Structured Features in Naive Bayes Classification

Arthur Choi
Computer Science Department
University of California, Los Angeles
aychoi@cs.ucla.edu

Nazgol Tavabi
Department of Computer Engineering
Sharif University of Technology
ntavabi@ce.sharif.edu

Adnan Darwiche
Computer Science Department
University of California, Los Angeles
darwiche@cs.ucla.edu

Abstract

We propose the structured naive Bayes (SNB) classifier, which augments the ubiquitous naive Bayes classifier with structured features. SNB classifiers facilitate the use of complex features, such as combinatorial objects (e.g., graphs, paths and orders) in a general but systematic way. Underlying the SNB classifier is the recently proposed Probabilistic Sentential Decision Diagram (PSDD), which is a tractable representation of probability distributions over structured spaces. We illustrate the utility and generality of the SNB classifier via case studies. First, we show how we can distinguish players of simple games in terms of play style and skill level based purely on observing the games they play. Second, we show how we can detect anomalous paths taken on graphs based purely on observing the paths themselves.

Introduction

Naive Bayes (NB) classifiers and their extensions are ubiquitous in domains such as data mining, artificial intelligence, and machine learning (Domingos and Pazzani 1997; Friedman, Geiger, and Goldszmidt 1997; Ng and Jordan 2001). In the classical version of these classifiers, attributes (aka features) are represented as discrete or continuous random variables. For discrete attributes, each variable typically has a small or manageable number of values. For continuous attributes, it is similarly assumed that variables have manageable distributions, such as a Gaussian. As a result, these classifiers are typically used to classify instances that have *simple features*, such as colors or intensities, words or frequencies, scores, etc. In contrast, our interest in this paper is in naive Bayes classifiers with *structured features* that can have an exponential number of values, representing complex entities such as combinatorial objects (e.g., permutations, graphs, paths and game traces).

To realize this objective, we propose the *structured naive Bayes classifier (SNB)*, which augments the NB classifier by allowing it to efficiently accommodate structured attributes. This extension is enabled by two recent advances in knowledge compilation. The first advance is the Sentential Decision Diagram (SDD), which we use to compactly represent attributes that may have an exponential number of values (Darwiche 2011; Xue, Choi, and Darwiche 2012; Choi and Darwiche 2013; Van den Broeck and Darwiche 2015). The second advance is the Probabilistic Sentential Decision Diagram (PSDD), which we use to represent and learn distributions over such attributes (Kisa et al. 2014a; Choi, Van den Broeck, and Darwiche 2015). Our goal is to realize the above objective while maintaining two attractive properties of NB classifiers: (1) the ability to learn and then classify in time that is linear in the size of the classifier, and (2) the ability to use the classifier in a systematic manner across multiple domains.

NB classifiers are defined by a class variable C and a set of simple attributes X_i, each represented by a discrete or continuous variable. In addition, an SNB classifier includes a set of SDDs S_i, each representing a structured attribute. The size of an NB classifier is characterized by the number of classes, the number of attributes, and the number of values that the (discrete) variables X_i can attain. On the other hand, the size of the SNB classifier is further characterized by the number and size of the SDDs S_i. As with NB classifiers, we show that one can learn and classify with SNB classifiers in time that is linear in their size.

We also show that using structured attributes for a particular domain amounts to simply defining the SDD corresponding to each attribute. Once the SDDs are defined, learning from data, and then classifying instances, can all proceed in a systematic and domain independent manner. We illustrate this systematic use of structured attributes through case studies, which delineate the domain specific investment needed. In particular, we consider two different classification tasks:

1. determining the play style and skill level of players in games such as tic-tac-toe and hex, given examples of games that they have played;
2. detecting anomalies in paths taken on graphs, based on observing the paths taken in a normal operating mode.

There are two alternative approaches to classification tasks that are targeted by the SNB classifier. The first is to simply use a classical NB classifier with attributes that have an exponential number of values (using sparse, tabular representations of the corresponding distributions). The second is to use a dedicated approach, that is specific to the domain at hand. We comment on both approaches in the context of

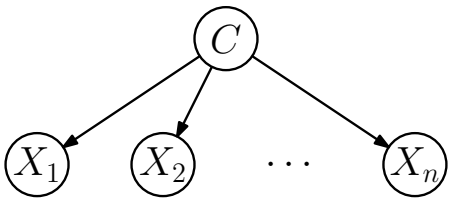


Figure 1: A naive Bayes classifier.

our case studies, arguing for the advantage of using SNB classifiers more broadly.

This paper is organized as follows. We first review the classical NB classifier and then show how structured attributes can be represented by SDDs and their distributions by PSDDs. We follow by proposing the SNB classifier. The final part of the paper illustrates the utility and generality of SNB classifiers via case studies, showing how one can define and represent SNB classifiers, learn them from data, and finally use them for classification.

Naive Bayes Classifiers

A naive Bayes classifier corresponds to a Bayesian network, as in Figure 1. Here, we have a single class variable C and m attribute variables X_i (for simplicity of exposition, we assume that attributes are discrete). Let c denote a class label and x_i denote a value of an attribute X_i. A naive Bayes classifier thus induces a distribution:

$$Pr(c, x_1, \ldots, x_m) = Pr(c) \cdot \prod_{i=1}^{m} Pr(x_i \mid c) \qquad (1)$$

where we have a class prior $Pr(C)$ and conditional distributions $Pr(X_i \mid C)$. We can estimate these parameters from (labeled) data, using maximum likelihood or MAP estimation.[1] Once we have learned a naive Bayes classifier from data, we can label new instances by selecting the class label $c^\star$ that has maximum posterior probability given observations $x_1, \ldots, x_m$. That is, we select

$$c^\star = \operatorname*{argmax}_c Pr(c \mid x_1, \ldots, x_m).$$

The SNB classifier has a structure similar to the NB classifier of Figure 1. The one exception is that the attributes X_i do not need to be discrete (or continuous) variables as they can also be SDDs. The only domain specific investment when using an SNB classifier is to define these SDDs. We will thus illustrate this process next, before we formally define the SNB classifier later.

Representing Structured Attributes

Suppose that we have an attribute whose values are the set of $n!$ permutations (total rankings) over n items. For example, we may want to build a classifier for some demographic, based on preference rankings over food, music or movies. In the standard NB classifier, we would represent this attribute by a discrete variable with $n!$ values.

In our proposal, this attribute is defined by an SDD, which is obtained by a *two-step process.* The first step is based purely on Boolean reasoning, in which we define a Boolean formula whose models correspond to the $n!$ permutations. The second step corresponds to compiling this formula into an SDD. The first step can be thought of as a *modeling* step, and is the one requiring the main domain specific investment. The second step is purely *computational* and is automated to a large extent. The key observation here is that while the attribute may have an exponential number of values, the corresponding SDD may not be exponentially sized.

To illustrate the *first step,* consider the Boolean variables A_{ij} for $i, j \in \{1, \ldots, n\}$. Here, the index i represents an *item* and the index j represents its *position* in a permutation of n items (Choi, Van den Broeck, and Darwiche 2015). There are 2^{n^2} instantiations of our n^2 Boolean variables, many of which do not correspond to valid permutations. In particular, some of these instantiations will place two items in the same position, or place the same item in multiple positions. However, using Boolean constraints, we can rule out all of these invalid instantiations, keeping only the ones that correspond to valid permutations. Assuming $n = 3$ items, the Boolean constraints are as follows:

– Each item i is assigned to exactly one position, leading to three constraints for $i \in \{1, 2, 3\}$:

$$\begin{aligned}&(A_{i1} \wedge \neg A_{i2} \wedge \neg A_{i3})\\ \vee &(\neg A_{i1} \wedge A_{i2} \wedge \neg A_{i3})\\ \vee &(\neg A_{i1} \wedge \neg A_{i2} \wedge A_{i3}).\end{aligned} \qquad (2)$$

– Each position j is assigned exactly one item, leading to three constraints for $j \in \{1, 2, 3\}$:

$$\begin{aligned}&(A_{1j} \wedge \neg A_{2j} \wedge \neg A_{3j})\\ \vee &(\neg A_{1j} \wedge A_{2j} \wedge \neg A_{3j})\\ \vee &(\neg A_{1j} \wedge \neg A_{2j} \wedge A_{3j}).\end{aligned} \qquad (3)$$

Our Boolean formula will then correspond to a conjunction of these six "exactly-one" constraints (in general, when we encode permutations, we have $2n$ "exactly-one" constraints given n items). Moreover, the resulting formula has $6 = 3!$ models. For example, the permutation $(2, 1, 3)$ is represented by the following model:

$$\neg A_{11}, A_{12}, \neg A_{13}, A_{21}, \neg A_{22}, \neg A_{23}, \neg A_{31}, \neg A_{32}, A_{33}.$$

The *second step* is to compile our Boolean formula into a Sentential Decision Diagram (SDD), which is handled by an SDD compiler.[2] Figure 2 (left) depicts an example SDD, which is a Boolean circuit with very specific properties. Here, circles represent disjunctions and paired boxes represent conjunctions.[3]

[1] We can also learn the parameters discriminatively (by maximizing the conditional log likelihood), which corresponds to logistic regression. However, we shall focus here on generative models.

[2] In our experiments, we use the publicly available SDD compiler at `http://reasoning.cs.ucla.edu/sdd`.

[3] The properties of an SDD allow certain operations, which are hard on arbitrary Boolean formulas, to be performed efficiently on the corresponding SDD. For example, model counting can be performed using SDDs in time that is linear in the size of the SDD (Darwiche 2011).

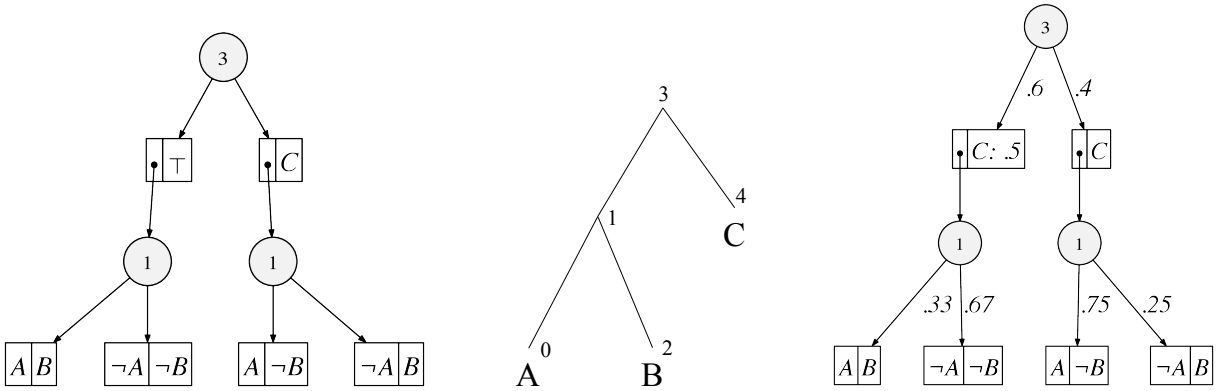


Figure 2: An SDD, a vtree, and a PSDD for the Boolean formula $(A \Leftrightarrow B) \vee ((A \Leftrightarrow \neg B) \wedge C)$.

There are many SDDs that are equivalent to a given Boolean formula, yet the SDD is canonical once a *vtree* is fixed. This is a binary tree whose leaves correspond to the Boolean variables of the formula; see Figure 2 (center). The SDD compiler we used tries to find the vtree leading to the smallest possible SDD. As we shall see later, this amounts to minimizing the number of parameters needed to define a distribution over the attribute encoded by an SDD.

Before we discuss SDD parameterization, we make note of the following point. While the process of compiling a Boolean formula into an SDD is automated, it is the one and only step in our approach that can fail in practice; that is, the compiler may fail to compile the formula. While this is unavoidable in general, given the hardness of compilation, the process can be aided by choosing a specific, good vtree, instead of relying on the compiler to find it. We will later mention the specific vtrees we used in our case studies.

We are now ready to discuss the final technical ingredient needed for defining the SNB classifier. An SDD represents the state space of an attribute, i.e., its values. To induce a probability distribution over this space, we will appeal to *Probabilistic Sentential Decision Diagrams* (PSDDs) (Kisa et al. 2014a). A PSDD is obtained by simply parameterizing the edges and leaves of an SDD; see Figure 2 (right). We make a few observations here. First, the number of parameters needed to induce a distribution over the models of an SDD (i.e., values of corresponding attribute) is proportional to the size of the SDD, not the number of its models (i.e., not the number of attribute values). Second, any distribution over a set of discrete variables can be represented by a PSDD (i.e., the PSDD is a complete representation). Third, computing marginals and most-likely instantiations can be done in time linear in the PSDD size. Finally, the maximum likelihood parameters of a PSDD can be learned in closed-from from a complete dataset. For more on PSDDs, please see (Kisa et al. 2014a; Choi, Van den Broeck, and Darwiche 2015).

Structured Naive Bayes Classifiers

A classical NB classifier is defined by a class variable C and attributes X_i. Moreover, its parameters include a prior distribution $Pr(C)$ for the class variable C, and a set of conditional distributions $Pr(X_i \mid c)$, one for each class label c and attribute X_i. One estimates these parameters from labeled data, corresponding to a set of examples of the form $c, x_1, \ldots, x_m$, where c is a class label, and each x_i is an observed attribute value.

The SNB classifier is defined similarly, except that one can also have structured attributes that are represented by SDDs and whose conditional distributions are represented by PSDDs. For simplicity of exposition, we only include structured attributes in the following definition, but one can also include simple attributes as we show later.

Definition 1 *A Structured Naive Bayes (SNB) classifier is composed of:*

- *a discrete variable C representing a class,*
- *SDDs $S_1, \ldots, S_n$ representing structured attributes.*

The parameters of an SNB classifier are defined as follows:

- *for the class variable C, there is a class prior θ_C,*
- *for each class label c and structured attribute j, there is a set of parameters $\theta_{j,c}$ for the SDD S_j, inducing a PSDD for the corresponding conditional distribution.*

To define the semantics of an SNB classifier, let $\mathbb{P}_{j,c}$ denote the distribution induced by the PSDD obtained from SDD S_j and its parameters $\theta_{j,c}$. Moreover, let s_j denote a model of SDD S_j. We then have

$$Pr(c, s_1, \ldots, s_n) = \theta_c \cdot \prod_{j=1}^{n} \mathbb{P}_{j,c}(s_j). \tag{4}$$

Hence, if the class variable has k values, then the SNB classifier will have $k \times n$ PSDDs.

It is straightforward to also include simple attributes represented by discrete and continuous variables. For example, if we include m discrete attributes X_i with parameters $\theta_{x_i|c} = Pr(x_i \mid c)$, we obtain the following distribution:

$$Pr(c, x_1, \ldots, x_m, s_1, \ldots, s_n) = \theta_c \cdot \prod_{i=1}^{m} \theta_{x_i|c} \cdot \prod_{j=1}^{n} \mathbb{P}_{j,c}(s_j).$$

Compare this form with that of Equations 1 & 4. To simplify further discussions, we will next assume that all attributes of an SNB are structured, and hence represented by SDDs.

Data. Like NB classifiers, an SNB classifier is also learned from labeled data, except that our examples now have the form $c, s_1, \ldots, s_n$. Here, c is a class label. Moreover, each s_j is a model of SDD S_j and corresponds to an instantiation of the variables over which the SDD S_j is defined. In other words, each s_j represents a particular value of the structured attribute represented by SDD S_j.

Training. Let $\mathcal{D}$ denote a given dataset as discussed above and let $\mathcal{D}^c$ denote the subset of the dataset $\mathcal{D}$ with class labels c. As in traditional NB classifiers, we estimate the class parameters θ_c, which capture the prior $Pr(C)$, based on the relative proportions of the class labels in the dataset $\mathcal{D}$. Moreover, we estimate the parameters $\theta_{j,c}$, which capture the conditional distributions $\mathbb{P}_{j,c}(.)$, from the dataset $\mathcal{D}^c$ using the PSDD parameter estimation algorithm (Kisa et al. 2014a). We note that ML and MAP parameters can be estimated in closed-form using a single pass over the dataset $\mathcal{D}^c$, as in traditional NB classifiers. The following proposition more formally summarizes the complexity of training an SNB classifier.

Proposition 1 *Given a dataset $\mathcal{D}$ of size N, with k class labels and n attributes, where each attribute is encoded as an SDD of size $O(s)$, we can learn an SNB classifier with $O(kns)$ parameters in time $O(Nkns)$.*

Classification. Given an unlabeled example $s_1, \ldots, s_n$, we assign to it a class label c that maximizes the probability:

$$Pr(c \mid s_1, \ldots, s_n) \propto \theta_c \cdot \prod_{j=1}^{n} \mathbb{P}_{j,c}(s_j).$$

This requires that we evaluate the conditional probability $\mathbb{P}_{j,c}(s_j)$. For NB classifiers with simple attributes, the corresponding computation amounts to a simple table lookup. For SNB classifiers, we need to evaluate a PSDD, which can be done in time that is linear in the PSDD size (Kisa et al. 2014a). To compute the probability $Pr(c \mid s_1, \ldots, s_n)$, we therefore need to perform n PSDD evaluations, one for each attribute of our SNB classifier. The following proposition summarizes the complexity of SNB classification.

Proposition 2 *Given an SNB classifier with k class labels and n attributes, where each attribute is encoded as an SDD of size $O(s)$, we can classify an unlabeled example in time $O(kns)$.*

Case Study: Learning from Game Traces

A game can be viewed as a structured attribute, whose values correspond to all game outcomes (traces). Moreover, an ability to represent distributions over game traces allows one to perform player modeling (Billings et al. 1998), by viewing a player as a distribution over the games that they play. Going further, we can view a particular style of game play, or even a particular level of skill, as a distribution over games. Hence, given the ability to represent distributions over games, we can seek to classify a particular player in terms of style or skill by simply observing the games that they play.

To portray the systematic nature of working with SNB classifiers to address this and similar problems, we consider the simple game of tic-tac-toe as an illustrative example. Here, we want to build a classifier that determines the play style of a player given examples of tic-tac-toe games that they have played. In the Appendix, we also consider the classification of player skill in the game of Hex.

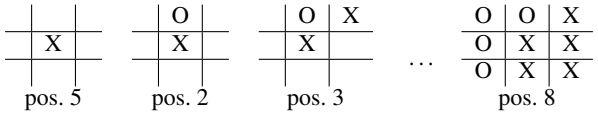
Briefly, tic-tac-toe is played on a 3×3 grid, where two players alternate, playing their symbol (either an "X" or an "O") on the board. The first player to place three of their symbols in a row (by row, column or diagonal) is the winner of the game. Consider the following tic-tac-toe board, where each position has been labeled by an index from 1 to 9:

1	2	3
4	5	6
7	8	9

To learn from traces of tic-tac-toe games, we first need to represent these game traces as structured attributes. This process is simplified, if we assume that a game is played until the board is completed, and not just when the first three-in-a-row is obtained. Under this assumption, we can view a tic-tac-toe game as a permutation of the board positions. For example, the following vector of 9 elements represents a game of tic-tac-toe:

$$(5, 2, 3, 7, 6, 4, 9, 1, 8)$$

where the i-th element represents the move played in turn i:



Under this representation, there are $9! = 362,880$ distinct tic-tac-toe game traces (we ignore board symmetries, here).

Representing Games. We showed in an earlier section how permutations over n items can be represented as an SDD. The only (semantical) difference here is that a Boolean variables A_{ij} represents the event that a position i is played at turn j. Moreover, in this case, we compiled the constraints of Equations 2 & 3 (for permutations with 9 elements) using a publicly available SDD compiler. In our experiments, we used a vtree that is found by (1) grouping all variables A_{ij} for each turn j into a right-linear sub-vtree, ordered by position, (2) constructing a right-linear vtree over these sub-vtrees, ordered by turn. Indeed, this is the only domain specific investment that we need, to address this problem using an SNB classifier. The remaining steps are systematic and shared by the use of SNB classifiers across other domains.

Data. Suppose that we have a labeled dataset $\mathcal{D}$ where each example consists of a tic-tac-toe game (i.e., a permutation) σ, and a class label c. In this case, the class label is a pair (c_1, c_2), which denotes the play styles of the first and second player. Hence, a labeled example is of the form $(c_1, c_2), \sigma$, where σ is a permutation encoded as an instantiation of variables A_{ij}.

Training. As in traditional NB classifiers, we estimate the class prior $Pr(C_1, C_2)$ based on the relative proportions of the class labels. To estimate the conditional distributions

$Pr(\sigma \mid c_1, c_2)$, we learn the corresponding PSDD distribution as discussed earlier. In particular, we learned MAP estimates as in (Kisa et al. 2014a).

Classification. Given a pair of tic-tac-toe players and m example games σ_i that the pair have played, we want to assign them a class label $c = (c_1, c_2)$ which best describes their play styles, by maximizing the probability:

$$Pr(c_1, c_2 \mid \sigma_1, \ldots, \sigma_m) \propto Pr(c_1, c_2) \prod_{i=1}^{m} Pr(\sigma_i \mid c_1, c_2).$$

We can also classify the first player alone, by marginalizing out the second player, and maximizing the probability:[4]

$$Pr(c_1 \mid \sigma_1, \ldots, \sigma_m) = \sum_{c_2} Pr(c_1, c_2 \mid \sigma_1, \ldots, \sigma_m).$$

We note that we did not use simple attributes here (say over discrete variables, as in a classical NB classifier), although we could have (e.g., to increase classification accuracy further). Also, the conditional distributions $Pr(\sigma \mid c_1, c_2)$ are shared among the different observed games σ_i.

Empirical Evaluation. To obtain tic-toe-toe games of varying but known styles, we simulated games from different tic-tac-toe programs. We considered three types:

- **random**: a player that makes moves at random;
- **simple**: a player that uses heuristic rules;[5]
- **optimal**: an optimal player, guaranteeing at least a draw.[6]

We considered all $3 \times 3 = 9$ combinations of first and second players. To represent each game as a permutation, we completed games that did not fill the board in a fixed way (unplayed squares were played in order by index).

We simulated training sets of 2^{16} games for each pair of player types. We thus learned 9 PSDDs for each conditional distribution $Pr(\sigma \mid c)$ using the same SDD structure. The resulting PSDDs had $3,328$ nodes and $1,793$ parameters.

We evaluated our SNB classifier based on independent testing sets, using simulated games for the players of each pair of playing styles. Figure 3 illustrates the results of classifying the *first* player's style in tic-tac-toe as described earlier (we obtained comparable results for the second player). On the x-axis, we increased the number of games available to classify the style of the first player. For each style of the first player, we simulated 2^{10} sets of games for each style of the second player. Hence, each plot point is an average of $3 \cdot 2^{10}$ sets of games. First, consider the SNB classifier with PSDDs to represent games (plotted with solid lines). One game is clearly not sufficient to classify the type of a player: a random player, for example, could by chance play a game optimally. However, after observing enough games from the same player, we can classify their play style with high accuracy.

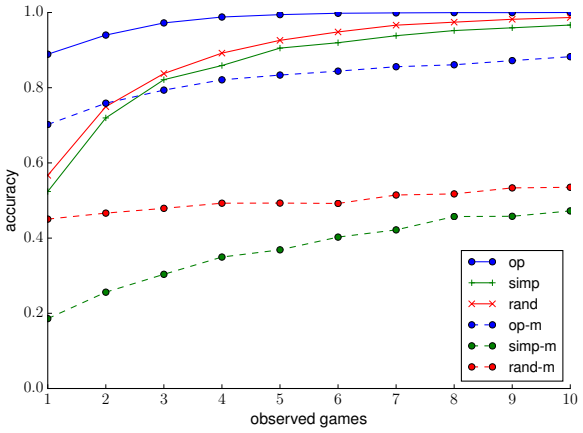


Figure 3: Classification accuracy in tic-tac-toe. Solid lines denote an SNB classifier with PSDDs. Dashed lines denote an NB classifier with Mallows models.

Alternative Approaches. To learn a traditional NB classifier for this problem, we need to learn the distribution over the class labels $Pr(c_1, c_2)$ and the conditional distributions over games $Pr(\sigma \mid c_1, c_2)$. A tabular representation of this conditional distribution would however require $9! - 1 = 362,879$ parameters, which is neither practical to represent nor to learn (contrast this to the $1,793$ parameters needed by a PSDD). Other (specialized) models for representing permutations, such as the Mallows model (Mallows 1957), do not appear well suited for representing games of tic-tac-toe. For one, the Mallows model has a unimodal shape, whereas tic-tac-toe has multiple symmetries, hence leading to multimodal distributions. We did indeed evaluate the Mallows model to represent these conditional distributions, by replacing the PSDDs in the above experiments. The classification results were much worse as shown in Figure 3. The Mallows model is likely *underfitting* the data, i.e., its underlying assumptions are too strong for representing tic-tac-toe games (as discussed earlier).

Case Study: Detecting Anomalous Paths

One common use of naive Bayes models is in anomaly and outlier detection; see, e.g., (Chandola, Banerjee, and Kumar 2009; Hamerly and Elkan 2001). For example, we may want to detect abnormal patterns in automobile traffic, which could be indicative of a traffic accident, or other type of emergency. We may also want to detect abnormal traffic in a packet switching network, which could be indicative of, say, a network intrusion by a computer worm (Dash et al. 2006).

We can view the route taken by a car or network packet as a path that connects two points on a graph. See, for example, Figure 4, where we have a source node s on the upper-left, and a destination node t on the lower-right. For an automobile driving on city streets, the path illustrated on the left may be considered normal, whereas the one on the right may be considered abnormal, as there was a detour en route to the

[4] It is also possible to classify the first player alone, given second players of different styles, although this requires another extension of the NB classifier, which we do not consider here.

[5] If a player can win or block a loss in their turn, they make the corresponding play; otherwise, they play a square at random.

[6] We used a player which uses brute-force enumeration to find an optimal move, at `http://code.activestate.com/recipes/578563/` (which contained a bug that we fixed).

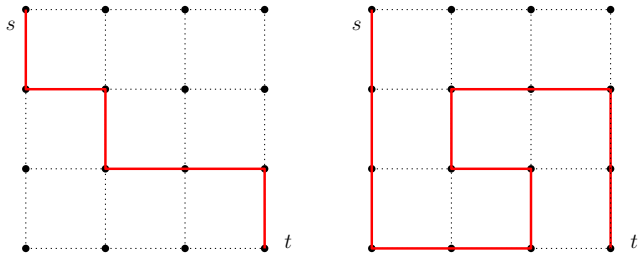


Figure 4: Paths on a 4×4 grid: a normal path (left) and an anomalous one (right).

destination.

By viewing automobile or network traffic as a distribution over paths on a graph, we can seek to detect anomalous traffic using an SNB classifier. Here, we have two class labels: "normal" and "anomalous". As in semi-supervised approaches to anomaly detection, we can learn a "normal" distribution over paths, and assume, say, a uniform distribution over paths as "anomalous". The basic idea here is to detect paths that do not follow the "normal" distribution.

Representing Paths. Suppose we are given an undirected graph $G = (V, E)$ with a fixed source node s and a fixed destination node t. We first want to define a structured attribute (as an SDD) whose values are all simple paths connecting s to t on the graph G. Here, a simple path refers to a path that does not revisit any node. To achieve this objective, we will follow the two-step process we discussed earlier: define a Boolean formula whose models correspond to simple paths, then compile the formula into an SDD.

For the first step, we define Boolean variables $A_{i,j}$, one for each edge $\{i, j\} \in E$. We want to construct a formula α where each model of α corresponds to a unique path connecting s to t. If a model sets variable $A_{i,j}$ to true, then the (s, t)-path of the model uses the edge $\{i, j\}$; if variable $A_{i,j}$ is false, then it is not used. For example, consider a 2×2 grid where the source s is the upper-left corner (labeled 1) and the sink t is the lower-right corner (labeled 4), with the other two corners labeled 2 and 3. In this case, we have two simple paths: $1 - 2 - 4$ and $1 - 3 - 4$. Our desired formula α also has two models, given by:

$$(A_{12} \wedge \neg A_{13} \wedge A_{24} \wedge \neg A_{34}) \vee (\neg A_{12} \wedge A_{13} \wedge \neg A_{24} \wedge A_{34}).$$

For a general graph $G = (V, E)$, we can construct our formula α recursively based on the formulas of sub-graphs. Let $G_{\setminus s}$ denote the graph where sink node s and its incident edges have been removed from G. Let $\alpha^G_{s,t}$ denote a formula representing the set of all simple paths connecting s to t in graph G. We can express $\alpha^G_{s,t}$ as a recurrence:

$$\alpha^G_{s,t} = \bigvee_{\{s,j\} \in E} \left[A_{s,j} \wedge \alpha^{G_{\setminus s}}_{j,t} \wedge \bigwedge_{\{s,k\} \in E : k \neq j} \neg A_{s,k} \right].$$

To obtain all paths connecting s to t in G, we consider all of the possible first edges $\{s, j\} \in E$ in our path, and then recursively find all paths from j to t, but in the smaller graph $G_{\setminus s}$. This leads to three sub-formula, where we:

1. set $A_{s,j}$ positively, i.e., we use the edge $\{s, j\}$,
2. recursively find all paths $\alpha^{G_{\setminus s}}_{j,t}$ connecting j to t in $G_{\setminus s}$,
3. set all other $A_{s,k}$ negatively, i.e., no other edges on the path can go through s (since paths are simple).

Our base case corresponds to the case where $s = t$:

$$\alpha^G_{s,s} = \bigwedge_{\{i,j\} \in E} \neg A_{i,j}$$

where only an empty path is allowed (since we cannot revisit a node, so no edges can be used). This concludes the first step of our two-step process as we now have a Boolean formula whose models correspond to the values of our structured attribute (i.e., simple paths).

The second step is compiling the formula $\alpha^G_{s,t}$ into an SDD using an SDD compiler as we did in the earlier case study. However, for the special case of OBDDs (which corresponds to an SDD with a right-linear vtree), a particularly efficient approach is summarized in (Minato 2013) based on Knuth's Simpath algorithm (Knuth 2009). For our experiments, we used the GRAPHILLION library to construct a ZDD for our formula (Inoue et al. 2014), which was converted into an SDD using a right-linear vtree (with the same ordering). As in the previous case study, this concludes the domain specific investment needed for handling this particular problem. The rest are systematic steps when using the SNB classifier approach, shared across other domains.

Empirical Evaluation. We now illustrate how the SNB classifier can perform anomaly detection in a scenario similar to the one depicted in Figure 4. We consider paths in 8×8 grid graphs, connecting the upper-left and lower-right corners. The resulting PSDD has 26,884 unique parameters, in contrast to the 789,360,053,252 possible simple paths, which would be too large to represent using a tabular representation (traditional NB classifier). We simulate training data based on the following iterative process: starting at node s, we randomly select an edge, either going right or down (when possible), that brings us closer to t. We repeat until we reach node t. We simulated training sets of size $2^8, 2^9, 2^{10}, 2^{11}$ and 2^{12}, for the "normal" distribution. We independently simulated testing datasets for the "normal" and "anomalous" cases, each of size 2^{12} (again, we assume a uniform distribution for the "anomalous" case). For learning PSDDs, we assumed Dirichlet priors with exponents 2 (corresponding to Laplace smoothing). We also learned an NB classifier, using a sparse tabular representation of the conditional distributions (also using Laplace smoothing).

Below, we report the proportion of the test instances which were correctly classified as anomalous or not.

training set size	2^8	2^9	2^{10}	2^{11}	2^{12}
SNB accuracy	98.97%	99.38%	99.58%	99.75%	99.87%
tabular accuracy	59.09%	65.34%	72.08%	79.98%	87.80%

We remark that the SNB classifier obtains relatively high accuracies, even with relatively small training sets. This further indicates the suitability of the PSDD as a representation for the process used to generate the "normal" dataset. In particular, in our generating process, the distribution over paths connecting an internal node i, and the destination t,

is (roughly) independent of how we arrived at node i. The PSDD is particularly suitable for handling these types of conditional independencies (Kisa et al. 2014a).

Related Work

There are a few notable examples where probabilistic models have been adapted to support complex features that are not easily representable using, e.g., discrete variables or Gaussians. For example, logistic regression models and some of their extensions have been adapted with convolutional neural networks to extract features from images (Deng et al. 2014; Krizhevsky, Sutskever, and Hinton 2012). Similarly, hidden Markov models have been adapted with deep neural networks to extract features from acoustic inputs (Hinton et al. 2012).

As a tractable probabilistic model, the PSDD is related to the Arithmetic Circuit (AC) representation of probability distributions (Darwiche 2002; 2003). ACs are also a tractable representation and their size can be quite small for some distributions whose graphical models have a high treewidth (Chavira and Darwiche 2008). A number of approaches have previously been proposed for learning ACs from data (Lowd and Domingos 2008; Lowd and Rooshenas 2013; Bekker et al. 2015). ACs with latent variables are also sometimes called sum-product networks (SPNs) (Poon and Domingos 2011; Gens and Domingos 2012; Rooshenas and Lowd 2014).

One of the primary differences between PSDDs and the above representations is the ability of PSDDs to operate on a structured probability space: one that corresponds to a user-specified subset of variable instantiations. The representations above all operate on unstructured spaces, which correspond to all instantiations of a set of variables. This difference is crucial for representing combinational objects, as we did in this work, and for learning distributions over them. Moreover, while the above representations are general and expressive, they can be prone to overfitting (Srivastava et al. 2014) unless we have access to massive amounts of data. Previous empirical studies have provided support that this distinction (structured versus unstructured spaces) can have a significant impact on the statistical efficiency of learning (Kisa et al. 2014b). Another notable advantage of PSDDs is the tractability of learning them, given closed-form estimates of MAP/ML parameters from complete data (Kisa et al. 2014a). Recently, closed-form estimates were also proposed for a restricted class of SPNs (Peharz, Gens, and Domingos 2014).

Conclusion

We proposed the structured naive Bayes (SNB) classifier, which extends the ubiquitous naive Bayes classifier. While naive Bayes classifiers are limited to simple attributes, SNB classifiers can also support structured attributes, such as combinatorial objects, in a general and systematic way. We illustrated the utility and effectiveness of this approach using two case studies, which delineate the domain-specific investment needed when using SNB classifiers. First, we illustrated how to classify the play style and skill level of a game player, based on the traces of games that they play. Second, we illustrated how to detect anomalous paths on graphs, by learning from, and classifying, the paths themselves.

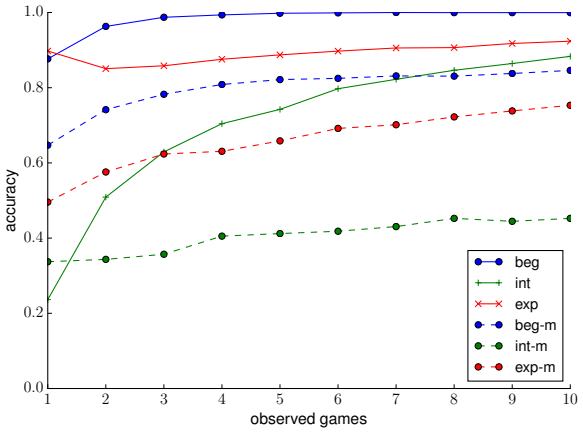


Figure 5: Classification accuracy in 3×3 Hex. Solid lines denote an SNB classifier with PSDDs. Dashed lines denote an NB classifier with Mallows models.

Acknowledgments

This work has been partially supported by ONR grant #N00014-12-1-0423 and NSF grant #IIS-1514253.

Appendix: Learning from Game Traces (Hex)

We also considered the board game Hex, played on an $n \times n$ hexagonal grid. The players take turns, playing a single stone on an initially empty grid. Each player is associated with opposing ends of the board. The winner is the first player to connect their ends of the board with their stones.

In this case, we consider the problem of learning the *skill level* of a Hex player. We consider a computer Hex player, which is based on Monte Carlo tree search.[7] Here, the skill level of a player corresponds to the number of samples that the program is allowed, with more samples corresponding to a higher skill. We considered a "beginner" player (a random player), an "intermediate" player, and an "expert" player, with the "expert" player receiving 5 times as many samples as the intermediate one.[8]

We assumed that the board is completed even after the game is won (as we did in tic-tac-toe). Each Hex game can then be viewed as a permutation over the $n \times n$ positions in the hexagonal grid. In our experiments, we consider the simple case of 3×3 Hex. Hence, we can use precisely the same SDD representation of games in 3×3 Hex as we used earlier for tic-tac-toe. Moreover, we have analogous processes for training an SNB classifier and for classifying with it. Our main goal here is to show that it is possible to classify the

[7] Available at `https://github.com/dpearson/Hex/`.

[8] In the terms of the program used, the "intermediate" player had skill level 30, and the "expert" player had skill level 150. The "skill level" was determined by examining the winning rates as skill levels were varied.

skill level of a game player using SNB classifiers, in addition to classifying play styles as we did in tic-tac-toe.

Figure 5 highlights the results of classifying the *first* player's style in 3×3 Hex (we obtained comparable results for the second player). On the x-axis, we increase the number of observed games used to classify the first player. Each point is an average of $3 \times 1,000 = 3,000$ sets of observations, $1,000$ each for the 3 levels of second players. Consider first the SNB classifier with PSDDs (solid lines). As we increase the number of games observed, we get more accurate classifications of a player's skill. Here, we see that a "beginner," who plays randomly, is easiest to classify, requiring only a few games to obtain a high level of accuracy.[9] The "intermediate" and "expert" players, however, are harder to distinguish as we need to observe more games before we obtain a higher level of confidence in their skill. Again, the NB classifier with the Mallows model (dashed lines) does not appear to be a good representation for Hex games. As with tic-tac-toe, Hex has board symmetries which leads to multimodal distributions. This does not appear to be a good fit for the Mallows model as it has a unimodal shape.

References

Bekker, J.; Davis, J.; Choi, A.; Darwiche, A.; and Van den Broeck, G. 2015. Tractable learning for complex probability queries. In *Advances in Neural Information Processing Systems 28 (NIPS)*.

Billings, D.; Papp, D.; Schaeffer, J.; and Szafron, D. 1998. Opponent modeling in poker. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, 493–499.

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41(3).

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6–7):772–799.

Choi, A., and Darwiche, A. 2013. Dynamic minimization of sentential decision diagrams. In *Proceedings of the 27th Conference on Artificial Intelligence (AAAI)*.

Choi, A.; Van den Broeck, G.; and Darwiche, A. 2015. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

Darwiche, A. 2002. A logical approach to factoring belief networks. In *Proceedings of KR*, 409–420.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *J. ACM* 50(3):280–305.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of IJCAI*, 819–826.

Dash, D.; Kveton, B.; Agosta, J. M.; Schooler, E. M.; Chandrashekar, J.; Bachrach, A.; and Newman, A. 2006. When gossip is good: Distributed probabilistic inference for detection of slow network intrusions. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, 1115–1122.

Deng, J.; Ding, N.; Jia, Y.; Frome, A.; Murphy, K.; Bengio, S.; Li, Y.; Neven, H.; and Adam, H. 2014. Large-scale object classification using label relation graphs. In *ECCV*, 48–64.

[9] Some first moves allow the second player to force a win, which contributes to the ease of detecting a random player. In contrast, in tic-tac-toe, any initial move can be forced into at least a draw.

Domingos, P. M., and Pazzani, M. J. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29(2-3):103–130.

Friedman, N.; Geiger, D.; and Goldszmidt, M. 1997. Bayesian network classifiers. *Machine Learning* 29(2-3):131–163.

Gens, R., and Domingos, P. M. 2012. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems 25 (NIPS)*, 3248–3256.

Hamerly, G., and Elkan, C. 2001. Bayesian approaches to failure prediction for disk drives. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, 202–209.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; and Kingsbury, B. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

Inoue, T.; Iwashita, H.; Kawahara, J.; and Minato, S.-i. 2014. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer* 1–10.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014a. Probabilistic sentential decision diagrams. In *KR*.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014b. Probabilistic sentential decision diagrams: Learning with massive logical constraints. In *ICML Workshop on Learning Tractable Probabilistic Models (LTPM)*.

Knuth, D. E. 2009. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS)*, 1106–1114.

Lowd, D., and Domingos, P. M. 2008. Learning arithmetic circuits. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI)*, 383–392.

Lowd, D., and Rooshenas, A. 2013. Learning Markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 406–414.

Mallows, C. L. 1957. Non-null ranking models. *Biometrika*.

Minato, S. 2013. Techniques of BDD/ZDD: brief history and recent activity. *IEICE Transactions* 96-D(7):1419–1429.

Ng, A. Y., and Jordan, M. I. 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *NIPS*, 841–848.

Peharz, R.; Gens, R.; and Domingos, P. 2014. Learning selective sum-product networks. In *LTPM workshop*.

Poon, H., and Domingos, P. M. 2011. Sum-product networks: A new deep architecture. In *UAI*, 337–346.

Rooshenas, A., and Lowd, D. 2014. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, 710–718.

Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR* 15(1):1929–1958.

Van den Broeck, G., and Darwiche, A. 2015. On the role of canonicity in knowledge compilation. In *AAAI*.

Xue, Y.; Choi, A.; and Darwiche, A. 2012. Basing decisions on sentences in decision diagrams. In *AAAI*, 842–849.