8

Bayesian Networks as Classifiers

You receive an email and wish to determine whether it is spam; you see a bird and wish to determine its species; you examine a patient and wish to diagnose him. These are only a few examples of the very common human task of classification.

Formally, you have a set of variables, $\{F_1, \ldots, F_n\}$, called features (or attributes) and a *class variable*, C, where the states of C correspond to the possible classes. For the bird example above, the feature variables would encode various characteristics of the bird, and the class variable would represent the possible species. Since it often happens that some feature values are not known, feature variables are often extended with state "?" for unknown (or "missing value"). A case is said to be *complete* if there are no missing values. A case set is said to be *consistent* if two complete cases with the same values on the features are of the same class.

A *classifier* is a function from $F_1 \times \cdots \times F_n$ to C. We shall deal only with classification tasks over a finite set of classes and with discrete features.

If you have a Bayesian network model, it can be used for classification. In fact, if there is only one hypothesis variable, the network is a model for classification. In the pregnancy model (Section 3.1.3), for example, test information is used to classify the state of the cow, the class being the state of highest probability.

In this chapter we consider learning of classifiers. Let $\mathcal{D}$ be a data set of cases over features $\{F_1, \ldots, F_m\}$ and class variable C; we do not require the data set to be consistent. We wish to use the data set for constructing a classifier. If the space of feature configurations is small and the amount of data is relatively large, you may use the data set to establish a look-up table: given a complete case $\mathbf{f}$ of features, look up $\mathbf{f}$ in the data set. If there are cases in $\mathcal{D}$ with feature values $\mathbf{f}$, then return the majority class value. If $\mathbf{f}$ is not present in $\mathcal{D}$, then return the most frequent class value in $\mathcal{D}$. However, this method is tractable only for small configuration sets; even with a moderate number of feature variables you will need a more compact representation of

the classification function. Any other method for learning classifiers should predominantly produce better classifiers.

8.1 Naive Bayes Classifiers

Consider the poker game model introduced in Section 3.2.3, and extend the model with a variable for my hand (*MH*) and for best hand (*BH*) (see Exercise 3.14). A Bayesian network model would be like the one in Figure 8.1.

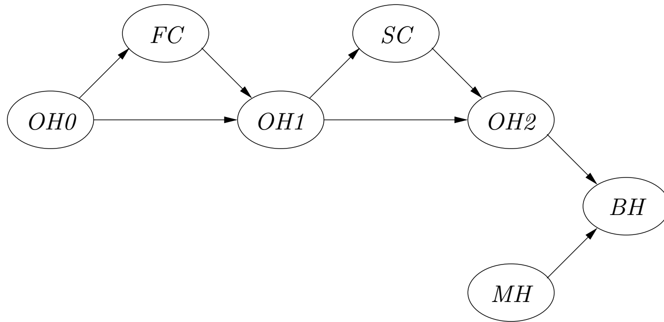

Fig. 8.1. A Bayesian network for the poker game extended with a node for my hand and best hand.

Assume that you have a set of cases over the observable features *MH*, opponent's change of cards, *FC*, *SC*), and the class (*BH*). Exploiting structural learning will most likely result in the model in Figure 8.2. The reader may test this by a manual run of the PC algorithm on Figure 8.1 with the variables (*OHi*) hidden.

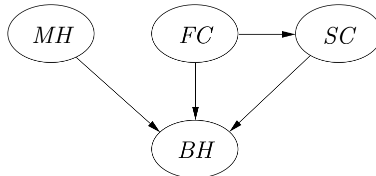

Fig. 8.2. A Bayesian network learned from a case set of poker games.

The model in Figure 8.2 does not provide a compact representation of the classification function, since the class variable has all features as parents, and therefore the conditional probability table for the class variable is as large as a look-up table for the classification problem. Unfortunately, it is often seen

in connection to Bayesian network classifiers that in the correct model, the class variable has (almost) all feature variables as parents, and the network therefore becomes intractably large. Instead, you can insist on working with a class of simpler structures and search for the model that best approximates the correct structure.

One such class of models could be naive Bayes networks (see Section 3.1.5), and for the poker game, the structure will be the one in Figure 8.3.

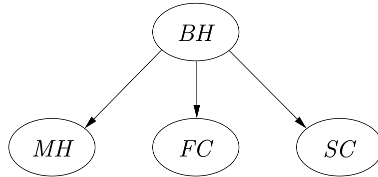

Fig. 8.3. Naive Bayes structure for the poker game.

In general, in a *naive Bayes classifier* (NBC) each feature variable has the class variable as its only parent. This means that the structure is fixed, and the only task involved in learning is to estimate the parameters.

The parameters for an NBC are easily determined by the methods presented in Chapter 6. If the cases are complete, you can determine a maximal likelihood model through simple counting. If a case contains missing values, the EM algorithm can be used; equivalently, disregard that case for the attributes that are missing.

All methods for learning classifiers from data have a problem with very rare cases, which may not be represented in the data set. Assume, for example, that the data set for learning a poker classifier does not contain a case in which I have lost with a hand with *3v*. If one is not careful, the classifier would deem this impossible regardless of the pattern of card changes. For a Bayesian network classifier, this problem corresponds to incorrectly setting a parameter to zero. To avoid zero values for parameters, you may simulate Bayesian learning by introducing virtual cases. An easy way of handling this is initially to give all parameters a small positive count.

Since NBCs are easy to learn, and easy to use as classifiers, and since they are very flexible with respect to missing values, they are very widespread. As mentioned in Section 3.1.5, NBCs assume the features to be independent given the class, and even though this is rarely the case, NBCs have proved surprisingly precise. A reason for this is that when doing classification we are interested only in the class of maximal probability and not in the exact probability distribution over the classes.

8.2 Evaluation of Classifiers

Assume that you have a classifier, *Clsf*, and a data set of cases covering the feature variables and the class variable. We wish to characterize the quality of *Clsf*. A way of characterizing *Clsf* is to calculate its *classification accuracy*: the fraction of correctly classified cases.

A more detailed description of a classifier would be to calculate the *confusion matrix*, $P^{\#}(\text{Classified value}, \text{Correct value})$. In addition to the confusion matrix you can also introduce a value for how bad a misclassification is, and thereby establish a *loss matrix*, describing a punishment for the various kinds of misclassification.

To illustrate this, consider again the poker game. Assume that you have established a classifier *Pcl*, and you have the set of cases in Table 8.1. Since 12 out of 20 cases are classified correctly, the classification accuracy is 0.6.

Case number:	*BH*	*MH*	*FC*	*SC*	*Pcl*
1	op	no	3	1	op
2	op	1a	2	1	op
3	draw	2 v	1	1	op
4	me	2 a	1	1	me
5	draw	fl	1	1	me
6	me	st	3	2	me
7	me	3 v	1	1	me
8	me	sfl	1	0	me
9	op	no	0	0	op
10	op	1 a	3	2	me
11	draw	2 v	2	1	op
12	me	2 v	3	2	draw
13	op	2 v	1	1	draw
14	op	2 v	3	0	op
15	me	2 v	3	2	me
16	draw	no	3	2	draw
17	draw	2 v	1	1	draw
18	op	fl	1	1	me
19	op	no	3	2	op
20	me	1 a	3	2	op

Table 8.1. Test cases for a poker classifier. The entry *Pcl* is the class value provided by the classifier.

The confusion matrix is given in Table 8.2, but it does not consider the stakes involved in the poker game. Let the situation be that both players initially have bet a euro, and you have to decide whether to *fold* (your opponent takes the pot) or to *call*. To simplify, assume that you place a euro when you call, and your opponent is forced to place a euro. The winner takes the pot,

		BH		
		me	draw	op
Plc	me	0.25	0.05	0.1
	draw	0.05	0.1	0.05
	op	0.05	0.1	0.25

Table 8.2. Confusion matrix for the poker classifier. The sum of the diagonal elements is the classification accuracy.

and in the case of a draw you share the pot. The wins and losses in the various situations are given in Table 8.3.

		BH		
		me	draw	op
Action	fold	0	0	0
	call	3	1	−1

Table 8.3. Wins and losses in the poker game.

Based on Table 8.3, you decide on the strategy to call if and only if the classifier says *m* or *draw*. The loss matrix tells you what you lose by following the classifier compared to a situation with certainty on BH. It is given in Table 8.4.

		BH		
		me	draw	op
Plc	me	0	0	−1
	draw	0	0	−1
	op	−3	−1	0

Table 8.4. Loss matrix for the poker classifier.

The confusion matrix and the cost matrix can now be used to calculate the expected loss of a strategy following the classifier (based on the data set $\mathcal{D}$):

Expected loss

$$= \sum_{\text{Classified},\text{Correct}} P^{\#}(\text{Classified} \mid \text{Correct}) P^{\#}(\text{Correct}) \times \text{Loss}(\text{Classified}, \text{Correct})$$

$$= \sum_{\text{Classified},\text{Correct}} P^{\#}(\text{Classified}, \text{Correct}) \text{Loss}(\text{Classified}, \text{Correct}).$$

That is, you first multiply the confusion matrix and the loss matrix term by term, and then you take the sum of all these elements.

The expected loss for the poker classifier is

$$\sum_{Plc,BH} P^{\#}(Plc, BH)\text{Loss}(Plc, BH) = -3.0.05 - 1.0.1 - 1.0.1 - 1.0.05 = -0.4.$$

A general problem in connection to machine learning is *overfitting*. What we are looking for is a classifier that can classify not-yet-seen cases. However, it may happen that the learned classifier is very accurate on the training data, but it is very poor when confronted with cases not represented there. To monitor overfitting, you usually divide the set data into training and test data, and you measure the classification accuracy on the test data set rather than on the training data set. A way of addressing overfitting in the choice of model is to reserve a part of the training set for validation and comparison of models only and not for establishing the models.

8.3 Extensions of Naive Bayes Classifiers

NBCs assume that the feature variables are independent given the class. Even though this assumption seldom holds, NBCs are surprisingly good with respect to classification accuracy. However, as described in the previous section, classification accuracy does not tell the full story. Often you are particularly interested in detecting a rare class. The class being rare also means that classification accuracy does not drop significantly if your classifier never identifies these cases.

A rare class is often identified through a set of feature values appearing together, where each value by itself does not point in that direction. NBCs cannot cope with that, since they assume the features to be independent given the class. Therefore, you may wish to extend NBCs to allow more elaborate dependency structure among feature variables. A simple extension of this kind is the *tree augmented naive Bayes classifier* (TAN): each feature variable has at most one feature variable as parent.

As opposed to the situation for NBCs, the structure is not given, and we have to look for a structure that with optimal parameter setting has maximal likelihood: out of the possible links between feature nodes we have to choose

a set forming a tree. This is similar to the situation described in Section 7.3.3, and not surprisingly, the problem is solved through a slight modification of the Chow–Liu algorithm using conditional mutual information rather than mutual information (see equation (7.2), Page 237).

We give the construction without proof.

Theorem 8.1 (Learning TANs). *Let $\mathcal{D}$ be a data set over the variables $\{F_1, \ldots, F_m, C\}$. A TAN of maximal likelihood can be constructed as follows:*

1. *Calculate the conditional mutual information $MI(F_i, F_j \mid C)$ for each pair (F_i, F_j).*
2. *Consider the complete MI-weighted graph: the complete undirected graph over $\{F_1, \ldots, F_n\}$, where the links $F_i - F_j$ have the weight $MI(F_i, F_j \mid C)$.*
3. *Build a maximal-weight spanning tree for the complete MI-weighted graph.*
4. *Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.*
5. *Add the node C and a directed link from C to each feature node.*
6. *Learn the parameters.*

Running the TAN algorithm on the data for the poker domain resulted in the TAN in Figure 8.4.

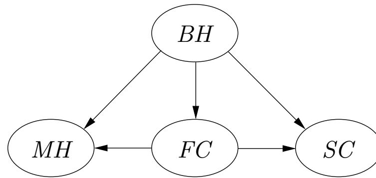

Fig. 8.4. A TAN for classifying poker.

Another extension is to introduce intermediate variables. For the poker example, the dependence between FC and SC can be mediated through a hidden variable C, as illustrated in Figure 8.5.

A problem with hidden variables is that even if you know how to connect the hidden variables introduced, you have to determine the number of states of the hidden variables. Let H be a hidden variable with n states and with children $\mathrm{ch}(H)$. If n is equal to the product of the number of states of the children, then H can represent any configuration of $\mathrm{ch}(H)$, and you cannot hope for a better fit. On the other hand, in that case, you should represent the product of $\mathrm{ch}(H)$ directly without a hidden variable. For the poker example it means that the number of states of C should be between 2 and 11. Now use the EM algorithm for these ten possible numbers of states. Since the likelihood increases with the number of states of C, the model of maximal likelihood has eleven hidden states, and that is not really what you are after. Therefore, you have to balance likelihood with size as described in Section 7.3.1.

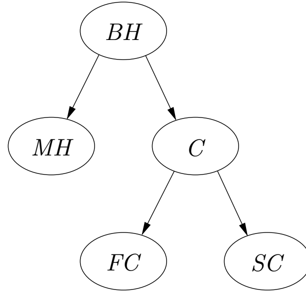

Fig. 8.5. The dependence between *FC* and *SC* is mediated by the hidden variable *C*.

8.4 Classification Trees

For the sake of completeness we shall in this section present a very popular method for doing classification. In the data mining literature the method is called a *decision tree*. However, since in this book we use this term differently (see Section 9.3), we shall call it a *classification tree*.

A classification tree is a directed tree whose internal nodes are feature variables. The links are labeled with values of the feature in question, and the leaves are labeled with class values (see Figure 8.6).

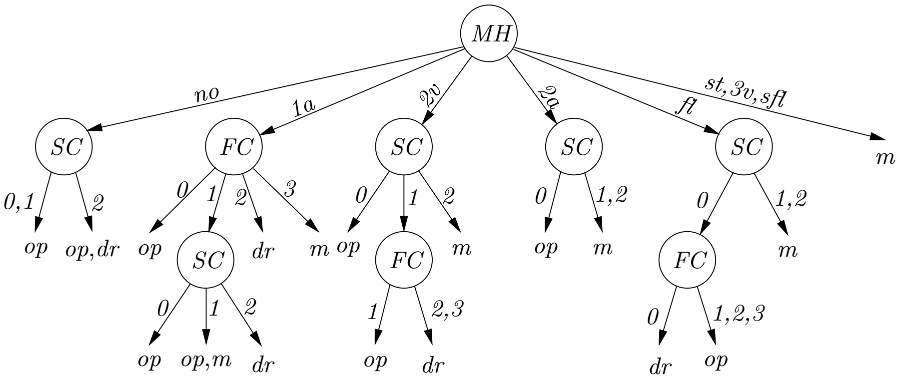

Fig. 8.6. A classification tree for poker

The tree in Figure 8.6 can be used to classify the situation with respect to *BH*. Classification is performed through processing the tree from the root toward the leaves. First you branch out based on the value of *MH*. Depending on the answer, you branch out according to the value of either *FC* or *SC*, and sometimes you also ask for the value of the other card change. When you reach a leaf, you read the classification.

To learn a classification tree, you first determine which feature variable to use as the root. Let C be the class variable with states $\{c_1, \ldots, c_n\}$, and let F be a feature variable with states $\{f_1, \ldots, f_k\}$. We wish to characterize how good a classifier F alone would be. That is, if we know the state of F, how close will we be at knowing the class value?

The values of F partitions $\mathcal{D}$ into the data sets $\mathcal{D}_1^1, \ldots, \mathcal{D}_k^1$, and for each data set $\mathcal{D}_i^1$ we have a distribution $P^\#(C|f_i)$. One way of measuring how close we are to knowing C in the data set $\mathcal{D}_i^1$ is to calculate the entropy for C. In general, for a variable X with distribution $P(X)$ (or $P^\#(X)$), the entropy is defined as

$$\mathrm{Ent}(P(X)) = -\sum_{x \in \mathrm{sp}(X)} P(x)\log_2(P(x)), \tag{8.1}$$

where we let $0\log_2(0) = 0$. If the probability of X being in a particular state approaches 1, then the entropy goes toward 0. On the other hand, the more dispersed the probability mass, the higher the entropy; in case we have a uniform distribution, the entropy attains its maximum value, $\log_2(|\mathrm{sp}(X)|)$.

Now, if the entropy of each distribution $P^\#(C|f_i)$ is small, then knowing F brings us close to knowing C, but if the entropies are large, then knowing F does not give us much information about C. There are various ways of using the entropies as a score for ranking the variables. A method called ID3 uses the expected entropy as a measure of how good a feature is at predicting the class:

$$\mathbb{E}[\mathrm{Ent}(F)] = \sum_F P^\#(F)\,\mathrm{Ent}(P^\#(C \mid F)).$$

Actually, the algorithm uses information gain,

$$\mathrm{Ent}(P^\#(C)) - \mathbb{E}[\mathrm{Ent}(F)],$$

but since $\mathrm{Ent}(P^\#(C))$ is independent of F, you look for a variable giving the lowest expected entropy.

Having chosen the feature F as the root, you continue recursively on the data sets $\mathcal{D}_1^1, \ldots, \mathcal{D}_k^1$.

As an illustration, the ID3 algorithm applied to the data set in Table 8.1 would first partition the data set for each variable. For the variable SC we have the sets $\{8, 9, 14\}$, $\{1, 2, 3, 4, 5, 7, 11, 13, 17, 18\}$, and $\{6, 10, 12, 15, 16, 19, 20\}$ corresponding to the states 0, 1, and 2, respectively. The set for state 0 has two cases with state *op*, and one with state *m*. This distribution has the entropy

$$-\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right) = -\frac{1}{3}\left(2 - 2\log_2 3 - \log_2 3\right) = 0.918,$$

yielding a contribution of $3/20 \cdot 0.918 = 0.138$ to the expected entropy.

The following expected entropies are calculated (note that the maximal entropy for a distribution over three states is $\log_2 3 = 1.585$):

$$\mathbb{E}[\text{Ent}(MH)] = 0.735, \quad \mathbb{E}[\text{Ent}(FC)] = 1.351, \quad \mathbb{E}[\text{Ent}(SC)] = 1.403.$$

Since *MH* has the lowest expected entropy, it is chosen as root. For each value of *MH* you now have a small data set, and you choose the best root for each. For *MH* = *no* you have four cases, and since *SC* separates these cases better than *FC*, *SC* is chosen. The full tree is given in Figure 8.7; the *?* indicates that no case covers the specified configuration, and for these situations you may take the majority class.

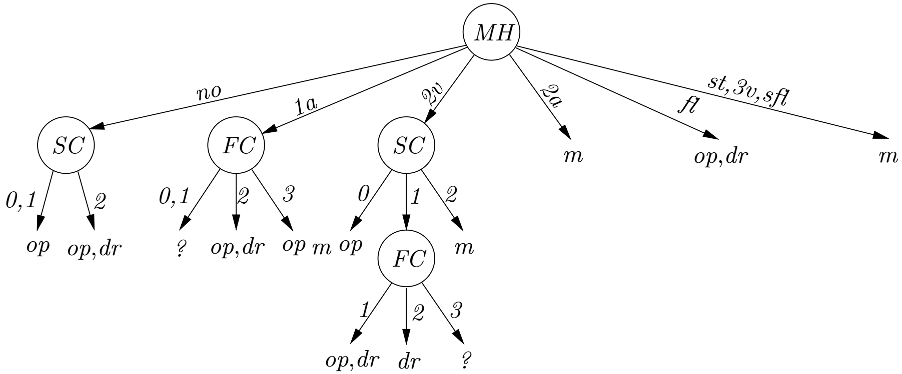

Fig. 8.7. The result of applying the ID3 algorithm on the data set in Table 8.1.

8.5 Summary

The Naive Bayes Classifier

In a naive Bayes classifier, each feature variable has the class variable as its only parent. This means that the structure is fixed, and learning a classifier therefore amounts to estimating the parameters.

Evaluating Classifiers

Two approaches for evaluating a classifier:

Classification accuracy: the fraction of correctly classified cases.

Expected loss:

$$\text{Expected loss} = \sum_{\text{Classified,Correct}} P^{\#}(\text{Classified, Correct})\text{Loss}(\text{Classified, Correct}).$$

The Tree-Augmented Naive Bayes Classifier

In the tree-augmented naive Bayes classifier (TAN classifier), each feature variable has at most one other feature variable as parent in addition to the class variable.

Learning TANs: Let $\mathcal{D}$ be a dataset over the variables $\{F_1, \ldots, F_m, C\}$. A TAN of maximal likelihood can be constructed as follows:

1. Calculate the conditional mutual information $MI(F_i, F_j \mid C)$ for each pair (F_i, F_j).
2. Consider the complete MI-weighted graph: the complete undirected graph over $\{F_1, \ldots, F_n\}$, where the links $F_i - F_j$ have the weight $MI(F_i, F_j \mid C)$.
3. Build a maximal-weight spanning tree for the complete MI-weighted graph.
4. Direct the resulting tree by choosing any variable as a root and setting the directions of the links to be outward from it.
5. Add the node C and a directed link from C to each feature node.
6. Learn the parameters.

Classification Trees

A classification tree is a directed tree whose internal nodes are feature variables. The links are labeled with values of the feature in question, and the leaves are labeled with class values.

To learn a classification tree, you start with the empty tree and iteratively insert the node X that tells you the most about the class variable C. One possible measure is the expected entropy:

$$\mathbb{E}[\mathrm{Ent}(X)] = \sum_X P^{\#}(X)\, \mathrm{Ent}(P^{\#}(C \mid X)),$$

where

$$\mathrm{Ent}(P(X)) = - \sum_{x \in \mathrm{sp}(X)} P(x) \log_2(P(x)).$$

8.6 Bibliographical Notes

As mentioned, naive Bayes was used by de Dombal *et al.* (1972) and can be traced back at least to Minsky (1963). It was introduced to classification by Duda and Hart (1973). Its role in classification has been thoroughly studied in the last decade or so, with Domingos and Pazzani (1997) providing theoretical results on concepts that naive Bayes can classify better than any other classifier, and with empirical results that show how violations of the independence assumptions of the model are often of no consequence. Jaeger (2003) further

clarifies the distinction between the concepts they can recognize, and the theoretical limits on the concepts that can be learned from data. Tree-augmented naive Bayes classifiers were introduced by Friedman *et al.* (1997). The ID3 algorithm for inferring classification trees was introduced by Quinlan (1979) and later improved in (Quinlan, 1986). For a general overview over classifiers, see (Mitchell, 1997).

8.7 Exercises

Exercise 8.1. Verify that the PC-algorithm results in the network in Figure 8.2 (or one of its equivalents) when run with an oracle based on the d-separation properties of the network in Figure 8.1, and with the variables *OH1* and *OH2* hidden.

Exercise 8.2. Learn the maximum likelihood parameters for the classifier in Figure 8.3 from the cases in Table 8.1. What class does your classifier assign to a case with *MH*=1a, *FC*=1, and *SC*=1?

Exercise 8.3. Verify that the TAN-algorithm constructs the classifier in Figure 8.4 and complete the classifier by learning the maximum likelihood parameters. What class does the classifier assign to the case with *MH*=1a, *FC*=1, and *SC*=1? What would the result be if you instead of maximum likelihood estimates used Bayesian parameter estimates?

Exercise 8.4. Consider the classification tree in Figure 8.6. How would this classifier classify the case with *MH*=1a, *FC*=1, and *SC*=1?

Exercise 8.5. Using the data in Table 7.4, construct a classification tree for classifying A. What class is assigned to $(B = 1, C = 2)$?