Занятие седьмое. О кластеризации, эвристических и иерархических алгоритмах, о функционалах качества разбиения и силуэтах, нечеткой кластеризации, а еще лесах и деревьях

Зачем нужна кластеризация?

Зачастую, рассматривая статистическую задачу, мы сталкиваемся с тем, что данные существенно неоднородны. Например, проводя социологический опрос, мы столкнемся с существенно неоднородными данными по разным слоям населения. В таких случаях нам интересно выделить эти группы и проанализировать каждую из них по отдельности. Задача разбиения и является задачей кластеризации.

Удобно ввести метрику d в пространстве наблюдений и смотреть на кластеризацию в разрезе расстояний $d_{i,j}$ между наблюдаемыми точками. Матрица из расстояний $d_{i,j}$ достаточна для кластеризации, поэтому большинство алгоритмов позволяют использовать в качестве входных данных эту матрицу.

При этом в зависимости от наших целей мы можем искать более округлые кластеры (то есть следить за средним внутриклассовым расстоянием) или более вытянутые кластеры (то есть для каждого элемента искать кластер, расстояние до ближайшего элемента которого будет как можно меньше).

Чаще всего данные стоит предварительно нормировать, чтобы привести оси к одному масштабу.

Нам понадобится библиотека cluster

Эвристические методы

Эвристические методы строят разбиение на заданное число k кластеров. Рассмотрим некоторые наиболее популярные:

1) **Метод k-средних (k-means).**

Этот метод основан на минимизации функционала

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2,$$

где μ_i — центр i-ого класса. На выходе такой алгоритм выдает округлые кластеры или кластеры с центром.

Базовый алгоритм носит название Lloyd, он происходит по следующему алгоритму:

1) Выполняется некоторое разбиение на кластеры

2) Вычисляются центры кластеров

3) Каждая точка относится к тому кластеру, в котором расстояние до центра минимально.

4) Пересчитываются центры кластеров и происходит переразбиение на кластеры. Так продолжается до тех пор, пока процесс не стабилизируется или число итераций не превысит некоторого числа.

Более быстрый алгоритм Mac-Queen пересчитывает центры кластеров после причисления к кластеру каждой новой точки на шаге 3).

Алгоритм Hartingan, Wong схож с алгоритмом Mac-Queen, но более сложен и более быстр.

Реализовать в R такую кластеризацию можно с помощью функции $kmeans(Data, clust)$, где $Data$ — матрица с кластеризуемыми точками или матрица расстояний между классами, $clust$ — либо число k, либо вектор длины k с начальными центрами кластеров. По умолчанию кластеризация проводится по алгоритму Hartigan-Wong, хотя указывая $algorithm =$ "$Lloyd$", "$Forgy$" или "$Mac-Queen$" мы можем сменить алгоритм на другой.

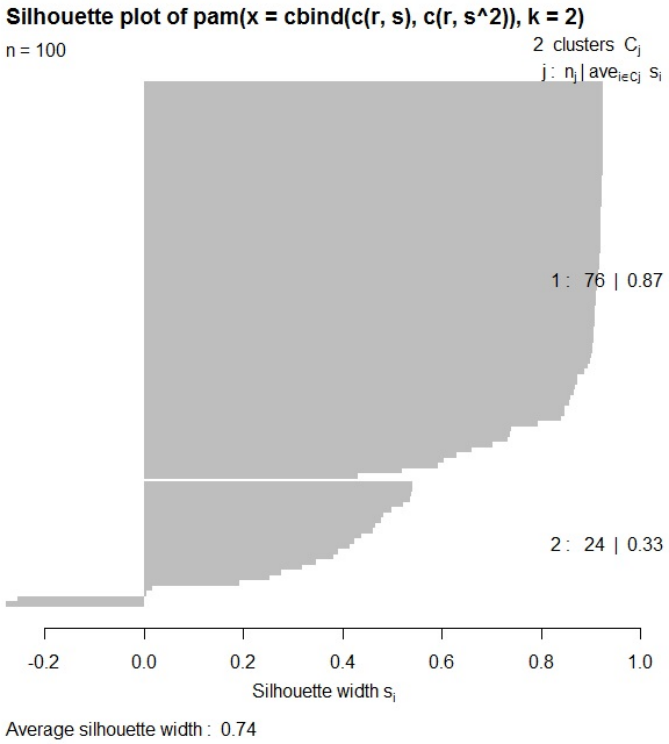
Алгоритм выдает центры кластеров, вектор разбиения на кластеры (вектор из номеров кластеров всех элементов), а также информацию о расстояниях внутри кластеров и между кластерами.

Упражнение 1. Сгенерировать выборки $\mathcal{N}((0,0),E), \mathcal{N}((5,0),E), \mathcal{N}((6,5),2E)$ по 50 элементов и кластеризовать их объединение методом k-средних при $k=3$, $k=2$, $k=4$. Построить график с помощью функции clusplot, аргументами которой являются матрица данных и вектор разбиения на кластеры.

2) Метод k-медоидов или partitions around medoids.

Этот метод является слегка усложненным методом k-средних. Основных отличий два. Во-первых, берется не центроид, а медоид — вместо центров кластеров, которые могли и не являться элементами выборки, рассматривается один из элементов. Во-вторых, здесь нет такой привязки к евклидовой метрике.

В R этот вид кластеризации реализован в функции pam.
Отметим, что при построении графика на основе данного метода, мы можем увидеть также график силуэтов (silhouette plot).
Силуэт элемента i кластера S строится следующим образом. Вычисляются $a(i)$ — среднее расстояние



от i до остальных элементов кластера, $d(i, S')$ — среднее расстояние от i до элементов кластера S'. Вычисляется $b(i) — \min d(i, S')$ по всем $S' \neq S$. Силуэтом называется $s(i) = 1 - a(i)/b(i)$ при $a(i) \leq b(i)$ или $s(i) = b(i)/a(i) - 1$ при $a(i) > b(i)$. Для одноэлементного кластера силуэт считается равным 0. График силуэтов представляет собой график $s(i)$ отсортированных по возрастанию внутри каждого кластера. Близкое к 1 $s(i)$ означает, что выбранный кластер сильно превосходит любой другой, близкое к 0 — что данный элемент примерно равноправно мог быть отнесен в несколько кластеров, отрицательное — что он вполне вероятно классифицирован неверно.
В приведенной картинке мы видим, что первый кластер построен достаточно хорошо, а во втором есть не очень удачно классифицированные точки.
Еще два эвристических метода мы рассмотрим чуть позже.

Иерархические методы

Другая категория методов кластеризации называется иерархическими. Идея в том, чтобы не фиксировать количество кластеров, а выстроить иерархию объединений, на основе который мы легко сможем построить разбиение на любое количество кластеров. В свою очередь, иерархические методы делятся на разделяющие (divisive) и объединяющие (agglomerative). Первые методы, стартуя с разбиения на один кластер, находят разбиение на два кластера, затем делят один из кластеров на две части и так далее до N кластеров по одному элементу. Вторые начинают с разбиения на N кластеров и объединяют элементы до тех пор, пока не получится один кластер.
Оба метода на выходе выдают дерево иерархии разбиения на кластеры. Объединяющие методы более удобны в тех случаях, когда мы хотим исследовать свойства отдельных элементов, разделяющие — если нам интересны большие массивные кластеры. При этом вычислительно второй класс методов значительно более сложен, поэтому он редко применяется для больших наборов данных.

Объединяющие методы на каждом шаге объединяют два ближайших друг к другу кластера, где определение "близости" зависит от формы желаемых кластеров. Рассмотрим несколько классических метрик:
1) Метод ближнего соседа ("single linkage") предлагает $d(S_1, S_2) = \min_{x \in S_1, y \in S_2} d(x, y)$. При этом на каж-

дом шаге мы объединяем два кластера, имеющих два наиболее близких друг к другу элементов. Кластеры получаются более вытянутые.
2) Метод дальнего соседа ("complete linkage") предлагает $d(S_1, S_2) = \max_{x \in S_1, y \in S_2} d(x, y)$. При этом на каждом шаге мы объединяем те два кластера, у которых наиболее удаленные элементы (среди элементов этих кластеров) наиболее близки (среди всех пар кластеров). Кластеры получаются небольшими и шарообразными, зачастую плохо отделимыми друг от друга.
3) Метод средней связи ("average linking" или UPGMA) предлагает $d(S_1, S_2)$ — среднее арифметическое расстояний между всеми парами точек из разных кластеров. Кластеры выходят округлыми и возможно существенно разными по размеру.
4) Метод Уорда предлагает $d(S_1, S_2) = n_1 n_2 |\mu_1 - \mu_2|^2/(n_1 + n_2)$, где μ_1, μ_2 — средние арифметические кластеров, n_1, n_2 — их размеры.
В R объединяющие методы содержатся в функции agnes. Наиболее полезным для нас (в случае шаровидных кластеров) будет метод средней связи.

Неудобством разделяющих методов является то, что изначально мы вынуждены перебрать все возможные разбиения на кластеры (C_n^2) штук, затем все разбиения, где один кластер тот же, а второй поделен на две части и так далее. Это приводит к высокой вычислительной сложности. Однако, можно обойти такого рода сложности, отказавшись от перебора разбиений.

Идея метода, предложенного МакНаутанном и Смитом, заключается в том, что на первом шаге мы удаляем из общего кластера элемент с наибольшим средним расстоянием до оставшихся. (назовем его плохим). Затем для каждого элемента кластера мы сравниваем средние расстояния до плохого кластера и до оставшихся элементов исходного. Те, для которых расстояние до плохого меньше, мы также называем плохими и переносим во второй кластер. Так мы продолжаем до тех пор, пока не получим разбиение на два кластера. Затем мы выбираем кластер с большим диаметром (наибольшим расстоянием внутри кластера) и делим таким же образом его и так далее.
В R этот алгоритм носит название DIANA и реализован функцией diana().

Как быть, если все черно-белое?

Если данные представляют собой бинарные векторы (т.е. принимают только два возможных значения 0 и 1), то есть более удобный алгоритм, чем описанные выше. Мы рассмотрим иерархический разделяющий метод MONA. Идея метода в том, чтобы на каждом шаге выбрать одну из переменных и разбить один из кластеров на те, у кого эта переменная равна 0 и те, у кого эта переменная равна 1. Метод разбиения заключаются с следующем. Для любых двух компонент вектора i, j мы рассматриваем следующую меру взаимосвязанности $D(i, j)$— абсолютное значение величины

$$N(a_i = 0, a_j = 1)N(a_i = 1, a_j = 0) - N(a_i = 0, a_j = 0)N(a_i = 1, a_j = 1),$$

где N — число наблюдений a, удовлетворяющих указанному свойству.
Если, скажем, у большинства наблюдений значения компонент i и j одинаковые, то первая величина будет малой, а вторая большой. Аналогично если i и j будут противоположными, то первая величина будет большой, а вторая малой. В обоих случаях мы получи большое $D(i, j)$. Если же i, j независимые признаки, то D будет близка к нулю. Алгоритм MONA (Monothetic Analysis, где monothetic означает, что шаг кластеризации основан на значениях только одной переменной) предлагает каждый раз выбирать переменную, имеющую наибольшую сумму $D(i, j)$ с остальными переменными кластера и делить в соответствии с ее значениями. Тогда отличие ее значений будет означать, что соответствующие данные отличаются по многим переменным.

Упражнение 2. Подсчитать $D(1, 2)$, $D(1, 3)$ для данных,

$$\begin{array}{lllllll} X_1 = & 1 & 1 & 0 & 1 & 1 & 0, \\ X_2 = & 1 & 1 & 0 & 0 & 0 & 1, \\ X_3 = & 1 & 1 & 1 & 1 & 1 & 0, \\ X_4 = & 1 & 1 & 1 & 1 & 1 & 0, \\ X_5 = & 0 & 0 & 0 & 1 & 0 & 1, \\ X_6 = & 0 & 0 & 0 & 0 & 0 & 0, \\ X_7 = & 0 & 0 & 1 & 1 & 0 & 1, \\ X_8 = & 0 & 0 & 1 & 1 & 1 & 0. \end{array}$$

Кластеризовать данные.

В R этот алгоритм запрограммирован функцией mona.

А причем здесь вероятность?

Остается вопрос — зачем эта тема на статистическом практикуме? Ведь задачу мы рассматриваем вполне детерминированную.
Во-первых, зачастую мы кластеризуем данные, полученные вследствие смешивания данных из нескольких распределений. Для их разделения нам потребуется кластеризация. Многие алгоритмы при этом также имеют вероятностный смысл, например, метод k-средних по сути базируется на максимизации правдоподобия для многомерных нормальных данных.
Во-вторых, мы имеем дело с задачами оптимизации, а значит, как мы знаем из прошлого семинара, нам могут пригодиться методы Монте-Карло. Алгоритм CLARA, предложенный ниже, на основе вероятностного подхода позволяет кластеризировать большие массивы данных.
В-третьих, алгоритм FANNY, который мы рассмотрим в этом разделе, позволит нам кластеризовать вероятностно — приписать элементам вероятности, с которыми они могут быть отнесены в тот или иной кластер.

Начнем с того случая, когда размерность данных велика. В таком случае методы навроде k-медоидов не слишком эффективны, поскольку требуют большого количества операций. С другой стороны, медоиды кластеров при больших объемах данных совсем необязательно определять на основе всей совокупности наблюдений. Именно так и работает Clara — выбирает выборку, ищет по ней медоиды и делит всю совокупность на кластеры по близости к этим медоидам. Такая процедура повторяется несколько раз, а затем выбирается результат с наименьшим суммарным разбросом кластеров.

В R этот алгоритм реализован функцией clara(x=Data,k=number,samples = n, sampsize = N), где *Data* — наши данные или матрица расстояний, *number* — число кластеров, n — количество взятых выборок, N — их размеры.

Еще один важный подход —- нечеткая кластеризация (fuzzy clustering). Это аналог эвристических методов, позволяющий решить проблему нечеткости отнесения отдельных элементов в кластер. Нечеткая кластеризация сопоставляет каждой точке k чисел, где k — число кластеров, представляющих собой как бы вероятность отнесения в соответствующий кластер.

Наиболее простой алгоритм такого плана — алгоритм c-means, в котором минимизация функционала

$$\sum_{l=1}^{k} \sum_{x_j \in S_l} d(x_j, \mu_l)$$

в алгоритме k-медоидов при этом заменяется на минимизацию

$$\sum_{l=1}^{k} \sum_{i=1}^{n} u_{i,l} d(x_i, c_l)$$

по $u_{i,l}$, т.ч. $u_{i,l} \geq 0$, $\sum_{l=1}^{k} u_{i,l} = 1$ при любом i. $u_{i,l}$ и будут выполнять роль наших вероятностей, а минимизируемыq функционал при этом представляет собой математическое ожидание суммарного расстояния от точек до кластеров, если считать, что распределение i точки в кластер осуществляется с вероятностями $u_{i,l}$, $l = 1...k$. Мы будем использовать алгоритм FANNY с целевой функцией

$$\sum_{l=1}^{k} \frac{\sum_{i,j=1}^{n} u_{i,l}^2 u_{j,l}^2 d(x_i, x_j)}{2 \sum_{j=1}^{n} u_{j,k}^2}.$$

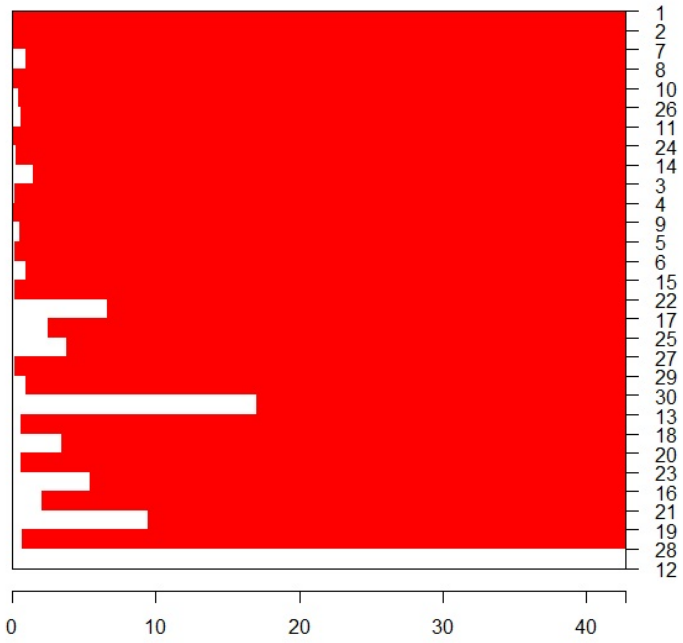
Реализация в R этого алгоритма сделана функцией fanny(x=Data, k =number), где *Data* — наши данные или матрица расстояний, *number* — число кластеров.

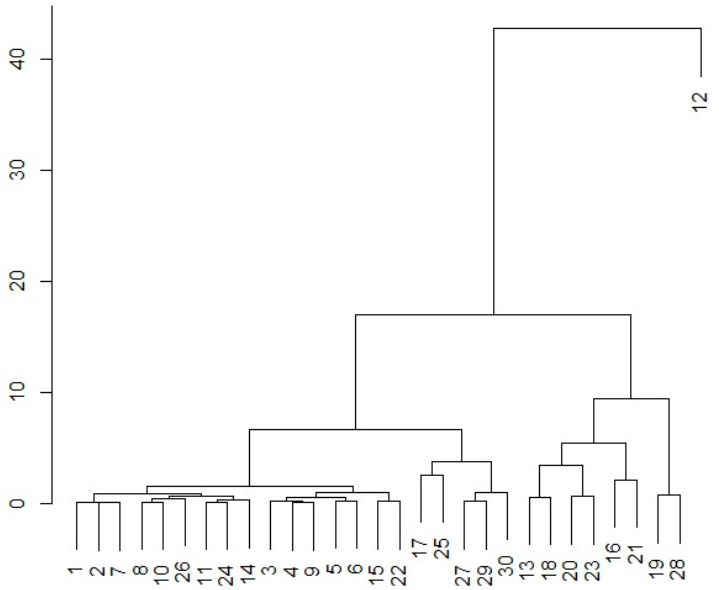
Радуем глаз или визуализация

После проведения кластеризации нам бы хотелось посмотреть на ее наблюдения графически. Для эвристических методов и двумерных данных мы можем посмотреть на наблюдения разбитые на классы графически. В случае трехмерных данных мы можем построить трехмерный график с помощью scatterplot3d из библиотеки scatterplot3d.

В случае многомерных данных возможно несколько вариантов. Один из вариантов — построить матрицу из графиков, соответствующих проекциям на плоскости, образованные каждой парой координат. Например, для данных представляющих собой случайную выборку из объединения единичных шаров с центрами (0,0,0) и (0,0,2) оно выглядит так
Другим вариантом является нелинейное шкалирование Сэммона. Метод Сэммона позволяет отыскать такое облако точек меньшей размерности, что матрица расстояний этого облака максимально похожа на матрицу расстояний исходного облака. В R это задается функцией sammon пакета MASS. Задается она sammon(dist,k), где dist — матрица расстояний, k — размерность желанного облака точек. График этого облака можно построить с помощью функции plot. Для той же выборки, что и в прошлом случае, оно выглядит так.

Для эвристических методов также можно посмотреть на график силуэтов, описанный выше. Получить его можно как clusplot, так и обычной функцией plot, куда подставили результаты кластеризации. В случае иерархических методов мы не выбираем число кластеров и потому нам нужны другие варианты визуализации —баннеры и дендрограммы, характеризующие последовательность разбиений.





Баннер выглядит подобно тому, как это изображено на рисунке. Если я хочу посмотреть на разбиение на 2 кластера, то мне нужно найти самую короткую красную полосу, все что лежит ниже нее отнести в один кластер, а остальное в другой. В случае 3 кластеров аналогичное разделение устроят две наиболее коротких полосы и так далее.
Дендрограмма выглядит иначе, но дает ту же информацию. Она показывает в каком порядке мы должны объединять элементы в кластеры. Чем ниже находится общий родитель двух элементов, тем раньше они объединятся.

В случае, изображенном на рисунке, мы видим одноточечный кластер из элемента 12 (на графике силуэтов соответствующая линия целиком белая), если мы хотим делить дальше, то самая маленькая линия баннера проведена между 13 и 30.

Некоторые замечания

Бдительный читатель мог заметить, что разобранные нами функции носят женские имена: pam, clara, fanny, daisy, diana и т.д. Эту романтическую линию создали Leonard Kaufman и Peter Rousseeuw, авторы

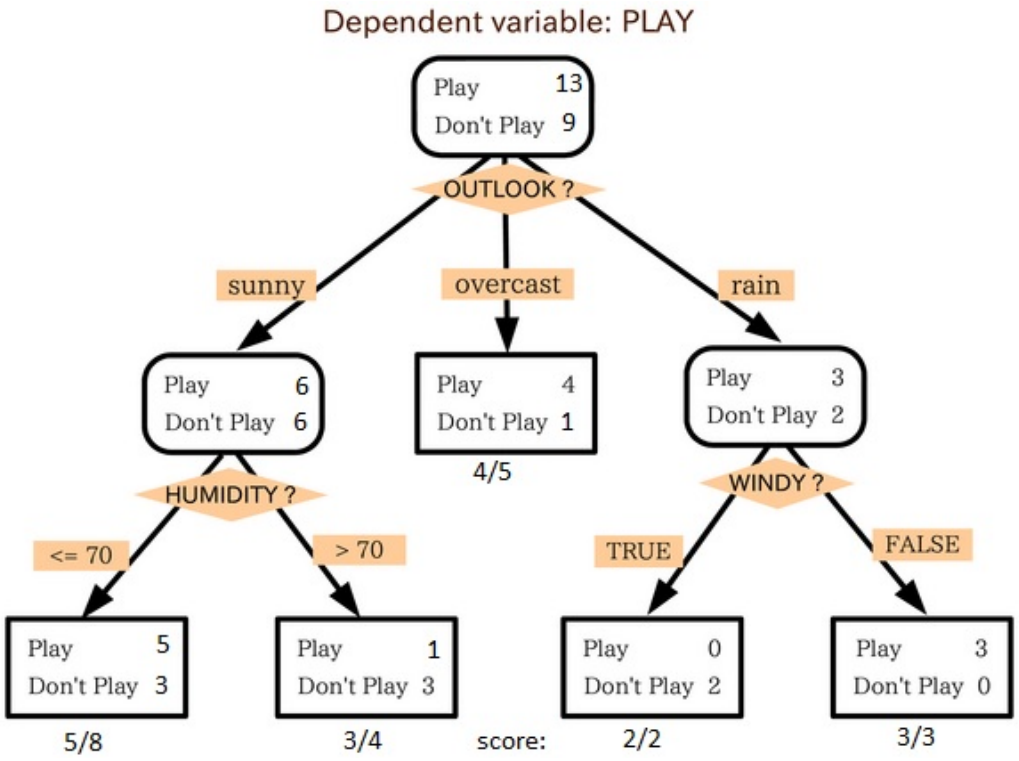
книги "Finding Groups in Data", которая легла в основу библиотеки clusters и данного материала.

Скажем также немного о проблемах с весом. В некоторых случаях мы сталкиваемся с необходимостью минимизировать не просто суммы расстояний от центра кластера до точек, но и учитывать при этом некий весовой коэффициент для каждой точки. К примеру, выстраивая сеть электростанций для снабжения региона энергией, мы должны будем минимизировать не суммарное расстояние от населенных пунктов до станций, а суммарное перемещение энергии, т.е. постараемся строить более короткие промежутки для более крупных пунктов. В таком случае вместо величины $\sum_{i=1}^{n} d(x_i, \mu_{K(i)})$, где μ — центры кластеров, $K(i)$ — кластер i-ой точки, как это было в методе k-медоидов, мы используем величину $\sum_{i=1}^{k} w_i d(x_i, \mu_{K(i)})$, где w_i — заданные веса. В этом случае нам понадобится библиотека WeightedClusters и функция wcKMedoids(diss=DissMatr,k=number, weights = w), где DissMatr — матрица расстояний, number — число кластеров, а w — вектор весов.

Решающие леса и случайные деревья

Посмотрим также на задачу классификации. Пусть у нас есть многомерные данные, один из параметров A которых мы хотели бы прогнозировать по остальным. Допустим у нас есть обучающая выборка, в которой все параметры известны и решающая выборка, в которой параметр A неизвестен.

В этом случае мы хотим научить аппроксимировать значения A функцией от остальных параметров как можно точнее. Эта задача достаточно похожа на задачу регрессии.
Один из алгоритмов подхода к такой задаче является построение дерева принятия решений, то есть иерархической структуры, классифицирующей данные. Это похоже на тест — вы отвечаете на некоторые вопросы о значениях параметров, спускаясь по дереву, а на последнем уровне вас ждет ответ — чему при таких значениях остальных параметрах на обучающей выборке равен параметр A.



Чтобы дерево получилось информативным, нужно стараться на каждом уровне выбирать тот параметр, который наиболее значимо влияет на значения параметра A. Одним из естественных функционалов для измерения такого показателя является энтропия: если из n элементов множества C n_i — число элементов с i-ым значением параметра A, то энтропия задается формулой

$$H(C, A) = -\sum_i \frac{n_i}{n} \ln \frac{n_i}{n}.$$

Назовем приростом информации при классификации множества A на основе параметра Q с i значени-

ями, разбивающими A на A_i, величину

$$G(C,Q) = H(C,A) - \sum_{i=1}^{q} \frac{|A_i|}{|A|} H(C,A_i).$$

На каждом шаге предлагается выбирать тот параметр Q, который дает максимальный прирост информации. Соответствующий алгоритм называется ID3.

Этот алгоритм склонен сначала выбирать те параметры, у которых большое количество значений, поэтому зачастую используют его модификацию — метод C4.5, использующий статистику Gini, снижающую этот эффект.
В R этот алгоритм реализован в функции rpart пакета rpart. Интерфейс этой функции имеет вид $rpart(formula = A \sim B_1 + B_2 + B_3,\ data = train,\ method = "class")$, где A — оцениваемый столбец матрицы data, а B_1, B_2, B_3 — те столбцы, которые мы хотим использовать для оценки. Для случая непрерывного параметра A можно использовать $method = anova$.

Получив дерево $tree$, мы можем предсказать для данных $test$ значение $predict(tree, test, type = "class")$.

При использовании дерева мы можем столкнуться с переобучением — частности, вызванные случайностью, дерево будет принимать за общие закономерности. В связи с этим нет большого смысла обращать внимания на все уровне дерева, значимы лишь первые из них. Это может быть достигнуто с помощью параметра control функции rpart, который можно задать как результат rpart.control(cp = , minsplit =), где cp останавливает разбиение, если прирост информации (изменение статистики Gini) меньше заданной константы, а minsplit — если разбиение на классы будет содержать меньше minsplit наблюдений. / Визуализацию полученного дерева можно делать с помощью обычной функции plot, но более изящно выглядит при использовании fancyRpartPlot пакета rattle.

Также уменьшить переобуславливание можно с помощью случайных лесов. Из исходной тестовой выборки выбираются подвыборки с возвращением и по ним на основе случайно выбранной части признаков строятся деревья принятия решений. После этого классификация производится на основе всего ансамбля деревьев. Для реализации этого метода используется функция $randomForest(A \sim B_1 + B_2 + B_3, data = train, importance = TRUE, ntree = 1000)$, где ntree — количество взятых деревьев, importance — параметр, отвечающий за необходимость расчета важности переменных. Прогнозирование переменных осуществляется после этого все тем же predict.
Если importance = TRUE, то столбец importance полученного в результате randomForest массива данных будет содержать прирост информации и изменения статстики Гини для каждого параметра, чем больше эти величины, тем влиятельнее величина. Удобный механизм визуализации этой картины представляет собой varImpPlot.
Упражнение 3. В http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv, http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv расположены данные о пассажирах Титаника. train содержит данные включая столбец Survived — выжил ли пассажир, а для test требуется предсказать значение Survived. Построить дерево принятия решений, исследовать какие параметры влияют на Survived (исключите очевидно незначимые, например, имя пассажира). Построить случайный лес для определения выживших пассажиров. Для этого вам понадобится заполнить пробелы в наблюдениях Age, что можно сделать с помощью решающего дерева. Какие параметры оказались наиболее значимыми для Survived и почему?

Метод решающих деревьев и случайных лесов достаточно просты, не требуют сложного анализа модели и работают в достаточно широком спектре ситуаций (например, он работает и в дискретном и в непрерывном случае), но склонны к переобучению.