Naiwny Klasyfikator Bayesa

Autor: Jan Retkowski

Importowanie bibliotek

```
library(tidyr)
library(dplyr)
library(ggplot2)
```

Wczytywanie danych

```
wine <- read.csv("wine.data")
colnames(wine) <- c(
    "wine_type",
    "alcohol",
      "malic_acid",
    "ash",
      "alcalinity_of_ash",
      "magnesium",
    "total_phenols",
      "flavanoids",
      "nonflavanoid_phenols",
      "proanthocyanins",
      "color_intensity",
      "hue",
      "OD280_OD315_of_diluted_wines",
      "proline"
)

wine <- wine %>%
    mutate(wine_type = as.factor(wine_type))
```

Implementacja Naiwnego Klasyfikatora Bayesa

Naiwny klasyfikator bayesowski oblicza prawdopodobieństwo wszystkich klas y w zależności od wektora predyktorów $\bar{x} = (x_1, x_2, ..., x_n)$.
Prawdopodobieństwo to wyraża się wzorem

$$P(y|x_1, x_2, ..., x_n) = \frac{P(x_1, x_2, ..., x_n|y)P(y)}{P(x_1, x_2, ..., x_n)}$$

Naiwnie zakładając, że predyktory są niezależne od siebie

$$P(x_i|y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|y)$$

możliwe jest przejście do postaci

$$P(y|x_1, x_2, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, x_2, ..., x_n)}$$

Ponieważ $P(x_1, x_2, ..., x_n)$ jest stałe dla danych zmiennych wejściowych, a dzielenie przez stałą nie wpływa na relatywne wartości prawdopodobieństw dla każdej z klas, to możliwe jest pominięcie tego elementu.

Reguła decyzyjna na podstawie której następuje klasyfikacja wyraża się wzorem:

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^{n} P(x_i|y)$$

By zachować lepszą precyzję obliczeń dla małych prawdopodobieństw wzór można przekształcić na postać logarytmiczną:

$$\hat{y} = \underset{y}{\operatorname{argmax}} \log(P(y)) + \sum_{i=1}^{n} \log(P(x_i|y))$$

Dla zmiennych kategorycznych $P(x_i|y) = \frac{N_{yi}+\alpha}{N_y+\alpha n}$, gdzie

N_{yi}, to liczba występowania danej wartości zmiennej kategorycznej,
N_y, to liczba występowania wszystkich wartości zmiennej kategorycznej,
n, to liczba unikalnych wartości zmiennej kategorycznej, a
$0 \leq \alpha \leq 1$, to hiperparametr wygładzenia laplace'a / lidtstone'a zapobiegający występowaniu wartośći prawdopodobieństw równych 0.

Dla zmiennych numerycznych do wyliczenia $P(x_i|y)$ stosuje się rozkład normalny:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \cdot \exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2})$$

```
create_gaussian_distr <- function(mean, sd) {
    force(mean)
    force(sd)

    function(x) dnorm(x, mean = mean, sd = sd)
}


naive_bayes.fit <- function(data, target, features, laplace_smoothing = 1) {

    NBC.model <- list()
    labels <- unique(data[, target])
```

```
    pior_prob <- numeric()
    cond_prob <- list()

    for (label in labels) {

        cond_prob[[label]] <- list()
        data_filt <- data[data[,target] == label, ]

        pior_prob[label] <- nrow(data_filt) / nrow(data)

        for (feature in features) {

            values <- data_filt[, feature]
            if (is.factor(values)) {
                cond_prob[[label]][[feature]] <- (
                    (summary(values) + laplace_smoothing)
                    / (length(values) + laplace_smoothing * length(unique(values)))
                )
            } else {
                mean_val <- mean(values)
                cond_prob[[label]][[feature]] <- create_gaussian_distr(mean(values), sd(values))
            }
        }
    }

    NBC.model$features <- features
    NBC.model$labels <- labels
    NBC.model$target <- target
    NBC.model$pior_prob <- pior_prob
    NBC.model$cond_prob <- cond_prob

    return(NBC.model)
}


naive_bayes.predict.single <- function(data, NBC.model) {

    probs <- numeric()

    for (label in NBC.model$labels) {

        probs[label] <- log(NBC.model$pior_prob[label])

        for (feature in NBC.model$features) {
            val <- data[, feature]

            if (is.factor(val)) {

                probs[label] <- probs[label] + log(NBC.model$cond_prob[[label]][[feature]][val])
            } else {

                probs[label] <- probs[label] + log(NBC.model$cond_prob[[label]][[feature]](val))
            }
```

```
        }
    }
    predicted <- NBC.model$labels[which.max(probs)]
    return(predicted)
}


naive_bayes.predict <- function(data, NBC.model) {

    mapply(function(i) naive_bayes.predict.single(data[i,], NBC.model), 1:nrow(data))
}
```

Testowanie algorytmu

Definiowanie funkcji służących walidacji krzyżowej, oceny celności modelu oraz doboru najlepszych par predyktorów

```
measure_accuracy <- function(data, NBC.model) {
    acc <- sum(naive_bayes.predict(data, NBC.model) == data[,NBC.model$target]) / nrow(data)

    return(acc)
}


cross_validate <- function(k, data, index_groups, target, features) {
    sum_acc <- 0
    for (i in 1:k) {
        model <- naive_bayes.fit(data[-index_groups[[i]], ], target, features)

        sum_acc <- sum_acc + measure_accuracy(data[index_groups[[i]], ], model)
    }

    return(sum_acc / k)
}


compare_feature_pairs <- function(k, data, target, features) {
    feature_combs <- combn(features, 2, simplify = F)
    best_features <- feature_combs[[1]]

    index_groups <- split(
        sample(nrow(data), nrow(data), replace = FALSE)[seq_len((nrow(data) %/% k) * k)],
        as.factor(1:k)
      )
    best_acc <- cross_validate(k, data, index_groups, target, feature_combs[[1]])

    for (feature_comb in feature_combs[-1]) {
        acc <- cross_validate(k, data, index_groups, target, feature_comb)

        if (acc > best_acc) {
            best_acc <- acc
            best_features <- feature_comb
        }
```

```
    }

    return(list(best_features = best_features, accuracy = best_acc))
}
```

W tym przypadku do testowania i wybrania najlpszej pary predyktorów zostanie użyta **4-krotna walidacja krzyżowa**.

```
set.seed(1)

k = 4
(best_model_data <- compare_feature_pairs(k, wine, colnames(wine)[1],
                                          features = colnames(wine)[-1]))
```

```
## $best_features
## [1] "alcohol"    "flavanoids"
##
## $accuracy
## [1] 0.9147727
```

W tym wypadku najlepszą parą okazały sie `alcohol` i `flavanoids` z ze średnią celnością 0.91

Teraz możliwe jest wytrenowanie modelu korzystającego z tej pary predyktorów

```
set.seed(1)
train_size <- 0.75
train_ind <- sample(nrow(wine), round(nrow(wine) * train_size), replace = F)
data_train <- wine[train_ind,]
data_test <- wine[-train_ind,]

model <- naive_bayes.fit(data = data_train, target = "wine_type",
                         features = best_model_data$best_features)

measure_accuracy(data_test, model)
```

```
## [1] 0.9090909
```

Jak widać celność właściwego modelu korzystającego z tych zmiennych osiąga 0.91