NBCT: A Toolkit for Naive Bayes Clustering

(version 0.1)

Yoshitaka Kameya
Tokyo Institute of Technology

August 16, 2007

Contents

1 Introduction

NBCT (Naive Bayes Clustering Toolkit) is a C implementation of several probabilistic inference algorithms related to *naive Bayes clustering* — probabilistic clustering based on a *naive Bayes model* [1]. Currently NBCT includes a discrete version of *AutoClass* [4, 5] augmented with the *split-merge EM* (SMEM) algorithm [18].[1] Besides, NBCT provides the following functions for each of three statistical frameworks — ML (*maximum likelihood*), MAP (*maximum a posteriori*) and VB (*Variational Bayes*):

- EM learning
  (with random restarts or SMEM algorithm)
- Clustering
- Other probabilistic inferences
  - Computing membership distributions
  - Computing dissimilarities between objects (cases), or between attribute values
  - Relevance analysis
- Model scoring

[1] In version 0.1, the SMEM algorithm is considered to be experimental. Furhtermore, in a future version, it is planned to incorporate Ueda and Ghahramani's SMEM algorithm that aims to simultaneously solve the problems of avoiding undesirable local maxima and finding the optimal number of clusters [17].

It is empirically shown in recent researches that the VB approach often works better than ML/MAP approaches in *model selection* (e.g. determining the optimal number of clusters) [2, 11], though ML/MAP is simpler and would be easier to work with. NBCT can be applied to many of discrete domains such as document clustering with bag-of-words representation [1] or distributional clustering of words [13].

The rest of this document is comprised of three parts. First, in Section 2, we will see a typical usage of NBCT with an exemplar artificial dataset included in the released package. Section 3 then gives a (rough) description of the algorithms used in NBCT. Section 4 is the third part which describes the detailed usage of NBCT.

2 Getting started

Let us try NBCT with an artificial dataset included in the released package. It is assumed here that the installation of NBCT (see §4.2 for the procedure) has been successfully done, and we are working in the `example` directory of the unfolded package.

In this section, we just focus on the look and feel of NBCT, and do not aim at listing all functions. We first describe the problem domain and probabilistic models applied to the domain (i.e. naive Bayes models). The users who are familiar with naive Bayes clustering or classification may skip the following two sections (§2.1 and §2.2), and jump to §2.3, which demonstrates NBCT.

2.1 Problem domain

Given a dataset D of *objects*, clustering is a task to group similar objects in D, or typically, to partition D into disjoint sets, called *classes* or *clusters*, of similar objects. So each of these objects uniquely belongs to one of K predefined classes. Also we consider that each object is represented by an vector of values of J predefined *attributes*. Please note that in the released program we will use the term 'case' instead of 'object' (just for a historical reason).

To be concrete, let us open `test1_freq.csv` in the directory by some text viewer:

```
V1,W1,X1,Y1,Z1
V1,W1,X1,Y1,Z1
V1,W1,X1,Y1,Z1
      :
V1,W1,X1,Y1,Z2
V1,W1,X1,Y1,Z2
      :
```

Each line in the file corresponds to an object, and each of comma-separated values in the line corresponds to a value of an attribute of the object. In the above, we can see that the first object is represented by the vector $\boldsymbol{x} =$ (`V1`, `W1`, `X1`, `Y1`, `Z1`) of attribute values.

As above, there are cases where several objects have the same attribute values. In NBCT, these objects will be regarded as the same object since there is no way to distinguish them. Then the data D will be a multiset of objects. `test2_freq.txt` contains the same dataset as that of `test1_freq.csv`, but the objects who have the same vector $\boldsymbol{x}$ of attribute values are suppressed, and instead their count $N(\boldsymbol{x})$ is added to the right-most column:

```
V1    W1    X1    Y1    Z1    201
V1    W1    X1    Y1    Z2    237
V1    W1    X1    Y1    Z3    30
V1    W1    X1    Y1    Z4    116
            :
```

For example, we have $N(\boldsymbol{x}) = 237$ for $\boldsymbol{x} =$ (`V1`, `W1`, `X1`, `Y1`, `Z2`).

To make notations simple, we give indices 1, 2, ... to attributes from left to right, and refer to each attribute by its index. Also, independently for each attribute, we give indices 1, 2, ... to the attribute values according to the order of appearances. Then each attribute value will be referred to by its index. For example, we will refer `Y1`, `Y2`, `Y3` and `Y4` by their indices 1, 2, 3 and 4, respectively.

2.2 Naive Bayes models

To build clusters of objects in the dataset D, we attempt to use a probabilistic model called a naive Bayes model. In naive Bayes models, it is considered that the objects in D were generated in a causal way depicted as a Bayesian network (Fig. 1), which has random variables C and X_j ($1 \leq j \leq J$). C is called a *class variable*, and X_j ($1 \leq j \leq J$) are called *attribute variables*. In such a naive Bayes model, the class of each object is firstly determined as k under the class distribution $p(C = k)$, and then each attribute value x_j ($1 \leq j \leq J$) is conditionally determined under the attribute distribution $p(X_j = x_j \mid C = k)$.

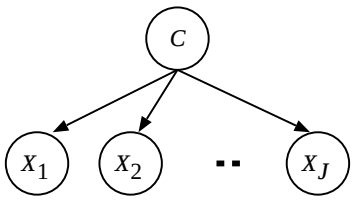

Figure 1: Bayesian network representation of a naive Bayes model.

We let V_j denote the number of possible values of the j-th attribute. The probability that an object with attribute values $\boldsymbol{x} = (x_1, x_2, \ldots, x_J)$ belongs to the class k is simply computed as:

$$\begin{aligned} & p(C=k, X_1=x_1, X_2=x_2, \ldots, X_J=x_J) \\ = \;& p(C=k) \prod_{j=1}^{J} p(X_j=x_j \mid C=k), \end{aligned} \tag{1}$$

where $1 \le k \le K$ and $1 \le x_j \le V_j$ $(1 \le j \le J)$.

Furthermore, we hereafter simplify the above as:

$$p(k, \boldsymbol{x}) = p(k, x_1, x_2, \ldots, x_J) = p(k) \prod_{j=1}^{J} p(x_j \mid k), \tag{2}$$

where $p(C=k, \ldots)$ and $p(X_j=x_j, \ldots)$ are abbreviated as $p(k, \ldots)$ and $p(x_j, \ldots)$, respectively. Since the probabilities $p(k)$ and $p(x_j \mid k)$ in the right hand side can be seen as *parameters* of $p(k, \boldsymbol{x})$, we write them explicitly as θ_k and θ_{j,k,x_j}, respectively. Then, $p(k, \boldsymbol{x})$ takes a parameterized form $(1 \le k \le K, 1 \le j \le J, 1 \le x_j \le V_j)$:

$$p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) = \theta_k \prod_{j=1}^{J} \theta_{j,k,x_j}\,, \tag{3}$$

where $\boldsymbol{\theta}$ is a vector consisting of θ_k and θ_{j,k,x_j}. $p(k, \boldsymbol{x} \mid \boldsymbol{\theta})$ is called the *joint distribution* specified by the naive Bayes model, and is used for various probabilistic inferences including clustering, which are described later.

Under some parameter settings, new objects can be sampled from the naive Bayes model. Indeed, the dataset in `test1_freq.csv` or `test2_freq.txt` has been artificially generated by sampling under the parameters shown in Tab. 1, where $K = 5$, $J = 5$, $V_1 = 2$, $V_2 = 2$, $V_3 = 3$, $V_4 = 4$ and $V_5 = 5$.

2.3 Running NBCT

At this point, we can try NBCT with the dataset contained in `test1_freq.csv` or `test2_freq.txt`.

Table 1: Parameters used in generating `test1_freq.csv` or `test2_freq.txt`

θ_1	0.1
θ_2	0.5
θ_3	0.2
θ_4	0.05
θ_5	0.15
$\theta_{1,1,1}$	0.9
$\theta_{1,1,2}$	0.1
$\theta_{1,2,1}$	0.7
$\theta_{1,2,2}$	0.3
$\theta_{1,3,1}$	0.1
$\theta_{1,3,2}$	0.9
$\theta_{1,4,1}$	0.2
$\theta_{1,4,2}$	0.8
$\theta_{1,5,1}$	0.5
$\theta_{1,5,2}$	0.5
$\theta_{2,1,1}$	0.8
$\theta_{2,1,2}$	0.2
$\theta_{2,2,1}$	0.99
$\theta_{2,2,2}$	0.01
$\theta_{2,3,1}$	0.7
$\theta_{2,3,2}$	0.3
$\theta_{2,4,1}$	0.05
$\theta_{2,4,2}$	0.95
$\theta_{2,5,1}$	0.4
$\theta_{2,5,2}$	0.6

$\theta_{3,1,1}$	0.3
$\theta_{3,1,2}$	0.3
$\theta_{3,1,3}$	0.4
$\theta_{3,2,1}$	0.8
$\theta_{3,2,2}$	0.1
$\theta_{3,2,3}$	0.1
$\theta_{3,3,1}$	0.05
$\theta_{3,3,2}$	0.9
$\theta_{3,3,3}$	0.05
$\theta_{3,4,1}$	0.3
$\theta_{3,4,2}$	0.6
$\theta_{3,4,3}$	0.1
$\theta_{3,5,1}$	0.2
$\theta_{3,5,2}$	0.1
$\theta_{3,5,3}$	0.7

$\theta_{4,1,1}$	0.05
$\theta_{4,1,2}$	0.05
$\theta_{4,1,3}$	0.3
$\theta_{4,1,4}$	0.6
$\theta_{4,2,1}$	0.2
$\theta_{4,2,2}$	0.05
$\theta_{4,2,3}$	0.5
$\theta_{4,2,4}$	0.25
$\theta_{4,3,1}$	0.1
$\theta_{4,3,2}$	0.8
$\theta_{4,3,3}$	0.05
$\theta_{4,3,4}$	0.05
$\theta_{4,4,1}$	0.8
$\theta_{4,4,2}$	0.05
$\theta_{4,4,3}$	0.05
$\theta_{4,4,4}$	0.1
$\theta_{4,5,1}$	0.7
$\theta_{4,5,2}$	0.1
$\theta_{4,5,3}$	0.1
$\theta_{4,5,4}$	0.1

$\theta_{5,1,1}$	0.15
$\theta_{5,1,2}$	0.1
$\theta_{5,1,3}$	0.2
$\theta_{5,1,4}$	0.5
$\theta_{5,1,5}$	0.05
$\theta_{5,2,1}$	0.3
$\theta_{5,2,2}$	0.4
$\theta_{5,2,3}$	0.05
$\theta_{5,2,4}$	0.2
$\theta_{5,2,5}$	0.05
$\theta_{5,3,1}$	0.4
$\theta_{5,3,2}$	0.1
$\theta_{5,3,3}$	0.1
$\theta_{5,3,4}$	0.1
$\theta_{5,3,5}$	0.3
$\theta_{5,4,1}$	0.95
$\theta_{5,4,2}$	0.01
$\theta_{5,4,3}$	0.01
$\theta_{5,4,4}$	0.02
$\theta_{5,4,5}$	0.01
$\theta_{5,5,1}$	0.8
$\theta_{5,5,2}$	0.05
$\theta_{5,5,3}$	0.05
$\theta_{5,5,4}$	0.05
$\theta_{5,5,5}$	0.05

Let us assume here that the true number of classes is known (i.e. $K = 5$ is known). Then, to get the (five) clusters of objects in `test2_freq.csv`, we invoke an executable named `nbc` with specifying a couple of option flags ('`%`' is the prompt symbol):

```
% nbc -f test2 -k 5 -x 5 -t mp
```

The option '`-f test2`' specifies the names of input/output files including `test2_freq.txt`. Besides, the options '`-k 5`' and '`-x 5`' indicate that the number of clusters is 5 (i.e. $K = 5$), and that the number of attributes is 5 (i.e. $J = 5$), respectively. The option '`-t mp`' enables clustering based on the membership probabilities (see §3.1.3 and §3.1.5 for details). When invoking the program, we may see the messages as follows:

```
#classes = 5:
  #iterations 0.........50.........100...
.....150.........200.........250........
.300.........350.........400.........450.
........500. (Converged: 508 iterations)
       L=-46204.232086
```

In advance of clustering, the program runs the EM (expectation-maximization) algorithm (§3.1.1) to estimate the parameters of the naive Bayes model. The EM algorithm is an iterative, hill-climbing algorithm for parameter estimation. The messages above report the progress of the EM algorithm. To use `test1_freq.csv` instead of `test2_freq.txt`, the `-i` flag will be required to deal with the CSV (comma-separated values) format.

```
% nbc -f test1 -k 5 -x 5 -r 12345 -i
```

After the run, three files named `test2_param.txt`, `test2_cluster.txt` and `test2_log.txt` will have been created. The first file, `test2_param.txt`, contains the parameters estimated by the EM algorithm:

```
#### ID:PARAM-CLASS
#### Class k, Class parameter P(k)
####
0       0.164739744393
1       0.048580087144
2       0.511626472252
3       0.196216832015
4       0.078836864196
#### ID:PARAM-ATT0
#### Class k, Attr. value x0, Attr. parameter
P(x0|k)
####
0       V1      0.522947982507
0       V2      0.477052017493
1       V1      0.145922471394
            :
4       V1      0.949635445666
4       V2      0.050364554334
#### ID:PARAM-ATT1
#### Class k, Attr. value x1, Attr. parameter
P(x1|k)
####
0       W1      0.389463446381
0       W2      0.610536553619
1       W1      0.068199167665
            :
```

We can see from the above that $\theta_1, \theta_2, \ldots$ were estimated as 0.165, 0.049, ..., respectively. In general, each output file contains several data matrices, each of which has header lines beginning with '#'.

The second file, `test2_cluster.txt`, contains the data matrices on the correspondences between clusters and objects:

```
#### ID:CLUSTER-CASE-BY-CASE
#### Case ID, Cluster k, Attr. x1, ...   Attr.
x5, Membership prob. P(k|x)
####
0   2   V1  W1  X1  Y1  Z1  0.805043454811
1   2   V1  W1  X1  Y1  Z2  0.981302362520
2   2   V1  W1  X1  Y1  Z3  0.863899435937
3   2   V1  W1  X1  Y1  Z4  0.912873199926
                :
```

It is found that the object 0, $\boldsymbol{x} = (\mathtt{V1}, \mathtt{W1}, \mathtt{X1}, \mathtt{Y1}, \mathtt{Z1})$, belongs to the cluster 2, and so on. One may also notice that the rows in the data matrix whose identifier is 'CLUSTER-CASE-BY-CASE' is ordered by the indices of objects. On the other hand, the rows in the data matrix named 'CLUSTER-CASE-BY-CLUSTER' is ordered by the indices of clusters:

```
#### ID:CLUSTER-CASE-BY-CLUSTER
#### Cluster k, Case ID, Attr. x1, ...   Attr.
x5, Membership prob. P(k|x)
####
0   22  V1  W1  X2  Y1  Z3  0.359072973437
0   33  V1  W1  X2  Y3  Z4  0.379421817602
0   37  V1  W1  X2  Y4  Z3  0.827169179316
                :
```

The above tell us that the cluster 0 includes the objects 22, 23, 37, and so on. Finally, the third file, `test2_log.txt`, contains the additional information on the last execution.

NBCT provides many options for the EM algorithm. For example, there are cases where we would like to restart with several different initial settings of the EM algorithm. This is because the EM algorithm is a hill-climbing algorithm, and is known to be often trapped in undesirable local maxima. In NBCT, this method is enabled by giving the number of restarts to the `-n` flag:

```
% nbc -f test2 -k 5 -x 5 -t mp -n 10
#classes = 5:
  [0] #iterations 0.........50.........100....
.....150.........200.........250.. .......300.
........350......... (Converged: 399 iteration
s)     L=-46204.232085
  [1] #iterations 0.........50.........100....
.....150.........200.........250.. .......300.
........350.........400.........450 (Converged
: 452
iterations)     L =-46204.232084
  [2] #iterations 0.........50.........100....
.....150.........200.........250.. .......300.
........350.........400.... (Converged: 422 it
erations)      L=-46204 .232089
        :
  [9] #iterations 0.........50.........100....
```

```
% vnbc -f test2 -k 2:10:1 -x 5 -t mp -c 1.0 -w 1.0
|Classes| = 2:
  #iterations 0.........50. (Converged: 59 iterations)  F=-47697.146904
  [2] Free Energy = -47697.146904 (temporarily optimal)
     :
|Classes| = 5:
  #iterations 0.........50.........100.........150.........200.........250.........300....
.....350.........400.... (Converged: 421 iterations)    F=-46458.661401
  [5] Free Energy = -46458.661401 (temporarily optimal)
     :
|Classes| = 10:
  #iterations 0.........50.........100.........150.........200.........250.........300....
.....350.........400.........450.........500.........550.........600.........650.........7
00.........750.........800.........850.........900 (Converged: 900 iterations)  F=-46519.8
96105
  [10] Free Energy = -46519.896105
Optimal |Classes| = 5
```

Figure 2: Messages from `vnbc`.

```
.....150.........200.........250.........300..
.......350.........400.........450.........500
.........550.........600.........650.........7
00.........750.........800.........850........
.900.........950.........1000.........1050....
.....1100.........1150.........1200......... 1
250...... (Converged: 1284 iterations) L=-4625
9.088233
  <<Resumed best parameter set #1>>
  [1] #iterations  (Converged: 453 iterations)
L=-46204.232074
```

We can find from the above the second initial setting (indexed by 1) provides the best estimate of parameters.

To be precise, the parameter estimation method we have run is called 'maximum likelihood (ML) estimation.' On the other hand, maximum a posteriori (MAP) estimation is said to be more robust against the problem of data-sparseness, which often arises with a small data. In MAP estimation, we should tell the 'pseudo counts' to the program. If the pseudo counts are uniformly set to 1 for all parameters, the estimation is commonly called Laplace's estimation. In NBCT, it is possible to specify arbitrary pseudo counts for the class parameters θ_k by the `-c` flag, and for the attribute parameters θ_{j,k,x_j} by the `-w` flag. For example, to perform Laplace's estimation, we may run:

```
% nbc -f test2 -k 5 -x 5 -t mp -c 1.0 -w 1.0
#classes = 5:
  #iterations 0.........50.........100........
.150.........200.........250.........300......
...350.........400.........450.........500....
.....550.........600...... (Converged: 630 ite
rations)  L=-46218.658868
```

We have assumed so far that the true number K of clusters is known in advance, but in more realistic situations, it is of course unknown. So finding the optimal number of clusters (based on the dataset) is a key issue in clustering. This can be seen a kind of model selection problem, which has been intensively discussed in the literature of machine learning. NBCT provides a facility that computes the scores on K based on the marginal likelihood $P(D \mid K)$, the plausibility of D given the number K of clusters.

Although the scores on K are provided for the frameworks of ML and MAP estimation, we here use variational free energy instead as a score on K in the framework of variational Bayes (VB). In VB, we use another executable named `vnbc`. The execution log is shown in Fig. 2, where the number of clusters to be examined ranges from 2 to 10. If we specify '`-k` *Kmin*`:`*Kmax*`:`*Kstep*', NBCT will be switched into the 'model selection' mode. One may find that, in this example, we fortunately recovered the true number of clusters (i.e. $\hat{K} = K = 5$).

At the end of this section, we will mention on an auxiliary tool `nbcsep` for post-processing the output files generated by `nbc` and `vnbc`. As is described before, each output file contains several data matrices, each of which has a unique identifier. `nbcsep` extracts these data matrices, and puts each of them into an individual file. The name of such a new file includes the ID of

the corresponding data matrix. For example, let us apply `nbcsep` to `test2_param.txt`:

```
% nbcsep -f foo test2_param.txt
Output: foo_PARAM-CLASS.txt
Output: foo_PARAM-ATT0.txt
Output: foo_PARAM-ATT1.txt
Output: foo_PARAM-ATT2.txt
Output: foo_PARAM-ATT3.txt
Output: foo_PARAM-ATT4.txt
```

Please note here that the common prefix of the names of newly created files was given by the `-f` flag.

3 Clustering algorithms

This section gives the detailed descriptions on the clustering algorithms and the other related algorithms provided in NBCT.

3.1 ML/MAP based clustering

3.1.1 Parameter estimation based on ML

In advance of clustering and the other probabilistic inferences based on the joint distribution $p(k, \boldsymbol{x} \mid \boldsymbol{\theta})$, we need to estimate the parameters $\boldsymbol{\theta}$ from the dataset D. Let us consider again that we are given a set D of objects where $N(\boldsymbol{x})$ is the number of occurrences of objects that have the attribute values $\boldsymbol{x}$. We define N as the number of total occurrences of objects, that is, $N = \sum_{\boldsymbol{x}} N(\boldsymbol{x})$. Since in clustering, we do not know the class to which each object belongs, the dataset D contains no information about k. In this sense, D is often called *incomplete data*. In *maximum likelihood (ML) estimation*, we try to find the parameters $\boldsymbol{\theta}$ that maximize the *likelihood* $p(D \mid \boldsymbol{\theta})$. That is, we have:

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\mathrm{ML}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(D \mid \boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log p(D \mid \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}} N(\boldsymbol{x}) \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}} N(\boldsymbol{x}) \log \sum_{k=1}^{K} p(k, \boldsymbol{x} \mid \boldsymbol{\theta}). \qquad (4)\end{aligned}$$

Due to the lack of the information on k, it is not easy to analytically maximize $p(D \mid \boldsymbol{\theta})$ or $\log p(D \mid \boldsymbol{\theta})$. Instead, we use the *EM (expectation-maximization) algorithm* [8].

1. Initialize randomly the parameters θ_k $(1 \le k \le K)$ and θ_{j,k,x_j} $(1 \le k \le K, 1 \le j \le J, 1 \le x_j \le V_j)$.
2. Repeat the following two steps alternately until the log-likelihood $\log p(D \mid \boldsymbol{\theta})$ converges $(m = 0, 1, 2, \ldots)$:

E-step:

$$p(k \mid \boldsymbol{x}, \boldsymbol{\theta}^{(m)}) \propto \theta_k^{(m)} \prod_{j=1}^{J} \prod_{x_j=1}^{V_j} \theta_{j,k,x_j}^{(m)} \qquad (5)$$

$$E^{(m)}[k] := \sum_{\boldsymbol{x}} N(\boldsymbol{x}) p(k \mid \boldsymbol{x}, \boldsymbol{\theta}^{(m)}) \qquad (6)$$

$$E_j^{(m)}[k, x_j] := \sum_{\boldsymbol{x}' : x'_j = x_j} N(\boldsymbol{x}') p(k \mid \boldsymbol{x}', \boldsymbol{\theta}^{(m)}) \qquad (7)$$

M-step:

$$\theta_k^{(m+1)} \propto E^{(m)}[k] \qquad (8)$$

$$\theta_{j,k,x_j}^{(m+1)} \propto E_j^{(m)}[k, x_j] \qquad (9)$$

Figure 3: The EM algorithm for ML estimation in naive Bayes models.

Fig. 3 is the EM algorithm derived for naive Bayes models.[2]

3.1.2 Parameter estimation based on MAP

It is well-known in the machine learning literature that ML estimation often suffers from the problem of *data-sparseness* when the data size N is not so large compared to the number of parameters. One way for avoiding this problem is to take a Bayesian approach, in which we consider a *prior distribution* $p(\boldsymbol{\theta})$ on the parameter space Θ. As is often done, in NBCT, we introduce a *Dirichlet distribution* for the prior distribution:

[2] In this document, the symbol '$\propto$' means a substitution with normalization — that is, we implicitly make the values in the left hand side form a probability distribution (that is, making them sum to unity). For example, in Eq. 5, $p(k \mid \boldsymbol{x}, \boldsymbol{\theta}^{(m)})$ is actually computed by:

$$p(k \mid \boldsymbol{x}, \boldsymbol{\theta}^{(m)}) = \frac{\theta_k^{(m)} \prod_{j=1}^{J} \prod_{x_j=1}^{V_j} \theta_{j,k,x_j}^{(m)}}{\sum_{k'=1}^{K} \theta_{k'}^{(m)} \prod_{j=1}^{J} \prod_{x_j=1}^{V_j} \theta_{j,k',x_j}^{(m)}}.$$

1. Initialize randomly the parameters θ_k ($1 \le k \le K$) and θ_{j,k,x_j} ($1 \le k \le K, 1 \le j \le J, 1 \le x_j \le V_j$).
2. Repeat the following two steps alternately until the log of the a posteriori probability $p(\boldsymbol{\theta} \mid D)$ converges ($m = 0, 1, 2, \ldots$):

E-step:

$$p(k \mid \boldsymbol{x}, \boldsymbol{\theta}^{(m)}) \propto \theta_k^{(m)} \prod_{j=1}^{J} \prod_{x_j=1}^{V_j} \theta_{j,k,x_j}^{(m)} \quad (12)$$

$$E^{(m)}[k] := \sum_{\boldsymbol{x}} N(\boldsymbol{x}) p(k \mid \boldsymbol{x}, \boldsymbol{\theta}^{(m)}) \quad (13)$$

$$E_j^{(m)}[k, x_j] := \sum_{\boldsymbol{x}' : x'_j = x_j} N(\boldsymbol{x}') p(k \mid \boldsymbol{x}', \boldsymbol{\theta}^{(m)}) \quad (14)$$

M-step:

$$\theta_k^{(m+1)} \propto E^{(m)}[k] + (\alpha_k - 1) \quad (15)$$

$$\theta_{j,k,x_j}^{(m+1)} \propto E_j^{(m)}[k, x_j] + (\alpha_{j,k,x_j} - 1) \quad (16)$$

Figure 4: The EM algorithm for MAP estimation in naive Bayes models.

$$p(\boldsymbol{\theta}) = \frac{1}{Z} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \prod_{j=1}^{J} \prod_{x_j=1}^{V_j} \theta_{j,k,x_j}^{\alpha_{j,k,x_j} - 1} \quad (10)$$

$$Z = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)} \cdot \frac{\prod_{k=1}^{K} \prod_{j=1}^{J} \prod_{x_j=1}^{V_j} \Gamma(\alpha_{j,k,x_j})}{\prod_{k=1}^{K} \prod_{j=1}^{J} \Gamma(\sum_{x_j}^{V_j} \alpha_{j,k,x_j})}, \quad (11)$$

where Z is a normalizing constant, and α_k and α_{j,k,x_j} are called *hyper-parameters*, each of which corresponds to θ_k and θ_{j,k,x_j} , respectively ($1 \le k \le K$, $1 \le j \le J$, $1 \le x_j \le V_j$).

Instead of maximizing $p(D \mid \boldsymbol{\theta})$, we try to maximize $p(\boldsymbol{\theta} \mid D)$, an *a posteriori probability* of $\boldsymbol{\theta}$ given the dataset D. That is, we also have:

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\mathrm{MAP}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid D) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid D) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}) p(D \mid \boldsymbol{\theta})}{p(D)} \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \Big\{ \log p(\boldsymbol{\theta}) + \log p(D \mid \boldsymbol{\theta}) \Big\}. \end{aligned} \quad (17)$$

This procedure is usually called *MAP (maximum a posteriori) estimation*.

The EM algorithm for MAP estimation is shown in Fig. 4. The algorithm is obtained by modifying the convergence condition and the procedure in M-step of the ML version (Fig. 3). Let us introduce $\delta_k \stackrel{\text{def}}{=} \alpha_k - 1$ and $\delta_{j,k,x_j} \stackrel{\text{def}}{=} \alpha_{j,k,x_j} - 1$ as the *pseudo counts* or the *smoothing constants* in estimating the corresponding parameters. If we let δ_k and δ_{j,k,x_j} be positive, the estimated parameters will also be positive, so we would be able to avoid the problem of data-sparseness.

For simplicity, these pseudo counts are set to be uniform. When specifying all δ_k and δ_{j,k,x_j} to be 1.0, the estimation procedure is sometimes called *Laplace's estimation*. In AutoClass [4], all δ_k are fixed at $1/K$, and all δ_{j,k,x_j} are fixed at $1/V_j$ ($1 \le k \le K$, $1 \le j \le J$). The *BDeu metric* [3, 9] assumes that α_k ($= 1 + \delta_k$) $= \alpha/K$ and α_{j,k,x_j} ($= 1 + \delta_{j,k,x_j}$) $= \alpha/KV_j$ for some common $\alpha > 0$,[3] where α is called the *equivalent sample size*.

3.1.3 Membership distribution

Given the estimated parameters $\hat{\boldsymbol{\theta}}$ ($\hat{\boldsymbol{\theta}}_{\mathrm{ML}}$ or $\hat{\boldsymbol{\theta}}_{\mathrm{MAP}}$) and an attribute vector $\boldsymbol{x}$ of some object, we can obtain a *membership distribution* $p(k \mid \boldsymbol{x}, \boldsymbol{\theta})$ under which the object belongs to the class k. The probabilities are computed by:

$$\begin{aligned} p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) &= \frac{p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}})}{p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}})} \\ &\propto p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}}) = \hat{\theta}_k \prod_{j=1}^{J} \hat{\theta}_{j,k,x_j} . \end{aligned} \quad (18)$$

Since we often consider that a class summarizes some (hidden) characteristic of an object, membership distributions play an important role in probabilistic inferences based on naive Bayes models.

In addition, we may consider the *attribute membership distributions* $p(k \mid x_j)$, where x_j is the j-th attribute value of an object, and k is the class to which the object belongs. Similarly to the above, the probabilities are computed as follows:

$$p(k \mid x_j, \hat{\boldsymbol{\theta}}) \propto \hat{\theta}_k \hat{\theta}_{j,k,x_j} . \quad (19)$$

[3] Of course this is a specialized form for naive Bayes models.

3.1.4 Dissimilarity

Since the membership distributions (§3.1.3) $p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}})$ can be considered to characterize the object represented by $\boldsymbol{x}$, we may measure the dissimilarity Δ between two objects $\boldsymbol{x}$ and $\boldsymbol{x}'$ by the Kullback-Liebler divergence between the corresponding membership distributions:

$$\begin{aligned}\Delta(\boldsymbol{x}, \boldsymbol{x}') &= \mathrm{KL}(p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) \parallel p(k \mid \boldsymbol{x}', \hat{\boldsymbol{\theta}})) \\ &= \sum_{k=1}^{K} p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) \log \frac{p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}})}{p(k \mid \boldsymbol{x}', \hat{\boldsymbol{\theta}})}.\end{aligned} \tag{20}$$

3.1.5 Clustering

As is mentioned above, clustering is a task to partition the objects in the dataset D into clusters of similar ones. One way is to regard each class as a cluster, and then to classify each object, whose attribute values are $\boldsymbol{x}$, into its most probable cluster (class) k^*. More specifically, based on the estimated parameters $\hat{\boldsymbol{\theta}}$, we compute k^* by using a membership probability as a score for clustering:

$$k^* = \operatorname*{argmax}_{k:1\le k\le K} p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) = \operatorname*{argmax}_{k:1\le k\le K} p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}}). \tag{21}$$

In this sense, naive Bayes clustering can be seen as an *unsupervised* classification task based on a naive Bayes model.

Furthermore, another clustering score can be considered. First, we rewrite the dissimilarity $\Delta(\boldsymbol{x}, \boldsymbol{x}')$ as follows:

$$\Delta(\boldsymbol{x}, \boldsymbol{x}') = \textstyle\sum_{k=1}^{K} \Delta_k(\boldsymbol{x}, \boldsymbol{x}'), \tag{22}$$

$$\Delta_k(\boldsymbol{x}, \boldsymbol{x}') = p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) \log \frac{p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}})}{p(k \mid \boldsymbol{x}', \hat{\boldsymbol{\theta}})}, \tag{23}$$

where each $\Delta_k(\boldsymbol{x}, \boldsymbol{x}')$ can be interpreted as the significance of class k's contribution to the dissimilarity between $\boldsymbol{x}$ and $\boldsymbol{x}'$. Then, the cluster k^* for the object $\boldsymbol{x}$ is obtained by:

$$\begin{aligned}k^* &= \operatorname*{argmax}_{k:1\le k\le K} \Delta_k(\boldsymbol{x}, \neg\boldsymbol{x}) \\ &= \operatorname*{argmax}_{k:1\le k\le K} p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) \log \frac{p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}})}{p(k \mid \neg\boldsymbol{x}, \hat{\boldsymbol{\theta}})},\end{aligned} \tag{24}$$

where we consider that k^* is the most contributing class to make $\boldsymbol{x}$ and $\neg\boldsymbol{x}$, the objects other than $\boldsymbol{x}$, being dissimilar. To compute $\Delta_k(\boldsymbol{x}, \neg\boldsymbol{x})$, it should be noted that $p(k, \neg\boldsymbol{x} \mid \hat{\boldsymbol{\theta}}) = p(k \mid \hat{\boldsymbol{\theta}}) - p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}})$ and $p(\neg\boldsymbol{x} \mid \hat{\boldsymbol{\theta}}) = 1 - p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}})$. $\Delta_k(\boldsymbol{x}, \neg\boldsymbol{x})$, the second clustering score in NBCT, can be seen as the *weighted log-odds ratio*. This score is just a modification of the relevance score between a particular class and a particular attribute value, proposed in [7] (see the next section).

Some may be interested in clustering of attribute values for each attribute. Given an attribute value x_j of j-th attribute, the most probable cluster is predicted as follows:

$$\begin{aligned}k^* &= \operatorname*{argmax}_{k:1\le k\le K} p(k \mid x_j, \hat{\boldsymbol{\theta}}) = \operatorname*{argmax}_{k:1\le k\le K} p(k, x_j \mid \hat{\boldsymbol{\theta}}) \\ &= \operatorname*{argmax}_{k:1\le k\le K} \theta_k \theta_{j,k,x_j}.\end{aligned} \tag{25}$$

3.1.6 Relevance analysis

Using the estimated parameters $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$, we may want to know the most relevant objects to the class k of interest. One promising way is to rank the objects $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$ according to the magnitude of $R_{\mathrm{MP}}(k, \boldsymbol{x}) \stackrel{\mathrm{def}}{=} p(k \mid \boldsymbol{x})$, i.e. the membership probability [16]. To understand this ranking, let us transform the probability as follows:

$$\begin{aligned}R_{\mathrm{MP}}(k, \boldsymbol{x}) &= p(k \mid \boldsymbol{x}, \boldsymbol{\theta}) \\ &= \frac{p(k \mid \boldsymbol{\theta}) p(\boldsymbol{x} \mid k, \boldsymbol{\theta})}{p(\boldsymbol{x} \mid \boldsymbol{\theta})} \propto \frac{p(\boldsymbol{x} \mid k, \boldsymbol{\theta})}{p(\boldsymbol{x} \mid \boldsymbol{\theta})}.\end{aligned} \tag{26}$$

Then, $R_{\mathrm{MP}}(k, \boldsymbol{x})$ can be seen as the significance of $p(\boldsymbol{x} \mid k, \boldsymbol{\theta})$ compared to $p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} p(k \mid \boldsymbol{\theta}) p(\boldsymbol{x} \mid k, \boldsymbol{\theta})$, the unconditional (or averaged) probability of $\boldsymbol{x}$ being occurred. In some applications, these relevant objects would be a help for characterizing the cluster [10, 16].

Another ranking score is the weighted log-odds ratio [7]. That is, for the target class k, the weighted log-odds ratio w.r.t. $\boldsymbol{x}$, denoted by $R_{\mathrm{WODD}}(k, \boldsymbol{x})$, is computed as follows:

$$R_{\mathrm{WODD}}(k, \boldsymbol{x}) \stackrel{\mathrm{def}}{=} p(\boldsymbol{x} \mid k, \boldsymbol{\theta}) \log \frac{p(\boldsymbol{x} \mid k, \boldsymbol{\theta})}{p(\boldsymbol{x} \mid \neg k, \boldsymbol{\theta})}. \tag{27}$$

Similarly to the discussion made in §3.1.5, we can see that $\sum_{\boldsymbol{x}} R_{\mathrm{WODD}}(k, \boldsymbol{x}) = \mathrm{KL}(p(\boldsymbol{x} \mid k, \boldsymbol{\theta}) \parallel p(\boldsymbol{x} \mid \neg k, \boldsymbol{\theta}))$, and hence $R_{\mathrm{WODD}}(k, \boldsymbol{x})$ is a contribution by $\boldsymbol{x}$ to the dissimilarity between k and the classes other than k. In other words, ranking by $R_{\mathrm{WODD}}(k, \boldsymbol{x})$ means that the most relevant objects to the class k are the ones that discriminate k most significantly from the others.

Also we can consider the attribute-wise version of the two ranking scores above. Both $R^j_{\mathrm{MP}}(k, x_j)$ and $R^j_{\mathrm{WODD}}(k, x_j)$ indicate the relevance between the class k and the value x_j of the attribute j, and are defined as follows:

$$R^j_{\mathrm{MP}}(k, x_j) \stackrel{\mathrm{def}}{=} p(k \mid x_j, \boldsymbol{\theta}) \tag{28}$$

$$R^j_{\mathrm{WODD}}(k, x_j) \stackrel{\mathrm{def}}{=} p(x_j \mid k, \boldsymbol{\theta}) \log \frac{p(x_j \mid k, \boldsymbol{\theta})}{p(x_j \mid \neg k, \boldsymbol{\theta})} . \tag{29}$$

These scores measure the degree of relevance between a particular class and a particular attribute value, and so would be useful in the case where we are interested in the behavior of each attribute (e.g. in distributional clustering [13]).

3.1.7 Model selection

In clustering, we are often in question of how to determine the number of clusters. This problem can be seen as a special case of model selection, and in NBCT, we attempt to find a solution in a Bayesian approach. To be specific, we first consider the joint distribution $p(D, M, \boldsymbol{\theta})$ of complete data D, a probabilistic model M, and its parameters $\boldsymbol{\theta}$. $p(D, M, \boldsymbol{\theta})$ is factored as $p(D \mid M, \boldsymbol{\theta})p(\boldsymbol{\theta} \mid M)p(M)$ by the chain rule, where $p(M)$ is the prior distribution of the model M, $p(\boldsymbol{\theta} \mid M)$ is the prior distribution of the parameters $\boldsymbol{\theta}$ of the model M, and $p(D \mid M, \boldsymbol{\theta})$ is the likelihood of the data D based on the model M with the parameters $\boldsymbol{\theta}$. In naive Bayes models, for instance, each $d \in D$ corresponds to the attribute vector $\boldsymbol{x}$ of an object. Also M corresponds to K, the number of classes (clusters), though it has been omitted in the descriptions so far.

Then, from the settings above, our goal is to find the most probable model M^* based on the data D at hand, that is, we attempt to find M^* such that:

$$\begin{aligned} M^* &= \mathrm{argmax}_M \; p(M \mid D) \\ &= \mathrm{argmax}_M \; \frac{p(D \mid M)p(M)}{p(D)} \\ &= \mathrm{argmax}_M \; p(D \mid M)p(M) \\ &= \mathrm{argmax}_M \; p(D \mid M), \end{aligned} \tag{30}$$

where we assume $p(M)$ to be uniform for simplicity. Now the goal is reduced to finding M $(= M^*)$ that maximizes $p(D \mid M)$. $p(D \mid M)$ is commonly called the *marginal likelihood* of D given M, and is used as a score for model selection. The marginal likelihood can be interpreted as the expectation (or the average) of the likelihood $p(D \mid M, \boldsymbol{\theta})$ with respect to the prior distribution $p(\boldsymbol{\theta} \mid M)$:

$$\begin{aligned} p(D \mid M) &= \int_\Theta p(D, \boldsymbol{\theta} \mid M) d\boldsymbol{\theta} \\ &= \int_\Theta p(D \mid M, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid M) d\boldsymbol{\theta} \\ &= \left\langle p(D \mid M, \boldsymbol{\theta}) \right\rangle_{p(\boldsymbol{\theta} \mid M)} . \end{aligned} \tag{31}$$

If the dataset were complete data D_c, where each $d \in D_c$ is a pair $(k, \boldsymbol{x})$ of the attribute vector $\boldsymbol{x}$ of an object and the class k to which the object belongs, then $p(D_c \mid M)$ is obtained in closed form (see [6, 9] for the case with a Bayesian network). On the other hand, when the data is incomplete, as in the case of probabilistic clustering, the integral in Eq. 31 is difficult to compute. So, including MCMC (Markov chain Monte Carlo) sampling, several approximation methods of the marginal likelihood are proposed so far [5]. *Bayesian information criterion* (BIC) [15] should be the most popular 'deterministic' approximation method, in which Laplace approximation is introduced based on the large-data assumption. The *Cheeseman-Stutz score* [4, 5] is used in AutoClass. The general forms of these two scores are respectively written as follows:

$$Score_{\mathrm{BIC}}(M) \stackrel{\mathrm{def}}{=} p(D \mid M, \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}) - \frac{|\boldsymbol{\theta}|}{2} \log N \tag{32}$$

$$\begin{aligned} Score_{\mathrm{CS}}(M) \stackrel{\mathrm{def}}{=} & \; p(\tilde{D}_c \mid M) - p(\tilde{D}_c \mid M, \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}) \\ & + p(D \mid M, \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}), \end{aligned} \tag{33}$$

where N is the total size of dataset, $|\boldsymbol{\theta}|$ denotes the number of free parameters, and $\tilde{D}_c$ is pseudo complete data whose sufficient statistics are the expected counts obtained by the EM algorithm.

3.1.8 Random restarts

Since the EM algorithm is just a hill-climbing algorithm, being trapped in undesirable *local maxima* is known as one of practical problems in the EM algorithm. NBCT provides two facilities for avoiding such local maxima — random restarts [14] and the split-merge EM (SMEM) algorithm [18].

In random restarts, we first prepare n different initial parameter sets. Then, from each initial parameter set, we run a series of EM iterations, and record the converged likelihood $p(D \mid \boldsymbol{\theta})$ or the a posteriori probability

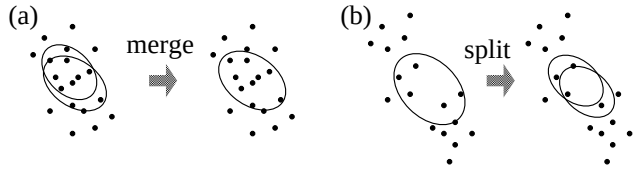

Figure 5: Image of a merge operation and a split operation, where the dots and the ovals stand for objects and clusters, respectively.

$p(\boldsymbol{\theta} \mid D)$. Finally we pick up the estimated parameters that bring the highest likelihood or the highest a posteriori probability.

3.1.9 Split-merge EM algorithm

The SMEM algorithm is applicable to mixture models, including naive Bayes models. In the SMEM algorithm, we attempt to escape from the local maxima by forcedly applying the *split operation* and the *merge operation* to unpromising clusters.

To be more specific, we merge two clusters, say k_1 and k_2, that closely overlap with each other (Fig. 5 (a)), and split a cluster, say k_3, that excessively covers objects (Fig. 5 (b)). In the paper that firstly proposed the SMEM algorithm [18], the split operation and the merge operation are always paired, so the number of resultant clusters will not change. After a split-merge operation pair executed, the EM algorithm is conducted until the convergence of likelihood $p(D \mid \boldsymbol{\theta})$ or the a posteriori probability $p(\boldsymbol{\theta} \mid D)$.

The possible triplets of clusters (k_1, k_2, k_3) are kept as prioritized candidates for the next split-merge operation. If the converged likelihood or a posteriori probability is improved with the first candidate (i.e. the candidate is said to be *accepted*), we will proceed to further split-merge operations. If there is no (significant) improvement with the first candidate, we will discard the result of the EM algorithm (i.e. the candidate is said to be *rejected*) and try the next candidate.

⋄ Prioritizing the split-merge candidates

To prioritize the split-merge candidates, we first get the pairs $\{(k_1, k_2) \mid 1 \leq k_1 < k_2 \leq K\}$ of classes to be merged, in the descending order of heuristic scores $J_{\mathrm{merge}}(k_1, k_2 \mid \boldsymbol{\theta})$. The score is defined as follows:

$$\begin{aligned} J_{\mathrm{merge}}(k_1, k_2 \mid \boldsymbol{\theta}) &\stackrel{\mathrm{def}}{=} \mathbf{p}_{k_1}(\boldsymbol{\theta})^T \mathbf{p}_{k_2}(\boldsymbol{\theta}) \\ &= \textstyle\sum_{\boldsymbol{x}} N(\boldsymbol{x}) p(k_1 \mid \boldsymbol{x}, \boldsymbol{\theta}) p(k_2 \mid \boldsymbol{x}, \boldsymbol{\theta}), \end{aligned} \tag{34}$$

where we have a multiset of objects $D = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}\}$ as the observed data, and $\mathbf{p}_k(\boldsymbol{\theta})$ is the vector of the membership probabilities to the class k:

$$\mathbf{p}_k(\boldsymbol{\theta}) = (p(k \mid \boldsymbol{x}^{(1)}, \boldsymbol{\theta}), p(k \mid \boldsymbol{x}^{(2)}, \theta), \ldots, P(k \mid \boldsymbol{x}^{(N)}, \theta))^T. \tag{35}$$

Intuitively, $J_{\mathrm{merge}}(k_1, k_2 \mid \boldsymbol{\theta})$ measures a (partially empirical) similarity between the classes k_1 and k_2 based on the data D. Besides, we may use the normalized version:

$$\tilde{J}_{\mathrm{merge}}(k_1, k_2 \mid \boldsymbol{\theta}) \stackrel{\mathrm{def}}{=} \frac{\mathbf{p}_{k_1}(\boldsymbol{\theta})^T \mathbf{p}_{k_2}(\boldsymbol{\theta})}{\|\mathbf{p}_{k_1}(\boldsymbol{\theta})\| \cdot \|\mathbf{p}_{k_2}(\boldsymbol{\theta})\|}. \tag{36}$$

Then, for each (k_1, k_2) pair to be merged, we also get the classes $\{k_3 \mid 1 \leq k_3 \leq K, k_3 \neq k_1, k_3 \neq k_2\}$ to be split, in the descending order of another heuristic score $J_{\mathrm{split}}(k_3 \mid \boldsymbol{\theta})$:

$$\begin{aligned} J_{\mathrm{split}}(k_3 \mid \boldsymbol{\theta}) &\stackrel{\mathrm{def}}{=} \mathrm{KL}(\tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta}) \parallel p(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})) \\ &= \sum_{\boldsymbol{x}} \tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta}) \log \frac{\tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})}{p(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})}, \end{aligned} \tag{37}$$

where $\tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})$ is a local empirical probability computed by:

$$\begin{aligned} \tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta}) &\propto \tilde{p}(\boldsymbol{x}) p(k_3 \mid \boldsymbol{x}, \boldsymbol{\theta}) \\ &\propto N(\boldsymbol{x}) p(k_3 \mid \boldsymbol{x}, \boldsymbol{\theta}). \end{aligned} \tag{38}$$

In the above, $\tilde{p}(\boldsymbol{x})$ denotes $N(\boldsymbol{x})/N$, the empirical unconditional probability of $\boldsymbol{x}$. So using $J_{\mathrm{split}}(k_3 \mid \boldsymbol{\theta})$, the class that does not fit to the data will be split earlier.

⋄ Partial EM iterations

After a split-merge operation, we should re-initialize the parameters of the modified classes. Let k_1' be the new class obtained by merging k_1 and k_2. Also we consider two classes k_2' and k_3' which are obtained by splitting k_3. Then, for the merged class k_1', we re-initialize the parameters related to k_1' as follows:

$$\theta_{k_1'} := \frac{E[k_1]\theta_{k_1} + E[k_2]\theta_{k_2}}{E[k_1] + E[k_2]} \tag{39}$$

$$\theta_{j,k_1',x_j} := \theta_{j,k_1,x_j} + \theta_{j,k_2,x_j} \quad (1 \leq j \leq J, 1 \leq x_j \leq V_j). \tag{40}$$

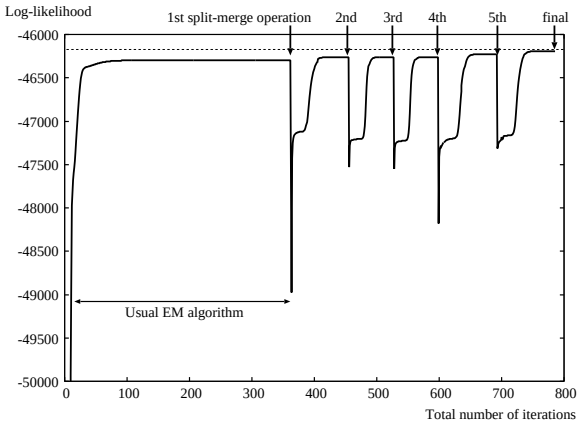

Figure 6: Changes in log-likelihood by the SMEM algorithm.

On the other hand, for the split classes k'_2 and k'_3, we re-initialize $\theta_{k'_2} := \frac{1}{2}\theta_{k_3}$, $\theta_{k'_3} := \frac{1}{2}\theta_{k_3}$, $\theta_{j,k'_2,x_j} := \theta_{j,k_3,x_j} + \epsilon$ and $\theta_{j,k'_3,x_j} := \theta_{j,k_3,x_j} + \epsilon'$ $(1 \leq j \leq J,\ 1 \leq x_j \leq V_j)$, where ϵ and ϵ' are some random noises.

In partial EM iterations, to make it reasonable to compute E-step and M-step only for the modified classes (k'_1, k'_2 and k'_3), M-step is modified as follows ($1 \leq j \leq J$, $1 \leq x_j \leq V_j$ and $k' = k'_1, k'_2, k'_3$):

$$\theta_{j,k',x_j} = \frac{E[k']}{\sum_{k''=k'_1,k'_2,k'_3} E[k'']} \cdot \sum_{k''=k'_1,k'_2,k'_3} \theta_{j,k'',x_j} \tag{41}$$

⋄ Behavior of the SMEM algorithm

Fig. 6 shows a typical pattern on the changes in log-likelihood by the SMEM algorithm, where we apply five split-merge operations. Please note here that we have only plotted a sequence with accepted split-merge operations. It is seen from Fig. 6 that the log-likelihood decreases at the moment of applying a split-merge operation, but as a whole, the log-likelihood steadily increases, and finally we can obtain the estimates with a higher log-likelihood than the one obtained by the usual EM algorithm.

3.2 VB based clustering

3.2.1 Model selection based on VB

As described in §3.1.7, to obtain the model M^* that explains best the data D at hand, we consider $M = M^*$ is the model that maximizes the marginal likelihood $p(D \mid M)$. In naive Bayes models, the optimal number K^* of classes is the number of classes in M^*. It has been also known that if D is complete data D_c, $p(D \mid M)$ can be obtained in closed form. However, when D is incomplete, i.e. there is some hidden data z such that $D_c = (D, z)$ (for instance, in naive Bayes clustering, the classes of the objects are hidden in D), some approximation method is required. In this section, we briefly describe the approximation via the VB approach.

First, let us consider log of the marginal likelihood $L(D) \stackrel{\text{def}}{=} \log p(D \mid M)$, and then we have:

$$\begin{aligned} L(D) &= \log \sum_z \int_\Theta p(D, z, \boldsymbol{\theta} \mid M) d\boldsymbol{\theta} \\ &= \log \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D, M) \frac{p(D, z, \boldsymbol{\theta} \mid M)}{q(z, \boldsymbol{\theta} \mid D, M)} d\boldsymbol{\theta} \\ &\geq \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D, M) \log \frac{p(D, z, \boldsymbol{\theta} \mid M)}{q(z, \boldsymbol{\theta} \mid D, M)} d\boldsymbol{\theta}. \\ &\quad \text{(from Jensen's inequality)} \end{aligned} \tag{42}$$

For the space limitation, we fix the model M for the moment, and simply write $p(\cdot \mid M) = p(\cdot)$ and $q(\cdot \mid D, M) = q(\cdot \mid D)$, and then obtain:

$$\begin{aligned} L(D) &\geq F[q] \\ &\stackrel{\text{def}}{=} \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \frac{p(D, z, \boldsymbol{\theta})}{q(z, \boldsymbol{\theta} \mid D)} d\boldsymbol{\theta}, \end{aligned} \tag{43}$$

where $F[q]$ can be seen as a lower limit of $L(D)$, and is called the *free energy*. So to get a good approximation of $L(D)$, we attempt to find a distribution function $q = q^*$ that maximizes a functional $F[q]$. In model selection, we use the free energy $F[q]$ as a model score.

Besides, to get another view, we have the following by considering $L(D) = \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log p(D) d\boldsymbol{\theta}$:

$$\begin{aligned} & L(D) - F[q] \\ &= \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \left\{ p(D) \cdot \frac{q(z, \boldsymbol{\theta} \mid D)}{p(D, z, \boldsymbol{\theta})} \right\} d\boldsymbol{\theta} \\ &= \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \frac{q(z, \boldsymbol{\theta} \mid D)}{p(z, \boldsymbol{\theta} \mid D)} d\boldsymbol{\theta} \\ &= \mathrm{KL}(q(z, \boldsymbol{\theta} \mid D) \parallel p(z, \boldsymbol{\theta} \mid D)). \end{aligned} \tag{44}$$

From the above, maximizing $F[q]$ implies minimizing the Kullback-Liebler divergence between $q(z, \boldsymbol{\theta} \mid D)$ and

$p(\boldsymbol{z}, \boldsymbol{\theta} \mid D)$. So finding q^* is to make a good approximation of $p(\boldsymbol{z}, \boldsymbol{\theta} \mid D)$, the conditional distribution of hidden variables and the parameters.

In VB learning, we further assume $q(\boldsymbol{z}, \boldsymbol{\theta} \mid D) \approx q(\boldsymbol{z} \mid D)q(\boldsymbol{\theta} \mid D)$, and obtain a generic form of *variational Bayesian EM (VB-EM) algorithm* as an iterative procedure consisting of the following two update rules:

$$q(\boldsymbol{z} \mid D) \propto \exp\left(\int_\Theta q(\boldsymbol{\theta} \mid D) \log p(D, \boldsymbol{z} \mid \boldsymbol{\theta}) d\boldsymbol{\theta}\right), \tag{45}$$

$$q(\boldsymbol{\theta} \mid D) \propto p(\boldsymbol{\theta}) \exp\left(\sum_{\boldsymbol{z}} q(\boldsymbol{z} \mid D) \log p(D, \boldsymbol{z} \mid \boldsymbol{\theta})\right). \tag{46}$$

Now we can derive a VB-EM algorithm specific to naive Bayes clustering, shown in Fig. 7, by substituting the distribution form of a naive Bayes model (Eq. 3) to the generic VB-EM procedure above. In Fig. 7, $\pi_k^{(m)}$ and $\pi_{j,k,x_j}^{(m)}$ are defined as follows:

$$\pi_k^{(m)} \stackrel{\text{def}}{=} \exp\left(\Psi\left(\alpha_k^{(m)}\right) - \Psi\left(\sum_{k'=1}^K \alpha_{k'}^{(m)}\right)\right), \tag{47}$$

$$\pi_{j,k,x_j}^{(m)} \stackrel{\text{def}}{=} \exp\left(\Psi\left(\alpha_{j,k,x_j}^{(m)}\right) - \Psi\left(\sum_{x'_j=1}^{V_j} \alpha_{j,k,x'_j}^{(m)}\right)\right), \tag{48}$$

where $\Psi(\cdot)$ is the digamma function. It can be noticed that the resultant VB-EM algorithm receives the data D as input, and outputs the learned hyper-parameters $\boldsymbol{\alpha}^*$ (or the pseudo counts $\boldsymbol{\delta}^*$).

In NBCT, $\alpha_k^{(0)}$ and $\alpha_{j,k,x_j}^{(0)}$ are given as follows, where α_k and α_{j,k,x_j} are specified by the user, and ϵ_k and ϵ_{j,k,x_j} are small random noises:

$$\alpha_k^{(0)} := \alpha_k + \epsilon_k\,, \tag{49}$$

$$\alpha_{j,k,x_j}^{(0)} := \alpha_{j,k,x_j} + \epsilon_{j,k,x_j}\,. \tag{50}$$

3.2.2 Clustering

As in the ML/MAP case (§3.1.5), we may regard the cluster k^* which has the highest membership probability for an object $\boldsymbol{x}$ as the cluster to which the object belongs. The difference is that we use the distribution $p(k, \boldsymbol{x}) = \int_\Theta q^*(\boldsymbol{\theta} \mid D) p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) d\boldsymbol{\theta}$, an averaged distribution on the prior distribution $q^*(\boldsymbol{\theta} \mid D)$ with learned hyper-parameters $\boldsymbol{\alpha}^*$, instead of $p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}})$ which relies on the point-estimated value $\hat{\boldsymbol{\theta}}$. That is, we have:

$$\begin{aligned} k^* &= \operatorname*{argmax}_{k:1\le k\le K} p(k \mid \boldsymbol{x}) = \operatorname*{argmax}_{k:1\le k\le K} p(k, \boldsymbol{x}) \\ &= \operatorname*{argmax}_{k:1\le k\le K} \int_\Theta q^*(\boldsymbol{\theta} \mid D) p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) d\boldsymbol{\theta} \\ &= \operatorname*{argmax}_{k:1\le k\le K} \bar{\theta}_k \prod_{j=1}^J \bar{\theta}_{j,k,x_j}\,, \end{aligned} \tag{56}$$

where $\bar{\theta}_k$ and $\bar{\theta}_{j,k,x_j}$ are obtained in closed form:

$$\bar{\theta}_k = \alpha_k^* / \textstyle\sum_{k'} \alpha_{k'}^* \tag{57}$$

$$\bar{\theta}_{j,k,x_j} = \alpha_{j,k,x_j}^* / \textstyle\sum_{x'_j} \alpha_{j,k,x'_j}^*\,. \tag{58}$$

1. Initialize randomly the hyper-parameters $\alpha_k^{(0)}$ $(1 \le k \le K)$ and $\alpha_{j,k,x_j}^{(0)}$ $(1 \le k \le K, 1 \le j \le J, 1 \le x_j \le V_j)$.
2. Repeat the following two steps alternately until the free energy $F[q]$ converges $(m = 0, 1, 2, \ldots)$:

E-step:

$$q^{(m)}(k \mid \boldsymbol{x}) \propto \pi_k^{(m)} \prod_{j=1}^J \pi_{j,k,x_j}^{(m)} \tag{51}$$

$$E^{(m)}[k] := \sum_{\boldsymbol{x}} N(\boldsymbol{x}) q^{(m)}(k \mid \boldsymbol{x}) \tag{52}$$

$$E_j^{(m)}[k, x_j] := \sum_{\boldsymbol{x}': x'_j = x_j} N(\boldsymbol{x}') q^{(m)}(k \mid \boldsymbol{x}') \tag{53}$$

M-step:

$$\alpha_k^{(m+1)} := \alpha_k^{(0)} + E^{(m)}[k] \tag{54}$$

$$\alpha_{j,k,x_j}^{(m)} := \alpha_{j,k,x_j}^{(0)} + E_j^{(m)}[k, x_j] \tag{55}$$

Figure 7: The VB-EM algorithm in naive Bayes models.

3.2.3 Other inference tasks

In VB learning, it seems not easy in straightforward ways to conduct the probabilistic inferences other than clustering based on the membership distribution (§3.2.2) — clustering based on the weighted log-odds ratio (Eq. 24), computing dissimilarities (Eq. 20), or relevance analysis (Eqs. 26 and 27). One compromise is to use the parameters $\bar{\boldsymbol{\theta}}$ obtained by Eqs. 57 and 58 instead of $\hat{\boldsymbol{\theta}}$, the point-estimated parameters in ML/MAP [2], and actually NBCT adopts this method. It is important to note

however that the resultant probabilistic inferences are *not* guaranteed to be good approximations of the inferences based on a posteriori quantities.

3.2.4 Random restarts and the SMEM algorithm

Basically, random restarts and the SMEM algorithm for VB-EM (say, VB-SMEM) follow almost the same procedures as the ones for the ML/MAP-EM algorithms (§3.1.8 and §3.1.9). The difference is that we need to use the parameters $\bar{\boldsymbol{\theta}}$ obtained by Eqs. 57 and 58 to compute two heuristic scores J_{merge} and J_{split} (Eqs. 34–37).

4 How to use NBCT

4.1 Overall organization of NBCT

NBCT 0.1 provides three executables named `nbc`, `vnbc` and `nbcsep`:

- `nbc` covers probabilistic inferences for ML/MAP-based clustering (§3.1).
- `vnbc` covers probabilistic inferences for VB-based clustering (§3.2).
- `nbcsep` is an auxiliary tool for post-processing of the output files from `nbc` or `vnbc`.

See §4.2 for the installation procedure. The usages of `nbc`, `vnbc` and `nbcsep` are described in §4.3, §4.4 and §4.5, respectively.

4.2 Installation

4.2.1 Preliminary installation (GNU libiconv)

To handle the dataset with non-ASCII-safe character encodings, NBCT provides a way to utilize GNU libiconv (`http://www.gnu.org/software/libiconv/`). If you are sure that the target dataset only contains ASCII characters or is ASCII-safe, GNU libiconv will not be required. If GNU libiconv is not installed on the user's environment, the user needs to install GNU libiconv in advance of the installation of NBCT.

On Windows, a DLL file `iconv.dll` should be placed in the folder specified by the `PATH` environment variable, or in one of the system folders (such as `c:\WINDOWS\system32`). We can download a zip archive `http://ftp.gnu.org/pub/gnu/libiconv/libiconv-1.9.1.bin.woe32.zip`, which includes a couple of executables and DLL files already built. On the other hand, the latest version can be built from the source files.[4]

4.2.2 Contents of the package

The original package of NBCT is distributed as a source tarball (`nbct-0.1.tar.gz`) or a zip archive (`nbct-0.1_win.zip`) including Windows executables. After the package unfolded, the user may find the following subdirectories (or sub-folders):

- `src/` contains C source code.
- `doc/` contains the document files including this manual.
- `bin/` contains executables or script files (but might be empty just after unfolding).
- `example/` contains data examples.

For the users who attempt to use the executables in `nbct-0.1_win.zip` on Windows, the installation is quite easy — that is, you place these executables in the folder that appears in the `PATH` environment variable. However it should be noted that the executables do not link with GNU libiconv, and so are only applicable to the datasets with ASCII-safe characters.

4.2.3 Using `configure`

For Linux, Mac OS X, and Win32 with Cygwin or MinGW, we can use the `configure` script in the package. That is, after moving to the `src` directory, just type (if necessary, please specify some appropriate *Options* to your system):

```
% ./configure Options
% make
```

Note that the symbol '`%`' is the shell prompt. For the details on *Options*, please consult the `INSTALL` file in the `src` directory. Also,

[4] We can use GNU C Compiler on Cygwin (+ MinGW) or MSVC++, for example. With MSVC++, please follow the instructions in `README.woe32`. However several modifications seem to be required for GNU libiconv 1.11, the latest version as of Aug. 14, 2007.

```
% ./configure --help
```

will show the detailed descriptions of the options for the `configure` script.

4.2.4 Using `Makefile.msvc`

To build NBCT on Win32 with MSVC++, we attempt to compile the source code by the `cl` command of VC++. Please follow the steps below:

1. Edit `src/Makefile.msvc` as suitable for your environment.
2. Invoke the Command Prompt window prepared for MSVC++. For instance, if you are using MS Visual Studio .NET 2003, please follow the menus: [Start] → [All Programs] → [Microsoft Visual Studio .NET 2003] → [Visual Studio .NET Tools] → [Visual Studio .NET 2003 Command Prompt].
3. At the Command Prompt invoked, visit the `src` folder.
4. Type the following command to compile NBCT:
   ```
   nmake -f Makefile.msvc
   ```
5. Type the following command to install NBCT:
   ```
   nmake -f Makefile.msvc install
   ```
   By default (i.e. without modifying `Makefile.msvc`), all executables will be copied into the `bin` directory of the unfolded package.

4.3 ML/MAP based clustering

4.3.1 General description

As is mentioned above, we use the executable `nbc` for ML/MAP based clustering. We can pass our settings and tasks to `nbc` through the command line arguments:[5]

```
nbc -f Base -x NumAttr \
  [-k NumClass|-R|-H] Options...
```

The above says that `-f` and `-x` are mandatory. Here '`-f` *Base*' indicates that we have a file named *Base*`_freq.txt`, which contains a set D of the objects to be clustered. *Base* will also be used as the base name of the input/output files listed in Table 2. *NumAttr*, given by `-x`, indicates the number J of attributes of the naive Bayes model we use (see §2.2 for the description of naive Bayes models). In addition, one of `-k`, `-R` and `-H` is required:

Table 2: Input/Output files for `nbc`.

	Filename	Content
Input	*Base*`_freq.txt`	Occurrences of objects
Input/Output	*Base*`_param.txt`	Parameters $\boldsymbol{\theta}$
Input	*Base*`_smooth.txt`	Pseudo counts $\boldsymbol{\delta}$
Output	*Base*`_dissim.txt`	Dissimilarities
Output	*Base*`_memp.txt`	Membership probs.
Output	*Base*`_cluster.txt`	Clustering results
Output	*Base*`_rank.txt`	Result of relevance analysis
Output	*Base*`_log.txt`	Logs
Output	*Base*`_msg.txt`	Messages on the display
Output	*Base*`_smem.txt`	Logs on the SMEM algorithm

- When the number K of classes is given by '`-k` *NumClass*', `nbc` will first run the EM algorithm (§3.1.1 and §3.1.2) to estimate the parameters from *Base*`_freq.txt`. Then, probabilistic inferences including clustering (§3.1.3–§3.1.6) will be conducted based on the estimated parameters.
- When `-R` is given, `nbc` will skip the EM algorithm and read the parameters from a file named *Base*`_param.txt`, which may have been created by manual or by a previous run of `nbc`. The number of classes is then determined according to the content of *Base*`_param.txt`.
- When `-H` is given, `nbc` will run the MAP-based EM algorithm (§3.1.2) under the pseudo counts (the smoothing constants) read from a file named *Base*`_smooth.txt`, which may have been created by a previous run of `vnbc` (§4.4). The number of classes is determined according to the content of *Base*`_smooth.txt`.

Other optional flags will be described in §4.3.3.

[5] In this document, the symbol '\' just means a continuation of the command line, so please do not type '\' itself.

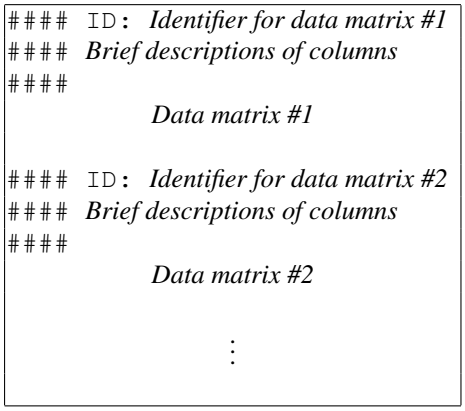

Figure 8: Common file format in NBCT.

4.3.2 File format

Fig. 8 illustrates the file format that is common to all NBCT's input/output files. In this format, a file contains several data matrices each of which has a header part, and the header part consists of the lines starting with '#'. A header part includes the identifier to the corresponding data matrix and gives brief descriptions of columns.

In the area of a data matrix, a line corresponds to a row of the matrix, and is separated to several fields each of which corresponds to a column of the matrix. Either tab characters or commas are allowed to be the delimiters of fields. A file which uses the tab characters (resp. commas) as delimiters is said to be in the *tab-separated format* (resp. the *CSV format*). Please note that the CSV format used here is just a restricted one — the current version of NBCT does not understand the values enclosed by double quote characters.

4.3.3 Command line options

◇ EM algorithm

The ML/MAP-based EM algorithms for naive Bayes models are described in §3.1.1 and §3.1.2.

`-c` *ClassPseudoCount*

This option specifies a uniform pseudo count for the class variable. That is, for each class k, the pseudo count δ_k is equally set to *ClassPseudoCount*. When '`auto`' is specified for *ClassPseudoCount*, the pseudo counts are set to the ones used in AutoClass (§3.1.2). The default value is 0.

`-w` *AttrPseudoCount*

This option specifies uniform pseudo counts for the attribute variables. If *AttrPseudoCount* takes the form '$\zeta_1,\zeta_2,\cdots,\zeta_J$', for each attribute j, the pseudo counts δ_{j,k,x_j} are equally set to ζ_j. If *AttrPseudoCount* is just a non-negative number ζ, all pseudo counts δ_{j,k,x_j} for attributes are equally set to ζ. When '`auto`' is specified for ζ or for each ζ_j, the corresponding pseudo counts are set to the ones used in AutoClass (§3.1.2). The default values are all 0.

`-V` *MinPseudoCount*

With this option, `nbc` runs the MAP-based EM algorithm under the pseudo counts used in the BDeu metric (§3.1.2), where *MinPseudoCount* is the minimum among them. That is, `nbc` first computes $\alpha = KV_{\max} \cdot (1 + \mathit{MinPseudoCount})$, where $V_{\max} = \max_{1\le j\le J} V_j$, and then sets uniform pseudo counts $\delta_k = \alpha/K - 1$ and $\delta_{j,k,x_j} = \alpha/KV_j - 1$.[6] This option is prioritized over the `-c` or the `-w` options.

`-r` *RandomSeed*

`nbc` will use *RandomSeed* as a random seed for initialization of parameters in the EM algorithm (Step 1 in Fig. 3 and Fig. 4).

`-e` *Threshold*

`nbc` will use *Threshold* as the threshold ξ for judging the convergence of the EM algorithm (Step 2 in Fig. 3 and Fig. 4). That is, if the difference between the value of log-likelihood (or log of a posteriori probability) before the update and the one after the update becomes less than ξ, we will consider that the parameters have been converged. The default value is 10^{-5}.

`-m` *MaxIter*

This option indicates the maximum number of iterations to be performed is *MaxIter*. That is, `nbc` will stop the EM iteration when the number of iterations exceeds *MaxIter*. If this option is omitted or *MaxIter* = 0, the EM iterations will be continued until the convergence.

`-n` *NumInit*

[6] This indirect way to specify the equivalent sample size α would be useful for the cases where K varies and it is tedious to identify V_j from the data in advance.

With this option, random restarts will be enabled in the EM algorithm (§3.1.8), where the number of initial trials is *NumInit*. By default, *NumInit* is set to 1, that is, `nbc` will not perform random restarts.

`-l` *MaxInitIter*

With this option, the maximum number of preliminary EM iterations in random restarts (§3.1.8) will be set to *MaxInitIter*.

`-I` *InitClassMethod*

This option specifies the initialization method for the class parameters (Step 1 in Fig. 3 and Fig. 4). There are two alternatives — '`noisy_u`' initializes the parameters based on a uniform distribution with small noises, and '`random`' initializes the parameters more randomly. The default method is '`noisy_u`'.

`-J` *InitAttrMethod*

This option specifies the initialization method for the attribute parameters (Step 1 in Fig. 3 and Fig. 4). Similarly to the `-I` flag, '`noisy_u`' initializes the parameters based on a uniform distribution with small noises, and '`random`' initialize the parameters more randomly. In addition, '`v_freq`' initializes the parameters based on the empirical frequencies with small noises. That is, each attribute parameter θ_{j,k,x_j} will be initialized to $\frac{1}{N}\sum_{\boldsymbol{x}':x'_j=x_j} N(\boldsymbol{x}')$ with a small noise. The default method is '`noisy_u`'.

`-M` (no argument)

With this option, the EM algorithm will save the memory space. However it should be noted that this option may slow down the EM algorithm, and that, for an implementational reason, this option is not available in the SMEM algorithm.

◇ Membership distribution

`-D` (no argument)

This option enables the computation of the membership distributions $p(k \mid \boldsymbol{x})$ (§3.1.3), and the results will be written into a file named '*Base*`_memp.txt`'.

◇ Dissimilarity

`-d` (no argument)

This option enables the computation of the dissimilarities between objects, and the dissimilarities between attribute values (§3.1.4). The computed dissimilarities are written into '*Base*`_dissim.txt`'. Since `nbc` will try to compute the dissimilarities for all possible pairs of objects in the dataset D, it should take a quite long time for large D.

◇ Clustering

`-t` *ClusterScore*

This option enables clustering (§3.1.5), where the clustering score is *ClusterScore*. When *ClusterScore* is given as '`mp`', we will use the score in Eq. 21. When *ClusterScore* is '`wodd`', we will use the score in Eq. 24. The results of clustering will be written into '*Base*`_cluster.txt`'. The default score is '`mp`'.

◇ Relevance analysis

`-z` *NumRank*

With this option, `nbc` will enable the relevance analysis (§3.1.6), in which the most relevant objects or attribute values to each class (each cluster) are written into '*Base*`_rank.txt`'. *NumRank* is the number of such relevant objects or attribute values. If *NumRank* is set to 0, the ranking over all objects or attribute values will be recorded.

`-Q` *RelvScore*

This option specifies the score for relevance analysis (§3.1.6). When *RelvScore* is given as '`mp`', we will use the score in Eq. 26. On the other hand, when *RelvScore* is given as '`wodd`', we will use the score in Eq. 27. The default score is '`mp`'.

◇ Model selection

If we give an option '`-k` *Kmin*`:`*Kmax*`:`*Kstep*', `nbc` will enter into the 'model scoring' mode (§3.1.7) — that is, if we specify '`-k 3:10:2`', `nbc` will compute the scores of the models with the number of classes $K = 3, 5, 7, 9$. For the environments in which the character '`:`' is not

allowed to be used in the command line, we can use ‘,’ or ‘−’ instead.

`-g` *ModelScore*

This option specifies the model score as *ModelScore*. The available model scores are ‘`bic`’ (BIC, Eq. 32) and ‘`cs`’ (the Cheeseman-Stutz score, Eq. 33). The default score is BIC.

◇ SMEM algorithm

Since the number of split-merge candidates is $K(K-1)(K-2)$, where K is the number of classes, the SMEM algorithm could take a very long time to find a better set of parameters. `nbc` provides several (ad-hoc) control flags not to explore unpromising search space.

`-o` *NumOp*

This option enables the SMEM algorithm, where the number of operations is set to *NumOp*.

`-a` *NumCand*

This option specifies the maximum number of the split-merge candidates to be examined.

`-G` *NumSplitCand*

This option limits the number of the split candidates to be taken into account for each merge candidate.

`-U` *MinClassPar*

This option specifies the lower limit of the class probability of a class to be split. That is, if the class probability of a split candidate is lower than *MinClassPar*, `nbc` will skip the candidate. If this option is omitted, no limit will be set.

`-C` *MinNewClassPar*

This option specifies the lower limit of the class probability of a new class obtained by a split-merge operation. That is, if one of the probabilities of three newly obtained classes is lower than *MinNewClassPar*, the result of the split-merge operation will be discarded. If this option is omitted, no limit will be set.

`-N` *MinImprove*

This option specifies the lower limit of the improvement of a split-merge operation. That is, the improvement of such a operation does not exceed *MinImprove*, the result of the operation will be discarded. The default value is $10 \times \xi$, where ξ is the threshold for judging convergence of the EM algorithm, which is specified with the `-e` flag.

`-F` (no argument)

This option enables the normalized version of J_{merge} (Eq. 36), instead of the original one (Eq. 34).

◇ Input/Output files

With GNU libiconv, we can handle datasets with non-ASCII-safe character codes. The default codeset can be specified at the installation time, by the `configure` script (§4.2.3) or `Makefile.msvc` (§4.2.4). Without such a configuration, the default codeset will be set to UTF-8. Running ‘`iconv -l`’ shows a list of the available codesets. Also, as described in §4.3.2, each input/output file is either in the CSV format or in the tab-separated format.

`-s` *Suffix*

`nbc` adds a suffix *Suffix* to the base name of the output files. For example, ‘*Base*`_param.txt`’ will be changed as ‘*Base_Suffix*`_param.txt`’.

`-p` *Charcode* (Available only with GNU libiconv)

`nbc` assumes the character codeset of the input file is *Charcode*. If this flag is omitted, the default codeset will be used.

`-q` *Charcode* (Available only with GNU libiconv)

`nbc` outputs the files with the character codeset *Charcode*. If this flag is omitted, the default codeset will be used.

`-i` (no argument)

`nbc` assumes the input file is in the CSV format, and has a file extension ‘`csv`’. If this flag is omitted, the tab-separated format will be assumed.

`-j` (no argument)

`nbc` outputs the files in the CSV format whose file extension is ‘`csv`’. If this flag is omitted, `nbc` outputs them in the tab-separated format.

◇ **Miscellaneous**

`-E` (no argument)

In some domains, the data file '*Base*_`freq.txt`' may contain the class information annotated by human. While a future version of NBCT is planned to provide a facility for evaluation based on such class information, in the current version, we will ignore such class information when this option is given. It is assumed that the class information is placed in the left-most column.

`-S` (no argument)

By default, `nbc` outputs the messages to the standard error output. This option switches the output to a file named '*Base*_`msg.txt`'. Then the buffer will not be flushed for each dot symbol, so for a relatively small dataset, this option would make the execution of the program more efficient.

`-Y` (no argument)

By default, the attribute values output from `nbc` are ordered according to their character codes. With this option, on the other hand, `nbc` will write them in the order of appearances in '*Base*_`freq.txt`'.

`-v` (no argument)

With this option, the intermediate computed results are recorded into '*Base*_`log.txt`'. It should be noted that the resultant log file may become quite large.

`-h` (no argument)

This option displays a short description on the optional flags.

4.4 VB based clustering

Generally speaking, the usage of `vnbc` is quite similar to that of `nbc`. The names of input/output files are shown in Table 3. Also the file format for `vnbc` is the same as that of `nbc` except the messages in the header parts (see §4.3.2). The distinguished difference lies in the command line options for the EM algorithm. First, the `-I` and the `-J` flags are not available in `vnbc` (to be more specific, only the method '`noisy_u`' is available). Also, the meanings of the following optional flags are different as those of `nbc` (each symbol ϵ with some subscript denotes a small random noise):

Table 3: Input/Output files for `vnbc`.

	Filename	Content
Input	*Base*_`freq.txt`	Occurrences of objects
Output	*Base*_`smooth.txt`	Pseudo counts δ
Output	*Base*_`dissim_vb.txt`	Dissimilarities
Output	*Base*_`memp_vb.txt`	Membership probs.
Output	*Base*_`cluster_vb.txt`	Clustering results
Output	*Base*_`rank_vb.txt`	Result of relevance analysis
Output	*Base*_`log_vb.txt`	Logs
Output	*Base*_`msg_vb.txt`	Messages on the display
Output	*Base*_`smem_vb.txt`	Logs on the SMEM algorithm

`-c` *ClassPseudoCount*

This option specifies the initial hyper-parameters for the class variable. That is, for each class k, the initial hyper-parameters $\alpha_k^{(0)}$ in Step 1 of Fig. 7 are almost equally set to (1 + *ClassPseudoCount*) + ϵ_k. The default value is 0 (i.e. uninformative).

`-w` *AttrPseudoCount*

This option specifies the initial hyper-parameters for the attribute variables. If *AttrPseudoCount* takes the form '$\zeta_1, \zeta_2, \cdots, \zeta_J$', for each attribute j, the initial hyper-parameters $\alpha_{j,k,x_j}^{(0)}$ are set to $(1 + \zeta_j) + \epsilon_j$. If *AttrPseudoCount* is just a non-negative number ζ, all pseudo counts $\alpha_{j,k,x_j}^{(0)}$ for attributes are set to $(1 + \zeta) + \epsilon_{j,k,x_j}$. The default values are all 0.

`-V` *MinPseudoCount*

With this option, `vnbc` runs the VB-EM algorithm with the initial hyper-parameters used in the BDeu metric (§3.1.2), where (1 + *MinPseudoCount*) is the minimum among them. That is, `nbc` first computes $\alpha = KV_{\max} \cdot (1 + \mathit{MinPseudoCount})$, where $V_{\max} = \max_{1 \le j \le J} V_j$, and then sets pseudo counts $\alpha_k^{(0)} = \alpha/K + \epsilon_k$ and $\alpha_{j,k,x_j}^{(0)} = \alpha/KV_j + \epsilon_{j,k,x_j}$. This option is prioritized over the `-c` and the `-w` options.

4.5 Auxiliary tools

NBCT provides an auxiliary tool named `nbcsep` for post-processing of the output files from the executables

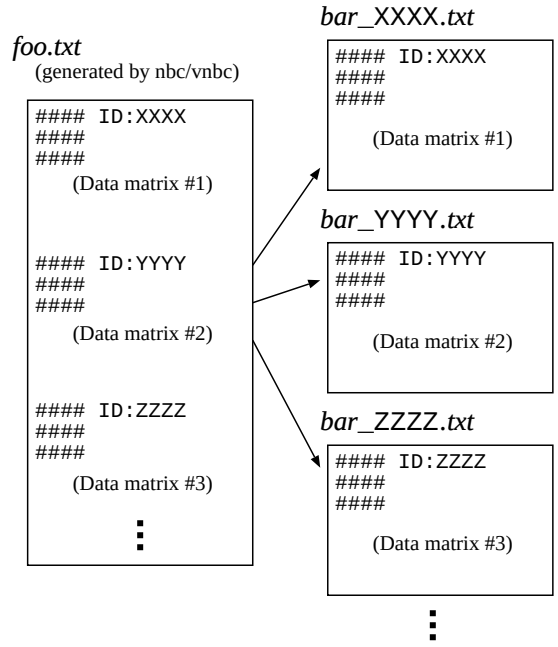

Figure 9: Output file split by `nbcsep` with an argument '`-f bar`'.

`nbc` and `vnbc`. As described above, an output file contains data matrices each of which has a header part with a predefined identifier. In a typical case, we use `nbcsep` to split the output file into the files whose names have the corresponding identifiers (Fig. 9). The synopsis of `nbcsep` is as follows:

`nbcsep` *Options InputFile*

If '`-`' is given to the argument *InputFile*, `nbcsep` assumes that the input is passed from the standard input. If no options are given, `nbcsep` displays all identifiers of the data matrices in *InputFile*. Here is the list of available options:

`-f` *Base*

`nbcsep` writes each data matrix in *InputFile* into the file named '*Base_ID.Ext*', where *ID* is the identifier of the data matrix, and *Ext* is the file extension of *InputFile*. The default value of *Base* is `out`.

`-s` *Suffix*

`nbcsep` adds *Suffix* to the name of the output file of `nbcsep`. The resultant filename will be '*Base_Suffix_ID*`.`*Ext*'.

`-e` *Ext*

`nbcsep` uses *Ext* as a new file extension for the output files.

`-v` *ID*

`nbcsep` outputs only the data matrices whose identifiers match with *ID*, ignoring the case of letters, to the standard output. The matching criterion can be specified by the `-m` flag.

`-o` *ID*

This is the same as '`-v` *ID*' except that the output will be saved into the file(s) whose names have a prefix specified with the `-f` flag.

`-m` *Criterion*

This option specifies the matching criterion as *Criterion* for the `-v` and the `-o` options. *Criterion* is one from `exact`, `infix`, `prefix` and `suffix`. The default value is `infix`.

`-p` *Charcode* (Available only with GNU libiconv)

`nbcsep` assumes the character codeset of the input file is *Charcode*.

`-q` *Charcode* (Available only with GNU libiconv)

`nbcsep` outputs the files with the character codeset *Charcode*.

`-h` (no argument)

`nbcsep` displays a simple help message.

4.6 Notes on computing environments

4.6.1 Computation time and space

The computation time and space required in the EM learning algorithm is almost linear in each of the number of classes, the number of attributes, and the number of distinct objects in the dataset. So, for the domain that contains many classes or attributes, the required resources will become quite large.

Especially, the amount of memory consumption could cause a severe problem. So for a remedy, we may reduce the required memory size by specifying the `-M` flag to the `nbc` or the `vnbc` commands. Unfortunately, however,

the use of the −M flag may slow down the EM algorithm.[7] Besides, for an implementational reason, the −M flag is not supported in the SMEM algorithm.

4.6.2 Settings for the EM algorithms

For some applications, the default value of the threshold for judging convergence of the EM algorithm, which is specified by the −e flag (§4.3.3), might be too small. In the early research of statistical natural language processing, it is reported that, at least in ML estimation, a large number of EM iterations do not necessarily lead to a good performance in probabilistic inferences after EM learning (e.g. Section 10.3.2 of [12] discusses the case of hidden Markov models). So there should be cases where it is reasonable to give a larger value to the −e flag, or set the maximum number of iterations with the −m flag (§4.3.3). In addition, it is empirically observed that the EM algorithm run with larger pseudo counts tends to converge with a smaller number of iterations.

4.6.3 Precision of floating-point numbers

For large-scale datasets, NBCT might show an unexpected behavior due to the problems with arithmetic precision such as underflow or precision errors. To avoid such problems, we may use the `long double` type for floating-point numbers, by adding the '`--enable-long-double`' flag to the `configure` script (Linux, Mac OS X, and Win32 with Cygwin/MinGW), or by deleting the '`-DNO_LDOUBLE`' flag from the `CFLAGS` variable in `Makefile.msvc` (Win32 with MSVC++). As is well known, the physical size of `long double` is system-dependent, and is only guaranteed not to be smaller than that of `double` (i.e. `long double` is only guaranteed not to be less precise than `double`). If you are using `long double`, the physical size will be recorded into the log file (*Base*`_log.txt`, etc).

4.6.4 Parallelization via OpenMP

Nowadays a couple of popular C compilers support OpenMP for shared-memory parallel computing. In NBCT's source code, several 'pragma' declarations for OpenMP are added to the 'for' loops which require massive numerical computation. An interested user may enable these 'pragma' declarations by giving compiler-dependent flags to the C compiler. For GNU C compiler (version 4.2 or later), for instance, we may add `CFLAGS='-fopenmp -O3'` and `LDFLAGS='-lgomp'` to the arguments of the `configure` script, and specify the number of threads at runtime by the `OMP_NUM_THREADS` environment variable.

Contact information

NBCT is still under development, and bug reports, questions, suggestions, or any other feedbacks are highly welcome. To make a contact with the author, please send an e-mail to `nbct[AT]mi.cs.titech.ac.jp`, with `[AT]` being replaced with `@`.

Acknowledgments

First of all, the author would like to thank Masanori Nakagawa, Taisuke Sato and Asuka Terai for providing the opportunity to develop this software, and for their valuable suggestions and feedbacks. Special thanks go to Kenichi Kurihara for the helpful advice on variational Bayesian learning, and to Yusuke Izumi for offering C code of the digamma and the log-gamma functions, which are originally implemented in SPECFUN,[8] and for the useful information on the development tools on MS Windows. This software gratefully uses many free software packages including GNU libiconv and Free Getopt (`http://freegetopt.sourceforge.net/`). The development of this software is supported by the 21st Century COE Program "Framework for Systematization and Application of Large-scale Knowledge Resources" at Tokyo Institute of Technology.

References

[1] P. Baldi, P. Fransconi, and P. Smyth. *Modeling the Internet and the Web*. John Wiley & Sons, 2003.

[7] This does not necessarily mean that there is inefficiency at the algorithmic level. It is probable that the cache mechanism of the computer does not work effectively with the −M flag.

[8] SPECFUN was developed by W. J. Cody et al. at Argonne National Laboratory, and is available in public domain at `http://www.netlib.org/specfun/`.

[2] M. J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, University College London, 2003.

[3] W. Buntine. Theory refinement on Bayesian networks. In *Proc. of the 7th Conf. on Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

[4] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. Fayyad, G. Piatesky, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. The MIT Press, 1995.

[5] D. Chickering and D. Heckerman. Efficient approximation for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning*, 29:181–212, 1997.

[6] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[7] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 118:69–113, 2000.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B39:1–38, 1977.

[9] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[10] A. Hotho, S. Staab, and G. Stumme. Explaining text clustering results using semantic structures. In *Proc. of the 7th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD-2003)*, 2003.

[11] K. Kurihara and T. Sato. Variational Bayesian grammar induction for natural language. In *Proc. of the 8th Intl. Colloquium on Grammatical Inference (ICGI-2006)*, pages 84–95, 2006.

[12] C. Manning and H. Shütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[13] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Proc. of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190, 1993.

[14] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[15] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[16] A. Terai. From computation to mind: examining the psychological validity of a computational model of metaphor understanding — targeting more human-like systems. In *Proc. of Symposium on Large-scale Knowledge Resources (LKR-2005)*, 2005.

[17] N. Ueda and Z. Ghahramani. Bayesian model search for mixture models based on optimizing variational bounds. *Neural Networks*, 15:1223–1241, 2002.

[18] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. SMEM algorithm for mixture models. In *Neural Information Processing Systems 11 (NIPS11)*, pages 599–605, 1999.