GRAPH-BASED LEARNING USING A NAIVE BAYESIAN CLASSIFIER

by

ROBERT N. HAWES

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2006

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge my family: to Saysamone, my wife and companion who has been deprived of many hours from her husband for the research in putting together this paper; to Alexandra and Melyssa, my daughters, for reminding me what life is all about; to my late father, Robert, Sr., who was always supportive, encouraging, and challenging me to be more than I could be, and to my mother, Jeanette, who never stopped believing in me.

Also, I would like to thank every teacher and mentor in my life for their encouragement and dedication to the joy of learning. In particular, I would like to thank my advisor Dr. Diane Cook for introducing me to the wonders of machine learning, Dr. Manfred Huber for help in reasoning with uncertainty, and Dr. Larry Holder for the use of the subgraph generator.

As always, there are many, many people, too many to list here, who have always looked up to me and encouraged me to shoot for the stars. May I never let you down.

December 3, 2005

ABSTRACT

GRAPH-BASED LEARNING USING

A NAIVE BAYESIAN

CLASSIFIER

Publication No. _____

Robert N. Hawes, M.S.

The University of Texas at Arlington, 2006

Supervising Professor: Diane J. Cook

Graph-based data representation is becoming increasingly more commonplace, as graphs can represent some kinds of data more efficiently than relational tables. As such, interesting patterns in the form of subgraphs can be discovered by mining these graph-based datasets. Because the learned patterns can be used to predict future occurrences, it is necessary to learn graphical concepts that can optimally classify the data in the presence of uncertainty.

This work explores the construction and learning of optimal naïve Bayesian graph classifiers to distinguish between positive and negative graphs given a set of graphs as examples. Whereas most previous work in graph-based data mining has been restricted to exact graph matching algorithms, the classifiers discovered using this

approach are not similarly restricted, and thus are able to classify graphs in the presence of missing data.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

Graph-based data representation is becoming increasingly more commonplace, as graphs can represent some kinds of data more efficiently than relational tables. As such, interesting patterns in the form of subgraphs can be discovered by mining these graph-based datasets. Because learned patterns can be used to predict future occurrences, it is necessary to learn graphical concepts that can optimally classify the data in the presence of uncertainty.

Traditional data mining is the search for patterns in data. The discovered patterns may then used as classifiers to predict the relations among unseen data. An underlying assumption is that the patterns discovered in a given sample are indicative of the population at large. That is, the relations discovered among the data attributes can be assumed to provide a categorical classification with a degree of certainty.

For example, a large terrorist database could hold information on the activity of known terrorists, as well as suspected terrorists. If the terrorists showed repetitive behavior or attributes just prior to an attack, a rule could be discovered with a certain degree of confidence, say 85%, that the pattern of behavior leads to a terrorist attack.

As the use of graphs for data increases and the uncertainty within that data also increases, it becomes more important to build classifiers for graph-based data that can

handle uncertainty well. Imagine what would happen if a classifier mispredicted or failed to predict a terrorist attack, and the counter-terrorism task force as a result was deployed to the wrong place at the wrong time to arrest the wrong individual. Consequently, if the classifier is to be used to do data prediction, it must have some basis in the actual underlying distribution of the relations predicted.

Statistics and probability are among the best known means for predicting properties of an actual population given samples from that population, and as a result, any probabilistic classifiers could only benefit from their predictive capability. One of the best known methods of probabilistic classification is the naïve Bayesian classifier, which has been used with great success in other domains.

We hypothesize that a graph-based learning algorithm built upon probability theory can form the basis of an accurate predictor and useful learning algorithm. In particular, we postulate that an algorithm that learns graph concepts using a naïve Bayes method can achieve comparable or better performance than other learning algorithms on structural data. To validate this hypothesis, we will adapt a naïve Bayes text-based classifier to handle graph-based data. Specifically, we will adapt the definition of attributes as words to attributes as substructures within a graph, and define how to determine the conditional probabilities of substructure-based attributes for naïve Bayes learning. We will test this hypothesis by building a classifier learning process and embedding it as an alternative evaluation method in the SUBDUE algorithm [Cook and Holder, 2000], where we further enhance the classifiers by learning a super-classifier as a sequence of such classifiers. This classification system will then be tested against two

sets of synthetic data and the mutagenesis data set. The first set of synthetic data consists of a small space of incrementally larger training samples, so that we can do a detailed observation of the algorithm's discovery process. The second set of synthetic data is a much larger set of small graph structures built by inducing random mutations in a common substructure at a specified mutation frequency to simulate uncertainty. Finally, classifiers are built and tested on the mutagenesis data, which is a publicly available dataset of large molecular structures and classifications on whether or not they have been found to mutate and cause cancer.

1.2 Thesis Outline

To yield strong predictive capability in graph classification, this thesis introduces an adaptation of the naïve Bayes method specifically for graph classification within the SUBDUE framework [Cook and Holder, 2000]. Therefore, Chapter 2 provides background information on the naïve Bayesian algorithm that is being adapted. Because this adaption is a probabilistic extension of SUBDUE, Chapter 3 discusses graph-based data mining and the SUBDUE algorithm, whereas Chapter 4 is an introduction to probabilistic graph concepts. After covering the required background material, the naïve Bayesian graph classifier algorithm is presented in Chapter 5, Chapter 6 details the experimental results on both synthetic data and the mutagenesis data set, and Chapter 7 presents the conclusions of this thesis.

CHAPTER 2

INTRODUCTION TO NAÏVE BAYESIAN CLASSIFIERS

The naïve Bayes classifier [Mitchell, 1997], also called the naïve Bayes learner, is a highly practical learning method. Its performance is comparable to that produced by neural network and decision tree learning.

A naïve Bayes classifier applies to situations in which a function is needed that maps a set of attributes to a single value from a finite set of values:

$$f_{nb}: a_1 \text{ X } a_2 \text{ X...X } a_n \rightarrow v \tag{2.1}$$

The classifier is given a training set that consists of attribute tuples $<a_1, a_2,..., a_n>$ with the corresponding target value, v, for each training instance. It responds by generating a function over the attributes that best predicts the training data. Afterwards, the classifier can be given tuples of unseen attributes, $<a_1, a_2,..., a_n>$ and asked to predict the target value, or classification of each new instance.

In Bayesian classification, new instances are classified by assigning the most probable target value, known as v_{MAP}, to the classification. Given a set of attributes $<a_1, a_2,..., a_n>$ for an instance v_{MAP} can be calculated as follows:

$$v_{MAP} = \underset{v_j \in V}{argmax}\, P(v_j | a_1, a_2, \cdots, a_n) \tag{2.1}$$

Using Bayes theorem, the equation for v_{MAP} can then be rewritten as

$$v_{MAP} = \underset{v_j \in V}{argmax} \frac{P(a_1, a_2, \cdots, a_n | v_j) P(v_j)}{P(a_1, a_2, \cdots, a_n)}$$

$$= \underset{v_j \in V}{argmax}\, P(a_1, a_2, \cdots, a_n | v_j) P(v_j) \qquad (2.2)$$

The naïve Bayes classifier assumes that the attribute values are mutually independent given a target value, [Mitchell, 1997]. That is, the probability of a conjunction of attribute values is simply the product of the probabilities of the individual attributes given the target value.

$$P(a_1, a_2, \cdots, a_n | v_j) = P(v_j) \prod_i P(a_i | v_j) \qquad (2.3)$$

Subsituting equation (2.3) into (2.2) produces the approach used by the naïve Bayesian classifier:

$$v_{NB} = \underset{v_j \in V}{argmax}\, P(v_j) \prod_i P(a_i | v_j) \qquad (2.4)$$

in which v_{NB} denotes the predicted class.

[Mitchell, 1997] provides an example algorithm for text classification. In this example the instances to classify are text documents. Examples of target classifications that can be learned include "interesting news articles," "machine learning," and "not interested." If a computer can learn to distinguish these categories based on the content, it can then be used to automatically filter a large volume of online documents, and present potentially interesting documents to the user. Some probabilistic systems to classify text have been proposed by [Lewis, 1991], [Lang, 1995], and [Joachims, 1996].

The text classifier described in [Mitchell, 1997] is a general algorithm based on the naïve Bayes classifier method. In it, the instance space *X* is the set of all possible text documents, each of which consists of variations of word strings and punctuation.

The training examples form a subset of these documents and their target classifications are discrete values within a finite set of classifications. For example, the set of target values could be *like* and *dislike*, depending on whether a person considers the articles interesting or not.

Two main design issues are encountered when applying the naïve Bayesian-based approach to a text classification problem. The first issue is how to represent the text document as a set of attributes. The second is how to estimate the probabilities used by the classifier.

To make the example concrete, assume we are given 1000 documents of which 700 are classified as *like*, and the remaining 300 as *dislike*. Furthermore, assume we are given as a new document the paragraph above and are asked to classifiy it as *like* or *dislike*. To classify the document we instantiate Equation 2.4:

$$v_{NB} = \underset{v_j \in \{like, dislike\}}{argmax} P(v_j) \prod_{i=1}^{44} P(a_i | v_j)$$

$$= \underset{v_j \in \{like, dislike\}}{argmax} P(v_j) P(a_1 = two | v_j) P(a_2 = main | v_j) \ldots P(a_3 = classifier | v_j)$$

To calculate the probabilities in the above equation, the algorithm needs to learn two kinds of estimates from the training data. First, the a priori probabilities of each classification, and second, the conditional probabilities of a word at a position in the document given each classification. The second probability is further simplified by assuming a conditional independence on the location of the word in the document.

The algorithm in [Mitchell, 1997] uses the given frequencies of the documents for the a priori values. For example, the probability of document being *like* in the given

example, P(*like),* is 700 / 1000 = 0.7 and for a document being *dislike* is 300/1000 = 0.3. The algorithm then calculates the conditional probabilities of each word for each classification $P(w_j \mid v_i)$ using m-conditioning as follows:

$$P(w_j|v_i)=\frac{n_j+1}{n+|Vocabulary|}$$

where n_j is the number of occurences of word w_j in the training set, n is the total word count in the training set, and *Vocabulary* is the total set of distinct words found in the training set.

This approach of discovering a vocabulary and determining frequencies on that vocabulary is further extended to handle graphs in the naïve Bayesian graph learner in Chapter 5. In-depth analyses and extensions of the naïve Bayesian approach can be found in [Elkan, 1997], [Zheng, et. al, 1999], [Zheng, 1998], and [Friedman et. al., 1997].

CHAPTER 3

INTRODUCTION TO GRAPH-BASED DATA MINING

3.1 Graph Concepts

In order to understand the fundamentals of graph-based data mining, it is helpful to understand some of the more important graph concepts. In this section, we will describe graphs, subgraphs, substructures, substructure instances, graph isomorphism, and graphs as concepts.

A *graph*, G is defined as a set of vertices, V, and edges, E, where each edge is defined by a pair of vertices in V, such that:

$$G(V, E) \equiv \{V, E \mid \forall e(v_i, v_j) \rightarrow e \in E \wedge v_i \in V \wedge v_j \in V\}$$

Edges may be undirected (traversible in both directions) or directed (traversible only from one node to the other). Undirected graphs are typically used to represent correlations among vertices or bidirectional transitivity. Directed graphs typically represent attributes or states and one-way relationships between those states.

For a definition of *subgraphs*, we can look to [Washio and Motoda, 2003]. Subgraphs can be general subgraphs, induced subgraphs, connected subgraphs, ordered or unordered trees, or paths.

1. *General subgraphs* are graphs whose set of vertexes form a subset of the set of vertexes in a larger graph, and whose edges are also a subset of the edges from the same graph.

2. *Induced subgraphs* are graphs that are general subgraphs and have been generated by removing vertices and the edges connected to those vertices.
3. *Connected subgraphs* are general subgraphs, where all nodes in the graph are reachable from at least one node by traversing the edges.
4. *Trees* are connected subgraphs without cycles and the number of edges into a vertex, a.k.a . the in-degree, is no larger than one.
5. *Ordered trees* are trees whose vertex and edge labels follow some canonical ordering along one or more dimensions, for instance top to bottom, or left to right.
6. *Paths* are trees with zero or more edges going out, a.k.a. the out-degree, of every vertex.

Substructure and substructure instances are defined in [Cook and Holder, 2000]. A *substructure* is a connected subgraph that is part of a given input graph. A *substructure instance* is a set of edges and vertices in an input graph that is considered to match graph-theoretically to a given substructure.

[Vanetik and Gudes, 2004] define two levels of *isomorphism,* or ways that two graphs can be considered to match. The first method is called an unlabeled isomorphism. Two graphs G_1 and G_2 are said to be isomorphic if for every edge in G_1 there is a mapping or *bijection* function to the edges in G_2:

$$Unlabeled\ isomorphism: (v,u) \in E(G_1) \Leftrightarrow (\varphi(v), \varphi(u)) \in E(G_2)$$

In the second method, labeled isomorphism, vertexes and edges may carry labels, and the related elements of the graphs must also share the same labels. [Vanetik

and Gudes, 2004] only consider the vertex labels for their definition, but if edges are also labeled (as they can be in SUBDUE [Cook and Holder, 2000]), edges as well as vertices on both sides of the isomorphism must share the same label for the graphs to be considered fully label isomorphic.

A subgraph isomorphism is an graph isomorphism between a substructure of one graph, say G1, and another graph, G2. In this paper, we denote the set of subgraph isomorphisms of G1 in G2 as I_{G1}^{G2} .

A graph concept is a collection of substructures or subgraphs that by its structure and a set of combination rules represents a potentially recurring idea or concept. Examples of concept learners include SUBDUECL [Gonzalez et al., 2000] and the Naïve Bayes Graph Classifier (NBGC) discussed in Chapter 5. As an idea, a graphical concept defines a set of vertice labels and the relationships between those labels necessary to identify instances of the concept. Once identified or discovered, instances of graphical concepts in a larger graph can be replaced with a single vertex representing the entire structure [Gonzalez et al., 2000].

3.2 Graph-Based Data Representation

Traditional databases are relational, organized as tables and pre-classified as bundles of like attributes, with foreign keys representing relations among the classes. However, in domains where the number of potential relations among entities is innumerable or some of the classes of entities are not known, representing the data as a graph can be better. The relations can be directly traversed, and placeholders (null

values in relational tables) do not have to occupy space for unknown or non-existing attributes.

A graph-based data representation has been used for many domains, including natural language processing [Rassinoux et al., 1998], molecular biology [Cook et al., 2001], email classification [Aery and Chakravarthy, 2004], and world-wide-web mining [Mendelzon et al., 1997].

3.3 Graph-Based Data Mining

Graph-based data mining is the process of finding graph concepts in a collection of attributes represented as a graph such that some property of the data can be predicted given the concepts discovered. The concepts discovered are typically subgraphs of the original data and represent recurring relations among the attributes.

Graph-based data mining can be supervised or unsupervised. In supervised data mining, positive and negative labeled examples are given to the data mining algorithm, and the algorithm searches for subgraphs that fit the positive examples better than the negative examples. In unsupervised learning, subgraphs are sought that best fit the given examples, typically using a frequency metric to identify "interesting" nuggets. There are many examples of unsupervised learners, including SUBDUE [Cook and Holder, 2000], AGM [Inokuchi et al., 2003], FSG [Kuramochi and Karypis, 2001], HSIGRAM/VSIGRAM [Kuramochi and Karypis, 2004], GBI [Yoshida et al., 1994], WARMR [Dehaspe and Tolvonen, 1999], FARMAR [Nijssen and Kok, 2001], and MolFea [De Raedt and Kramer, 2001]. There has also been recent research into

supervised graph-based learners, including SUBDUE, [Gärtner, 2005],and [Deshponde et al., 2003].

Three types of graph-based evaluation strategies are subgraph frequency, minimum description length, and inductive logic; the last two of which are implemented in SUBDUE. Examples of frequent subgraph discovery algorithms include AGM and FSG. Frequent subgraph discovery algorithms search for subgraphs whose frequency are greater than a given support threshold [Vanetik et al., 2002]. Minimum description length discovery algorithms search for the subgraph which will best compress the graph space using an information theoretic measure [Cook and Holder, 2000]. Finally, inductive logic discovery algorithms search a hypothesis space for a graph-based concept that can be described as disjunctive normal form or conjunctive normal form, i.e. logical rules, such that they describe more positive input examples than negative input examples [Gonzalez et al., 2000].

3.4 The SUBDUE Algorithm

The SUBDUE algorithm [Cook and Holder, 2000] is a graph-based discovery process that heuristically searches for substructures which compress the original input graph and represent structural concepts. The input graph can be a single labeled graph structure or multiple graph structures. Each vertex in the structures can support labels to identify entities and attributes in the graph structure. Labels can also be applied to edges so that relationships between entities and attributes can be named. In addition, to support concept-based learning, the input graph instances themselves can be labeled as

positive or negative examples of the concept to learn. The SUBDUE algorithm is a beam search as shown in Figure 3.1.

The SUBDUE discovery algorithm begins by initializing parent, child, and best lists to empty sets, and the number of processed substructures to 0. The parent list is then updated to contain all single vertex substructures. At this point the main loop of the algorithm begins. While there are still substructures in the parent list and the number of processed substructures has not exceeded the user defined limit, the algorithm continues.

The main processing loop of SUBDUE proceeds as follows. For all the parents in the list, a nested loop is entered. In the nested loop, the parent with the best value is removed from the list and a set of child substructures is generated by finding all instances of parent and adding one edge extensions. Then, each child is evaluated to return a value and then sorted into the child list by its value, keeping the child list size at or below the user defined beam width.

Once all the children of a parent have been processed, the parent is added to the best list and sorted by value, truncating it to the user defined limit of best substructures to keep. When the current parent list is emptied, the child list becomes the parent list by swapping the two lists.

Finally, when there are no more extensions to be found or the user-defined search limit has been met, the algorithm terminates and returns the best substructures found in the best list.

SUBDUE(*Graph, BeamWidth, MaxBest, MaxSubSize, Limit*)
Return the list of best substructures.

1. *Initialize substructure lists.*
   - ParentList ← {}
   - ChildList ← {}
   - BestList ← {}
   - ProcessedSubs ← 0
   - Create a substructure from each unique vertex label and its single-vertex instances.
   - Insert the remaining substructures in ParentList.
2. *Perform a beam search to find the best substructures until the limit is met or there is nothing left to search.*
   - **while** ProcessedSubs <= Limit **and** ParentList is not empty **do:**
     - **while** ParentList is not empty **do:**
       - Parent = RemoveHead(ParentList)
       - Extend each instance of Parent in all possible ways.
       - Group the extended instances into Child substructures.
       - **foreach** Child **do:**
         - **if** SizeOf(Child) <= MaxSubSize **then:**
           - Evaluate the Child.
           - Insert Child in ChildList in order by value.
           - **if** Length(ChildList) > BeamWidth **do:**
             - Destroy the substructure at the end of the list.
       - ProcessedSubs ← ProcessedSubs + 1
       - Insert Parent in BestList order by value.
       - **if** Length(BestList) > MaxBest **then:**
         - Destroy the substructure at the end of BestList.
     - Switch ParentList and ChildList.
3. *Return the best substructures..*
   - Return BestList.

Figure 3.1. The SUBDUE algorithm.

In addition, if the user has specified multiple iterations, the graph is compressed using the best substructure, and the algorithm repeats until no more positive example graphs remain or the graph can no longer be compressed.

In the original algorithm, the child evaluation step uses minimum description length compressibility to determine the child values. In the naïve Bayes graph classifier approach (Chapter 5), we will evaluate each child by evaluating the performance of a

classifier built from the child's substructure and evaluated over the user supplied examples.

CHAPTER 4

PROBABILISTIC GRAPH CONCEPTS

4.1 Probabilistic Graph Concepts

Probabilistic graph concepts are graph concepts that add transitional or correlational probabilities to components of a graph. This means that the vertices, edges, or even substructures within the graph can be assigned a probability.

Examples of probabilistic graph concepts include Markov models, Bayesian belief networks, dynamic belief networks, Markov Decision Processes, and linear filters, among others. See [Murphy, 1998] for a description of the relations between many popular models.

4.2 M-Estimates

One of the problems that occurs when estimating frequencies from sampled data is the dominance of the more frequent data over the less frequent data. When taking the Bayesian product of multiple frequencies, it is likely that some of the properties will have an unusually low frequency probability, due to the paucity of the training samples. One of the methods used to deal with this is *m-estimates* [Mitchell, 1997].

m-estimates use the prior probability of the value being measured and an equivalent sample size to augment the frequencies. Where the original frequency given the sample space is count(value=X)/count(sample), the m-estimate becomes:

$$\hat{P}(value = X \mid Samples) = \frac{count(value = X) + mp}{count(samples) + m} \tag{4.1}$$

where m is the *equivalent sample size,* and p is the prior probability of the value being X. The uniform prior assumption [Mitchell, 1997] sets m=count(values) and p=1/count (values). The resulting m-estimate, and the one we will use is:

$$\hat{P}(value = X \mid Samples) = \frac{count(value = X) + 1}{count(samples) + count(values)} \tag{4.2}$$

4.3 Calculating Substructure Probabilities

Three methods in current literature for calculating the probability of a substructure for a given graph are 1) transaction-based selection, 2) maximum likelihood estimation [Coble, 2005], and 3) substructure extension. We also derive a method to calculate the probability of a feature as a member of a set of candidate features. The main distinguishing characteristic of these four methods is the granularity of the random field drawn for sampling. They also differ in runtime estimations.

4.3.1 Transaction-Based Selection

This technique calculates the probability that a graph instance randomly drawn from a set of graph instances contains a given substructure [Kuramochi and Karypis, 2004]. The graph space is divided into independent transactions or instances of connected graphs. The probability of the graph being drawn at random using this metric is:

$$P(s_i \mid T) = \frac{|\{t_j \, s.t. \, |I_{s_i}^{t_j}| > 0 \wedge t_j \in T\}|}{|T|} \tag{4.3}$$

where s_i is the substructure under consideration, T is the set of transactions or user provided graph instances in the graph space, and *I* is the set of subraph isomorphisms of s_i in T.

Consider the set of strings {“abc”, “aab, ” “cc”}, where consecutive letters are joined via an edge, and the graph space The probability of “ab” would then be 2/3.

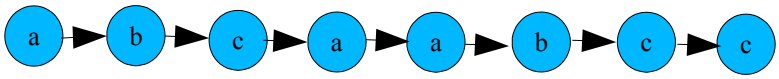
4.3.2 Maximum Likelihood Estimation

This technique [Coble, 2005] for calculating substructure probabilities calculates the probability that the substructure is encountered in the given instances or transactions. This formula is:

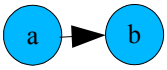
$$P(s_i|T)=\frac{(|V(s_i)|+|E(s_i)|)|I_{s_i}^T|}{|V(T)|+|E(T)|} \tag{4.4}$$

where *I* is the set of isomorphic instances of substructure *s* in T. The frequency is calculated as a ratio of the number of nodes and edges that are part of the given substructure's instances to the total number of vertices and edges in the training data as a whole.

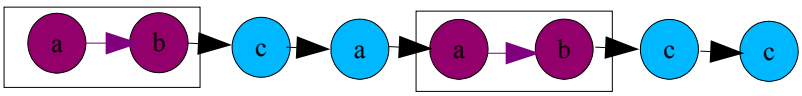
Consider the string “abcaabcc,” where consecutive letters are joined via an edge and shown in Figure 4.1. The size of two-character substructure like “ab”, would be 3, represented by 2 vertices and one edge. The probability of “ab” would then be 3 * 2 / (8+7) = 2/5.



(a)



(b)



(c)

Figure 4.1. Maximal likelihood coverage on example string: (a) “abcaabcc” as directed graph, (b) two character substructure with two vertices and one edge, and (c) maximal likelihood coverage.

4.3.3 Substructure Extension

The substructure extension technique [Noble and Cook, 2003] calculates substructure probabilities conditioned on induced subgraphs of a given size. The child substructures are considered extensions of their parent substructures, and the parent substructures represent the structural environmental for the children. The formula for this calculation conditions the probability of the child graph on its parent by counting the isomorphic instances of the child and dividing by the number of isomorphic instances of the parent:

$$P(c_i | p_j, T) = \frac{|I^T_{c_i}|}{|I^T_{p_j}|} \qquad (4.5)$$

For example, as shown in Figure 4.2 for the string “abcaabcc,” to calculate the probability of “ab” given “a” would be 2/3.

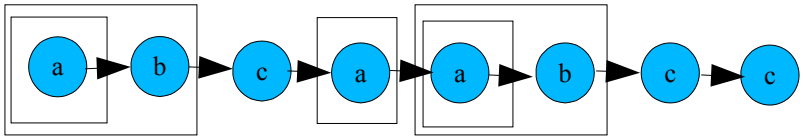


Figure 4.2. Substructure extension probabilities.

4.3.4 Graph Feature Selection

For the purposes of this paper, we will define a graph-based feature as a potentially recurring graph substructure within a specific domain. The feature selection technique will then be a method for calculating the probability of the existence of a specific feature in a graph instance for a given domain. Because features are derived from graph-based instances, they take on domain-specific biasing as represented in the example instances and form a vocabulary within the domain. The formula for this calculation is to take the ratio of the number of isomorphic instances of a feature to the number of isomorphic instances of all features:

$$P(f_i|T)=\frac{|I_{f_i}^T|}{|I_F^T|} \tag{4.6}$$

For example, in the string “abcaabcc,” to calculate the probability of “ab” given minimum and maximum feature size of 2 vertexes, the substructures of that size would first be enumerated to yield the features {“ab”, “bc”, “ca”, “aa”, “ab”, “bc”, “cc”}. Because there are only two instances of “ab” among seven total feature instances, the

resulting probability of “ab” would then be 2/7. On the other hand, if the minimum feature size is set to 0 and the maximum to 2, as shown in figure 4.3, the features would then be {“a”, “b”, “c”, “ab”, “bc”, “ca”, “aa”, “ab”, “bc”, “cc”, “abc”, “bca”, “caa”, “aab”, “abc”, “bcc”}. In this case, the total number of substructures would be 3 + 2 + 3 + 2+ 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 = 20, and the probability of “ab” would be 2/20. Clearly, the size of the candidate features impacts the probabilities and the number of isomorphic checks that have to be done.

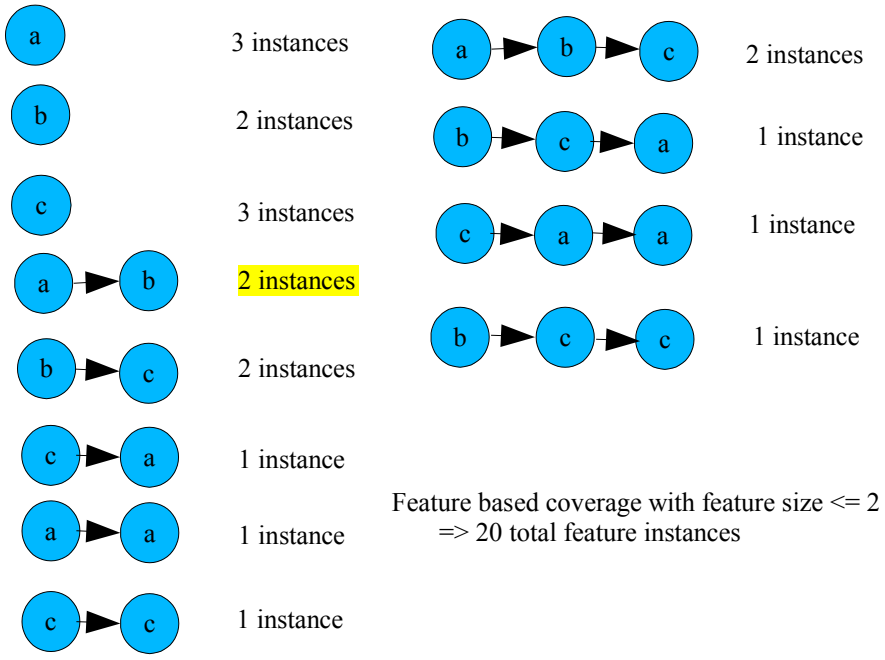


Figure 4.3. Graph feature probability.

If all substructures within a given size range are considered features, Equation (4.6) can be rewritten as:

$$P(f_i|T)=\frac{|I_{f_i}^T|}{|\{S(T)\,s.t.\,minFeatureSize\leq|E(s_i)|\leq maxFeatureSize\}|} \qquad (4.7)$$

where S(T) is the set of substructures in the input transaction graphs, and only those instances between the feature size bounds are counted, avoiding the expensive graph isomorphism checks.

4.3.4 Comparison of Techniques

The substructure extension technique is applicable primarily to calculating the probability of a substucture given a subset of its nodes and vertices, and specifically designed for subgraph extensions. The remaining three techniques calculate the probability of the existence of a substructure given a larger encapsulating substructure, making them more applicable to the desired conditional calculation. All three techniques require scanning the data to calculate the number of isomorphic occurrences of the target substructure in the graph, which has been shown to be NP-complete [Cook, 1971] and will therefore dominate all calculations. As a result, we focus on the additional cost invoked for each algorithm.

Transaction-based estimation can terminate the isomorphic scan as soon as a single instance is found, therefore requiring scanning half of the vertices and edges in each transaction on average, so the number of comparisons would be O(|E(G)| + |V(G)|). After doing the isomorphic comparisons, this method has all the information it needs and can complete in O(C) time.

Maximum likelihood estimation adds in a calculation of the size of the graph. This can be completed in O(|E(S)| + |V(S)|), and a one-time calculation of the size of the graph-space, which requires counting all nodes and edges (O(|E(G)| + |V(G)|). This

calculation is still relatively small, but the calculations are not easily extensible to account for conditional independence between subgraphs.

Finally, graph feature selection does allow for independence between relations by being able to control which features are in the feature set, and therefore allows for higher statistical precision. This precision, however, comes at a price. In this case there is a one time added calculation of the count, O(|F|) of subgraph extensions, since we need to count all feature instances.

Since the naïve Bayesian method's accuracy assumes conditional independence among the features used for classification, the graph-based feature method proves to be the most promising of the methods. In addition, because it is possible for features to be completely missing from some subsets of the data, m-conditioning should be applied for all calculations. Therefore, in the next chapter we apply both of these techniques, graph-based feature probabilities and m-conditioning, to generate the a priori and conditional probabilities for the Naïve Bayes Graph Classifier algorithm and embed it in the SUBDUE algorithm [Cook and Holder, 2000].

CHAPTER 5

THE NAÏVE BAYESIAN GRAPH CLASSIFIER

5.1 Classifying Graphs

The Naïve Bayesian Graph Classifier (NBGC) determines the likelihood that a graph belongs to a given class. It makes the naïve Bayes assumptions that all measured attributes occur independently of one another and occur with a normal distribution. The classifier is an extension based on the naïve Bayesian Text classifier.

The Naïve Bayesian Graph classifier differs from the text classifier in several respects. It classifies a graph, whereas the text classifier classified a document string. In the current implementation, because the embedding environment (SUBDUE) only allows for two classes, NBGC also considers only two classes, positive membership and negative membership, but can easily be extended for multiple classes like the text-based algorithm. Instead of a vocabulary of words, however, the graph classifier uses a vocabulary of graph features. And finally, instead of returning the most likely category, it returns the probability of membership.

Recall that the probability of a classification of independent attributes given Bayes theorem is:

$$P(v_j|a_1, a_2, \cdots, a_n) = \frac{P(a_1, a_2, \cdots, a_n|v_j)P(v_j)}{P(a_1, a_2, \cdots, a_n)} \quad (5.1)$$

The denominator on the right hand side of Equation (5.1) is a normalizing constant and the ratio of the numerators of the probabilities for each classification is equal to the ratio of the probabilities of the classification. Therefore, to simplify calculations we deal with only result of the numerators, known as the likelihood [Mitchell, 1997], which we shall denote as λ. In other words, the likelihoods of each classification are calculated by using the following calculations, assuming conditional independence on the graph features:

$$For\ each\ c_j \in C, \lambda_{c_j} = P(c_j) \prod_{s_i \in G} \prod_{s_i \in F} P(s_i | c_j) \tag{5.2}$$

that is, the likelihood of each classification in the set of classifications is the product of the prior probability of the classification and the conditional probability of each substructure in the graph that is a feature instance for the classification, iterating over all substructure instances in the graph to classify. However, if you allow the features in the classifier to be position independent and iterate over the classifier substructures, you can rewrite Equation (5.2) as follows:

$$\lambda_{c_j} = P(c_j) \prod_{f_i \in F} P(f_i | c_j)^{|I_{f_i}^G|} \tag{5.3}$$

where $I_{f_i}^G$ is the set of isomorphic instances of feature f_i in graph G using the notation in Section 3.1and the main difference is that the iteration is over the feature vocabulary instead of the substructure instances that match a feature. Consequently, the conditional probability for each feature given each classification is raised to the power of the number of isomorphic instances of the feature in the graph to classify. The algorithm to calculate this value can now iterate over the features in the vocabulary,

count the isomorphic instances of each feature once, and update the likelihoods for each classification in parallel.

Finally, the overall probability of a classification for a graph is calculated from the likelihoods by normalization [Mitchell, 1997]. By normalizing the likelihoods, we are effectively reintroducing the denominator of Equation (5.1). The graph has probability 1 that it belongs to one of the classifications, so the sum of the conditional probabilities must be 1. This is accomplished by scaling each classification's likelihood over the total likelihood as follows:

$$P(c_j|G)=\frac{\lambda_{c_j}}{\sum_i \lambda_{c_i}} \tag{5.4}$$

The resulting complete algorithm instantiated for binary classification is shown in Figure 5.1.

NB_CLASSIFY_GRAPH(*Graph*)
Return the a posteriori probability that the Graph is classified by this classifier.
1. *Initialize likelihoods to prior probabilities using Equation (5.5).*
2. *For each feature in the classifier, condition the likelihoods on the isomorphic occurrences of the feature.*
    - For f in F:
        - $\lambda\oplus \leftarrow \lambda\oplus * P(f \mid \oplus)^{|I(Graph, f)|}$
        - $\lambda\ominus \leftarrow \lambda\ominus * P(f \mid \ominus)^{|I(Graph, f)|}$
3. *Return a posteriori probability of classification.*
    - Return $\lambda\oplus / (\lambda\oplus + \lambda\ominus)$

Figure 5.1 NB_CLASSIFY_GRAPH algorithm

5.2 Learning a Classifier from a Set of Features

The Naïve Bayesian Graph Classifier uses a set of probability estimates for determining an *a posteriori* estimate that a graph belongs to the class given each of its

substructures. The probabilities used for this calculation include the *a priori* estimates of a graph belonging to the class without further information, as well as the conditional probability that a graph belongs given each of its subgraphs. This is done using the LEARN_NB_GRAPH_CLASSIFIER algorithm given in Figure 5.2.

LEARN_NB__GRAPH_CLASSIFIER takes a set of features, the target concept for the classifier, and the maximum feature size to be used for classification. The examples provide the sample space from which to extract probabilities, and the maximum feature size is used to provide a means of avoiding overfit (the maximum feature size in SUBDUE is defaulted to 2, as per experiments in Chapter 6). The algorithm calculates $P(\oplus)$, the prior probability that a given graph is a member of the class. For each subgraph of the target concept, the algorithm also calculates $P(f|\oplus)$, the conditional probability that a feature can be found in the class.

LEARN_NB_GRAPH_CLASSIFIER(*Features*, *Examples*)
This functions learns the a priori values for $P(\oplus)$ *and* $P(\ominus)$, *as well as the conditional probabilities* $P(f \mid \oplus)$ *and* $P(f \mid \ominus)$ *for every feature f in Features*. *Examples is a set of Graphs along with a* $\oplus$ *or* $\ominus$ *classification.*
1. Learn the a priori values for P($\oplus$*) and P(*$\ominus$*). Use m-conditioning on uniform prior probility of positive or negative.*
- $P(\oplus)$, $P(\ominus)$ ← calculate prior pobabilities using Equation (5.5)

2. Collect the required $P(f \mid \oplus)$ *and* $P(f \mid \ominus)$ *for each f in Features. Use m-conditioning on uniform prior probability of subgraphs.*
- For each f in Features:
  - calculate feature probabilities using Equation (5.6)

Figure 5.2. LEARN_NB_GRAPH_CLASSIFIER algorithm.

A classifier for NBGC is a set of features with associated probabilities, and a confidence threshold. The probabilities consist of the calculated a priori values for the

classification values, and the probabilities of each feature for each possible classification value. The confidence threshold represents the minimum amount of certainty that must be present to accept the classification from the classifier (see Section 5.5 for more on the confidence threshold). For example, one of the classifiers in the experiments (Chapter 6) was recorded as shown in Figure 5.3.

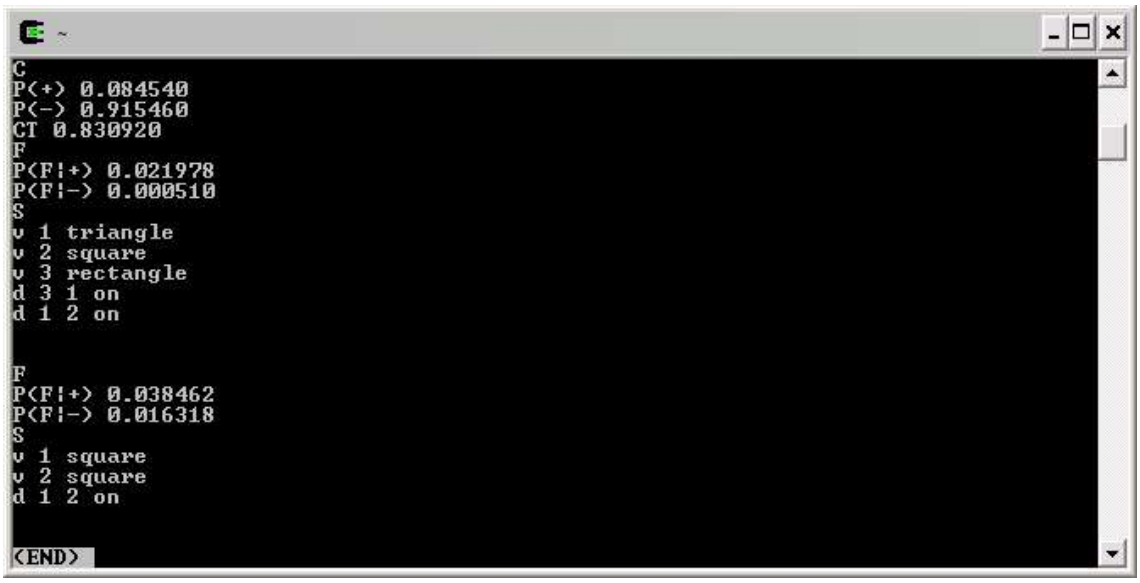


Figure 5.3. Classifier output.

In the figure, the prior probability of a graph being a positive instance is 0.0845, the confidence threshold is 0.831, and there are two features used for classification (see Section 4.2.4 for definition of a graph-based feature): 1) a rectangle on a triangle on a square, with a probability of 0.0220 in positive instances and 0.000510 in negative instances; and 2) a square on a square, with a probability of 0.0385 in positive instances and 0.00163 in negative instances.

The prior probabilities in the classifier are calculated using feature selection probabilities and m-estimation. The prior probability of a graph being positive or negative can then be calculated using the following formula:

$$\hat{P}(c)=\frac{|I_F^c|+1}{|I_F^T|+|C|} \tag{5.5}$$

that is, the ratio of all isomorphic feature occurrences in transactions with classification C to all feature occurences in all transactions, m-conditioned on the probability that each classification has a uniform prior probability of occurrence.

It is important to discount the subsumed features, as retaining them will violate the naïve Bayesian independence assumption. A subsumed feature is a substructure that is subgraph isomorphic to another feature. Naturally, a string such as "ab" has a natural dependence on both "a" and "b".

For example, in the string "abcaabcc," the following are candidate substructures for use as features if the maximum length is 3: {"a", "b", "c", "ab", "bc", "ca", "aa", "cc", "abc", "bca", "caa", "aab", "bcc"}. If the feature "ab" is selected, then features "a" and "b" must not be selected. The probability of "ab" would then be 2 / (0 + 0 + 3 + 2 + 2 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 1) = 2/16 or 0.125 instead of 2/(3 + 2 + 16) = 2/25 = 0.08.

The next step in the algorithm is to calculate the conditional probabilities for the features given the target classifications using the following formula:

$$\hat{P}(f_i|c_j)=\frac{|I_{f_j}^{c_i}|+1}{|I_F^{c_i}|+|F|} \tag{5.6}$$

that is, the total number of isomorphic occurrences of each feature in the transactions with the given classification to the total number of feature occurrences in graphs with the target classification, m-conditioned on the probability that each feature has a uniform prior probability.

5.3 Feature Extraction

In Naïve Bayesian text classifiers, the classifiers use the frequency of word appearance to classify text because they are the building blocks for natural language. In natural language, the granularity of the building blocks to use for frequency could have been set to any of a number of different levels, from phonemes, tokens, and roots on one extreme, to phrases and clauses at the other extreme.

In a similar vein, a graph classifier should also have a measurable building block or attribute to use as the basis for comparison. Because the classifier assumes the building blocks are independent from one another, it also needs a granularity that maximizes the independence assumption.

Independence granularity in a graph can exist at one of several levels. It could exist at the vertices themselves, in the relations between the vertices, as transitive relations between vertices (acyclic paths), or as clustered relations between vertices (subgraphs). Because the first three forms of granularity are all specific instances of subgraph granularity, the Naïve Bayesian graph classifier uses subgraph granularity for the attributes.

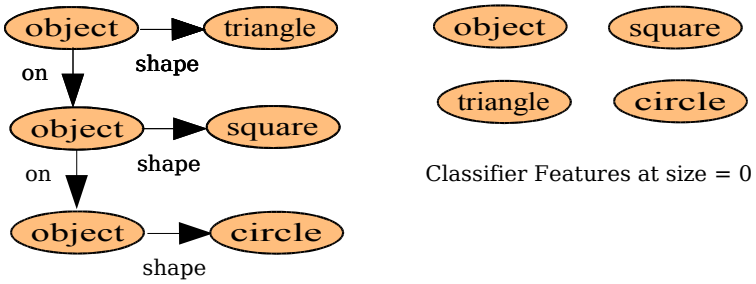
To extract the features as subgraphs, NBGC extracts the features from a graph instance using the EXTRACT_UNIQUE_SUBGRAPHS algorithm shown in Figure 5.4

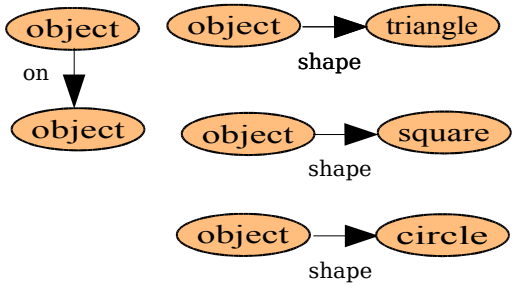
and Figure 5.5 shows the results of running the algorithm. All unique subgraphs with no more than a given number of edges in the target graph are used for the features. To extract the features, first the individual vertices are added to an empty feature list, then the feature list is iteratively grown by adding features that have only one more edge than than those currently in the feature list. The algorithm terminates when there are no more instances to add or the number of edges extended is equal to the maximum feature size.

EXTRACT_UNIQUE_SUBGRAPHS(*Graph, ExtractedSubgraphs, MaxFeatureSize*)
This function extracts all unique connected subgraphs of Graph by adding one edge at a time. ExtractedSubgraphs is the set of unique graphs already extracted from Graph.

1. *Get the Root vertex instances.*
    - Instances = getVertexes(Graph)
    - CurrentSize = 0;
2. *While new instances have been extracted, add them and look for more.*
    - While Instances→size > 0 and CurrentSize < MaxFeatureSize
        - ExtractedSubgraphs ← ExtractedSubgraphs ∪ Instances
        - Instances = extendInstances(Instances)
        - CurentSize++

Figure 5.4. EXTRACT_UNIQUE_SUBGRAPHS algorithm.





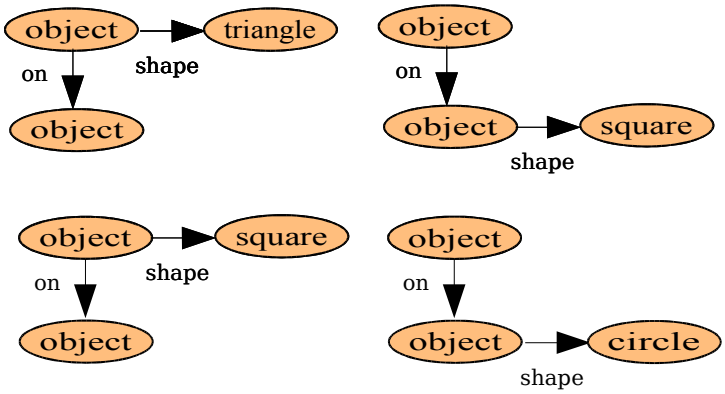


Figure 5.5. Extracting classifier features.

5.4 Discovering the Best Feature Set

Once the set of features has been extracted, they cannot be used as is. This is because the existence of an edge between two substructures defines a relation between

the substructures, and the more often this relation occurs the stronger the correlation between the substructures. The very existence of a graph depends on the existence of its subgraphs, so using features that share subgraph isomorphism violates the dependency assumption necessary for naïve Bayes to be accurate.

Of course, extracting a set of independent subgraphs comes at a cost. The number of ways to partition a graph into independent sets is exponential in the number of edges, and what is desired is not just any feature set, but the one that best explains the data. To find the best independent set using brute force would therefore be expensive computationally, because it would require testing an exponential number of combinations.

SUBDUE [Cook and Holder, 2000] uses a beam search to find the best subgraph. It does this to reduce the search time from an exponential search to a polynomial search bounded on the beam size. When the algorithm searches for the best subgraph, it starts with single vertex instances, gives them each a score, and discards the candidate graphs whose score falls below the beam threshold. It then incrementally grows the set by adding all single edge instances to the best candidates, and iterates the score-prune-extend procedure until no graph extensions outperform the current best.

Likewise, a beam search is employed by NBGC to find the best classifier that can be constructed given a candidate graph G and a set of training instances (this search is done at the evaluate child step of the SUBDUE algorithm of section 3.4). A classifier for NBGC consists of a set of prior probabilities for each of the classifications, a set of features with the probability of each feature given each classification, and a

confidence threshold (discussed in Section 5.5). For each candidate graph, NBGC finds the classifier to use by building trial classifiers using different combinations of substructures found in the candidate graph and evaluating the performance of the constructed classifiers over the training samples provided.

The NBGC heuristic is a greedy heuristic that starts by initializing a list of best feature sets with an empty feature set as the only member. The heuristic then iteratively selects each best feature set *bfs* from the best feature set list. If the best feature set hasn't changed from the last iteration, that feature set is returned. If there are any untried independent features remaining, a classifier is built for each remaining feature by concatenating only that feature with the features in *bfs*. Each classifier is tested against the training data and the feature set is inserted into the list of feature sets ordered by value of the classifier built using the set, truncating the list to the beam size. In the experiments for the NBGC SUBDUE implementation, we fixed the beam size at 3.

The classifiers are scored with respect to their ability to minimize error in classification. The error of a classifier with respect to a training sample is defined as the difference between the probability returned by the classifier on a training instance and the target probability of the training instances. For each training instance, the target probability is 1 if it is a member of the class and 0, otherwise. The average squared error for the classifier is therefore the sum of the squared error of the training instances divided by the number of training instances. The score is defined as the difference from 1, so that maximizing the score is equivalent to minimizing the error:

$$\bar{Error}(\hat{P}(C|T))^2 \equiv \frac{\sum_j (\hat{P}(c_i|T_j) - P(c_i|T_j))^2}{|T|} \tag{5.7}$$

$$Score \equiv 1 - \bar{Error}(\hat{P}(C|T))^2 \tag{5.8}$$

The complete algorithm is shown in Figure 5.6.

```
NB_EVALUATE_CHILD(Graph, Examples, MaxFeatureSize)
This function returns the accuracy of the Graph over the Examples using naïve Bayesian classification.
1. Collect all the features to be used for classification.
    • Features ← {}
    • EXTRACT_UNIQUE_SUBGRAPHS(Graph, Features, MaxFeatureSize)
    • BestFeatureSets ← {}
2. While true.
    2a. If there are no independent graphs remaining in Features score based on minimum error.
            return 1 – sqrt(avgError)
    2b. For each bfs in BestFeatureSets
        2b1. For each remaining independent subgraph f in Features not in bfs
            FeatureSet ← bfs ∪ f
            LEARN_NB_GRAPH_CLASSIFIER(FeatureSet, Examples)
            Reset counts
                correct⊕ ← 0
                correct⊖ ← 0
                sqerror = 0.0
            Test the learned classifier against each example
            For each ex in ⊕ Examples
                score ← NB_CLASSIFY_GRAPH (ex)
                error = 1.0 – score
                sqerror += sq(error)
            For each ex in ⊖ Examples
                score ← NB_CLASSIFY_GRAPH (ex)
                error = 0.0 – score
                sqerror += sq(error)
            avgError = sqerror / |Examples|
            If FeatureSet is at least as good as three best, add to BestFeatureSets.
    2c. Prune all sets from BestFeatureSet that don't perform as well as the best 3.
```

Figure 5.6. NB_EVALUATE_CHILD algorithm.

5.5 Learning Staged Classifiers

A modern technique to improve classification is to move from single classifiers to staged classifiers. Two examples are hierarchical classifiers [Dumais and Chen, 2000] and chain classifiers [Xiao et al., 2003]. In the chain classifier, each stage acts as a filter, weeding out data not easily classified by a later, more finely tuned filter. In the hierarchical classifier, the upper levels act as a kind of ontological ordering, by predicting the subgoups that are valid for classification at lower levels. In both cases, the classifiers are grouped as networks, with the express purpose of streamlining the classification. In general, a classifier network takes as input some data to classify, categorically subclassifies the data through a directed acyclic network of classifiers, and determines a final category. Examples of classifier networks are shown in Figure 5.7.

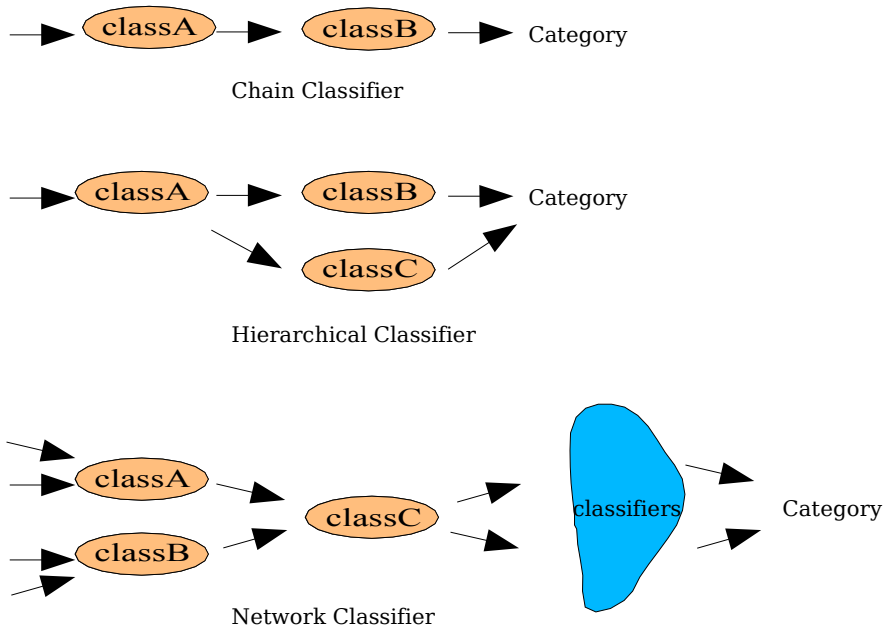


Figure 5.7. Classifier networks.

By combining classifiers, we will take advantage of SUBDUE's [Cook and Holder, 2000] compression part of the algorithm to set the stage for learning a refining classifier on each iteration, such that when they are all combined in a "super-classifier", the results are improved.

The way classifiers are chained together depends on how the data is treated by the classification system. For a binary classifier, as we implement in SUBDUE based on the naïve Bayesian Graph classifier, the two sources of error in the classifier are false positives and false negatives. If the problem is too many false positives, then the classifier C1 can perform better if another classifier C2 precedes C1 that performs better by correctly classifying the C1's false positives. On the other hand, if there are too many false negatives, then a separate classifier C2 that performs better on those false negatives will certainly enhance the performance of C1.

After each iteration of SUBDUE a classifier is learned and its performance is calculated. The examples correctly classified by the new classifier are subsequently removed from the training examples prior to the next iteration and the classifier details are recorded.

For classifiers that return only the category of classification, the classifier to learn must be determined from the performance on the training data. For example, if the classifier is generating too many false positives, it can be made more specific by chaining another classifier trained to filter out the results of the previous classifier, as in the chained classifier shown in Figure 5.7. To do this, the second classifier is trained on all the samples that the first classifier classified as positive. The overall classification

probability is the probability of the conjunction of the classifiers' probabilities, which can be the product of the individual probabilities if the classifiers' output is assumed to be independent.

If, on the other hand, there are too many false negatives, the classifier can be split into a hierarchical system of three classifiers, as shown in the hierarchical classifier in Figure 5.7. One classifier classA would be placed immediately before the existing classifier, say classB, and trained to predict which data classB can classify correctly. The data predicted as problematic for classB would be sent to a new classifier classC, to be trained on the data classB would filter erroneously. The probability of the stage would then be a calculated by calculating the probability of the disjunction of the two classifiers' probabilities, which is the the sum of the individual properties less the probability for overlap. The overlap calculation gets more complex with every added classifier.

A standard classifier generally returns the most probable classification without a degree of membership, whereas the naïve Bayesian graph classifier, implemented as a binary classifier, returns the probability of class membership. This returned probability makes the calculation of the overall probability given multiple classifiers more tractable.

The ideal probability of a classification given multiple naïve Bayesian classifiers is given by the following formulas using Bayes theorem:

$$\hat{P}(+|Classifiers, graph) = \frac{\hat{P}(+)\hat{P}(Classifiers, graph|+)}{\hat{P}(Classifiers, graph)} \qquad (5.8)$$

where $\hat{P}(+|Classifiers, graph)$ represents the *conditional* probability that the graph

is a positive instance of the class given a set of classifiers for the class, and $\hat{P}(+)$ is the *a priori* probabilitiy of a graph being a positive or negative instance. As in single classifier calculations, we can use the method of likelihoods [Mitchell, 1997] , treating the numerator as the likelihood and the denominator as a normalizing constant:

$$\hat{\lambda}(+|Classifiers, graph) = \hat{P}(+)\hat{P}(Classifiers, graph|+) \tag{5.9}$$

$$\hat{\lambda}(-|Classifiers, graph) = \hat{P}(-)\hat{P}(Classifiers, graph|-) \tag{5.10}$$

(5.11)

$$\hat{P}(+|Classifiers, graph) = \frac{\lambda(+|Classifiers, graph)}{\lambda(+|Classifiers, graph) + \lambda(-|Classifiers, graph)}$$

$$\hat{P}(-|Classifiers, graph) = 1 - \hat{P}(+|Classifiers, graph) \tag{5.12}$$

Formulated this way, the only unknown is the probability of the classifier values and the graph, given the positive or negative instances. Assuming the probability of the graph is independent of the probability of positive instances allows us to rewrite the likelihood calculations as follows:

$$\hat{\lambda}(+|Classifiers, graph) = \hat{P}(+)P(graph|+)\hat{P}(Classifiers|+) \tag{5.13}$$

$$\hat{\lambda}(-|Classifiers, graph) = \hat{P}(-)P(graph|-)\hat{P}(Classifiers|-) \tag{5.14}$$

where $\hat{P}(Classifiers|+)$ is the probability of the classifiers returning a true positive, $\hat{P}(Classifiers|-)$ is the probability of the classifiers returning a false positive, and $\hat{P}(+)P(graph|+)$ is the likelihood calculation of the graph being positive given the set of classifiers. Furthermore, assuming the classifiers are independent, the final likelihood calculations are:

$$\hat{\lambda}(+|Classifiers, graph) = \prod_i (\hat{P}(+|class_i)\hat{P}(class_i|graph)) \tag{5.15}$$

$$\hat{\lambda}(-|Classifiers, graph) = \prod_i (\hat{P}(-|class_i)\,\hat{P}(class_i|\ graph)) \qquad (5.16)$$

in which each classifier output is weighted with the probability of producing a true positive or a false negative, both values of which are available using information in the evaluation stage during learning.

However, there are two fundamental problems with this approach. The first problem is that it assumes the data being classified has unimodal characteristics; that is, there is a single underlying best description of the data. When the data is actually multi-modal, the classifiers will tend to cancel one another out, with each classifier classifying as negative the positive results of other classifiers.

The second fundamental problem is that the data for the second classifier is a subset of the data used to train the first classifier, and it is unknown how the classifier would actually perform on the original training set. Furthermore, it is best for the classifiers to be as trained as similarly as possible to its actual usage to verify its accuracy. Removing training data that the previous classifiers get correct is not reproducible against actual data, because you cannot tell which data the first classifier is getting correct.

These two problems necessitate creating an approximation method to that above. Instead of clustering all the classifiers into a single node, each classifier will become one of the nodes connected together with confidence thresholds as shown in Figure 5.8.

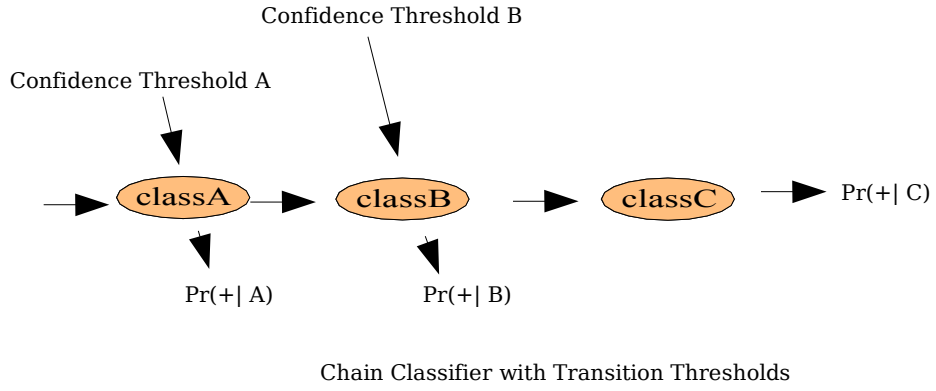


Figure 5.8. Classifiers chained using confidence thresholds.

Each of the sub-classifiers then becomes good at solving problems passed on from previous classifiers. For instance, if the first classifier is equivalent to a parts clerk looking for your order, then he will be able to handle most of your requests; when he is not sure enough of his job, he asks the shift supervisor, and so on up the chain.

At all stages in the classifier path except the last, the classifier's probability is compared against a threshold to determine if the answer is "good enough." This *confidence threshold* defines how confident the classifier is in the probability. We calculate this value by scaling the probability's distance from the uniform probability of classification over the total possible distance, such that for a binary classifier, 0.5 becomes 0% confidence, and 0 and 1 become 100% confidence as follows:

$$\hat{Confidence}: P(+|graph) \rightarrow \frac{|P(+|graph) - 0.5|}{0.5} \tag{5.17}$$

If the classification probability is strong enough, the process stops and the highest likelihood class is returned. If, on the other hand, the probability is not strong

enough, the data is then sent to the next classifier in the chain. The last classifier in the chain has no threshold, and simply outputs its classification probability.

Essentially the “super-classifier” we are learning is a disjunction of substructures, each with an embedded confidence threshold that determines whether it can trust the probability generated for the classification. You can picture the result as a tree, where each node in the tree is a classifier that can output a “yes”, “no”, or "ask the next guy" link.

In order to learn the next classifier in a chain during training, the confidence threshold is calculated for the last classifier, and all training examples that fall above that threshold are removed from the set. This threshold is optimized to maximize the number of answers that are confidently answered. The approximation used here is based on the threshold itself:

$$conf_{Thresh} = \underset{conf_i}{argmax} \frac{(conf_i \times |samples\ classified\ above\ confidence|)}{100} \qquad (5.18)$$

This method overcomes the weaknesses of the last method, because the classifiers are now being trained as they are expected to be used, and once a classifier generates a strong enough likelihood as either positive or negative, it will not be overridden.

As an example, suppose we have 10 training examples of which the first 4 are labeled negative and the remaining 6 are labeled positive. Furthermore, on the first iteration the best classifier returns probabilities of {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7. 0.8, 0.9, 1.0}. The confidences become {0.8, 0.6, 0.4, 0.2, 0, 0.2, 0.4, 0.6, 0.8, 1} with unique confidences of {1, 0.8. 0.6, 0.4, 0.2, 0}. Each confidence is then multiplied by the percentage of occurrence at or above the value to yield {0.1, 0.16, 0.24, 0.24, 0.16,

0.1}. The confidence value just before the dropoff, 0.4, is selected as the threshold for Classifier A and the classifier is recorded as in Figure 5.7. The training samples classified at or above the confidence threshold of 0.4 are now pruned from the data set, so we remove examples 1, 2, 3, 7, 8, 9, and 10. The new training set now consists of 1 negative example from the original set and 2 positive examples, ready to find the next classifier, Classifier B, on the next iteration.

We implemented this classification framework in SUBDUE with the NBGC heuristic, and tested it against both synthetic databases, where the quality of the data is controlled, and the mutagenesis data set, a collection of large graphs representing molecules classified as to whether they have been known to mutate and become cancerous. The results of these experiments of are detailed in the next chapter.

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Synthetic Results

6.1.1 The House Dataset

The viability of the NBGC algorithm was checked by running the Naïve Bayesian Graph Classifier algorithm against several simple benchmark data sets previously developed for evaluating SUBDUE [Cook and Holder, 2000] learning algorithms and watching the results as data was added. The house data sets are described in Table 6.1. In all the data sets the target concept to learn is that a house is a triangle on a square as shown in Figure 6.1.

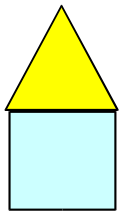


Figure 6.1. Target concept, a house is a triangle on a square.

To test supervised learning, the training sets house1.g through house3.g have both positive and negative examples. The smallest set is house1.g with 3 positive and 3 negative examples. More positive and negative examples are incrementally added in house2.g and house3.g.

Table 6.1 Description of sample data sets used.

Data Sample	Description	Positive Examples	Negative Examples
house1.g	House domain where a house consists of a triangle on a square. Separate examples for each house. Uniform prior probability for vertexes.	3	3
house2.g	One more positive and negative example added to house1.g	4	4
house3.g	One more positive example added to house2.g	5	4

6.1.1.1 Initial Discoveries

The three positive and negative examples in the house datasets are shown in Figure 6.2, and a sample of the data representation is shown in Figure 6.3. SUBDUE's minimum substructure size was set to 4 vertices and the limit on the number of substructures to evaluate was set to 1000 to get past non-discriminating structures in the data. The maximum feature size was set to 5. The resulting best substructures discovered by SUBDUE using NBGC are listed in Tables 6.2 and 6.3. SUBDUE discovered that the triangle on square substructure had the best discriminating features in the house1 dataset, but was still unable to build a classifier that got any samples correct. This is because SUBDUE starts with single vertex instances and expands the best substructures found until the no better children can be found (see Section 3.4). NBGC can only improve the score if there are more instances of a substructure in either the positive or the negative dataset, not if the instances are uniform (see Chapter 5). The smallest distinguishing substructure in house1 is 4 vertices and 3 edges large , so

SUBDUE never gives NBGC a chance to evaluate it. However, as more data samples were added in house2, NBGC in SUBDUE was able to discriminate 60% of the positive examples and 50% of the negative examples. In this case, it selects the triangle on square on rectangle substructure because the rectangle is the only 1 vertex substructure that differentiates between positive and negative examples, and occurs in at least one positive example (SUBDUE draws candidate graphs from the positive examples). Finally, with the addition of another positive example in house3, 80% of the positive examples were classified in addition to 50% of the negative examples.

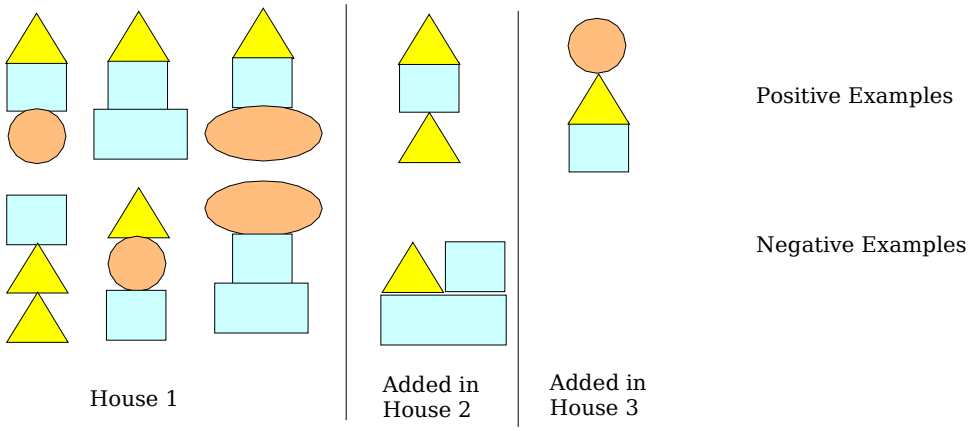


Figure 6.2. Examples in the house datasets.

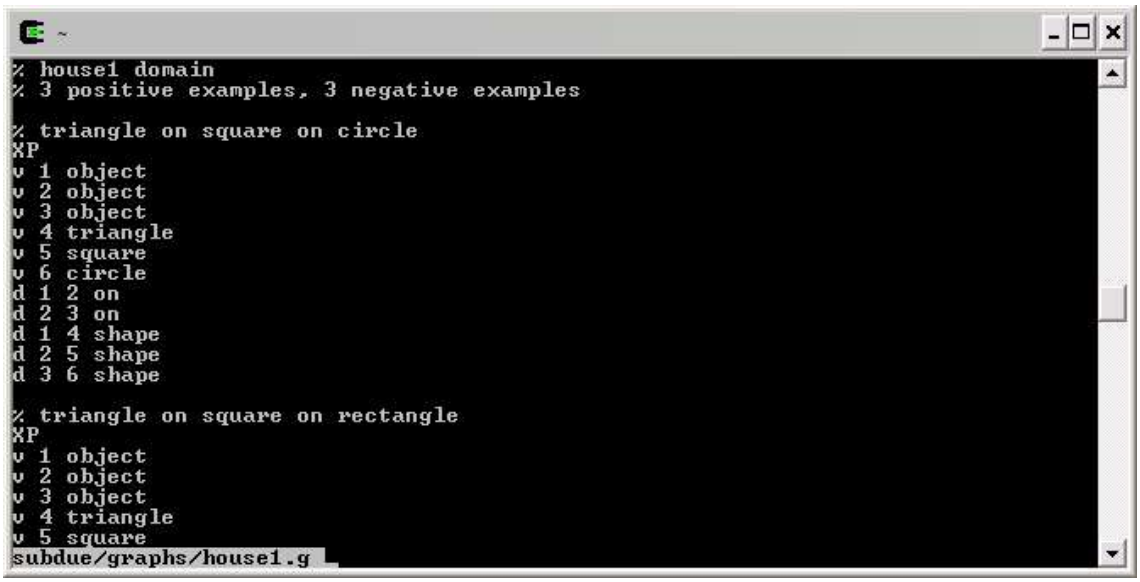


Figure 6.3. Training sample representation in a house dataset.

Table 6.2 Best substructures found by data set

Dataset	Best substructure 1	Best substructure 2	Best substructure 3
house1	triangle on square	object on square on object	triangle on object
house2	triangle on square on rectangle	triangle on object on rectangle	object on square on rectangle
house3	object on square on rectangle	triangle on square on rectangle	object on object on rectangle

Table 6.3 Evaluation of best classifier found by dataset

Dataset	Value	+ Classified Correctly	- Classified Correctly
house1	0.75	0	0
house2	0.77	3	2
house3	0.77	4	2

6.1.1.2 Initial Classifier Features

The breakdown of the classifier features of the best classifier learned by SUBDUE on the examples of the house3 dataset are shown in Figure 6.4. As shown in Table 6.4, the relative likelihoods captured by the "rectangle" feature emphasize

substructures that are stronger in the negative examples, whereas, the feature “object on object on object” shows a slight positive preference.

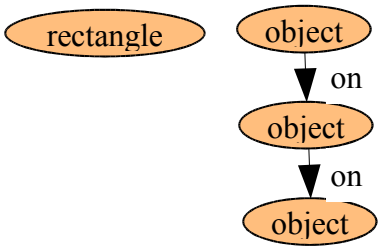


Figure 6.4. Classifier features in house3 best substructure, 1 iteration, 6 edge maximum for features.

Table 6.4 Relative likelihood of features of best classifier found in house3 dataset.

Feature	P(Feature | +)	P(Feature | -)	Relative Likelihood (+)
object on object on object	0.0500	0.0435	53.5%
rectangle	0.0167	0.0326	33.9%

Although it may seem surprising that SUBDUE with NBGC did not discover the "object shape house on object shape square" pattern as one of the classifying features, this is, in reality, due to the makeup of the training data, a bias inherent in NBGC, and a bias in the SUBDUE algorithm. NBGC's bias is to learn features that "explain away"; that is, it adds features more prevalent in other classifications. One of SUBDUE's biases is to only consider positive examples for candidate substructures. As a consequence, SUBDUE with NBGC first finds features in "positive" examples that are more prevalent in "negative" examples. With the house3 dataset (see Figure 6.2), there

are more positive feature instances than the negative feature instances. This results in a higher a priori probability for positive instances than for negative instances. Therefore, with no features added, NBGC already can correctly predict the majority of example cases by defaulting to the a priori value. Consequently, SUBDUE with NBGC discovers in the positive examples the rectangle feature, which occurs more frequently in the negative examples, and the "object on object on object" feature which occurs slightly more often in the positive examples.

The next test was to evaluate the impact of the maximum number of edges to use for a classifier attribute by adjusting the maximum number of edges between trial runs on the house3.g dataset and observing the resulting scores of the best three classifiers. The results are list in Table 6.5. The best substructures found remained the same despite the maximum feature size. However, the classification probability accuracy increased steadily through feature size three as marked the increase in score which represents lower error classification probability (see Chapter 5). Feature sizes four through six resulted in the same classifiers discovered using feature size 3.

Table 6.5 Impact of the maximum number of edges in classifier features for house3 dataset.

Maximum Edges	Best Substructures Found	Value	+ Classified Correctly	- Classified Correctly
0	triangle on object on rectangle	0.768	4	2
	object on square on rectangle	0.768	4	2
	triangle on square on rectangle	0.768	4	2
1	triangle on object on rectangle	0.769	4	2
	object on square on rectangle	0.769	4	2
	triangle on square on rectangle	0.769	4	2
2	triangle on object on rectangle	0.773	4	2
	object on square on rectangle	0.773	4	2
	triangle on square on rectangle	0.773	4	2
3-7	object on square on rectangle	0.773	4	2
	triangle on square on rectangle	0.773	4	2
	triangle on object on rectangle	0.773	4	2

6.1.1.3 Learning Multiple Classifiers

To evaluate if SUBDUE with NBGC could learn the intuitive best result, the algorithm was allowed to continue learning a new classifier for each iteration on the house3 dataset until no more classifiers could be learned. The results are shown in Table 6.6. Two iterations were enough to generate classifiers that exhaust the sample space. The classifiers from the first iteration correctly classify 67% of the 9 examples in the training set. By removing the covered examples using the confidence threshold, it correctly classified all the training examples removed. The remaining 3 samples are

passed on to one of the classifiers learned in iteration 2, which classified at 67% accuracy on the training set but removed only the examples it correctly classified. Finally, the last classifier correctly classified the remaining positive examples. The net result for the combination is therefore 9 out of 9, or 100%. This result represents a distinct improvement over each of the the first two classifiers alone.

The combination of the bias toward smaller substructure sizes from SUBDUE and the complexity of the target feature (4 vertices and 3 edges, see Section 6.1.1.3) resulted in SUBDUE with NBGC overfitting the discovered features to the dataset. For example, the first classifier selected as features "object on object on object" and "rectangle," neither of which is the target feature "object shape triangle on object shape house."

6.1.2 The Church Dataset

The church data is a synthetic dataset generated to test the performance of SUBDUE with NBGC over a large number of smaller graphs with fixed uncertainty. 1000 graphs were generated using an artificial graph generator available at the UTA SUBDUE site(http://cygnus.uta.edu/subdue/). Each sample was generated using a core concept of a church being a triangle on a square beside a rectangle, as shown in Figure 6.5, and nested in a larger graph with a mutation rate of 10%.

Table 6.6 Classifiers discovered in house3 dataset.

Iteration	*Best Substructures*	*Score*	*Pos Correct*	*Neg Correct*	*Conf Thresh*	*Pos Covered*	*Neg Covered*
1	triangle on object on rectangle	0.773	4	2	0.199	4	2
	object on square on rectangle	0.773	4	2	0.199	4	2
	triangle on square on rectangle	0.773	4	2	0.199	4	2
2	triangle on square on object	0.807	0	2	0.683	0	2
	triangle on object on rectangle	0.806	0	2	0.681	0	2
	triangle on square on rectangle	0.806	0	2	0.680	0	2
3	triangle on square on rectangle	1.000	1	0	0.964	1	0
	triangle on square on object	1.000	1	0	0.959	1	0
	object on square on rectangle	1.000	1	0	0.959	1	0

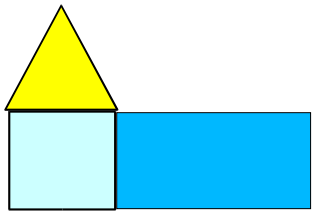


Figure 6.5. Church concept in church dataset, triangle on a square beside a rectangle.

In the first test, SUBDUE with NBGC was run over different sample sizes from size 10 to size 500 selected in order from the generate church dataset. The last 500 samples of the church dataset were set aside for validation. The algorithm was run in Cygwin in Windows XP on a Toshiba Satellite with a 2 Ghz Intel® Celeron® processor and 240 MB of RAM. The maximum feature size was set to six and the minimum graph size set to five. The number of iterations was specified as unlimited. Figures 6.6 and 6.7 show the error rates and training times respectively. In sample sizes under 80, the classifier had better accuracy on the training data than the validation data. For larger sample sizes, the accuracy on the unseen validation data was better than the accuracy on the test set, indicating that the classifiers learned generalized well to unseen data. The validation accuracy was also surprising, because the classifier achieved 95% accuracy after training on only 80 samples and 97.5% accuracy after training on 200 samples.

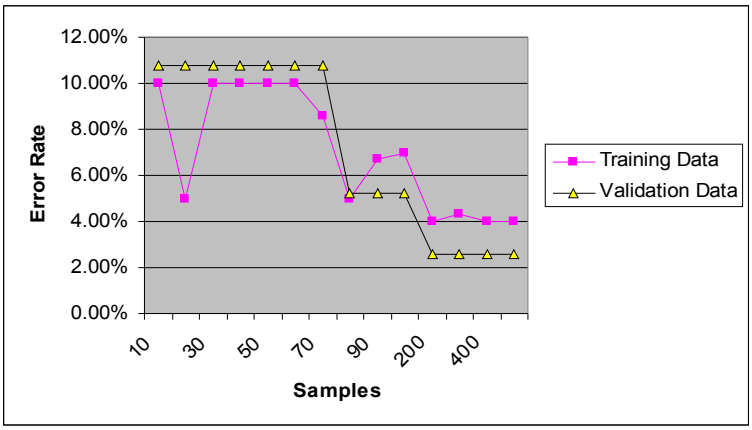


Figure 6.6. Accuracy of SUBDUE NBGC on church dataset.

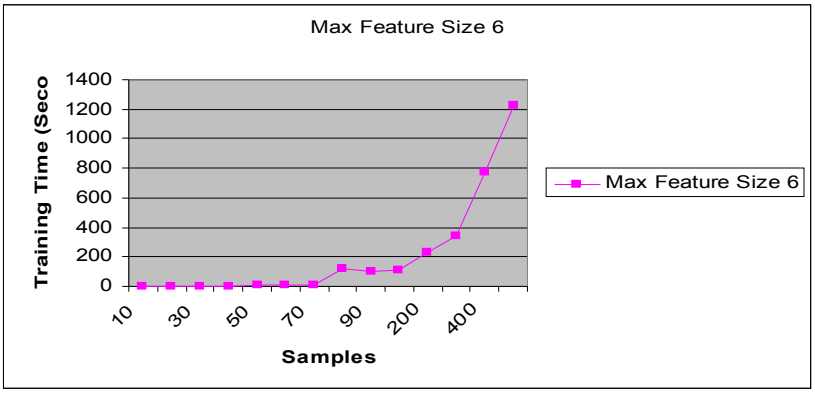


Figure 6.7. Training time of SUBDUE NBGC on church dataset.

In the second test, we fixed the sample size to 200 and simulated noise in the data set with varying degrees of mutation. Each sample was an instance of the target concept. Each sample in the set had a probability equal to the noise rate of undergoing mutation. Each mutation had an equal probability of either adding an edge, adding a vertex, removing an edge, or removing a vertex. All the sample sets were tested using

the validation samples from the previous experiment. The results are displayed in Figure 6.8.

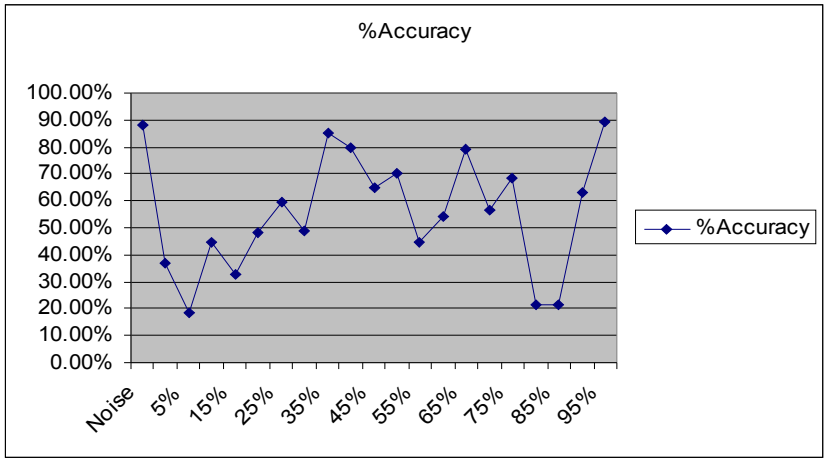


Figure 6.8. Effect of Noise on SUBDUE NBGC accuracy.

As can be seen in the figure, SUBDUE with NBGC tended to learn better classifiers when there was more noise and fewer positive training examples, with the exception of a dip in accuracy when the noise level was around 90%.

6.2 Mutagenesis Dataset

The purpose of the mutagenesis dataset test is to position the performance of SUBDUE with NBGC as compared to the performance of other classifiers on the mutagenesis domain. The mutagenesis dataset (http://ranger.uta.edu/mgd/mutagenesis-f.pl) is a collection of 188 compounds, each classified as to whether or not the compound is mutagenic (that is, can mutate and become cancerous). The data structure of each consists of a set of four continuous indicators (ind1, inda, lumo, and logp) and a

molecular structure. Each molecule consists of an average of 26 atoms per molecule, each of which has properties including element, type, and charge. In addition, there are an average of 28 bonds per molecule linking the atoms together in pairs, and each bond possesses a bond strength.

For this set of experiments, the prolog data format was converted to a graph format suitable for SUBDUE. A molecule vertex was constructed for each example. The atom, molecule, and bond properties were represented as labeled edges of the property names linked to vertices labeled with the property values. Atom and bond vertices were attached to the molecule vertices using edges named molecule_atom and molecule_bond respectively. Similarly, bond_atom edges were added to represent the atoms participating in a bond.

The large vertex degree (number of edges going in and out) of the molecule nodes in this format prevented SUBDUE from completing the mining in an efficient manner necessitating the following changes from the original data structure and parameters to enable SUBDUE to complete in a reasonable amount of time:

1. The maximum feature size had to be dropped from six to three.
2. The graphs were denormalized from having a single molecule vertex to having a molecule vertex for every atom and bond.
3. SUBDUE's limit on the number of candidate graphs to evaluate was set to 200.
4. SUBDUE's pruning parameter was set, so that child substructures of unpromising substructures were not considered.

The experiments were performed in Slackware® Linux on an E-machines® T6522 with a 2.2 Ghz AMD Athlon® processor, 512 KB L2 cache, and 1 GB of memory.

6.2.1 Linear Sampling

In this experiment, SUBDUE with NBGC was trained on the first *n* samples from the dataset and validated on the last 94 samples in the dataset. *n* was set incrementally in values ranging from 5 to 65 samples. The training was done using a maximum feature size of two and a maximum feature size of 3. The accuracy results are plotted in Figure 6.9, and the running times are plotted in Figure 6.10. The accuracy peaked at close to 90% on the first 10 samples for both feature set sizes, but declined soon afterward due to the clustering of samples in the data. There was not much difference in the accuracy between a maximum feature size two or three, even though the maximum feature size of three used a significant amount of more time as shown in Figure 6.10. Consequently, the remaining experiment using random resampling was done using a maximum feature size of two.

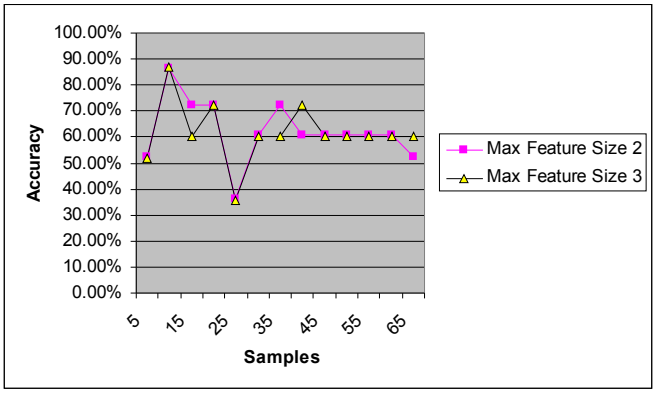


Figure 6.9. Accuracy of SUBDUE NBGC on mutagenesis dataset using linear sampling.

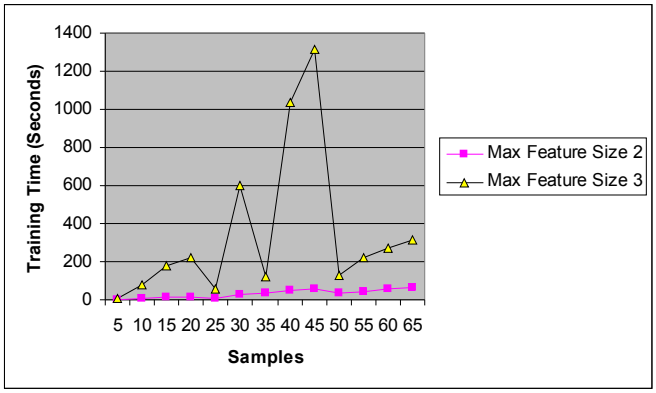


Figure 6.10. Training time of SUBDUE NBGC on mutagenesis dataset by sample size.

6.2.2 Random Sampling without Replacement

Because the data samples showed a high degree of clustering and low uniformity during linear sampling, an experiment was done to see if using random sampling without replacement would allow for the training curve to be observed.

For this round of tests, the validation set for each training iteration was generated by drawing 90 samples from the sample set and setting them aside. The samples for training were generated by randomly drawing without replacement from the remaining 97 original samples. This procedure was repeated 30 times for training sets ranging in size from 5 to 30 samples. SUBDUE was then trained using NBGC on each training set and the resulting classifier was validated against its corresponding 90 unseen training examples. Those results were then batched by training sample size to generate the performance curves shown in Figures 6.11 and 6.12. Due to the size of the

training instances, SUBDUE was limited to evaluating 100 graphs per iteration on the first run, and 200 on the second run.

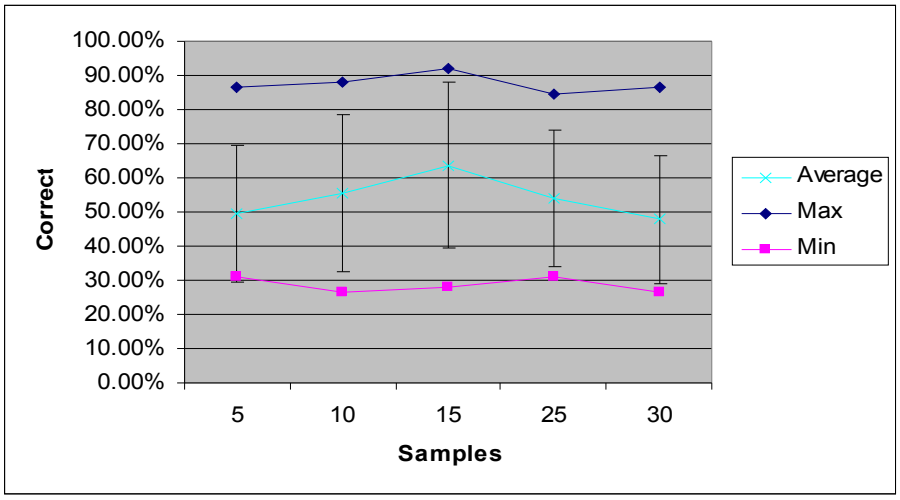


Figure 6.11. Resampling performance with limit set to 100.

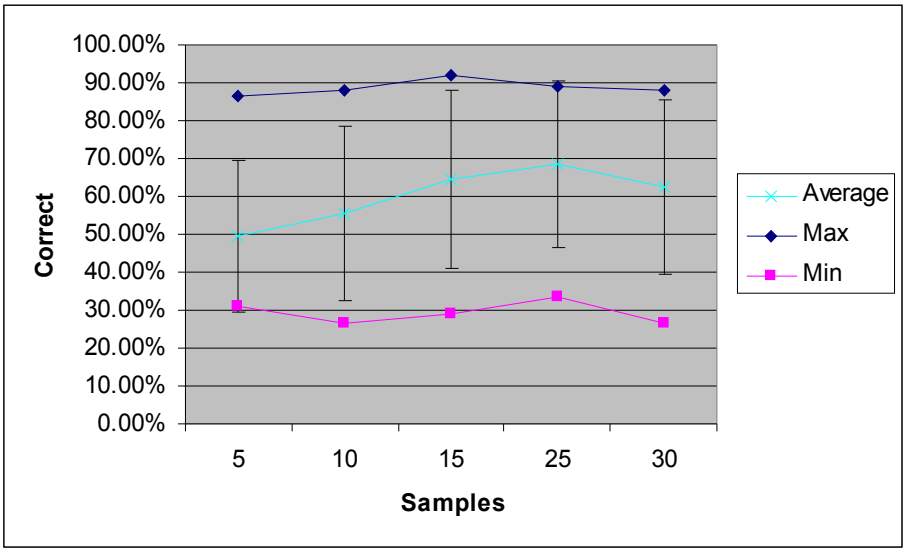


Figure 6.12. Resampling performance with Limit set to 200.

As can be seen in Figures 6.11 and 6.12, SUBDUE with NBGC and random sampling produces a positive learning curve for the average classifier. With the limit of substructures to evaluate set to 100 the curve peaks at about 62% with sample sizes of size 15, and declines steadily afterward. Increasing the limit to 200 results in a peak average of 70% at 25 samples, before starting to decline. The best classifier in both cases scores around 91% on the 90 unseen examples using only 15 training examples.

SUBDUE with NBGC compares favorably with other graph-based classifiers trained on the mutagenesis data set [Karwath and De Raedt, 2004] [Ketkar et. al, 2005] [Bowers et. al, 2000], as shown in Table 6.7. NBGC boosted SUBDUE's performance by adding the capability to discover features that classify negative examples as well as positive examples. However, it still does not perform as well as state-of-the-art ILP systems for structural data [Lodhi and Muggleton, 2005] [Sebag, 1997].

Table 6.7 Mutagenesis performance of classifiers

Classifier	*Performance*	*Data Type*	*Training and Validation*
SUBDUE [Ketkar et. al, 2005]	***82%***	***Graph***	***10-fold cross-validation***
SMIREP [Karwath and De Raedt, 2004]	*85%*	*Graph*	*90% training, 10% validation*
Naïve Bayes + ILP [Lodhi and Muggleton, 2005]	85%	Structured	10-fold cross-validation
Complex Structure Decision Trees [Bowers et. al, 2000]	*87%*	*Graph*	*10-fold cross-validation*
SUBDUE NBGC	***91%***	***Graph***	***Random Sampling*** ***Best classifier validated on 90 random samples***
ILP STILL [Lodhi and Muggleton, 2005]	94%	Structured	10-fold cross-validation
PPILP MFLOG [Lodhi and Muggleton, 2005]	96%	Structured	10-fold cross-validation
EMILP RS [Lodhi and Muggleton, 2005]	96%	Structured	10-fold cross-validation
DISTILL [Sebag, 1997]	97%	Structured	10-fold cross-validation

CHAPTER 7

CONCLUSION AND FUTURE WORK

A graph-based learning algorithm built upon probability theory can form the basis of an accurate predictor and useful learning algorithm. In particular, an algorithm that learns graph concepts using a naïve Bayes method can achieve comparable or better performance than other learning algorithms on structural data.

The Naïve Bayesian Graph Classifier, NBGC, is an adaption of the traditional naïve Bayes text-based classifier adapted to handle graph-based data. Specifically, the definition of attributes as words was adapted to attributes as substructures, or graph-based features, within a graph. We also defined a method for calculating the conditional probabilities of substructure-based attributes for the naïve Bayes learning method.

NBGC was tested by embedding it as an alternative evaluation method in the SUBDUE algorithm [Cook and Holder, 2000], where we further enhanced the classifiers by learning a super-classifier as a sequence of such classifiers. The classification system was then tested against two sets of synthetic data and the mutagenesis data set. In the synthetic house dataset, we demonstrated that SUBDUE with NBGC has a built in bias to find substructures in the positive examples that "explain away" from the a priori prevalent classification. In the synthetic church dataset, SUBDUE with NBGC was able to accurately classify 97.5% of unseen examples, and was shown to be relatively immune to noise with a few exceptions.

Finally, SUBDUE with NBGC outperformed several recent classifiers built for graph-based data, while still underperforming state-of-the-art ILP systems.

There are a number of things that we would still like to do with NBGC. First and foremost, we would like to explore different methods for calculating the best feature set and dynamically determining the best feature size. The house example demonstrated that when there is sufficient uniformity in smaller substructures, the growth of candidate features from single positive vertices can fail to identify differentiating features. Also, the algorithm to find the best feature set significantly increases SUBDUE's runtime as the number of candidate features increases.

There are two related approaches for performing supervised graph-based learning. [Deshpande et al., 2003] have developed a method that first finds frequent topological and geometric subgraphs, encodes them as feature vectors, and then applies SVM learning to learn a classifier. [Gärtner, 2005] is developing kernel methods for graphs that return a positive real value which can be used in any classifier system developed for structural data that makes use of kernel functions.

There are some natural extensions that can be performed on the NBGC algorithm. One extension would be to adapt SUBDUE with NBGC to be used as a utilitarian classifier by providing features to use or ignore as separate input. Another natural extension is to enable SUBDUE and NBGC to discriminate multiple labeled classes simultaneously, instead of just positive and negative examples. This extension could in turn lead to work exploring different methods for automatically building the "super-classifier" networks, possibly through the use of a more informed confidence threshold function, or different classifier architectures

To conclude, this first implementation of NBGC in SUBDUE resulted in a surprisingly accurate classifier and useful learning algorithm built on probability theory, NBGC demonstrates that the use of the naïve Bayes method to classify graphs can achieve a classification accuracy on graph-based data comparable to within 10% of the current learning algorithms for structured data.

REFERENCES

Aery, M., and Chakravarthy S. (2004). eMailSift: mining-based approaches to email classification. Published in *Proceedings of the 27th Annual International Conference on Research and Development in Information Retrieval, July 25-29.* Sheffield, United Kingdom.

Bowers, A., Giraud-Carrier, C., and Lloyd, J. (2000). Classification of individuals with complex structure. Published in *Proceedings of the 17th International Conference on Machine Learning, pages 81-88*.

Coble, J. (2005). Sequential structure discovery in graph-based data. *PhD dissertation, University of Texas at Arlington, Tx.*

Cook, D., and Holder, L. (2000). Graph-based data mining. Published in *IEEE Intelligent Systems, Volume 15, Number 2.*

Cook, D., Holder, L., Su, S., and Maglothin, R. (2001). Structural mining of molecular biology data. Published in *IEEE Engineering in Medicine and Biology Special Issue on Advances in Genomics 20(4), pages 67-74.*

Cook, S. (1971). The complexity of theorem-proving procedures. Published in *Conference Record of 3rd Annual ACM Symposium on Theory of Computing, pages 151-158.*

De Raedt, L., and Kramer, S. (2001). The levelwise version space algorithm and its application to molecular fragment finding. Published in *Proceedings of the 17th International Joint Conference on Artificial Intelligence, volume 2, pages 853-859.*

Dehaspe, L., and Tolvonen, H. (1999). Discovery of frequent datalog patterns. Published in *Data Mining and Knowledge Discovery, volume 3, pages 7-36.*

Deshpande, M., Kuramochi, M., and Karypis, G. (2003). Frequent sub-structure-based approaches for classifying chemical compounds. *Technical Report 03-016, University of Minnesota* .

Dumais, S., and Chen, H. (2000). Hierarchical classification of web content. Published in *Proceedings of 23rd ACM International Conference on Research and Development in Information Retrieval.*

Elkan C. (1997). Boosting and Naive Bayesian Learning. *Technical report, Department of Computer Science and Engineering, University of California, San Diego*.

Friedman, N., Geiger, D., and Godszmidt, M. (1997). Bayesian network classifiers. Published in *Machine Learning Journal, volume 29, pages 131-163.*

Gärtner, T. (2005). Kernel methods for graphs. Published in *Proceedings of the 4th International Workshop on Multi-relational Mining, page 1.*

Gonzalez, J., Holder, L., and Cook, D. (2000). Graph based concept learning. Published in *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence, page 1072*. AAAI Press.

Inokuchi, A., Washio, T., and Motoda, H. (2003). Complete mining of frequent patterns from graphs: mining graph data. Published in *Machine Learning, volume 50, pages 321-354.*

Joachims, T. (1996). *A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization.* (Computer Science Technical Report CMU-CS-96-118). Carnegie Mellon University.

Karwath, A., and De Raedt, L. (2004). Predictive graph mining. Published in *Proceedings of Discovery Science 2004, Padova, Italy, pages 1-15.*

Ketkar, N., Holder, L., and Cook, D. (2005). Comparison of graph-based and logic-based MRDM. Published in the *Proceedings of the 11th ACM Special Interest Group on Knowledge Discovery and Data Mining.*

Kuramochi, M., and Karypis, G (2001). Frequent subgraph discovery. Published in *Proceedings of the 1st International Conference on Data Mining, pages 313-320.*

Kuramochi, M., and Karypis, G (2004). Finding frequent patterns in a large sparse graph. Published in *Proceedings of the 2004 SIAM Data Mining Conference*.

Lang, K, (1995). Newsweeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning, pages 331-339.* San Francisco: Morgan-Kaufmann Publishers.

Lewis, D. (1991). *Representation and learning in information retrieval,* (Ph.D. Thesis), (COINS Technical Report 91-93). Department of Computer and Information Science, University of Massachusetts.

Lodhi, H., and Muggleton, S. (2005). Is Mutagenesis still challenging? Publishe d in *Proceedings of the 15th International Conference on Inductive Logic Programming.*

Mendelzon, A., Mihaila, G., and Milo, T. (1997). Querying the world wide web. Published in *International Journal on Digital Libraries, volume 1, number 1, pages 54-67.*

Mitchell, T. (1997). *Machine Learning*. McGrawHill.

Murphy, K. (1998). A brief introduction to graphical models and Bayesian networks. *http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html.*

Nijssen, S., and Kok, J. (2001). Faster association rules for multiple relations. Published in *Proceedings of the 17th International Joint Conference on Artificial Intelligence, volume 2, pages 891-896.*

Noble, C., and Cook, D. (2003). Graph-based anomaly detection. Published in *Proceedings of the 9th ACM Special Interest Group in Knowledge Discovery in Databases International Conference on Knowledge Discovery and Data Mining, pages 631-636.*

Rassinoux, A., Baud, R., Lovis, C., Wagner, J., Scherrer, J. (1998). Tuning up conceptual graph representation for multilingual natural processing in medicine. Published in *Proceedings of the 6th International Conference on Conceptual Structures: Theory, Tools and Applications, pages 390-400.* London: Springer-Verlag.

Sebag, M. (1997). Distance induction in first order logic. Published in *Proceedings of the 7th International Conference on Inductive Logic Programming, pages 264-272.*

Vanetik, N., and Gudes, E. (2004). Mining frequent labeled and partially labeled graph patterns. Published in *Proceedings of the 20th International Conference on Data Engineering, pages 91-102*.

Vanetik, N., Gudes, E. and Shimony, S.(2002). Computing frequent graph patterns from semi-structured data. Published in *Proceedings of the 2nd International Conference on Data Mining, pages 458-465*.

Washio, T., and Motoda, H. (2003). State of the art of graph-based data mining. Published in *Special Interest Group on Knowledge Discovery in Databases Explorations, July 2003, pages 59-68.*

Yoshida, K., Motoda, H., and Indurkhya, N. (1994). Graph-based induction as a unified learning framework. Published in *Journal of Applied Intelligence, volume 4, pages 297-328.*

Xiao, R., Zhu, L., and Zhang, H.J. (2003). Boosting chain learning for object detection. Published in *Proceedings of the 9th International Conference on Computer Vision, pages 709-715.*

Zheng, Z. (1998). Naive Bayesian classifier committees. Published in *Proceedings of the 10th European Conference on Machine Learning, pages 196-207.* Berlin: Springer-Verlag.

Zheng, Z., Webb, G. I., and Ting, K. M. (1999). A lazy seminaive Bayesian learning technique competitive to boosting decision trees. Published in *Proceedings of the 16th International Conference on Machine Learning, pages 493-502.* Morgan Kauffman.

BIOGRAPHICAL INFORMATION

Robert Hawes holds a B.S. in Computer Science and Engineering from the University of Texas at Arlington. His research interests include graph-based data mining, machine learning, and natural language processing, and intends to continue work toward a Ph.D. His senior design team's project, ADAM (Automated Drawing Arm Machine), earned top honors in 1997. In addition to serving for four years in the United States Navy from 1989 to 1993 as a Torpedoman's Mate on board the USS Lynde McCormick (DDG-8) and the USS Mahlon S. Tisdale (FFG-27), Robert has worked on EDMS for Medhost Inc., the ZixPolicyManager for Zix Corporation, and is currently working on Ferret and Question-Answering for Language Computer Corporation.