NBCTK: Naive Bayes Clustering Toolkit

(version 0.7)

Yoshitaka Kameya
Tokyo Institute of Technology

May 2, 2011

Contents

1 Introduction

NBCTK (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* [2]. NBCTK receives a dataset in a tabular format with discrete (nominal) and continuous (numeric) values, and performs the following tasks:

- Clustering via EM learning
  - Avoiding undesired local maxima by:
    - Random restarts
    - Deterministic annealing EM algorithm
    - Split-merge EM algorithm (experimental)
  - Determining the number of clusters
- Others:

– Relevance analysis

– Evaluation of obtained clusters

NBCTK is designed to be generic to handle a variety of datasets including bag-of-words representation (with TF-IDF weighting) of documents [2].

Although NBCTK was much influenced by *AutoClass* [4, 5], a popular probabilistic clustering tool in the data mining field, it has a couple of advantages over AutoClass. First, NBCTK runs in three statistical frameworks — ML (*maximum likelihood*), MAP (*maximum a posteriori*) and VB (*variational Bayes*) [1, 3]. It is empirically found in recent researches that the VB approach often shows a better performance than that by ML/MAP approaches in *model selection* (e.g. determining the optimal number of clusters) [3, 15], though ML/MAP is simpler and would be easier to work with. The second advantage is that NBCTK is augmented with two EM algorithms aiming to escape from undesired local maxima, called the *deterministic annealing EM* (DAEM) algorithm [24] and the *split-merge EM* (SMEM) algorithm [25]. Lastly, NBCTK provides a flexible way of various configurations and fully (i.e. even for multivariate normal distributions) deals with missing values.

The rest of this document is comprised of three parts. First, in Section 2, we will see a typical usage of NBCTK with an exemplar dataset included in the released package. Section 3 then gives a (rough) description of the algorithms used in NBCTK. Section 4 is the third part which describes the detailed usage of NBCTK. Lastly Section 5 makes some remarks on effective use of NBCTK. We annotated '$*$' to the title of the sections that relate to the advanced functions of NBCTK, so the readers can skip these sections unless necessary.

2 Getting started

Let us use NBCTK with an artificial dataset included in the released package. We suppose here that the installation of NBCTK (see §4.2 for the procedure) has been successfully done. Also consider that we are working in the `example` directory of the unfolded package. In this section, we just focus on the look and feel of NBCTK, and do not aim at listing all functions. We first describe the task of clustering.

2.1 What is clustering?

Given a dataset D of *objects*,[1] clustering [13] is a task to group similar objects in D into sets called *clusters*. For example, one may wish to obtain some groups of Web pages that seem to have the same topic, or one may group the patients who show similar medical states. Like association rule mining, clustering requires no (or less) human annotations, and hence can be utilized as a first step in knowledge discovery.

[1] For a historical reason, in the terminal messages from the programs in NBCTK, the term '*case*' is used instead of 'object.'

Table 1: The descriptions of attributes in `ex1_freq.csv`.

Index	Type	Possible values	Formally denoted by
0	ID	`P0`, `P1`, ..., `P999`	—
1	Answer class	`0`, `1`, `2`, `3`	—
2	Discrete	`a`, `b`, `o`, `ab`	A_1
3	Continuous	(floating-point number)	A_2
4	Ignored	—	—
5	Discrete	`male`, `female`	A_3
6	Continuous	(floating-point number)	A_4
7	Continuous	(floating-point number)	A_5
8	Discrete	`high`, `mid`, `low`	A_6
9	Ignored	—	—

Naive Bayes clustering can be seen as unsupervised classification of objects, based on a naive Bayes model. Hence we may use the term '*classes*' instead of clusters. Each object is classified into one of K anonymous classes. We also consider each object to be represented by a vector of values of one or more predefined *attributes*.

2.2 Target data

The dataset D is supposed to be in tabular form. To be concrete, let us open `ex1_freq.csv`, an artificial dataset, in the directory by some text viewer:

```
P0,0,b,4.74,6.48,female,0.36,1.51,mid,x0
P1,0,a,4.06,1.36,female,-0.38,0.99,mid,x0
P2,0,a,5.04,4.38,male,1.59,1.01,mid,x0
  :
```

Each of 1,000 lines 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. Each object has 10 attributes, which are described in Table 1. The attribute 0 (the indices of attributes are zero-based) indicates the ID of the corresponding object. The attribute 1 indicates the *answer class* to which the object is known to belong. Actually, in this dataset, all objects were sampled from a naive Bayes model (formally described in §2.3) with 4 classes, and the answer classes were determined in the process of sampling. The attributes 4 and 9 are noisy ones, and should be ignored. Only the rest of the attributes are used for clustering, and hereafter called *descriptive attributes*. In a formal context, we only consider these descriptive attributes, and T denotes the number of descriptive attributes. For instance, the first object in `ex1_freq.csv` is written as $\boldsymbol{x}$ = (`b`, `4.74`, `female`, `0.36`, `1.51`, `mid`), and $T = 6$.

On the other hand, there can be a dataset where several objects have the same attribute values. In NBCTK, unless having continuous attributes or the ID attribute, these objects will be regarded as identical since there is no way to distinguish them. The dataset is then a multiset of objects. In the dataset `ex2_`

`freq.txt`, objects who have the same vector $\boldsymbol{x}$ of attribute values were suppressed, and instead their count $N(\boldsymbol{x})$ is added to the right-most column, which is called the *count column*:

```
v1    w1    x1    y1    z1    16
v1    w1    x1    y1    z2    21
v1    w1    x1    y1    z3    8
                :
```

For example, we have $N(\boldsymbol{x}) = 21$ for $\boldsymbol{x} = (\texttt{v1}, \texttt{w1}, \texttt{x1}, \texttt{y1}, \texttt{z2})$. Please note here that the count column is not considered as an attribute. For the dataset where the count column is omitted, like `ex1_freq.csv`, $N(\boldsymbol{x})$ is considered to be one for every object $\boldsymbol{x}$. Also note that NBCTK accepts the tab character as the delimiter of attribute values for the input files whose file extension is '`txt`'.

2.3 Naive Bayes models

2.3.1 Overall structure

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, we consider that the objects in D were generated in a causal way depicted as a Bayesian network (Fig. 1), which has random variables C and A_i ($1 \leq i \leq T$). C is called the *class variable*, and A_i are called the *attribute variables*. We then write the vector of A_1, A_2, ..., A_T as $\boldsymbol{X}$. In the case of `ex1_freq.csv`, the descriptive attributes are referred to by A_i in the rightmost column of Table 1.

In a naive Bayes model, the class of each object is firstly determined as k under the class distribution $p(C=k)$, and then attribute values $\boldsymbol{x}$ are conditionally determined under the attribute distribution $p(\boldsymbol{X}=\boldsymbol{x} \mid C=k)$.

2.3.2 Attribute distribution (in general)

For the attribute distribution $p(\boldsymbol{X}=\boldsymbol{x} \mid C=k)$, NBCTK assumes that each attribute A_i is of three types:

- Discrete (and single-dimensional) — A_i follows a categorical distribution.[2]
- Continuous and single-dimensional — A_i follows a univariate normal distribution.
- Continuous and multidimensional — some attribute variables including A_i jointly follow a multivariate normal distribution.

AutoClass also introduces these types. In a usual definition of naive Bayes models, on the other hand, all attributes are considered to be single-dimensional. In this sense, such usual naive Bayes models are a special case of the naive Bayes models considered in NBCTK (and in AutoClass). Also one may find that Gaussian mixture models are a special case.

[2] The categorical distribution is defined as a generalization of the Bernoulli distribution to more than 2 categories.

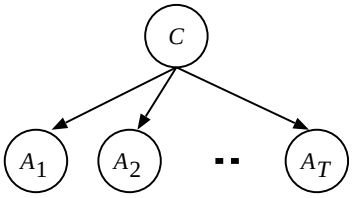

Figure 1: Bayesian network representation of a naive Bayes model, where all attributes are single-dimensional.

2.3.3 Attribute distribution (in detail)

We let $\mathcal{X}_{\text{disc}}$ be a set of discrete attributes, $\mathcal{X}_{\text{cont-u}}$ be a set of continuous and single-dimensional attributes, and $\mathcal{X}_{\text{cont-m}}$ be a set of vectors each of which is comprised of the continuous attributes jointly following a certain multivariate normal distribution. We define $\mathcal{X} \stackrel{\text{def}}{=} \mathcal{X}_{\text{disc}} \cup \mathcal{X}_{\text{cont-u}} \cup \mathcal{X}_{\text{cont-m}}$ and let $\mathcal{X} = \{X_1, \ldots, X_j, \ldots, X_J\}$. Note that $J \leq T$ obviously holds, and that $\boldsymbol{X}_j \in \mathcal{X}$ is a vector $(A_{i_1}, A_{i_2}, \ldots, A_{i_n})$ of attributes. In the case of `ex1_freq.csv`, we may have $J = 4$, $\boldsymbol{X}_1 = (A_1)$, $\boldsymbol{X}_2 = (A_2, A_4, A_5)$, $\boldsymbol{X}_3 = (A_3)$ and $\boldsymbol{X}_4 = (A_6)$, where $\boldsymbol{X}_1, \boldsymbol{X}_3, \boldsymbol{X}_4 \in \mathcal{X}_{\text{disc}}$ and $\boldsymbol{X}_2 \in \mathcal{X}_{\text{cont-m}}$ (see also Table 1). Also it is possible to have $J = 5$, $\boldsymbol{X}_1 = (A_1)$, $\boldsymbol{X}_2 = (A_2, A_5)$, $\boldsymbol{X}_3 = (A_3)$, $\boldsymbol{X}_4 = (A_4)$ and $\boldsymbol{X}_5 = (A_6)$, where $\boldsymbol{X}_1, \boldsymbol{X}_3, \boldsymbol{X}_5 \in \mathcal{X}_{\text{disc}}$, $\boldsymbol{X}_2 \in \mathcal{X}_{\text{cont-m}}$ and $\boldsymbol{X}_4 \in \mathcal{X}_{\text{cont-u}}$.

For simplicity, in the context where we know $\boldsymbol{X}_j$ (often) contains only a single attribute, we write $\boldsymbol{X}_j$ and its value with non-bold letters (i.e. X_j and x_j, respectively). Furthermore, we refer to all nominal values by integers. That is, for an attribute X_j that can have V_j values, we number them from 1 to V_j. For instance, for A_1 in Table 1, the values `a`, `b`, `o` and `ab` are referred to by 1, 2, 3 and 4, respectively.

Now, the *joint probability*, i.e. the probability that an object with attribute values $\boldsymbol{x} = (a_1, a_2, \ldots, a_T)$ belongs to the class k, is simply computed as:

$$\begin{aligned} & p(C=k, A_1=a_1, A_2=a_2, \ldots, A_T=a_T) \\ &= p(C=k) \prod_{j=1}^{J} p(\boldsymbol{X}_j=\boldsymbol{x}_j \mid C=k). \end{aligned} \tag{1}$$

According to the type of $\boldsymbol{X}_j$, $p(\boldsymbol{X}_j=\boldsymbol{x}_j \mid C=k)$ is predefined as follows:

- When $X_j \in \mathcal{X}_{\text{disc}}$, $p(X_j \mid C=k)$ follows a categorical distribution. We consider $\theta_{j,k,x_j} \stackrel{\text{def}}{=} p(X_j = x_j \mid C = k)$ as a parameter of the distribution. V_j stands for the number of its possible values, and we have $1 \leq x_j \leq V_j$.
- When $X_j \in \mathcal{X}_{\text{cont-u}}$, $p(X_j \mid C = k)$ follows a univariate normal distribution. That is, $p(X_j=x_j \mid C=k)$ is:

$$\mathcal{N}(x_j \mid \mu_{j,k}, \sigma^2_{j,k}) = \frac{1}{\sqrt{2\pi\sigma^2_{j,k}}} \exp\left(-\frac{(x_j-\mu_{j,k})^2}{2\sigma^2_{j,k}}\right), \tag{2}$$

where $\mu_{j,k}$ and $\sigma^2_{j,k}$ are the mean and the variance, respectively.

- When $\boldsymbol{X}_j \in \mathcal{X}_{\text{cont-m}}$, $p(\boldsymbol{X}_j \mid C = k)$ follows a multivariate normal distribution. That is, $p(\boldsymbol{X}_j = \boldsymbol{x}_j \mid C = k)$ is:

$$\mathcal{N}(\boldsymbol{x}_j \mid \boldsymbol{\mu}_{j,k}, \boldsymbol{\Sigma}_{j,k}) = \frac{1}{\sqrt{(2\pi)^{n_j}|\boldsymbol{\Sigma}_{j,k}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_j - \boldsymbol{\mu}_{j,k})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}_{j,k}(\boldsymbol{x}_j - \boldsymbol{\mu}_{j,k})\right), \tag{3}$$

  where n_j is the dimension of $\boldsymbol{X}_j$, and $\boldsymbol{\mu}_{j,k}$ and $\boldsymbol{\Sigma}_{j,k}$ are the mean vector and the covariance matrix, respectively.

The class variable C, on the other hand, is discrete and follows a categorical distribution, whose parameters are $\theta_k \stackrel{\text{def}}{=} p(C = k)$ $(1 \le k \le K)$. We use $\boldsymbol{\theta}$ for a vector of all parameters above. Eq. 1 is then simplified and written in a parameterized form as:

$$p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) = p(k \mid \boldsymbol{\theta}) \prod_{j=1}^{J} p(\boldsymbol{x}_j \mid k, \boldsymbol{\theta}), \tag{4}$$

where $p(C = k, \ldots)$ and $p(\boldsymbol{X}_j = \boldsymbol{x}_j, \ldots)$ are abbreviated as $p(k, \ldots)$ and $p(\boldsymbol{x}_j, \ldots)$, respectively.

2.4 Running NBCTK

2.4.1 Basic usage

Now try NBCTK with the dataset contained in `ex1_freq.csv`. Let us assume here that the true number of classes is known as 4. Then, to get the clusters of objects in `ex1_freq.csv`, we invoke an executable named `nbc` with several option flags:

```
nbc -f ex1 -k 4 -x 10 -I 0 -E 1 -y 4,9 -u 3,6,7
```

The option '`-f ex1`' specifies the names of input/output files including `ex1_freq.csv`. Besides, the options '`-k 4`' and '`-x 10`' indicate that the number of clusters is 4 ($K = 4$), and that the number of attributes is 10, respectively. The rest of option flags specify the attribute types. The options '`-I 0`', '`-E 1`' and '`-y 4,9`' respectively say that the attribute 0 is the ID attribute, the attribute 1 is the answer class attribute, and the attributes 4 and 9 should be ignored (see also Table 1). The option '`-u 3,6,7`' says the attributes 3, 6 and 7 jointly follow a multivariate normal distribution. It should be noted that the attributes that are not explicitly specified are considered as discrete attributes. After running the program, we may see the terminal messages below:

```
Reading cases from ex1_freq.csv...done
#classes = 4:
  #iters 0 (Converged: 19 iters)
  L=-5747.688373
  BIC = -5923.836132
```

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, i.e. we made 19 EM iterations, the log-likelihood was converged to −5747.688373, and so on.

One may notice it possible to have a different setting for continuous attributes. For example, we may specify[3] that attribute 3 and 7 jointly follow a bivariate normal distribution and attribute 6 follows a univariate normal distribution:[4]

```
nbc -f ex1 -k 4 -x 10 -I 0 -E 1 -y 4,9 \
                                -u 3,7 -u 6
```

As mentioned before, the count column is not considered as an attribute, so for a dataset having the count column, the number of attributes, which is specified by `-x` option, should be the number of columns minus one. Specifically, for `ex2_freq.txt`, we may run:

```
nbc -f ex2 -k 4 -x 5
```

2.4.2 Output files

After the run with `ex1_freq.csv`, three files named `ex1_{param,cluster,eval,log}.csv` should have been created. The first file, `ex1_param.csv`, contains the parameters estimated by the EM algorithm:

```
#### ID:PARAM-CLASS
#### Class k, Class parameter P(k)
####
0,0.299999927172
1,0.200000072828
2,0.400000024986
3,0.0999999750144
#### ID:PARAM-CAT2
#### Attr. ID, Class k, Attr. value x2, Attr. par
ameter P(x2|k)
####
2,0,a,0.0933333559908
2,0,ab,0.196666862475
  :
#### ID:PARAM-GAUSS-MEAN0
#### Gauss ID, Class k, Attr. ID, Mean
####
0,0,3,0.999999808665
0,0,6,5.00016666618
  :
```

We see from above that the class parameters were estimated as 0.299, 0.200, ..., and so on. In general, each output file contains several data matrices, each of which has header lines beginning with '#'.

The second file, `ex1_cluster.csv`, contains the data matrices on a mapping from objects to clusters:

[3] '\' means a continuation of the command line (as used in popular shell programs in UNIX/Linux), and is used here just for space limitation.

[4] The question of which is better is a problem of *model selection*, and as will be mentioned later, we may decide it by a Bayesian score provided by NBCTK.

```
#### ID:CLUSTER-CASE-BY-CASE
#### Case ID, Answer class, Cluster k, Attribute
values x, Membership prob. P(k|x)
####
P0,0,3,b,4.74,female,0.36,1.51,mid,1
P1,0,3,a,4.06,female,-0.38,0.99,mid,1
P2,0,3,a,5.04,male,1.59,1.01,mid,0.999999999999
                :
```

We see that the object named `P0`, $\boldsymbol{x}$ = (`b`, `4.74`, `female`, `0.36`, `1.51`, `mid`), belongs to the cluster 3, and so on. One may also notice that the rows in the data matrix named 'CLUSTER-CASE-BY-CASE' are ordered by the indices of objects. On the other hand, in the data matrix 'CLUSTER-CASE-BY-CLUSTER', the rows are ordered by the indices of clusters:

```
#### ID:CLUSTER-CASE-BY-CLUSTER
#### Cluster k, Answer class, Case ID, Attribute
values x, Membership prob. P(k|x)
####
0,2,P300,a,-0.2,female,5.25,5.08,low,1
0,2,P301,ab,0.58,male,5.03,5.9,low,1
0,2,P302,a,1.74,male,4.12,4.45,low,1
                :
```

The above tell us that the cluster 0 includes the objects `P300`, `P301`, and so on.

In recent versions, a file containing the results of cluster evaluation, like the third file, `ex1_eval.csv`, will be generated when objects have the answer class attribute, which has been specified by `-E` option. From the contents, we can see that purity (see §3.3) is 0.9, and Rand index (§3.3) is about 0.96:

```
#### ID:EVAL-PURITY
#### Evaluation measure, Value
#### -- Weight in F-measure of Purity = 1
####
Purity,0.9
Inverse Purity,1
F-Purity,0.947368421053
#### ID:EVAL-RAND
#### Evaluation measure, Value
#### -- Weight in F-measure of Wallance's measu
res = 1
####
Rand index,0.95995995996
Wallance's measure I,0.882005899705
  :
#### ID:CLUSTER-MATCH-BY-PREDICTED
#### Cluster k [predicted], Answer class, Num.
of matched cases
####
0,0,100
0,1,200
0,2,0
0,3,0
1,0,0
1,1,0
1,2,0
1,3,400
2,0,0
2,1,0
2,2,0
2,3,0
3,0,0
3,1,0
3,2,300
3,3,0
  :
```

Also from `CLUSTER-MATCH-BY-PREDICTED`, we can find that the cluster 0 made (predicted) by `nbc` contains the object that originally belong to the answer classes 0 and 1 (i.e. the cluster 0 is too large).

Lastly, the fourth file, `ex1_log.csv`, contains additional information on the (last) execution. It is desirable for users to look into the log file to check if the execution has been done as intended.

2.4.3 Various execution options

NBCTK provides many options for the EM algorithm. For example, since the EM algorithm is a hill-climbing algorithm, and is known to be often trapped in undesirable local maxima, we sometimes wish to restart the EM algorithm with several different initial settings. In NBCTK, this method is enabled by giving the number of restarts to `-n` option:[5]

```
% nbc -f ex1 -k 4 -x 10 -I 0 -E 1 -y 4,9 \
                                    -u 3,6,7 -n 10
Reading cases from ex1_freq.csv...done
#classes = 4:
 [0] #iters 0 (Converged: 18 iters)   L=-5747.688373
 [1] #iters 0 (Converged: 16 iters)   L=-5747.688373
 [2] #iters 0 (Converged: 14 iters)   L=-5747.688373
 [3] #iters 0. (Converged: 23 iters)  L=-5747.688373
 [4] #iters 0. (Converged: 22 iters)  L=-5747.688373
 [5] #iters 0 (Converged: 16 iters)   L=-5747.688373
 [6] #iters 0. (Converged: 24 iters)  L=-5747.688373
 [7] #iters 0. (Converged: 24 iters)  L=-5747.688373
 [8] #iters 0 (Converged: 15 iters)   L=-5747.688373
 [9] #iters 0 (Converged: 14 iters)   L=-5747.688373
 <<Resumed best parameter set #5>>
 [5] #iters  (Converged: 17 iters)    L=-5747.688373
 BIC = -5923.836132
```

We can find from above the sixth initial setting provides the best estimate of parameters (though the differences were subtle).

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* or *hyperparameters* to the program. In NBCTK, it is possible to specify pseudo counts or hyperparameters in a flexible form, and we can make similar settings to AutoClass with `-A` option as well:

```
% nbc -f ex1 -k 4 -x 10 -I 0 -E 1 -y 4,9 -u 3,6,7 -A

Reading cases from ex1_freq.csv...done
#classes = 4:
  #iters 0. (Converged: 31 iters)     L=-5907.149381
  Cheeseman-Stutz score = -6024.130102
```

[5] '%' is the prompt symbol.

```
% vnbc -f ex1 -k 2:10:1 -x 10 -I 0 -E 1 -y 4,9 -u 3,6,7
Reading cases from ex1_freq.csv...done
|Classes| = 2:
  #iters 0 (Converged: 9 iters) F=-7485.127335
  Variational Free Energy = -7485.127335 (temporarily optimal)
          :
|Classes| = 4:
  #iters 0 (Converged: 16 iters)        F=-6014.731751
  Variational Free Energy = -6014.731751 (temporarily optimal)
|Classes| = 5:
  #iters 0. (Converged: 30 iters)       F=-6017.392785
  Variational Free Energy = -6017.392785
          :
|Classes| = 10:
  #iters 0. (Converged: 28 iters)       F=-6027.032856
  Variational Free Energy = -6027.032856
Optimal |Classes| = 4
```

Figure 2: Terminal messages from `vnbc` in the 'model selection' mode.

We have assumed so far that the true number K of clusters is known in advance, but in practical situations, it is often unknown. Consequently finding the optimal number of clusters (based on the dataset) is a key issue in clustering. Again, this can be seen a kind of model selection problem. NBCTK 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 a variational Bayesian (VB) approach. In VB, we use another executable named `vnbc`. The terminal messages from `vnbc` are shown in Fig. 2, where the number of clusters to be examined ranges from 2 to 10. If we specify '`-k` *Kmin*`:`*Kmax*`:`*Kstep*', NBCTK will be switched into the 'model selection' mode. In this example, we fortunately recovered the true number of clusters as 4.

2.4.4 Auxiliary tool for post-processing

Lastly, we mention `nbcsep`, an auxiliary tool for post-processing the outputs from `nbc` and `vnbc`. As seen before, each output file contains several data matrices, each having an identifier. Typically, `nbcsep` extracts these data matrices, and puts each of them into an individual file. The name of a new file includes the identifier of the corresponding data matrix. For example, let us apply `nbcsep` to `ex1_param.csv`:

```
% nbcsep -f foo ex1_param.csv
Output: foo_PARAM-CLASS.csv
Output: foo_PARAM-CAT2.csv
Output: foo_PARAM-CAT5.csv
Output: foo_PARAM-CAT8.csv
Output: foo_PARAM-GAUSS-MEAN0.csv
Output: foo_PARAM-GAUSS-COVAR0.csv
```

Here, the common prefix of the names of newly created files was given by `-f` option.

3 Clustering algorithms

This section gives a detailed description on the clustering algorithms and the other related algorithms provided in NBCTK.

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 recall that we are given a set D of objects where $N(\boldsymbol{x})$ is the number of occurrences of objects that have the attribute value $\boldsymbol{x}$. 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:

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{\mathrm{ML}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(D \mid \boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log p(D \mid \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}} N(\boldsymbol{x}) \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}} N(\boldsymbol{x}) \log \sum_{k=1}^{K} p(k, \boldsymbol{x} \mid \boldsymbol{\theta}). \end{aligned} \tag{5}$$

Due to the lack of the information on the cluster (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* [9]. Fig. 3 is the EM algorithm derived for naive Bayes models.[6]

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

$$p(k \mid \boldsymbol{x}, \boldsymbol{\theta}) \propto p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) = p(k \mid \boldsymbol{\theta}) \prod_{j=1}^{J} p(x_j \mid k, \boldsymbol{\theta}) \tag{6}$$

$$E[k] := \sum_{\boldsymbol{x}} N(\boldsymbol{x}) p(k \mid \boldsymbol{x}, \boldsymbol{\theta}) \tag{7}$$

$$\theta_k \propto E[k] \tag{8}$$

$\underline{X_j \in \mathcal{X}_{\text{disc}}}$:

$$\theta_{j,k,x_j} \propto \sum_{\boldsymbol{x}' \in D : x'_j = x_j} N(\boldsymbol{x}') p(k \mid \boldsymbol{x}', \boldsymbol{\theta}) \tag{9}$$

$\underline{X_j \in \mathcal{X}_{\text{cont-u}}}$:

$$\mu_{j,k} := \frac{1}{E[k]} \sum_{\boldsymbol{x} \in D} p(x_j \mid k, \boldsymbol{\theta}) x_j \tag{10}$$

$$\sigma^2_{j,k} := \frac{1}{E[k]} \sum_{\boldsymbol{x} \in D} p(x_j \mid k, \boldsymbol{\theta})(x_j - \mu_{j,k})^2 \tag{11}$$

$\underline{\boldsymbol{X}_j \in \mathcal{X}_{\text{cont-m}}}$:

$$\boldsymbol{\mu}_{j,k} := \frac{1}{E[k]} \sum_{\boldsymbol{x} \in D} p(\boldsymbol{x}_j \mid k, \boldsymbol{\theta}) x_j \tag{12}$$

$$\boldsymbol{\Sigma}_{j,k} := \frac{1}{E[k]} \sum_{\boldsymbol{x} \in D} p(\boldsymbol{x}_j \mid k, \boldsymbol{\theta})(\boldsymbol{x}_j - \boldsymbol{\mu}_{j,k})(\boldsymbol{x}_j - \boldsymbol{\mu}_{j,k})^{\mathrm{T}} \tag{13}$$

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

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 NBCTK, we use *conjugate prior distributions* [8] for $p(\boldsymbol{\theta})$. To be more concrete:

- As a prior distribution of $\boldsymbol{\theta}_{\text{class}}$, (a vector of) the parameters θ_k $(1 \leq k \leq K)$ for the class variable C, we introduce the Dirichlet distribution:

$$p(\boldsymbol{\theta}_{\text{class}}) \stackrel{\text{def}}{=} \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \quad . \tag{14}$$

  Each α_k corresponds to θ_k, and is called a *hyperparameter*.

- As a prior distribution of $\boldsymbol{\theta}_{\text{disc}}$, the parameters θ_{j,k,x_j} for $X_j \in \mathcal{X}_{\text{disc}}$, we also introduce the Dirichlet distribution:

$$p(\boldsymbol{\theta}_{\text{disc}}) \stackrel{\text{def}}{=} \prod_{k=1}^{K} \prod_{j \in \mathcal{J}_{\text{d}}} \left(\frac{\Gamma\left(\sum_{x_j=1}^{V_j} \alpha_{j,k,x_j}\right)}{\prod_{x_j=1}^{V_j} \Gamma\left(\alpha_{j,k,x_j}\right)} \prod_{x_j=1}^{V_j} \theta_{j,k,x_j}^{\alpha_{j,k,x_j} - 1} \right) \tag{15}$$

  where $\mathcal{J}_{\text{d}}$ is a set of the indices of discrete attributes, i.e. $\mathcal{J}_{\text{d}} \stackrel{\text{def}}{=} \{j \mid X_j \in \mathcal{X}_{\text{disc}}\}$. Each α_{j,k,x_j} is the hyperparameter corresponding to θ_{j,k,x_j}.

- As a prior distribution of $\boldsymbol{\theta}_{\text{cont-u}}$, the parameters $\mu_{j,k}$ and $\sigma^2_{j,k}$ for $X_j \in \mathcal{X}_{\text{cont-u}}$, we introduce the normal-gamma distribution:

$$\begin{aligned} p(\boldsymbol{\theta}_{\text{cont-u}}) &\stackrel{\text{def}}{=} \prod_{k=1}^{K} \prod_{j \in \mathcal{J}_{\text{u}}} p(\mu_{j,k}, \lambda_{j,k}) \\ &= \prod_{k=1}^{K} \prod_{j \in \mathcal{J}_{\text{u}}} \mathcal{N}(\mu_{j,k} \mid \nu_{j,k}, (\tau \lambda_{j,k})^{-1}) \cdot \\ &\qquad \mathcal{G}(\lambda_{j,k} \mid a, b), \end{aligned} \tag{16}$$

$$\mathcal{G}(\lambda_{j,k} \mid a_{j,k}, b_{j,k}) \stackrel{\text{def}}{=} \frac{1}{\Gamma(a_{j,k})} {b_{j,k}}^{a_{j,k}} \lambda_{j,k}^{a_{j,k}-1} \exp(-b_{j,k}\lambda_{j,k}), \tag{17}$$

  where $\mathcal{J}_{\text{u}} \stackrel{\text{def}}{=} \{j \mid X_j \in \mathcal{X}_{\text{cont-u}}\}$, and $\lambda_{j,k} \stackrel{\text{def}}{=} \frac{1}{\sigma^2_{j,k}}$ is called the *precision* of the normal distribution. $\nu_{j,k}$, τ, a and b are the hyperparameters.

- As a prior distribution of $\boldsymbol{\theta}_{\text{cont-m}}$, the parameters $\boldsymbol{\mu}_{j,k}$ and $\boldsymbol{\Sigma}_{j,k}$ for $\boldsymbol{X}_j \in \mathcal{X}_{\text{cont-m}}$, we introduce the normal-Wishart distribution:

$$\begin{aligned} &p(\boldsymbol{\theta}_{\text{cont-m}}) \\ &\stackrel{\text{def}}{=} \prod_{k=1}^{K} \prod_{j \in \mathcal{J}_{\text{m}}} p(\boldsymbol{\mu}_{j,k}, \boldsymbol{\Lambda}_{j,k}) \cdot \\ &= \prod_{k=1}^{K} \prod_{j \in \mathcal{J}_{\text{m}}} \mathcal{N}(\boldsymbol{\mu}_{j,k} \mid \boldsymbol{\nu}_{j,k}, (\tau \boldsymbol{\Lambda}_{j,k})^{-1}) \cdot \\ &\qquad \mathcal{W}(\boldsymbol{\Lambda}_{j,k} \mid \phi_{j,k}, \boldsymbol{S}_{j,k}), \end{aligned} \tag{18}$$

$$\begin{aligned} &\mathcal{W}(\boldsymbol{\Lambda}_{j,k} \mid \phi_{j,k}, \boldsymbol{S}_{j,k}) \\ &\stackrel{\text{def}}{=} c(n_j, \phi_{j,k}) |\boldsymbol{S}_{j,k}|^{\frac{\phi_{j,k}}{2}} |\boldsymbol{\Lambda}_{j,k}|^{\frac{\phi_{j,k} - n_j - 1}{2}} \cdot \\ &\qquad \exp\left(-\frac{1}{2} \mathrm{tr}(\boldsymbol{S}_{j,k} \boldsymbol{\Lambda}_{j,k})\right), \end{aligned} \tag{19}$$

$$\begin{aligned} &c(n_j, \phi_{j,k}) \\ &\stackrel{\text{def}}{=} \left(2^{\frac{\phi_{j,k} n_j}{2}} \pi^{\frac{n_j(n_j-1)}{4}} \prod_{i=1}^{n_j} \Gamma\left(\frac{\phi_{j,k} + 1 - i}{2}\right) \right)^{-1}, \end{aligned} \tag{20}$$

  where n_j is the dimension of $\boldsymbol{X}_j$, $\mathcal{J}_{\text{m}} \stackrel{\text{def}}{=} \{j \mid X_j \in \mathcal{X}_{\text{cont-m}}\}$, and $\boldsymbol{\Lambda}_{j,k} \stackrel{\text{def}}{=} \boldsymbol{\Sigma}^{-1}_{j,k}$ is also the precision. $\nu_{j,k}$, τ, $\phi_{j,k}$ and $\boldsymbol{S}_{j,k}$ are the hyperparameters.

[6] In this document, the symbol '$\propto$' means a substitution with normalization.

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

$$p(k \mid \boldsymbol{x}, \boldsymbol{\theta}) \propto p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) = p(k \mid \boldsymbol{\theta}) \prod_{j=1}^{J} p(x_j \mid k, \boldsymbol{\theta}) \tag{21}$$

$$E[k] := \sum_{\boldsymbol{x}} N(\boldsymbol{x}) p(k \mid \boldsymbol{x}, \boldsymbol{\theta}) \tag{22}$$

$$\theta_k \propto E[k] + (\alpha_k - 1) \tag{23}$$

$\underline{X_j \in \mathcal{X}_{\text{disc}}}$:

$$\theta_{j,k,x_j} \propto \sum_{\boldsymbol{x}' \in D : x'_j = x_j} N(\boldsymbol{x}') p(k \mid \boldsymbol{x}', \boldsymbol{\theta}) + (\alpha_{j,k,x_j} - 1) \tag{24}$$

$\underline{X_j \in \mathcal{X}_{\text{cont-u}}}$:

$$\mu_{j,k} := \frac{\tau \nu_{j,k} + \sum_{x \in D} p(x_j \mid k, \boldsymbol{\theta}) x_j}{\tau + E[k]} \tag{25}$$

$$\sigma^2_{j,k} := \frac{\tau(\mu_{j,k} - \nu_{j,k})^2 + \sum_{x \in D} p(x_j \mid k, \boldsymbol{\theta})(x_j - \mu_{j,k})^2 + 2b_{j,k}}{2a_{j,k} - 1 + E[k]} \tag{26}$$

$\underline{\boldsymbol{X}_j \in \mathcal{X}_{\text{cont-m}}}$:

$$\boldsymbol{\mu}_{j,k} := \frac{\tau \boldsymbol{\nu}_{j,k} + \sum_{x \in D} p(\boldsymbol{x}_j \mid k, \boldsymbol{\theta}) \boldsymbol{x}_j}{\tau + E[k]} \tag{27}$$

$$\boldsymbol{\Sigma}_{j,k} := \frac{1}{\phi_{j,k} - n_j + E[k]} \cdot \Big(\sum_{x \in D} p(\boldsymbol{x}_j \mid k, \boldsymbol{\theta})(\boldsymbol{x}_j - \boldsymbol{\mu}_{j,k})(\boldsymbol{x}_j - \boldsymbol{\mu}_{j,k})^{\mathrm{T}} + \tau(\boldsymbol{\mu}_{j,k} - \boldsymbol{\nu}_{j,k})(\boldsymbol{\mu}_{j,k} - \boldsymbol{\nu}_{j,k})^{\mathrm{T}} + \boldsymbol{S}_{j,k} \Big) \tag{28}$$

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

Finally, the whole prior distribution is the product of the above distributions: $p(\boldsymbol{\theta}) = p(\boldsymbol{\theta}_{\text{class}}) p(\boldsymbol{\theta}_{\text{disc}}) p(\boldsymbol{\theta}_{\text{cont-u}}) p(\boldsymbol{\theta}_{\text{cont-m}})$.

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

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\text{MAP}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid D) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta} \mid D) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}) p(D \mid \boldsymbol{\theta})}{p(D)} \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \Big\{ \log p(\boldsymbol{\theta}) + \log p(D \mid \boldsymbol{\theta}) \Big\}. \end{aligned} \tag{29}$$

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) [10]. 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 $1/V_j$.

A similar discussion is possible for continuous attributes. To see this, let us introduce $\bar{x}_{j,k}$ defined as:

$$\bar{x}_{j,k} \stackrel{\text{def}}{=} \frac{1}{E[k]} \sum_{\boldsymbol{x} \in D} p(x_j \mid k, \boldsymbol{\theta}) x_j. \tag{30}$$

Then, we can rewrite Eqs. 10 and 25 respectively as:

$$\mu_{j,k} := \bar{x}_{j,k} \tag{31}$$

$$\mu_{j,k} := \frac{\tau \nu_{j,k} + E[k] \bar{x}_{j,k}}{\tau + E[k]}, \tag{32}$$

where the former (resp. the latter) is the ML (resp. MAP) estimate of the mean of X_j given the class k. It is easily seen that the latter is an weighted average of the former and $\nu_{j,k}$, where the weights are $E[k]$ and τ, respectively. Thus, τ works as a pseudo count for continuous attributes, and if we set τ and $\nu_{j,k}$ appropriately, the estimation procedure will show a robust behavior even for a small dataset. Also in Eq. 28 with an appropriate Σ_0, we can prevent the updated $\Sigma_{j,k}$ from being singular.

3.1.3 Additional notes on 'pseudo counts'

Since version 0.6, we use the term 'pseudo counts' in two ways. First, in the context of MAP estimation, this term refers to $\delta_k = \alpha_k - 1$ and $\delta_{j,k,x_j} = \alpha_{j,k,x_j} - 1$, where α_k and α_{j,k,x_j} are hyperparameters in Dirichlet distributions. Accordingly, the command for MAP estimation (i.e. `nbc`) is designed so that the users configure α_k and α_{j,k,x_j} through the corresponding pseudo counts δ_k and δ_{j,k,x_j}. To make MAP estimation meaningful, any of δ_k and δ_{j,k,x_j} should not be negative.

On the other hand, in variational Bayesian (VB) learning, hyperparameters α_k and α_{j,k,x_j} themselves can be seen as pseudo counts. Hence, in the command for VB learning (i.e. `vnbc`), we configure α_k and α_{j,k,x_j} directly. From the definition of Dirichlet distributions, α_k and α_{j,k,x_j} should be positive.

3.1.4 Sample statistics*

In NBCTK, the initial values of parameters or hyperparameters are often specified by the sample statistics, which are directly computed from the dataset D. To be more specific, for each discrete attribute $X_j \in \mathcal{X}_{\text{disc}}$, we have relative frequencies $\tilde{\theta}_{j,x_j}$ ($1 \leq x_j \leq V_j$):

$$\tilde{\theta}_{j,x_j} := \frac{1}{N} \sum_{\boldsymbol{x}' \in D : x'_j = x_j} N(\boldsymbol{x}'). \tag{33}$$

Note that $\tilde{\theta}_{j,x_j}$ does not depend on k, the identifier of a cluster, since the dataset D itself contains no class information.

For a continuous attribute $X_j \in \mathcal{X}_{\text{cont-u}}$, we have the sample mean $\tilde{\mu}_j$ and the sample covariance $\tilde{\sigma}_j^2$:

$$\tilde{\mu}_j = \frac{1}{N}\sum_{\boldsymbol{x}\in D} x_j \tag{34}$$

$$\tilde{\sigma}_j^2 = \frac{1}{N}\sum_{\boldsymbol{x}\in D}(x_j - \tilde{\mu}_j)^2, \tag{35}$$

and for continuous attributes $\boldsymbol{X}_j \in \mathcal{X}_{\text{cont-m}}$, we have the sample mean $\tilde{\boldsymbol{\mu}}_j$ and the sample covariance $\tilde{\boldsymbol{S}}_j$:

$$\tilde{\boldsymbol{\mu}}_j = \frac{1}{N}\sum_{\boldsymbol{x}\in D} \boldsymbol{x}_j \tag{36}$$

$$\tilde{\boldsymbol{S}}_j = \frac{1}{N}\sum_{\boldsymbol{x}\in D}(\boldsymbol{x}_j - \tilde{\boldsymbol{\mu}}_j)(\boldsymbol{x}_j - \tilde{\boldsymbol{\mu}}_j)^{\mathrm{T}}. \tag{37}$$

These sample statistics are sensitive to outliers, so NBCTK provides an option flag `--regular-stat` that ignores the values larger than the Q_1%-tile value or smaller than the Q_2%-tile value of the corresponding continuous attribute (only) when computing the sample statistic. Here Q_1 and Q_2 are specified by the `--extreme` flag (by default, $Q_1 = 95$ and $Q_2 = 5$).

3.1.5 Initialization of parameters*

In the first steps in Fig. 3 and Fig. 4, we randomly initialize the parameters. In NBCTK, for the class variable C, we initialize the parameters θ_k in two ways:

$$\theta_k \propto 1 + r \tag{38}$$

$$\theta_k \propto 1 + \varepsilon_1 r \tag{39}$$

where r is a real number randomly chosen from $[0, 1)$. We can switch these initialization methods by the `--init-c` flag (the default is the second one), and specify ε_1 by the `--noise` flag ($\varepsilon_1 = 0.1$ by default). Similarly, the parameters θ_{j,k,x_j} of each discrete attribute $X_j \in \mathcal{X}_{\text{disc}}$ conditioned on the k-th cluster are initialized in three ways:

$$\theta_{j,k,x_j} \propto 1 + r \tag{40}$$

$$\theta_{j,k,x_j} \propto 1 + \varepsilon_1 r \tag{41}$$

$$\theta_{j,k,x_j} \propto \tilde{\theta}_{j,x_j} + \varepsilon_1 \frac{r}{V_j} \tag{42}$$

where r is a real number randomly chosen from $[0, 1)$ and $\tilde{\theta}_{j,x_j}$ is the relative frequency of the objects that satisfy $X_j = x_j$ (§3.1.4). We can switch these initialization methods by the `--init-a` flag (the default is the second one).

For continuous attributes, on the other hand, we initialize the means of the attributes $\boldsymbol{X}_j$ as follows:

$$\boldsymbol{\mu}_{j,k} := \tilde{\boldsymbol{\mu}}_j + \varepsilon_2 \tilde{\boldsymbol{\sigma}}_j (r - 0.5) \tag{43}$$

where $\tilde{\boldsymbol{\mu}}_j$ is the sample means of $\boldsymbol{X}_j$, $\tilde{\boldsymbol{\sigma}}_j$ is the the diagonal elements of the sample covariance matrix $\tilde{\boldsymbol{S}}_j$, and r is a real number randomly chosen from $[0, 1)$. §3.1.4 gives a detailed description on sample statistics. ε_2 is specified by the `--noise-gauss` flag ($\varepsilon_2 = 0.1$ by default). On the other hand, NBCTK does not perform any perturbation when initializing $\boldsymbol{\Sigma}_{j,k}$.

3.1.6 Specifying hyperparameters*

It is obvious from Fig. 4 that the settings of hyperparameters often affect the result of MAP estimation. Practically, on the other hand, it is not straightforward to specify these hyperparameters. Sometimes we determine them by cross-validation using held-out datasets. In this section, we describe how the hyperparameters can be specified in NBCTK.

As described above, in MAP estimation, δ_k and δ_{j,k,x_j} (§3.1.2) work as pseudo counts for the class variable and discrete attributes, respectively, and τ works as a pseudo count for continuous attributes. We may give these pseudo counts manually, taking into account the size of the dataset. The default values are all zero, which make the estimation procedure equivalent to ML estimation.

We can also give the hyperparameters manually for continuous attributes. On the other hand, for a continuous attribute X_j that follows a univariate Gaussian, NBCTK sets the following hyperparameters by default:

$$\nu_{j,k} = \tilde{\mu}_j \tag{44}$$

$$a_{j,k} = \frac{1+\epsilon}{2} \tag{45}$$

$$b_{j,k} = \frac{1}{2}\tilde{\sigma}_j^2 \tag{46}$$

and for continuous attributes $\boldsymbol{X}_j$ that jointly follow a multivariate Gaussian, NBCTK sets the following hyperparameters by default:

$$\boldsymbol{\nu}_{j,k} = \tilde{\boldsymbol{\mu}}_j \tag{47}$$

$$\phi_{j,k} = n_j + \epsilon \tag{48}$$

$$\boldsymbol{S}_{j,k} = \tilde{\boldsymbol{S}}_j \tag{49}$$

where ϵ is a small random noise, $\tilde{\mu}_j$ and $\tilde{\boldsymbol{\mu}}_j$ are the sample means, and $\tilde{\sigma}_j^2$ and $\tilde{\boldsymbol{S}}_j$ are the sample (co)variances. For a detailed description on sample statistics, see §3.1.4.

3.1.7 Membership distribution

Given the estimated parameters $\hat{\boldsymbol{\theta}}$ ($\hat{\boldsymbol{\theta}}_{\text{ML}}$ or $\hat{\boldsymbol{\theta}}_{\text{MAP}}$) 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:

$$p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) = \frac{p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}})}{p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}})} \propto p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}}). \tag{50}$$

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, for an attribute (vector) X_j, we may consider the *attribute membership distribution* $p(k \mid x_j)$, which is computed as follows:

$$p(k \mid x_j, \hat{\boldsymbol{\theta}}) \propto p(k \mid \hat{\boldsymbol{\theta}}) p(x_j \mid k, \hat{\boldsymbol{\theta}}). \tag{51}$$

In inferences using attribute membership distributions (described in §3.1.8 and §3.1.9), NBCTK makes different treatments between discrete and continuous attributes. That is, for a *discrete* attribute X_j, $p(k \mid x_j, \hat{\boldsymbol{\theta}})$ is taken into account for each class k and each possible value $1 \leq x_j \leq V_j$. For each *continuous* attribute (vector) $X_{j'}$, on the other hand, we cannot enumerate all possible values, and thus $p(k \mid x_{j'}, \hat{\boldsymbol{\theta}})$ is taken into account for each class k and each value $x_{j'}$ appeared in the dataset D.

3.1.8 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 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:

$$k^* \;=\; \operatorname*{argmax}_{k:1\leq k\leq K} p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) \;=\; \operatorname*{argmax}_{k:1\leq k\leq K} p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}}). \tag{52}$$

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

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

$$k^* = \operatorname*{argmax}_{k:1\leq k\leq K} p(k \mid x_j, \hat{\boldsymbol{\theta}}) = \operatorname*{argmax}_{k:1\leq k\leq K} p(k, x_j \mid \hat{\boldsymbol{\theta}}) \tag{53}$$

3.1.9 Relevance analysis

Using the estimated parameters $\hat{\boldsymbol{\theta}}$, 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 $R(k, \boldsymbol{x}) \stackrel{\text{def}}{=} p(k \mid \boldsymbol{x})$, i.e. the membership probability [23]. To understand this way of ranking, let us transform the probability as follows:

$$\begin{aligned} R(k, \boldsymbol{x}) &= p(k \mid \boldsymbol{x}, \hat{\boldsymbol{\theta}}) \\ &= \frac{p(k \mid \hat{\boldsymbol{\theta}}) p(\boldsymbol{x} \mid k, \hat{\boldsymbol{\theta}})}{p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}})} \propto \frac{p(\boldsymbol{x} \mid k, \hat{\boldsymbol{\theta}})}{p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}})} \end{aligned} \tag{54}$$

(note that k is fixed here). Now $R(k, \boldsymbol{x})$ can be seen as a relevance score, because it indicates the significance of $p(\boldsymbol{x} \mid k, \hat{\boldsymbol{\theta}})$ compared to $p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}}) = \sum_{k=1}^{K} p(k \mid \hat{\boldsymbol{\theta}}) p(\boldsymbol{x} \mid k, \hat{\boldsymbol{\theta}})$, the unconditional (or averaged) probability of $\boldsymbol{x}$ being occurred. In some applications, some highly relevant objects would be a help for characterizing the cluster [12, 23]. One may find here that, for a fixed k, the relevance score $R(k, \boldsymbol{x})$ yields the same ranking (among different $\boldsymbol{x}$'s) as that via the *pointwise* (or *pairwise*) *mutual information* [6], which is defined as follows:

$$I(k, \boldsymbol{x}) \stackrel{\text{def}}{=} \log \frac{p(\boldsymbol{x}, k \mid \hat{\boldsymbol{\theta}})}{p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}}) p(k \mid \hat{\boldsymbol{\theta}})} = \log \frac{p(\boldsymbol{x} \mid k, \hat{\boldsymbol{\theta}})}{p(\boldsymbol{x} \mid \hat{\boldsymbol{\theta}})}. \tag{55}$$

One (subtle) practical advantage of $R(k, \boldsymbol{x})$ over $I(k, \boldsymbol{x})$ is that it is normalized (i.e. $0 \leq R(k, \boldsymbol{x}) \leq 1$ by definition). For example, we may collect $\{\boldsymbol{x} \mid R(k, \boldsymbol{x}) \geq \rho\}$ as relevant objects to the class k, where ρ is a threshold *common* to all classes ($0 < \rho < 1$).

Furthermore, we can consider the attribute-wise version of $R(k, \boldsymbol{x})$. That is, $R^j(k, x_j)$ indicates the relevance between the class k and the value x_j of the attribute (vector) X_j, and is defined as $R^j(k, x_j) = p(k \mid x_j, \hat{\boldsymbol{\theta}})$. This score measures 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 [20]). It should be addressed that AutoClass uses the attribute-wise version of $I(k, \boldsymbol{x})$ to measure the 'influence' of x_j to the class k, so essentially NBCTK yields the same rankings of attribute values as those by AutoClass.

3.1.10 Model selection

In clustering, we are often in question of how to determine the number of clusters. This is a problem of model selection, and in NBCTK, 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. Here, M corresponds to K, the number of classes (clusters). From the settings above, our goal is to find the most probable model M^* based on the data D at hand, that is, we aim to find M^* such that:

$$\begin{aligned} M^* &= \operatorname{argmax}_M \, p(M \mid D) \\ &= \operatorname{argmax}_M \, \frac{p(D \mid M) p(M)}{p(D)} \\ &= \operatorname{argmax}_M \, p(D \mid M), \end{aligned} \tag{56}$$

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 $p(\boldsymbol{\theta} \mid M)$:

$$\begin{aligned} p(D \mid M) &= \int_\Theta p(D, \boldsymbol{\theta} \mid M) d\boldsymbol{\theta} \\ &= \int_\Theta p(D \mid M, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid M) d\boldsymbol{\theta} \\ &= \left\langle p(D \mid M, \boldsymbol{\theta}) \right\rangle_{p(\boldsymbol{\theta}|M)} . \end{aligned} \tag{57}$$

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 [7, 11] for the case with Bayesian networks). On the other hand, when the data is incomplete, as in the case of probabilistic clustering, the integral in Eq. 57 is difficult to compute. From this background, including MCMC (Markov chain Monte Carlo) sampling, several approximation methods of log of the marginal likelihood are proposed so far [5]. *Bayesian information criterion* (BIC) [22] 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:

$$Score_{\mathrm{BIC}}(M) \stackrel{\mathrm{def}}{=} p(D \mid M, \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}) - \frac{|\boldsymbol{\theta}|}{2} \log N \tag{58}$$

$$\begin{aligned} Score_{\mathrm{CS}}(M) \stackrel{\mathrm{def}}{=}\ & p(\tilde{D}_c \mid M) - p(\tilde{D}_c \mid M, \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}) \\ & + p(D \mid M, \hat{\boldsymbol{\theta}}_{\mathrm{MAP}}), \end{aligned} \tag{59}$$

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 statistics obtained in the E-step of the EM algorithm.

3.1.11 Random restarts

Since the EM algorithm is a hill-climbing algorithm, being trapped in undesirable *local maxima* is known as one of practical problems in the EM algorithm. NBCTK provides three facilities for avoiding such local maxima — random restarts [21], the deterministic annealing EM (DAEM) algorithm [24], and the split-merge EM (SMEM) algorithm [25].

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

3.1.12 Deterministic annealing EM algorithm*

Since the final estimate of the parameters depends on the choice of initial parameters, in the DAEM algorithm, we attempt to reduce an undesirable influence from the initial parameters in the early stage of EM iterations. To achieve this, by an analogy from statistical mechanics, the free energy is introduced first as:

$$\mathcal{F}_\beta = -\frac{1}{\beta} \log \sum_{k=1}^{K} p(k, \boldsymbol{x} \mid \boldsymbol{\theta})^\beta, \tag{60}$$

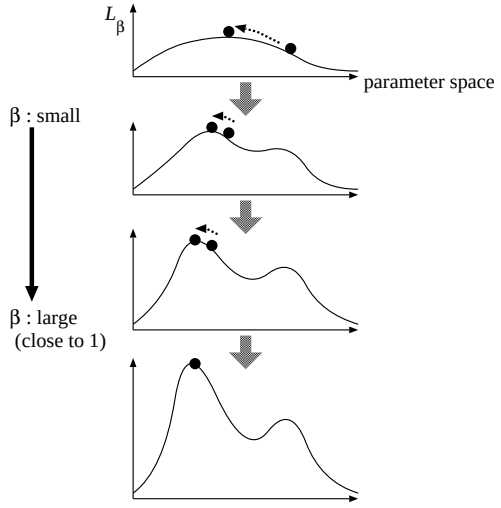

Figure 5: Image of the deterministic annealing EM algorithm.

where β is the *inverse temperature* which controls the influence from the initial parameters. Then we can obtain the DAEM algorithm, which tries to minimize the free energy $\mathcal{F}_\beta$ at each temperature $\frac{1}{\beta}$. Fig. 5 shows an expected behavior of the DAEM algorithm, where L_β is introduced as $-\mathcal{F}_\beta$ (then we will try to maximize L_β). In the DAEM algorithm, we start from the small β, under which the free energy is expected to have a smooth shape, and hopefully has only one local maximum (i.e. the global maximum). Thus, under the smaller β, we may be able to find the global maximum or a good local maximum. When β increases, on the other hand, the shape of the free energy changes (becomes sharper), and hence we should continue to update the parameters by EM iterations. However please note that the starting point of these EM iterations is expected to be more promising than the initial parameters. Finally we perform EM iterations at $\beta = 1$, which is equivalent to the usual EM iterations.

To be more concrete, we show the DAEM algorithm in naive Bayes models in Fig. 6. For an effective use of DAEM algorithm, the temperature schedule is important. In NBCTK, following [24], we start from $\beta_0 = \beta_{\mathrm{init}}$ and then update β by the update rule $\beta_{t+1} \leftarrow \beta_t \cdot \beta_{\mathrm{rate}}$, where β_{init} and β_{rate} are given by the user (the default values are 0.1 and 1.2, respectively).

3.1.13 Split-merge EM algorithm*

The SMEM algorithm is applicable to mixture models, and with this algorithm, we attempt to escape from 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. 7 (a)), and split a cluster, say k_3, that excessively covers objects (Fig. 7 (b)). In the paper that firstly proposed the SMEM algorithm [25], the split operation and the merge operation are always paired, so

1. Initialize randomly the parameters θ_k $(1 \leq k \leq K)$ and θ_{j,k,x_j} $(1 \leq k \leq K, 1 \leq j \leq J, 1 \leq x_j \leq V_j)$.
2. Repeat Step 3 for each $\beta = \beta_0, \beta_1, \beta_2, \ldots, 1.0$
3. Repeat the following until the log-likelihood $\log p(D \mid \boldsymbol{\theta})$ converges:

$$p(k \mid \boldsymbol{x}, \boldsymbol{\theta}) \propto p(k \mid \boldsymbol{\theta})^{\beta} \prod_{j=1}^{J} p(x_j \mid k, \boldsymbol{\theta})^{\beta}. \tag{61}$$

(The rest are the same as those in Fig. 3, the original EM algorithm.)

Figure 6: The DAEM algorithm for ML estimation in naive Bayes models.

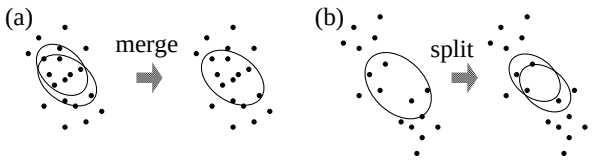

Figure 7: Image of a merge operation and a split operation, where the dots and the ovals stand for objects and clusters, respectively.

the number of resulting 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. In the sequel, for making simpler descriptions, we will concentrate on the case that the attributes are all discrete.

⋄ 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:

$$\begin{aligned} J_{\mathrm{merge}}(k_1, k_2 \mid \boldsymbol{\theta}) &\stackrel{\mathrm{def}}{=} \mathbf{p}_{k_1}(\boldsymbol{\theta})^T \mathbf{p}_{k_2}(\boldsymbol{\theta}) \\ &= \textstyle\sum_{\boldsymbol{x}} N(\boldsymbol{x}) p(k_1 \mid \boldsymbol{x}, \boldsymbol{\theta}) p(k_2 \mid \boldsymbol{x}, \boldsymbol{\theta}), \end{aligned} \tag{62}$$

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:

$$\mathbf{p}_k(\boldsymbol{\theta}) = (p(k \mid \boldsymbol{x}^{(1)}, \boldsymbol{\theta}), p(k \mid \boldsymbol{x}^{(2)}, \theta), \ldots, P(k \mid \boldsymbol{x}^{(N)}, \theta))^T. \tag{63}$$

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:

$$\tilde{J}_{\mathrm{merge}}(k_1, k_2 \mid \boldsymbol{\theta}) \stackrel{\mathrm{def}}{=} \frac{\mathbf{p}_{k_1}(\boldsymbol{\theta})^T \mathbf{p}_{k_2}(\boldsymbol{\theta})}{\|\mathbf{p}_{k_1}(\boldsymbol{\theta})\| \cdot \|\mathbf{p}_{k_2}(\boldsymbol{\theta})\|}. \tag{64}$$

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 $J_{\mathrm{split}}(k_3 \mid \boldsymbol{\theta})$:

$$\begin{aligned} J_{\mathrm{split}}(k_3 \mid \boldsymbol{\theta}) &\stackrel{\mathrm{def}}{=} \mathrm{KL}(\tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta}) \,\|\, p(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})) \\ &= \sum_{\boldsymbol{x}} \tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta}) \log \frac{\tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})}{p(\boldsymbol{x} \mid k_3, \boldsymbol{\theta})}, \end{aligned} \tag{65}$$

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

$$\begin{aligned} \tilde{p}(\boldsymbol{x} \mid k_3, \boldsymbol{\theta}) &\propto \tilde{p}(\boldsymbol{x}) p(k_3 \mid \boldsymbol{x}, \boldsymbol{\theta}) \\ &\propto N(\boldsymbol{x}) p(k_3 \mid \boldsymbol{x}, \boldsymbol{\theta}). \end{aligned} \tag{66}$$

In the above, $\tilde{p}(\boldsymbol{x})$ denotes $N(\boldsymbol{x})/N$, the empirical unconditional probability of $\boldsymbol{x}$. By 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:

$$\theta_{k_1'} := \theta_{k_1} + \theta_{k_2} \tag{67}$$

$$\theta_{j,k_1',x_j} := \frac{E[k_1]\theta_{j,k_1,x_j} + E[k_2]\theta_{j,k_2,x_j}}{E[k_1] + E[k_2]} \tag{68}$$

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 $\theta_{j,k_3',x_j} := \theta_{j,k_3,x_j} + \epsilon'$, 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 ($k' = k_1', k_2', k_3'$):

$$\theta_{j,k',x_j} = \frac{E[k']}{\sum_{k''=k_1',k_2',k_3'} E[k'']} \cdot \sum_{k''=k_1',k_2',k_3'} \theta_{j,k'',x_j} \tag{69}$$

⋄ Behavior of the SMEM algorithm

Fig. 8 shows a typical pattern on the changes in log-likelihood by the SMEM algorithm, where we apply five split-merge operations. Note here that we have only plotted a sequence with

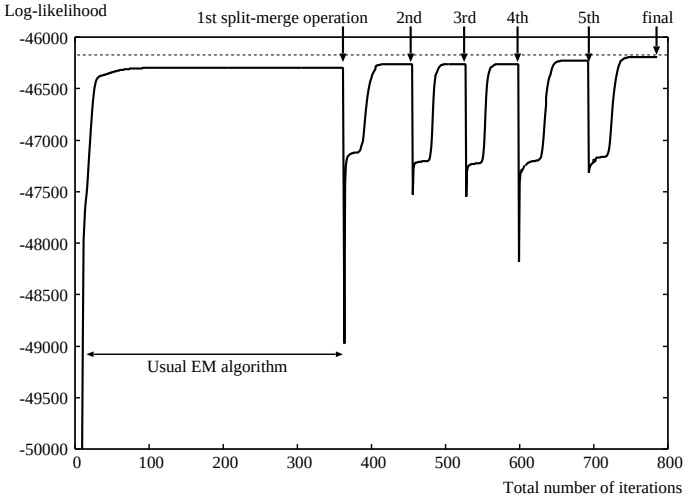

Figure 8: Changes in log-likelihood by the SMEM algorithm.

the accepted split-merge operations. It is seen from Fig. 8 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 §3.1.10, 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 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:

$$\begin{aligned} L(D) &= \log \sum_z \int_\Theta p(D, z, \boldsymbol{\theta} \mid M) d\boldsymbol{\theta} \\ &= \log \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D, M) \frac{p(D, z, \boldsymbol{\theta} \mid M)}{q(z, \boldsymbol{\theta} \mid D, M)} d\boldsymbol{\theta} \\ &\geq \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D, M) \log \frac{p(D, z, \boldsymbol{\theta} \mid M)}{q(z, \boldsymbol{\theta} \mid D, M)} d\boldsymbol{\theta}. \\ &\quad \text{(from Jensen's inequality)} \end{aligned} \tag{70}$$

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:

$$\begin{aligned} L(D) &\geq F[q] \\ &\stackrel{\text{def}}{=} \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \frac{p(D, z, \boldsymbol{\theta})}{q(z, \boldsymbol{\theta} \mid D)} d\boldsymbol{\theta}, \end{aligned} \tag{71}$$

where $F[q]$ can be seen as a lower limit of $L(D)$, and is called the *variational* (or *negative*) *free energy*. Then, to get a good approximation of $L(D)$, we 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 $L(D) = \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log p(D) d\boldsymbol{\theta}$:

$$\begin{aligned} & L(D) - F[q] \\ &= \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \left\{ p(D) \cdot \frac{q(z, \boldsymbol{\theta} \mid D)}{p(D, z, \boldsymbol{\theta})} \right\} d\boldsymbol{\theta} \\ &= \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \frac{q(z, \boldsymbol{\theta} \mid D)}{p(z, \boldsymbol{\theta} \mid D)} d\boldsymbol{\theta} \\ &= \mathrm{KL}(q(z, \boldsymbol{\theta} \mid D) \parallel p(z, \boldsymbol{\theta} \mid D)). \end{aligned} \tag{72}$$

Here, maximizing $F[q]$ implies minimizing the Kullback-Leibler divergence between $q(z, \boldsymbol{\theta} \mid D)$ and $p(z, \boldsymbol{\theta} \mid D)$. As a result, finding q^* leads to a good approximation of $p(z, \boldsymbol{\theta} \mid D)$, the conditional distribution of hidden variables and the parameters.

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

$$q(z \mid D) \propto \exp\left(\int_\Theta q(\boldsymbol{\theta} \mid D) \log p(D, z \mid \boldsymbol{\theta}) d\boldsymbol{\theta} \right), \tag{73}$$

$$q(\boldsymbol{\theta} \mid D) \propto p(\boldsymbol{\theta}) \exp\left(\sum_z q(z \mid D) \log p(D, z \mid \boldsymbol{\theta}) \right). \tag{74}$$

Now we can derive a VB-EM algorithm specific to naive Bayes clustering, shown in Fig. 9, by substituting the distribution form of a naive Bayes model (Eq. 1 and the predefined distributions for attributes) to the generic VB-EM procedure above. For simplicity, we assume in Fig. 9 that the attributes are all discrete. For the descriptions of the VB-EM algorithm for Gaussian mixture models, please see [19, 26].[7] In Fig. 9, π_k and π_{j,k,x_j} are defined as follows:

$$\pi_k \stackrel{\text{def}}{=} \exp\left(\Psi(\alpha_k) - \Psi\left(\sum_{k'=1}^K \alpha_{k'} \right) \right), \tag{75}$$

$$\pi_{j,k,x_j} \stackrel{\text{def}}{=} \exp\left(\Psi\left(\alpha_{j,k,x_j} \right) - \Psi\left(\sum_{x'_j=1}^{V_j} \alpha_{j,k,x'_j} \right) \right), \tag{76}$$

where $\Psi(\cdot)$ is the digamma function. One may find that the resulting VB-EM algorithm receives the data D as input, and outputs the learned hyperparameters $\boldsymbol{\alpha}^*$ (or the learned pseudo counts $\boldsymbol{\delta}^*$). In NBCTK, the initial hyperparameters such as $\alpha_k^{(0)}$

[7] Precisely speaking, the VB-EM algorithm implemented in NBCTK is based on the author's derivation, and is a bit different from the ones in these papers (e.g. some coefficients of terms in the update rules).

1. Initialize randomly the hyperparameters $\alpha_k^{(0)}$ $(1 \leq k \leq K)$ and $\alpha_{j,k,x_j}^{(0)}$ $(1 \leq k \leq K, 1 \leq j \leq J, 1 \leq x_j \leq V_j)$.
2. Repeat the following steps until the free energy $F[q]$ converges:

$$q(k \mid \boldsymbol{x}) \propto \pi_k \prod_{j=1}^{J} \pi_{j,k,x_j} \tag{78}$$

$$E[k] := \sum_{\boldsymbol{x}} N(\boldsymbol{x}) q(k \mid \boldsymbol{x}) \tag{79}$$

$$E_j[k, x_j] := \sum_{\boldsymbol{x}' \in D : x'_j = x_j} N(\boldsymbol{x}') q(k \mid \boldsymbol{x}') \tag{80}$$

$$\alpha_k := \alpha_k^{(0)} + E[k] \tag{81}$$

$$\alpha_{j,k,x_j} := \alpha_{j,k,x_j}^{(0)} + E_j[k, x_j] \tag{82}$$

Figure 9: The VB-EM algorithm in naive Bayes models.

are obtained by adding small random noises. For instance, we obtain $\alpha_k^{(0)}$ by:

$$\alpha_k^{(0)} := \alpha_k + \epsilon_k \ , \tag{77}$$

where α_k is the quantity specified by the user, and ϵ_k is a small random noise.

3.2.2 Clustering

As in the ML/MAP case (§3.1.8), 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 hyperparameters $\boldsymbol{\alpha}^*$, instead of $p(k, \boldsymbol{x} \mid \hat{\boldsymbol{\theta}})$ which relies on the point-estimated value $\hat{\boldsymbol{\theta}}$. That is, we have:

$$\begin{aligned} k^* &= \operatorname*{argmax}_{k:1 \leq k \leq K} p(k \mid \boldsymbol{x}) = \operatorname*{argmax}_{k:1 \leq k \leq K} p(k, \boldsymbol{x}) \\ &= \operatorname*{argmax}_{k:1 \leq k \leq K} \int_\Theta q^*(\boldsymbol{\theta} \mid D) p(k, \boldsymbol{x} \mid \boldsymbol{\theta}) d\boldsymbol{\theta} \\ &= \operatorname*{argmax}_{k:1 \leq k \leq K} p^*(k) \prod_{j=1}^{J} p^*(x_j \mid k) \, , \end{aligned} \tag{83}$$

where p^* is the *predictive distribution*. That is, for the class C and each discrete attribute X_j, the predictive distributions are obtained in closed form:

$$p^*(k) = \alpha_k^* / \textstyle\sum_{k'} \alpha_{k'}^* \tag{84}$$

$$p^*(x_j \mid k) = \alpha_{j,k,x_j}^* / \textstyle\sum_{x'_j} \alpha_{j,k,x'_j}^* \ . \tag{85}$$

On the other hand, for continuous attributes, each $p^*(x_j \mid k)$ is analytically obtained as the Student's t-distribution.

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) such as relevance analysis (§3.1.9). One compromise is to use the probabilities p^* obtained by Eqs. 84 and 85 instead of $\hat{\boldsymbol{\theta}}$, the point-estimated parameters in ML/MAP [3], and actually NBCTK adopts this way. It is important to note however that the resulting probabilistic inferences might *not* be good approximations of the inferences based on a posteriori quantities.

3.2.4 Deterministic annealing EM algorithm*

In NBCTK, the deterministic-annealing version of VB-EM is available. To be specific, following [14], let us revisit the definition of the variational free energy (Eq. 71):

$$\begin{aligned} F[q] &\stackrel{\text{def}}{=} \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log \frac{p(D, z, \boldsymbol{\theta})}{q(z, \boldsymbol{\theta} \mid D)} d\boldsymbol{\theta} \\ &= \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log p(D, z, \boldsymbol{\theta}) d\boldsymbol{\theta} \\ &\quad - \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log q(z, \boldsymbol{\theta} \mid D) d\boldsymbol{\theta} \end{aligned} \tag{86}$$

Again, by an analogy to statistical mechanics, we correspond $F[q]$ with $-\mathcal{F}$ ($\mathcal{F}$: the free energy), the first term in the right hand side of Eq. 86 with $-\mathcal{U}$ ($\mathcal{U}$: the internal energy) and the second term with $\mathcal{S}$ ($\mathcal{S}$: the entropy). Then we newly introduce the variational free energy that takes into account the (inverse) temperature:

$$\begin{aligned} F_\beta[q] &\stackrel{\text{def}}{=} \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log p(D, z, \boldsymbol{\theta}) d\boldsymbol{\theta} \\ &\quad - \frac{1}{\beta} \sum_z \int_\Theta q(z, \boldsymbol{\theta} \mid D) \log q(z, \boldsymbol{\theta} \mid D) d\boldsymbol{\theta}. \end{aligned} \tag{87}$$

The VB-EM algorithm that tries to maximize $F_\beta[q]$ (i.e. the DA-version of the VB-EM algorithm) has a similar procedure to that of the DAEM algorithm for ML/MAP estimation.

3.2.5 Split-merge EM algorithm*

Basically, the SMEM algorithm for VB-EM (say, VB-SMEM) follow almost the same procedures as the ones for the ML/MAP-EM algorithms (§3.1.11 and §3.1.13). The difference is that we need to use the probabilities p^* obtained by Eqs. 84 and 85 to compute two heuristic scores J_{merge} and J_{split} (Eqs. 62–65).

3.3 Evaluation of clusters

As mentioned before, naive Bayes clustering can be viewed as a unsupervised classification of objects, and hence the clusters

form a partition. In this document, such a partition is called a *cluster partition* and denoted by $\pi = \{C_1, C_2, \ldots, C_K\}$, where C_k is a set of objects that belong to the k-th cluster ($1 \leq k \leq K$) and K is the number of clusters.

Sometimes we may want to compare the cluster partition π made by NBCTK with π', the one by the other clustering method. Besides, for the applications where a cluster partition π' made by human is given in advance, we may regard such a cluster partition as the gold standard, and evaluate the quality of our cluster partition. Furthermore, such an evaluation would enable us to tune the control parameters (such as a setting of hyperparameters).

For the purposes above, a number of similarity/dissimilarity measures between two different cluster partitions (π and π') have been proposed so far, and NBCTK automatically computes some of them [16, 18] — *purity/inverse purity*, *Rand index*, *normalized mutual information* and so on — when a cluster partition π' by human or another system is given. These measures take into account that the numbers of clusters in π and π' can differ, and that the obtained clusters are anonymous.

Here let us introduce some terminologies. Suppose that we have a cluster partition $\pi = \{C_1, C_2, \ldots, C_K\}$ obtained by NBCTK and a cluster partition $\pi' = \{C'_1, C'_2, \ldots, C'_{K'}\}$ given by human or by another clustering system (K may not equal K'). Hereafter each C_k is called a *predicted cluster* and each $C'_{k'}$ is called an *answer cluster*. Also let us denote the total number of objects by N (obviously $N = \sum_{k=1}^{K} |C_k| = \sum_{k'=1}^{K'} |C'_{k'}|$ holds).

3.3.1 Purity/Inverse purity

Based on the settings above, purity, inverse purity and their *F-measure* are respectively defined as follows:

$$\begin{aligned} \text{Purity}(\pi, \pi') &\stackrel{\text{def}}{=} \frac{1}{N} \sum_{k=1}^{K} \max_{1 \leq k' \leq K'} |C_k \cap C'_{k'}| \\ &= \sum_{k=1}^{K} \frac{|C_k|}{N} \max_{1 \leq k' \leq K'} \frac{|C_k \cap C'_{k'}|}{|C_k|} \end{aligned} \tag{88}$$

$$\begin{aligned} \text{Inv-Purity}(\pi, \pi') &\stackrel{\text{def}}{=} \text{Purity}(\pi', \pi) \\ &= \frac{1}{N} \sum_{k'=1}^{K'} \max_{1 \leq k \leq K} |C_k \cap C'_{k'}| \\ &= \sum_{k'=1}^{K'} \frac{|C_{k'}|}{N} \max_{1 \leq k \leq K} \frac{|C_k \cap C'_{k'}|}{|C'_{k'}|} \end{aligned} \tag{89}$$

$$\text{Purity}_\beta(\pi, \pi') \stackrel{\text{def}}{=} F_\beta(\text{Purity}(\pi, \pi'), \text{Inv-Purity}(\pi, \pi')) \tag{90}$$

where the F-measure based on P and R ($P, R \in [0, 1]$) with weight β to R is a weighted harmonic mean of P and R:

$$F_\beta(P, R) = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}. \tag{91}$$

As β gets larger, $F_\beta(P, R)$ will put more weights on the recall.

Both purity and inverse purity range over $[0, 1]$ and measures the similarity between π and π' (e.g. purity ≈ 1 means that π is very close to π'). From Eq. 88, purity (resp. inverse purity) can be interpreted as the weighted average of the ratios of the objects that belong to the dominating answer cluster (resp. predicted cluster). One may notice from Eqs. 88 and 89 that purity tends to increase as we have more predicted clusters (K is larger and each predicted cluster C_k becomes smaller accordingly), while inverse purity tends to decrease as we have more predicted clusters. To summarize, purity and inverse purity have biases in opposite directions w.r.t. the number of predicted clusters, and the F-measure is therefore introduced to take a balance between them.

3.3.2 Rand index and related measures

Rand index (RI) is based on the pairs of objects. To define RI, we first introduce the following statistics:

M_{11}: the number of pairs of objects that are in the same cluster under both π and π',

M_{10}: the number of pairs of objects that are in the same cluster under π but not under π',

M_{01}: the number of pairs of objects that are in the same cluster under π' but not under π, and

M_{00}: the number of pairs of objects that are in different clusters under both π and π'.

These statistics are computed by:

$$M_{11} = \sum_{k=1}^{K} \sum_{k'=1}^{K'} \binom{N_{kk'}}{2} = \frac{1}{2} \left(\sum_{k,k'} N_{kk'}^2 - N \right) \tag{92}$$

$$M_{10} = \sum_{k=1}^{K} \binom{N_k}{2} - M_{11} \tag{93}$$

$$M_{01} = \sum_{k'=1}^{K'} \binom{N'_{k'}}{2} - M_{11} \tag{94}$$

$$M_{00} = \binom{N}{2} - (M_{11} + M_{10} + M_{01}) \tag{95}$$

where $N_{kk'} \stackrel{\text{def}}{=} |C_k \cap C'_{k'}|$, $N_k \stackrel{\text{def}}{=} |C_k|$ and $N'_{k'} \stackrel{\text{def}}{=} |C'_{k'}|$. Then, RI is defined as:

$$\text{RI}(\pi, \pi') \stackrel{\text{def}}{=} \frac{M_{11} + M_{00}}{M_{11} + M_{10} + M_{01} + M_{00}} = \frac{M_{11} + M_{00}}{\binom{N}{2}}. \tag{96}$$

Several related measures, Wallace's (asymmetric) measures (W_I, W_II and their F-measure W_β), Fowlkes and Mallows index (FMI), adjusted Rand index (ARI), Jaccard index (JI) and

Mirkin metric (MM), are defined as follows [16, 18]:

$$\mathrm{W_I}(\pi,\pi') \stackrel{\mathrm{def}}{=} M_{11}/(M_{11}+M_{10}) \quad (97)$$

$$\mathrm{W_{II}}(\pi,\pi') \stackrel{\mathrm{def}}{=} M_{11}/(M_{11}+M_{01}) \quad (98)$$

$$\mathrm{W}_\beta(\pi,\pi') \stackrel{\mathrm{def}}{=} F_\beta(\mathrm{W_I}(\pi,\pi'),\mathrm{W_{II}}(\pi,\pi')) \quad (99)$$

$$\mathrm{FMI}(\pi,\pi') \stackrel{\mathrm{def}}{=} \sqrt{\mathrm{W_I}(\pi,\pi')\mathrm{W_{II}}(\pi,\pi')} \quad (100)$$

$$\mathrm{ARI}(\pi,\pi') \stackrel{\mathrm{def}}{=} \frac{\sum_{k,k'}\binom{N_{kk'}}{2} - \sum_k\binom{N_k}{2}\sum_{k'}\binom{N'_{k'}}{2}/\binom{N}{2}}{\frac{1}{2}\left(\sum_k\binom{N_k}{2}+\sum_{k'}\binom{N'_{k'}}{2}\right) - \sum_k\binom{N_k}{2}\sum_{k'}\binom{N'_{k'}}{2}/\binom{N}{2}} \quad (101)$$

$$\mathrm{JI}(\pi,\pi') \stackrel{\mathrm{def}}{=} M_{11}/(M_{11}+M_{10}+M_{01}) \quad (102)$$

$$\mathrm{MM}(\pi,\pi') \stackrel{\mathrm{def}}{=} \sum_k N_k^2 + \sum_{k'} {N'_{k'}}^2 - 2\sum_k\sum_{k'} N_{kk'}^2 \quad (103)$$

$$= 2(M_{10}+M_{01}) = 2\binom{N}{2}(1-\mathrm{RI}(\pi,\pi')).$$

Similarly to purity, the measures based on object pairs are intuitive and would help our evaluation. On the other hand, however, there are some issues we need to note, as described in [18].

3.3.3 Normalized mutual information

Normalized mutual information (NMI) is a similarity measure between $\pi = \{C_1, C_2, \ldots, C_K\}$ and $\pi' = \{C'_1, C'_2, \ldots, C'_{K'}\}$ based on information theory, and defined as:

$$\mathrm{NMI}(\pi,\pi') \stackrel{\mathrm{def}}{=} \frac{I(\pi,\pi')}{(H(\pi)+H(\pi'))/2}, \quad (104)$$

where $I(\pi,\pi')$ is the mutual information between π and π', and $H(\pi)$ is the entropy of π:

$$I(\pi,\pi') \stackrel{\mathrm{def}}{=} \sum_{k=1}^{K}\sum_{k'=1}^{K'} \frac{|C_k \cap C'_{k'}|}{N} \log\frac{N\cdot|C_k\cap C'_{k'}|}{|C_k||C'_{k'}|}, \quad (105)$$

$$H(\pi) \stackrel{\mathrm{def}}{=} -\sum_{k=1}^{K}\frac{|C_k|}{N}\log\frac{|C_k|}{N}, \quad (106)$$

$$H(\pi') \stackrel{\mathrm{def}}{=} -\sum_{k'=1}^{K'}\frac{|C_{k'}|}{N}\log\frac{|C_{k'}|}{N}. \quad (107)$$

As mentioned in [16], NMI is actually normalized (i.e. it ranges over $[0,1]$), since $I(\pi,\pi') \le \min(H(\pi),H(\pi')) \le \frac{1}{2}(H(\pi)+H(\pi'))$. NMI ≈ 1 means that π is very close to π'.

Variation of information (VI) [18] is an information-based distance measure between π and π':

$$\mathrm{VI}(\pi,\pi') \stackrel{\mathrm{def}}{=} H(\pi)+H(\pi')-2I(\pi,\pi'). \quad (108)$$

We can see that a normalized version of VI (NVI) can be obtained using NMI:

$$\mathrm{NVI}(\pi,\pi') \stackrel{\mathrm{def}}{=} \frac{\mathrm{VI}(\pi,\pi')}{H(\pi)+H(\pi')} = 1-\mathrm{NMI}(\pi,\pi'). \quad (109)$$

4 How to use NBCTK

4.1 Overall organization of NBCTK

NBCTK 0.7 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 Contents of the package

NBCTK is packaged depending on platforms:

- Linux (32bit): `nbctk-0.7_linux32.tar.gz`
- Linux (64bit): `nbctk-0.7_linux64.tar.gz`
- Mac OS X (v10.6): `nbctk-0.7_mac.tar.gz`
- Windows: `nbctk-0.7_win.zip`

After the package unfolded, you may find the following subdirectories (or sub-folders):

- `src/` contains C source code.
- `doc/` contains the document files including this manual.
- `bin/` contains executables.
- `example/` contains data examples.

Installation is quite easy — we only need to move/copy the executables above into the folder that appears in the `PATH` environment variable.

To build the executables from the source code, see §4.2.3 for Linux, Mac OS X and Win32 with Cygwin/MinGW, or §4.2.4 for Win32 with MSVC++. If you wish to deal with the files which are not ASCII-safe, the executables need to be rebuilt from the source code after GNU libiconv is additionally installed (§4.2.2).

4.2.2 Preliminary installation (GNU libiconv)

To handle the dataset with non-ASCII-safe character encodings, NBCTK 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. Otherwise and if GNU libiconv is not installed in your environment, you need to install GNU libiconv in advance of the installation of NBCTK.

On Windows, `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`. To obtain `iconv.dll`, we may download `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.[8] On the other hand, the latest version can be built from the source files.[9]

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:

```
./configure Options
make
make install
```

where *Options* are some appropriate setting to your system. For the details on *Options*, please consult the `INSTALL` file in the `src` directory. Also,

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

will show the details of the options for the `configure` script.

In the default setting of the `configure` script, like many of GNU tools, the prefix of the installation directories is set to `/usr/local/`, and then the compiled executables will be installed under `/usr/local/bin`. Since `/usr/local/` is a system area in most systems, there are two typical choices:

- If you wish to install the executables into `/usr/local/bin`, you need to be a super user before typing '`make install`', or need to use the `sudo` command together (i.e. type '`sudo make install`' instead of '`make install`').
- If not, it is possible to install the executables into the directory '*somewhere*`/bin`' under your home directory by giving the `--prefix` option to the `configure` script:

```
./configure --prefix=${HOME}/somewhere
```

[8] It is confirmed that the NBCTK binaries built using MSVC++ 9.0 can run with `iconv.dll` from `libiconv-1.9.1.bin.woe32`.

[9] According to `README.woe32` in the released package of GNU libiconv 1.12 (the latest version as of Dec. 2008), building requires MinGW, and MSVC++ is no longer supported.

4.2.4 Using `Makefile.msvc`

To build NBCTK on Win32 with MSVC++, we attempt to compile the source code by the `cl` command of MSVC++. 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 2008 on Windows XP, please follow the menus: [Start] → [All Programs] → [Microsoft Visual Studio 2008] → [Visual Studio Tools] → [Visual Studio 2008 Command Prompt].
3. At the Command Prompt invoked, visit the `src` folder.
4. Type the following command to compile NBCTK:

   ```
   nmake -f Makefile.msvc
   ```

5. Type the following command to install NBCTK:

   ```
   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 Overview

As is mentioned above, we use the executable `nbc` for ML/MAP based clustering. Roughly speaking, `nbc` conducts the EM algorithm first to estimate the parameters, and then, based on the estimated parameters, makes a couple of requested probabilistic inferences including clustering. We can pass our settings and tasks to `nbc` through the command line arguments:[10]

```
nbc -f Base -x NumAttr \
       -k NumClass Options...
```

Here '`-f` *Base*' indicates that we have a file named *Base*`_freq.`*Ext* containing the target dataset *D*. *Base* will work as the base name of the input/output files listed in Table 2. *Ext* is the file extension, which can be `txt` or `csv` according to the format (See §4.3.3). *NumAttr*, given with `-x`, indicates the number of attributes in *Base*`_freq.`*Ext* (Note that the count column is not counted as an attribute). Typically we also specify the number of classes by `-k` option. The details of optional flags will be described in §4.3.6.

[10] The symbol '\' just means a continuation of the command line, and is used here just for space limitation.

Table 2: Input/Output files for `nbc`. *Base* is the base name, and *Ext* is `txt` (for files in the tab-separated value format) or `csv` (for files in the comma-separated value format).

	Filename	Content
Input	*Base*_`freq`.*Ext*	Description of objects
Input	*Base*_`name.txt`	Attribute names
Input/Output	*Base*_`param`.*Ext*	Parameters
Input	*Base*_`hparam`.*Ext*	Hyperparameters
Output	*Base*_`cluster`.*Ext*	Clustering results
Output	*Base*_`memp`.*Ext*	Membership probs.
Output	*Base*_`rank`.*Ext*	Result of relevance analysis
Output	*Base*_`eval`.*Ext*	Evaluation results
Output	*Base*_`log`.*Ext*	Logs
Output	*Base*_`msg.txt`	Terminal messages

4.3.2 Attribute types

By default, in `nbc`, all attributes in the dataset *D* are considered to be discrete. Hence we need to specify (the indices of) continuous attributes explicitly, using `--gauss` (`-u`) option. Also we may have to specify the following attributes which can be included in real datasets:

- The *ID attribute*, which only identifies the object, but is irrelevant to the characteristics of the object.
- The *answer class attribute*, which is annotated by human for the purpose of cluster evaluation.
- *Ignored attributes*, whose values are recorded in the dataset, but are not used in clustering.

The indices of these attributes are specified by `--id` (`-I`), `--eval` (`-E`) and `--ignore` (`-y`) options (see §4.3.6 for details). When the answer class attribute is specified, `nbc` will conduct cluster evaluation (§3.3).

4.3.3 File format

Fig. 10 illustrates the file format that is common to most of NBCTK's input/output files (the only exception is the input file for attribute names — see §4.3.4). In this format, a file contains several data matrices each having a header part which consists of the lines starting with '`#`'. A header part includes the identifier to the corresponding data matrix and gives a brief description on columns. Each data matrix follows the *tab-separated value format* or the *CSV format*, where a row in the matrix is separated into data cells by tab characters or commas, respectively. Note that the CSV format used here is just a restricted one — e.g. the current version of NBCTK does not understand the quotation characters. Missing values, denoted by '`?`' or '`NA`', are allowed to exist in the data matrix.

The input file format is automatically determined based on the file extension, and the files in the tab-separated value (resp. CSV) format must have the file extension `txt` (resp. `csv`). By default, NBCTK first tries to read *Base*_`freq.txt`,[11] and if the file does not exist, NBCTK then tries to read *Base*_`freq.csv`. As a result, we only need to put the input files with the valid file extension in the current working directory. On the other hand, when '`--input` *Ext*' is given, NBCTK reads *Base*_`freq`.*Ext* only. The output file format is the same as the input file format by default, while we can change it by giving '`--output` *Ext*' or '`-j` *Ext*'.

```
#### ID: Identifier for data matrix #1
#### Brief descriptions of columns
####
                Data matrix #1

#### ID: Identifier for data matrix #2
#### Brief descriptions of columns
####
                Data matrix #2

                    ⋮
```

Figure 10: Common file format in NBCTK.

4.3.4 Attribute names

Since version 0.7, we can *optionally* give NBCTK the names of attributes to get more comprehensible output files. These names are specified in the file *Base*_`name.txt`, which have two columns separated by space or tab characters. Each line corresponds to an attribute, where the first column specifies the name of the attribute and the second column specifies the type of the attribute:

- '`id`' indicates that the attribute is the ID attribute.
- '`eval`' indicates that the attribute is the answer class attribute.
- '`ignored`' indicates that the attribute is an ignored attribute.
- '`nominal`' indicates that the attribute is a discrete attribute which takes on the values appearing in *Base*_`freq`.*Ext*.
- '{*Value*1, *Value*2, . . . , *ValueN*}' indicates that the attribute is a discrete attribute which takes on the values *Value*1, *Value*2, . . . , *ValueN*.
- '`real`' indicates that the attribute is a continuous attribute which essentially takes on real numbers.

[11] When `--read-param` (`-R`) option specified, NBCTK first tries to read *Base*_`param.txt`, and NBCTK then tries to read *Base*_`param.csv`.

- '`integer`' indicates that the attribute is a continuous attribute which essentially takes on real numbers.

The order of the name/type specifications should follow the order of the attributes in the dataset *Base*_`freq`.*Ext* and should be consistent with the type specification with command line options (§4.3.2 and §4.3.6). Currently, the type specifications '`real`' and '`integer`' are *not* distinguished in the clustering algorithms — the attributes of these types are just treated as continuous attributes that follow Gaussian distributions. The characters after '`#`' are ignored in each line, and empty lines are skipped. For example, for the attributes in Table 1 (page 2), `ex1_name.txt` may contain the following specifications:

```
customer-id      id
customer-group   eval
blood-type       nominal
income           real
deposit          ignored
gender           {male, female}
      ⋮                ⋮
```

In the type specification of the form '{*Value*1,*Value*2,...,*ValueN*}', we can specify the values that do not appear in the dataset *Base*_`freq`.*Ext*, but can potentially appear in future or under another study. This may affect the clustering results under a Bayesian learning setting, i.e. MAP estimation (§3.1.2) or variational Bayes (§3.2), and would also be useful in classifying new objects into one of the clusters we already have.

4.3.5 Compression of the indistinguishable objects

As described in §2.1, the objects having the same attribute values will be compressed into a single object unless they have the ID attribute or continuous attributes. This implies that the number of distinct objects can be less than the number of lines in the data file *Base*_`freq`.*Ext*. The number of occurrences of the resulting object is of course the sum of the numbers of occurrences of the distinct objects. On the other hand, we can inhibit such a compression by giving `--uncompress` option to the command.

4.3.6 Command line options

NBCTK provides dozens of command line options for flexible configurations. The options having a short form are considered as major, or as frequently used.

◇ Model specification (general)

`--n-class` *NumClass*
`-k` *NumClass*

By this option, we set *NumClass* to *K*, the number of classes. Given '`-k` *Kmin*:*Kmax*:*Kstep*', `nbc` will enter into the 'model scoring' mode (§3.1.10). That is, with varying the number *K* of classes from *Kmin* to *Kmax* with step *Kstep*, we evaluate the plausibility of *K* based on the score specified by `--score` (`-g`) option. This option is mandatory except when `--read-param` (`-R`) or `--read-hparam` (`-H`) is given.

`--n-attr` *NumAttr*
`-x` *NumAttr*

This option tells `nbc` that the total number of attributes, *including* the ID attribute, the answer class attribute, ignored attributes (§4.3.2), is *NumAttr*.

`--gauss` *ContAttrs*
`-u` *ContAttrs*

This option specifies the zero-based indices of continuous attributes. Basically we specify A_i, the i-th attribute, as continuous and single-dimensional by '`-u` i', whereas the n continuous attributes that jointly follow a multivariate normal distribution is specified by '`-u` i_1`,`i_2`,...,`i_n'. For two or more normal distributions, we use this option for each of them. For example, '`-u 1,3 -u 2`' declares that we have two normal distributions, where the first corresponds to a bivariate normal distribution the attributes (A_1, A_3) follow, and the second corresponds to a univariate normal distribution followed by A_2. For convenience, `nbc` allows a variety of abbreviated forms:

- '`-u 0-3`' is the same as '`-u 0,1,2,3`'.
- '`-u -3`' is the same as '`-u 0-3`'.
- '`-u 2-`' is the same as '`-u 2,3,...,`$(T-1)$', where T is the number of attributes specified by `-x` option.
- '`-u 2-4,8`' is the same as '`-u 2,3,4,8`'.
- '`-u 1#3,4`' is the same as '`-u 1 -u 3,4`'.
- '`-u 2~4`' is the same as '`-u 2#3#4`', and hence as '`-u 2 -u 3 -u 4`'.

`--id` *IdAttr*
`-I` *IdAttr*

This option specifies the zero-based index of the ID attribute.

`--ignore` *IgnoreAttrs*
`-y` *IgnoreAttrs*

This option specifies the zero-based index (indices) of ignored attribute(s). If there are more than one ignored attribute, we can specify them using hyphen and comma. The meanings of hyphen and comma are the same as those for `--gauss` (`-u`) option, i.e. '`-y 2-4,8`' will be interpreted as '`-y 2,3,4,8`', and hence as '`-y 2 -y 3 -y 4 -y 8`'.

`--eval` *AnsAttr*
`-E` *AnsAttr*

This option specifies the zero-based index of the answer class attribute. The answer classes will be recorded into the output files, and NBCTK automatically evaluates the predicted clusters based on the answer classes. The results of evaluation will be recorded into *Base*_`eval`.*Ext*.

`--read-param` (no argument)
`-R` (no argument)

With this option, `nbc` will skip the EM algorithm and read the parameters from *Base*_`param`.*Ext*, which may have been created by hand or by a previous run of `nbc`. With this option, the number of classes is then determined according to the content of *Base*_`param`.*Ext* (so `--n-class` (`-k`) option will be ignored).

◇ Model specification (hyperparameters)

On contrary to AutoClass, NBCTK allows us to specify raw hyperparameters. In addition, for the cases where it is tedious to specify them, `--autoclass` (`-A`) option is provided to make a similar[12] configuration to AutoClass.

`--smooth-c` *ClassPseudoCount*
`-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 as the ones used in AutoClass (§3.1.2). The default is 0.

`--smooth-a` *AttrPseudoCount*
`-w` *AttrPseudoCount*

This option specifies uniform pseudo counts for discrete attributes. If *AttrPseudoCount* is just a non-negative number ζ, all pseudo counts δ_{j,k,x_j} for discrete attributes are equally set as ζ. If *AttrPseudoCount* takes a comma-separated form '$\zeta_{j_1}, \zeta_{j_2}, \cdots, \zeta_{j_M}$', where X_{j_m} is a discrete attribute and M is the number of discrete attributes, the pseudo counts $\delta_{j_m,k,x_{j_m}}$ ($1 \le x_{j_m} \le V_{j_m}$) are equally set as ζ_{j_m}. When '`auto`' is specified for ζ or for some ζ_{j_m}, the corresponding pseudo counts are set as the ones used in AutoClass (§3.1.2). The default values are all 0.

`--tau` *Tau*
`-T` *Tau*

Basically this option sets *Tau* to τ, a hyperparameter of the normal-gamma or the normal-Wishart distribution (see §3.1.2). If *Tau* is given as '`auto`', similarly to the pseudo counts for discrete attributes, τ will be set as $1/K$, where K is the number of classes. If this option is not given, ML estimation will be conducted for the continuous attributes.

[12] To be precise, the configuration with `--autoclass` (`-A`) for continuous attributes is different from Autoclass, because AutoClass uses different prior distributions for continuous attributes.

`--alpha` *Alpha*
`-O` *Alpha*

This option sets *Alpha*/2 (resp. *Alpha*) to $a_{j,k}$ (resp. $\phi_{j,k}$), a hyperparameter of the normal-Gamma (resp. the normal-Wishart) distribution (see §3.1.2). If this option is not given, $a_{j,k}$ will be set as $(1+\epsilon)/2$ (resp. $\phi_{j,k}$ will be set as $n_j + \epsilon$, where n_j is the dimension of attributes X_j), and ϵ is a small positive number. The name of this option might be confusing, and so will be changed in the future release.

`--smooth-mean` *Mean*
`-K` *Mean*

This option sets *Mean* to $\nu_{j,k}$, a hyperparameter of the normal-gamma or the normal-Wishart distribution (see §3.1.2). If this option is not given, we use $\tilde{\boldsymbol{\mu}}_j$ instead for attributes $\boldsymbol{X}_j$, where $\tilde{\boldsymbol{\mu}}_j$ is the sample mean of $\boldsymbol{X}_j$ in the dataset D as if there is no latent class variable (§3.1.6).

`--smooth-var` *Var*
`-L` *Var*

This option configures $b_{j,k}$ (resp. $\boldsymbol{S}_{j,k}$), a hyperparameter of the normal-gamma (resp. the normal-Wishart) distribution (see §3.1.2) as follows:

- If *Var* takes the form '`sample:`κ', b (resp. Σ_0) will be set to $\kappa\tilde{\sigma}_j^2/2$ (resp. $\kappa\tilde{\Sigma}_j$) for attribute X_j (resp. attributes $\boldsymbol{X}_j$), where $\tilde{\sigma}_j^2$ (resp. $\tilde{\Sigma}_j$) is the sample variance $\tilde{\sigma}_j^2$ (resp. the sample covariances $\tilde{\boldsymbol{S}}_j$) of X_j in the dataset D as if there is no latent class variable (§3.1.6).
- If *Var* takes the form '`const:`κ', b (resp. Σ_0) will be set as $\kappa/2$ (resp. $\kappa\boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix).
- If *Var* is '`sample`' (resp. '`const`'), it is interpreted as '`sample:1.0`' (resp. '`const:1.0`).
- If *Var* is a floating-point number κ, it is interpreted as '`sample:`κ'.

`--autoclass` (no argument)
`-A` (no argument)

With this option, we have a similar configuration to AutoClass. This option is equivalent to '`--smooth-c auto --smooth-a auto --tau auto --score cs`'.

`--read-hparam` (no argument)
`-H` (no argument)

With this option, `nbc` will run the MAP-based EM algorithm (§3.1.2) under the pseudo counts (the smoothing constants) read from a file named *Base*_`hparam`.*Ext*, which may have been created by hand or by a previous run of `vnbc` (§4.4). The number of classes is determined according to the content of *Base*_`hparam`.*Ext* (so `--n-class` (`-k`) option will be ignored).

◇ EM algorithm

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

`--seed` *RandomSeed*
`-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).

`--epsilon` *Threshold*
`-e` *Threshold*

`nbc` will use *Threshold* as the threshold ξ for judging the convergence of the likelihood or the a posteriori probability (Step 2 in Fig. 3 and Fig. 4). That is, if the difference between the values of the log-likelihood (or log of the 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^{-3}.

`--restart` *NumInit*
`-n` *NumInit*

With this option, random restarts will be enabled in the EM algorithm (§3.1.11), where the number of trials is *NumInit*. By default, *NumInit* is set as 1, that is, `nbc` will not perform random restarts.

`--max-iter` *MaxIter*
`-m` *MaxIter*

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

`--max-iter-restart` *MaxInitIter*
`-l` *MaxInitIter*

With this option, the maximum number of preliminary EM iterations in random restarts (§3.1.11) will be set as *MaxInitIter*.

`--init-c` *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 (Eq. 39), and '`random`' initializes the parameters more randomly (Eq. 38). The default method is '`noisy_u`'. See §3.1.5 for more details.

`--init-a` *InitAttrMethod*

This option specifies the initialization method for the parameters of discrete attributes (Step 1 in Fig. 3 and Fig. 4). Similarly to `--init-c` option, '`noisy_u`' initializes the parameters based on a uniform distribution with small noises (Eq. 41), and '`random`' initialize the parameters more randomly (Eq. 40). In addition, '`v_freq`' initializes the parameters based on the relative frequencies with small noises (Eq. 42). That is, each parameter θ_{j,k,x_j} will be initialized to $\frac{1}{N}\sum_{\boldsymbol{x}' \in D: x'_j = x_j} N(\boldsymbol{x}')$ with a small noise. The default method is '`noisy_u`'. See §3.1.5 for more details.

`--noise` *Noise*

This option controls the potential magnitude of noises in initialization of the parameters of discrete attributes. For details, see §3.1.5, where *Noise* corresponds to ε_1.

`--noise-gauss` *Noise*

This option controls the potential magnitude of noises in initialization of the parameters of continuous attributes. For details, see §3.1.5, where *Noise* corresponds to ε_2.

◇ Membership distribution

`--mem-prob` (no argument)
`-D` (no argument)

With this option, the membership probabilities $p(k \mid \boldsymbol{x})$ (§3.1.7) will be written into a file '*Base*`_memp.`*Ext*'.

◇ Relevance analysis

`--rank` *RankArg*
`-z` *RankArg*

This option enables relevance analysis (§3.1.9), in which the most relevant objects or attribute values to each class (each cluster) are recorded into '*Base*`_rank.`*Ext*'. If *RankArg* is a floating-point number ρ between 0 and 1, `nbc` only records the objects (resp. attribute values) whose relevance score $R(k, x) = p(k \mid x, \boldsymbol{\theta})$ (resp. $R^j(k, x_j) = p(k \mid x_j, \boldsymbol{\theta})$) is greater than or equal to ρ. If *RankArg* is a positive integer $N \geq 1$, only N most relevant objects or attribute values will be recorded. If *RankArg* is set as 0, the ranking over all objects or attribute values will be recorded.

`--hide-rank` (no argument)

With this option, the objects (resp. the attribute values) whose relevance score $R(k, x) = p(k \mid x, \boldsymbol{\theta})$ (resp. $R^j(k, x_j) = p(k \mid x_j, \boldsymbol{\theta})$) is smaller than $1/K$, where K is the number of classes, will not be written into '*Base*`_rank.`*Ext*'.

◇ Model selection

If we give an option '`-k` *Kmin*`:`*Kmax*`:`*Kstep*', `nbc` will enter into the 'model scoring' mode (§3.1.10) — that is, if we specify '`-k 3:10:2`', `nbc` will compute the scores of the models with the numbers of classes $K = 3, 5, 7, 9$.

`--score` *ModelScore*
`-g` *ModelScore*

This option specifies the model score as *ModelScore*. The available model scores are '`bic`' (BIC, Eq. 58) and '`cs`' (the Cheeseman-Stutz score, Eq. 59). The default score is BIC. The Cheeseman-Stutz score is also enabled by `--autoclass` (`-A`) option.

◇ DAEM algorithm

`--daem` (no argument)
`-b` (no argument)

This option enables the DAEM algorithm.

`--itemp` *InvTempInit*`:`*InvTempRate*
`-B` *InvTempInit*`:`*InvTempRate*

This option sets the initial value β_{init} of the inverse temperature as *InvTempInit*, and the updating coefficient β_{rate} as *InvTempRate*. It is required that $0 < \beta_{\text{init}} \leq 1$ and $\beta_{\text{rate}} > 1$. If '*InvTempInit*`:`' is given as the argument, the default value 1.2 is used for β_{rate}. If '`:`*InvTempRate*' is given, the default value 0.1 is used for β_{init}. From our experience, the appropriate values of β_{init} and β_{rate} seem quite different according to the target data.

When we enable the DAEM algorithm, the terminal messages like below will be displayed ($\beta_{\text{init}} = 0.4$ and $\beta_{\text{rate}} = 1.2$). In these messages, each '$\star$' indicates that the inverse temperature has just been updated.

```
% nbc -f ex1 -k 4 -x 10 -I 0 -E 1 -y 4,9 -u 3,6,7 -b
-B 0.4: -n 3
Reading cases from ex1_freq.csv...done
#classes = 4:
  [0] #iters *0*....100.*.***.* (Converged: 163 iters)
L=-5747.688373
  [1] #iters *0*....100.*.*.*** (Converged: 167 iters)
L=-5747.688373
  [2] #iters *0**...**** (Converged: 75 iters)  L=-574
7.688373
  <<Resumed best parameter set #0>>
  [0] #iters * (Converged: 165 iters)   L=-5747.688373
  BIC = -5923.836132
```

◇ SMEM algorithm

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

`--n-sm` *NumOp*
`-o` *NumOp*

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

`--n-cand` *NumCand*
`-a` *NumCand*

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

`--n-split` *NumSplitCand*
`-G` *NumSplitCand*

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

`--min-c-param` *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.

`--min-new-c-param` *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.

`--min-sm-imp` *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 likelihood or the a posteriori probability, which is specified with `--epsilon` (`-e`) option.

`--norm-jmerge` (no argument)

This option enables the normalized version of J_{merge} (Eq. 64), instead of the original one (Eq. 62).

◇ Evaluation of clusters

When the index of the answer class attribute is given by `--eval` (`-E`) option, NBCTK automatically evaluates the predicted clusters based on the answer clusters. The results under various evaluation measures, described in §3.3, will be written into '*Base*`_eval.`*Ext*'.

`--weight-purity` *Beta*

This option specifies the weight β in the F-measure of purity and inverse purity (Eq. 90) as *Beta*. The default value is 1.

`--weight-wallace` *Beta*

This option specifies the weight β in the F-measure of Wallace's measures (Eq. 99) as *Beta*. The default value is 1.

◇ **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 as UTF-8. Running '`iconv -l`' would show a list of the available codesets. As described in §4.3.3, each input/output file is either in the CSV format or in the tab-separated value format.

`--target` *Base*
`-f` *Base*

This option indicates that we have a file named *Base*`_freq.`*Ext* containing the target dataset *D*. *Base* will also be used as the base name of the input/output files as listed in Table 2.

`--suffix` *Suffix*
`-s` *Suffix*

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

`--input` *Ext*

As described in §4.3.3, `nbc` tries to determine the input file format automatically. On the other hand, with this option, `nbc` only reads the input files that have the file extension *Ext*. The input files should be in the tab-separated value (resp. CSV) format when *Ext* is `txt` (resp. `csv`).

`--output` *Ext*
`-j` *Ext*

As described in §4.3.3, the output file format is the same as the input file format by default. On the other hand, with this option, `nbc` outputs the files whose file extension is *Ext*, where the output files are in the tab-separated value (resp. CSV) format when *Ext* is `txt` (resp. `csv`).

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

`nbc` assumes the character codeset of the input files is *Charcode*. If this option is omitted, the default codeset (which can be specified at compilation time) will be used.

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

`nbc` outputs files with the character codeset *Charcode*. If this option is omitted, the default codeset (which can be specified at compilation time) will be used.

◇ **Miscellaneous**

`--log-scale` (no argument)
`-J` (no argument)

With this option, most of probability values are stored in log-scale. This would remedy the problem of underflow of very small probabilities, but on the other hand, the probability computations will take a bit longer time. Also it is highly recommended to combine this option with the settings for MAP estimation (e.g. with `--autoclass` (`-A`) option).

`--save-msg` (no argument)
`-S` (no argument)

By default, `nbc` outputs the terminal 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 small model/dataset, this option would shorten the runtime.

`--regular-stat` (no argument)

With this option, NBCTK ignores the values of each continuous attribute that are larger than the Q_1%-tile value, and smaller than the Q_2%-tile value, (only) when computing the sample means and the sample variances (see also §3.1.4).

`--extreme` *Q*

When computing the sample statistics with this option and `--regular-stat`, NBCTK ignores the values of each continuous attribute that are larger than the Q_1%-tile value, and smaller than the Q_2%-tile value, where $0 < Q < 1$ should hold, $Q_2 = 100 \times Q$ and $Q_1 = 100 \times (1 - Q)$. Without this option, $Q = 0.05$. See also §3.1.4.

`--uncompress` (no argument)

This option inhibits the compression of indistinguishable objects, which is described in §2.1 and §4.3.5.

`--hide-attr` (no argument)
`-V` (no argument)

By default, the attribute values appearing in the dataset *D* will always be recorded into files. With this option, on the other hand, they will not be recorded.

`--show-attr-memp`

By this option, attribute-wise membership probabilities will be output into *Base*`_memp.`*Ext* .

`--show-attr-cluster`

By this option, attribute-wise clusters will be output into *Base*`_cluster.`*Ext* .

`--prec-out` *NumDigits*

This option sets *NumDigits* to the number of significant digits in each floating-point number to be written into files. At the implementation level, *NumDigits* will be passed as *N* in "`%.`*N*`g`" to `fprintf()` in the standard C library.

`--prec-msg` *NumDigits*

This option sets *NumDigits* to the number of digits after the decimal point in each floating-point number to be displayed on the terminal. At the implementation level, *NumDigits* will be passed as *N* in "`%.Nf`" or "`%.Ne`" to `fprintf()` in the standard C library.

`--verbose` (no argument)

With this option, several temporary results are recorded into '*Base*_`log.`*Ext*'. It should be noted that the resulting log file may become quite large.

`--occ-order` (no argument)

By default, the discrete 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.`*Ext*'.

`--version` (no argument)
`-v` (no argument)

This option displays the version number.

`--help` (no argument)
`-h` (no argument)

This option displays a short description on major optional flags.

`--helpall` (no argument)
`-X` (no argument)

This option displays a short description on all 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 descriptions in the header parts (see §4.3.3). The differences are as follows:

- The options `--read-param` (`-R`), `--autoclass` (`-A`), `--score` (`-g`), `--init-c`, `--init-a` and `--log-scale` (`-J`)[13] are ignored in `vnbc`.
- The meaning of `--read-hparam` (`-H`) is changed as follows:

  `--read-hparam` (no argument)
  `-H` (no argument)

  If this option is given, `vnbc` will skip the VB-EM algorithm and read the hyperparameters from a file named *Base*_`hparam.`*Ext*. The number of classes is then determined according to the content of *Base*_`hparam.`*Ext*.

[13] Most of probability(-like) values are unconditionally stored in log-scale.

Table 3: Input/Output files for `vnbc`. *Base* is the base name, and *Ext* is `txt` (for the file in the tab-separated value format) or `csv` (for the file in the comma-separated value format).

	Filename	Content
Input	*Base*_`freq.`*Ext*	Description of objects
Input	*Base*_`name.txt`	Attribute names
Input/Output	*Base*_`hparam.`*Ext*	Hyperparameters
Output	*Base*_`cluster.`*Ext*	Clustering results
Output	*Base*_`memp.`*Ext*	Membership probs.
Output	*Base*_`rank.`*Ext*	Result of relevance analysis
Output	*Base*_`predict.`*Ext*	Predictive distribution
Output	*Base*_`eval.`*Ext*	Evaluation results
Output	*Base*_`log.`*Ext*	Logs
Output	*Base*_`msg.txt`	Terminal messages

- The meanings of `--smooth-c` (`-c`), `--smooth-a` (`-w`), `--tau` (`-T`), `--alpha` (`-O`), `--smooth-mean` (`-K`) and `--smooth-var` (`-L`) are changed to specify the initial hyperparameters such as $\alpha_k^{(0)}$ and $\alpha_{j,k,x_j}^{(0)}$ in Step 1 of Fig. 9. For instance, if we give '`--smooth-c` *ClassPseudoCount*', the initial hyperparameters $\alpha_k^{(0)}$ are set as (1 + *ClassPseudoCount*) + ϵ_k, where ϵ_k is a small random noise.

We also have an option that is only available in `vnbc`:

`--predict` (no argument)
`-P` (no argument)

With this option, the parameters of the predictive distribution (§3.2.2) will be output into *Base*_`predict.`*Ext*. More specifically, for discrete attributes, $p^*(k)$ and $p^*(x_j \mid k)$ in Eqs. 84 and 85 will be output, and for continuous attributes, the means, the (co-)variances and the degrees of freedom of the Student's *t*-distribution will be output.

4.5 Auxiliary tool for post-processing: `nbcsep`

NBCTK provides a simple auxiliary tool named `nbcsep` for post-processing of the outputs from the executables `nbc` and `vnbc`. Hopefully, using `nbcsep` combined with some UNIX commands such as `grep`, `cut`, `sort`, `join`, and so on, we may not have to write extra post-processing programs (in Perl, etc.).

As described above, an output file contains several 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. 11). The synopsis of `nbcsep` is as follows:

`nbcsep` [*Options*] *InputFile* [*InputFile* `...`]

If '–' is given as an *InputFile*, `nbcsep` assumes that one input is passed from the standard input. If no options are given,

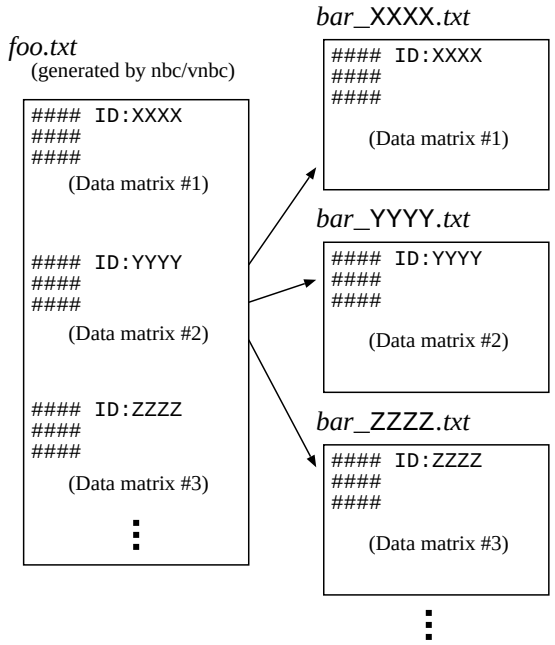

Figure 11: Output file split by `nbcsep` with '`-f bar`'.

`nbcsep` displays all identifiers of the data matrices in *InputFile*s. The below is a list of available options:

`--target` *Base*
`-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`.

`--view` *ID*
`-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 configured by `-m` option.

`--output` *ID*
`-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 `-f` option.

`--match` *Criterion*
`-m` *Criterion*

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

`--suffix` *Suffix*
`-s` *Suffix*

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

`--ext` *Ext*
`-e` *Ext*

`nbcsep` uses *Ext* as a new file extension for the output files. If this option is omitted, the file extension will be set as the one commonly appearing in *InputFile*s. If different extensions are used in *InputFile*s, '`out`' will be used as the default file extension.

`--del-head` (no argument)
`-d` (no argument)

`nbcsep` outputs only the data matrices.

`--del-body` (no argument)
`-D` (no argument)

`nbcsep` outputs only the header parts.

`--hide-file` (no argument)
`-F` (no argument)

`nbcsep` does not display the filenames even if two or more input files are given.

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

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

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

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

`--help` (no argument)
`-h` (no argument)

`nbcsep` displays a simple help message.

5 Miscellaneous remarks

5.1 Settings for attribute types

Even in recent versions, the way of specifying the attribute types might be error-prone. That is, if we miss specifying `--id` (`-I`), `--eval` (`-E`) or `--gauss` (`-u`) option for some attributes, the programs will regard these attributes as discrete descriptive attributes (floating-point numbers such as `"1.234"` will be treated as a label), and finish without reporting the error. To avoid this problem, it is recommended to write *Base*`_name.txt`, which additionally specifies the names/types of attributes (§4.3.4), and then to check the data matrix `SETTING-MODEL-BASE` in the log file *Base*`_log.`*Ext*.

5.2 Settings for EM algorithms

The computation time and space required for an EM iteration is almost linear in each of the number of classes, the number of single-dimensional attributes, and the number of distinct objects in the dataset. Exceptionally, however, it is cubic in the largest dimension of the multivariate normal distributions included in the model.

For some applications, the default value of the threshold for judging convergence of the likelihood or the a posteriori probability, which is specified by `--epsilon` (`-e`) option (§4.3.6), might be small. Early researches on statistical natural language processing 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 [17] discusses the case of hidden Markov models). In other words, there can be cases where it is reasonable to give a larger value to `--epsilon` (`-e`) option, or set the maximum number of iterations with `--max-iterate` (`-m`) option (§4.3.6). In addition, it is empirically observed that the EM algorithm run with larger pseudo counts tends to converge with a smaller number of iterations.

5.3 Precision of floating-point numbers

For large datasets, NBCTK might show an unexpected behavior due to the problems with arithmetic precision such as underflow or precision errors. `--log-scale` option provides a way to avoid underflow. Also, NBCTK uses the `long double` type for floating-point numbers on Linux, Mac OS X, and Win32 with Cygwin. Of course `long double` can be memory consuming than `double`,[14] and thus, to use `double` instead of `long double`, we add the '`--disable-long-double`' flag to the `configure` script. Also it should be addressed that, in the current implementation of the digamma function $\Psi(\cdot)$, floating-point numbers are treated as `double`.

5.4 Numerical problems in ML/MAP-based clustering

As described in §3.1.2, ML/MAP-based clustering with multivariate Gaussian distributions tends to show an unstable behavior for a small dataset (e.g. the updated covariances may be singular). For such cases, the MAP estimation is more preferred than the ML estimation, and `--log-scale` (`-J`) option would make the probability computation more stable.

5.5 Parallelization via OpenMP*

Nowadays a couple of popular C compilers support OpenMP for shared-memory parallel computing. In NBCTK'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. With the GNU C compiler (version 4.2 or later), for instance, we may add `CFLAGS='-fopenmp -O3'` and `LIBS='-lgomp'` to the arguments of the `configure` script, and specify the number of threads at runtime by the `OMP_NUM_THREADS` environment variable.

[14] The physical size of `long double` is system-dependent, and is only guaranteed not to be smaller than that of `double`. If you are using `long double`, the physical size will be recorded into the log file.

Contact information

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

Acknowledgments

First of all, the author would like to thank Masanori Nakagawa, Taisuke Sato and Asuka Terai for offering 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 helpful information on the development tools on Microsoft Windows, and for offering C code of the digamma and the log-gamma functions, which were originally implemented in Fortran for SPECFUN.[15] As a random number generator, we use a C implementation of Mersenne Twister by Takuji Nishimura and Makoto Matsumoto.[16] The development of this software is supported in part by the 21st Century COE Program "Framework for Systematization and Application of Large-scale Knowledge Resources" at Tokyo Institute of Technology.

References

[1] H. Attias. A variational Baysian framework for graphical models. In *Advances in Neural Information Processing Systems 12 (NIPS-99)*, pages 209–215, 1999.

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

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

[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.

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

[16] The code is available at `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`.

[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] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. In *Proc. of the 27th annual meeting on Association for Computational Linguistics (ACL-89)*, pages 76–83, 1989.

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

[8] M. H. DeGroot. *Optimal Statistical Decisions*. John Wiley & Sons, 1970.

[9] 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.

[10] J.-L. Gauvain and C.-H. Lee. Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. *IEEE Trans. on Speech Audio Process*, 2(2):291–298, 1994.

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

[12] 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.

[13] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[14] K. Katahira, K. Watanabe, and M. Okada. Deterministic annealing variant of variational Bayes method. *Journal of Physics*, Conference Series, 95:012015, 2008.

[15] 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.

[16] C. Manning, P. Raghavan, and H. Shütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

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

[18] M. Meilă. Comparing clusterings – an information based measure. *Journal of Multivariate Analysis*, 98:873–895, 2007.

[19] N. Nasios and A. G. Bors. Variational learning for Gaussian mixture models. *IEEE Trans. on Systems, Man, and Cybernetics*, B36(4):849–862, 2006.

[20] 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.

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

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

[23] 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.

[24] N. Ueda and R. Nakano. Deterministic annealing EM algorithm. *Neural Networks*, 11(2):271–282, 1998.

[25] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. SMEM algorithm for mixture models. *Neural Computation*, 12:2109–2128, 2000.

[26] D. Williams, X. Liao, Y. Xue, L. Carin, and B. Krishnapuram. On classification with incomplete data. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(3):427–436, 2007. As described in the paper, their VB-EM algorithm for incomplete data was found in a supplementary document at http://computer.org/tpami/archives.htm, though it seems not available now.