Метод поиска ближайшего соседа

**Package ‘kknn’ - Весовой** метод ближайших соседей.

**Использование:**

kknn(formula = formula(train), train, test, na.action = na.omit(),

k = 7, distance = 2, kernel = "optimal", ykernel = NULL, scale=TRUE,

contrasts = c('unordered' = "contr.dummy", ordered = "contr.ordinal"))

kknn.dist(learn, valid, k = 10, distance = 2)

**Аргументы:**

**formula** – формула

**train** – матрица или фрейм примеров обучающей выборки

**test** – матрица или фрейм примеров тестирующей выборки.

**na.action** – функция, которая указывает, что должно произойти, когда есть пропущенные данные ’NA’

**k** – число ближайших соседей

**distance** – параметр расстояния Минковского

**kernel** – ядра: "rectangular" – стандартный невесовой knn; "triangular", "epanechnikov", "biweight", "triweight", "cos", "inv", "gaussian", "rank" и "optimal"

**ykernel** – ширина окна y-ядра

**scale** – логическая переменная, TRUE, когда переменные масштабируются, чтобы иметь равные дисперсии

**Выходное значение**:

kknn возвращает объект-список, включающий элементы:

**fitted.values** – вектор прогнозов

**CL** – матрица классов k ближайших соседей

**W** – матрица весов k ближайших соседей

**D** – матрица расстояний k ближайших соседей

**C** – матрица индексов k ближайших соседей

**prob** – матрица вероятностей прогнозируемых классов

**response** – тип переменных (непрерывный, номинальный или порядковый)

**distance** – параметр расстояния Минковского

**Пример 1.**

**Обучающее множество: Iris**

**Признаки:**

1. длина чашелистика (sepal) в см

2. ширина чашелистика (sepal) в см

3. длина лепестка в см

4. ширина лепестка в см

**Классы:**

-- Iris Setosa

-- Iris Versicolour

-- Iris Virginica

library(kknn)

## загружаем данные Iris

data(iris)

## определяем общее количество примеров в обучающей выборке

m <- dim(iris)[1]

## посмотрим диаграмму рассеяния

pairs(iris[1:4], col = c("blue", "red", "black"))

## сгенерируем m/3 случайных целых чисел в диапазоне от 1 до m

val <- sample(1:m, size = round(m/3), replace = FALSE, prob = rep(1/m, m))

## формируем обучающую выборку

iris.learn <- iris[-val,]

## формируем выборку для тестирования

iris.valid <- iris[val,]

## запускаем процедуру kknn, используя

## расстояние Минковского с параметром 1 и треугольное ядро

iris.kknn <- kknn(Species~., iris.learn, iris.valid, distance = 1, kernel = "triangular")

## смотрим результаты

summary(iris.kknn)

fit <- fitted(iris.kknn)

## а какое это растение (5.8, 3.1, 4.8, 0.5) ?

example <- c(5.8, 3.1, 4.8, 0.5)

kknn(Species~., iris.learn, example, distance = 1, kernel = "triangular")

**Кросс-валидация k ближайших соседей**

**Использование:**

train.kknn(formula, data, kmax = 11, distance = 2, kernel = "optimal",

ykernel = NULL, scale = TRUE,

contrasts = c('unordered' = "contr.dummy", ordered = "contr.ordinal"), ...)

cv.kknn(formula, data, kcv = 10, ...)

**Аргументы:**

**kmax** – максимальное значение оптимального k

**kcv** – число разделений для k-fold cross validation

**Выходное значение**:

**best.parameters** – список, содержащий оптимальные параметры ядра и k.

**Пример 2.**

**Обучающее множество: Ionospere**

**Признаки:** 34 признака – данные радаров

**Классы**: "Good" – сигналы радаров, показывающие наличие некоторой структуры в ионосфере, "Bad" – сигнал проходит через ионосферу и ничего не показывает

## загружаем данные Ionospere

data(ionosphere)

## смотрим количество примеров и признаков в ionosphere

dim(ionosphere)

## смотрим заголовки признаков и классов

ionosphere[0,]

## смотрим метки классов всех примеров

ionosphere[,"class"]

## выбираем первые 200 примеров для обучения

ionosphere.learn <- ionosphere[1:200,]

## оставшиеся примеры используем для тестирования

ionosphere.valid <- ionosphere[-c(1:200),]

## запускаем процедуру kknn

fit.kknn <- kknn(class ~ ., ionosphere.learn, ionosphere.valid)

## запускаем процедуру train.kknn, используя

## расстояние Минковского с параметром 1 и с различными ядрами, максимальное K для поиска оптимального K равно 15

fit.train1 <- train.kknn(class ~ ., ionosphere.learn, kmax = 15,kernel = c("triangular", "rectangular", "epanechnikov", "optimal"), distance = 1)

fit.train2 <- train.kknn(class ~ ., ionosphere.learn, kmax = 15, kernel = c("triangular", "rectangular", "epanechnikov", "optimal"), distance = 2)

plot(fit.train1)

plot(fit.train2)

**Задание**

1. Исследуйте, как объем обучающей выборки и количество тестовых данных, влияет на точность классификации или на вероятность ошибочной классификации в примере крестики-нолики и примере о спаме e-mail сообщений.

2. Постройте классификатор для обучающего множества **Glass**, данные которого характеризуются 10-ю признаками:

1. Id number: 1 to 214; 2. RI: показатель преломления; 3. Na: сода (процент содержания в соотвествующем оксиде); 4. Mg; 5. Al; 6. Si; 7. K; 8. Ca; 9. Ba; 10. Fe.

Классы характеризуют тип стекла:

(1) окна зданий, плавильная обработка

(2) окна зданий, не плавильная обработка

(3) автомобильные окна, плавильная обработка

(4) автомобильные окна, не плавильная обработка (нет в базе)

(5) контейнеры

(6) посуда

(7) фары

Посмотрите заголовки признаков и классов. Перед построением классификатора необходимо также удалить первый признак Id number, который не несет никакой информационной нагрузки. Это выполняется командой **glass <- glass[,-1]**.

Постройте графики зависимости ошибки классификации от значения k и от типа ядра.

Исследуйте, как тип метрики расстояния (параметр **distance**) влияет на точность классификации.

Определите, к какому типу стекла относится экземпляр с характеристиками

RI =1.516 Na =11.7 Mg =1.01 Al =1.19 Si =72.59 K=0.43 Ca =11.44 Ba =0.02 Fe =0.1

Определите, какой из признаков оказывает наименьшее влияние на определение класса путем последовательного исключения каждого признака.

3. Для построения классификатора используйте заранее сгенерированные обучающие и тестовые выборки, хранящиеся в файлах svmdata4.txt, svmdata4test.txt. Найдите оптимальное значение k, обеспечивающее наименьшую ошибку классификации. Посмотрите, как выглядят данные на графике, используя функцию

**plot(mydata.train$X1, mydata.train$X2, pch=21, bg=c("red","blue") [unclass(mydata.train$Colors)], main="My train data")**