МАШИННОЕ ОБУЧЕНИЕ
ЛАБОРАТОРНЫЙ ПРАКТИКУМ
(предварительный вариант)

Н.Ю. Золотых, А.Н. Половинкин

19 октября 2007 г.

1. *Практическое машинное обучение*

1.1. Деревья решений

Здесь мы рассматриваем библиотеку `tree` Брайана Рипли.

1.1.1. Функция `tree`

Описание:

```
tree(formula, data, weights, subset,
    na.action = na.pass,
    control = tree.control(nobs, ...),
    method = "recursive.partition",
    split = c("deviance", "gini"),
    model = FALSE, x = FALSE, y = TRUE, wts = TRUE, ...)
```

Аргументы:

`formula` — формульное выражение. Левая часть формулы (ответ) должна быть либо числовым вектором (в случае, когда строится регрессионное дерево), либо вектором категорий (в случае, когда строится дерево классификации). Правая часть формулы должна быть набором либо числовых, либо категориальных переменных, разделенных знаком `+` (здесь не должно быть эффекта взаимодействия). В данной части формулы поддерживаются операции `'.'`, `'-'`: регрессионные деревья могут иметь offset terms.

`data` — набор данных, к которому применяется `formula`, `weights` и `subset`.

`weights` — вектор неотрицательных весов прецедентов(дробные веса допускаются).

`subset` — выражение, определяющее подмножество прецедентов, которые должны быть использованы.

`na.action` — функция, предназначенная для фильтрования отсутствующих данных в обучающей выборке. По умолчанию используется функция `na.pass` (не делать ничего). При этом отсутствующие значения обрабатываются в процессе построения дерева таким образом, чтобы поместить их в дерево как можно глубже.

`control` — список, возвращаемый функцией `tree.control`.

`method` — символьная строка, определяющая метод, который должен быть использован. Единственное другое полезное значение `'model.frame'`.

`split` — используемый критерий разбиения.

`model` — если данный аргумент является данными модели, тогда `formula` и `data` аргументы игнорируются; и `model` используется, чтобы определить модель. Если аргумент является логическим и равен TRUE, тогда данные модели сохраняются как компонент `model` в результате.

`x` — булева переменная. Если равна TRUE, то для каждого прецедента возвращается матрица переменных.

`y` — булева переменная. Если равна TRUE, то возвращается переменная-ответ.

`wts` — булева переменная. Если TRUE, то возвращаются веса.

`...` — дополнительные аргументы, передаваемые функции

`tree.control`. Обычно используется для `mincut`, `minsize` или `mindev`.

Детали: Дерево увеличивается путем бинарного рекурсивного разбиения, используя определенную формулу и выбирая разбиения из переменных в правой части формулы. Числовые переменные делятся на `X<a` и `X>a`; переменные, соответствующие неупорядоченным категориям делятся на две непустые группы. Разбиение, которое максимизирует уменьшение ошибки выбирается, набор данных разделяется и процесс повторяется. Разбиение повторяется до тех пор, пока терминальные узлы не становятся очень маленькими для разбиения. Глубина дерева ограничивается 31 уровнем (это обусловлено использованием целых чисел типа `integer`, чтобы помечать узлы дерева. Категориальные переменные могут иметь до 32 возможных значений. Это ограничение наложено для простоты их labbeling, но т.к. их использование в дереве классификации с тремя или больше уровнями в ответе требует поиска на $2^{k-1}-1$ группах для k уровней, практическое ограничение намного меньше.

Возвращаемое значение: Возвращаемое значение является объектом класса `tree`, который имеет следующие компоненты:

`frame` — набор данных, содержащий строки для каждого узла:

`row.names` дает номера узлов. Столбцы включают `var`, переменную, используемую для разбиения (или `<leaf>` для терминального узла), `n` — взвешенное число прецедентов, соответствующее данному узлу, `dev` — deviance данного узла, `yval`, подогнанное значение в узле (математическое ожидание для задач регрессионых деревьев, преобладающий класс для задач классификации) и `split` — матрица с 2 столбцами, содержащими метки левого и правого потомков для данного узла. Деревья классификации также имеют `yprob`, матрицу подогнанных вероятностей для каждого возможного значения ответа.

`where` — целочисленный вектор, определяющий, к какому из узлов относится соответствующий прецедент в обучающей выборке.

`terms` — переменные в формуле.

`call` — the matched call to Tree

`model` — если `model = TRUE`, данные модели.

`x` — если x = TRUE, матрица, соответствующая построенной модели.

`y` — если y = TRUE, ответ.

`wts` — если wts = TRUE, веса.

и аттрибуты `xlevels` и (для деревьев классификации) `ylevels`. Дерево без разбиений относится к классу `singlenode`, которое наследуется из класса `tree`.

1.1.2. Функция snip.tree

snip.tree содержит две связанные функции. Если аргумент nodes определен, то функция удаляет заданные узлы и всех их потомков из дерева.

Если аргумент nodes не определен, пользователю предлагается выбрать узлы вручную (на графике); это имеет смысл, если дерево уже нарисовано. Узел выбирается щелчком по левой кнопке мыши: при этом номер узла, deviance текущего дерева и то, что могло бы остаться, если данный узел удаляется, печатается. Если выбрать данный узел еще раз, то он будет удален (и линии, сооответствующие поддереву с корнем в данном узле, стираются). Щелчок по другим кнопкам мыши прекращает процесс выбора.

```
snip.tree(tree, nodes, xy.save = FALSE,
          digits = getOption("digits") - 3)
```

Аргументы:

tree — объект класса tree.

nodes — целочисленный вектор, определяющий узлы, являющиеся корнями поддеревьев, которые должны быть «отрезаны». Если данный аргумент не задан, то пользователю предлагается выбрать узел, в котором дерево «обрезается».

xy.save — если данный аргумент равен TRUE, то x и y координаты, выбранные интерактивно, сохраняются в объекте .xy в глобальном окружении.

digits — точность (количество цифр), используемое для вывода статистики для выбранных узлов.

Возвращаемое значение: Объект класса tree, содержащий узлы, которые остались после того, как заданные (или выбранные) поддеревья были «отрезаны».

Определяет nested последовательность поддеревьев заданного дерева путем рекурсивного удаления («обрезания») наименее важных разбиений.

1.1.3. Функция prune.tree

```
prune.tree(tree, k = NULL, best = NULL, newdata, nwts,
           method = c("deviance", "misclass"), loss, eps = 1e-3)

prune.misclass(tree, k = NULL, best = NULL, newdata,
               nwts, loss, eps = 1e-3)
```

Аргументы:

tree — обученная модель класса tree. Полагается, что она может быть результатом некоторой функции, которая создает объект с теми же компонентами, что и возвращаемая функцией tree().

k — параметр cost-complexity, определяющий или определенное поддерево заданного дерева (если k скаляр) или (дополнительно) последовательность поддеревьев, минимизирующих cost-complexity measure (если k вектор). Если данный параметр не задан, то он определяется алгоримически,

`best` — целое число, «предлагающее» размер (т.е. число терминальных узлов) заданного поддерева в cost-complexity поседовательности, которая должны быть возвращена. Это является альтернативным путем выбора поддерева, чем скалярный cost-complexity параметр `k`. Если в последовательности нет дерева требуемого размера, возвращается следующее большее по величине.

`newdata` — данные, на которых вычисляется последовательность cost-complexity поддеревьев. Если данный аргумент не задан, используются те же данные, что и при построении дерева.

`nwts` — веса прецедентов из `newdata`.

`method` — символьная строка, означающая функцию измерения неоднородности узлов, используемая для cost-complexity отсечения. Для регрессионных деревьев принимается только используемая по умолчанию deviance. Для деревьев классификации по умолчанию используется deviance (поддерживается также 'misclass' — число ошибок классификации или total loss).

`loss` — матрица потерь: определяет для каждого класса (соответствует строке) численное значение потерь при возможной ошибке, которая заключается в предсказании другого класса (соответствует столбцу). Обычно данная матрица должна иметь нулевую диагональ. По умолчанию используется матрица с нулями на диагонали и остальными элементами, равными `1`.

`eps` — нижняя граница для вероятностей, используемых для вычисления deviances, если

a lower bound for the probabilities, used to compute deviances if events of predicted probability zero occur in newdata.

Детали: Определяет вложенную последовательность поддеревьев заданного дерева путем рекурсивного удаления («отсечения») наименее важных разбиений, основываясь на измерении cost-complexity. `prune.misclass` — это обозначение для `prune.tree(method = "misclass")` для использования с `cv.tree`.

Если параметр `k` определен, возвращается оптимальное поддерево для данного значения.

Ответ, так же как и предикторы, на которые ссылаются в правой части формулы в дереве, должны быть представлены именем в `newdata`. Эти данные

These data are dropped down each tree in the cost-complexity sequence and deviances or losses calculated by comparing the supplied response to the prediction.

Функция `cv.tree()` вспомогательно использует аргумент `newdata` в перекрестной проверке процедуры pruning. Метод `plot` существует для объектов этого класса. Он показывает значение deviance, число ошибок классификации или total loss для каждого поддерева в cost-complexity последовательности. На дополнительной оси отображаются значения параметр cost-complexity для каждого поддерева.

Возвращаемое значение: Если параметр `k` определен и является скаляром, возвращается объект минимизирующий cost-complexity measure для этого `k`. Если параметр `best` определен, возвращается объект класса `tree` размера `best`. В противном случае возвращается объект класса `tree.sequnce`. Объект содержит следующие компоненты:

`size` — число терминальных узлов в каждом дереве cost-complexity pruning

последовательности.

`deviance` — суммарная deviance каждого дерева the cost-complexity pruning последовательности.

`k` — значение cost-complexity pruning параметра каждого дерева в последовательности.

1.1.4. Пример

Загрузим пакет, содержащий функции для работы с деревьями решений

```
> library(tree)
```

Построим дерево классификации для данных `iris`:

```
> ir.tr <- tree(Species ~., iris)
```

Само дерево можно нарисовать с помощью функции `plot`:

```
> plot(ir.tr, type = "uniform")
> text(ir.tr)
```

Но более красивые изображения дает функция `draw.tree` из библиотеки `maptree`:

```
> library(maptree)
> draw.tree(ir.tr)
```

Результат приведен на рис. 1.1. Выведем информацию о построенном дереве:

```
> ir.tr
node), split, n, deviance, yval, (yprob)
   * denotes terminal node

 1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
   2) Petal.Length < 2.45 50   0.000 setosa ( 1.00000 0.00000 0.00000 ) *
   3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
     6) Petal.Width < 1.75 54  33.320 versicolor ( 0.00000 0.90741 0.09259 )
      12) Petal.Length < 4.95 48   9.721 versicolor ( 0.00000 0.97917 0.02083 )
        24) Sepal.Length < 5.15 5   5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
        25) Sepal.Length > 5.15 43   0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
      13) Petal.Length > 4.95 6   7.638 virginica ( 0.00000 0.33333 0.66667 ) *
     7) Petal.Width > 1.75 46   9.635 virginica ( 0.00000 0.02174 0.97826 )
      14) Petal.Length < 4.95 6   5.407 virginica ( 0.00000 0.16667 0.83333 ) *
      15) Petal.Length > 4.95 40   0.000 virginica ( 0.00000 0.00000 1.00000 ) *
```

«Обрежем» построенное дерево в узле с номером 12, результат поместим в `ir.tr1`:

```
> ir.tr1 <- snip.tree(ir.tr, nodes = 12)
> draw.tree(ir.tr1)
```

Результат см. на рис. 1.2. Запустим процедуру построения «отсечений» в дереве `ir.tr` со значением параметра `k`, равным 10; результат поместим в `ir.tr2`:

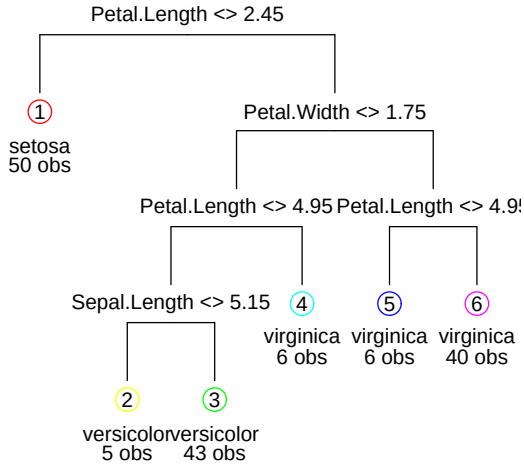

Рис. 1.1. Дерево решений для задачи классификации цветов ириса

```
> ir.tr2 <- prune.tree(ir.tr, k = 10)
> draw.tree(ir.tr2)
```

Результат см. на рис. 1.3.

Построим разбиение пространства предикторов, соответствующее построенному дереву `ir.tr1`:

```
> par(pty = "s")
> plot(iris[, 3], iris[, 4], type = "n",
    xlab = "X1 petal length", ylab = "X2 petal width")
> text(iris[, 3], iris[, 4], c("s", "c", "v")[iris[, 5]],
    col=c("blue", "red", "black")[iris[, 5]])
> partition.tree(ir.tr1, add = TRUE)
```

1.1.5. Задания

1) Загрузите набор данных `monica` из пакета `DAAG` (`http://www.stats.uwo.ca/DAAG/DAAG_0.95.zip`). Постройте дерево классификации для модели, задаваемой следующей формулой: `outcome ~ .`, дайте интерпретацию полученным результатам. Является ли построенное дерево избыточным? Если да, то выполните операцию «snip off» над соответствующими узлами.

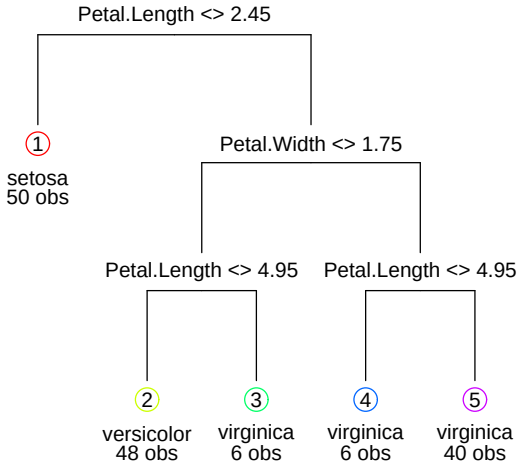

Рис. 1.2. Дерево решений для задачи классификации цветов ириса с обрезанной фершиной 12

2) Загрузите набор данных `spam7` из пакета `DAAG`. Постройте дерево классификации для модели, задаваемой следующей формулой: `yesno ~.`, дайте интерпретацию полученным результатам. Запустите процедуру «cost-complexity prunning» с выбором параметра `k` по умолчанию, `method = 'misclass'`, выведите полученную последовательность деревьев. Какое из полученных деревьев, на Ваш взгляд, является оптимальным? Объясните свой выбор.

3) Загрузите набор данных `nsw74psid1` из пакета `DAAG`. Постройте регрессионное дерево для модели, задаваемой следующей формулой: `re78 ~.`. Постройте регресионную модель и SVM-регрессию для данной формулы. Сравните качество построенных моделей, выберите оптимальную и объясните свой выбор.

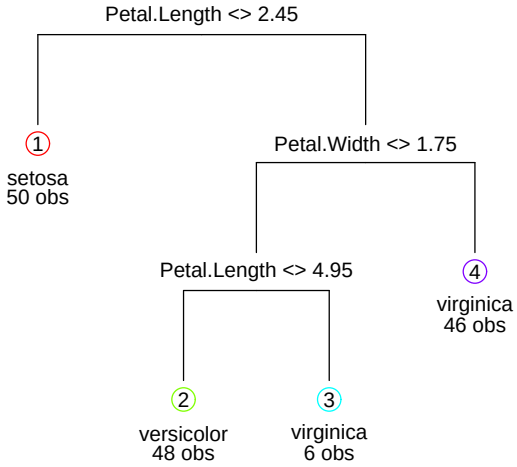

Рис. 1.3. Дерево решений для задачи классификации после проведения процедуры отсечения со значением параметра $k = 10$