Introduction to naivebayes package

Michal Majka

March 16, 2024

1 Introduction

The naivebayes package presents an efficient implementation of the widely-used Naïve Bayes classifier. It upholds three core principles: efficiency, user-friendliness, and reliance solely on Base R. By adhering to the latter principle, the package ensures stability and reliability without introducing external dependencies [1]. This design choice maintains efficiency by leveraging the optimized routines inherent in Base R, many of which are programmed in high-performance languages like C/C++ or FORTRAN. By following these principles, the naivebayes package provides a reliable and efficient tool for Naïve Bayes classification tasks, ensuring that users can perform their analyses effectively and with ease, even in the presence of missing data. The purpose of this vignette is to offer a comprehensive understanding of the naivebayes package and specifically naive_bayes() function, providing clear insights into its mechanisms and covering its functionalities in detail.

2 Installation

The naivebayes package can be installed from the CRAN repository by executing the following command in the R console:

```r
install.packages("naivebayes")
```

An alternative way of obtaining the package is first downloading the package source from https://CRAN.R-project.org/package=naivebayes, specifying the location of the file and running in the console:

```r
install.packages("path_to_tar.gz", repos = NULL, type = "source")
```

Remember to replace "path_to_tar.gz" with the actual file path. The full source code can be viewed either on the official CRAN repository: https://github.com/cran/naivebayes or on the development repository: https://github.com/majkamichal/naivebayes.

After a successful installation, the package can be loaded with:

[1]Specialized Naïve Bayes functions within the package may optionally utilize sparse matrices if the Matrix package is installed. However, the Matrix package is not a dependency, and users are not required to install or use it.

```
library(naivebayes)
```

This will enable you to utilize the functionality provided by the `naivebayes` package in your `R` environment.

3 Main functions

The general `naive_bayes()` function is designed to determine the class of each feature in a dataset, and depending on user specifications, it can assume various distributions for each feature. It currently supports the following class conditional distributions:

- Categorical distribution for discrete features (with Bernoulli distribution as a special case for binary outcomes)
- Poisson distribution for non-negative integer features
- Gaussian distribution for continuous features
- non-parametrically estimated densities via Kernel Density Estimation for continuous features

In addition to the general Naïve Bayes function, the package provides specialized functions for various types of Naïve Bayes classifiers. The specialized functions are carefully optimized for efficiency, utilizing linear algebra operations to excel when handling dense matrices. Additionally, they can also exploit *sparsity* of matrices for enhanced performance:

- Bernoulli Naïve Bayes via `bernoulli_naive_bayes()`
- Multinomial Naïve Bayes via `multinomial_naive_bayes()`
- Poisson Naïve Bayes via `poisson_naive_bayes()`
- Gaussian Naïve Bayes via `gaussian_naive_bayes()`
- Non-Parametric Naïve Bayes via `nonparametric_naive_bayes()`

These specialized classifiers are tailored to different assumptions about the underlying data distributions, offering users versatile tools for classification tasks. Moreover, the package incorporates various helper functions aimed at enhancing the user experience. Notably, the model fitting functions provided by the package are robust and can effectively handle missing data, ensuring that users can utilize the classifiers even in the presence of incomplete information.

4 Naïve Bayes Model

The Naïve Bayes is a family of probabilistic models that utilize Bayes' theorem under the assumption of conditional independence between the features to predict the class label for a given problem instance. This section introduces the Naïve Bayes framework in a somewhat formal way.

Let us assume that a single problem instance $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ is given. It consists of d values, each being an outcome of a measurement of a different characteristic X_i. For instance, for $d = 3$, the characteristics X_1, X_2 and X_3 may represent age, yearly income and education level, respectively, and x_1, x_2, x_3 are their measurements of a particular person. Furthermore, given $\boldsymbol{X} = \boldsymbol{x}$, which is a compact notation for $(X_1 = x_1, \ldots, X_d = x_d)$, we are interested in predicting another characteristic Y, which can take on K possible values denoted by $(C_1, \ldots, C_K)$. In other words, we have a multi-class classification problem with K specifying the number of classes. If $K = 2$, the problem reduces to the binary classification. The X_is are usually referred to as "features" or "independent variables" and Y as "response" or "dependent variable". In the following, X_is are assumed to be random variables.

In the Naïve Bayes framework, this classification problem is tackled first by applying the Bayes' theorem to the class specific conditional probabilities $\mathbb{P}(Y = C_k | \boldsymbol{X} = \boldsymbol{x})$ and hence decomposing them into the product of the likelihood and the prior scaled by the likelihood of the data:

$$\mathbb{P}(Y = C_k | \, \boldsymbol{X} = \boldsymbol{x}) = \frac{\mathbb{P}(Y = C_k) \, \mathbb{P}(\boldsymbol{X} = \boldsymbol{x} \, | \, Y = C_k)}{\mathbb{P}(\boldsymbol{X} = \boldsymbol{x})} \tag{1}$$

Since the random variables $\boldsymbol{X} = (X_1, X_2, \ldots, X_d)$ are (naïvely) assumed to be conditionally independent, given the class label C_k, the likelihood $\mathbb{P}(\boldsymbol{X} = \boldsymbol{x} | \, Y = C_k)$ on the right-hand side can be simply re-written as

$$\mathbb{P}(Y = C_k | \, \boldsymbol{X} = \boldsymbol{x}) = \frac{\mathbb{P}(Y = C_k) \, \prod_{i=1}^{d} \mathbb{P}(X_i = x_i | \, Y = C_k)}{\mathbb{P}(X_1 = x_1, \ldots, X_d = x_d)} \tag{2}$$

Since the denominator $\mathbb{P}(X_1 = x_1, \ldots, X_d = x_d)$ is a constant with respect to the class label C_k, the conditional probability $\mathbb{P}(Y = C_k | \, \boldsymbol{X} = \boldsymbol{x})$ is proportional to the numerator:

$$\mathbb{P}(Y = C_k | \, \boldsymbol{X} = \boldsymbol{x}) \propto \mathbb{P}(Y = C_k) \prod_{i=1}^{d} \mathbb{P}(X_i = x_i | \, Y = C_k) \tag{3}$$

In order to avoid a numerical underflow (when $d >> 0$), these calculations are performed on the log scale:

$$\log \mathbb{P}(Y = C_k | \, \boldsymbol{X} = \boldsymbol{x}) \propto \log \mathbb{P}(Y = C_K) + \sum_{i=1}^{d} \log \mathbb{P}(X_i = x_i | \, Y = C_k) \tag{4}$$

Finally, to obtain a final prediction, a *maximum a posteriori* decision rule is applied, selecting the class with the highest sum of the log-prior probability and the log-likelihood:

$$\hat{C} = \underset{k \in \{1, \ldots, K\}}{\arg\max} \left(\log \mathbb{P}(Y = C_K) + \sum_{i=1}^{d} \log \mathbb{P}(X_i = x_i | \, Y = C_k) \right) \tag{5}$$

This process is equivalent to using the `predict()` function with the argument `type = "class"`, which performs the classification based on the same principle.

If the main focus is on obtaining class posterior probabilities $\mathbb{P}(Y = C_k | \boldsymbol{X} = \boldsymbol{x})$, which is equivalent to using the `predict(..., type = "prob")` function. The sum of log-prior probability and log-likelihood in (4) is transformed back to the original scale for each class and are normalized to ensure they sum up to 1.

4.1 Prior distribution

Since the response variable Y can take on K distinct values denoted as $C_1, \ldots, C_K$, each prior probability $\mathbb{P}(Y = C_k)$ in (1) can be interpreted as the probability of observing the label C_k. By default, these prior probabilities are modelled with a Categorical distribution in the `naivebayes` package. The parameters are estimated using Maximum Likelihood Estimation (MLE), whereby the prior probabilities correspond to the proportions of classes in the sample (i.e., the number of samples in the class divided by the total number of samples).

For specifying custom prior probabilities, the parameter `prior` can be used. For instance, if there are three classes ($K = 3$) and we believe that they are equally likely then we may want to assign a uniform prior simply with `naive_bayes(..., prior = c(1/3, 1/3, 1/3)`. Note that the manually specified probabilities have to follow the order of factor levels.

4.2 Available class conditional distributions

Each individual feature X_i can take on either discrete or continuous values. In the case of discrete features, X_i can assume values from a finite or infinite set of items, where the set may be countable or uncountable. For example, this includes scenarios such as categorical variables with a finite number of categories or distributions like the Poisson where the set of possible values is infinitely countable. On the other hand, for continuous features, X_i can take any real-valued number within a certain range, allowing for a continuum of possible values.

Discrete features are identified in `naive_bayes()` as variables of class "character," "factor," and "logical." Additionally, "integer" is treated as a discrete variable when `naive_bayes(..., usepoisson = TRUE)` is used. On the other hand, continuous features are identified as variables with the class "numeric." Depending on the type of feature X_i, the `naive_bayes()` function employs different probability distributions to model the class conditional probability $\mathbb{P}(X_i = x_i | Y = C_k)$

In this subsection, the available class conditional distributions are introduced, and the process of assigning them to the features is elaborated upon.

4.2.1 Categorical distribution

If X_i is discrete feature which takes on m possible values denoted by $\mathcal{X}_i = \{value_1, \ldots, value_m\}$, then the Categorical distribution is assumed:

$$\mathbb{P}(X_i = value_l \, | Y = C_k) = p_{ikl}$$

where $l \in \{1, \ldots, m\}$, $p_{ikl} > 0$ and $\sum_{j=1}^{m} p_{ikj} = 1$. This mathematical formalism can be translated into plain English as follows: given the class label C_k, the probability that the i-th feature takes on the l-th value is non-negative and the sum of M such probabilities is 1. The Bernoulli distribution is the special case for $m = 2$. It is important to note that the logical (TRUE/FALSE) vectors are internally coerced to character ("TRUE"/"FALSE") and hence are assumed to be discrete features. Also, if the feature X_i takes on 0 and 1 values only and is represented in `R` as a "numeric" then the Gaussian distribution is assumed by default.

If the Bernoulli distribution ($m = 2$) applies to all features, the model can be referred to as a "Bernoulli Naïve Bayes". In such a case, the model can also be estimated using the specialized function `bernoulli_naive_bayes()`.

4.2.2 Poisson distribution

For a non-negative integer feature X_i, if the Poisson distribution is explicitly requested using `naive_bayes(..., usepoisson = TRUE)`, then the model assumes:

$$\mathbb{P}(X_i = v | \, Y = C_k) = \frac{\lambda_{ik}^{v} e^{-\lambda_{ik}}}{v!},$$

where $\lambda_{ik} > 0$ and $v \in \{0, 1, 2, \ldots\}$. When this distribution applies to all features, the model is referred to as a "Poisson Naïve Bayes". In such a case, the model can also be estimated using the specialized function `poisson_naive_bayes()`.

4.2.3 Gaussian distribution

For a continuous feature X_i, the Gaussian distribution is assumed by default:

$$\mathbb{P}(X_i = v | \, Y = C_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} \exp\left(-\frac{(v - \mu_{ik})^2}{2\sigma_{ik}^2}\right),$$

where μ_{ik} and σ_{ik}^2 are the class conditional mean and variance. If this applies to all features, then the model can be called a "Gaussian Naïve Bayes". In such a case, the model can also be estimated using the specialized function `gaussian_naive_bayes()`.

4.2.4 Kernel distribution

If X_i is continuous, instead of the Gaussian distribution, a kernel density estimation (KDE) can be alternatively used to obtain a non-parametric representation of the conditional probability density function. It can be requested via `naive_bayes(..., usekernel = TRUE)`. If this applies to all features then the model can be called a "Non-parametric Naïve Bayes". In such a case, the model can also be estimated using the specialized function `nonparametric_naive_bayes()`.

4.2.5 Multinomial distribution

For the data $X_1 = x_1, \ldots, X_d = x_d$, where features represent counts or frequency, as commonly encountered in text classification tasks, the Multinomial distribution can be assumed for the class conditional distributions:

$$P(X_1 = x_1, \ldots, X_d = x_d | Y = C_k) = \frac{n!}{\prod_{i=1}^{d} x_i!} \cdot \prod_{i=1}^{d} p_{ik}^{x_i}$$

where d is the number of features, x_i represents the count or frequency of feature X_i, and $p_{ik} = P(X_i = x_i | C_k)$ is the probability of observing feature X_i in class C_k. Additionally, the following constraints apply: $\sum_{j=1}^{d} p_{jk} = 1$, ensuring that the sum of probabilities for all features in a class equals one, and $n = \sum_{i=1}^{d} x_i$, ensuring that the sum of features for a given sample equals n.

It is important to note that the Multinomial distribution is **not** available in the general `naive_bayes()` function. Instead, the specialized function `multinomial_naive_bayes()` is provided, specifically designed to handle data where features follow a Multinomial distribution. This model is commonly referred to as Multinomial Naïve Bayes.

4.3 Assignment of distributions to the numeric features

The assignment of distributions to numeric features in the `naive_bayes()` function depends on the parameters `usekernel` and `usepoisson`, defining the type of class conditional distributions applied to numeric variables. Below are the scenarios based on different combinations of these parameters:

- Scenario 1: Gaussian Distribution (Default)
  - Description: When both `usekernel` and `usepoisson` are set to FALSE (default), the Gaussian distribution is applied to each numeric variable.
  - Example: If the dataset contains numeric variables representing continuous measurements such as height and weight, the Gaussian distribution can be considered suitable for modelling the distributions of these variables.
- Scenario 2: Kernel Density Estimation (KDE)
  - Description: When `usekernel` is set to TRUE and `usepoisson` is FALSE, kernel density estimation (KDE) is applied to each numeric variable.
  - Example: If the dataset contains numeric variables with non-standard distributions, such as multimodal or skewed data, using KDE can provide a more flexible representation of their distributions compared to the Gaussian assumption.
- Scenario 3: Gaussian and Poisson Distributions
  - Description: When `usekernel` is FALSE and `usepoisson` is TRUE, Gaussian distribution is applied to each double vector, while the Poisson distribution is applied to each integer vector.
  - Example: In a dataset where some numeric variables represent continuous measurements (e.g., IQ) while others represent count data (e.g., the number of students achieving a low and high mark in an exam), this scenario allows for modelling each type of variable appropriately.
- Scenario 4: KDE and Poisson Distributions
  - Description: When both `usekernel` and `usepoisson` are set to TRUE, KDE is applied to each double vector, and the Poisson distribution is applied to each integer vector.
  - Example: If the dataset contains a mixture of continuous and count variables, this scenario provides a flexible approach by using KDE for continuous variables and Poisson for count variables.

5 Parameter estimation

In the context of parameter estimation, a training set $(y^{(j)}, \boldsymbol{x}^{(j)})$ is considered, where $y^{(j)} \in \{C_1, \dots, C_k\}$ denotes the class labels and $\boldsymbol{x}^{(j)} = (x_1^{(j)}, \dots, x_d^{(j)})$ represents the feature values for each observation $j = 1, \dots, n$. All observations are assumed to be independent. The objective is to fit a Naïve Bayes model using this training data, which necessitates the estimation of parameters for the class conditional distributions $\mathbb{P}(X_i = x_i | Y = C_k)$. Details regarding the specification of the prior distribution were previously discussed in subsection 4.

5.1 Categorical distribution

Each class conditional Categorical distribution is estimated from the data using the Maximum-Likelihood method by default. However, when the discrete feature X_i encompasses a large number of possible values relative to the sample size, certain combinations of its values and class labels may be absent, resulting in zero probabilities when using Maximum-Likelihood. This issue, known as the zero-frequency problem, can be addressed through a technique called additive smoothing.

Additive smoothing involves adding a small amount, often referred to as a pseudo-count, to the count for every feature value-class label combination. By doing so, the probabilities of rare or unseen combinations are adjusted, preventing them from being assigned zero probabilities. This adjustment ensures a more robust estimation of the class conditional distributions.

In the context of Naïve Bayes classification, additive smoothing can be easily implemented by setting the parameter `laplace` to a positive value. For example, `naive_bayes(..., laplace = 1)` applies additive smoothing by adding a pseudo-count of 1 to every feature value-class label combination. The parameter controlling additive smoothing is named `laplace` because it is the most popular special case of the additive-smoothing when a pseudo-count of 1 is being used.

Interestingly, the application of additive smoothing in this context can be viewed as a Bayesian estimation approach. By incorporating prior knowledge, in the form of the pseudo-count, into the estimation process, the estimation procedure moves from the pure Maximum-Likelihood framework and embraces Bayesian principles. This adjustment not only mitigates the zero-frequency problem but also introduces a degree of regularization, improving the overall robustness of the model.

It is important to note that the `laplace` parameter applies globally, affecting all discrete features and integer features modelled with the Poisson distribution.

5.1.1 Maximum Likelihood

When i-th feature takes on m values in $\mathcal{X}_i = \{value_1, \dots, value_m\}$, then the corresponding Maximum-Likelihood estimates are given by:

$$\hat{p}_{ikl} = \frac{\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k \text{ } and \text{ } x_i^{(j)} = value_l)}{\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)} = \frac{c_{ikl}}{\sum_{j=1}^{m} c_{ikj}}$$

where $\mathbb{1}$ is an indicator function that is 1 when the condition is satisfied and is 0 otherwise. Thus, the Maximum-Likelihood yields very natural estimates: it is a ratio of the number of time the class label C_k

is observed together with the l-th value of the i-th feature to the the number of times the class label C_k is observed.

5.1.2 Additive Smoothing and Bayesian estimation

Applying additive smoothing is commonly used to avoid zero probabilities in the context of Naïve Bayes classification. It involves adding a pseudo-count $\alpha > 0$ to the frequency of each feature value, thereby adjusting the expected probabilities and ensuring that the resulting estimates are guaranteed to be non-zero. The adjusted estimates are given by:

$$\hat{p}_{ikl} = \frac{c_{ikl} + \alpha}{\sum_{j=1}^{m} c_{ikj} + m\alpha}$$

where c_{ikl} is the frequency of the l-th value for the i-th feature and the k-th class, while m denotes the total number of different values. When $\alpha = 0$, each $\hat{p}_{ikl}$ coincides with the Maximum-Likelihood estimates. Conversely, as α increases towards infinity, these estimates converge towards uniform probabilities, represented as $\frac{1}{m}, \frac{1}{m}, \ldots, \frac{1}{m}$.

In the context of Bayesian inference, these estimates correspond to the expected value of the posterior distribution[2], when the symmetric Dirichlet distribution with the parameter $\boldsymbol{\alpha} = (\alpha, \ldots, \alpha)$ is chosen as a prior for probabilities $(p_{ik1}, ..., p_{ikm})$. The prior distribution is parametrized with m equal values of α, which can be interpreted as representing α additional counts observed for each feature value. By incorporating these pseudo-counts into the estimation process, prior knowledge is explicitly included, ensuring that estimates cannot be zero. Moreover, since the same amount is added to each count of feature values, no parameter is favored over any other. Typically, α is chosen to be 1, as this results in a symmetric Dirichlet prior that is equivalent to a uniform distribution. In scenarios with a larger number of observations, such a uniform prior has minimal impact on the estimates. Another common choice for α is 0.5, corresponding to the widely-used non-informative Jeffreys prior.

5.2 Poisson distribution

Estimating parameters for class conditional Poisson distributions, similar to the Categorical distribution, can be accomplished through either Maximum-Likelihood estimation or a Bayesian approach by incorporating pseudo-counts into the data.

5.2.1 Maximum Likelihood

In Maximum Likelihood Estimation, the parameter estimates for the Poisson parameter λ_{ik} are simply sample averages. This means that each class conditional parameter λ_{ik} is estimated using the following algorithm:

$$\hat{\lambda}_{ik} = \frac{\sum_{j=1}^{n} x_i^{(j)} \mathbb{1}(y^{(j)} = C_k)}{\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)} = \frac{N_{ik}}{N_k}.$$

[2]Details on the derivation of the posterior: `https://www.youtube.com/watch?v=UDVNyAp3T38` - this resource was chosen because it is very accessible and provides great explanations.

5.2.2 Bayesian estimation via pseudo-counts

When partitioning the sample into different classes C_k, it is possible to encounter sub-samples where only zero counts are observed. In such cases, Maximum-Likelihood estimation yields zero estimates, posing a challenge. To address this issue, pseudo-counts can be introduced using a parameter `laplace`, adding a Bayesian flavor to the parameter estimation while mitigating the problem of zero estimates.

Similar to Maximum-Likelihood estimation, the values of the i-th feature are initially partitioned according to the k-th class C_k, resulting in a sub-sample with a potentially varying number of data points. This is denoted by $N_k = \sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)$, with a sub-total $N_{ik} = \sum_{j=1}^{n} x_i^{(j)} \mathbb{1}(y^{(j)} = C_k)$. Then, a pseudo-count $\alpha > 0$ is added to the sub-total, and the parameter λ_{ik} is estimated as follows:

$$\hat{\lambda}_{ik} = \frac{N_{ik} + \alpha}{N_k}$$

The estimate $\hat{\lambda}_{ik}$ aligns closely with the expected value of the posterior distribution, which follows a $\text{Gamma}(N_{ik} + \alpha, N_k)$ distribution. This interpretation emerges when we adopt an improper (degenerate) Gamma distribution as the prior for the Poisson likelihood, characterized by a shape parameter $\alpha > 0$ and a rate parameter $\beta \to 0$.

Introducing pseudo-counts, such as 1 and 0.5 for α, corresponds to employing specific priors. For instance, setting $\alpha = 1$ implies using a uniform prior, where all parameter values are equally likely. On the other hand, choosing $\alpha = 0.5$ corresponds to the non-informative Jeffreys prior.

5.3 Gaussian distribution

The parameters of each class conditional Gaussian distribution are estimated via Maximum-Likelihood:

$$\hat{\mu}_{ik} = \frac{\sum_{j=1}^{n} x_i^{(j)} \mathbb{1}(y^{(j)} = C_k)}{\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)}$$

$$\hat{\sigma}_{ik}^2 = \frac{\sum_{j=1}^{n} (x_i^{(j)} - \hat{\mu}_{ik})^2 \mathbb{1}(y^{(j)} = C_k)}{\left[\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)\right] - 1}$$

5.4 Kernel distribution

Kernel density estimation offers a non-parametric approach to estimating the probability density function of each class. This technique is particularly useful when the underlying distribution is unknown or complex (multimodal, etc). The class conditional probability density function for the k-th class can be estimated using kernel density estimation:

$$\hat{f}_{h_{ik}}(x) = \frac{1}{n_k h_{ik}} \sum_{j=1}^{n} K\left(\frac{x - x_i^{(j)}}{h_{ik}}\right) \mathbb{1}(y^{(j)} = C_k),$$

where n_k is number of samples in the k-th class, $K(\cdot)$ is a kernel function that defines the shape of the density curve and h_{ik} is a class specific bandwidth controlling smoothness. The estimation is performed using built in `R` function `stats::density()`. In general, there are 7 different smoothing kernels and 5 different bandwidth selectors available.

Table 1: Available smoothing kernels and bandwidth selectors in stats::density(...).

Kernels	**Bandwidth selectors**
Gaussian	nrd0 (Silverman's rule-of-thumb)
Epanechnikov	nrd (variation of the rule-of-thumb)
Rectangular	ucv (unbiased cross-validation)
Triangular	bcv (biased cross-validation)
Biweight	SJ (Sheather & Jones method)
Cosine	
Optcosine	

By default, the Gaussian kernel and Silverman's rule-of-thumb bandwidth selector are chosen. For more details on available kernel functions and bandwidth selectors, refer to `help(density)` and `help(bw.nrd0)`.

5.5 Multinomial distribution

The parameter estimation for the Multinomial distribution is very analogous to that in the Categorical distribution described in 5.1. In fact, Categorical distribution is a special case of the Multinomial distribution.

5.5.1 Maximum Likelihood

For d features, $X_1 = x_1, \dots, X_m = x_d$, let $N_{ik} = \sum_{j=1}^{n} x_i^{(j)} \mathbb{1}(y^{(j)} = C_k)$ denote the total count of the feature X_i occurring in instances belonging to class C_k, and $N_k = \sum_{i=1}^{d} N_{ik}$ denotes the total count of all features observed within class C_k. Then, the Maximum-Likelihood estimates for the Multinomial distribution parameters are given by:

$$\hat{p}_{ik} = \frac{N_{ik}}{N_k}$$

This equation computes the relative frequency of feature X_i in class C_k, obtained by dividing the count of X_i in class C_k by the total count of all features in that class.

5.5.2 Bayesian estimation via pseudo-counts

In practice, encountering the zero-count problem, where certain feature values do not occur in some classes, is common. To address this issue and prevent zero probabilities, additive smoothing can be applied by adding a small pseudo-count $\alpha > 0$ to each count via the `laplace` parameter. The adjusted estimates are given by:

$$\hat{p}_{ik} = \frac{N_{ik} + \alpha}{N_k + \alpha d}$$

where N_{ik} represents the count of feature X_i occurring in instances belonging to class C_k, N_k is the count of all features observed within class C_k, d is the number of features, and α is the smoothing parameter.

As previously discussed in Section 5.1, these estimates align with the Bayesian perspective, where the symmetric Dirichlet distribution with parameter $\boldsymbol{\alpha} = (\alpha, \ldots, \alpha)$ serves as a prior for the Multinomial probabilities $(p_{1k}, \ldots, p_{dk})$. This choice ensures that the counts are always greater than zero and allows for the incorporation of additional information without favoring any specific parameter. Popular choices for α include 1, corresponding to the uniform prior, and 0.5, corresponding to the Jeffreys prior.

6 General usage

This section provides a comprehensive demonstration of the functionalities of the `naive_bayes()` function through two illustrative examples.

6.1 Training with Formula Interface

The first example showcases the process of training a Naïve Bayes classification model using the formula interface. It covers steps such as data preparation, model fitting, summary of the model, classification of new data, and visualization of fitted distributions. This example serves as a practical guide for users looking to utilize the formula interface for classification tasks.

```
# Section: General usage - Training with formula interface
library(naivebayes)

## naivebayes 1.0.0 loaded
## For more information please visit:
## https://majkamichal.github.io/naivebayes/

# Simulate data
n <- 100
set.seed(1)
data <- data.frame(class = sample(c("classA", "classB"), n, TRUE),
                   bern = sample(LETTERS[1:2], n, TRUE),
                   cat  = sample(letters[1:3], n, TRUE),
                   logical = sample(c(TRUE,FALSE), n, TRUE),
                   norm = rnorm(n),
                   count = rpois(n, lambda = c(5,15)))

# Split data into train and test sets
train <- data[1:95, ]
test <- data[96:100, -1]

# General usage via formula interface
nb <- naive_bayes(class ~ ., train, usepoisson = TRUE)
```

```
# Show summary of the model
summary(nb)
```

```
##
## ================================= Naive Bayes ==================================
##
## - Call: naive_bayes.formula(formula = class ~ ., data = train, usepoisson = TRUE)
## - Laplace: 0
## - Classes: 2
## - Samples: 95
## - Features: 5
## - Conditional distributions:
##     - Bernoulli: 2
##     - Categorical: 1
##     - Poisson: 1
##     - Gaussian: 1
## - Prior probabilities:
##     - classA: 0.4842
##     - classB: 0.5158
##
## --------------------------------------------------------------------------------
```

```
# Classification
predict(nb, test, type = "class") # nb %class% test
```

```
## [1] classA classB classA classA classA
## Levels: classA classB
```

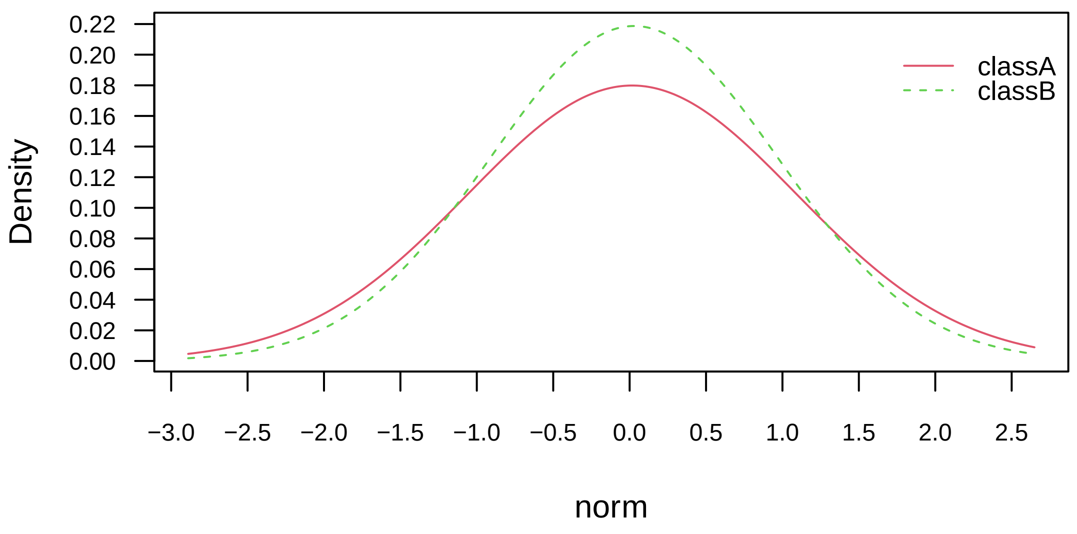
```
# Posterior probabilities
predict(nb, test, type = "prob") # nb %prob% test
```

```
##         classA    classB
## [1,] 0.6708181 0.3291819
## [2,] 0.2792804 0.7207196
## [3,] 0.6214784 0.3785216
## [4,] 0.5806921 0.4193079
## [5,] 0.7074807 0.2925193
```

```
# Tabular and visual summaries of fitted distributions for a given feature
tables(nb, which = "norm")
```

```
## --------------------------------------------------------------------------------
## :: norm (Gaussian)
## --------------------------------------------------------------------------------
```

```
##
## norm       classA     classB
##   mean 0.01676159 0.02924558
##   sd   1.07402111 0.94078797
##
## ---------------------------------------------------------------------------------

plot(nb, which = "norm")
```



```
# Get names of assigned class conditional distributions
get_cond_dist(nb)

##          bern           cat       logical          norm         count
##   "Bernoulli" "Categorical"   "Bernoulli"    "Gaussian"     "Poisson"
```

6.2 Training with matrix/data.frame - vector interface

The second example demonstrates model fitting and classification using the matrix/data.frame - vector interface. It involves simulating data with multiple predictors, fitting a Naïve Bayes model, summarizing the model, and performing classification on new data.

```
# Section: Model fitting and classification using matrix/data.frame - vector interface
library(naivebayes)
```

```r
# Simulate data
n_vars <- 10
n <- 1e6
y <- sample(x = c("a", "b"), size = n, replace = TRUE)

# Discrete features
X1 <- matrix(data = sample(letters[5:9], n * n_vars, TRUE),
             ncol = n_vars)
X1 <- as.data.frame(X1)

# Fit a Naive Bayes model using matrix/data.frame - vector interface
nb_cat <- naive_bayes(x = X1, y = y)

# Show summary of the model
summary(nb_cat)
```

```
##
## ================================= Naive Bayes ==================================
##
## - Call: naive_bayes.default(x = X1, y = y)
## - Laplace: 0
## - Classes: 2
## - Samples: 1000000
## - Features: 10
## - Conditional distributions:
##     - Categorical: 10
## - Prior probabilities:
##     - a: 0.5005
##     - b: 0.4995
##
## --------------------------------------------------------------------------------
```

```r
# Classification
system.time(pred2 <- predict(nb_cat, X1))
```

```
##    user  system elapsed
##   0.208   0.033   0.243
```

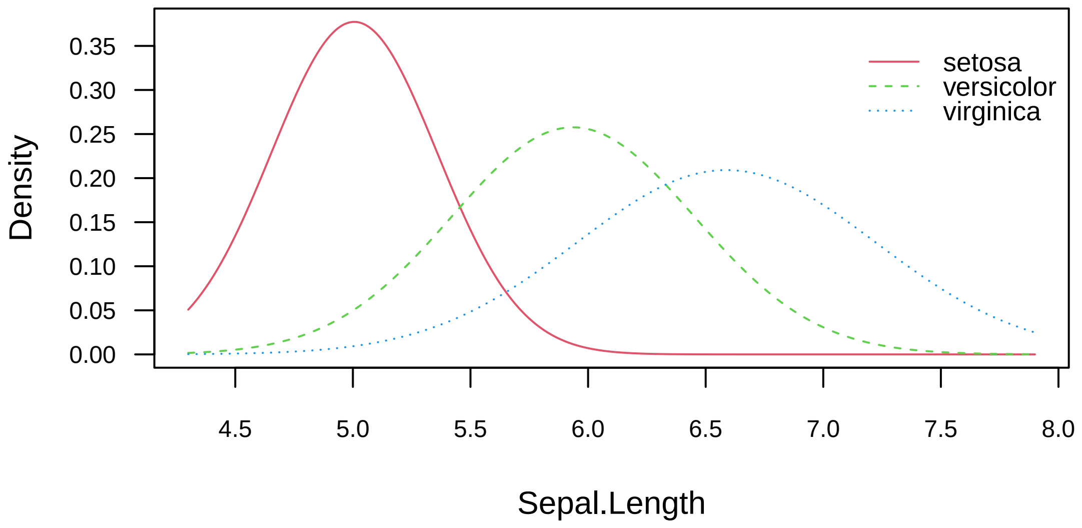
```r
head(pred2)
```

```
## [1] a a b a b a
## Levels: a b
```

6.3 Specialized Gaussian Naive Bayes

This example focuses on showcasing the functionalities of the `gaussian_naive_bayes()` function, which is specifically designed for handling continuous data with Gaussian distribution assumptions. It includes steps such as data preparation, model fitting, summary, and classification, providing insights into the practical application of this specialized function. Other specialized fitting functions have analogous interface.

```
# Section: Model estimation through a specialized fitting function
library(naivebayes)

# Prepare data (matrix and vector inputs are strictly necessary)
data(iris)
M <- as.matrix(iris[, 1:4])
y <- iris$Species

# Train the Gaussian Naive Bayes
gnb <- gaussian_naive_bayes(x = M, y = y)
summary(gnb)

##
## ============================= Gaussian Naive Bayes =============================
##
## - Call: gaussian_naive_bayes(x = M, y = y)
## - Samples: 150
## - Features: 4
## - Prior probabilities:
##     - setosa: 0.3333
##     - versicolor: 0.3333
##     - virginica: 0.3333
##
## --------------------------------------------------------------------------------

# Parameter estimates
coef(gnb)

##              setosa:mu setosa:sd versicolor:mu versicolor:sd virginica:mu
## Sepal.Length     5.006 0.3524897         5.936     0.5161711        6.588
## Sepal.Width      3.428 0.3790644         2.770     0.3137983        2.974
## Petal.Length     1.462 0.1736640         4.260     0.4699110        5.552
## Petal.Width      0.246 0.1053856         1.326     0.1977527        2.026
##              virginica:sd
## Sepal.Length    0.6358796
## Sepal.Width     0.3224966
## Petal.Length    0.5518947
## Petal.Width     0.2746501

coef(gnb)[c(TRUE, FALSE)] # show only means

##              setosa:mu versicolor:mu virginica:mu
## Sepal.Length     5.006         5.936        6.588
## Sepal.Width      3.428         2.770        2.974
## Petal.Length     1.462         4.260        5.552
## Petal.Width      0.246         1.326        2.026

tables(gnb, 1)

## --------------------------------------------------------------------------------
## :: Sepal.Length (Gaussian)
## --------------------------------------------------------------------------------
##       setosa versicolor virginica
## mu 5.0060000  5.9360000 6.5880000
## sd 0.3524897  0.5161711 0.6358796
##
## --------------------------------------------------------------------------------

# Visualization of fitted distributions
plot(gnb, which = 1)
```



```
# Classification
head(predict(gnb, newdata = M, type = "class")) # head(gnb %class% M)

## [1] setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica

# Posterior probabilities
head(predict(gnb, newdata = M, type = "prob")) # head(gnb %prob% M)

##      setosa   versicolor    virginica
## [1,]      1 2.981309e-18 2.152373e-25
## [2,]      1 3.169312e-17 6.938030e-25
## [3,]      1 2.367113e-18 7.240956e-26
## [4,]      1 3.069606e-17 8.690636e-25
## [5,]      1 1.017337e-18 8.885794e-26
## [6,]      1 2.717732e-14 4.344285e-21

# Equivalent calculation via naive_bayes
gnb2 <- naive_bayes(M, y)
head(predict(gnb2, newdata = M, type = "prob"))

##      setosa   versicolor    virginica
## [1,]      1 2.981309e-18 2.152373e-25
## [2,]      1 3.169312e-17 6.938030e-25
## [3,]      1 2.367113e-18 7.240956e-26
## [4,]      1 3.069606e-17 8.690636e-25
## [5,]      1 1.017337e-18 8.885794e-26
## [6,]      1 2.717732e-14 4.344285e-21
```

7 Appendix

7.1 Practical examples: parameter estimation

This subsection provides a practical demonstration of parameter estimation in the Naïve Bayes model. It is aimed at students who are learning about the technical aspects of model fitting for the first time.

7.1.1 Categorical distribution

In this example, the well-known `iris` dataset is modified by introducing a random categorical feature called "new" with 3 levels/categories. The parameters are then estimated using Maximum-Likelihood Estimation (MLE) and Bayesian estimation with pseudo-counts.

It's important to highlight that the example is intentionally designed to have one level with a very low probability of occurrence, which may not appear in the sample. Consequently, MLE may fail to provide a meaningful estimate for this level, resulting in a 0 probability estimate. This situation could lead to misleading results since it implies that the event is impossible, which might not be the case. Moreover, assigning a 0 probability to a category makes the calculation of posterior probabilities misleading, as it effectively renders the posterior probability 0 regardless of the probabilities assigned by other features.

In contrast, Bayesian estimation with the addition of pseudo counts offers a more reliable estimate, ensuring that even unlikely levels receive non-zero probabilities.

```r
library(naivebayes)

# Prepare data: --------------------------------------------------------
data(iris)
iris2 <- iris
N <- nrow(iris2)
n_new_factors <- 3
factor_names <- paste0("level", 1:n_new_factors)

# Add a new categorical feature, where one level is very unlikely
set.seed(2)
iris2$new <- factor(sample(paste0("level", 1:n_new_factors),
                           prob = c(0.005, 0.7, 0.295),
                           size = 150,
                           replace = TRUE), levels = factor_names)

# Define class and feature levels: -------------------------------------
Ck <- "setosa"
level1 <- "level1"
level2 <- "level2"
level3 <- "level3"
```

```r
# level1 did not show up in the sample but we know that it
# has 0.5% probability to occur.
table(iris2$new)

# Parameter estimation: -----------------------------------------------

# ML-estimates
ck_sub_sample <- table(iris2$new[iris$Species == Ck])
ck_mle <-  ck_sub_sample / sum(ck_sub_sample)

# Bayesian estimation via symmetric Dirichlet prior with concentration parameter 0.5
# (corresponds to the Jeffreys' uninformative prior)

laplace <- 0.5
N1 <- sum(iris2$Species == Ck & iris2$new == level1) + laplace
N2 <- sum(iris2$Species == Ck & iris2$new == level2) + laplace
N3 <- sum(iris2$Species == Ck & iris2$new == level3) + laplace
N <-  sum(iris2$Species == Ck) + laplace * n_new_factors
ck_bayes <- c(N1, N2, N3) / N

# Compare estimates
rbind(ck_mle, ck_bayes)

# Unlike MLE, the Bayesian estimate for level1 assigns positive probability
# but is slightly overestimated. Compared to MLE,
# estimates for level2 and level3 have been slightly shrunken.

# In general, the higher value of laplace, the more resulting
# distribution tends to the uniform distribution.
# When laplace would be set to infinity
# then the estimates for level1, level2 and level3
# would be 1/3, 1/3 and 1/3.

# Comparison with estimates obtained with naive_bayes function:
nb_mle <- naive_bayes(Species ~ new, data = iris2)
nb_bayes <- naive_bayes(Species ~ new, data = iris2, laplace = laplace)

# MLE
rbind(ck_mle,
      "nb_mle" = tables(nb_mle, which = "new")[[1]][ ,Ck])
```

```r
# Bayes
rbind(ck_bayes,
      "nb_bayes" = tables(nb_bayes, which = "new")[[1]][ ,Ck])


# Impact of 0 probabilities on posterior probabilities:
new_data <- data.frame(new = c("level1", "level2", "level3"))

# The posterior probabilities are NaNs, due to division by 0 when normalization
predict(nb_mle, new_data, type = "prob", threshold = 0)

# By default, this is remediated by replacing zero probabilities
# with a small number given by threshold.
# This leads to posterior probabilities being equal to prior probabilities
predict(nb_mle, new_data, type = "prob")
```

7.1.2 Gaussian distribution

In this example, the famous `iris` dataset is again used to demonstrate the Maximum-Likelihood estimation of the mean and variance in class conditional Gaussian distributions.

```r
# Prepare data: ---------------------------------------------------------
data(iris)

# Define the feature and class of interest
Xi <- "Petal.Width" # Selected feature
Ck <- "versicolor"  # Selected class

# Build class sub-sample for the selected feature
Ck_Xi_subsample <- iris[iris$Species == Ck, Xi]

# Maximum-Likelihood Estimation (MLE)
mle_norm <- cbind("mean" = mean(Ck_Xi_subsample),
                  "sd" = sd(Ck_Xi_subsample))

# MLE estimates obtained using the naive_bayes function
nb_mle <- naive_bayes(x = iris[Xi], y = iris[["Species"]])
rbind(mle_norm,
      "nb_mle" = tables(nb_mle, which = Xi)[[Xi]][ ,Ck])
```

7.1.3 Kernel Density Estimation

In this example, kernel density estimation (KDE) is used to estimate class conditional densities for Sepal.Width variable from the `iris` dataset.

```r
# Prepare data: ----------------------------------------------------------
data(iris)

# Selected feature
Xi <- "Sepal.Width"

# Classes
C1 <- "setosa"
C2 <- "virginica"
C3 <- "versicolor"

# Build class sub-samples for the selected feature
C1_Xi_subsample <- iris[iris$Species == C1, Xi]
C2_Xi_subsample <- iris[iris$Species == C2, Xi]
C3_Xi_subsample <- iris[iris$Species == C3, Xi]

# Estimate class conditional densities for the selected feature
dens1 <- density(C1_Xi_subsample)
dens2 <- density(C2_Xi_subsample)
dens3 <- density(C3_Xi_subsample)

# Visualisation: ---------------------------------------------------------
plot(dens1, main = "", col = "blue", xlim = c(1.5, 5), ylim = c(0, 1.4))
lines(dens2, main = "", col = "red")
lines(dens3, main = "", col = "black")
legend("topleft", legend = c(C1, C2, C3),
       col = c("blue", "red", "black"),
       lty = 1, bty = "n")

# Compare to the naive_bayes: --------------------------------------------
nb_kde <- naive_bayes(x = iris[Xi], y = iris[["Species"]], usekernel = TRUE)
plot(nb_kde, prob = "conditional")

dens3
nb_kde$tables[[Xi]][[C3]]
tables(nb_kde, Xi)[[1]][[C3]]
```

```r
# Use custom bandwidth selector: ---------------------------------------
?bw.SJ
nb_kde_SJ_bw <- naive_bayes(x = iris[Xi], y = iris[["Species"]],
                        usekernel = TRUE, bw = "SJ")
plot(nb_kde, prob = "conditional")


# Visualize all available kernels: -------------------------------------
kernels <- c("gaussian", "epanechnikov", "rectangular","triangular",
            "biweight", "cosine", "optcosine")
iris3 <- iris
iris3$one <- 1

sapply(kernels, function (ith_kernel) {
    nb <- naive_bayes(formula = Species ~ one, data = iris3,
                        usekernel = TRUE, kernel = ith_kernel)
    plot(nb, arg.num = list(main = paste0("Kernel: ", ith_kernel),
                              col = "black"), legend = FALSE)
    invisible()
})
```

7.1.4 Poisson distribution

This example illustrates the parameter estimation for class conditional Poisson features based on simulated data.

```r
# Simulate data: -------------------------------------------------------
cols <- 2
rows <- 10
set.seed(11)
M <- matrix(rpois(rows * cols, lambda = c(0.1,1)), nrow = rows,
            ncol = cols, dimnames = list(NULL, paste0("Var", seq_len(cols))))
y <- factor(sample(paste0("class", LETTERS[1:2]), rows, TRUE))
Xi <- M[ ,"Var1", drop = FALSE]

# MLE: -----------------------------------------------------------------
# Estimate lambdas for each class
tapply(Xi, y, mean)

# Compare with naive_bayes
pnb <- naive_bayes(x = Xi, y = y, usepoisson = TRUE)
tables(pnb, 1)
```

```r
# Adding pseudo-counts via laplace parameter: --------------------------
laplace <- 1
Xi_pseudo <- Xi
Xi_pseudo[y == "classB",][1] <- Xi_pseudo[y == "classB",][1] + laplace
Xi_pseudo[y == "classA",][1] <- Xi_pseudo[y == "classA",][1] + laplace

# Estimates
tapply(Xi_pseudo, y, mean)

# Compare with naive_bayes
pnb <- naive_bayes(x = Xi, y = y, usepoisson = TRUE, laplace = laplace)
tables(pnb, 1)
```

7.1.5 Multinomial distribution

This example is based on simulated data generated for an artificial scenario. It simulates word counts for a collection of documents, distinguishing between spam and non-spam categories. The data generation process is artificial and does not represent real-world documents. It serves as a simplified illustration of estimating multinomial probabilities for each class.

```r
# Prepare data for an artificial example: --------------------------------------
set.seed(1)
cols <- 3       # words
rows <- 100     # all documents
rows_spam <- 10 # spam documents

# Probability of no-spam for each word
prob_non_spam <- prop.table(runif(cols)) # C_1

# Probability of spam for each word
prob_spam <- prop.table(runif(cols)) # C_2

# Simulate counts of words according to the multinomial distributions
M1 <- t(rmultinom(rows - rows_spam, size = cols, prob = prob_non_spam))
M2 <- t(rmultinom(rows_spam,        size = cols, prob = prob_spam))
M <- rbind(M1, M2)
colnames(M) <- paste0("word", 1:cols) ; rownames(M) <- paste0("doc", 1:rows)
head(M)

# Simulate response with spam/no-spam
y <- c(rep("non-spam", rows - rows_spam), rep("spam", rows_spam))
```

```r
# Additive smoothing
laplace <- 0.5

# Estimate the multinomial probabilities p_{ik} (i is word, k is class)
# p1 = (p_11, p_21, p_31) (non-spam)
# p2 = (p_12, p_22, p_32) (spam)

N_1 <- sum(M1)
N_i1 <- colSums(M1)
p1 <- (N_i1 + laplace) / (N_1 + cols * laplace)

N_2 <- sum(M2)
N_i2 <- colSums(M2)
p2 <- (N_i2 + laplace) / (N_2 + cols * laplace)

# Combine estimated Multinomial probabilities for each class
cbind("non-spam" = p1, "spam" = p2)

# Compare to the multinomial_naive_bayes
mnb <- multinomial_naive_bayes(x = M, y = y, laplace = laplace)
coef(mnb)
# colSums(coef(mnb))
```