Introduction to naivebayes package

Michal Majka

October 27, 2019

1 Introduction

The **naivebayes** package provides an efficient implementation of the popular Naïve Bayes classifier. It was developed and is now maintained based on three principles: it should be efficient, user friendly and written in **Base R**. The last implies no dependencies, however, it neither denies nor interferes with the first as many functions from the **Base R** distribution use highly efficient routines programmed in lower level languages, such as **C** or **FORTRAN**. In fact, the **naivebayes** package utilizes only such functions for resource-intensive calculations. This vignette should make the implementation of the general **naive_bayes()** function more transparent and give an overview over its functionalities.

2 Installation

Just like many other **R** packages, the **naivebayes** can be installed from the **CRAN** repository by simply typing into the console the following line:

```
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:

```
# path_to_tar.gz file <- " "
install.packages(path_to_tar.gz, repos = NULL, type = "source")
```

The full source code can be viewed either on the official **GitHub CRAN** repository: `https://github.com/cran/naivebayes` or on the development repository: `https://github.com/majkamichal/naivebayes`. After successful installation, the package can be used with:

```
library(naivebayes)
```

3 Main functions

The general function **naive_bayes()** detects the class of each feature in the dataset and, depending on the user choices, assumes possibly different distribution for each feature. It currently supports following class conditional distributions:

- categorical distribution for discrete features
- Poisson distribution for non-negative integers
- Gaussian distribution for continuous features
- non-parametrically estimated densities via Kernel Density Estimation for continuous features

In addition to that, specialized Naive Bayes classifiers are available and are listed below. They are implemented based on the linear algebra operations which makes them efficient on the dense matrices. In future, sparse matrices will be also supported.

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

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 $\boldsymbol{x} = (x_1, x_2, ..., x_d)$ 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 $(X_1 = x_1, ..., X_d = x_d)$, we are interested in predicting another characteristic Y, which can take on K possible values denoted by $(C_1, ..., C_K)$. 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 Naive 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 $\boldsymbol{X} = (X_1, X_2, ..., X_d)$ 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

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

Since the denominator $\mathbb{P}(X_1 = x_1, ..., X_d = x_d)$ 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:

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

Lastly, the class with the highest log-posterior probability is chosen to be the prediction:

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

which is equivalent to `predict(..., type = "class")`.

If instead, the conditional class probabilities $\mathbb{P}(Y = C_k | \boldsymbol{X} = \boldsymbol{x})$ are of the main interest, which, in turn, is equivalent to `predict(..., type = "prob")`, then the log-posterior probabilities in (4) are transformed back to the original space and then normalized.

4.1 Prior distribution

Since the response variable Y can take on K distinct values denoted as $(C_1, ..., C_K)$, each prior probability $\mathbb{P}(Y = C_k)$ in (3) can be interpreted as the probability of seeing the label C_k and they are modelled, by default, with a Categorical distribution in the `naivebayes` package. The parameters are estimated with MLE and thus the prior probabilities correspond to proportions of classes in the sample ((number of samples in the class) / (total number of samples)). Prior probabilities can be also specified using the parameter `prior`. 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 a value from a finite/infinte set of m individually identified items (discrete feature) or it can be any real valued number (continuous feature). Discrete features are identified in `naive_bayes()` as variables of class "character", "factor", "logical" and "integer" when `naive_bayes(..., usepoisson = TRUE)`. On the other hand, continuous features are identified as variables with the class "numeric". Depending on the kind of the feature X_i, the `naivebayes()` function uses a different probability distribution for modelling of the class conditional probability $\mathbb{P}(X_i = x_i | Y = C_k)$. In this subsection, available class conditional distributions are first introduced and then it is elaborated on how they are assigned to the features.

4.2.1 Categorical distribution

If X_i is discrete feature which takes on M possible values denoted by $\mathcal{X}_i = \{item1, ..., item_M\}$, then the Categorical distribution is assumed:

$$\mathbb{P}(X_i = l \,|Y = C_k) = p_{ikl}, \tag{6}$$

where $l \in \mathcal{X}_i$, $p_{ikl} > 0$ and $\sum_{j \in \mathcal{X}_i} p_{ikj} = 1$. 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 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 tha 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.

4.2.2 Poisson distribution

If X_i is a non-negative integer feature and explicitly requested via `naive_bayes(..., usepoisson = TRUE)` then the Poisson distribution is assumed:

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

where $\lambda_{ik} > 0$ and $v \in \{0, 1, 2, ...\}$. If this applies to all features, then the model can be called a "Poisson Naive Bayes".

4.2.3 Gaussian distribution

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

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

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 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 Naive Bayes".

4.3 Assignment of distributions to the features

The class "numeric" contains "double" (double precision floating point numbers) and "integer". Depending on the parameters `usekernel` and `usepoisson` different class conditional distributions are applied to columns in the dataset with the class "numeric":

- If `usekernel=FALSE` and `poisson=FALSE` then Gaussian distribution is applied to each "numeric" variable ("numeric"&"integer" or "numeric"&"double")

- If `usekernel=TRUE` and `poisson=FALSE` then kernel density estimation (KDE) is applied to each "numeric" variable ("numeric"&"integer" or "numeric"&"double")
- If `usekernel=FALSE` and `poisson=TRUE` then Gaussian distribution is applied to each "double" vector and Poisson to each "integer" vector:
  - Gaussian: "numeric"&"double"
  - Poisson: "numeric"&"integer"
- If `usekernel=TRUE` and `poisson=TRUE` then kernel density estimation (KDE) is applied to each "double" vector and Poisson to each "integer" vector:
  - KDE: "numeric"&"double"
  - Poisson: "numeric"&"integer"

By default `usekernel=FALSE` and `poisson=FALSE`, thus Gaussian is applied to each numeric variable. On the other hand, "character", "factor" and "logical" variables are assigned to the Categorical distribution with Bernoulli being its special case.

5 Parameter estimation

Let us assume that we have some training set $(y^{(j)}, \boldsymbol{x}^{(j)})$ for $j \in \{1, ..., n\}$, where each $y^{(j)} \in \{C_1, ..., C_k\}$ and $\boldsymbol{x}^{(j)} = (x_1^{(j)}, ..., x_d^{(j)})$. All observations are assumed to be independent and based on this sample we want to fit the Naive Bayes model, which requires parameters of class conditional distributions $\mathbb{P}(X_i = x_i | Y = C_k)$ to be estimated. Specifying the prior distribution was already discussed in the subsection 4.

5.1 Categorical distribution

Every class conditional Categorical distribution is estimated from the data with Maximum-Likelihood method, by default. However, when the discrete feature X_i takes on a large number of possible values relative to the sample size, some combinations of its values and class labels may not be present, which inevitably leads to zero probabilities, when using Maximum-Likelihood. In order to avoid the zero-frequency problem, usually, a small amount (pseudo-count) is added to the count for every feature value - class label combination. This is known as additive smoothing and can be easily accomplished by setting the parameter `laplace` to any positive value. For instance: `naive_bayes(..., laplace = 1)` smooths each discrete feature by adding pseudocount 1 for every feature value-class label combination. The parameter responsible for controlling the additive smoothing is called `laplace` because it is the most popular special case when one pseudo-count is added. Interestingly, by applying the additive smoothing, we leave the Maximum-Likelihood world and enter the Bayesian estimation realm. It is important to note that the `laplace` parameter is global, i.e. it is applied to all discrete features and also non-negative integer features when they are modelled with the Poisson distribution.

5.1.1 Maximum Likelihood

When i-th feature takes values in $\mathcal{X}_i = \{item_1, ..., item_M\}$, then the corresponding Maximum-Likelihood estimates are given by:

$$\hat{p}_{ikl} = \frac{\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k \text{ and } x_i^{(j)} = l)}{\sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)} = \frac{c_{ikl}}{\sum_{j \in \mathcal{X}_i} c_{ikj}}$$

where $l \in \mathcal{X}_i$ and $\mathbb{1}$ 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 the additive smoothing is commonly thought of as means of avoiding zero probabilities - adding a pseudo-count $\alpha > 0$ to the frequency of each item (feature value) changes the expected probabilities and the resulting estimates are guaranteed to be non-zero. They are given by:

$$\hat{p}_{ikl} = \frac{c_{ikl} + \alpha}{\sum_{j \in \mathcal{X}_i} c_{ikj} + M\alpha}, \quad l \in \mathcal{X}_i = \{item_1, ..., item_M\},$$

where c_{ikl} is, again, the frequency of the l-th item for the i-th feature and k-th class and M is the number of different items. It can be easily seen that when the parameter $\alpha = 0$ then each $\hat{p}_{ikl}$ coincides with the Maximum-Likelihood and for $\alpha \to \infty$ they are shrunk towards the uniform probabilities $\{\frac{1}{M}, ..., \frac{1}{M}\}$.

In the Bayesian realm, these estimates correspond to the expected value of the posterior distribution[1], when the symmetric Dirichlet distribution with the parameter $\boldsymbol{\alpha} = (\alpha, ..., \alpha)$ is chosen as a prior. The latter is parametrised with M equal values α which, in this context, can be interpreted as having observed α additional counts of each item and then incorporating them into the estimation process. Thus, adding pseudo-counts allows to explicitly include the prior information into the parameter estimation that the estimates cannot be zero. Also, since the same amount is added to each item's count, no parameter is favoured over any other. The parameter α is usually chosen to be 1 because in such case the symmetric Dirichlet prior is equivalent to the uniform distribution and for bigger number of observations in the data such (uniform) prior has small effect on the estimates. The other popular value for α is 0.5, which corresponds to the popular (non-informative) Jeffreys prior.

5.2 Poisson distribution

In the current implementation, the parameters of class conditional Poisson distributions, similarly as in case of the categorical distribution, can be estimated using either Maximum-Likelihood method or the Bayesian approach by adding pseudo-counts to the data.

[1] 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.

5.2.1 Maximum Likelihood

The classical maximum likelihood parameter estimates for the Poisson lambda are simply sample averages. This means that each class conditional parameter λ_{ik} is estimated via applying following algorithm:

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

5.2.2 Bayesian estimation via pseudo-counts

When the sample is segmented according to different classes C_k, it may happen that in some sub-samples only zero counts are to be found and in such case Maximum-Likelihood estimates yields zero estimates. In such case, pseudo-counts can be introduced via global `laplace` parameter to add the Bayesian flavour to the parameter estimation and to alleviate zero-estimates problem in the same time.

Analogously to the Maximum-Likelihood estimation, the values of the i-th feature are first segmented according to the k-th class C_k, which results in a sub-sample with a possibly different number of data points denoted by $n_k = \sum_{j=1}^{n} \mathbb{1}(y^{(j)} = C_k)$ and the sub-total $T_{ik} = \sum_{j=1}^{n} x_i^{(j)} \mathbb{1}(y^{(j)} = C_k)$. Then a pseudo-count $\alpha > 0$ is added to the sub-total and the parameter λ_{ik} is estimated via:

$$\hat{\lambda}_{ik} = \frac{T_{ik} + \alpha}{n_k}$$

The estimate $\hat{\lambda}_{ik}$ could be considered to coincide with the expected value of posterior distribution given by $\text{Gamma}(T_{ik} + \alpha, n_k)$, when the improper (degenerate) Gamma distribution with shape parameter $\alpha > 0$ and rate $\beta \to 0$ is chosen as a prior for the Poisson likelihood. Adding pseudo-counts 1 and 0.5 ($\alpha = 1$ or $\alpha = 1$) corresponds to the estimation using the uniform prior and Jeffreys prior, respectively.

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

The non-parametric estimate for the k-th class conditional probability density function can be obtained using a 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 that controls the 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	

The Gaussian smoothing kernel and Silverman's rule-of-thumb are chosen by default. Please see `help(density)` and `help(bw.nrd0)` for more details on available kernel functions and bandwidth selectors.

6 General usage

```
library(naivebayes)

## naivebayes 0.9.6 loaded

### 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)))
train <- data[1:95, ]
test <- data[96:100, -1]


### General usage via formula interface
nb <- naive_bayes(class ~ ., train, usepoisson = TRUE)
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: 0.5263
##     - classB: 0.4737
##
## -------------------------------------------------------------------------------
```

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

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

```
# Alternatively
nb %class% test
```

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

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

```
##         classA    classB
## [1,] 0.4815380 0.5184620
## [2,] 0.4192209 0.5807791
## [3,] 0.6882270 0.3117730
## [4,] 0.4794415 0.5205585
## [5,] 0.5209152 0.4790848
```

```
# Alternatively
nb %prob% test
```

```
##         classA    classB
## [1,] 0.4815380 0.5184620
## [2,] 0.4192209 0.5807791
## [3,] 0.6882270 0.3117730
## [4,] 0.4794415 0.5205585
## [5,] 0.5209152 0.4790848
```

```
### Helper functions

# Obtain first table
tables(nb, 1)
```

```
##
## ------------------------------------------------------------------------------
##  ::: bern (Bernoulli)
## ------------------------------------------------------------------------------
##
## bern    classA    classB
##    A 0.4400000 0.4888889
##    B 0.5600000 0.5111111
##
## ------------------------------------------------------------------------------
```

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

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

Fit the Naive Bayes model based on 10 simulated predictors, each having 10mn observations, and then perform classification:

```
vars <- 10
rows <- 1000000
y <- sample(c("a", "b"), rows, TRUE)

# Discrete features
X1 <- as.data.frame(matrix(sample(letters[5:9], vars * rows, TRUE),
                           ncol = vars))
nb_cat <- naive_bayes(x = X1, y = y)
system.time(pred2 <- predict(nb_cat, X1))
```

```
##    user  system elapsed
##   0.357   0.103   0.466
```

7 Appendix

7.1 Practical examples: parameter estimation

This is a practical subsection that is aimed mostly to the students who learn the Naive Bayes model for the first time and are interested in the technical aspects of the model fitting.

7.1.1 Categorical distribution

In this example, the famous `iris` dataset is appended with a random categorical feature "new" with 3 levels/categories and then the parameters are estimated using the Maximum-Likelihood as well as the Bayesian estimation via adding pseudo-counts.

```
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 artificial features with three levels/categories:
# level1 is very unlikely and has 0.5% chance to occur
# level2 and level3 happen with probability 75% and 29.5%, respectively

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"

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

# For this reason level1 is also not available in any class sub-sample
table(iris2$new[iris$Species == Ck])

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

```r
# ML-estimates
ck_sub_sample <- table(iris2$new[iris$Species == Ck])
ck_mle_estim <-  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 # Jeffreys  prior / Dirichlet
               # with the concentration parameter 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_estim, ck_bayes)

# Bayesian estimate for level1 has 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_estim,
      "nb_mle" = tables(nb_mle, which = "new")[[1]][ ,Ck])

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

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.

```
data(iris)
Xi <- "Petal.Width" # i-th feature
Ck <- "versicolor"  # k-th class

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

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

# MLE in 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 is used to estimate class conditional densities for one variable from the `iris` dataset.

```
# Prepare data: ------------------------------------------------------

data(iris)
Xi <- "Sepal.Width" # i-th feature
C1 <- "setosa"      # 1st class
C2 <- "virginica"   # 2nd class
C3 <- "versicolor"  # 3rd class

# Build class sub-samples for the i-th 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 i-th feature
dens1 <- density(C1_Xi_subsample)
dens2 <- density(C2_Xi_subsample)
dens3 <- density(C3_Xi_subsample)

# Visualisation: -----------------------------------------------------
plot(dens2, main = "", col = "red")
lines(dens1, main = "", col = "blue")
```

```
lines(dens3, main = "", col = "black")
legend("topleft", legend = c(C1,C2,C3),
       col = c("blue", "red", "black"),
       lty = 1)

# 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]]


# 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

In this example, parameter estimation for the class conditional Poisson features is demonstrated.

```
# Simulate data: -----------------------------------------------------
cols <- 2
rows <- 10
set.seed(11)
M <- matrix(rpois(rows * cols, lambda = c(0.1,1)), nrow = rows,
```

```
            ncol = cols)
y <- factor(sample(paste0("class", LETTERS[1:2]), rows, TRUE))
colnames(M) <- paste0("Var", seq_len(ncol(M)))

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)

# 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)
```