Sisterhood of Classifiers: A Comparative Study of Naive Bayes and Noisy-or Networks

David Chen[*]
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095

Abstract

Classification is a task central to many machine learning problems. In this paper we examine two Bayesian network classifiers, the naive Bayes and the noisy-or models. They are of particular interest because of their simple structures. We compare them on two dimensions: expressive power and ability to learn.

As it turns out, naive Bayes, noisy-or, and logistic regression classifiers all have equivalent expressiveness. We show mathematical derivations of how to transform a classifer in one model into the other two.

These classifiers differ on their ability to learn though. We conducted an experiment confirming the intuition that naive Bayes performs better than noisy-or when the data fits its independence assumptions, and vice versa. However, we still do not have a clear set of criteria for determining under exactly what conditions would each classifier excel.

Further study of the strenghts and weaknesses of each classifier should provide deeper insight on how to improve the current models. One possible extension would be to combine the naive Bayes and noisy-or model so that the network will more closely depict the actual relationship between the attributes.

1 Introduction

Classification has applications in many different fields including biology [11], information retrieval [23, 24, 32], national security [1], spam filtering [3], etc. It is a basic task in performing data analysis or pattern recognition. The main goal of classification is to construct a function that will correctly assign instances of events or objects to their respective *classes*. Each instance is described by a number of *attributes*, which can have discrete or continuous values. Building such functions, or classifiers, automatically from labeled datasets is central to

[*]With special thanks to the assistance of Dr. Adnan Darwiche

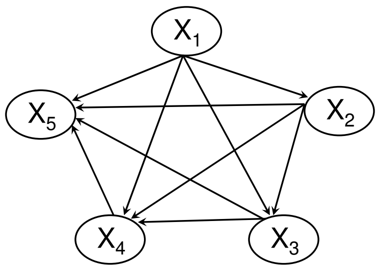

Figure 1: Fully connected Bayesian network

machine learning. There have been many different approaches to the problem including using decision trees, decision graphs, decision lists, neural networks, support vector machines, etc.

A particular type of classifier that has been gaining popularity among computer scientists in the last 20 years is called the *Bayesian network classifier*. As the name implies, these classifiers all use *Bayesian networks* to represent the relationships between the attributes and the class label. The term *Bayesian networks* was first coined by Pearl in 1985 [29]. It refers to a type of model that uses *directed acyclic graphs (DAGs)* and *conditional probability tables (CPTs)* to represent causal or temporal relationships between the nodes of the graphs. The graph represents the qualitative component of the network whereas the tables represent the quantitative component of the network. Bayesian networks find their origins in Thomas Bayes's paper of 1763 where he introduced the first definition of conditional probability, the Bayes ratio formula. However, it was not until 1980's that Bayesian networks and classifiers start to appear as applications for machine learning.

Before we continue our discussion about Bayesian networks, we will first define some notations that will be used throughout this paper. All uppercase symbols (i.e., A, B, C) refer to single random variables. All boldface symbols (i.e., $\mathbf{X}$) refer to vectors of random variables. Lowercase symbols (i.e., a, b, c) and lowercase boldface symbols (i.e., $\mathbf{x}$) refer to values of single variables and vectors of random variables, respectively. We will often omit variable names, for example writing $Pr(a)$ instead of $Pr(A = a)$ for the probability of $A = a$. Since we will deal mostly with binary variables in this paper, we will use A or a to denote $A = 1$ or $A = true$ and $\neg A, \overline{A}$ or $\neg a, \overline{a}$ to denote $A = 0$ or $A = false$.

Bayesian network is powerful because it can represent exponentially-sized probability distributions compactly and we can perform inference on these probabilities without explicitly constructing them. Each node in the DAG represents a particular variable we are interested in, and each edge represents a direct influence between two nodes. The CPTs give numerical values to these direct influences. Perhaps counterintuitively though, it is the missing edges in the DAG that are the most important because they imply *conditional independences.*

Definition 1.1. *Given random sets of variables* $\mathbf{X}$*,* $\mathbf{Y}$*,* $\mathbf{Z}$*, we say* $\mathbf{X}$ *is conditionally independent of* $\mathbf{Y}$ *given* $\mathbf{Z}$*, if and only if*

$$Pr(\mathbf{X} \mid \mathbf{Y}, \mathbf{Z}) = Pr(\mathbf{X} \mid \mathbf{Z}) \textit{ whenever } Pr(\mathbf{Y}, \mathbf{Z}) \neq 0$$

In words, learning the value of $\mathbf{Y}$ *does not provide additional information about* $\mathbf{X}$*, once we know* $\mathbf{Z}$

Knowing these independences allow us to essentially ignore the variables that do not affect the probability distributions of the variables we are interested in. In general, Bayesian networks can model any situation. A trivial network for any problem would be a completely connected graph as shown in Figure 1. In such networks, no independences can be assumed and every variable can affect the values of all the other variables. As a result, we are usually interested in minimal networks where removing any edges in the networks would create more independences than specified by the problem.

Even though a well-built Bayesian network can give a very accurate model of a problem, they can be difficult to come by. Usually a mix of expert knowledge and large datasets are required to build such networks. The experts would either handcraft the network structure or at least give strong intuitions to what the network should look like. The computer will then learn the specific numerical parameters from the data. However, expert knowledge is not always readily available and handcrafting is infeasible for large networks with thousands of nodes. As a result, there has been a lot research toward learning Bayesian networks [4, 14, 21, 15, 17]. These learning algorithms are unsupervised, in the sense that the learner does not differentiate between the attributes and the class variables. The objective is to find the network structure that best fit the probability distribution of the training data. This is usually done by employing a heuristic search over the possible network spaces and finding the highest scoring network. However, many scoring functions do not accurately measure the goodness of the networks [12]. Moreover, it is proven recently that in general, identifying high-scoring structures is NP-hard [6]. Thus, we will concentrate our attentions to two particular types of Bayesian networks that have known structures: the *naive Bayes* and the *noisy-or* models, which are shown in Figures 2 and 3 respectively. Both models will be described in greater detail in Section 2.

The reason why these two models are particularly interesting is because they both have very simple structures but depict almost opposite situations. Whereas in the naive Bayes model all the attributes are effects of the class, in the noisy-or model, all the attributes are causes of the class. Of course, in most real-life scenarios, neither are correct since most likely some attributes would be causes while the others are effects. Despite their apparent opposite structures, they have equal expressive powers as classifiers, at least in the case where all

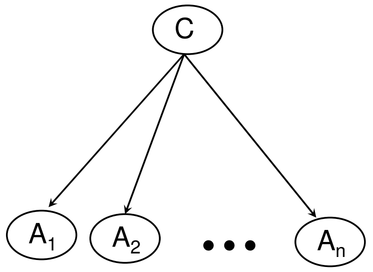

Figure 2: Naive Bayes network

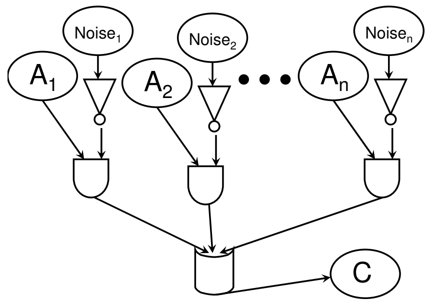

Figure 3: Noisy-or network

variables are binary. In other words, given a naive Bayes classifier, it can be transformed into an equivalent noisy-or classifier. For all possible combinations of attribute values, the two equivalent classifiers will assign exactly the same class labels to the same instances. Additionally, both of these classifiers have the same expressive power as the *logistical regression* classifier [26, 38]. Despite their equivalent expressiveness, the three classifiers differ in how well they can learn from data. Logistical regression usually hold a slight advantage because it is well studied in statistics and there exist many optimized learning methods for it. However, Ng and Jordan has shown that these classifiers converge toward their asymptotic accuracies at different rates [27]. Thus, it is possible that different classifiers are preferred for learning depending on the dataset size and the particular problem. This is promising since a careful study of the different learning algorithms of these three classifiers can provide insight to the strengths and weaknesses of each. Consequently they may be combined for an even stronger learning algorithm. Alternatively, we might be able to combine these classifiers to form something that more closely model the actual situation. For example, the structure in Figure 4 is a combined naive Bayes and noisy-or classifier.

Another interesting aspect of the comparison between the naive Bayes and noisy-or classifiers is that they form a *Generative-Discriminative* pair. A generative classifier learns estimates of $Pr(\mathbf{A} \mid C)$ and $Pr(C)$ where $\mathbf{A}$ is the attributes, and C is the class. New instances can then be classified using these estimated probability distributions and Bayes Rule. It is call a *generative* classifier because we can use the distribution $Pr(\mathbf{A} \mid C)$ to generate random new instances conditioned on the class. On the other hand, a discriminative classifier learns the desired value of $Pr(C \mid \mathbf{A})$ directly. It is called a *discriminative* classifier because the distribution $Pr(C \mid \mathbf{A})$ directly discriminates the value of the class for any given instances [27, 26]. There are compelling reasons to prefer the discriminative model as Vapnik stated that "one should solve the [classification] problem directly and never solve a more general problem as an intermediate

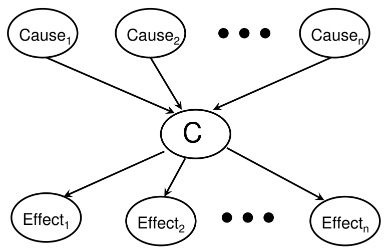

Figure 4: Combined naive Bayes and noisy-or classifier

step [such as modeling $p(\mathbf{A} \mid C)$]" [37]. However, as mentioned earlier, there are indications that no one classifier is superior in all cases.

The classification problem can be formulated formally as follows: Let $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ be the set of attributes describing the instances and C the class label. The goal is to build a classifier that will assign any new instances with the attributes $\mathbf{a}$ to class c such that $Pr(c \mid \mathbf{a})$ is maximized. In general, classifiers are trained on pre-classified instances, usually meaning someone hand labeled the class values for each instance. Due to the difficulty of collecting data and privacy issues though, oftentimes these records have missing data. Since in some cases a large number of instances will have missing values, it is not feasible to simply prune the incomplete records. Instead, learning algorithms have to find a way to deal with these missing attributes. After the classifiers are built, they are then tested on new instances to see if their classifications match the actual class of those instances. The performance of a classifier is usually based on its misclassification rate, or how often it assigns an incorrect class label to a class. This is important because it is the main reason why simple models such as naive Bayes and noisy-or networks can be competitive as classifiers even though they do not approximate the probability distributions of the data well in general [9, 34].

With sufficient background information and a clear statement of the problem, we will dive in full details of the classifiers (naive Bayes, noisy-or, and logistic regression) in section 2. We will follow that up with a mathematical derivations of the equivalent expressiveness of the classifiers in section 3. In section 4, we will discuss the different learning methods for each classifiers and how they compare with each other, including an experiment and the results. Section 5 will be a brief discussion about *Ordered Decision Diagrams (ODDs)* and how they figure in the picture of classifiers. Finally, we wrap everything up in section 6 with some discussions about future work.

2 Classifiers

2.1 Naive Bayes Network Classifier

The naive Bayes classifier is a simple, efficient, and compact classifier that is still competitive with state-of-the-art classifiers today. An early description of it can be found in Duda and Hart [10]. The first thing to notice about naive Bayes network is its simplicity. Unlike Bayesian networks in general, all of its nodes have at most one parent. This is crucial because the size of the CPTs grow exponentially with the number of parents. Thus, naive Bayes network requires a relatively very small number of parameters. More precisely, the number of parameters of a naive Bayes network is only on the order of the number of attributes. The compact size is important in many different ways. First, it is much easier to construct them from training data. In general, naive Bayes learning converges on the order of $\log n$ examples, n being the number of parameters [27]. Moreover, the small number of parameters allow for easy storage and fast inference. Finally, we can easily increment the model to include more attributes without having to learn all the parameters all over again.

Let us now discuss what exactly is a naive Bayes classifier. We already know it is a Bayesian network with the structure shown in Figure 2. The class variable is the root of the network while each attribute is a child of the class. Naive Bayes classifier makes a strong assumption about the relationships between the attributes. Namely, all the attributes are independent of each other given the class. In general, to classify an instance, we want to find the value c that will maximize $Pr(c \mid \mathbf{a})$. Since we are mainly dealing with binary variables in this paper, we will define the classifier accordingly. Given a naive Bayes network N and the threshold t, the output of the classifier is 1 if $Pr(C = 0 \mid \mathbf{A} = \mathbf{a}) < t$ and 0 otherwise.

Naive Bayes classifiers have been used in many different applications, such as separating housekeeping genes from tissue specific genes [11], filtering spam e-mails [3], analyzing Arabic text related to fanaticism [1], and probably the most prominently in text classification [23, 24, 32, 33]. Naive Bayes is so successful that Rennie called it the "de-facto standard text classifier" [32]. Despite its shortcomings stemming from its strong independence assumptions, naive Bayes is competitive with state-of-the-art classifiers such as C4.5 [12]. However, naive Bayes is not without its critics. The main criticism lies in the fact that naive Bayes assumes that the attributes are all independent of each other given the class. This is usually not true since most likely the attributes are related to each other. For example, consider a classifier for assessing the risk of issuing a credit card. It would be incorrect to assume that there are no correlations between age, income, and education level given that the applicant is classified as a worthy customer.

Nevertheless, naive Bayes performs quite well empirically. This was puzzling to researchers until Domingos and Pazzani shed some light on it in 1997. They found that “although the [naive Bayes's] probability estimates are only optimal under quadratic loss if the independence assumption holds, the classifier itself can be optimal under zero-one loss (misclassification rate) even when this assumption is violated by a wide margin” [9]. Thus, while naive Bayes is poor at answering queries about probabilities in general, it can excel at the single task of finding the most likely class value given the attributes. In fact, Domingos and Pazzani found that the naive Bayes is optimal for learning strongly dependent functions such as conjunctions and disjunctions [9]. This point is proved further when Rish found in 2001 that “naive Bayes works best in two cases: completely independent features (as expected) and functionally dependent features (which is surprising). Naive Bayes has its worst performance between the extremes” [34]. Thus, despite its simplicity, naive Bayes seems promising as a basis for building classifiers. Naive Bayes is usually the preferred classifier when the dataset is small and there are many attributes [9]. This is because naive Bayes converges to its asymptotic error quicker [27].

As a result of its empirical success, many researchers have tried to extend the naive Bayes model. Since the main problem with naive Bayes is that its independence assumption often do not occur in natural data, relaxing the independence assumption of naive Bayes ought to allow for superior text classification [23]. The idea is that after the naive Bayes is constructed, we will refine the model by adding edges between the attribute nodes to remove some of the independence assumptions. However, finding the best set of edges to add is an intractable problem, as it amounts to learning the best Bayesian network in which C is the root. Thus, Friedman et al proposed an extended naive Bayes model called the Tree Augmented Naive Bayes (TAN) in which the additional edges form a tree [12]. The algorithm picks the tree structure that maximizes conditional mutual information between the nodes of the added edges. Roughly speaking, conditional mutual information measures how much information one node provides about the other when the value of a third node is known. Thus, the goal is to remove the independence assumptions between nodes that are the most correlated. However, learning tree structured Bayesian network is not trivial [19, 40]. As a result, in 2004, Peng et al proposed a model in between called the Chain augmented Naive Bayes (CAN) [31]. Instead of finding a tree structure, the problem reduces down to the simpler one of finding a chain structure to augment the network.

While the most intuitive approach to improving the naive Bayes model is to remove attribute dependencies, it may not necessarily be the right approach [9]. As Rish found in 2001, “the accuracy of naive Bayes is not directly correlated with the degree of feature dependencies measure as the class-conditional mutual information between the features. Instead, a better predictor of naive Bayes accuracy is the amount of information about the class that is lost because of the independence assumption” [34]. Thus, perhaps connecting the most correlated nodes as proposed in the TAN and CAN models is not the most effective way of improving naive Bayes.

Other extensions to the naive Bayes model include processing the data so it aligns better with the independence assumptions [33]. Also, in 2003, Yang et al proposed the LinEar-Equation-based noise-aWare bAYes classifier (LEE-WAY) that constructs the naive Bayes classifier from noisy datasets [39]. This is significant because noise is inevitable in real measurements.

While naive Bayes makes strong assumptions that is rarely found in real data, its empirical success is undeniable. There are indications that these assumptions do not significantly affect the accuracy of the classifier. There are also numerous attempts to try to improve the naive Bayes framework so that the model can perform even better while maintaining its advantages. As Domingos and Pazzani summarizes, naive Bayes "has much broader applicability than previously thought. Since it also has advantage in terms of simplicity, learning speed, classification speed, storage space and incrementality, its use should perhaps be considered more often" [9].

2.2 Noisy-or Classifier

Noisy-or classifiers are another type of Bayesian network classifier that reduces the complexity of the network. For an example of a noisy-or network, see Figure 3. Introduced by Kim and Pearl in 1983 [20], the noisy-or gate has become widely used in many different fields, most notably in medical contexts [35, 25, 28]. However, unlike naive Bayes networks, there has been relatively little work on applying the noisy-or model to the classification problem. While naive Bayes is usually a complete network in and of itself, noisy-or gates are usually embedded in a larger Bayesian network. Despite their different usages, the two actually have similar characteristics. Noisy-or networks are compact, fast, and have small numbers of parameters that are linear in the numbers of nodes. Thus, it shares many of the same advantages the naive Bayes network has.

There have been a number of different formulations of the noisy-or model, but the central idea is the same. For an early discussion of the simplest and most intuitive canonical model, the binary noisy-or gate, see [20, 30]. We know that the size of the CPT of a node grows exponentially with the size of its parents. Thus, a node with many parents can be very costly to model. This is especially true in the noisy-or classifier where all the attributes are parents of the class variable. The CPT for the node representing the class variable will have size 2^n assuming we have n binary attributes. This is disastrous in many aspects. First, the storage cost would be very high and in some cases infeasible. Also, doing inference involving such a node would be very computationally expensive. Finally, as Friedman and Goldszmidt noted, "learning many parameters is a liability, since a large number of parameters requires a large training set to be assessed reliably" [13]. Noisy-or solves this problem by assuming independences between the different parents or causes. In essence, the noisy-or model approximates the CPT for the common effect by using an OR function. This is an acceptable approximation in cases where the different causes are all each individually sufficient to produce the effect and their ability to produce the effect is not affected by the presence of the other causes. That explains the "or" part of the noisy-or. So where does the "noisy" part come from? Since the OR function is deterministic, it renders the node fairly trivial. If any one of the causes is present, then the effect has to be present, and only when no causes are present is the effect absent. Since this does not happen in most situations we want to model, we need to introduce noise to the model for it to be able to accommodate more complex scenarios. With noise, each cause only produces the effect with a certain probability.

Henrion later extended the noisy-or model for situations where the effect is present even when all of its causes are absent [18]. The extended model, the leaky noisy-or gate, is applicable to situations where a model does not capture all possible causes. Arguably, almost all situations encountered in practice belong to this class. The model introduces an additional parameter called the leak probability, which is the combined effect of all causes not in the network. Diez later proposed an alternative way to formulate the leaky noisy-or gate [8]. The two proposals differ in how they define their parameters. In Henrion's model, the probabilities for each cause to produce the effect is a combined influence of the cause in question and the leak. On the other hand, Diez explicitly refer to the mechanism between the cause in question and the effect with the leak absent. For learning from data though, Henrion's model seems more useful since in all observed instances the leak is always present by definition.

Some other extensions of the noisy-or model include the introduction of multi-valued variables [18, 8] as well as nodes that include multiple outcomes [36, 8]. In these models, the variables are no longer binary. Rather, their values represent the degree of intensity. The degree of the outcome is a maximum of the degrees produced by the causes. Thus, this model of interaction could also be called a noisy-max. Heckerman went further and introduced independence of causal influence (ICI) [16, 38]. In an ICI model, the function need not to be an OR any more. Examples include noisy-and, noisy-max, noisy-min, noisy-add, etc. Causal independence is a collection of conditional independence assertions and functional relationships that are often appropriate to apply to the representation of the uncertain interactions between cause and effect. Its use can greatly simplify probability assessment as well as probabilistic inference [16]. Srinivas also described a slightly less general from of causal independence [36].

In this paper, we will follow the ICI model as our formulation for the noisy-or classifier. In an ICI classifier, each attribute $A_i, i = 1, \ldots, n$ has a child A'_i that represents the result of applying the noise. Variables $A'_i, I = 1, \ldots, n$ are parents of the class variable C. $Pr(C \mid \mathbf{A}')$ represents a deterministic function f that assigns to each combination of values $(a'_1, \ldots, a'_k)$ a class c. Following Srinivas [36], we can represent an ICI model using the Bayesian network structure in Figure 5. Since we are dealing with binary noisy-or classifiers, the function f will be an OR. Thus, whenever any A'_i is 1, then $C = 1$. The classification rule will be the same as for the naive Bayes classifier. Namely, given a noisy-or network N and the threshold t, the output of the classifier is 1 if $Pr(C = 0 \mid \mathbf{A} = \mathbf{a}) < t$

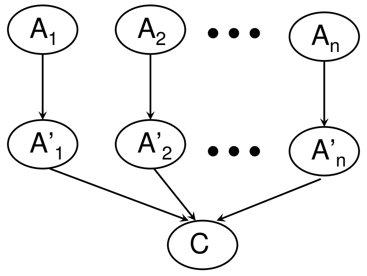

Figure 5: ICI model

and 0 otherwise.

Noisy-or model is useful in many different ways other than being a classifier. It is a natural way to model situations where there many causes for a common effect. Consequently, noisy-or is often employed in medical diagnosis [35, 25, 28] because normally a variety of diseases can all cause the same symptoms. Moreover, the situation mostly satisfies the two conditions pearl set out for using the noisy-or model: each cause is sufficient for producing the effect and the ability of each cause being sufficient is independent of all the other cause [30]. Any disease by itself is usually enough to cause the observed symptom and the diseases do not interfere with each other. Onisko et al tested the noisy-or model on HEPAR II, a model for diagnosis of liver disorders and observed a 6-10% increase in diagnostic accuracy [28]. Noisy-or is also applied when there are a small number of learning samples. Since noisy-or greatly reduces the number of parameters that need to be learned, it makes the resulting network more reliable [28, 13].

As mentioned earlier, there has been relatively little literature on noisy-or classifiers. Thus, its performance as a classifier remains to be investigated. However, since it is a discriminative classifier that has the same expressive power as naive Bayes, one would expect it to perform better, at least in the limit case. Regardless, a deeper investigation into noisy-or classifiers could provide insight into how to improve the other types of classifiers.

2.3 Logistic Regression

Logistic regression, or sometimes called logistic discrimination [2], has long been a tool for the statistics community. It is a statistical regression model for binary dependent variables. It attempts to learn functions of the form $f : X \rightarrow Y$. In the context of our classification problem, it attempts to learn $Pr(C \mid \mathbf{A})$. Since it is learning the posterior distribution directly, it is a discriminative classifier. Although we are not focusing our attention on logistic regression in this paper, we will use it in our derivations in the next section. Moreover, it comes up often in the literature in discussion about comparing different classifiers [27, 38, 26]. The fundamental assumption of logistic regression is that the logit, or the log-likelihood ratio, is assumed to be linear:

$$\log \frac{Pr(C=0 \mid \mathbf{A}=\mathbf{a})}{1-Pr(C=0 \mid \mathbf{A}=\mathbf{a})} = \beta_0 + \sum_j \beta_j \cdot a_j = \beta_0 + \sum_{j:a_j=1} \beta_j \tag{1}$$

Typically, $\beta_0 = \beta_0' + \log \frac{Pr(C=0)}{Pr(C=1)}$. We can rewrite (1) as

$$Pr(C=0 \mid \mathbf{A}=\mathbf{a}) = \frac{\exp(\beta_0 + \sum_{j:a_j=1} \beta_j)}{1+\exp(\beta_0 + \sum_{j:a_j=1} \beta_j)} \tag{2}$$

We also know that

$$Pr(C=1 \mid \mathbf{A}=\mathbf{a}) = \frac{1}{1+exp(\beta_0 + \sum_{j:a_j=1} \beta_j)} \tag{3}$$

Since these two probabilities have to add up to 1. For the classification task, we want to assign an instance with a class value c that maximizes $Pr(c \mid \mathbf{a})$. In other words, we will assign an instance to $C = 1$ if

$$Pr(C=1 \mid \mathbf{a}) > Pr(C=0 \mid \mathbf{a}) \tag{4}$$

We can rewirte condition (4),

$$\frac{Pr(C=0 \mid \mathbf{a})}{Pr(C=1 \mid \mathbf{a})} < 1 \tag{5}$$

$$\log \frac{Pr(C=0 \mid \mathbf{A}=\mathbf{a})}{1-Pr(C=0 \mid \mathbf{A}=\mathbf{a})} < 0 \tag{6}$$

By substituting our first assumption (1), we have

$$\beta_0 + \sum_{j:a_j=1} \beta_j < 0 \tag{7}$$

As we will see in the next section, naive Bayes, noisy-or, and logistic regression are all equivalent as classifiers in the sense that given any one of the classifiers, we can construct the other two classifiers that classify every possible instance in the same way. Nevertheless, logistic regression is a powerful tool because it benefits from the many tools available in statistics. In practice, logistic regression can be improved upon by regularization techniques such as "shrinking the parameters via the L_1 constraint, imposing a margin constraint in the separable case, or various forms of averaging" [27]. Also, logistic regressions are not bound to the Bayesian network assumptions as tightly as the other two classifiers. Given data that disobeys the assumption, the conditional likelihood maximization algorithm for logistic regression will adjust its parameters to fit the data [26].

However, such adjustments and the before mentioned regularization techniques can be viewed as changing the model class. Thus, this does not contradict with our claim that the three classifiers have equivalent expressive powers. On the other hand, their abilities to learn from data do vary and thus provide different results depending on the data. Ng and Jordan provide an in-depth study of the comparison between logistic regression and naive Bayes [27].

3 Expressiveness

There are two dimensions to consider when evaluating a classifier. The first is its expressive power, or how flexible is the model in representing the relationship between the attributes and the class. On one extreme would be a general Bayesian network which could represent any probabilistic relationship between the variables. On the other hand would be naive Bayes which impose strong independence assumptions. The second dimension to consider is the classifier's ability to learn from data. We will first tackle the issue of expressiveness in this section. As stated before, naive Bayes, noisy-or, and logistic regression classifiers all have the same expressive power. They are all linear classifiers. In the binary case, this means they define a hyperplane over the attribute space and separates it into two halves. Having equivalent expressive power means that they will induce the same classifiers in the limit as the training sample size approaches infinity.

First we will review the defintions for our classifiers.

Definition 3.1 (Naive Bayes classifier)**.** *Given a naive Bayes network N, and the threshold t, the naive Bayes classifier F^t_N is defined as follows:*

$$F^t_N = \begin{cases} 1 & \textit{if } Pr(C = 0 \mid \mathbf{A} = \mathbf{a}) < t; \\ 0 & \textit{otherwise.} \end{cases}$$

The network N consists of the parameters $Pr(C)$ and $Pr(A_i \mid C)$.

Definition 3.2 (Noisy-or classifier)**.** *Given a noisy-or network N, and the threshold t, the noisy-or classifier F^t_N is defined as follows:*

$$F^t_N = \begin{cases} 1 & \textit{if } Pr(C = 0 \mid \mathbf{A} = \mathbf{a}) < t; \\ 0 & \textit{otherwise.} \end{cases}$$

The network N consists of the parameters $Pr(A_i)$ and $Pr(A'_i \mid A_i)$. However, the prior $Pr(A_i)$ is not as important because typically the values of the attributes are given.

Definition 3.3 (Logistic regression classifier)**.** *Given the set of parameters β, the logistic regression classifier F_β is defined as follows:*

$$F_\beta = \begin{cases} 1 & \textit{if } \beta_0 + \sum_{j:a_j=1} \beta_j < 0; \\ 0 & \textit{otherwise.} \end{cases}$$

3.1 Correspondence between noisy-or and logistic regression

The derivation for this correspondence is detailed in Vomlel's paper [38]. Thus, we will only provide the end results here.

3.1.1 From noisy-or to logistic regression

Given a noisy-or classifier, we can define an equivalent logistic regression classifier with the following parameters:

$$\beta_j = \log \frac{Pr(A'_j = 0 \mid A_j = 1)}{Pr(A'_j = 0 \mid A_j = 0)} \tag{8}$$

$$\beta_0 = \sum_j \log Pr(A'_j = 0 \mid A_j = 0) - \log t \tag{9}$$

3.1.2 From logistic regression to noisy-or

Given a logistic regression classifier with the parameters β, we can define an equivalent noisy-or classifiers with the follwing parameters:

$$Pr(A'_j = 0 \mid A_j = a_j) = \begin{cases} \frac{exp(\beta_j)}{1+exp(\beta_j)} & \text{for } a_j = 1; \\ \frac{1}{1+exp(\beta_j)} & \text{for } a_j = 0. \end{cases} \tag{10}$$

$$t = \frac{1}{exp(\beta_0) \cdot (\prod_j (1 + exp(\beta_j)))} \tag{11}$$

3.2 Correspondence between naive Bayes and logistic regression

3.2.1 From naive Bayes to logistic regression

Given a naive Bayes classifier, the instance is classified to $C = 1$ when $Pr(C = 0 \mid \mathbf{a}) < t$. We can rewrite the equation in log-odds space [5]. The condition for $C = 1$ would now be:

$$\log O(C = 0 \mid \mathbf{a}) < \log(\frac{t}{1-t}) \tag{12}$$

Following Chan et al's derivation [5], we have

$$\log O(C = 0 \mid \mathbf{a}) = \log O(C = 0) + \sum_j \log \frac{Pr(a_j \mid C = 0)}{Pr(a_j \mid C = 1)} \tag{13}$$

And let

$$f_j(a_j) \quad = \quad \log \frac{Pr(a_j \mid C = 0)}{Pr(a_j \mid C = 1)} \tag{14}$$

$$\tag{15}$$

We can now rewrite condtion (12) further

$$\log O(C=0) + \sum_j \log \frac{Pr(a_j \mid C=0)}{Pr(a_j \mid C=1)} < \log(\frac{t}{1-t}) \tag{16}$$

$$\log O(C=0) + \sum_{j:a_j=0} f_j(0) + \sum_{j:a_j=1} f_j(1) < \log(\frac{t}{1-t}) \tag{17}$$

$$\log O(C=0) + \sum_j f_j(0) + \sum_{j:a_j=1} (f_j(1) - f_j(0)) < \log(\frac{t}{1-t}) \tag{18}$$

$$\log O(C=0) + \sum_j f_j(0) - \log(\frac{t}{1-t}) + \sum_{j:a_j=1} (f_j(1) - f_j(0)) < 0 \tag{19}$$

$$\tag{20}$$

Thus, we can define an equivalent logistic regression classifier with the following parameters:

$$\begin{aligned}
\beta_0 &= \log O(C=0) + \sum_j f_j(0) - \log(\frac{t}{1-t}) && (21)\\
&= \log O(C=0) + \sum_j \log \frac{Pr(A_j=0 \mid C=0)}{Pr(A_j=0 \mid C=1)} - \log(\frac{t}{1-t}) && (22)\\
\beta_j &= f_j(1) - f_j(0) && (23)\\
&= \log \frac{Pr(A_j=1 \mid C=0)}{Pr(A_j=1 \mid C=1)} - \log \frac{Pr(A_j=0 \mid C=0)}{Pr(A_j=0 \mid C=1)} && (24)
\end{aligned}$$

3.2.2 From logistic regression to naive Bayes

Given a logistic regression classifer, the instance is classified to $C = 1$ when $\beta_0 + \sum_{j:a_j=1} \beta_j < 0$. Let the parameters of a naive Bayes classifer be defined as

$$\log \frac{Pr(a_j \mid C=0)}{Pr(a_j \mid C=1)} = \begin{cases} \beta_j + \log K_j & \text{for } a_j = 1; \\ \log K_j & \text{for } a_j = 0. \end{cases} \tag{25}$$

$$\log O(C=0) = \beta_0 - \sum_j \log K_j \tag{26}$$

$$\log \frac{t}{1-t} = 0 \tag{27}$$

where K_j are constants whose value depends on β_j. We will discuss how to pick the values for K_j in a bit. We can now rewrite the condition of the logistic regression classifier.

$$\beta_0 + \sum_{j:a_j=1} \beta_j < 0 \tag{28}$$

$$\beta_0 - \sum_j \log K_j + \sum_{j:a_j=0} \log K_j + \sum_{j:a_j=1} (\beta_j + \log K_j) < 0 \tag{29}$$

$$\log O(C=0) + \sum_j \log \frac{Pr(a_j \mid C=0)}{Pr(a_j \mid C=1)} < \log \frac{t}{1-t} \tag{30}$$

This is exactly the condition for a naive Bayes classifier (see (16) on how we transformed the original naive Bayes condition). Thus, the naive Bayes classifier we defined (31, 32, 33) is equivalent to the logistic regression classifier. Before we finish though, let's discuss how to choose the appropriate values for K_j. First, let's do some algebra and simply the parameters for the naive Bayes classifier (31, 32) into the following:

$$Pr(A_j = 0 \mid C = 0) = \frac{1 - K_j \cdot \exp\beta_j}{1 - \exp\beta_j} \tag{31}$$

$$Pr(A_j = 0 \mid C = 1) = \frac{1 - K_j \cdot \exp\beta_j}{K_j - K_j \cdot \exp\beta_j} \tag{32}$$

$$Pr(C=0) = \frac{\exp\beta_0}{\exp\beta_0 + \prod_j K_j} \tag{33}$$

$$t = 0.5 \tag{34}$$

We need to ensure all of these probabilities have values between 0 and 1. Thus, we impose the following conditions on K_j:

If $\exp\beta_j > 1$ then we choose K_j such that $K_j < 1$ and $K_j \cdot \exp\beta_j > 1$. (35)

If $\exp\beta_j < 1$ then we choose K_j such that $K_j > 1$ and $K_j \cdot \exp\beta_j < 1$. (36)

One possible such value for K_j is $\frac{1+\exp\beta_j}{2\cdot\exp\beta_j}$. If we plug it back into equations (31, 32, 33, 34), we get:

$$Pr(A_j = 0 \mid C = 0) = \frac{1}{2} \tag{37}$$

$$Pr(A_j = 0 \mid C = 1) = \frac{\exp\beta_j}{1 + \exp\beta_j} \tag{38}$$

$$Pr(C=0) = \frac{\exp\beta_0}{\exp\beta_0 + \prod_j \frac{1+\exp\beta_j}{2\cdot\exp\beta_j}} \tag{39}$$

$$t = 0.5 \tag{40}$$

It's easy to verifty that these probabilities indeed have values between 0 and 1. One thing interesting to note is that we have reduced about half the parameters to constants. Since we can transform any naive Bayes classifier into a logistic regression classifier then back, this means for any naive Bayes classifier, there is an equivalent one with only $n+1$ parameters where n is the number of attributes. This is perhaps not surprising since logistic regression only has $n+1$ parameters.

3.3 Correspondence between naive Bayes and noisy-or

3.3.1 From naive Bayes to noisy-or

We simply combine the results we have derived above. Thus, we first transform the naive Bayes classifer to a logistic regression classifer (22, 24), and then we tranform it to a noisy-or classifier (10, 11). Thus, given a naive Bayes classifier with parameters $Pr(C)$, $Pr(A_j|C)$, and t we can define an equivalent noisy-or classifer with the following parameters:

$$Pr_{noisy}(A'_j = 0 \mid A_j = a_j) = \begin{cases} \frac{Pr(A_j=1|C=0)\cdot Pr(A_j=0|C=1)}{Pr(A_j=1|C=1)\cdot Pr(A_j=0|C=0)+Pr(A_j=1|C=0)\cdot Pr(A_j=1|C=0)} & \text{for } a_j = 1; \quad (41) \\ \frac{Pr(A_j=1|C=1)\cdot Pr(A_j=0|C=0)}{Pr(A_j=1|C=1)\cdot Pr(A_j=0|C=0)+Pr(A_j=1|C=0)\cdot Pr(A_j=1|C=0)} & \text{for } a_j = 0. \quad (42) \end{cases}$$

$$t_{noisy} = \frac{1}{O(C=0)\cdot \prod_j \frac{Pr(A_j=0|C=0)}{Pr(A_j=0|C=1)} \cdot \frac{1-t}{t} \cdot \prod_j (1 + \frac{Pr(A_j=1|C=0)\cdot Pr(A_j=0|C=1)}{Pr(A_j=1|C=1)\cdot Pr(A_j=0|C=0)})} \quad (43)$$

$$= \frac{1}{O(C=0)\cdot \frac{1-t}{t} \cdot \prod_j (\frac{Pr(A_j=0|C=0)}{Pr(A_j=0|C=1)} + \frac{Pr(A_j=1|C=0)}{Pr(A_j=1|C=1)})} \quad (44)$$

$$= \frac{Pr(C=1)}{Pr(C=0)} \cdot \frac{t}{1-t} \cdot \prod_j (\frac{Pr(A_j = 0 \mid C = 1)}{Pr(A_j = 0 \mid C = 0)} + \frac{Pr(A_j = 1 \mid C = 1)}{Pr(A_j = 1 \mid C = 0)}) \quad (45)$$

3.3.2 From noisy-or to naive Bayes

Similarly, we can derive the transformation the other way. We first transform a noisy-or classifier into a logistic regression classifier (8, 9), and then into a naive Bayes classifier (37, 38, 39, 40). Given a noisy-or classifier with parameters $Pr(A'_j \mid A_j)$ and t, we can define an equivalent naive Bayes classifier with the following parameters:

$$Pr_{naive}(A_j = 0 \mid C = 0) = \frac{1}{2} \quad (46)$$

$$Pr_{naive}(A_j = 0 \mid C = 1) = \frac{Pr(A'_j = 0 \mid A_j = 1)}{Pr(A'_j = 0 \mid A_j = 0) + Pr(A'_j = 0 \mid A_j = 1)} \quad (47)$$

$$Pr_{naive}(C = 0) = \frac{\prod_j (2 \cdot p_{j01} \cdot p_{j00})}{t \cdot \prod_j (p_{j00} + p_{j01}) + \prod_j (2 \cdot p_{j01} \cdot p_{j00})} \quad (48)$$

where

$$p_{j00} = Pr(A'_j = 0 \mid A_j = 0) \quad (49)$$

$$p_{j01} = Pr(A'_j = 0 \mid A_j = 1) \quad (50)$$

4 Learning

4.1 Overview

Having shown the equivalent expressive powers of the classifiers, we will now deal with the other aspect of evaluating a classifier: its learning ability. At first glance, one may expect that naive Bayes, noisy-or and logistic regression will also all be equal in this regard. After all, we can readily transform one type of classifier into the other two. Thus, theoretically they would all produce the same classifier if the learning algorithms find the optimal parameters. However, empirical evidence suggests otherwise. In Vomlel's experiments, he found logistic regression to be the best, noisy-or second, and naive Bayes worst when tested on his selected datasets [38]. The reason for this being that the learning algorithms differ for the different classifiers. Thus, depending on the particular training data, they will produce different results.

Learning Bayesian networks are divided into four categories depending on whether the network structure is known and whether the training data is complete. In the last decade, there has been a lot of work done in this area [4, 14, 21, 15, 17]. There are two parts to the learning process: learning the structure and learning the parameters. Of the two, identifying good structures is much more difficult and is generally NP-hard [6]. Thus, most of the algorithms for learning the structure are iterative. They start out with a simple network and incrementally add or delete edges to increase the quality of the network. The problem becomes even more difficult with incomplete data. In such cases, we have to optimize both the structure and the parameters simultaneously. On the other end, we have the simplest case of known structure and complete data. The goal then is to simply find a set of parameters that will maximize the likelihood of the data, also known as maximum likelihood parameters. With complete data, the problem has a unique solution and the process involves simple counting and some smoothing.

Unfortunately, in real-life situations, we often do not have complete data due to the difficulty of collecting data and privacy issues. Moreover, we often do not know the network structures. This makes learning very hard to do and very computationally expensive. Thus, to alleviate the problem, we use simple Bayesian networks such as naive Bayes and noisy-or and try to optimize just the parameters. This works surprisingly well perhaps because classification is a small subset of the capability of a general Bayesian network. Since we are assuming the structure of the networks, the kind of learning we are focusing on here is with known structure and incomplete data.

Two common methods for dealing with incomplete data include gradient ascent and the expectation maximization (EM) algorithm. However, neither is optimal and both can be computationally intensive. Gradient ascent works by starting out with an initial set of parameters and gradually changing them based on the gradients in attempt to find the maximum likelihood parameters.

The algorithm is not optimal since it is only guaranteed to find local maximums. The EM-algorithm consists of two parts, expectation and maximization as its name suggests. In the expectation step, we try to complete the data using the current set of parameters. Each possible value of the missing variable is assigned a weight based on its probability of occurring. In the maximization step, we compute the new set of parameters as in the complete data case using the weighted data we obtained in the previous step. The process continues until a certain convergence criteria is met. The algorithm was first introduced by Dempster et al [7]. It was later extended for graphical models by Lauritzen [22].

Of the three classifiers discussed, logistic regression seems to perform the best overall in practice. However, since it is not a Bayesian network classifier, we will not dwell on its learning method. Part of the reason why it does so well is due to the fact that it is a discriminative classifier as opposed to a generative classifier such as the naive Bayes. Prevailing folk wisdom suggests that discriminative classifiers produce better results because they have a more focused task of optimizing only the classification power. Posting queries about the other variables to a discriminative classifier, however, will often result in meaningless answers. On the other hand, generative classifiers can be used to answer questions about correlations between any sets of variables in the model.

Since noisy-or is also a discriminative model, it is expected to perform better than naive Bayes. However, its learning method is perhaps not as well polished as the other two classifiers. While noisy-or gates are widely studied and used in larger networks, its use as a classifier has been relatively unexplored. This is true especially compared to the many smoothing and optimizing technique available to statistic regression. Nevertheless, since it is a Bayesian network, it enjoys many of the advancements that have been made in learning Bayesian networks.

Although people have long thought that discriminative classifiers are to be preferred over generative classifiers such as naive Bayes, there are several reasons to consider using a naive Bayes network. Naive Bayes has been shown to work really well empirically, especially on small datasets with many attributes [9]. Also, naive Bayes converges to its asymptotic error quicker, in $\log n$ time in general [27]. Finally, its ability to answer other queries is worth considering, even though it is not relevant to the classification task. Let's take the example of credit scoring. The main goal is to decide whether to grant a loan to an applicant, which is a straightforward classification task. However, understanding the correlations between the attributes may also help the bank in making other business decisions. For example, the bank can identify the applicant's weak areas and closely monitor them.

4.2 Experiment

We already know that different classifiers excel in different situations. However, it would be useful to know under exactly which conditions will each classifier perform the best. Moreover, understanding their behaviors better can lead to new classifiers or new learning algorithms that combine their strengths. Thus, we conducted experiments with the naive Bayes and noisy-or classifiers in attempt to determine how their learning abilities stack up to each other. Logistic regression is left out because we are mainly concerned with Bayesian network classifiers and how to improve upon the existing models.

4.2.1 Data

We generate two sets of data to test our classifiers on. One set of data is generated by a naive Bayes network, the other by a noisy-or network. The networks are all randomly initialized. The hypothesis is that a naive Bayes classifier would perform better with data generated by a naive Bayes network, and vice versa.

Generating data from a naive Bayes network is straightforward since its structure is conducive to such tasks. We first initialize the prior probability of the class variable and also the conditional probabilities of the attributes. We then proceed to produce instances according to those probabilities. The nice thing about naive Bayes is that we can control the probability of the class variable easily. Thus, we can ensure there are enough cases for each possible class value to be trained on.

Generating data from a noisy-or network is slightly trickier. If we simply initialized the parameters completely randomly as we did in the naive Bayes network, we would end up with very uneven data. The reason for this is that it only takes one positive input to the or-gate to make the instance's class value to be 1. Thus, with completely random parameters, almost all the instances produced would belong to class 1. To deal with this issue, we look to real-life situations where the noisy-or model is applied. One such application is in the medical field where a noisy-or gate is used to model the relationship between diseases and symptoms. The diseases are inputs and the symptoms are the outputs. Normally, only a few inputs would be active. In other words, the patient usually has one of the diseases, or at most a few of the diseases. Thus, in our noisy-or network, we limit the prior probabilities of the attributes to a low probability, scaled by the number of inputs. Also, we limit the probability of the leak variable as well. If the probability of the leak variable being present is high, that would mean there are important causes that we are not modeling. After we initialize the noisy-or network, we generate instances according to those parameters.

We generated datasets ranging from 10 attributes to 100 attributes. We also varied the rate of missing data, from complete data to half the data missing. Notice only the attributes can be missing. An instance without a class label is essentially useles.

4.2.2 Learning Methods

To learn the network parameters from the data, we will use the EM-algorithm for both the naive Bayes and the noisy-or classifiers. As described earlier, the algorithm consists of two parts. In the E-step we consider all possible completions of the data and assign each complete a weight. Then in the M-step, we update the parameters based on these weighted data.

We will first describe the M-step, or the only step required in cases of complete data. Given the dataset D and current parameters θ, we update the parameters as follows:

$$\theta'_{x|\mathbf{u}} = \frac{\sum_{d\in D} Pr_\theta(x\mathbf{u} \mid d)}{\sum_{d\in D} Pr_\theta(\mathbf{u} \mid d)}$$

More specifically, for naive Bayes, we set

$$Pr'(A_i = a_i \mid C = c) = \frac{\sum_{d\in D} Pr(A_i = a_i, C = c \mid d)}{\sum_{d\in D} Pr(C = c \mid d)}$$

For complete data, computing these quantities reduce down to simple counting. For incompleted data, we would first perform the E-step of the algorithm. In this case, it's also fairly simple since

$$Pr(A_i = a_i, C = c \mid d) = \begin{cases} Pr(A_i = a_i \mid C = c) & \text{if } C = c \text{ in } d; \\ 0 & \text{otherwise.} \end{cases}$$

where $Pr(A_i = a_i \mid C = c)$ is simply the current parameter value.

For noisy-or, we set the parameters to be

$$Pr'(A'_i = a'_i \mid A_i = a_i) = \frac{\sum_{d\in D} Pr(A'_i = a'_i, A_i = a_i \mid d)}{\sum_{d\in D} Pr(A_i = a_i \mid d)}$$

This is slightly more difficult to compute since the values of A'_i are not observed. For the complete data case, we have

$$Pr(A'_i = a'_i, A_i = a_i \mid d) = \begin{cases} Pr(A'_i = a'_i \mid d) & \text{if } A_i = a_i \text{ in } d; \\ 0 & \text{otherwise.} \end{cases}$$

From Vomlel's results [38], we have

$$Pr(A'_i = 0 \mid d) = \begin{cases} 1 & \text{if } c = 0 \text{ in } d; \\ \frac{Pr(A'_i=0|A_i=a_i) - \prod_j Pr(A'_j=0|A_j=a_j)}{1-\prod_j Pr(A'_j=0|A_j=a_j)} & \text{otherwise.} \end{cases}$$

The value of $Pr(A'_i = 1 \mid d)$ is also defined since $Pr(A'_i = 0 \mid d) + Pr(A'_i = 1 \mid d) = 1$

For the incomplete data case, we perform the E-step by multiplying the above defined values by the prior probabilities of the attributes. Thus, we have,

$$Pr(A'_i = a'_i, A_i = a_i \mid d) = Pr(A'_i = a'_i \mid d) * Pr(A_i = a_i)$$

4.2.3 Results

The results of the experiments are shown in Table 1 and Table 2 attached at the end of the paper. The numbers in the tables represent the accuracy rates defined as follows:

$$\text{accuracy} = \frac{\text{\# correct classifications}}{\text{\# test cases}}$$

We ran each scenario (# attributes and % data missing) 10 times and averaged the results. Each time we use a different, independently initialized network to generate the data. The classifiers are trained on 5000 instances and then tested on another 5000 new instances.

The results are more or less what we would expect to see. For the data generated by naive Bayes networks, the naive Bayes classifier clearly performs better than the noisy-or classifier in all cases (Table 1). The naive Bayes classifier is so effective in certain cases that it reaches 100% accuracy. This is perhaps not surprising since its weakness lies in its strong independence assumptions. However, if the data indeed satisfies those independence conditions, as is the case here, then the naive Bayes classifier becomes very accurate. In comparing naive Bayes to logistic regression, Mitchell stated,

> Naive Bayes is a learning algorithm with greater bias, but lower variance, than Logistic Regression. If this bias is appropriate given the actual data, Naive Bayes will be preferred. Otherwise, Logistic Regression will be preferred. [26]

Also worth noting is that naive Bayes does better with more attributes and less missing data. This follows our intuition that if we model more of the attributes and have more complete data, we can produce better classifiers. Noisy-or classifier, on the other hand, does not seem to benefit from these characteristics. Its performance indicate no clear patterns.

Interpreting the results of learning from the data generated by noisy-or networks is slightly more difficult (Table 2). Overall, noisy-or classifier performs better, but not much better than the naive Bayes classifier. The difference is the greatest when there are only 10 attributes. As the number of attributes increase, the difference in accuracies between the two classifiers diminish.

There are several possible explanations. First, as we noted before, naive Bayes classifiers tend to excel at problems with many attributes. Thus, it is possible that the naive Bayes classifier closes the gap as the number of attributes increase. Another possibility is that the learning algorithm we used for noisy-or is not really suitable. As discussed in Section 4.2.2, the EM algorithm is more complicated when applied to the noisy-or model. Finally, it is possible that the data generated by the noisy-or networks are simply very difficult to learn. As a result, both classifiers perform equally poorly. This last possibility is supported by the fact that it is difficult to generate good, balanced data from a noisy-or network as noted in Section 4.2.1. We had to restrict the parameters of the networks to even get data that do not all have class value 1.

Noisy-or classifiers exhibit a preference for fewer attributes and more missing data when learning from noisy-or data. Interesting enough, naive Bayes also shows an improvement in performance at higher data missing rate, except in the case of complete data. This is possibly related to the way the data is generated. Since we had to limit the prior probabilities based on the number of attributes, having many attributes also means the probability of each attribute occurring is small. This in turn affects learning because we would have fewer instances of where a particular attribute is present. Consequently, the 5000 training samples we used may have become insufficient for scenarios with many attributes. On the other hand, the apparent preferences for higher data missing rate is perhaps more a reflection of the distribution of the data. In some cases the classifier do so poorly with the data that they classify all the instances to one class. Thus, having a distribution that is lopsided will seemingly boost the accuracies of the classifiers in those situations.

4.3 Discussion

Our experiments have reaffirmed the notion that different classifier indeed excel on different datasets despite their equivalent expressive powers. Specifically, we have shown that naive Bayes and noisy-or classifiers perform better when the data is more aligned with its independence assumptions. However, we need stricter criteria to determine which classifier will be better. For most problems, the data is more complicated with some of the attributes being causes while the other being effects of the class value. This prompts us to wonder about using hybrid structures such as the one shown in Figure 4. Although building networks that more closely resemble the actual relationships between the variables would probably result in better classifiers, it is costly. Thus, we need to explore the trade-offs between the complexity of the network and the quality of the classifier.

Various extensions to the naive Bayes model already exist, such as TAN [12] and CAN [31]. However, it is not clear whether the improvement in performance is enough to justify the higher computing cost. Ultimately, we would like to build classifiers with just the right balance of speed and accuracy. Thus, we want to find network structures as simple as possible yet still achieve the level of accuracy we desire.

There is still room for improvement concerning the learning methods as well. Clearly, none of the existing algorithms are optimal. Since the three classifiers we discussed can be transformed into one another, their best classifiers would be equivalent. Thus, if we can learn the optimal parameters in one model, we should get a classifier that outperforms all the other classifiers in the other models as well. Of the three classifiers we discussed, logistic regression probably has the most polished learning method, and noisy-or the least polished.

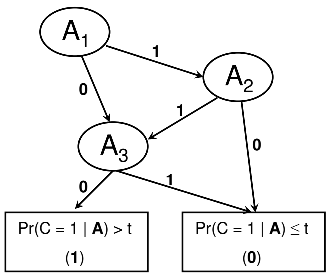

Figure 6: Ordered Decision Diagram

5 Ordered Decision Diagram

In this section we discuss another type of representation for classifiers, Ordered Decision Diagram (ODD).

Definition 5.1. *An Ordered Decision Diagram is a directed, acyclic graph with a single root that is defined with respect to a specific variable ordering $(X_1, \ldots, X_n)$. The leaves of the graph, or the terminal nodes, are called sinks. Each sink represents a possible value of the output. In the case of classifiers, the sinks correspond to the class values. The rest of the nodes are associated with a specific variable X_i and labeled accordingly. For a classifier, these variables are the attributes. Note that there can be many nodes associated with a single variable. The only restriction is that in any directed paths, if a node X_i comes before a node X_j, then $i < j$. For a node X_i, the outgoing edges represent the possible values of X_i, or ranges of values in the case of continuous variables. Note that every possible value must be associated with exactly one edge.*

An ODD is used as a classifier as follows. Given an instance with attributes $\mathbf{A} = A_1, \ldots, A_n$, the ODD determines its class value by traversing the graph starting at its root. For every node A_i it encounters, it follows the edge a_i out of the node if $A_i = a_i$ in the instance. When it eventually reaches a sink c_i, the instance is assigned the class value c_i. An example of an ODD classifier can be seen in Figure 6.

Transforming classifiers into ODDs allow us to efficiently perform several operations. Given two ODDs D and D' with sizes s and s', respectively, we can test their equivalence in $O(s+s')$ time and conjoin or disjoin them in $O(ss')$ time. Other operations include counting how many instances are mapped to each class and testing whether a classifier with binary class value can be reduced down to a conjunction or disjunction of the attributes.

Testing for equivalence is especially helpful in our studies of classifiers. It allows us to directly compare two seemingly different networks and see whether they define the same classifying function. ODDs also allow us to determine the similarity between two classifiers. By joining two ODDs together and then doing counting, we can determine on how many instances the two classifiers agree on. Moreover, these operations let us examine how much effect certain modifications to the networks have on the classifiers. This is important if we want to find a good balance between network complexity and performance.

Chan et al have already shown how to efficiently convert a naive Bayes classifier into an ODD [5]. Since we can transform noisy-or and logistic regression classifiers into a naive Bayes classifier, we can readily transform those into ODDs as well. Thus, we have a complete framework for comparing the performances of each of the three classifiers.

6 Conclusion

In this paper we explored the problem of classification and couple Bayesian network classifiers that are used to solve the problem. Namely, we looked into naive Bayes and noisy-or classifiers. We compared them on two dimensions, their expressive powers and their learning abilities. We also compared them against a third classifier, the logistic regression classifier. While all three have equivalent expressiveness, they differ on how well they learn from different sets of data. As one would expect, naive Bayes and noisy-or classifiers perform better when the data is aligned with their independence assumptions. The issue of converging speed is also an issue too. While logistic regression may produce superior classifiers given enough training, naive Bayes may be preferred when there are few training samples. Thus, so far it seems that each classifier would have its own niche.

While we acknowledge the fact that different classifiers may be preferred under different circumstances, we do not have a clear set of criteria for determining which classifier should be used. There remains much work to be done in this area to assess how and perhaps why one type of classifier outperforms another on a certain dataset. Also, once we gain a better understanding of the relationship between the characteristic of the data and the performance of the classifiers, we may be able to build new classifiers that combine the strong areas of each of the classifiers.

One way to enhance the quality of the Bayesian network classifiers is to introduce more complex structures that model the actual relationships between the variables more realistically. However, finding a good structure is often difficult and costly. Moreover, it is unclear how much increase in accuracy do we gain by doing so. Ideally, we would like to have classifiers that have simple structures but have high accuracies. To work toward that goal, we need to figure out what changes to the network structures are the most effective in improving the classifier's performance.

To facilitate such studies of comparisons between classifiers, we look at

ODDs. They provide efficient ways to test how similar two classifiers are. Thus, we can quantify the effects certain changes to the networks have on the classifiers. This will allow us to identify those modifications that have the most impact.

Ultimately, we want to build classifiers that are fast in terms of classification and learning speed, as well as have high accuracy. Perhaps such classifiers won't be built upon Bayesian networks at all. Nevertheless, it is unlikely that there will be one type of classifier that will always be superior in all situations. Depending on the nature of the data, the number of training samples available, and the level of accuracy we desire, different types of classifiers may be preferred. Thus, it is important for us to closely study and compare the different types of classifiers.

References

[1] A. Almonayyes. Multiple explanations driven naïve bayes classifier. *J. UCS*, 12(2):127–139, 2006.

[2] J. A. Anderson. Logistic discrimination. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics 2: Classification, Pattern Recognition and Reduction of Dimensionality*, pages 169–191, Amsterdam, 1982. North Holland.

[3] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras, and Constantine D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering, 2000.

[4] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Transactions On Knowledge And Data Engineering*, 8:195–210, 1996.

[5] Hei Chan and Adnan Darwiche. Reasoning about bayesian network classifiers. In Christopher Meek and Uffe Kjærulff, editors, *UAI*, pages 107–115. Morgan Kaufmann, 2003.

[6] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, October 2004.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Proceedings of the Royal Statistical Society*, B-39:1–38, 1977.

[8] F. J. Díez. Parameter adjustment in bayes networks. the generalized noisy OR- gate. In David Heckerman and Abe Mamdani, editors, *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*, pages 99–105, San Mateo, CA, USA, July 1993. Morgan Kaufmann Publishers.

[9] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[10] Richard Duda and Peter Hart. *Pattern Recognition and Scene Analysis.* John Wiley and Sons, 1973.

[11] L. De Ferrari. Mining housekeeping genes with a naive bayes classifier. Master's thesis, The University of Edinburgh, 2005.

[12] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[13] Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. In Eric J. Horvitz and Finn Verner Jensen, editors, *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, pages 252–262. Morgan-Kaufmann, 1996.

[14] Nir Friedman, Iftach Nachman, and Dana Peér. Learning bayesian network structure from massive datasets: The "sparse candidate" algorithm. In Kathryn B. Laskey and Henri Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 206–215, S.F., Cal., July 30–August 1 1999. Morgan Kaufmann Publishers.

[15] David Heckerman. Tutorial on learning in bayesian networks. Technical Report MSR-TR-95-06, Microsoft, 1995.

[16] David Heckerman and John S. Breese. Causal independence for probability assessment and inference using bayesian networks. Technical Report MSR-TR-94-08, Microsoft Research (MSR), March 1994. D. Heckerman and J.S. Breese, "Causal Independence for Probability Assessment and Inference Using Bayesian Networks," *IEEE Transactions on Systems, Man & Cybernetics, Part A (Systems & Humans)*, Vol. 26, No. 6, pp.826-831.

[17] David Heckerman, Dan Geiger, and David M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197, 1995.

[18] Max Henrion. Some practical issues in constructing belief networks. In Laveen N. Kanal, Tod S. Levitt, and John F. Lemmer, editors, *UAI*, pages 161–174. Elsevier, 1987.

[19] E. Keogh and M. Pazzani. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, pages 225–230, 1999.

[20] Jin H. Kim and Judea Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *IJCAI*, pages 190–193, 1983.

[21] W. Lam and F. Bacchus. Learning Bayesian belief networks. an approach based on the MDL principle. *Computational Intelligence*, 10(4):269–293, 1994.

[22] Steffen L. Lauritzen. The EM algorithm for graphical association models with missing data. Technical Report TR-91-05, Department of Statistics, Aalborg University, 1991.

[23] David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE. Published in the "Lecture Notes in Computer Science" series, number 1398.

[24] Andrew McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of AAAI-98, Workshop on Learning for Text Categorization*, 1998.

[25] B. Middleton, M. Shwe, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base: Part II. Evaluation of diagnostic performance. *SIAM Journal on Computing*, 30:256–267, 1991.

[26] Tom M. Mitchell. Generative and discriminative classifiers: Naive bayes and logistic regression. In *Draft of Machine Learning*, chapter 1, pages 1–17. McGraw Hill, 2005.

[27] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 841–848. MIT Press, 2001.

[28] Agnieszka Onisko, Marek J. Druzdzel, and Hanna Wasyluk. Learning bayesian network parameters from small data sets: application of noisy-OR gates. *Int. J. Approx. Reasoning*, 27(2):165–182, 2001.

[29] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. Technical Report CSD-850021, R-43, UCLA Computer Science Department, June 1985.

[30] J. Pearl. *Probabilistic reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[31] Fuchun Peng, Dale Schuurmans, and Shaojun Wang. Augmenting naive Bayes classifiers with statistical language models. *Information Retrieval*, 7(3-4):317–345, 2004.

[32] J. D. M. Rennie. Improving multi-class text classification with naive bayes. In *MIT AI-TR*, 2001.

[33] Jason D. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 616–623. AAAI Press, 2003.

[34] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*, 2001.

[35] M. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base: Part I. The probabilistic model and inference algorithms. *SIAM Journal on Computing*, 30:241–250, 1991.

[36] S. Srinivas. A generalization of the noisy-Or model. In D. Heckerman and A. Mamdani, editors, *Uncertainty in Artificial Intelligence: Proceedings of the Ninth Conference*, pages 208–215, Washington, DC, 1993.

[37] V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[38] J. Vomlel. Noisy-or classifier. In *Proceedings of the 6th Workshop on Uncertainty Processing (WUPES 2003)*, pages 291–302, Hejnice, September 2003.

[39] Yirong Yang, Yi Xia, Yun Chi, and Richard R. Muntz. Learning naive bayes classifier from noisy data. Technical Report CSD-TR No. 030056, ftp://ftp.cs.ucla.edu/tech-report/2003-reports/030056.pdf, UCLA, 2003.

[40] Huajie Zhang and Charles X. Ling. Learnability of augmented naive Bayes in nominal domains. In *Proc. 18th International Conf. on Machine Learning*, pages 617–623. Morgan Kaufmann, San Francisco, CA, 2001.

# attributes		Missing data rate 0.0	0.1	0.2	0.3	0.4	0.5
10	naive Bayes	**0.9381**	**0.9426**	**0.9214**	**0.9217**	**0.8840**	**0.8414**
	noisy-OR	0.7624	0.8176	0.7746	0.8695	0.7044	0.5772
20	naive Bayes	**0.9947**	**0.9830**	**0.9765**	**0.9439**	**0.9430**	**0.9371**
	noisy-OR	0.6841	0.8249	0.7153	0.7412	0.7741	0.8337
30	naive Bayes	**0.9958**	**0.9963**	**0.9894**	**0.9812**	**0.9711**	**0.9656**
	noisy-OR	0.7889	0.7870	0.6766	0.8273	0.7822	0.8247
40	naive Bayes	**0.9986**	**0.9989**	**0.9955**	**0.9905**	**0.9831**	**0.9685**
	noisy-OR	0.7678	0.6994	0.7561	0.7609	0.7262	0.7988
50	naive Bayes	**0.9997**	**0.9993**	**0.9993**	**0.9971**	**0.9929**	**0.9844**
	noisy-OR	0.8091	0.6979	0.7465	0.8354	0.7899	0.6892
60	naive Bayes	**0.9998**	**0.9996**	**0.9994**	**0.9992**	**0.9953**	**0.9892**
	noisy-OR	0.7296	0.7797	0.6630	0.7669	0.7525	0.7884
70	naive Bayes	**0.9999**	**0.9999**	**0.9994**	**0.9992**	**0.9971**	**0.9943**
	noisy-OR	0.7316	0.7177	0.6855	0.7581	0.6768	0.6920
80	naive Bayes	**1.0**	**0.9999**	**0.9997**	**0.9997**	**0.9980**	**0.9962**
	noisy-OR	0.6909	0.7475	0.8084	0.7021	0.8351	0.7367
90	naive Bayes	**0.9998**	**0.9998**	**0.9999**	**0.9998**	**0.9992**	**0.9977**
	noisy-OR	0.7287	0.7928	0.7630	0.8166	0.7140	0.7968
100	naive Bayes	**0.9998**	**1.0**	**1.0**	**0.9999**	**0.9996**	**0.9987**
	noisy-OR	0.728	0.8467	0.8061	0.7471	0.7302	0.8324

Table 1: Learning from data generated by a naive Bayes network

# attributes		Missing data rate 0.0	0.1	0.2	0.3	0.4	0.5
10	naive Bayes	0.8987	0.8214	0.8006	0.8396	0.8687	0.8853
	noisy-OR	**0.9045**	**0.9028**	**0.9079**	**0.9231**	**0.9353**	**0.9408**
20	naive Bayes	**0.8915**	0.7877	0.8009	0.8354	0.8623	0.8844
	noisy-OR	0.7856	**0.8112**	**0.8304**	**0.8913**	**0.9165**	**0.9198**
30	naive Bayes	**0.8963**	0.7838	**0.8157**	0.8305	0.8519	0.8756
	noisy-OR	0.7671	**0.7853**	0.8143	**0.8327**	**0.8549**	**0.8784**
40	naive Bayes	**0.8929**	**0.7975**	**0.8164**	**0.8301**	0.8557	0.8824
	noisy-OR	0.7708	0.7954	0.8151	0.8277	**0.8558**	**0.8836**
50	naive Bayes	**0.8942**	0.7872	0.8193	0.8325	0.8548	0.8791
	noisy-OR	0.7684	**0.7917**	**0.8248**	**0.8363**	**0.8551**	**0.8806**
60	naive Bayes	**0.8934**	0.7878	**0.8101**	0.8404	**0.8548**	**0.8748**
	noisy-OR	0.7636	**0.7963**	0.8066	**0.8420**	0.8543	0.8724
70	naive Bayes	**0.8891**	**0.7943**	0.8138	**0.8354**	0.8546	**0.8825**
	noisy-OR	0.7757	0.7927	**0.8162**	0.8349	**0.8571**	0.8824
80	naive Bayes	**0.8924**	0.7911	**0.8300**	0.8320	0.8502	**0.8873**
	noisy-OR	0.7689	**0.7921**	0.8252	**0.8348**	**0.8541**	0.8860
90	naive Bayes	**0.8868**	0.7926	0.8189	0.8433	**0.8579**	**0.8808**
	noisy-OR	0.7788	**0.7941**	**0.8189**	**0.8435**	0.8567	0.8808
100	naive Bayes	**0.8856**	0.7876	**0.8162**	0.8370	**0.8581**	0.8754
	noisy-OR	0.7750	**0.7894**	0.8151	**0.8371**	0.8564	**0.8760**

Table 2: Learning from data generated by a noisy-or network