Classification using Hierarchical Naive Bayes Models

HNB workshop

Motivation

Previous work on learning a HNBs focused on scientific modeling, i.e.:

- Find an interesting latent structure (based on the BIC score).

We focus on learning a HNB for classification, i.e., taking the technological modeling approach:

- Build an accurate classifier.
- Provide a semantic interpretation to the latent variables.
  - A latent variable aggregates the information from its children which is relevant for classification.

Bayesian classifiers

In a probabilistic framework, classification is the calculation of $P(C|\mathcal{A})$. A new instance is classified as c^*, where:

$$c^* = \arg \min_{c \in \text{sp}(C)} \sum_{c' \in \text{sp}(C)} L(c, c') P(C = c' | \bar{a}),$$

and $L(c, c')$ is the loss function.

The two loss functions which are commonly used:

- The $0/1$-loss: $L(c, c') = 1$ if $c \neq c'$ and 0 otherwise.
- The log-loss: $L(c, c') = \log P(c'|\bar{a})$ independently of c.

Both loss functions have the property that the Bayes classifier should classify an instance $\bar{a}$ to c^* s.t.:

$$c^* = \arg \max_{c \in \text{sp}(C)} P(C = c | \bar{a})$$

Learning a classifier therefore reduces to estimating $P(C|\mathcal{A})$ from training examples.

The score

One approach is to learn a classifier is to use a standard BN learning algorithm, e.g. MDL:

$$\text{MDL}(D|\mathcal{D}_N) = \frac{\log N}{2}\left|\hat{\Theta}_{B_S}\right| - \sum_{i=1}^{N} \log\left(P_B\left(c^{(i)}, \bar{a}^{(i)}|\hat{\Theta}_{B_S}\right)\right).$$

However, as:

$$\sum_{i=1}^{N} \log\left(P_B\left(c^{(i)}, \bar{a}^{(i)}|\hat{\Theta}_{B_S}\right)\right) = \sum_{i=1}^{N} \log\left(P_B\left(c^{(i)}|\bar{a}^{(i)}, \hat{\Theta}_{B_S}\right)\right) + \sum_{i=1}^{N} \log\left(P_B\left(\bar{a}^{(i)}|\hat{\Theta}_{B_S}\right)\right)$$

the last term will dominate as $|\mathcal{A}|$ grows large.

Instead we could use predictive MDL:

$$\text{MDL}_p(D|\mathcal{D}_N) = \frac{\log N}{2}\left|\hat{\Theta}_{B_S}\right| - \sum_{i=1}^{N} \log\left(P_B\left(c^{(i)}|\bar{a}^{(i)}, \hat{\Theta}_{B_S}\right)\right).$$

but, in general, this score cannot be calculated efficiently.

Predictive MDL and the wrapper approach

The argument for using predictive MDL is that it is guaranteed to find the best classifier as $N \rightarrow \infty$.

However, as J. H. Friedman (1997) noted:

> Good probability estimates are not necessary for good classification; similarly, low classification error does not imply that the corresponding class probabilities are being estimated (even remotely) accurately.

As predictive MDL may not be successful for finite datasets, we use the wrapper approach instead:

- Calculate an approximate accuracy of a given classifier by cross-validation, and use this as the scoring function (unfortunately, it has a higher computational complexity).

The basic algorithm I

The algorithm performs a greedy search over the space of HNBs:

- Initiate model search with H_0 (the NB model).
- For $k = 0, 1, \ldots$
  a. Select $H' \in \arg\max_{H \in \mathcal{B}(H_k)}$ Score$(H_k|\mathcal{D}_N)$.
  b. If Score$(H'|\mathcal{D}_N) >$ Score$(H_k|\mathcal{D}_N)$, then
     $H_{k+1} \leftarrow H'$ and $k \leftarrow k + 1$
     else
     Return H_k.

The search boundary $\mathcal{B}(H_k)$ defines the models that are reachable from H_k:

- Each model in $\mathcal{B}(H_k)$ has exactly one more hidden variable, say L, than H_k, and
- L is a child of C and L has exactly two children.

When moving from H_k we choose the model in $\mathcal{B}(H_k)$ with the highest score.

The basic algorithm II

Note that:

- The final HNB model has a binary tree structure.
- There is a model in $\mathcal{B}(H_k)$ for each possible way to define the cardinalities of each possible new latent variable!

Pinpoint a few promising models without examining all models in $\mathcal{B}(H_k)$:

1. Find a candidate hidden variable.
2. Find the cardinality of the new hidden variable.

Find a candidate hidden variable

Recall that hidden variables are introduced to relax the independence assumptions of the NB structure.

For all pairs $\{X, Y\} \subseteq \text{ch}(C)$ we could therefore calculate $I(X, Y|C)$:

$$I(X, Y|C) = \sum_{c,x,y} P(x, y, c) \log \left(\frac{P(x, y|c)}{P(x|c)P(y|c)} \right)$$

and choose the pair with highest conditional mutual information given C.

However, $I(X, Y|C)$ is increasing in both $|\text{sp}(X)|$ and $|\text{sp}(Y)|$ so this strategy would favor pairs of variables with larger state spaces.

Instead we utilize:

$$2N \cdot I(X, Y|C) \xrightarrow{\mathcal{L}} \chi^2_{|\text{sp}(C)|(|\text{sp}(Y)|-1)(|\text{sp}(X)|-1)}$$

and pick the pair with highest probability $P(Z \leq 2N \cdot I(X, Y|C))$.

Find the cardinality

We use an algorithm similar to the one by Elidan and Friedman (2001):

1. Initially $|\text{sp}(L)| = \prod_{X \in \text{ch}(L)} |\text{sp}(X)|$, and each state corresponds to exactly one combination of the states of the children.
2. Iteratively collapse two states as long as it is "beneficial".

Here it is important to note that:

- We can now easily infer the data for the hidden variables.
- We can perform a "deterministic propagation" in the hidden part of the model $\Rightarrow$ we end up with an NB model!

But how do we find the states that should be collapsed?

Which states to collapse?

Unfortunately, it is computationally hard to measure the benefit of collapsing two states by using the wrapper approach.

Instead we approximate the benefit using predictive MDL$_p$:

- Two states l_i and l_j should be collapsed into l' if MDL$_p(H') <$ MDL$_p(H)$.

This allows us to exploit that the states are locally decomposable.

Locally decomposable I

Two states l_i and l_j should be collapsed if:

$$\Delta_L(l_i, l_j) = \mathsf{MDL}_p(H, \mathcal{D}_N) - \mathsf{MDL}_p(H', \mathcal{D}_N) > 0$$

Thus,

$$\Delta_L(l_i, l_j) = \frac{\log(N)}{2}(|\Theta_{B'_S}| - |\Theta_{B_S}|) + \sum_{i=1}^{N}(\log(P_B(c^{(i)}|a^{(i)})) - \log(P_{B'}(c^{(i)}|a^{(i)}))).$$

Since all the hidden variables are "observed" we have:

$$|\Theta_{B_S}| = (|\mathsf{sp}(C)| - 1) + |\mathsf{sp}(C)| \sum_{X \in \mathsf{ch}(C)} (|\mathsf{sp}(X)| - 1),$$

and the first term therefore reduces to:

$$\frac{\log(N)}{2}(|\Theta_{B'_S}| - |\Theta_{B_S}|) = \frac{\log(N)}{2}|\mathsf{sp}(C)|$$

Locally decomposable II

$$\Delta_L(l_i, l_j) = \frac{\log(N)}{2}|\mathrm{sp}(C)| + \sum_{i=1}^{N}(\log(P_B(c^{(i)}|a^{(i)})) - \log(P_{B'}(c^{(i)}|a^{(i)}))).$$

For the second term we note that:

$$\sum_{i=1}^{N}(\log(P_B(c^{(i)}|a^{(i)})) - \log(P_{B'}(c^{(i)}|a^{(i)}))) = \sum_{i=1}^{N}\log\frac{P_B(c^{(i)}|a^{(i)})}{P_{B'}(c^{(i)}|a^{(i)})}$$

$$= \sum_{D\in\mathcal{D}:f(D,l_i,l_j)}\log\frac{P_B(c^D|a^D)}{P_{B'}(c^D|a^D)},$$

where $f(D, l_i, l_j)$ is true if case D includes either state l_i or l_j

Locally decomposable III

To avoid having to consider all possible combinations of attributes we approximate the second term:

$$\sum_{D \in \mathcal{D}: f(D,s',s'')} \log \frac{P_B(c^D|a^D)}{P_{B'}(c^D|a^D)} \approx$$

$$\log \prod_{c \in \mathrm{sp}(C)} \left[\left(\frac{N(c,l_i)}{N(l_i)} \right)^{N(c,l_i)} \cdot \left(\frac{N(c,l_j)}{N(l_j)} \right)^{N(c,l_j)} / \left(\frac{N(c,l_i)+N(c,l_j)}{N(l_i)+N(l_j)} \right)^{N(c,l_i)+N(c,l_j)} \right]$$

where $N(c,s)$ and $N(s)$ are the sufficient statistics, e.g.:

$$N(c,s) = \sum_{i=1}^{|\mathcal{D}|} \gamma(C=c, L=s : D_i),$$

where $\gamma(C=c, L=s : D_i)$ takes on the value 1 if $(C=c, L=s)$ appears in case D_i, and 0 otherwise.

Locally decomposable IV

When combining it all we get:

$$\Delta_L(l_i, l_j) \approx \frac{\log(N)}{2} |\mathsf{sp}(C)| - \sum_{c \in \mathsf{sp}(C)} N(c, l_i) \log \left(\frac{N(c, l_i)}{N(c, l_i) + N(c, l_j)} \right) - \sum_{c \in \mathsf{sp}(C)} N(c, l_j) \log \left(\frac{N(c, l_j)}{N(c, l_i) + N(c, l_j)} \right) + N(l_i) \log \left(\frac{N(l_i)}{N(l_i) + N(l_j)} \right) + N(l_j) \log \left(\frac{N(l_j)}{N(l_i) + N(l_j)} \right)$$

Complexity

- Initiate model search with H_0 (the NB model).
- For $k = 0, 1, \ldots$
  a. Select $H' \in \arg\max_{H \in \mathcal{B}(H_k)}$ Score$(H_k|\mathcal{D}_N)$.
  b. If Score$(H'|\mathcal{D}_N) >$ Score$(H_k|\mathcal{D}_N)$, then
     $H_{k+1} \leftarrow H'$ and $k \leftarrow k + 1$
     else
     Return H_k.

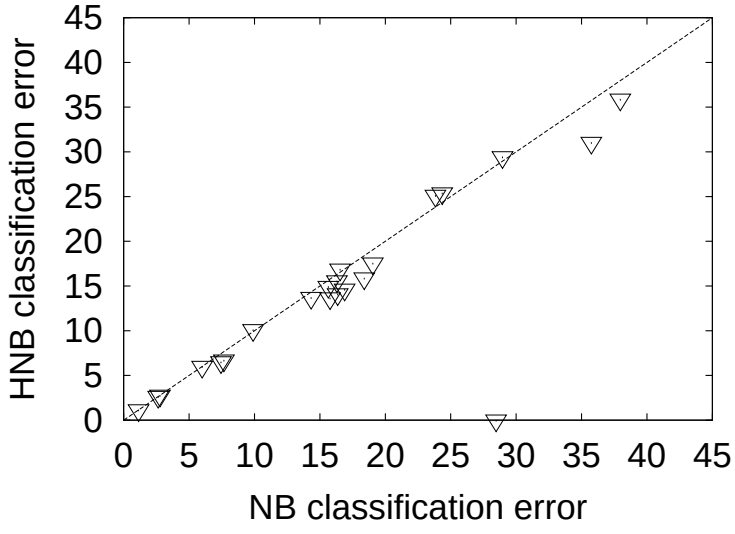
The algorithm can now be shown to have complexity:

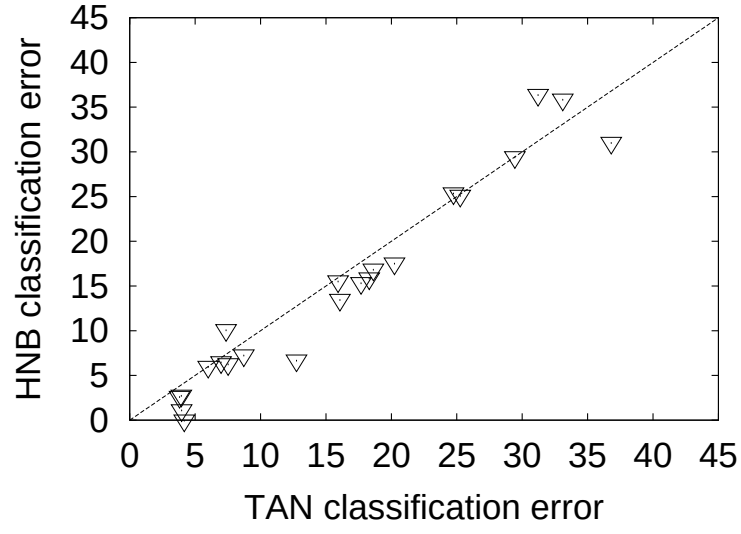
$$\mathcal{O}(n^2 \cdot N).$$

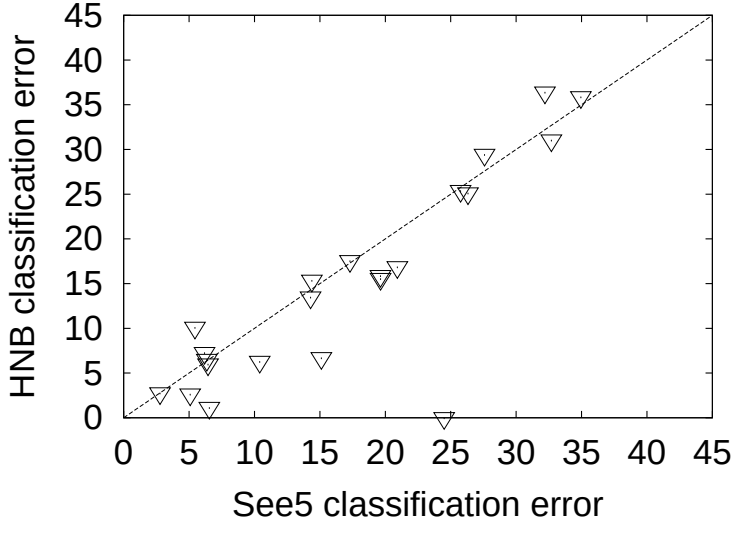
Data sets

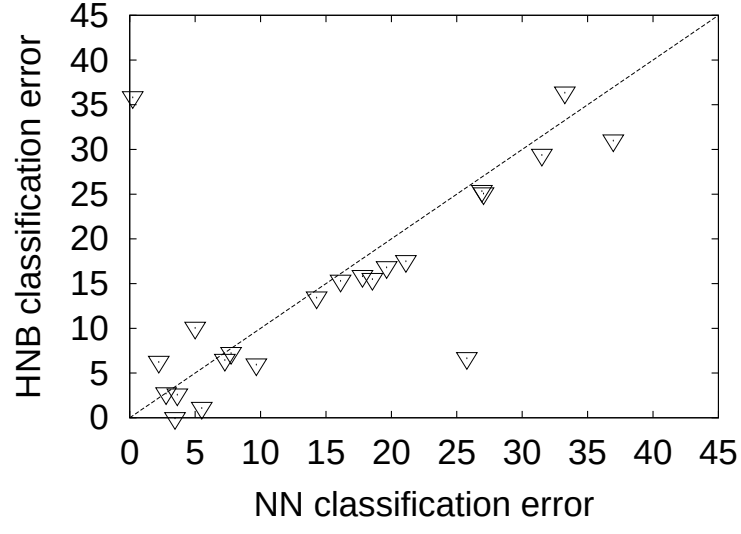
Database	# Attributes	# Classes	#Instances	
			Train	Test
postop	8	3	90	XVal(5)
iris	4	3	150	XVal(5)
monks-1	6	2	124	432
monks-2	6	2	124	432
monks-3	6	2	124	432
glass	9	7	214	XVal(5)
glass2	9	2	163	XVal(5)
diabetes	8	2	768	XVal(5)
heart	13	2	270	XVal(5)
hepatitis	19	2	155	XVal(5)
pima	8	2	768	XVal(5)
cleve	13	2	296	XVal(5)
wine	13	3	178	XVal(5)
thyroid	5	3	215	XVal(5)
ecoli	7	8	336	XVal(5)
breast	10	2	683	XVal(5)
vote	16	2	435	XVal(5)
crx	15	2	653	XVal(5)
australian	14	2	690	XVal(5)
chess	36	2	2130	1066
vehicle	18	4	846	XVal(5)
soybean-large	35	19	562	XVal(5)

Results


HNB classification error
NB classification error
0
5
10
15
20
25
30
35
40
45


HNB classification error
TAN classification error
0
5
10
15
20
25
30
35
40
45


HNB classification error
See5 classification error
0
5
10
15
20
25
30
35
40
45


HNB classification error
NN classification error
0
5
10
15
20
25
30
35
40
45