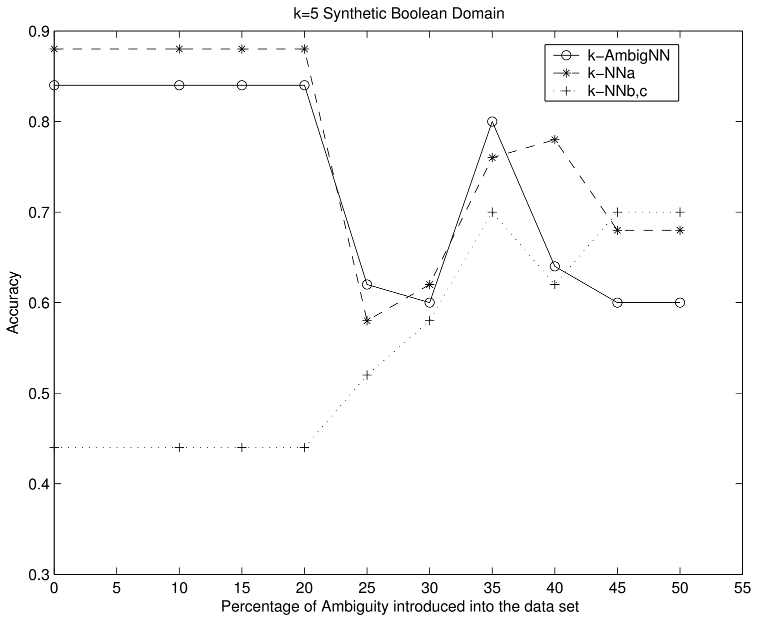

Figure 35. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on synthetic Boolean domain.

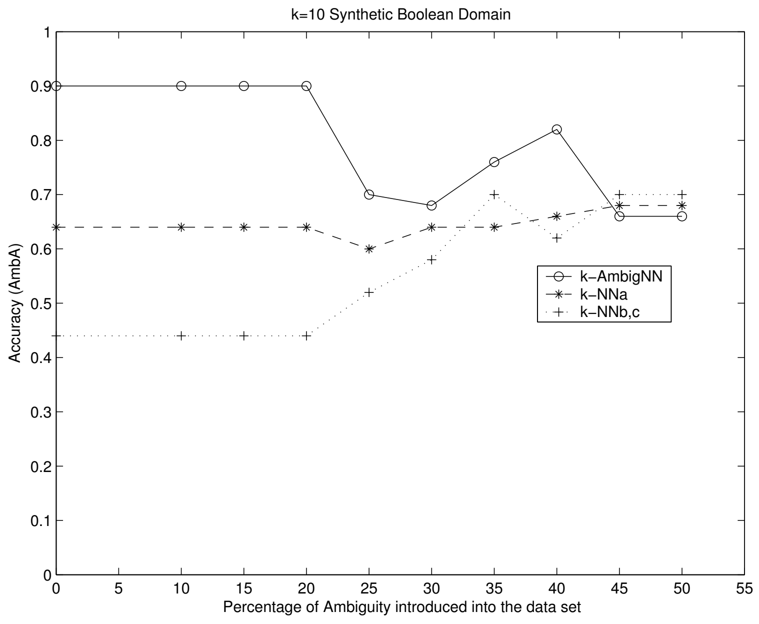

Figure 36. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on synthetic Boolean domain.

Shown in Figures 37 - 40 the k-AmgibNN approach performed as well, and in a few cases better, than the traditional approaches. These is a clear convergence of the approaches at 25% ambiguity. This trend can be explained by the fact that as ambiguity increases above a threshold the probability of randomly choosing the right label is balanced by the fact that no matter how many attributes are present in the class label there is always a contribution to the resulting label. More work may be need in order to more accurately model the amount of ambiguity in the accuracy measures.

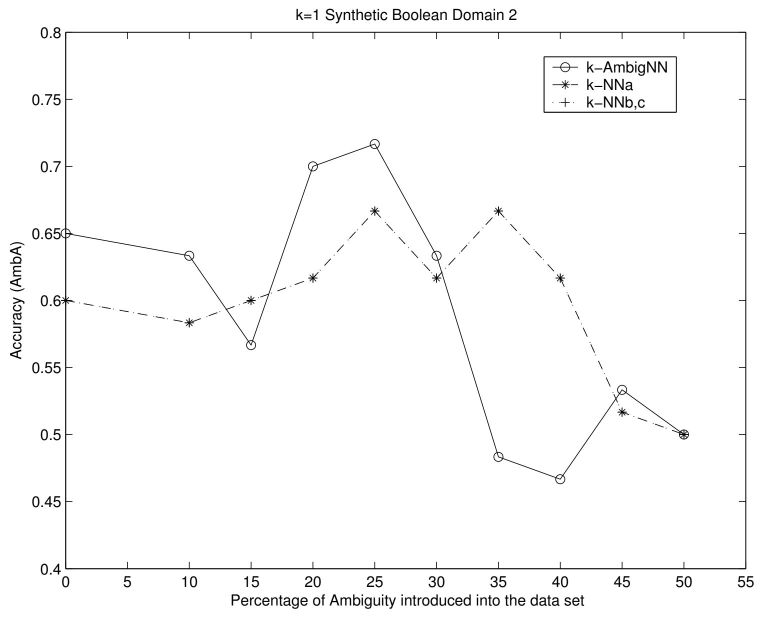

Figure 37. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on second synthetic Boolean domain.

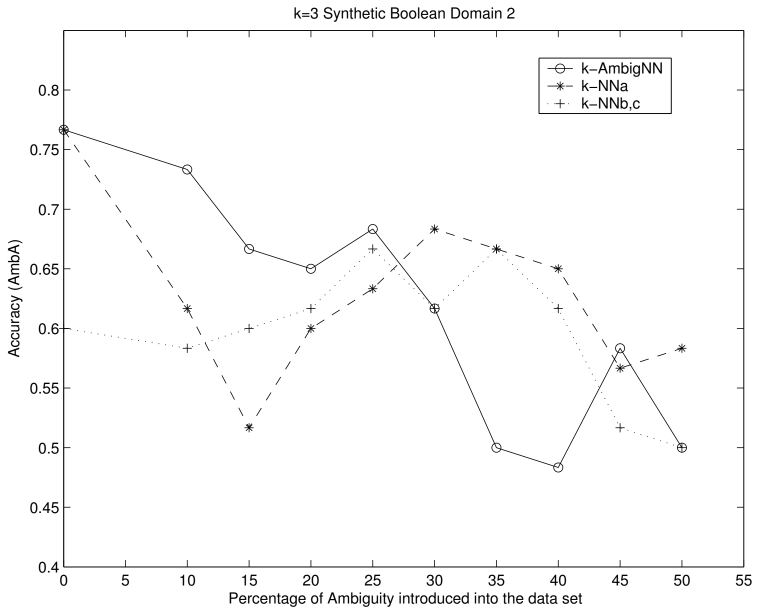

Figure 38. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on second synthetic Boolean domain.

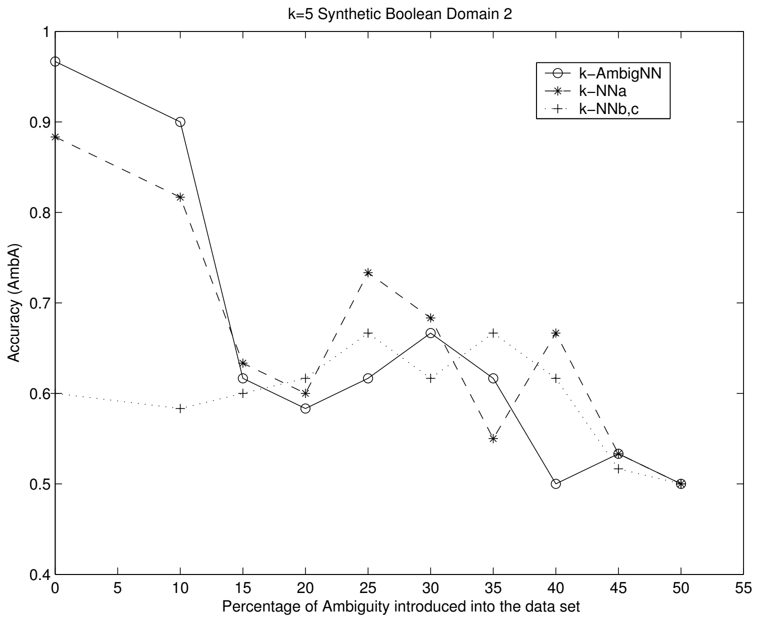

Figure 39. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on second synthetic Boolean domain.

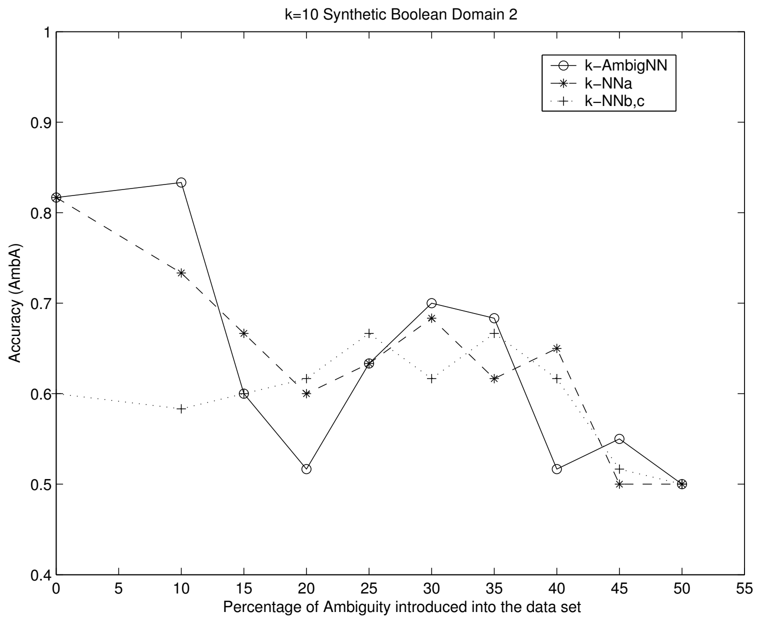

Figure 40. Performance of k-AmbigNN and k-NN approaches on synthetically introduced ambiguity on second synthetic Boolean domain.

Continued experimentation supporting this claim can be reviewed in Figure 41 and Figure 42. The domains presented are both UCI domains that have been modified in the same manner as the pure synthetic domains, to introduce ambiguity. For this UCI domain the results are a bit sharp due to the low number of data points in the domain. This experiment really shows the spread between the approaches.

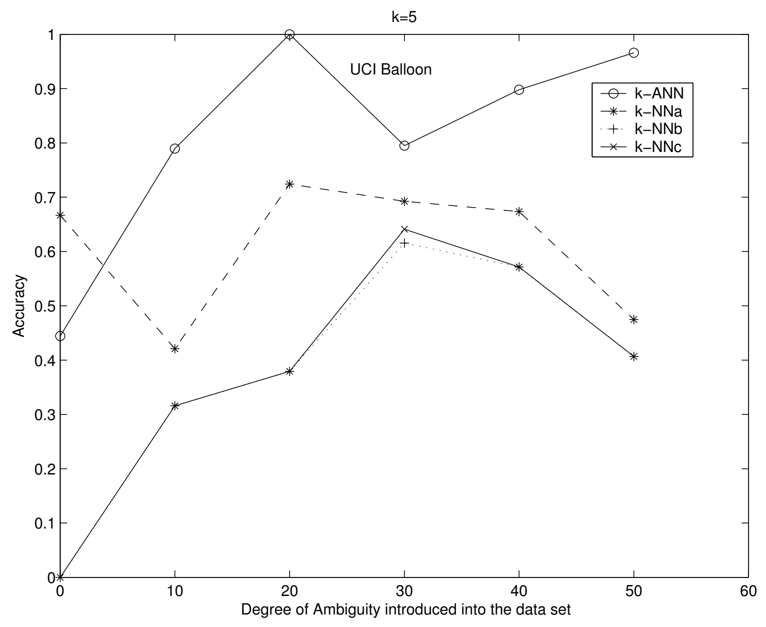

Figure 41. Performance of k-AmbigNN and k-NN on UCI domains with synthetic ambiguity.

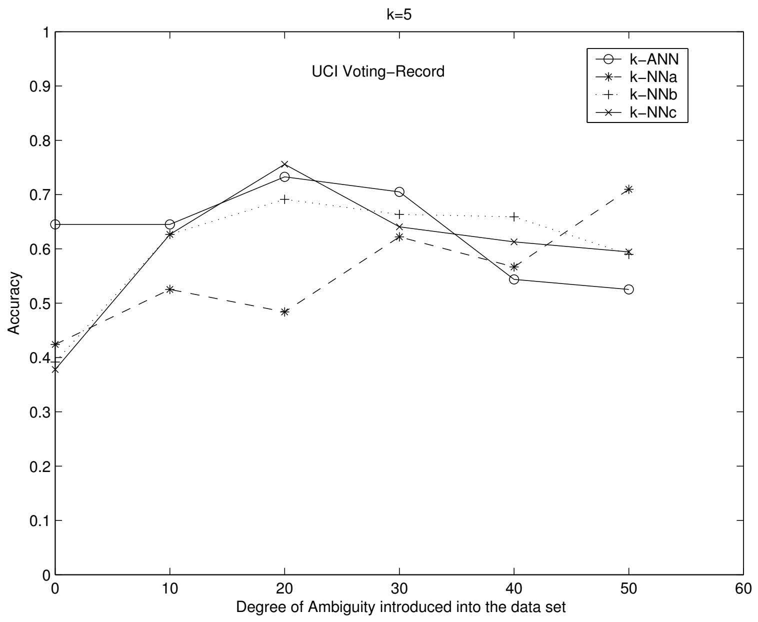

Figure 42. Performance of k-AmbigNN and k-NN on UCI domains with synthetic ambiguity.

The final phase of the experimentation is shown in Figures 43-Figure 49. This final series of domains are straight UCI Repository domains. These older domains show the performance of the k-AmbigNN on data with missing values and how it compares to the traditional approaches. In many cases k-AmbigNN performs as well if not better than the traditional approach. The notable exceptions are Figure 46 the mushroom domain, and Figure 43. It is surmised that the nature of these domains point to the poor performance. In the case of the tic-tac-toe domain there are very few instances and ambiguity and missing values are very impactful. While in the case of mushroom domain the domain is quite large and thus impacts performance. With little or no ambiguity the similarity approach given for k-AmbigNN performs as well as traditional approaches.

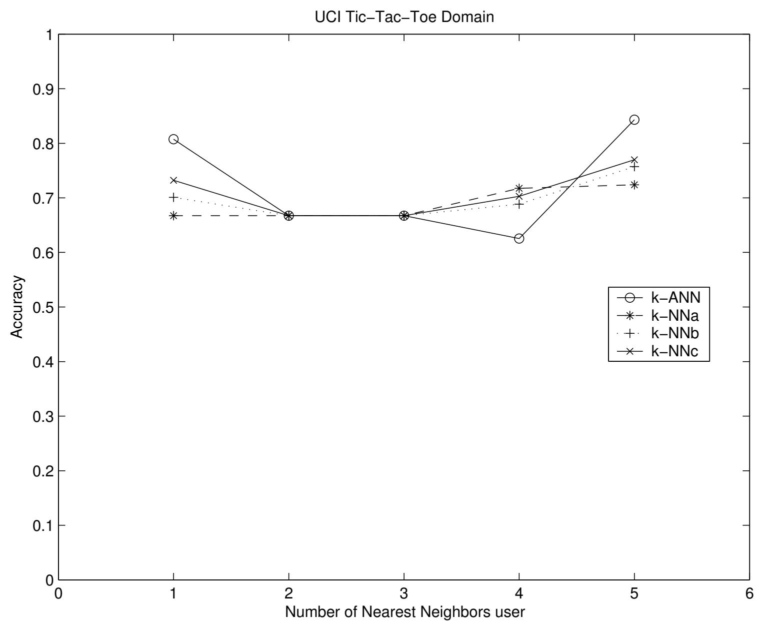

Figure 43. Performance of k-AmbigNN and k-NN approaches on Tic-Tac-Toe UCI domain.

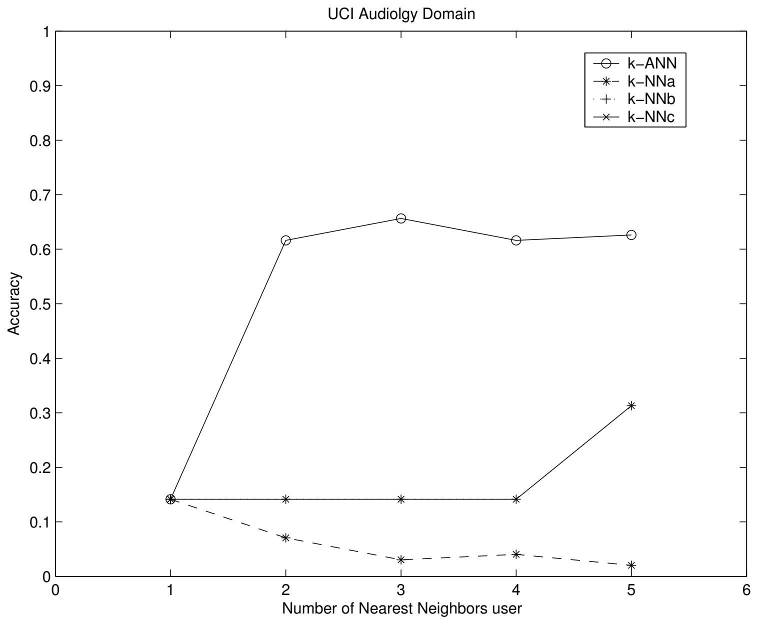

Figure 44. Performance of k-AmbigNN and k-NN approaches on Audiology UCI domain.

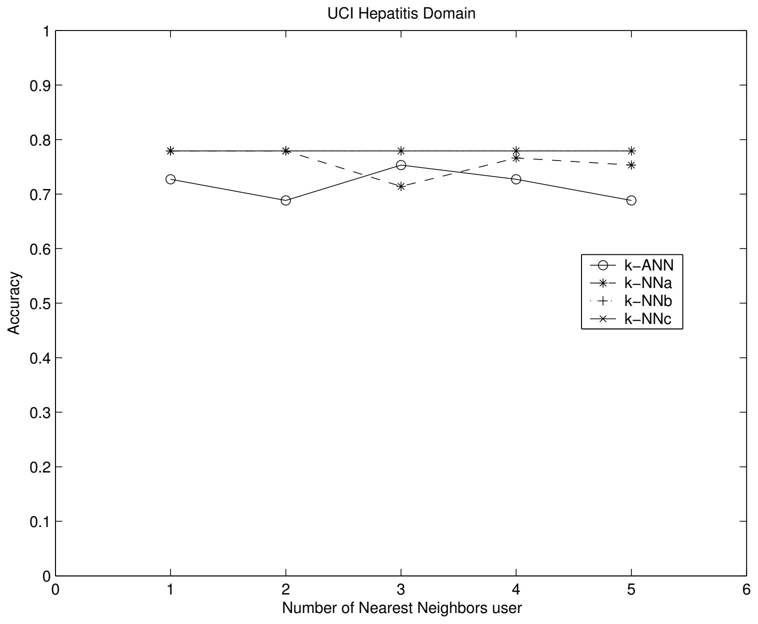

Figure 45. Performance of k-AmbigNN and k-NN approaches on Hepatitis UCI domain.

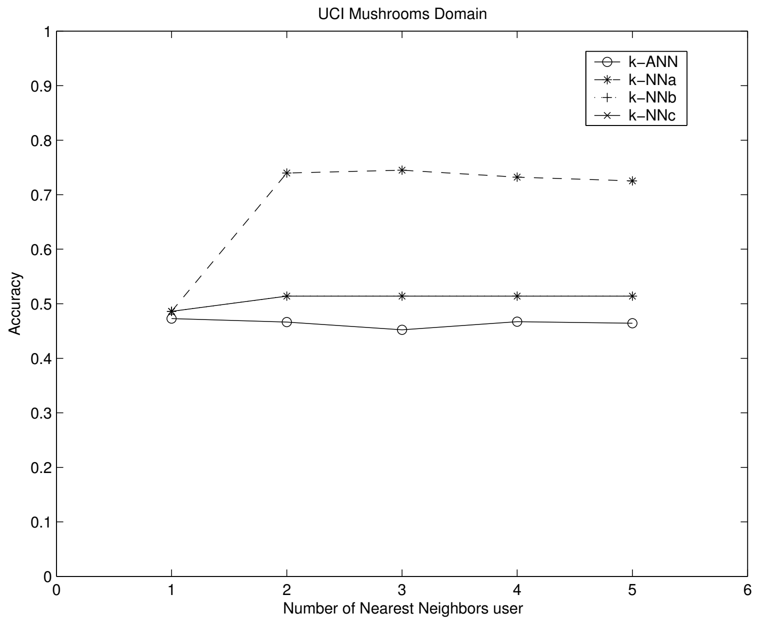

Figure 46. Performance of k-AmbigNN and k-NN approaches on Mushroom UCI domain.

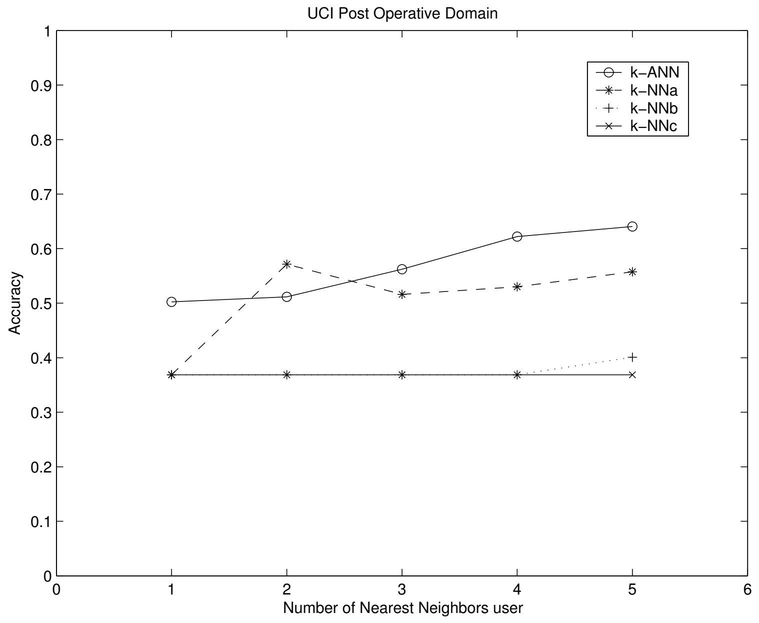

Figure 47. Performance of k-AmbigNN and k-NN approaches on Post Operative UCI domain.

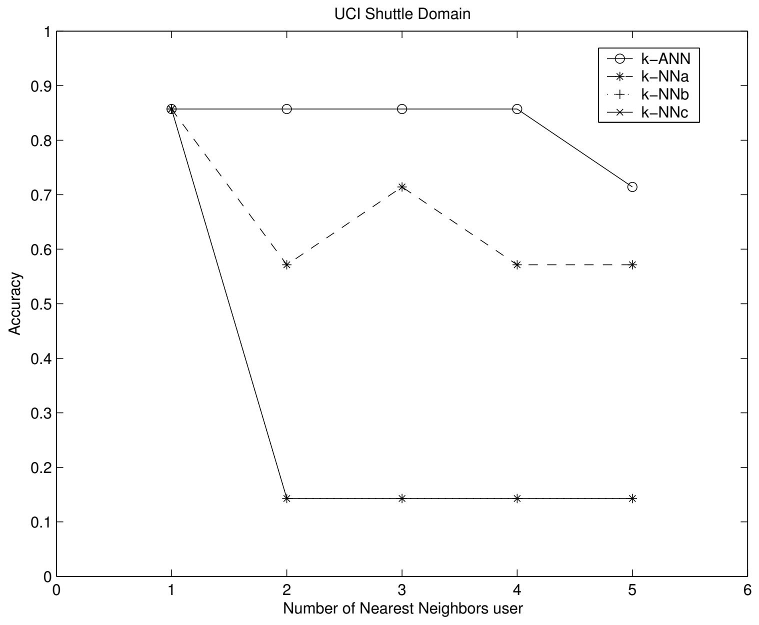

Figure 48. Performance of k-AmbigNN and k-NN approaches on Shuttle UCI domain.

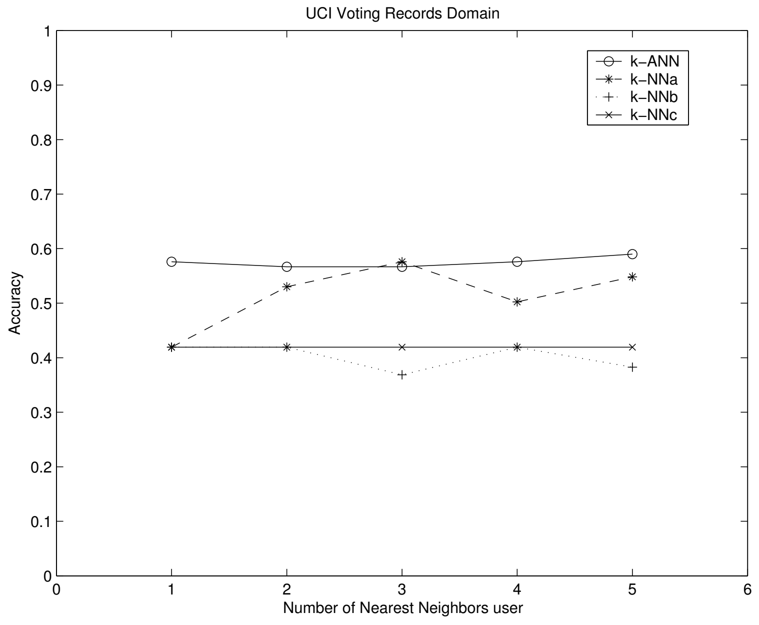

Figure 49. Performance of k-AmbigNN and k-NN approaches on Voting UCI domain.

In general the k-AmbigNN approach out performed the traditional approaches on ambiguous and non-ambiguous domains. On domains with moderate amounts of ambiguity the k-AmbigNN approach shines. The Micro, Macro experiments show that there is a clear improvement using the non-traditional approaches even though the results seem a bit flat. More experimentation is needed into an improved accuracy measure that accounts for partial class label correctness better than the approach presented in this thesis.

CHAPTER 5

Conclusion

In an attempt to pioneer research in the field of learning from ambiguously described examples, this paper has presented a simple solution on how to handle ambiguity in instance based classifiers. Three issues have been addressed: how to evaluate an example-to-example similarity function, how to choose the winning class label, and how to evaluate the classification accuracy. Because of the lack of available benchmark domains with ambiguous data, we had to experiment with artificially created data sets into which we introduced attribute vector ambiguity. In our experiments we observed some unexpected anomalies caused by the idiosyncrasies of the way we evaluated performance. Obviously, these issues call for more systematic research and can be expanding into other areas and other algorithms. To attract the attention of the scientific community to this unfairly neglected problem is the main motivation for this thesis.

In order to address the possible criticism that we had few real world domains that represent a significant amount of missing values, we believe that some of the UCI Repository data files have been sanitized in an effort to make the data easier for a wider variety of algorithms to use. This sanitation causes a lack in real world data that would include attribute vector ambiguity. It seems that for highly disparate data sets our application outperforms the traditional approaches. Our approach excels on un-sanitized data sets. It is our belief that more data domains would fall into the un-sanitized category if there were algorithms that could account for such data sets. More real world, un-sanitized, testing would aid in the continued improvement and contribution to this field.

There are two main areas of extension for this thesis project: first, the technology used for implementation and second, the scope of the algorithm. When we discuss the technology of implementation, we are discussing the efficiency in how the algorithm uses its resources and the ease of use by the user. A JAVA runtime environment is used for this current implementation. Little regard was given to the execution time of the algorithm. Improvements in this area are definite if this implementation of the k-AmbgNN Algorithm is to be distributed commercially. Extensions that allow the user to graph accuracies directly from the application are also a good idea for a properly constructed standalone application. Packages for MatLab would aid the adoption of the algorithm by the scientific community as well. If this algorithm presented in this thesis were a part of a larger Machine Learning program that would include other pattern recognition approaches, then again that would aid in its adoption.

With regards to the algorithm, we limited our research to the simple case of instance-based classifiers, specifically k-Nearest Neighbor algorithm. Weighted distance algorithm along with other case-based approaches should be considered for comparison or alteration to account for ambiguous attribute labels. As mentioned earlier, it is our intention to bring this data use case to the attention of the scientific community with the hopes that other classification approaches will be looked at again.

As data sets grow and algorithms mature, the machine learning community will look for new ways to improve upon old ideas and processes to better simulate what goes on in the world around us. To crack the shell of real learning is the ultimate goal. This research is very important because as humans we make decisions on associations (generalizations) on imperfect data every moment of every day. Accounting for these imperfections, or ambiguity, should be the focus of these data mining and pattern recognition algorithms to come. I stand on the shoulders of those who started down the path like Vannoorenberghe and Denoeux with their work with uncertain labels in belief decision trees.

Having surmised that classical nearest neighbor classifiers may be inadequate to deal with ambiguously described examples, a new modification to the traditional is suggested, k-AmbigNN (ambiguous nearest neighbor). Through the course of experimentation the k-AmbigNN algorithm had its performance compared with that of the classical k-NN rule and with that of a solution suggested by the fuzzy-sets community.

Importantly, it was realized that ambiguous domains call for a specially designed performance metric and voting scheme. The experiments with three different domains indicate that the performance of the k-AmbigNN compares very favorably with the performance of modest modifications of the classical k-NN classifier and with k-fuzzyNN. While k-NN apparently suffers from the replacement of partial ambiguity with total ambiguity, k-fuzzyNN seems to have been meant for more sophisticated uncertainties than those encountered in the tested domains.

One may complain that the experiments were not performed on a large set of benchmark domains as is common in the many machine-learning projects. The truth is that it was found that in the UCI repository only one such domain, `university` existed. It is speculated that other UCI domains might originally have contained ambiguities, too, but their authors sanitized them because the community was only used to totally unknown attribute values. However, the experience reported in this treatment indicates the fact that, say, season is either `spring` or `summer`—but not `fall` or `winter`—is qualitatively different from the "question-mark" situation and should not be ignored.

Finally, the work suggests there is a need to search for better methods to quantify such aspects as the degree of data ambiguity. It is the contention of this thesis to present one possible method for defining the ambiguity and handling the assignment in a robust manner. A simple solution has been presented here.

There is work being done at the University of Miami in Coral Gables, Florida, to use the Dempster-Shafer approach of accounting for ambiguous data and applying it to the attribute vectors of a Bayesian classifier. This work along with the research presented in this thesis lay the groundwork for other approaches to this ambiguous input pattern recognition problem. The ramifications are wide, spreading from medical diagnosis systems that are able to better predict outcomes based on imperfect data to Homeland Security issues with giving confidence to possible terrorist actions from data-mined phone conversations. The biggest place an algorithm like this can impact is real time image recognition. Again, this all comes back to the simple pattern recognition problem whether it can be done with imperfect data to an acceptable level of accuracy.

LIST OF REFERENCES

[1] Bezdek, J.C. (1981). *Pattern Recognition with Fuzzy Objective Function Algoritms*, Plenum Press, New York

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Belmont, CA: Wadsworth International Group, 1984.

[3] B. Dasarathy, *Nearest-Neighbor Classification Techniques.* Los Alomitos, CA: IEEE Computer Society Press, 1991.

[4] Dietterich, T. G. (1998). Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation, 10*, (7) pp. 1895–1924.

[5] Denoeux, T. (1995). The k-Nearest Neighbor Classification Rule Based on Dempster-Shafer Theory. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.25, pp. 804–813

[6] Denoeux, T. and Skarstein, M. (2000). Induction of Decision Trees from Partially Classified Data Using Belief Functions, *Proceedings of SMC* pp. 2923–2928

[7] Dunn, J.C. (1973). A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters, *Journal of Cybernetics* 3: pp. 32–57

[8] Elouedi, Z. Mellouli, and K. Smets, P. (2001). Belief Decision Trees: Theoretical Foundations *International Journal of Approximate Reasoning* Vol.28, pp. 91–124

[9] R. Fisher, *The Use of Multiple Measurement in Taxonomic Problems.* 7, 111–132: Annals of Eugenics, 1936.

[10] Ham, F. M. and Kostanic ,I.,*Principlces of Neurocomputing for Science and Engineering.* New York, NY, United States of America: McGraw-Hill, 2001.

[11] Keller, J. M., Gray, M. R, and Givens, J. A. (1985) A Fuzzy k-Nearest Neighbor Algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, 15(4), pp. 80–85.

[12] T. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[13] Newman, D.J., Hettich, S., Blake, C.L. and Merz, C.J. (1998). UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[14] J. Quinlan, *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA: Elsevier B.V., 2004.

[15] G. Shafer, *Dempster-Shafer Theory.* United States of America: online, 1999.

[16] Shafer, Glen (1997). *A Mathematical Theory of Evidence*, Princeton University Press

[17] Shafer, G., and Pearl, J. (eds.) (1990). *Readings in Uncertain Reasoning*. Morgan Kaufmann.

[18] Vannoorenberghe, P. (2004). On Aggregating Belief Decision Trees *Information Fusion*, Vol. 5, pp. 179–188

[19] Vannoorenberghe, P. & Denoeux, T (2002). Handling Uncertain Labels in Multiclass Problems Using Belief Decision Trees.*Proceedings of IPMU'2002*, Anneey, France, pp. 1919–1926

[20] I. Witten and E. Frank, *Data Mining Practical Machine Learning Tools and Techniques with JAVA Implementations. 72–75, 114–118.* here: Morgan Kaufmann Pub., 2000.

[21] Z. Elouedi, K. Mellouli, and P. Smets, *Classification with Belief Decision Trees.* Universite Libre de Bruxelles: IRIDIA, 2001.