Обучение

ИЦ «Гевисста»

Артем Груздев

Прогнозное моделирование в R и Python

Модуль 4. Введение в метод случайного леса

Москва — 2017

СОДЕРЖАНИЕ

Модуль 4 Введение в метод случайного леса

Лекция 4.1. Описание метода случайного леса

Как уже отмечалось, основным недостатком деревьев решений является их склонность к переобучению и нестабильность результатов, когда небольшие изменения в наборе данных могут приводить к построению совершенно другого дерева (особенно это актуально для метода CART). Случайный лес стал одним из способов решения этой проблемы. По сути случайный лес – это набор деревьев решений, где каждое дерево немного отличается от остальных. Идея случайного леса заключается в том, что каждое дерево может довольно хорошо прогнозировать, но скорее всего переобучается на определенной части данных. Если мы построим много деревьев, которые хорошо работают и переобучаются с разной степенью, мы можем уменьшить переобучение путем усреднения их результатов.

Для реализации вышеизложенной стратегии нам нужно построить большое количество деревьев решений, то есть ансамбль деревьев. Каждое дерево должно на приемлемом уровне прогнозировать зависимую переменную и должно отличаться от других деревьев (условие декоррелированности деревьев). Для этого в процесс построения деревьев вносим случайность, которая призвана обеспечить уникальность каждого дерева (отсюда случайный лес и получил свое название). Для получения рандомизированных деревьев в случайном лесе последовательно применяются две техники: сначала случайным образом отбираем наблюдения, которые будут использоваться для построения дерева, а затем для каждого узла дерева осуществляем случайный отбор фиксированного количества предикторов для поиска наилучшего расщепления.

Чтобы построить случайный лес, сначала необходимо определиться с количеством деревьев. Допустим, мы хотим построить ансамбль из 5 деревьев. Для построения каждого дерева мы сначала сформируем *бутстреп-выборку* (*bootstrap sample*) наших данных. То есть из набора данных объемом n наблюдений мы случайным образом выбираем наблюдение с возвращением n раз (поскольку отбор с возвращением, то одно и то же наблюдение может быть выбрано несколько раз). Мы получаем выборку, которая имеет такой же размер, что и исходный набор данных, однако некоторые наблюдения будут отсутствовать в нем (примерно 37% наблюдений исходного набора), а некоторые попадут в него несколько раз.

Чтобы проиллюстрировать это, предположим, что мы хотим создать бутстреп-выборку для списка из 10 наблюдений ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']. Возможная бутстреп-выборка может выглядеть как ['10', '9', '7', '8', '1', '3', '9', '10', '10', '7']. Другой возможной бутстреп-выборкой

может быть ['4', '8', '5', '8', '3', '9', '2', '6', '1', '6']. В нашем случае нам нужно построить 5 деревьев, поэтому будет сформировано 5 бутстреп-выборок. Наглядно механизм бутстрепа показан на рисунке 4.1.

Рис. 4.1 Механизм работы бутстрепа

На основе каждой сформированной бутстреп-выборки строится полное бинарное дерево решений[1], то есть разбиения узлов будут продолжаться до тех пор, пока не будет достигнуто минимальное количество наблюдений в терминальных узлах (изначально в качестве минимального количества наблюдений в терминальном узле Лео Брейман, один из авторов случайного леса, предложил для дерева классификации брать значение 1, а для дерева регрессии – значение 5). Однако алгоритм, который был описан для полного бинарного дерева решений (смотрите главу 3), теперь немного изменен. Вместо поиска наилучшей точки расщепления по каждому предиктору, алгоритм для разбиения каждого узла случайным образом отбирает фиксированное подмножество предикторов и затем находит наилучшую точку расщепления среди наилучших точек, найденных по каждому из случайно отобранных предикторов. Для выбора наилучшей точки разбиения используется уже знакомый вам критерий уменьшения неоднородности (для количественной зависимой переменной

[1] В оригинальном подходе Лео Бреймана и в большинстве пакетов используется дерево решений CART, однако существуют реализации, где в качестве деревьев ансамбля используются деревья QUEST, деревья C4.5.

используется среднеквадратичная ошибка, а для категориальной зависимой переменной – мера Джини). Отбор подмножества предикторов повторяется отдельно для каждого узла, поэтому в каждом узле дерева может быть принято решение с использованием «своего» подмножества предикторов.

Необходимо отметить, что идея случайного отбора определенного количества предикторов в каждом узле дерева появилась не сразу. Сначала Лео Брейман в 1996 году предложил метод бэггинга или бутстреп-агрегирования, когда на основе исходного набора данных мы генерируем бутстреп-выборки, по ним строим полные бинарные деревья и затем агрегирум их результаты путем голосования или простого усреднения. Бэггинг стал предшественником случайного леса. Примерно в это же время Тин Кам Хо предложил идею ансамбля деревьев, в рамках которого для построения каждого дерева используется фиксированное количество случайно отобранных признаков. Томас Диттерих в 2000 году предложил улучшить бэггинг дополнительной рандомизацией. Его подход заключался в том, чтобы для каждого узла определять 20 наилучших вариантов разбиений и случайным образом выбирать один вариант. Используя моделирование на синтетических и реальных наборах данных, он доказал, что дополнительная рандомизация улучшает качество работы бэггинга.

Использование бутстрепа приводит к тому, что деревья решений в случайном лесе строятся на немного отличающихся между собой бутстреп-выборках. Из-за случайного отбора переменных в каждом узле все расщепления в деревьях будут основаны на отличающихся подмножествах предикторов. Вместе эти два механизма приводят к тому, что все деревья в случайном лесе будут отличаться друг от друга.

Решая задачу классификации, каждое дерево сначала вычисляет для наблюдения листовые вероятности классов зависимой переменной. Листовая вероятность класса – это доля объектов класса в листе (терминальном узле) дерева, в который попало классифицируемое наблюдение. Каждое дерево голосует за класс с наибольшей вероятностью в листе. В итоге побеждает класс, за который проголосовало большинство деревьев. Итоговыми вероятностями классов для наблюдения будут доли голосов деревьев, поданных за данный класс.

Данный подход реализован в оригинальном программном коде Лео Бреймана и Адель Катлер, на базе которого написан пакет R `randomForest`. Для каждого дерева фиксируется только «победивший класс», листовые вероятности классов, полученные для отдельного дерева, отбрасываются и в дальнейших расчетах не участвуют. Поэтому в пакете `randomForest` итоговые вероятности классов для наблюдения – это доли голосов деревьев, поданных за данный класс.

№	фактический класс	спрогнозированный класс (итог голосования деревьев, построенных по всем бутстреп-выборкам)	итоговые вероятности классов (доли голосов деревьев, поданных за соответствующие классы)
1	Остается	Уходит	0.3 0.7
2	Уходит	Уходит	0.2 0.8
3	Уходит	Уходит	0.4 0.6

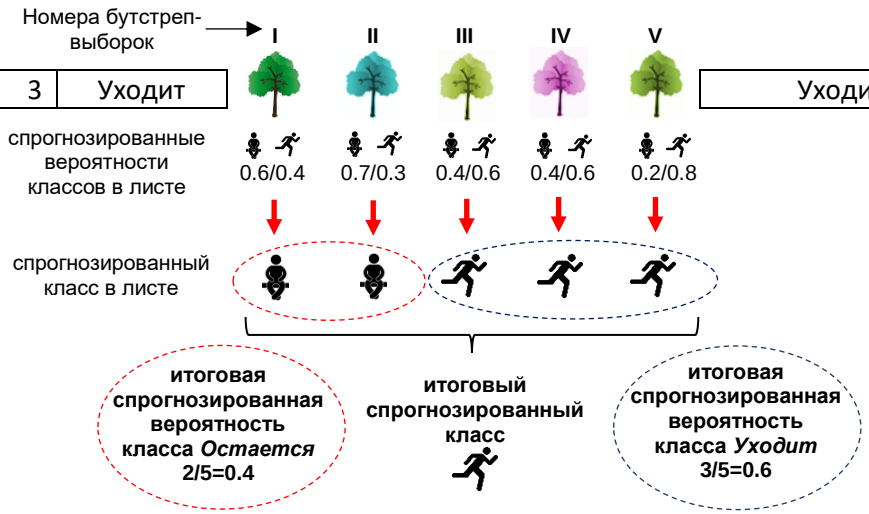
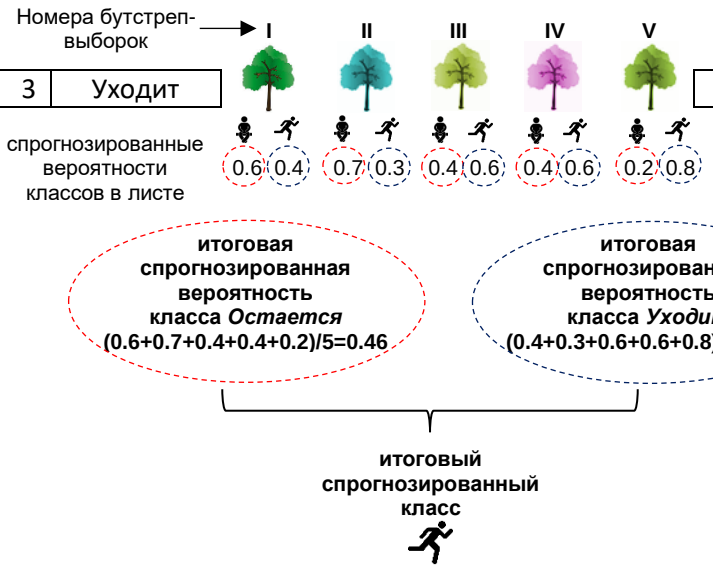


Рис. 4.2 Получение прогнозов для задачи классификации (оригинальный подход «усреднение прогнозов», реализован в пакете R randomForest)

В питоновской библиотеке `scikit-learn` для моделей `RandomForestClassifier` и `RandomForestRegressor` используется другой подход. Каждое дерево вычисляет для наблюдения листовые вероятности классов. Эти листовые вероятности усредняются по всем деревьям и в итоге прогнозируется класс с наибольшей усредненной листовой вероятностью. Поэтому для классов `RandomForestClassifier` и `RandomForestRegressor` итоговыми вероятностями классов будут листовые вероятности классов, усредненные по всем деревьям.

№	фактический класс
1	Остается
2	Уходит

спрогнозированный класс (итог голосования деревьев, построенных по всем бутстреп-выборкам)	итоговые вероятности классов (листовые вероятности классов, усредненные по всем деревьям)
Остается	0.54 0.46
Уходит	0.42 0.58



3	Уходит	Уходит	0.46 0.54

Рис. 4.3 Получение прогнозов для задачи классификации (подход «усреднение листовых вероятностей», реализован в классе RandomForestClassifier питоновской библиотеки scikit-learn)

Решая задачу регрессии, каждое дерево прогнозирует для наблюдения среднее значение зависимой переменной в листе (терминальном узле), в который это наблюдение попало, и в результате происходит усреднение полученных средних значений по всем деревьям.

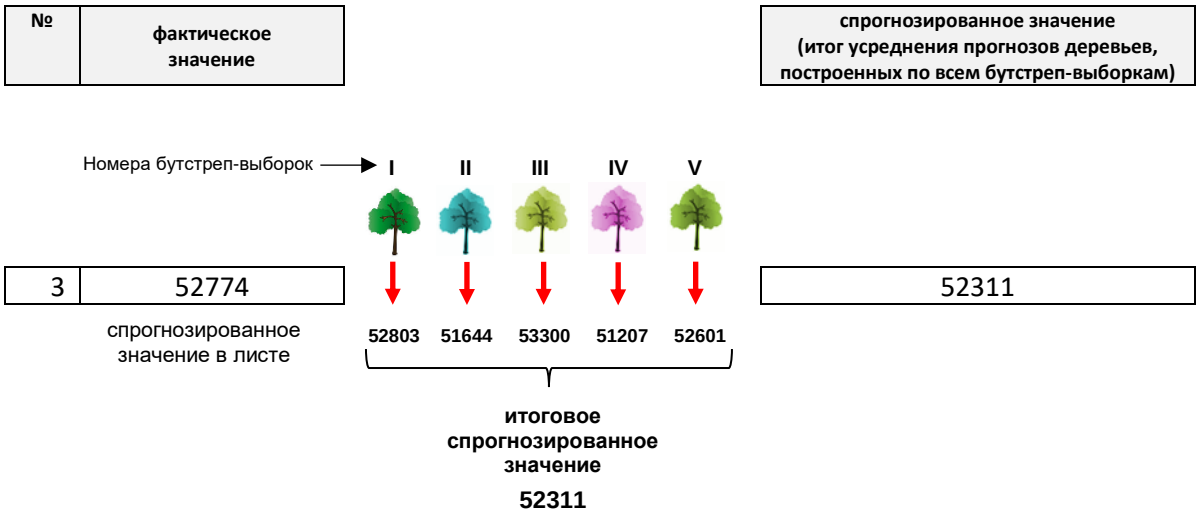


Рис. 4.4 Получение прогнозов для задачи регрессии

В итоге математически алгоритм случайного леса можно описать следующим образом:

1. Для $b = 1, 2... B$ (где B – количество деревьев в ансамбле):
   (a) извлечь бутстреп-выборку S размера N из обучающих данных;
   (b) по бутстреп-выборке S построить полное дерево T_b, рекурсивно повторяя следующие шаги для каждого терминального узла, пока не будет достигнуто минимальное количество наблюдений в нем (для классификации – одно наблюдение, для регрессии – 5 наблюдений):
      i. из первоначального набора M предикторов случайно выбрать m предикторов;
      ii. из m предикторов выбрать предиктор, который обеспечивает наилучшее расщепление;
      iii. расщепить узел на два узла-потомка.
2. В результате получаем ансамбль деревьев решений $\{T_b\}_{b=1}^{B}$
3. Предсказание новых наблюдений осуществлять следующим образом:

для регрессии: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$;

для классификации: пусть $\hat{C}_b(x)$ - класс, спрогнозированный деревом решений T_b, то есть $T_b(x) = \hat{C}_b(x)$; тогда $\hat{C}_{rf}^{B}(x)$ - это класс, наиболее часто встречающийся в множестве $\{\hat{C}_b(x)\}_{b=1}^{B}$

Попытки улучшить качество случайного леса предпринимаются вплоть до настоящего времени.

В 2006 году Пьер Герц, Дэмиен Эрнст и Луи Вехенкель представили научному сообществу метод полностью рандомизированных деревьев (Extremely Randomized Trees). Как и случайный лес, этот метод строит ансамбль полных деревьев классификации и регрессии. Однако у него есть два отличия от случайного леса. Теперь для разбиения каждого узла мы случайным образом отбираем m предикторов, для каждого случайно отобранного предиктора случайным образом получаем расщепляющие значения и в итоге выбираем для разбиения наилучшее расщепляющее значение. Кроме того, для построения каждого дерева используется вся обучающая выборка, а не бутстреп-выборка. Как и в случайном лесе, прогнозы деревьев агрегируются для получения итогового прогноза, для задачи классификации используется голосование и побеждает класс, набравший большинство голосов, для задачи регрессии происходит усреднение результатов.

Метод полностью рандомизированных деревьев был реализован в пакете R `extraTrees`, а также в классах `ExtraTreesClassifier` и `ExtraTreesClassifier` питоновской библиотеки `scikit-learn`.

Для достижения лучшего качества случайного леса вместо одномерных расщеплений Лео Брейман рекомендовал использовать многомерные расщепления, когда наилучшее разбиение узла определяется не отдельным признаком, а линейными комбинациями признаков. Такую комбинацию можно найти с помощью обычной модели машинного обучения (гребневой регрессии, метода частичных наименьших квадратов, метода опорных векторов и др.). Идея получила развитие в работах Бьерна Менце и Микаэля Кельма, предложивших в 2011 году метод oblique random forest или косоугольный случайный лес, сам метод был реализован в пакете R `obliqueRF`.

Кроме того, для улучшения случайного леса Лео Брейман предлагал при построении ансамбля варьировать размер терминального узла. Идея была развита Хемантом Ишвараном и Джеймсом Мэлли в 2014 году в рамках метода synthetic random forests или синтетических случайных лесов, а сам метод реализован в пакете R `rfsrcSyn`.

Лекция 4.2. Оценка качества модели

Качество модели случайного леса может быть оценено обычным способом и с помощью метода OOB.

В рамках обычного способа мы берем каждое наблюдение и используем для прогноза деревья, построенные по всем бутстреп-выборкам. Для задачи классификации применяем голосование, подсчитываем количество неправильно классифицированных наблюдений

(спрогнозированный класс для наблюдения – это класс, за который проголосовало большинство деревьев, построенных по всем бутстреп-выборкам). Затем делим количество неправильно классифицированных наблюдений на общее количество наблюдений и получаем ошибку классфикации. Для задачи регрессии подсчитываем сумму квадратов остатков – разностей между фактическим и спрогнозированным значением зависимой переменной (спрогнозированное значение для наблюдения – это результат усреднения средних значений, которые вычислили деревья, построенные по всем бутстреп-выборкам). Затем эту сумму делим на количество всех наблюдений и получаем среднеквадратичную ошибку.

В рамках метода OOB мы берем каждое наблюдение и используем для прогноза только те деревья, которые строились по бутстреп-выборкам, не содержащим данное наблюдение (т.е. наблюдение «выпало» из бутстреп-выборки и данную бутстреп-выборку для этого наблюдения можно назвать out-of-bag выборкой, отсюда и название метода). Для задачи классификации применяем голосование, подсчитываем количество неправильно классифицированных наблюдений (спрогнозированный класс для наблюдения – это класс, за который проголосовало большинство деревьев, построенных по out-of-bag выборкам). Затем делим количество неправильно классифицированных наблюдений на общее количество наблюдений и получаем ошибку классификации по методу OOB. Для задачи регрессии подсчитываем сумму квадратов остатков – разностей между фактическим и спрогнозированным значением зависимой переменной (спрогнозированное значение для наблюдения – это результат усреднения средних значений, которые вычислили деревья, построенные по out-of-bag выборкам). Затем эту сумму делим на количество всех наблюдений и получаем среднеквадратичную ошибку по методу OOB.

Метод OOB используется только для оценки качества модели на обучающей выборке. К каждому наблюдению контрольной выборки мы просто применяем правила, сформулированные каждым деревом леса в ходе обучения. Затем осуществляем голосование или усреднение и на основе полученных прогнозов обычным способом вычисляем метрику качества.

Рассмотрим процесс оценки качества модели с помощью метода OOB наглядно (рис. 4.5). Допустим, у нас есть исходный набор из 10 наблюдений, на его основе мы сгенерировали 5 бутстреп-выборок. Для каждого наблюдения мы должны зафиксировать бутстреп-выборки, в которых оно отсутствует. Например, на рис. 4.5 видно, что наблюдение 4 отсутствует в бутстреп-выборках I, III, IV и V. Эти выборки будут для наблюдения 4 out-of-bag выборками. Для классификации или вынесения

прогноза по наблюдению 4 нас как раз будут интересовать голоса или прогнозы деревьев, построенных по этим четырем out-of-bag выборкам.

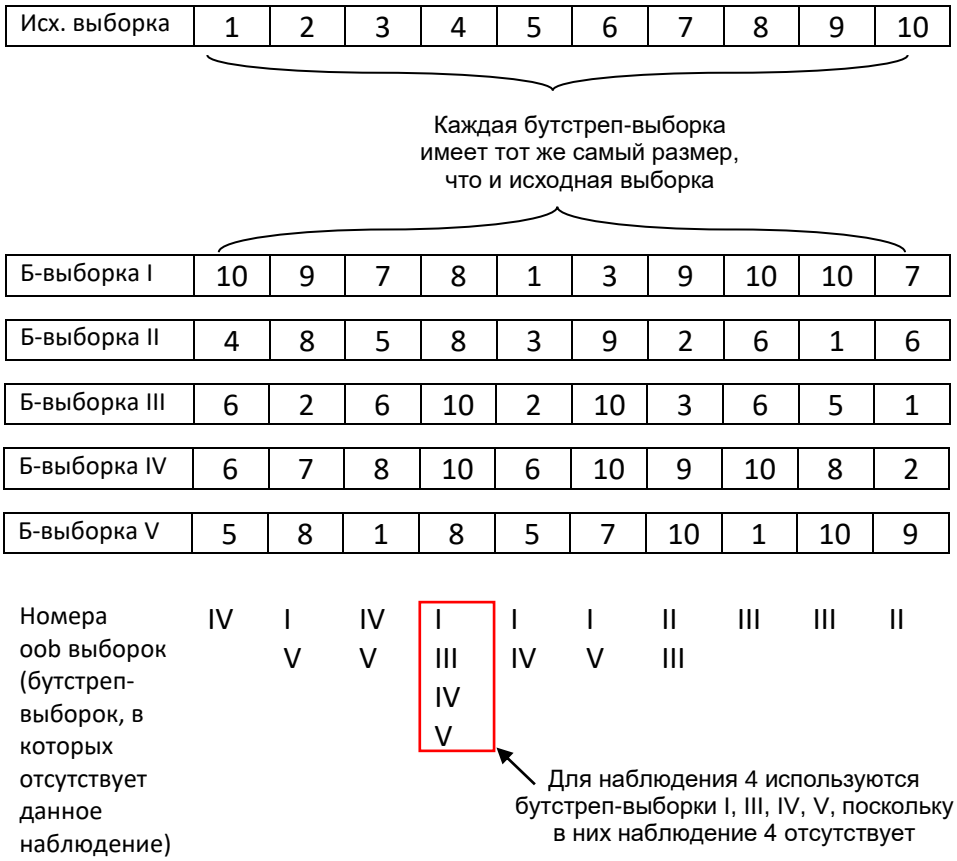


Рис. 4.5 Out-of-bag выборки для оценки качества модели

Возьмем задачу классификации (рис. 4.6). Допустим, необходимо отнести наблюдение 4 к тому или иному классу зависимой переменной *Статус клиента [status]*. Фактически оно принадлежит классу *Уходит*. Для оценки качества модели используются только те деревья решений, которые строились по бутстреп-выборкам, не содержащим наблюдение 4, и затем проводится голосование деревьев. Наблюдение 4, фактически принадлежащее классу *Уходит*, отсутствует в 4 бутстреп-выборках: I, III, IV, V. В голосовании участвуют 4 дерева, построенных по этим 4 out-of-bag выборкам. Мы предъявляем наше наблюдение каждому дереву, оно проверяет наблюдение на соответствие своим правилам классификации, вычисляет листовые вероятности классов и соответствующий класс. Например, деревья классифицировали наблюдения так: *Остается*, *Остается*, *Остается* и *Уходит*. В итоге побеждает класс *Остается*. Случайный лес ошибочно относит наблюдение 4 к классу *Остается*. В итоге мы подсчитываем количество таких неверно классифицированных наблюдений, делим на общее количество наблюдений и получаем ошибку классификации по методу OOB или OOB ошибку для классификации.

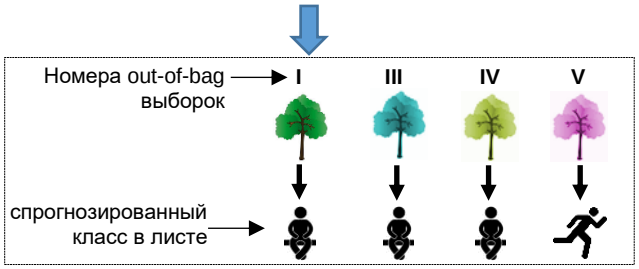
$$ER^{OOB} = \frac{1}{n}\sum_{i=1}^{n} 1\left(\hat{Y}^{OOB}(X_i) \neq Y_i\right)$$

где:

Y_i – фактический класс зависимой переменной;

$\hat{Y}^{OOB}(X_i)$ – спрогнозированный класс зависимой переменной (класс, за который проголосовало большинство деревьев, построенных по выборкам, не содержащим X_i)

№	номера out-of-bag выборок, участвующих в голосовании	фактический класс	спрогнозированный класс (итог голосования деревьев, построенных по out-bag выборкам)	результат классификации
1	IV	Остается	Остается	ВЕРНО
2	I, V	Уходит	Уходит	ВЕРНО
3	IV, V	Уходит	Уходит	ВЕРНО
4	I, III, IV, V	Уходит	Остается	НЕВЕРНО



5	I, IV	Остается	Остается	ВЕРНО
6	I, V	Уходит	Уходит	ВЕРНО
7	II, III	Уходит	Уходит	ВЕРНО
8	III	Остается	Уходит	НЕВЕРНО
9	III	Остается	Остается	ВЕРНО
10	II	Уходит	Уходит	ВЕРНО

количество неверных ответов=2

Ошибка классификации = количество неверно классифицированных наблюдений/общее количество наблюдений = 2/10=0,2

Рис. 4.6 Вычисление OOB ошибки для задачи классификации

Теперь возьмем задачу регрессии (рис. 4.7). Допустим, необходимо спрогнозировать значение зависимой переменной *Оценка дохода [income]* для наблюдения 4. Для оценки качества модели используются только те деревья решений, которые строились по бутстреп-выборкам, не содержащим наблюдение 4, и затем проводится усреднение прогнозов, выданных деревьями.

Наблюдение 4, имеющее фактическое значение *45304*, отсутствует в 4 бутстреп-выборках: I, III, IV, V. Таким образом, в усреднении участвуют 4 дерева, построенных по этим 4 бутстреп-выборкам. Мы предъявляем наше наблюдение каждому дереву, оно проверяет наблюдение на соответствие своим правилам прогнозирования, вычисляет среднее значение. Допустим, деревья прогнозируют следующие значения: *44470*, *45112*, *46790* и *47230*. На этот раз нас интересует квадрат остатка – разницы между фактическим значением зависимой переменной *45304* и ее спрогнозированным значением *45901* (результатом усреднения средних значений, вычисленных деревьями). В итоге по каждому наблюдению вычисляем квадрат остатка, суммируем и полученную сумму квадратов остатков делим на общее количество наблюдений. Сумма квадратов остатков, поделенная на общее количество наблюдений, становится оценкой качества случайного леса для регрессии. Ее еще называют среднеквадратичной ошибкой по методу OOB или OOB ошибкой для регрессии.

$$MSE^{OOB} = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}^{OOB}(X_i) - Y_i\right)^2$$

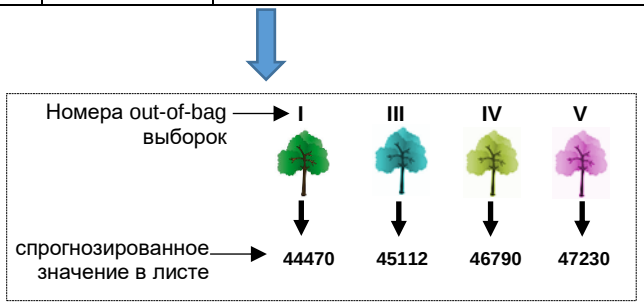
где:

Y_i – фактическое значение зависимой переменной;

$\hat{Y}^{OOB}(X_i)$ – спрогнозированное значение зависимой переменной (результат усреднения средних значений, которые вычислили деревья, построенные по выборкам, не содержащим X_i)

Наглядно процесс вычисления среднеквадратичной ошибки по методу OOB показан на рисунке 4.7.

№	номера out-of-bag выборок, участвующих в прогнозе	фактическое значение	спрогнозированное значение (результат усреднения прогнозов деревьев, построенных по out-bag выборкам)	квадрат остатка (факт. знач. – спрогн. знач.)²
1	IV	50451	50037	171396
2	I, V	52700	52127	328329
3	IV, V	32704	32028	456976
4	I, III, IV, V	45304	45901	356409



5	I, IV	29518	29067	203401
6	I, V	29508	29029	229441
7	II, III	24018	23501	267289
8	III	26369	26938	323761
9	III	26109	26905	633616
10	II	19369	19857	238144

сумма квадратов остатков=3208762
Среднеквадратичная ошибка = сумма квадратов остатков/общее количество наблюдений = 3208762/10=320876,2

Рис. 4.7 Вычисление OOB ошибки для задачи регрессии

На практике оценка OOB ошибки достоверна, когда количество деревьев в ансамбле достаточно велико. В ситуации, когда вы используете мало деревьев, то есть мало бутстреп-выборок, высока вероятность того, что наблюдение встретится во всех бутстреп-выборках (иными словами, ни разу не выпадет из бутстреп-выборок) и таким образом для него не будет получена OOB оценка. Необходимо настраивать качество модели, увеличивая количество деревьев до достаточно большого числа, пока OOB ошибка не перестанет уменьшаться. Обратите внимание, что вычисление OOB ошибки не заменяет собой проверку на контрольной выборке. Эксперименты показывают, что ошибка классификации и среднеквадратичная ошибка, вычисленные по методу OOB, являются более оптимистичными, чем ошибка классификации и среднеквадратичная ошибка, вычисленные на контрольной выборке или усредненные по контрольным блокам перекрестной проверки.

В различных статистических пакетах оценка качества случайного леса может быть вычислена либо с помощью обычного способа, либо с помощью OOB метода.

В пакете R `randomForest` функция `print` для ансамбля деревьев классификации выводит ошибку классификации по методу OOB, а для

ансамбля деревьев регрессии – среднеквадратичную ошибку и процент объясненной дисперсии, вычисленные по методу OOB. Функция `predict` позволяет спрогнозировать значения (классы) зависимой переменной и вероятности классов двумя способами. При наличии аргумента, задающего выборку, значения (классы) зависимой переменной и вероятности классов для заданной выборки прогнозируются обычным способом, при пропуске этого аргумента для наблюдений обучающей выборки будут возвращены значения (классы) зависимой переменной и вероятности классов, спрогнозированные по методу OOB. Если для параметра `predict.all` задать значение `TRUE`, можно получить листовые спрогнозированные значения (классы), т.е. прогнозы конкретных деревьев, полученные по обычному методу.
Давайте установим и загрузим пакет R `randomForest`. Пакет `randomForest` был разработан Энди Лиавом и Мэттью Винером и базируется на оригинальном программном коде Лео Бреймана и Адель Катлер, написанном на Fortran.

```
# устанавливаем пакет randomForest
# install.packages("randomForest")

# загружаем пакет randomForest
library(randomForest)
```

Начнем с задачи классификации. Сейчас мы вычислим ошибку классификации, спрогнозируем классы зависимой переменной и вероятности классов с помощью обоих методов.
Загрузим данные для задачи классификации и взглянем на них.

```
# загружаем данные
data <- read.csv2("C:/Trees/RF_classification.csv")

# смотрим данные
data
```

Сводка 4.1. Исходные данные для построения случайного леса деревьев классификации

	age	gender	default
1	70	Мужской	Да
2	64	Мужской	Нет
3	69	Женский	Да
4	68	Мужской	Нет
5	65	Женский	Нет

Строим модель случайного леса из 10 деревьев классификации и вычисляем ошибку классификации по методу OOB.

```
# задаем стартовое значение генератора случайных
# чисел для воспроизводимости результатов
set.seed(152)

# строим случайный лес из 10 деревьев классификации
model<-randomForest(default ~., data, ntree=10, importance=TRUE, norm.votes=FALSE)

# выводим информацию об ошибке классификации
# по методу OOB
print(model)
```

Сводка 4.2. Оценка качества модели (используется ошибка классификации по методу OOB)

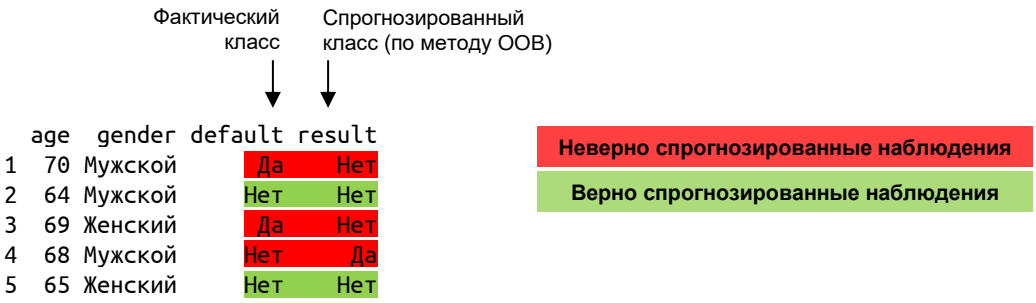
```
Call:
 randomForest(formula = default ~ ., data = data, ntree = 10,      importance = TRUE,
norm.votes = FALSE)
               Type of random forest: classification
                     Number of trees: 10
No. of variables tried at each split: 1

        OOB estimate of  error rate: 60%
Confusion matrix:
    Да Нет class.error
Да   0   2   1.0000000
Нет  1   2   0.3333333
```

Из сводки 4.2 видим, что для наших данных мы получили ошибку классификации по методу OOB, равную 60%. Теперь посмотрим на классы зависимой переменной, спрогнозированные по методу OOB. Для этого в функции `predict` опускаем аргумент, задающий выборку, а для параметра `type` задаем значение `"response"`. В пакете R `randomForest` класс, спрогнозированный для наблюдения по методу OOB, – это класс, за который проголосовало большинство деревьев, построенных по бутстреп-выборкам, где данное наблюдение отсутствовало (out-of-bag выборкам).

```
# выводим классы, спрогнозированные по методу OOB
oob_predictions <- predict(model, type="response")
oob_results <-data.frame(data, result=oob_predictions)
oob_results
```

Сводка 4.3. Фактические и спрогнозированные классы зависимой переменной, для прогнозирования используется метод OOB



	age	gender	default	result
1	70	Мужской	Да	Нет
2	64	Мужской	Нет	Нет
3	69	Женский	Да	Нет
4	68	Мужской	Нет	Да
5	65	Женский	Нет	Нет

Из сводки 4.3 видим, что из 5 наблюдений 3 неверно спрогнозированы, отсюда и получаем нашу ошибку классификации 3/5=0,6 или 60%.

Теперь выясним, как мы спрогнозировали итоговые классы по методу OOB. Для этого нам по каждому наблюдению нужно посмотреть, как распределились голоса деревьев, построенных по out-of-bag выборкам. Для параметра `type` вместо значения `"response"` задаем значение `"vote"`. Дополнительно воспользуемся параметром `norm.votes`, он определяет, в каком виде должны выводиться голоса деревьев для каждого класса – в виде исходных частот (значение `FALSE`) или процентных долей (значение `TRUE`). Обратите внимание, чтобы получить возможность выводить частоты голосов деревьев с помощью функции `predict`, при построении модели случайного леса нужно задать `norm.votes=TRUE`. Сначала выведем частоты голосов деревьев.

```
# смотрим частоты голосов деревьев для каждого класса
# (деревья построены по out-of-bag выборкам)
oob_vote_freq <- predict(model, type="vote", norm.votes=FALSE)
oob_vote_freq
```

Сводка 4.4. Частоты голосов деревьев, поданных за каждый класс (деревья построены по out-of-bag выборкам)

```
  Да Нет
1  1   2
2  0   5
3  0   3
4  1   0
5  0   4
attr(,"class")
[1] "matrix" "votes"
```

Возьмем наблюдение 1. Из сводки 4.3 мы видим, что для наблюдения 1 фактическим классом является класс *Да*. Мы строили лес из 10 деревьев классификации, значит было сгенерировано 10 бутстреп-выборок. В трех из 10 бутстреп-выборок наблюдение 1 отсутствует. Нас будут интересовать деревья, построенные по этим 3 бутстреп-выборкам. Их по отношению к наблюдению 1 можно назвать out-of-bag выборками. Одно дерево проголосовало за класс *Да*, 2 дерева проголосовали за класс *Нет*. В итоге для наблюдения 1 неверно прогнозируется класс *Нет*. Если мы представим голоса деревьев не в виде частот, а в виде процентных долей, получим вероятности классов (вспомним, что в пакете R `randomForest` вероятности классов, спрогнозированные по методу OOB – это доли голосов деревьев, поданных за каждый класс, при этом нас интересуют деревья, которые построены по out-of-bag выборкам).

```
# смотрим процентные доли голосов деревьев для каждого класса
# (деревья построены по out-of-bag выборкам)
oob_vote_fract <- predict(model, type="vote", norm.votes=TRUE)
oob_vote_fract
```

Сводка 4.5. Процентные доли голосов деревьев, поданных за каждый класс, или вероятности классов (деревья построены по out-of-bag выборкам)

```
         Да       Нет
1 0.3333333 0.6666667
2 0.0000000 1.0000000
3 0.0000000 1.0000000
4 1.0000000 0.0000000
5 0.0000000 1.0000000
attr(,"class")
[1] "matrix" "votes"
```

Возьмем наблюдение 1. Из сводки 4.4 мы видим, что из 3 деревьев одно проголосовало за класс *Да*, получаем вероятность класса *Да* 1/3=0,33, а остальные деревья проголосовали за класс *Нет*, получаем вероятность класса *Нет* 2/3=0,67. Можно вывести вероятности классов, непосредственно задав для параметра `type` значение `"prob"`.

```
# выводим вероятности классов,
# спрогнозированные по методу OOB
oob_probabilities <- predict(model, type="prob")
oob_probabilities
```

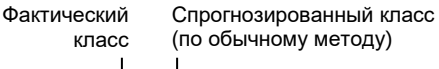
Сводка 4.6. Вероятности классов, спрогнозированные по методу OOB

```
         Да       Нет
1 0.3333333 0.6666667
2 0.0000000 1.0000000
3 0.0000000 1.0000000
4 1.0000000 0.0000000
5 0.0000000 1.0000000
attr(,"class")
[1] "matrix" "votes"
```

Итак, мы выяснили, как с помощью метода OOB можно вычислить ошибку классификации, спрогнозировать классы зависимой переменной и вероятности классов. Теперь вычислим ошибку классификации, спрогнозируем классы и вероятности классов обычным методом. Для функции `predict` указываем аргумент, задающий обучающую выборку, и прогнозируем классы зависимой переменной, установив для параметра `type` значение `"response"`. В пакете R `randomForest` класс, спрогнозированный для наблюдения по обычному методу – это класс, за который проголосовало большинство деревьев, построенных по всем бутстреп-выборкам.

```
# выводим классы, спрогнозированные по обычному методу
predictions <- predict(model, data, type="response")
results <-data.frame(data, result=predictions)
results
```

Сводка 4.7. Фактические и спрогнозированные классы зависимой переменной, для прогнозирования используется обычный метод



	age	gender	default	result
1	70	Мужской	Да	Да
2	64	Мужской	Нет	Нет
3	69	Женский	Да	Да
4	68	Мужской	Нет	Нет
5	65	Женский	Нет	Нет

Верно спрогнозированные наблюдения

Из сводки 4.7 видим, что неверно спрогнозированных наблюдений нет и мы получаем нулевую ошибку классификации. Теперь выясним, как мы получили классы, спрогнозированные по обычному методу. Для этого нам по каждому наблюдению нужно посмотреть, как распределились голоса деревьев, построенных по всем бутстреп-выборкам.

```
# смотрим частоты голосов деревьев для каждого класса
# (деревья построены по всем бутстреп-выборкам)
vote_freq <- predict(model, data, type="vote", norm.votes=FALSE)
vote_freq
```

Сводка 4.8. Частоты голосов деревьев, поданных за каждый класс (деревья построены по всем бутстреп-выборкам)

```
  Да Нет
1  7   3
2  0  10
3  7   3
4  1   9
5  2   8
attr(,"class")
[1] "matrix" "votes"
```

Возьмем наблюдение 1. Из сводки 4.7 мы знаем, что для наблюдения 1 фактический класс – это класс *Да*. Мы строили лес из 10 деревьев классификации, значит было сгенерировано 10 бутстреп-выборок. Мы используем обычный метод, поэтому нас будут интересовать деревья, построенные по всем 10 бутстреп-выборкам. 7 деревьев проголосовали за класс *Да*, 3 дерева проголосовали за класс *Нет*. В итоге по правилу большинства верно прогнозируется класс *Да*.

Теперь представим голоса деревьев в виде процентных долей, чтобы получить вероятности классов. Вероятности классов, спрогнозированные по обычному методу – это по-прежнему доли голосов деревьев, поданных за каждый класс, но теперь нас интересуют деревья, построенные по всем бутстреп-выборкам, а не только по out-of-bag выборкам.

```
# смотрим процентные доли голосов деревьев для каждого класса
# (деревья построены по всем бутстреп-выборкам)
vote_fract <- predict(model, data, type="vote", norm.votes=TRUE)
vote_fract
```

Сводка 4.9. Процентные доли голосов деревьев, поданных за каждый класс, или вероятности классов (деревья построены по всем бутстреп-выборкам)

```
   Да Нет
1 0.7 0.3
2 0.0 1.0
3 0.7 0.3
4 0.1 0.9
5 0.2 0.8
attr(,"class")
[1] "matrix" "votes"
```

Возьмем наблюдение 1. Из сводки 4.8 мы видим, что по данному наблюдению из 10 деревьев семь проголосовали за класс *Да*, получаем вероятность класса *Да* 7/10=0,7, а остальные деревья проголосовали за класс *Нет*, получаем вероятность класса *Нет* 3/10=0,3. И вновь мы можем вывести вероятности классов, непосредственно задав для параметра `type` значение `"prob"`.

```
# смотрим вероятности классов,
# спрогнозированные по методу OOB
probabilities <- predict(model, data, type="prob")
probabilities
```

Сводка 4.10. Вероятности классов зависимой переменной, спрогнозированные по обычному методу

```
   Да Нет
1 0.7 0.3
2 0.0 1.0
3 0.7 0.3
4 0.1 0.9
5 0.2 0.8
attr(,"class")
[1] "matrix" "votes"
```

До этого момента мы вычисляли частоты голосов деревьев, проголосовавших за тот или иной класс, однако пакет `randomForest` позволяет узнать, как голосовало каждое дерево в отдельности. Достаточно воспользоваться параметром `predict.all` и задать для него значение `TRUE`.

```
# смотрим частоты голосов деревьев по
# каждому классу и голос каждого дерева
ind_predictions <- predict(model, data, type="vote",
                           norm.votes=FALSE, predict.all=TRUE)
ind_predictions
```

Сводка 4.11. Частоты голосов и голоса отдельных деревьев

```
$aggregate
  Да Нет
1  7   3
2  0  10
3  7   3
4  1   9
5  2   8
attr(,"class")
[1] "matrix" "votes"

$individual
  [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
1 "Да"  "Да"  "Нет" "Да"  "Да"  "Да"  "Нет" "Да"  "Нет" "Да"
2 "Нет" "Нет" "Нет" "Нет" "Нет" "Нет" "Нет" "Нет" "Нет" "Нет"
3 "Да"  "Да"  "Да"  "Да"  "Нет" "Нет" "Да"  "Да"  "Нет" "Да"
4 "Нет" "Нет" "Нет" "Нет" "Нет" "Да"  "Нет" "Нет" "Нет" "Нет"
5 "Нет" "Нет" "Да"  "Нет" "Нет" "Нет" "Да"  "Нет" "Нет" "Нет"
```


В разделе `$aggregate` приводятся частоты голосов деревьев, а в разделе `$individual` можно увидеть, как каждое из 20 деревьев проголосовало за тот или иной класс. Например, для наблюдения 1 видим, что дерево №1 (или дерево, построенное по бустреп-выборке №1) проголосовало за класс *Да*), дерево №2 (или дерево, построенное по бустреп-выборке №2) тоже проголосовало за класс *Да*, а дерево №3 (или дерево, построенное по бустреп-выборке №3) проголосовало за класс *Нет* и так далее. По сути мы получаем информацию о листовых спрогнозированных классах. Листовой спрогнозированный класс – это класс, спрогнозированный для листа отдельного дерева, в которое попало наше наблюдение.

Теперь перейдем к задаче регрессии. Мы вычислим среднеквадратичную ошибку и R-квадрат (процент объясненной дисперсии), спрогнозируем значения зависимой переменной с помощью обоих методов.

Загрузим наши данные для задачи регрессии и взглянем на них.

```
# загружаем данные
data <- read.csv2("C:/Trees/RF_regression.csv")

# смотрим данные
data
```

Сводка 4.12. Исходные данные для построения случайного леса деревьев регрессии

```
   age  gender days_of_delinquency
1   70 Мужской                  18
2   64 Мужской                  15
3   69 Женский                  17
4   68 Мужской                  19
5   65 Женский                  15
6   63 Мужской                  16
7   55 Женский                  13
8   60 Мужской                  14
9   67 Мужской                  16
10  68 Женский                  14
```

Строим модель случайного леса из 10 деревьев регрессии, вычисляем среднеквадратичную ошибку и R-квадрат по методу OOB.

```
# задаем стартовое значение генератора случайных
# чисел для воспроизводимости результатов
set.seed(152)

# строим случайный лес из 10 деревьев регрессии
model<-randomForest(days_of_delinquency ~., data, ntree=10, importance=TRUE)

# выводим информацию о среднеквадратичной ошибке
# и R-квадрате по методу OOB
print(model)
```

Сводка 4.13. Оценка качества модели (используется среднеквадратичная ошибка и R-квадрат по методу OOB)

```
Call:
 randomForest(formula = days_of_delinquency ~ ., data = data,      ntree = 10, importance =
TRUE)
               Type of random forest: regression
                     Number of trees: 10
No. of variables tried at each split: 1

          Mean of squared residuals: 3.133282
                    % Var explained: 2.39
```

Из сводки 4.13 видим, что для наших данных с помощью метода OOB мы получили среднеквадратичную ошибку, равную 3,13, и R-квадрат, равный 2,39. Теперь посмотрим на значения зависимой переменной, спрогнозированные по методу OOB. Для этого в функции `predict` опускаем аргумент, задающий обучающую выборку, а для параметра `type` задаем значение `"response"`. В пакете R `randomForest` значение зависимой переменной, спрогнозированное по методу OOB, – это результат усреднения средних значений, которые вычислили деревья, построенные по out-of-bag выборкам.

```
# выводим значения, спрогнозированные по методу OOB
oob_predictions <- predict(model, type="response")
oob_results <-data.frame(data, result=oob_predictions)
oob_results
```

Сводка 4.14. Фактические и спрогнозированные значения зависимой переменной, для прогнозирования используется метод OOB

Фактическое значение

Спрогнозированное значение (по методу OOB)

	age	gender	days_of_delinquency	result
1	70	Мужской	18	17.00000
2	64	Мужской	15	16.10000
3	69	Женский	17	15.36667
4	68	Мужской	19	15.99333
5	65	Женский	15	15.50000
6	63	Мужской	16	15.75000
7	55	Женский	13	15.36667
8	60	Мужской	14	16.85000
9	67	Мужской	16	16.13333
10	68	Женский	14	15.83333

Теперь по каждому наблюдению вычисляем разность между фактическим значением зависимой и спрогнозированным значением зависимой переменной, эту разность возводим в квадрат. Квадраты остатков складываем и полученную сумму делим на количество наблюдений. В итоге получаем нашу среднеквадратичную ошибку по методу OOB, которая была приведена ранее в сводке 4.13.

```
# вычисляем среднеквадратичную ошибку по методу OOB, для этого сумму
# квадратов разностей между фактическими и спрогнозированными значениями
# зависимой переменной делим на количество наблюдений, при этом
# каждое спрогнозированное значение – результат усреднения средних
# значений, вычисленных деревьями по OOB выборкам
oob_MSE <- sum((data$days_of_delinquency - oob_predictions)^2)/nrow(data)
oob_MSE
```

Сводка 4.15. Среднеквадратичная ошибка по методу OOB, вычисленная вручную

```
[1] 3.133282
```

А сейчас вручную вычислим R-квадрат по методу OOB. Сначала вычисляем общую сумму квадратов отклонений – сумму квадратов отклонений фактических значений зависимой переменной от ее среднего значения. Затем вычисляем остаточную сумму квадратов отклонений – сумму квадратов отклонений фактических значений зависимой переменной от спрогнозированных. Остаточную сумму квадратов отклонений делим на общую сумму квадратов отклонений, частное вычитаем из единицы и полученный результат умножаем на 100. В итоге получаем наш R-квадрат по методу OOB, который был приведен ранее в сводке 4.13.

```
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от ее среднего значения
TSS <- sum((data$days_of_delinquency-(mean(data$days_of_delinquency)))^2)
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от спрогнозированных, при этом каждое
# спрогнозированное значение - результат усреднения средних
# значений, вычисленных деревьями по OOB выборкам
RSS <- sum((data$days_of_delinquency-oob_predictions)^2)
# вычисляем R-квадрат по методу OOB
oob_R2 <- (1-(RSS/TSS))*100
oob_R2
```

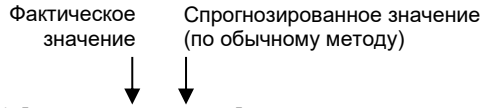
Сводка 4.16. R-квадрат по методу OOB, вычисленный вручную

```
[1] 2.389962
```

Итак, мы выяснили, как с помощью метода OOB можно вычислить среднеквадратичную ошибку и R-квадрат, спрогнозировать значения зависимой переменной. Теперь вычислим среднеквадратичную ошибку и R-квадрат, спрогнозируем значения зависимой переменной обычным методом. Для этого в функции `predict` указываем аргумент, задающий обучающую выборку, а для параметра `type` задаем значение `"response"`. В пакете R `randomForest` значение зависимой переменной, спрогнозированное по обычному методу, – это результат усреднения средних значений, которые вычислили деревья, построенные по всем бутстреп-выборкам.

```
# выводим значения, спрогнозированные по обычному методу
predictions <- predict(model, data, type="response")
results <-data.frame(data, result=predictions)
results
```

Сводка 4.17. Фактические и спрогнозированные значения зависимой переменной, для прогнозирования используется обычный метод



	age	gender	days_of_delinquency	result
1	70	Мужской	18	17.19333
2	64	Мужской	15	15.69000
3	69	Женский	17	16.20583
4	68	Мужской	19	16.72667
5	65	Женский	15	14.90250
6	63	Мужской	16	15.69000
7	55	Женский	13	14.07917
8	60	Мужской	14	14.86667
9	67	Мужской	16	16.04000
10	68	Женский	14	15.73917

Вручную вычислим среднеквадратичную ошибку и R-квадрат по обычному методу.

```
# вычисляем среднеквадратичную ошибку по обычному методу, для этого сумму
# квадратов разностей между фактическими и спрогнозированными значениями
# зависимой переменной делим на количество наблюдений, при этом
# каждое спрогнозированное значение – результат усреднения средних
# значений, вычисленных деревьями по всем бутстреп-выборкам
MSE <- sum((data$days_of_delinquency - predictions)^2)/nrow(data)

# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от ее среднего значения
TSS <- sum((data$days_of_delinquency-(mean(data$days_of_delinquency)))^2)
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от спрогнозированных, при этом каждое
# спрогнозированное значение – результат усреднения средних
# значений, вычисленных деревьями по всем бутстреп-выборкам
RSS <- sum((data$days_of_delinquency-predictions)^2)
# вычисляем R-квадрат по обычному методу
R2 <- (1-(RSS/TSS))*100

# печатаем результаты
output <- c(MSE, R2)
names(output) <- c("MSE", "R2")
output
```

Сводка 4.18. Среднеквадратичная ошибка и R-квадрат по обычному методу, вычисленные вручную

```
      MSE          R2
 1.197317   62.700389
```

В библиотеке `scikit-learn` для экземпляра класса `RandomForestClassifier` вычисляется обычная правильность, а для экземпляра класса `RandomForestRegressor` обычный R-квадрат. Значения (классы) зависимой переменной и вероятности классов вычисляются также обычным способом с помощью методов `predict` и `predict_proba` соответственно. Однако если задать значение параметра `oob_score=True`, то:

- для экземпляра класса `RandomForestClassifier` в атрибут `oob_score_` будет записана правильность, вычисленная по методу OOB, а в атрибут `oob_decision_function_` будут записаны вероятности классов, вычисленные по методу OOB;
- для экземпляра класса `RandomForestRegressor` в атрибут `oob_score_` будет записан R-квадрат, вычисленный по методу OOB, а в атрибут `oob_prediction_` будут записаны спрогнозированные значения, вычисленные по методу OOB.

Кроме того, с помощью атрибута `estimators_` можно получить листовые вероятности классов по каждому дереву.

Начнем с задачи классификации. Сейчас мы вычислим правильность, спрогнозируем классы зависимой переменной и вероятности классов с помощью обоих методов.

Давайте загрузим наши данные для задачи классификации и взглянем на них.

In[1]:

```
# импортируем необходимые библиотеки
import numpy as np
import pandas as pd

# записываем CSV-файл в объект DataFrame
data = pd.read_csv("C:/Trees/RF_classification.csv", encoding='cp1251', sep=';')

# смотрим данные
data
```

Out[1]:

	age	gender	default
0	70	Мужской	Да
1	64	Мужской	Нет
2	69	Женский	Да
3	68	Мужской	Нет
4	65	Женский	Нет

Выполняем дамми-кодирование.

In[2]:

```
# выполняем дамми-кодирование
print("Исходные переменные:\n", list(data.columns), "\n")
data_dummies = pd.get_dummies(data)
print("Переменные после get_dummies:\n", list(data_dummies.columns))
```

Out[2]:

```
Исходные переменные:
 ['age', 'gender', 'default']

Переменные после get_dummies:
 ['age', 'gender_Женский', 'gender_Мужской', 'default_Да', 'default_Нет']
```

Теперь можно подготовить массив меток зависимой переменной и массив признаков.

In[3]:

```
# создаем массив меток зависимой переменной
y = data_dummies.loc[:, 'default_Да']
# удаляем из массива признаков дамми-переменные
# default_Нет и default_Да, поскольку они
# представляют собой зависимую переменную
data_dummies.drop('default_Нет', axis=1, inplace=True)
data_dummies.drop('default_Да', axis=1, inplace=True)
# создаем массив признаков
X = data_dummies
```

Строим модель случайного леса из 10 деревьев классификации и смотрим правильность, вычисленную по методу OOB. Чтобы получить правильность по методу OOB при создании экземпляра класса `RandomForestClassifier` нужно задать для параметра `oob_score` значение `True` (по умолчанию задано значение `False`), а затем воспользоваться атрибутом `oob_score_` объекта-модели.

In[4]:

```
# импортируем класс RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
# создаем экземпляр класса RandomForestClassifier
forest=RandomForestClassifier(n_estimators=10, random_state=152,
                              oob_score=True, n_jobs=-1)
# подгоняем модель
forest.fit(X, y)
# вычисляем правильность по методу OOB
print("Правильность по методу OOB: {:.2f}".format(forest.oob_score_))
```

Out[4]:

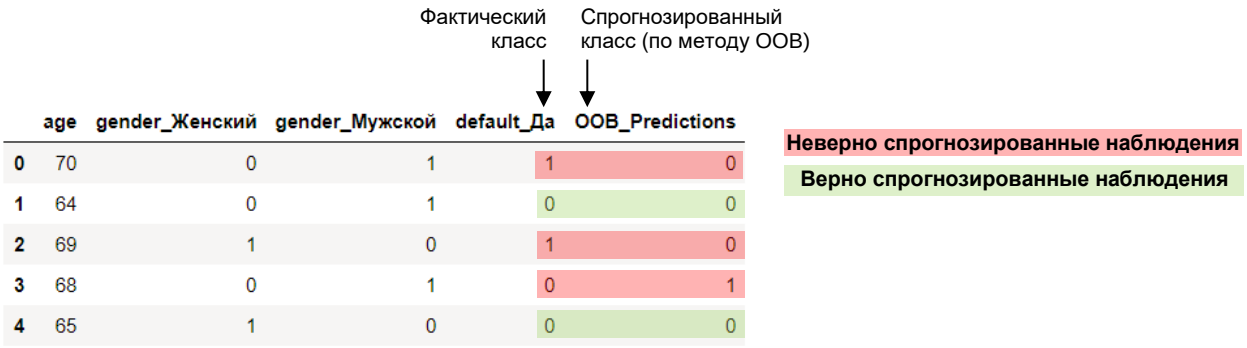
```
Правильность по методу OOB: 0.40
```

Из **Out[4]** видим, что для наших данных мы получили правильность по методу OOB, равную 40%. Теперь посмотрим на классы зависимой переменной, спрогнозированные по методу OOB. Для этого воспользуемся атрибутом `oob_decision_function_`. Поскольку мы работаем в библиотеке `scikit-learn`, класс зависимой переменной, спрогнозированный по методу OOB, – это класс с наибольшей усредненной листовой вероятностью, при этом усреднение происходит по деревьям, построенным по out-of-bag выборкам.

In[5]:

```
# прогнозируем классы зависимой переменной по методу OOB
oob_predictions=np.argmax(forest.oob_decision_function_, axis=1)
# преобразуем прогнозы в объект DataFrame
oob_predictions=pd.DataFrame(oob_predictions,
                             index=data.index, columns=['OOB_Predictions'])
# конкатенируем массив признаков, массив меток и OOB прогнозы
oob_results=pd.concat([X, y, oob_predictions], axis=1)
oob_results
```

Out[5]:



	age	gender_Женский	gender_Мужской	default_Да	OOB_Predictions
0	70	0	1	1	0
1	64	0	1	0	0
2	69	1	0	1	0
3	68	0	1	0	1
4	65	1	0	0	0

Из **Out[5]** видим, что из 5 наблюдений 2 верно спрогнозированы, отсюда и получаем нашу правильность 2/5=0,4 или 40%.

Теперь выясним, как мы получили классы, спрогнозированные по методу OOB. Для каждого наблюдения вычисляем усредненные листовые вероятности классов, усреднение происходит по деревьям, которые построены по out-of-bag выборкам. Вновь воспользуемся атрибутом `oob_decision_function_`.

In[6]:
```
# прогнозируем вероятности классов по методу OOB
print("Вероятности, спрогнозированные по методу OOB:\n{}".
      format(forest.oob_decision_function_))
```

Out[6]:
```
Вероятности, спрогнозированные по методу OOB:
[[ 0.8         0.2       ]
 [ 1.          0.        ]
 [ 0.8         0.2       ]
 [ 0.          1.        ]
 [ 0.66666667  0.33333333]]
```

По наблюдению 1 видим, что классом с наибольшей усредненной листовой вероятностью является класс 0 (усредненная листовая вероятность класса 0 равна 0.8), поэтому и спрогнозирован класс 0.
Теперь вычислим правильность по обычному методу. Для этого воспользуемся методом `score`.

In[7]:
```
# вычисляем правильность по обычному методу
print("Правильность по обычному методу: {:.3f}".format(forest.score(X, y)))
```

Out[7]:
```
Правильность по обычному методу: 1.000
```

Из **Out[7]** видим, что для наших данных мы получили идеальную правильность по обычному методу. Теперь посмотрим на классы зависимой переменной, спрогнозированные по обычному методу. Для этого воспользуемся методом `predict`. Класс, спрогнозированный по обычному методу, – это класс с наибольшей усредненной листовой вероятностью, однако теперь усреднение происходит по деревьям, построенным по всем бутстреп-выборкам.

In[8]:
```
# прогнозируем классы зависимой переменной по обычному методу
predictions=forest.predict(X)
# преобразуем прогнозы в объект DataFrame
predictions=pd.DataFrame(predictions,
                         index=data.index, columns=['Predictions'])
# конкатенируем массив признаков, массив меток и прогнозы
results=pd.concat([X, y, predictions], axis=1)
results
```

Out[8]:

Фактический класс — Спрогнозированный класс (по обычному методу)

	age	gender_Женский	gender_Мужской	default_Да	Predictions
0	70	0	1	1	1
1	64	0	1	0	0
2	69	1	0	1	1
3	68	0	1	0	0
4	65	1	0	0	0

Верно спрогнозированные наблюдения

Из Out[8] видим, что из 5 наблюдений 5 верно спрогнозированы, отсюда и получаем нашу идеальную правильность 5/5=1 или 100%.
Теперь выясним, как мы получили классы, спрогнозированные по обычному методу. Для каждого наблюдения вычисляем усредненные листовые вероятности классов, усреднение происходит по деревьям, построенным по всем бутстреп-выборкам. Здесь мы должны воспользоваться методом `predict_proba`.

```
In[9]:
# прогнозируем вероятности классов по обычному методу
print("Вероятности, спрогнозированные по обычному методу:\n{}".
      format(forest.predict_proba(X)))
```

```
Out[9]:
Вероятности, спрогнозированные по обычному методу:
[[ 0.4  0.6]
 [ 1.   0. ]
 [ 0.4  0.6]
 [ 0.7  0.3]
 [ 0.9  0.1]]
```

По наблюдению 1 видим, что классом с наибольшей усредненной листовой вероятностью является класс 1 (усредненная листовая вероятность класса 1 равна 0,6), поэтому и спрогнозирован класс 1.
Кроме того, с помощью атрибута `estimators_` для наблюдения можно вывести не усредненные, а обычные листовые вероятности классов, вычисленные каждым деревом. Давайте выведем их для наблюдения 1.

```
In[10]:
# вычисляем листовые вероятности каждого дерева
# для первого наблюдения
for tree_in_forest in forest.estimators_:
    print(tree_in_forest.predict_proba(X)[0])
```

```
Out[10]:
[ 1.  0.]
[ 0.  1.]
[ 1.  0.]
[ 0.  1.]
[ 0.  1.]
[ 0.  1.]
[ 1.  0.]
[ 0.  1.]
[ 0.  1.]
[ 1.  0.]
```

Из Out[10] видно, что если сложить листовые вероятности класса 0 и усреднить их по всем 10 деревьям, получим усредненную листовую вероятность класса 0, равную 0,4, а если сложить листовые вероятности класса 1 и усреднить их по всем 10 деревьям, получим усредненную листовую вероятность класса 1, равную 0,6.
Перейдем к задаче регрессии. Сейчас мы вычислим R-квадрат, спрогнозируем значения зависимой переменной с помощью обоих методов.
Давайте загрузим наши данные и взглянем на них.

In[11]:
```
# записываем CSV-файл в объект DataFrame
data = pd.read_csv("C:/Trees/RF_regression.csv", encoding='cp1251', sep=';')

# смотрим данные
data
```

Out[11]:

	age	gender	days_of_delinquency
0	70	Мужской	18
1	64	Мужской	15
2	69	Женский	17
3	68	Мужской	19
4	65	Женский	15
5	63	Мужской	16
6	55	Женский	13
7	60	Мужской	14
8	67	Мужской	16
9	68	Женский	14

Выполняем дамми-кодирование.

In[12]:
```
# выполняем дамми-кодирование
print("Исходные переменные:\n", list(data.columns), "\n")
data_dummies = pd.get_dummies(data)
print("Переменные после get_dummies:\n", list(data_dummies.columns))
```

Out[12]:
```
Исходные переменные:
 ['age', 'gender', 'days_of_delinquency']

Переменные после get_dummies:
 ['age', 'days_of_delinquency', 'gender_Женский', 'gender_Мужской']
```

Подготавливаем массив значений зависимой переменной и массив признаков.

In[13]:
```
# создаем массив значений зависимой переменной
y = data_dummies.loc[:, 'days_of_delinquency']
# удаляем из массива признаков переменную
# days_of_delinquency, поскольку она
# представляет собой зависимую переменную
data_dummies.drop('days_of_delinquency', axis=1, inplace=True)
# создаем массив признаков
X = data_dummies
```

Строим модель случайного леса из 10 деревьев регрессии и смотрим R-квадрат, вычисленный по методу OOB. Чтобы получить R-квадрат по методу OOB при создании экземпляра класса `RandomForestRegressor` нужно задать для параметра `oob_score` значение `True` (по умолчанию задано значение `False`), а затем воспользоваться атрибутом `oob_score_` объекта-модели.

In[14]:

```
# импортируем класс RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
# создаем экземпляр класса RandomForestRegressor
forest=RandomForestRegressor(n_estimators=10, random_state=152,
                             oob_score=True, n_jobs=-1)
# подгоняем модель
forest.fit(X, y)
# вычисляем R-квадрат по методу OOB
print("R-квадрат по методу OOB: {:.2f}".format(forest.oob_score_))
```

Out[14]:

```
R-квадрат по методу OOB: -0.36
```

Из Out[14] видим, что для наших данных мы получили отрицательный R-квадрат по методу OOB, что говорит об очень плохом качестве модели. Теперь посмотрим на значения зависимой переменной, спрогнозированные по методу OOB. Для этого воспользуемся атрибутом `oob_prediction_`. Как и в пакете R `randomForest`, в библиотеке `scikit-learn` для экземпляра класса `RandomForestRegressor` значение зависимой переменной, спрогнозированное по методу OOB, – это результат усреднения средних значений, которые вычислили деревья, построенные по out-of-bag выборкам.

In[15]:

```
# прогнозируем значения зависимой переменной по методу OOB
oob_predictions=forest.oob_prediction_
# преобразуем прогнозы в объект DataFrame
oob_predictions=pd.DataFrame(oob_predictions,
                             index=data.index, columns=['OOB_Predictions'])
# конкатенируем массив признаков, массив значений
# зависимой переменной и OOB прогнозы
oob_results=pd.concat([X, y, oob_predictions], axis=1)
oob_results
```

Out[15]:

Фактическое значение

Спрогнозированное значение (по методу OOB)

	age	gender_Женский	gender_Мужской	days_of_delinquency	OOB_Predictions
0	70	0	1	18	17.800000
1	64	0	1	15	16.000000
2	69	1	0	17	16.000000
3	68	0	1	19	14.666667
4	65	1	0	15	14.600000
5	63	0	1	16	15.000000
6	55	1	0	13	14.333333
7	60	0	1	14	15.800000
8	67	0	1	16	17.750000
9	68	1	0	14	17.666667

Вручную вычислим R-квадрат по методу OOB.

In[16]:
```
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от ее среднего значения
TSS=((oob_results['days_of_delinquency']-oob_results['days_of_delinquency'].mean())**2).sum()
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от спрогнозированных, при этом каждое
# спрогнозированное значение - результат усреднения средних
# значений, вычисленных деревьями по OOB выборкам
RSS=((oob_results['days_of_delinquency']-oob_results['OOB_Predictions'])**2).sum()
# вычисляем R-квадрат по методу OOB
print("R-квадрат по методу OOB, вычисленный вручную: {:.2f}".format(1-(RSS/TSS)))
```

Out[16]:
```
R-квадрат по методу OOB, вычисленный вручную: -0.36
```

Теперь вычислим R-квадрат обычным методом. Для этого воспользуемся методом `score`.

In[17]:
```
# вычисляем R-квадрат по обычному методу
print("R-квадрат по обычному методу: {:.2f}".format(forest.score(X, y)))
```

Out[17]:
```
R-квадрат по обычному методу: 0.85
```

Из **Out[17]** видим, что для наших данных мы получили довольно высокое значение R-квадрат по методу OOB, что говорит о прекрасном качестве модели. Теперь посмотрим на значения зависимой переменной, спрогнозированные по методу OOB. Значение зависимой переменной, спрогнозированное по обычному методу, – это результат усреднения средних значений, которые вычислили деревья, построенные по всем бутстреп-выборкам.

In[18]:
```
# прогнозируем значения зависимой переменной по обычному методу
predictions=forest.predict(X)
# преобразуем прогнозы в объект DataFrame
predictions=pd.DataFrame(predictions,
                         index=data.index, columns=['Predictions'])
# конкатенируем массив признаков, массив значений
# зависимой переменной и прогнозы
results=pd.concat([X, y, predictions], axis=1)
results
```

Фактическое значение

Спрогнозированное значение (по обычному методу)

Out[18]:

	age	gender_Женский	gender_Мужской	days_of_delinquency	Predictions
0	70	0	1	18	17.9
1	64	0	1	15	15.3
2	69	1	0	17	16.5
3	68	0	1	19	17.7
4	65	1	0	15	14.8
5	63	0	1	16	15.9
6	55	1	0	13	13.4
7	60	0	1	14	14.9
8	67	0	1	16	16.7
9	68	1	0	14	15.1

Теперь вручную вычислим R-квадрат по обычному методу.

```
In[19]:
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от ее среднего значения
TSS=((results['days_of_delinquency']-results['days_of_delinquency'].mean())**2).sum()
# вычисляем сумму квадратов отклонений фактических значений
# зависимой переменной от спрогнозированных, при этом каждое
# спрогнозированное значение - результат усреднения средних
# значений, вычисленных деревьями по всем бутстреп-выборкам
RSS=((results['days_of_delinquency']-results['Predictions'])**2).sum()
# вычисляем R-квадрат по обычному методу
print("R-квадрат по обычному методу, вычисленный вручную: {:.2f}".format(1-(RSS/TSS)))

Out[19]:
R-квадрат по обычному методу, вычисленный вручную: 0.85
```

Лекция 4.3. Настройка параметров случайного леса

4.3.1. Количество деревьев в ансамбле

Главный параметр случайного леса – это количество деревьев в ансамбле. Как правило, большее количество деревьев практически всегда дает лучший результат. Это обусловлено тем, что усреднение результатов по большему количеству деревьев позволит получить более устойчивый ансамбль за счет снижения переобучения. Следует отметить, что если при увеличении числа деревьев улучшения качества не происходит или даже наблюдается уменьшение качества прогноза, то это может говорить о плохом качестве выборки, о присутствии значительного шума в данных. Подобное явление также часто наблюдается на небольших выборках. Количество деревьев в ансамбле, необходимое для хорошего качества модели, возрастает с числом предикторов и ростом объема данных. Помните, что с ростом количества деревьев требуется больше памяти и больше времени для обучения.

Наилучший способ определить, сколько деревьев построить, это сравнить метрики качества моделей с разным количеством (последовательно увеличиваем значение, строим 100, 200, 300, 400, 500 деревьев) на контрольной выборке. Как вариант, можно воспользоваться перекрестной проверкой, здесь мы смотрим метрики качества, усредненные по контрольным блокам перекрестной проверки. Обычно по мере увеличения числа деревьев качество модели на обучающей выборке увеличивается (можно добиться даже 100%-ной правильности или значения AUC, равного 1), а на контрольной выборке качество увеличивается и затем стабилизируется на определенном значении.

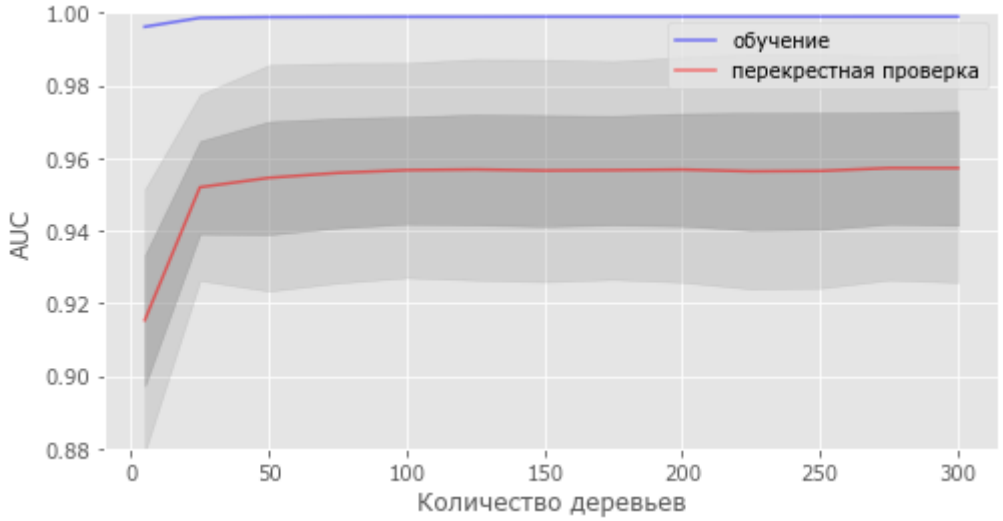


Рис. 4.8 График зависимости AUC от количества деревьев, используется перекрестная проверка

Взглянем на рис. 4.8. Анализируя значения AUC, усредненные по обучающим блокам перекрестной проверки, мы видим, что даже совсем небольшое количество деревьев дает идеальную дискриминирующую способность (значение AUC равно 1). Однако рассматривая метрики, усредненные на контрольных блоках перекрестной проверки, мы видим, что качество увеличивается медленнее и стабилизируется примерно на 150 деревьях при значении AUC, равном 0,957.

Как мы уже говорили выше, большее количество деревьев всегда дает лучшее качество, однако вы должны регулировать стоимость улучшения качества с вычислительной точки зрения. Например, мы можем использовать ансамбль из 300 деревьев и получить на контрольной выборке AUC 0.811, затем дополнительно натренировать еще 100 деревьев и получить AUC 0.83, в этом случае идея увеличить количество деревьев имеет смысл, если же дополнительная тренировка 100 деревьев дает в итоге AUC 0.812, то скорее всего стоимость такого улучшения будет сомнительна.

В пакете R `randomForest`, представляющим собой реализацию классического алгоритма случайного леса, параметр, задающий количество деревьев в ансамбля, называется `ntree`, в пакете R `ranger` – быстрой реализации случайного леса, использующей параллельные вычисления, он называется `num.trees`, в классах `RandomForestClassifier` и `RandomForestRegressor` питоновской библиотеки `scikit-learn` он называется `n_estimators`, в версиях библиотеки `h2o` для R (функция `h2o.randomForest`) и Python (класс `H2ORandomForestEstimator`) он называется `ntrees`.

4.3.2. Количество случайно отбираемых предикторов

Определив такое количество деревьев в ансамбле, которое дает приемлемое качество модели на контрольной выборке и обучается за приемлемое время, мы задаем количество случайно отбираемых предикторов. Лео Брейман называл этот параметр mtry. Поясним, как работает mtry. Количество случайно отбираемых предикторов, равное 1, означает, что при разбиении отбор признаков не будет осуществляться вообще, будет выполнен поиск точек расщепления для одного случайно выбранного признака. Если количество случайно отбираемых предикторов задать равным общему количеству предикторов в наборе данных, это будет обозначать, что в каждом разбиении смогут участвовать все предикторы набора данных, в отбор признаков не будет привнесена случайность (останется лишь случайность, обусловленная бутстрепом). При таком варианте все деревья в случайном лесе будут в большей степени схожи между собой, нежели при более низких значениях mtry, и качество модели может ухудшится. На практике подбирают значения mtry, которые составляют примерно 20–40% от общего числа предикторов. Эти значения были сформулированы на основе дополненных правил Лео Бреймана, приведенных на рис. 4.9.

Количество случайно отбираемых предикторов (mtry)	**Для классификации**	**Для регрессии**
корень/треть от общего количества предикторов, деленная пополам	$m = 0{,}5 \times \sqrt{M}$	$m = 0{,}5 \times (M/3)$
корень/треть от общего количества предикторов	$m = \sqrt{M}$	$m = M/3$
удвоенный корень/треть от общего количества предикторов	$m = 2 \times \sqrt{M}$	$m = 2 \times (M/3)$
общее количество предикторов		$m = M$*
	m – случайно отбираемое число предикторов, *M* – общее число предикторов в наборе	
Примечания: Полученные значения mtry округляем в меньшую сторону *Правило было сформулировано по итогам экспериментов уже после смерти Лео Бреймана		

Рис. 4.9 Правила для определения оптимального количества случайно отбираемых предикторов

Если есть несколько переменных с сильной прогнозной силой (прогнозную силу обычно определяют с помощью пермутированной важности, см. раздел 4.1.4 *Важность переменных*), меньшие значения mtry могут дать лучшее качество. Если данные содержат много переменных со слабой прогнозной силой, нужно попробовать бо́льшие значения mtry. При построении случайного леса с помощью классов `RandomForestClassifier` и `RandomForestRegressor` библиотеки `scikit-`

learn помимо правил Бреймана необходимо попробовать бо́льшие значения mtry. По мнению ряда специалистов[2], на практике варьирование значений mtry не оказывает существенного влияния на качество модели.

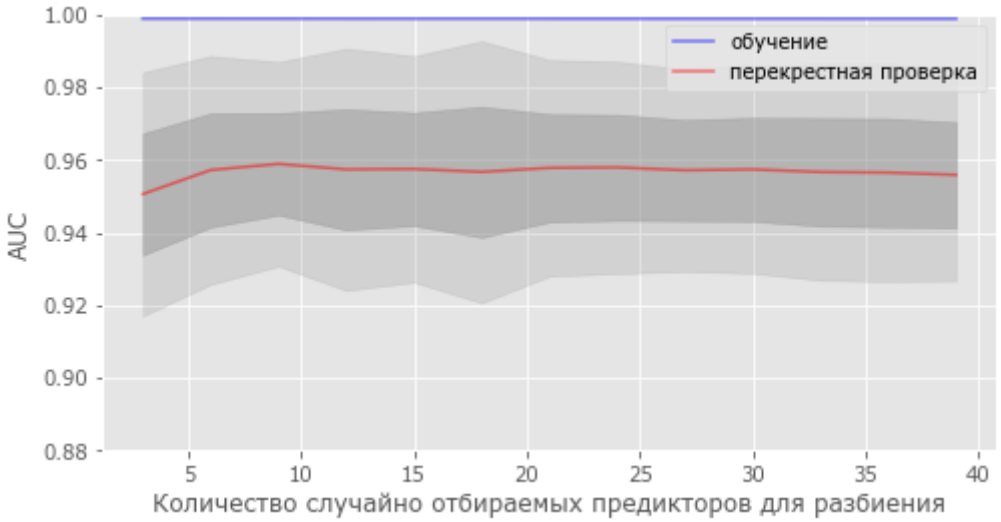


Рис. 4.10 График зависимости AUC модели из 300 полных деревьев от количества случайно отбираемых предикторов, используется перекрестная проверка

Взглянем на рис. 4.10. Здесь мы строим модели на основе полных деревьев. Набор содержит 40 предикторов. Видно, что варьирование количества случайно отбираемых предикторов не очень сильно влияет на качество модели.

Обрезка деревьев, как правило, требует корректировки mtry. Посмотрите на рис. 4.11. Здесь мы уже строим модели, используя обрезку деревьев (оставлено 10 уровней ниже корневого узла). Рассматривая значения AUC, усредненные на контрольных блоках перекрестной проверки, мы видим, что при увеличении количества случайно отбираемых предикторов качество модели постепенно снижается. Оптимальное качество мы получим, используя от 9 до 15 признаков.

[2] D. Richard Cutler, Thomas C. Edwards, Jr., Karen H. Beard, Adele Cutler, Kyle T. Hess, Jacob Gibson, and Joshua J. Lawler. 2007. Random forests for classification in ecology. Ecology 88:2783–2792.

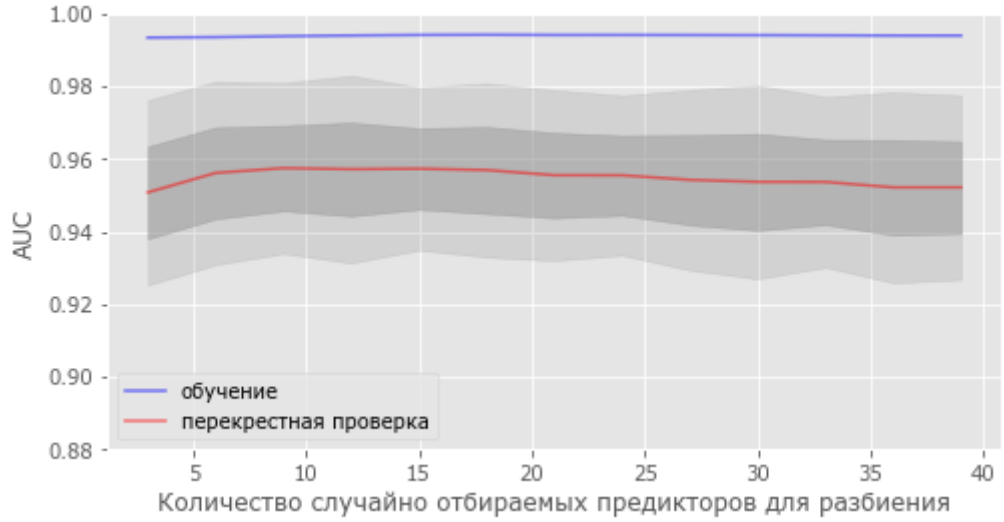


Рис. 4.11 График зависимости AUC модели из 300 обрезанных деревьев от количества случайно отбираемых предикторов, используется перекрестная проверка

В итоге при фиксированном количестве деревьев необходимо выбрать такое количество случайно отбираемых предикторов, которое дает максимальное качество модели на контрольной выборке.

В пакетах R `randomForest` и `ranger` параметр, задающий количество случайно отбираемых предикторов, называется `mtry` (оригинальное название, предложенное Брейманом), в классах `RandomForestClassifier` и `RandomForestRegressor` питоновской библиотеки `scikit-learn` он называется `max_features`, в версиях библиотеки `h2o` для R (функция `h2o.randomForest`) и Python (класс `H2ORandomForestEstimator`) он называется `mtries`.

В реализациях случайного леса, предлагаемых пакетами R `randomForest` и `ranger`, а также классами `RandomForestClassifier` и `RandomForestRegressor` питоновской библиотеки `scikit-learn` алгоритм по каждому дереву для разбиения каждого узла случайным образом отбирает фиксированное подмножество предикторов и затем находит наилучшую точку расщепления среди наилучших точек, найденных по каждому из случайно отобранных предикторов. Например, мы задали `mtry`/`max_features` равным 3, для всех деревьев для каждого узла мы будем случайным образом отбирать 3 предиктора от общего количества предикторов. В реализации случайного леса, предлагаемой библиотекой H2O, мы можем задать количество случайно отбираемых предикторов для каждого дерева (параметр `col_sample_rate_per_tree`) и задать количество случайно отбираемых предикторов для каждого узла (`mtries`). В итоге для каждого узла дерева мы будем случайным образом отбирать предикторы из числа случайно отобранных предикторов для каждого дерева. Например, у нас есть набор из 100 столбцов (100

предикторов). Мы задали `col_sample_rate_per_tree=0.602` и mtries=3. Для каждого дерева мы будем случайным образом отбирать 0.602*100=60 предикторов из 100. Из этих 60 предикторов мы для каждого узла будем случайно отбирать по 3 предиктора. Еще раз обратите внимание, что теперь не только у каждого узла, но и у каждого дерева будет «свой» случайный набор предикторов. Параметр `mtries` настраивается независимо от параметра `col_sample_rate_per_tree`, но его применение может быть ограничено значением `col_sample_rate_per_tree`. Например, если задать `col_sample_rate_per_tree=0.01`, то для каждого узла будет выбран лишь один предиктор независимо от значения `mtries`.

4.3.3. Глубина и минимальное количество наблюдений в терминальном узле

Еще один параметр, который стоит учитывать при построении модели случайного леса – это глубина деревьев. Как правило, при увеличении глубины возрастает время обучения, но при этом увеличивается качество модели на обучающей и контрольной выборках.

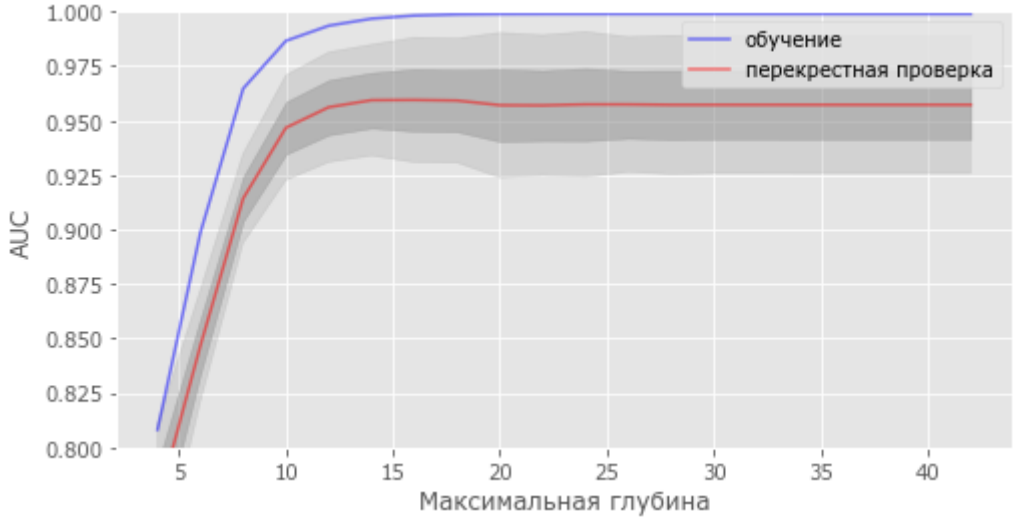


Рис. 4.12 График зависимости AUC модели из 300 деревьев от глубины, используется перекрестная проверка

В ряде случаев, например, при работе с зашумленными данными, построение деревьев с максимальной глубиной не дает хорошего качества модели. Это обусловлено тем, что при очень высоких значениях глубины деревья становятся излишне сложными и чувствительными к случайным возмущениям данных и в случае большого количества шумовых объектов рандомизация и усреднение по ансамблю не позволяют скомпенсировать возникшее переобучение. В таких случаях нужно выбрать меньшее значение глубины. Вместе с тем нужно избегать и слишком низкого

значения глубины, при котором деревья не смогут в достаточной степени обучиться и возникнет эффект недообучения.

В пакетах R `randomForest` и `ranger` данный параметр отсутствует, поэтому максимальную глубину в них регулируют косвенно с помощью варьирования минимального количества наблюдений в терминальном узле (увеличивая количество наблюдений в терминальном узле, уменьшаем глубину). В пакете R `randomForest` минимальное количество наблюдений в терминальном узле регулируется параметром `nodesize`, а в пакете R `ranger` – параметром `min.node.size`. В классах `RandomForestClassifier` и `RandomForestRegressor` питоновской библиотеки `scikit-learn`, в версиях библиотеки `h2o` для R (функция `h2o.randomForest`) и Python (класс `H2ORandomForestEstimator`) параметр, задающий максимальную глубину, называется `max_depth`. Кроме того, в этих инструментах максимальную глубину можно настроить косвенно, изменив количество наблюдений в терминальном узле. Соответствующий параметр в классах `RandomForestClassifier` и `RandomForestRegressor` питоновской библиотеки `scikit-learn` называется `min_samples_leaf`, в версиях библиотеки `h2o` для R (функция `h2o.randomForest`) и Python (класс `H2ORandomForestEstimator`) он получил название `min_rows`.

Дополнительно в Python для подбора оптимальных значений `max_depth` и `min_samples_leaf` можно написать функцию, которая будет принимать обученную модель случайного леса и на основе этой модели строить гистограмму распределения глубин терминальных узлов по всему ансамблю деревьев и гистограмму распределения количества наблюдений в терминальных узлах также по всему ансамблю деревьев.

ln[20]:
```
from sklearn.tree import _tree

# Пишем функцию, которая вычисляет глубину терминальных
# узлов в отдельном дереве
def leaf_depths(tree, node_id = 0):

    '''
    tree.children_left и tree.children_right записывают идентификационные
    номера левого и правого узлов-потомков для данного узла
    '''

    left_child = tree.children_left[node_id]
    right_child = tree.children_right[node_id]

    '''
    Если данный узел является терминальным,
    то оба дочерних узла будут иметь значение _tree.TREE_LEAF,
    что позволяет нам проверить, является ли данный узел терминальным
    '''

    if left_child == _tree.TREE_LEAF:

       '''
       Задать глубину терминальных узлов равной 0
       '''
```

```
        depths = np.array([0])

    else:

        '''
        Получить значения глубины узлов-потомков
        и увеличить их на 1
        '''

        left_depths = leaf_depths(tree, left_child) + 1
        right_depths = leaf_depths(tree, right_child) + 1

        depths = np.append(left_depths, right_depths)

    return depths
```

ln[21]:

```
# Пишем функцию, которая вычисляет минимальное количество наблюдений
# в терминальных узлах отдельного дерева
def leaf_samples(tree, node_id = 0):

    left_child = tree.children_left[node_id]
    right_child = tree.children_right[node_id]

    if left_child == _tree.TREE_LEAF:

         samples = np.array([tree.n_node_samples[node_id]])

    else:

        left_samples = leaf_samples(tree, left_child)
        right_samples = leaf_samples(tree, right_child)

        samples = np.append(left_samples, right_samples)

    return samples
```

ln[22]:

```
# Пишем функцию, которая строит гистограмму распределения
# глубин терминальных узлов и гистограмму распределения
# количества наблюдений в терминальных узлах по ансамблю
def draw_ensemble(ensemble):

    plt.figure(figsize=(8,8))
    plt.subplot(211)

    depths_all = np.array([], dtype=int)

    for x in ensemble.estimators_:
        tree = x.tree_
        depths = leaf_depths(tree)
        depths_all = np.append(depths_all, depths)
        plt.hist(depths, histtype='step', color='#ddaaff',
                 bins=range(min(depths), max(depths)+1))

    plt.hist(depths_all, histtype='step', color='#9933ff',
             bins=range(min(depths_all), max(depths_all)+1),
             weights=np.ones(len(depths_all))/len(ensemble.estimators_),
             linewidth=2)
    plt.xlabel("Глубина терминальных узлов")

    samples_all = np.array([], dtype=int)

    plt.subplot(212)

    for x in ensemble.estimators_:
        tree = x.tree_
        samples = leaf_samples(tree)
        samples_all = np.append(samples_all, samples)
        plt.hist(samples, histtype='step', color='#aaddff',
                 bins=range(min(samples), max(samples)+1))
```

```
plt.hist(samples_all, histtype='step', color='#3399ff',
         bins=range(min(samples_all), max(samples_all)+1),
         weights=np.ones(len(samples_all))/len(ensemble.estimators_),
         linewidth=2)
plt.xlabel("Количество наблюдений в терминальных узлах")

plt.show()
```

На рис. 4.13 приведена гистограмма распределения глубин терминальных узлов по ансамблю деревьев, а также гистограмма оценок AUC на контрольной выборке для соответствующих значений `max_depth`. В качестве примера использовались данные для задачи *Выбор кредита Tinkoff.ru* (в рамках соревнования *Tinkoff Data Science Challenge*, по которым), по которым строился лес из 800 полных деревьев.

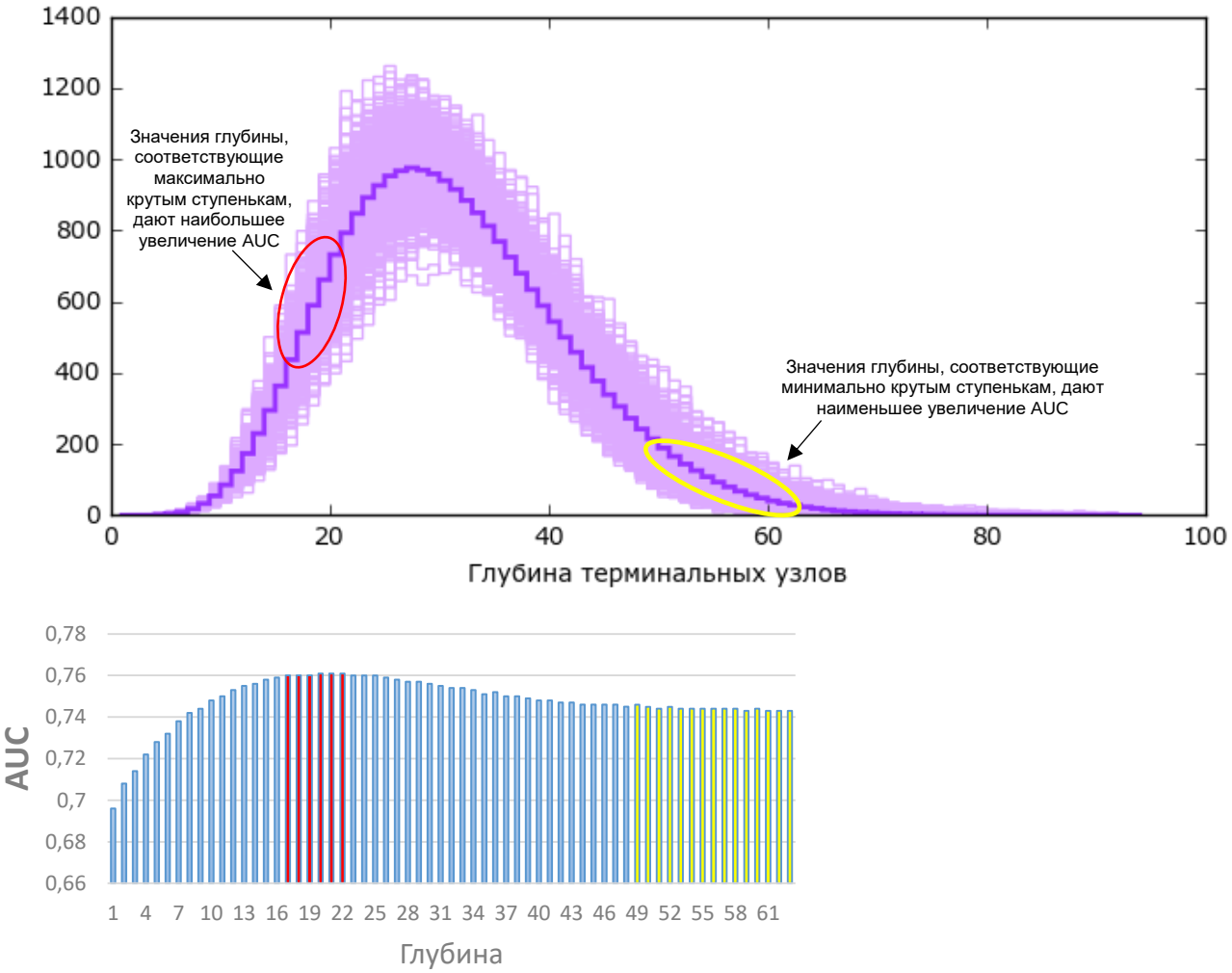


Рис. 4.14 Гистограмма распределения глубин терминальных узлов по ансамблю деревьев и гистограмма оценок AUC на контрольной выборке для соответствующих значений глубины

На верхнем графике рисунка 4.14 видно, что некоторые ветви ушли очень глубоко, некоторые не очень, но в среднем глубина равна примерно 28.

Первый вывод, который можно сделать – если мы будем использовать решетчатый поиск по значениям `max_depth`, значения `max_depth>50` можно не включать в сетку параметров, потому что они очень слабо влияют на получающиеся деревья. Об этом говорит разница в высоте ступенек, которая характеризует степень непохожести деревьев с т.з. глубины терминальных узлов (вспомним, чем больше деревья будут непохожи друг на друга, тем выше будет качество модели). Оптимальным будет такое значение `max_depth`, при котором мы получаем максимальную разницу в высоте ступенек, в данном случае это значения глубины от 17 до 22 (левый хвост распределения). Если для построения ансамбля взять значения глубины от 17 до 22, можно получить максимальное качество модели. Теперь давайте взглянем на правый хвост распределения (значения глубины от 28 и выше). Видно, что здесь разница в высоте ступенек меньше, чем в левом хвосте распределения. Если для моделирования брать значения `max_depth` из этого диапазона значений, улучшение качества модели будет меньше по сравнению с тем улучшением, которое мы получили бы, используя значения `max_depth` в диапазоне от 17 до 22.

На нижнем графике рисунка 4.14 мы видим, как будет варьировать оценка AUC на контрольной выборке в зависимости от выбранного значения `max_depth`. На нем как раз видно, что если для ансамбля взять значения глубины от 17 до 22, мы можем получить максимальное значение AUC, равное 0,761.

4.3.4. Биннинг количественных предикторов

Улучшения качества случайного леса можно добиться за счет использования различных стратегий поиска точек расщепления.

В классической реализации случайного леса для количественного предиктора с k категориями может быть рассмотрено k-1 вариантов разбиения. Значения будут отсортированы в порядке возрастания и в качестве точек расщепления могут быть рассмотрены средние по каждой паре упорядоченных смежных значений. Например, у нас есть значения количественной переменной *Возраст* 74, 70, 64 66, 65, 68, 69. Точки расщепления будут выглядеть следующим образом:

Предиктор *Возраст*

Значения	64	65	68	69	70	74
Точки расщепления	64,5	66,5	68,5	69,5	72	

Рис. 4.14 Создание точек расщепления классическим способом

В реализации случайного леса, предлагаемой библиотекой H2O, по каждому количественному предиктору будет создана гистограмма, состоящая из интервалов или бинов. Количество бинов для количественных предикторов регулируется параметрами `nbins` и `nbins_top_level`. Тип гистограммы для количественных предикторов регулируется параметром `histogram_type`. По умолчанию для количественного предиктора создается не менее 20 бинов одинаковой ширины (для параметра `nbins` по умолчанию используется значение `20`, а для параметра `histogram_type` по умолчанию задано значение `UniformAdaptive`, которое назначает интервалы одинаковой ширины). При этом на первом уровне дерева гистограмма не может иметь количество бинов, превышающее значение `nbins_top_level` (по умолчанию используется значение `1024`), затем на каждом последующем уровне происходит уменьшение этого максимального значения вдвое. Параметр `nbins_top_level` работает в паре с параметром `nbins`, который регулирует, когда нужно прекратить уменьшение количества бинов вдвое. Например, если на определенном уровне дерева `nbins_top_level` становится равным 32, а `nbins` задан равным `20`, то на последующем уровне разбиение количества бинов на 2 не происходит. Ширина бина будет определена по формуле (max-min)/N, где max – максимальное значение, min – минимальное значение, N – количество интервалов. В нашем случае будет создано 20 бинов с шагом (74-64)/20=0,5: от 64 до 64,5, от 64,5 до 65, ..., от 73,5 до 74. По границам этих интервалов будут найдены точки расщепления.

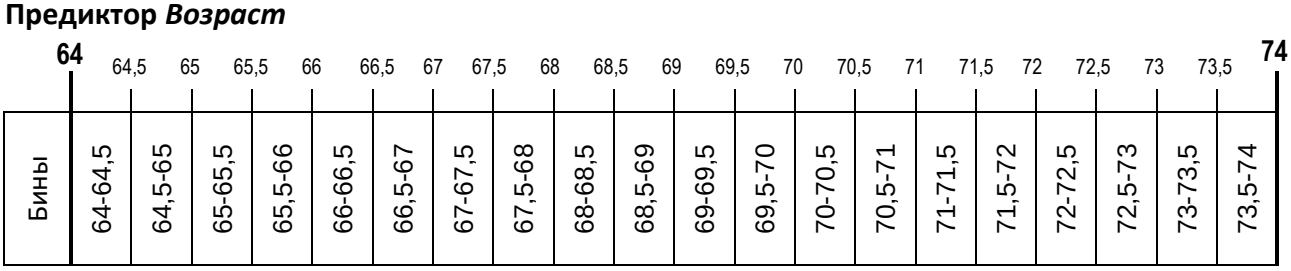


Рис. 4.15 Создание точек расщепления с помощью гистограммирования

Перебор значений `nbins` и `nbins_top_level` может улучшить качество модели.

Обратите внимание, при обработке количественных переменных с асимметричными распределениями и выбросами значение `UniformAdaptive`, установленное по умолчанию для параметра `histogram_type`, может привести к выбору менее оптимальных вариантов расщепления. Например, у нас есть количественная переменная, которая меняется в диапазоне от 0 до 10. Помимо этого у нас есть экстремальные значения 9999 (также вспомните, что часто пропуски кодируют большим

отрицательным или положительным значением, например, -9999 или 9999). В этом случае биннинг, подразумеваюший создание интервалов одинаковой ширины, может дать неоптимальное решение, потому что при создании интервалов будет опираться на значения минимума и максимума. Допустим, мы зададим `nbins` равным 10. Тогда ширина интервала будет вычислена как (9999-0)/10=999,9. У нас будут всего два бина: 0-999,9 и 999,9-9999. Все значения переменной, кроме значений 9999, окажутся в одном бине 0-999,9, в итоге будет рассмотрен только один вариант разбиения. Поэтому при обработке переменных с асимметричными распределениями и выбросами для параметра `histogram_type` лучше использовать значение `QuantilesGlobal`, которое создает интервалы, содержащие одинаковое количество наблюдений (квантили). В данном случае `nbins` уже будет задавать количество квантилей.

Кроме того, вы можете получить более оптимальные расщепления по итогам гистограммировании переменной с асимметричным распределением и выбросами, предварительно выполнив преобразования, максимизирующие нормальность распределения. Для распределения, скошенного вправо (положительный коэффициент асимметрии), обычно применяются следующие преобразования: квадратный корень sgn(x)*(abs(x)^1/2), кубический корень sgn(x)*(abs(x)^1/3), свернутый корень sgn(x)*sqrt(sqrt(abs(x))) и логарифм.

Для распределения, скошенного влево (отрицательный коэффициент асимметрии), обычно применяются следующие преобразования: квадратный корень (константа – x), кубический корень (константа – x) и логарифм (константа – x). Поскольку логарифм нуля, а равно и любого отрицательного числа, неопределен, перед использованием логарифмического преобразования ко всем значениям нужно добавить константу, чтобы сделать их положительными. При использовании корней обычно корень берут от модуля числа (чтобы не вычислять корни отрицательных чисел) и затем учитывают знак числа.

В ряде случаев улучшения можно добиться, задав для параметра `histogram_type` значение `Random`. В этом случае алгоритм случайным образом отбирает N–1 точек расщепления из диапазона значений и затем использует упорядоченный список этих точек разбиения для поиска наилучшей точки расщепления. N определяется параметрами `nbins` и `nbins_top_level`.

Перспективным является использование циклического перебора типов гистограмм расщепляющих значений. Его можно применить, задав для параметра `histogram_type` значение `RoundRobin`. Для первого дерева расщепляющие значения будут определены по границам интервалов одинаковой ширины, для второго дерева расщепляющие значения будут

определены по границам интервалов одинакого размера, для третьего дерева расщепляющие значения будут получены случайным образом, для четвертого дерева расщепляющие значения вновь будут определены по границам интервалов одинаковой ширины и так по кругу.

4.3.5. Биннинг категориальных предикторов

В классической реализации случайного леса для категориального предиктора с k категориями может быть рассмотрено $2^{k-1}-1$ вариантов разбиения. Категории делятся всеми возможными способами на две группы.

В реализации случайного леса, предлагаемой библиотекой H2O, по каждому категориальному предиктору будет создана гистограмма, состоящая из интервалов или бинов. При этом в H2O в отличие от библиотеки `scikit-learn` не используется one-hot-кодирование и бины будут сформированы из исходных категорий. Количество бинов для категориальных предикторов регулируется параметрами `nbins_cats` и `nbins_top_level`. По умолчанию для категориального предиктора создается не менее 1024 бинов (для параметра `nbins_cats` по умолчанию используется значение `1024`). При этом на первом уровне дерева гистограмма не может иметь количество бинов, превышающее значение `nbins_top_level` (по умолчанию используется значение `1024`), затем на каждом последующем уровне происходит уменьшение этого максимального значения вдвое. Параметр `nbins_top_level` работает в паре с параметром `nbins_cats`, который регулирует, когда нужно прекратить уменьшение количества бинов вдвое. Например, если на определенном уровне дерева `nbins_top_level` становится равным 32, а `nbins_cats` задан равным `20`, то на последующем уровне разбиение количества бинов на 2 не происходит.

Давайте посмотрим, как происходит разбиение на бины. Если количество категорий меньше значения параметра `nbins_cats`, каждая категория получает свой бин. Допустим, у нас есть переменная *Class*. Если у нее есть уровни A, B, C, D, E, F, G и мы зададим `nbins_cats=8`, то будут сформировано 7 бинов: {A}, {B}, {C}, {D}, {E}, {F} и {G}. Каждая категория получает свой бин. Будет рассмотрено $2^6-1=63$ точки расщепления. Если мы зададим `nbins_cats=10`, то все равно будут получены те же самые бины, потому что у нас всего 7 категорий. Если количество категорий больше значения параметра `nbins_cats`, категории будут сгруппированы в бины в лексикографическом порядке. Например, если мы зададим `nbins_cats=2`, то будет сформировано 2 бина: {A, B, C, D} и {E, F, G}. У нас будет одна точка расщепления. A, B, C и D попадут в один и тот же узел и будут разбиты только на последующем, более нижнем уровне или вообще не будут разбиты.

Значение параметра `nbins_cats` для категориальных предикторов оказывает гораздо большее влияние на обобщающую способность модели, чем значение параметра `nbins` для количественных предикторов (обычно более высокие значения параметра `nbins` приводят к выбору более оптимальных точек расщепления). Для предикторов с большим количеством категорий небольшое значение параметра `nbins_cats` может внести в процесс создания точек расщепления дополнительную случайность (поскольку категории группируется в определенном смысле произвольным образом), в то время как большие значения параметра `nbins_cats` (например, значение параметра `nbins_cats`, совпадающее с количеством категорий), наоборот, снижают эту случайность, каждая отдельная категория может быть рассмотрена при формировании точки разбиения, что приводит к переобучению на обучающем наборе (AUC= 1). Таким образом, этот параметр является очень важным параметром настройки. Как мы уже говорили, значение по умолчанию для `nbins_cats` равно `1024`. Значение `nbins_cats` может достигать 65 тыс. и этого должно хватить при работе с большими наборами данных. Если вы хотите получить более простую модель, необходимо уменьшить значения параметров `nbins_top_level` и `nbins_cats`. Если вы хотите получить более сложную, гибкую модель, необходимо увеличить значения `nbins_top_level` и `nbins_cats`. Имейте в виду, что увеличение числа `nbins_cats` может существенно повлиять на переобучение.

4.3.6. Способ обработки категориальных предикторов

Качество случайного леса также зависит от способа обработки категориальных предикторов. Одним из преимуществ деревьев решений и случайных лесов является их способность работать как с количественными, так и с категориальными переменными напрямую, без необходимости one-hot-кодирования, которое обычно требуется, например, обобщенным линейным моделям и нейронным сетям. Реализации случайного леса в пакетах R `randomForest`, `ranger` и `h2o` обрабатывают категориальные переменные по принципу «как есть», то есть каждое значение категориальной переменной представляет собой уровень (категорию) переменной. Однако в питоновской библиотеке `scikit-learn` каждый уровень категориальной переменной должен быть представлен дамми-переменной. Для этого нужно выполнить one-hot-кодирование. Помимо того, что one-hot-кодирование может привести к огромному увеличению размерности пространства данных, применительно к деревьям решений и случайному лесу оно стирает важную информацию о структуре категориального признака, по сути разбив один цельный признак на множество отдельных бинарных признаков. Бинарный признак может быть разбит только одним способом, а категориальный признак с k уровнями может быть разбит 2^{k-}

[1]–1 способами. Таким образом, в полученном пространстве признаков количественные переменные получат большую важность, чем категориальные переменные, представленные бинарными признаками. Все это может привести к ухудшению качества модели. Поэтому при построении случайного леса на данных, содержащих большое количество категориальных переменных, бывает полезно сравнить результаты, полученные с помощью традиционных классов `RandomForestClassifier`/`RandomForestRegressor` и класса `H2ORandomForestEstimator`.

Помимо того, что библиотека H2O обрабатывает категории по принципу «как есть», она предлагает и другие способы кодировки категориальных предикторов, что позволяет в ряде случаев повысить качество модели. Кодировку категориального предиктора можно изменить с помощью параметра `categorical_encoding`. По умолчанию используется значение `auto`, которое использует кодировку `Enum`. Кодировка `Enum` обрабатывает категории напрямую, то есть для каждого категориального предиктора создается по одному столбцу. При этом под капотом категориям в лексикографическом порядке будут присвоены целочисленные значения. Допустим, у нас есть переменная *Class*. Если у нее есть уровни A, B, C, D, E, F, G, то внутренне им будут присвоены целочисленные значения: A – 0, B – 1, C – 2, D – 3, E – 4, F – 5, G – 6. Кодировка `OneHotExplicit` задает N+1 столбцов для каждого категориального признака с N уровнями. Дополнительный столбец создается для пропущенных значений. Кодировка `Binary` задает не более 32 столбцов для категориального признака (используется хеширование). Кодировка `Eigen` выполняет one-hot-кодирование и оставляет k первых главных компонент для категориального признака.

В случае использования кодировки `LabelEncoder` категории в лексикографическом порядке будут преобразованы в целочисленные значения (начиная с 0) и в итоге теряют свою категориальную природу. Таким образом, с помощью `LabelEncoder` мы получаем порядковую переменную. Допустим, у нас есть переменная с 4 категориями A, B, C и D. Тогда `LabelEncoder` присвоит A – 0, B – 1, C – 2, D – 3.

Переменная Class	
A	0
B	1
C	2
D	3

Переменная Class	Кодировка LabelEncoder для переменной Class
B	1
A	0
C	2
A	0
A	0
D	3
D	3
B	1

Рис. 4.16 Пример кодировки LabelEncoder

Если задана кодировка `SortByResponse`, то для задачи 2-классовой классификации каждой категории предиктора, отсортированной по процентной доле наблюдений, попавших во второй класс зависимой переменной, а для задачи регрессии каждой категории предиктора, отсортированной по среднему значению зависимой переменной в этой категории, сопоставляется целое число, начиная с 0. Допустим, у нас есть бинарная зависимая переменная и предиктор с 4 категориями A, B, C и D. Сначала `SortByResponse` посчитает для каждой категории процентную долю наблюдений, попавших во второй класс зависимой переменной: A – 0.25, B – 0.5, C – 0.33, D – 0.66. Потом `SortByResponse` отсортирует категории по возрастанию: A – 0.25, C – 0.33, B – 0.5, D – 0.66. Затем `SortByResponse` каждой категории присвоит порядковый номер, начиная с 0: A – 0, C – 1, B – 2, D – 3.

Переменная Class		
A	0.25	1 из 4
B	0.5	1 из 2
C	0.33	1 из 3
D	0.66	2 из 3

A	0.25
C	0.33
B	0.5
D	0.66

A	0
C	1
B	2
D	3

Переменная Class	Зависимая переменная	Кодировка SortByResponse для переменной Class
A	0	0
A	1	0
C	1	1
D	1	3
A	0	0
A	0	0
B	0	2
B	1	2
C	0	1
D	1	3
C	0	1
D	0	3

Рис. 4.17 Пример кодировки SortByResponse

Выше мы говорили, что в классической реализации случайного леса для категориального предиктора с k категориями будет рассмотрено $2^{k-1}-1$ вариантов вариантов разбиений. Количество возможных разбиений возрастает лавинообразно с увеличением количества категорий.

Например, при k=33 будет рассмотрено четыре миллиона возможных вариантов расщепления. Поэтому в некоторых реализациях случайного леса, например, в пакете R `randomForest`, налагается ограничение на количество уровней категориальных предикторов (в пакете `randomForest` можно обрабатывать категориальные предикторы не более чем с 32 уровнями). Как вариант, можно перекодировать категориальный предиктор с большим количеством уровней в количественный предиктор и таким образом перейти к k–1 вариантам разбиения. Для этого каждую категорию предиктора заменяют относительной или абсолютной частотой категории. Заменяя категорию ее относительной частотой, мы по сути получаем вероятность встретить данную категорию в наборе данных.

Переменная Class	
A	0.44 (4 из 9)
B	0.33 (3 из 9)
C	0.22 (2 из 9)

Переменная Class	Кодировка FrequencyEncoding для переменной Class	
	относительная частота	абсолютная частота
A	0.44	4
B	0.33	3
A	0.44	4
C	0.22	2
B	0.33	3
B	0.33	3
A	0.44	4
A	0.44	4
C	0.22	2

Рис. 4.18 Пример кодировки FrequencyEncoding

Такую кодировку еще называют FrequencyEncoding, она не реализована в рассматриваемых пакетах, но ее легко сделать самостоятельно. При этом следует помнить, что если у вас есть категории с одинаковыми относительными частотами, деревья случайного леса не смогут их отделить друг от друга. Обратите внимание, что поскольку мы используем вычисления, то FrequencyEncoding нужно выполнять строго после разбиения на обучающую и контрольную выборки. Частотами, вычисленными для категорий переменной в обучающей выборке, заменяем категории переменной в обучающей и контрольной выборках.

4.3.7. Стартовое значение генератора случайных чисел

Поскольку случайный лес использует рандомизацию, установка различных стартовых значений генератора случайных чисел (или вообще отказ от использования стартового значения) может кардинально изменить построение модели. Существует даже шутка, которую опытные моделеры часто отпускают в адрес новичков: «если не удалось повысить качество модели за счет увеличения количества деревьев и количества

отбираемых признаков, попробуй стартовое значение». Чем больше деревьев в лесу, тем более устойчивым он будет к изменению стартового значения. Однако если вы хотите получить результаты, которые потом нужно будет воспроизвести, то важно перед построением модели зафиксировать стартовое значение.

Лекция 4.4. Важность предикторов

Случайный лес обладает возможностью оценивать важность отдельного предиктора с точки зрения улучшения классификации и прогнозирования. Первая мера важности – это усредненное уменьшение неоднородности, вторая – усредненное уменьшение правильности.

4.4.1. Важность предиктора на основе усредненного уменьшения неоднородности

В основе расчета важности переменных лежит критерий уменьшения неоднородности в узлах-потомках дерева.

В деревьях классификации оценивается уменьшение неоднородности распределения категорий зависимой переменной при разбиении родительского узла на узлы-потомки. Как уже говорилось в модуле 3, однородным узлом является тот, в котором все наблюдения относятся к одной и той же категории зависимой переменной, в то время как узел с максимальной неоднородностью содержит равное количество наблюдений во всех категориях зависимой переменной. Для расчета неоднородности в деревьях классификации используется уже знакомая мера Джини.

В деревьях регрессии под уменьшением неоднородности понимается уменьшение разброса значений зависимой переменной относительно среднего значения при разбиении родительского узла на узлы-потомки. Здесь уже вместо меры Джини используется среднеквадратичная ошибка. Если эти метрики трактовать с точки зрения неоднородности, то абсолютно однородным узлом является узел, в котором все наблюдения имеют одинаковые значения зависимой переменной, в то время как узлом с высоким значением неоднородности (в случае количественной зависимой переменной ограничения максимально возможного значения неоднородности не существует) является узел, включающий наблюдения с сильно различающимися значениями зависимой переменной. Уменьшение неоднородности еще называют улучшением.

Алгоритм вычисления важности предиктора на основе усредненного уменьшения неоднородности выглядит так:

1. Для каждого дерева случайного леса вычисляем сумму уменьшений неоднородности (улучшений) на всех ветвлениях, связанных с данным предиктором.
2. Итоговую сумму уменьшений неоднородности, полученную по ансамблю, усредняем путем деления на общее количество деревьев.
3. Вышеописанные шаги повторяем для всех остальных предикторов.
Наиболее важный предиктор – тот, который дает наибольшее усредненное уменьшение неоднородности (для деревьев классификации – уменьшение меры Джини, для деревьев регрессии – уменьшение среднеквадратичной ошибки).
Вновь вспомним про недостаток важности на основе уменьшения неоднородности. По сути важность складывается из частоты использования переменной в качестве предиктора разбиения, то есть наиболее важными будут переменные, по которым можем быть рассмотрено больше вариантов разбиения и у них больше шансов стать предиктором разбиения. Поэтому наиболее важными переменными чаще будут переменные с большим количеством уникальных значений.

4.4.2. Важность предиктора на основе усредненного уменьшения качества прогнозирования

Кроме уменьшения важности на основе усредненного уменьшения неоднородности Лео Брейман предложил алгоритм вычисления важности предиктора на основе усредненного уменьшения качества прогнозирования. Для задачи классификации вычисляем усредненное уменьшении правильности – количестве правильно классифицированных наблюдений от общего количества наблюдений. Для задачи регрессии вычисляется усредненное увеличение среднеквадратичной ошибки. Рассмотрим подробнее вычисление важности на основе усредненного уменьшения правильности.
1. Для каждого дерева классификации случайного леса берем out-of-bag выборку (наблюдения, не попавшие в бутстреп-выборку, по которой строилось данное дерево).
Допустим, у нас есть набор данных из 10 наблюдений. Наблюдение может принадлежать либо классу N, либо классу P. Мы построили ансамбль из 5 деревьев. По каждому из 5 деревьев получаем out-of-bag выборку. Например, для дерева I out-of-bag выборкой будут наблюдения 2, 4, 5, 6.

Исх. выборка	1	2	3	4	5	6	7	8	9	10
Фактический класс	**N**	**P**	**P**	**N**	**N**	**P**	**N**	**N**	**N**	**P**

											Out-of-bag выборки
Дерево I											
Б-выборка I	10	9	7	8	1	3	9	10	10	7	2 4 5 6
Дерево II											
Б-выборка II	4	8	5	8	3	9	2	6	1	6	7 10
Дерево III											
Б-выборка III	6	2	6	10	2	10	3	6	5	1	4 7 8 9
Дерево IV											
Б-выборка IV	6	7	8	10	6	10	9	10	8	2	1 3 4 5
Дерево V											
Б-выборка V	5	8	1	8	5	7	10	1	10	9	2 3 4 6

Рис. 4.19 Определение out-of-bag выборок

2. Для каждой out-of-bag выборки вычисляем правильность. Считаем количество раз, когда спрогнозированный класс для наблюдения out-of-bag выборки совпал с фактическим, и делим на размер out-of-bag выборки.
Например, для out-of-bag выборки 1 мы получаем 3 верных ответа и делим на 4 наблюдения, получаем правильность 3/4=0,75.

Out-of-bag выборка 1	2	4	5	6	
Фактический класс	**P**	**N**	**N**	**P**	
Спрогнозированный класс до пермутации	**P**	**P**	**N**	**P**	
Верные ответы до пермутации (отмечены X)	**X**		**X**	**X**	**правильность 3/4=0,75**

. . . . . .

Рис. 4.20 Вычисление правильности для каждой out-of-bag выборки

2. В каждой out-of-bag выборке осуществляем случайную перестановку значений предиктора и вычисляем правильность в каждой out-of-bag выборке с перестановленными значениями предиктора.
Например, для out-of-bag выборки 1 после пермутации мы получаем 2 верных ответа и делим на 4 наблюдения, получаем правильность 2/4=0,5.

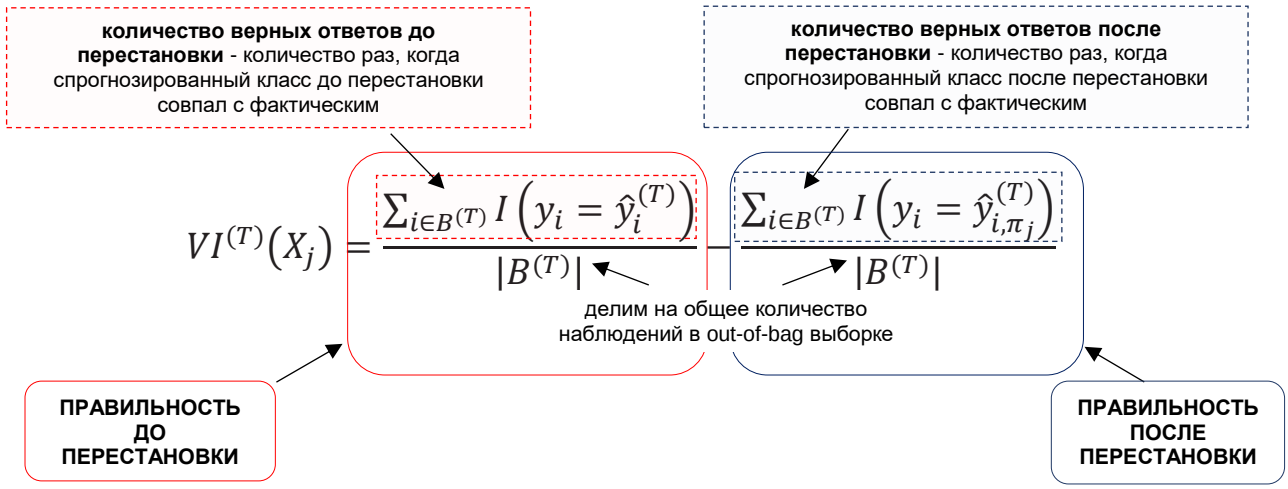
Out-of-bag выборка 1	2	4	5	6	
Фактический класс	**P**	**N**	**N**	**P**	
Спрогнозированный класс после пермутации	**N**	**P**	**N**	**P**	
Верные ответы после пермутации (отмечены X)			**X**	**X**	**правильность 2/4=0.5**

. . . . . .

Рис. 4.21 Вычисление правильности для каждой out-of-bag выборки после пермутации

4. Вычисляем разность между правильностью с исходными значениями предиктора и правильностью с перестановленными значениями предиктора в каждой out-of-bag выборке.
5. Суммируем разности по out-of-bag выборкам и делим на количество деревьев. Получаем сырое значение важности переменной.
6. Сырое значение важности переменной нормализуем путем деления на стандартную ошибку.
7. Повторяем шаги 2-5 для всех остальных предикторов.
На рис. 4.22 приводится математический аппарат вычисления важности на основе усредненного уменьшения правильности.

1. Вычисляем важность переменной как уменьшение правильности после перестановки в каждой out-of-bag выборке



где:
$VI^{(T)}(X_j)$ – это важность переменной X_j для дерева T ;
$|B^{(T)}|$ – это out-of-bag выборка для дерева T ;
y_i – фактический класс зависимой переменной;
$\hat{y}_i^{(T)}$ – спрогнозированный класс зависимой переменной перед перестановкой значений предиктора;
$\hat{y}_{i,\pi_j}^{(T)}$– спрогнозированный класс зависимой переменной после перестановки значений предиктора.
Обратите внимание, что $VI^{(T)}(X_j) = 0$, если X_j не находится в дереве T.

2. Вычисляем сырую важность переменной по ансамблю, просуммировав важности по всем out-of-bag выборкам и усреднив по всем деревьям

$$VI(X_j) = \frac{\sum_{T=1}^{N} VI^{(T)}(X_j)}{N}$$

3. Вычисляем нормализованную важность переменной по ансамблю

$$z_j = \frac{VI(X_j)}{\frac{\sigma}{\sqrt{N}}}$$

Рис. 4.22 Математический аппарат вычисления сырой и нормализованной важности предиктора на основе усредненного уменьшения правильности

Логическое объяснение алгоритма перестановки состоит в следующем: случайно переставляя значения предиктора, мы разрушаем взаимосвязь между ним и зависимой переменной. При использовании перестановленных значений предиктора (вместе с неперестановленными значениями остальных предикторов) для прогнозирования/ классификации по out-of-bag данным правильность существенно уменьшается (а среднеквадратичная ошибка существенно увеличивается), если между исходным предиктором и зависимой

переменной была взаимосвязь. Поэтому чем больше уменьшение правильности (увеличение среднеквадратичной ошибки), тем важнее предиктор. Важность на основе усредненного уменьшения правильности/увеличения среднеквадратичной ошибки в результате перестановок еще называют пермутированной важностью.
Обратите внимание, что при наличии высоко коррелированных предикторов обе метрики важности не способны определить релевантные переменные. Кроме того, обнаружив предикторы с небольшими значениями важностей, не спешите их удалять, посмотрите, как они будут работать, если включить дополнительные переменные. Необходимо понимать, что вычисляемая важность показывает, как данный предиктор работает не по отдельности, а в сочетании с другими переменными. Может оказаться, что предиктор, который является маловажным при совместном использовании с одними переменными, в сочетании с другими переменными станет важным.

Лекция 4.5. Графики частной зависимости

Несмотря на то что важность переменных несет ценную информацию, не меньший интерес представляет взаимосвязь предиктора с зависимой переменной. Метод частной зависимости довольно прост, основывается на прогнозах, получаемых с помощью случайного леса, и позволяет визуализировать взаимосвязь между зависимой переменной и предикторами. Основная идея заключается в том, чтобы изучить взаимосвязь между конкретным предиктором и зависимой переменной при условии, что все остальные предикторы остаются неизменными. Поскольку случайный лес может аппроксимировать практически любую функциональную зависимость между откликом и предиктором, с помощью графика можно обнаружить нелинейные зависимости, не выдвигая предварительных гипотез о характере взаимосвязей. Это особенно ценно, когда у нас отсутствует какая-либо априорная информация о виде распределения данных.
График частной зависимости строится следующим образом.
1. Для каждого значения интересующего предиктора создается специальный набор данных, в котором всем наблюдениям присваивается одно и то же значение интересующего предиктора, а все остальные предикторы фиксируются в своих текущих значениях. Например, если предиктор – это возраст в годах и есть 40 уникальных значений этой переменной, будет создано 40 специальных наборов данных, по одному для каждого уникального значения возраста. В каждом наборе во всех наблюдениях возраст принимает одно из 40 уникальных значений (например, все наблюдения получают значение возраста 21), независимо от того, так ли это на самом деле или нет. Остальные предикторы

фиксируются в своих текущих значениях. Важно понять, что значения остальных предикторов в специальных наборах не меняются и в этом смысле остаются неизменными.
2. Затем этот специальный набор, соответствующий конкретному значению интересующего предиктора (например, набор для значения возраста 26), прогоняется через случайный лес и получаем прогноз для каждого наблюдения набора.
3. Усредняем эти прогнозы по всем наблюдениям и получаем единый прогноз для набора в целом. Для количественной зависимой переменной таким прогнозом будет усредненное значение зависимой переменной. Для категориальной зависимой переменной прогнозом становится разность между логарифмом доли голосов, поданных деревьями за интересующий класс зависимой переменной, и усредненной суммой логарифмов голосов, поданных деревьями за каждый класс. Он вычисляется по формуле:

$$f(x) = log[p_k(x)] - \frac{1}{K}\sum_{j=1}^{K} log\left[p_j(x)\right]$$

где:
x – предиктор, для которого строится график частной зависимости;
K – количество классов;
k – интересующий класс;
p_j – доля голосов, поданных за класс j.

4. Повторяем шаги 2-3 для остальных значений интересующего предиктора.
5. Строим график, по оси абсцисс откладываем значения интересующего предиктора, по оси ординат – прогнозы.
6. Повторяем вышеописанные шаги для каждого предиктора.
Таким образом, график показывает, как значение интересующей переменной влияет на прогнозы модели при том, что все остальные переменные фиксируются и рассматриваются как константы.
Графики частной зависимости можно построить как для количественной, так и для категориальной зависимой переменной. Однако если в случае с количественной зависимой переменной значения, отложенные по оси y – это естественные значения зависимой переменной, то в случае с категориальной зависимой переменной значения отклика, отложенные по вертикальной оси, выглядят необычно и могут легко сбить с толку.

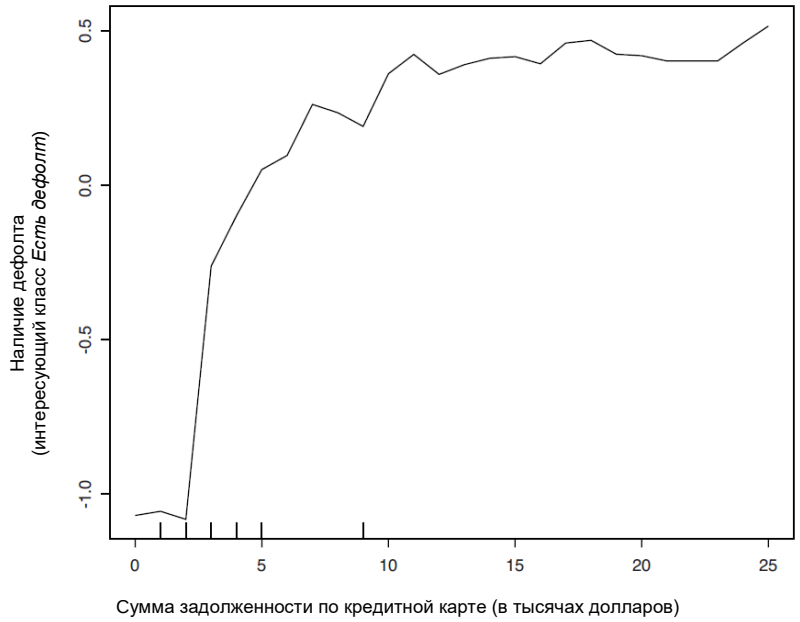


Рис. 4.23 График частной зависимости для категориальной зависимой переменной

Взгляните на рис. 4.23. Допустим, у нас есть предиктор *Сумма задолженности по кредитной карте (в тысячах долларов)* и бинарная зависимая переменная *Наличие дефолта*, которая принимает два значения *Нет дефолта* и *Есть дефолт*. Интересующим классом является класс *Есть дефолт*. Для значения суммы задолженности, равного 1, доля голосов, отданных за класс *Есть дефолт*, оказалась равна 0,11. Если доля голосов, поданных за класс *Есть дефолт*, равна 0,11, то значение, откладываемое по оси y, будет равно $\log(0{,}11)-[\log(0{,}11)+\log(0{,}89)]/2 = -1{,}04537$ (используются натуральные логарифмы). Точно такой же подход можно применить, чтобы понять какое значение по оси y будет отложено, если интересующим классом станет класс *Нет дефолта*. Доля голосов, поданных за класс *Нет дефолта*, равна 0,89, таким образом, значение, откладываемое по оси y, равно $\log(0{,}89)-[\log(0{,}89)+\log(0{,}11)]/2 = 1{,}04537$. В случае бинарной классификации график частной зависимости для одного класса зависимой переменной является зеркальным отражением графика частной зависимости для другого класса.

В бинарной классификации вычисления логитов не представляют трудностей. Значение, которое мы вычисляем с помощью вышеприведенного уравнения – это половина обычного логарифма отношения шансов. Поэтому легко получить привычные вероятности. Например, умножаем -1,04537 на 2, экспоненцирование дает нам отношение шансов, равное 0,1236. Теперь умножаем 0,1236 на 0,89 и получаем значение 0,11. Мы вернулись к тому, с чего начали.

Лекция 4.6. Матрица близостей

Близости – один и наиболее полезных инструментов случайного леса. Близость между двумя наблюдениями – это частота, с которой они оба попадают в один и тот же терминальный узел.

Метод вычисления близостей включает четыре этапа:

1. Сначала все близости приравниваются к нулю.
2. После того, как дерево построено, оно применяется ко всем наблюдениям (включая обучающую и out-of-bag выборку) и для каждой пары наблюдений вычисляются близости (и так по каждому дереву).
3. Если два наблюдения попадают в один и тот же терминальный узел, их близость увеличивается на единицу.
4. Оценки близости агрегируются по всем деревьям и нормализуются путем деления на общее количество деревьев.

В итоге получаем квадратную матрицу близостей N x N. Ниже приведена матрица близостей для первых 9 наблюдений (рис. 4.24). Значения по главной диагонали (единицы) – это «идеальные» близости наблюдений по отношению к самим себе (выделены серым фоном). Наблюдения, которые «схожи», имеют значения, близкие к 1 (выделены светло-зеленым фоном). Наблюдения, которые «непохожи», имеют значения, близкие к 0 (выделены светло-коричневым фоном).

	A	B	C	D	E	F	G	H	I	J
1	RECORD	X0000001	X0000002	X0000003	X0000004	X0000005	X0000006	X0000007	X0000008	X0000009
2	1	1	0.488	0.198	0.13	0.092	0.12	0.076	0.106	0.026
3	2	0.488	1	0.146	0.11	0.086	0.104	0.074	0.086	0.01
4	3	0.198	0.146	1	0.252	0.1	0.208	0.032	0.062	0.032
5	4	0.13	0.11	0.252	1	0.046	0.194	0.028	0.068	0.038
6	5	0.092	0.086	0.1	0.046	1	0.332	0.076	0.094	0.058
7	6	0.12	0.104	0.208	0.194	0.332	1	0.052	0.08	0.064
8	7	0.076	0.074	0.032	0.028	0.076	0.052	1	0.514	0.04
9	8	0.106	0.086	0.062	0.068	0.094	0.08	0.514	1	0.048
10	9	0.026	0.01	0.032	0.038	0.058	0.064	0.04	0.048	1
11										

Рис. 4.24 Матрица близостей для первых 9 наблюдений

Необходимо помнить, что с матрицей близостей удобно работать, когда наборы данных невелики. Например, если набор данных превышает более 5000 наблюдений, понадобится более 25 миллионов ячеек данных. Поэтому при обработке больших массивов обычно выводится «сжатая» форма матрицы близостей – для каждого наблюдения записывается лишь M ближайших наблюдений. M обычно меньше 100.

Близости между наблюдениями образуют матрицу сходств, которая симметрична, положительно определена, каждый элемент матрицы принимает значение в интервале от 0 до 1. Матрица сходств может быть использована для выполнения многомерного шкалирования.

Классическое многомерное шкалирование – это метод обучения без учителя, задача которого – поиск и интерпретация латентных (ненаблюдаемых) переменных, которые позволяют объяснить сходства между объектами, представленными в виде точек в пространстве низкой размерности. При этом объекты размещаются в пространстве таким образом, чтобы расстояния между точками были максимально близки изначально найденным мерам сходства этих объектов. На выходе мы получаем числовые значения координат по каждому объекту в некоторой новой системе координат, в которой оси соответствуют латентным переменным.
Кроме того, матрица близостей может использоваться для импутации пропущенных значений, обнаружения выбросов.

Лекция 4.7. Обработка пропущенных значений

В случайном лесе используются два метода импутации пропущенных значений.
В экономичном методе импутации пропущенные значения непрерывных переменных заменяются медианой, а пропущенные значения категориальных переменных – модой.
Оптимальный метод импутации, более затратный по времени, включает четыре этапа:
1. Сначала используется экономичный метод для первоначальной импутации пропущенных значений.
2. Затем по этому импутированному набору данных выращивается лес деревьев.
3. Затем матрица близостей, полученная с помощью леса деревьев, используется для итеративного изменения импутированных значений.
Если предиктор является количественным, импутированное значение для наблюдения – это взвешенное среднее всех наблюдений с непропущенными значениями, в качестве весов используются близости между данным наблюдением и наблюдением с непропущенным значением. Для категориальных предикторов импутированное значение – это наиболее часто встречающееся значение (категория), где частота взвешивается по близости.
4. Шаги 2 и 3 повторяются несколько раз до достижения сходимости. Обычно 4-6 итераций достаточно.
Если задана контрольная выборка, алгоритм запускает различные варианты импутации и определяется лучший кандидат-замена для пропущенного значения.

Лекция 4.8. Обнаружение выбросов

Выбросы можно определить как наблюдения, сильно отличающиеся от основной массы элементов выборки. В случайном лесе выброс – это наблюдение, которое имеет маленькие значения близостей по отношению ко всем остальным наблюдениям. Применительно к категориальным зависимым переменным выбросы определяются внутри категорий зависимой переменной. Таким образом, выброс в конкретной категории зависимой переменной – это наблюдение, у которого маленькие значения близостей по отношению ко всем остальным наблюдениям, принадлежащим к данной категории зависимой переменной. Вычисление показателя выброса происходит следующим образом.

1. Для конкретного наблюдения n_c, принадлежащей категории c, вычисляем сумму квадратов близостей до всех остальных наблюдений, относящихся к этой же категории. Берем обратную величину суммы квадратов близостей.

$$out_{raw}(n_c) = \frac{1}{\sum_c [prox(n_c, k)]^2}$$

Показатель выброса будет большим при малом значении знаменателя. Повторяем ту же самую процедуру для всех остальных наблюдений в этой категории. Будем считать полученные значения нестандартизированными.

2. Вычисляем медиану нестандартизированных значений для категории c и среднее абсолютное отклонение относительно медианы нестандартизированных значений для категории c.

3. Вычитаем медиану из каждого нестандартизированного значения и делим на среднее абсолютное отклонение. Таким образом, стандартизируем нестандартизированные значения.

$$out(n_c) = \frac{out_{raw}(n_c) - median(c)}{deviation(c)}$$

4. Значения меньше нуля приравниваются к 0.

Вышеперечисленные шаги повторяем для всех остальных категорий зависимой переменной. Наблюдения со значениями больше 10 рассматриваются в качестве выбросов.

Лекция 4.9. Построение случайного леса на несбалансированном наборе данных

Очень многие практические задачи классификации являются несбалансированными, то есть, по меньшей мере, один из классов содержит очень небольшое количество наблюдений. При работе с такими задачами нас интересует правильная классификация редко встречающего класса (положительного класса). Примерами таких задач являются обнаружение мошенничества, прогнозирование дефолта и диагностика редких заболеваний. Однако наиболее часто используемые алгоритмы классификации плохо решают такие задачи, потому что они направлены на минимизацию общей ошибки классификации, а не на минимизацию ошибки классификации положительных наблюдений.

4.9.1 Два подхода к решению проблемы дисбаланса классов: присвоение весов и семплинг

Существует два распространенных подхода к решению проблемы крайне несбалансированных данных.

4.9.1.1. Присвоение весов

Первый подход предлагает использовать в ходе обучения разные стоимости ошибочной классификации. Ошибкам отнесения к классам зависимой переменной мы можем назначить разные цены в зависимости от ценности категории. Например, предположим, что пациент относится к одному из двух классов: *Здоров* (отрицательный класс) и *Болен* (положительный класс). Ошибочное отнесение больного пациента к классу *Здоров*, вероятно, является ошибкой, имеющей более высокую цену, чем ошибочное отнесение здорового пациента к классу *Болен*. Итак, мы определяем, что стоимость ошибочной классификации наблюдений миноритарного класса выше и пытаемся минимизировать общую стоимость ошибочной классификации.

4.9.1.2. Семплинг

Второй подход заключается в использовании семплинга. Мы можем удалить некоторое количество примеров мажоритарного класса (данную технику называют андерсемплингом) или увеличить количество примеров миноритарного класса (эта техника называется оверсемплингом). Удалить примеры мажоритарного класса или увеличить количество примеров миноритарного класса можно случайным образом или по специальным правилам.

4.9.1.2.1. Случайный андерсемлинг

В рамках случайного андерсемплинга мы вычисляем K – количество мажоритарных примеров, которое необходимо удалить для достижения требуемого уровня соотношения различных классов. Затем случайным образом мы отбираем K мажоритарных примеров и удаляем.

Примеры миноритарного класса могут удаляться не только случайным образом, но и по определенным правилам. Одной из таких стратегий являются связи Томека.

4.9.1.2.2. Связи Томека

Пусть примеры E_i и E_j принадлежат различным классам, $d(E_i, E_j)$ – расстояние между указанными примерами. Пара (E_i, E_j) называется связью Томека, если не найдется ни одного примера E_l такого, что будет справедлива система неравенств:

$$\begin{cases} d(E_i, E_l) < d(E_i, E_j) \\ d(E_j, E_l) < d(E_i, E_j) \end{cases}$$

Согласно данному подходу, все мажоритарные записи, входящие в связи Томека, должны быть удалены из набора данных. Этот способ хорошо удаляет записи, которые можно рассматривать в качестве «зашумляющих».

4.9.1.2.3. Случайный оверсемлинг

В рамках случайного оверсемплинга мы случайным образом дублируем примеры миноритарного класса. В зависимости от того, какое соотношение классов требуется, выбирается необходимое количество случайных записей для дублирования.

4.9.1.2.4. SMOTE

SMOTE (Syntetic Minority Oversampling Technique – оверсемплинг за счет создания синтетических примеров миноритарного класса) генерирует синтетические примеры миноритарного класса, чтобы увеличить долю миноритарного класса в выборке. Для каждого примера миноритарного класса мы вычисляем его k ближайших соседей (обычно k=5) и затем из них случайным образом выбираем некоторые примеры в соответствии с нашим уровнем оверсемплинга. После этого вдоль линий, соединяющих пример миноритарного класса с его выбранными ближайшими соседями генерируются новые синтетические примеры.

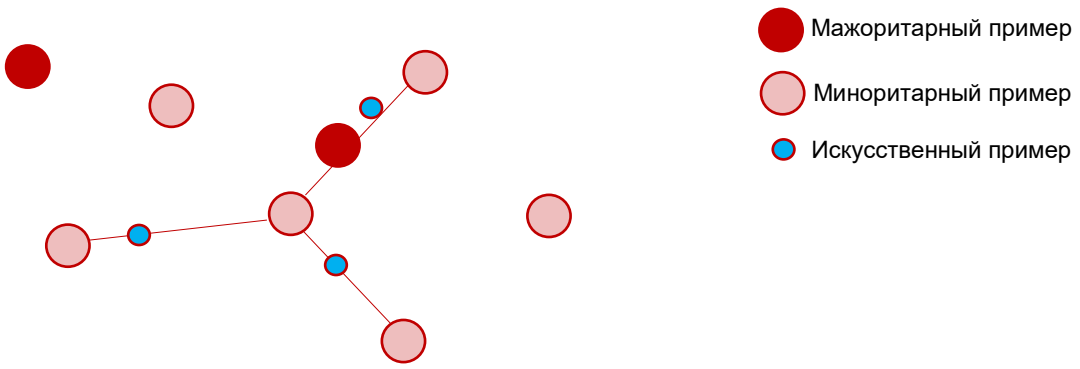


Рис. 4.25 Алгоритм SMOTE

В отличие от случайного оверсамплинга, реализованного в классическом SMOTE, модифицированный SMOTE (borderline-SMOTE1 и borderline-SMOTE2) выполняет отбор и усиление пограничных примеров миноритарного класса (примеров миноритарного класса, расположенных на границе принятия решений). Сначала мы находим пограничные примеры миноритарного класса, затем генерируем из них синтетические примеры и затем добавляем в исходную обучающую выборку.
Пусть весь обучающий набор – это T, миноритарный класс – P, а мажоритарный класс – N и тогда

$$P = \{p_1, p_2, \dots, p_{pnum}\}, N = \{n_1, n_2, \dots, n_{nnum}\}$$

где $pnum$ и $nnum$ – это количество примеров миноритарного и мажоритарного классов. Тогда детальная процедура borderline-SMOTE1 выглядит так:
Этап 1. Для каждого примера $p_i\{i = 1,2, \dots, pnum\}$ в миноритарном классе P мы вычисляем m ближайших соседей из всей обучающей выборки T. Количество примеров мажоритарного классов, попавших в m ближайших соседей, помечается как $m'(0 \leq m' \leq m)$.
Этап 2. Если $m'=m$, т.е. все m ближайших соседей примера p_i являются примерами мажоритарного класса, пример p_i рассматривается как зашумленный и в дальнейших этапах не участвует. Если $m/2 \leq m' < m$, т.е. если для примера p_i среди его ближайших соседей больше половины – это примеры мажоритарного класса, то пример p_i рассматривается как пример, имеющий риск неправильной классификации, и попадает в набор под названием $DANGER$ («опасная зона»). Если $0 \leq m' < m / 2$, т.е. если для примера p_i среди его ближайших соседей больше половины – это примеры миноритарного класса, то такой пример считается надежным и не участвует в следующих этапах.

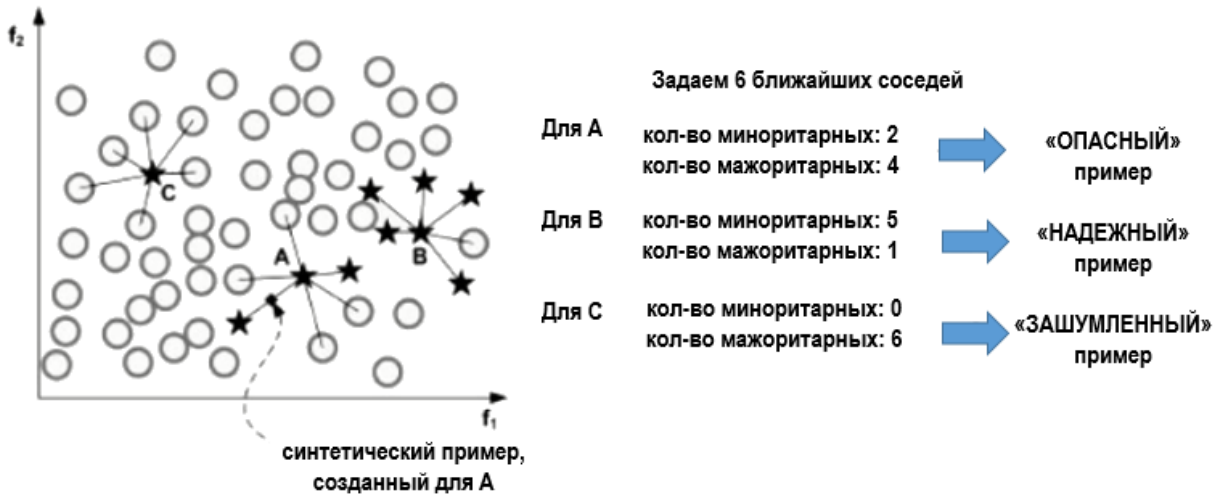


Рис. 4.26 Три типа примеров в алгоритме SMOTE

Этап 3. Примеры, попавшие в набор *DANGER,* являются пограничными примерами миноритарного класса P и мы можем увидеть, что каждый пример из набора *DANGER*, также является примером миноритарного класса.

Мы задаем

$$DANGER = \{p'_1, p'_2, \dots, p'_{dnum}\}, \qquad 0 \le dnum \le pnum$$

Для каждого примера, входящего в набор *DANGER*, мы вычисляем k ближайших соседей из P.

Этап 4. На этом этапе мы генерируем $s \times dnum$ синтетических примеров миноритарного класса из примеров, вошедших в набор *DANGER*, где s – это целое число от 1 до k. Для каждого p'_i мы случайно отбираем s ближайших соседей из k ближайших соседей в P. Сначала мы вычисляем разности $diff_j (j = 1,2, \dots, s)$ между p'_i и s ближайшими соседями из P, затем умножаем $diff_j$ на случайное число $r_j (j = 1,2, \dots, s)$, лежащее в интервале от 0 до 1, в итоге генерируем s новых синтетических примеров миноритарного класса, расположенных между p'_i и его ближайшими соседями:

$$synthetic_j = p'_i + r_j \times dif_j, \;\; j = 1,2, \dots, s$$

Мы повторяем вышеприведенную процедуру для каждого примера p'_i, вошедшего в набор *DANGER*, и можем получить $s \times dnum$ синтетических примеров.

В процедуре выше p_i, n_i, p'_i, dif_j и $synthetic_j$ являются векторами. Мы видим, что новые синтетические данные сгенерированы вдоль линий, соединяющих пограничные примеры миноритарного класса с их ближайшими соседями – примерами того же класса, таким образом, происходит усиление этих пограничных примеров.

Borderline-SMOTE2 не только генерирует синтетические примеры на основе каждого примера, вошедшего в набор *DANGER*, и его ближайших соседей – примеров миноритарного класса *P*, но и делает то же самое, используя ближайших соседей – примеры мажоритарного класса *N*. Разность между p_i' и его ближайшим соседом – примером мажоритарного класса умножается на случайное число, лежащее в интервале от 0 до 0,5, таким образом, новые сгенерированные примеры располагаются ближе к миноритарному классу.
Обратите внимание, что присвоение весов и любой семплинг, будь то андерсемплинг или оверсемплинг, случайный или по определенным правилам, мы осуществляем только на обучающей выборке, а затем смотрим, как модель, обученная по взвешенным/семлированным данным, работает на контрольных данных, не подвергшихся взвешиванию/семплингу. Мы должны помнить, что когда мы будем работать с новыми данными, у нас не будет никакой зависимой переменной и мы не можем вновь применить взвешивание/семплинг, у нас есть лишь модель, построенная на взвешенных/семплированных обучающих данных.

4.9.2 Сбалансированный случайный лес и взвешенный случайный лес

На основе двух вышеописанных подходов (присвоения весов и семплинга) Чао Чен, Энди Лиав и Лео Брейман предложили два метода для решения проблемы дисбаланса классов. Первый метод, получивший название сбалансированный случайный лес (balanced random forest или BRF), сочетает технику семлинга с ансамблированием. Он выполняет случайное удаление наблюдение мажоритарного класса и строит каждое дерево на более сбалансированных данных, для прогноза используется мажоритарное голосование. Второй метод, получивший название взвешенный случайный лес (weighted random forest или WRF), заключается в том, чтобы использовать в случайном лесе веса классов, таким образов, сделав случайный лес чувствительным к стоимости классификации и штрафуя его за неправильную классификацию наблюдений.

4.9.2.1. Сбалансированный случайный лес

Случайный лес строит каждое дерево по бутстреп-выборке. При работе с крайне несбалансированными данными возникает значительная вероятность того, что бутстреп-выборка будет содержать мало наблюдений из миноритарного класса или вообще не будет содержать их, что приводит к плохому качеству прогнозированию миноритарного класса. Самый простой подход – использование стратифицированного

бутстрепа. Если простой бутстреп генерирует выборки с учетом равной вероятности появления каждого объекта, то стратифицированный бутстреп учитывает соотношение частот между относительно гомогенными группами (стратами), на которые могут быть разделены выборочные объекты. В роли страт выступают классы зависимой переменной. Однако этот подход не решает проблемы полностью. Однако этот подход не решает проблемы полностью. В ходе экспериментов было показано, что задав равные априорные вероятности классов, использовав случайное удаление примеров мажоритарного класса или случайное дублирование примеров миноритарного класса можно добиться лучшего качества.

Лео Брейман, Энди Лиав и Чао Чень скомбинировали идею андерсемплинга и ансамблирования, предложив алгоритм сбалансированного случайного леса:

1. На каждой итерации случайного леса извлекаем бутстреп-выборку из миноритарного класса. Случайным образом извлекаем такое же количество наблюдений с возвращением из мажоритарного класса.
2. Строим полное дерево CART по полученным данным. В каждом узле используем для поиска оптимальной точки расщепления не все предикторы, а mtry случайно отобранных предикторов.
3. Повторяем вышеописанные шаги нужное количество раз. Агрегируем прогнозы и получаем итоговый прогноз для каждого наблюдения.

4.9.2.2. Взвешенный случайный лес

Поскольку случайный лес, как и большинство остальных классификаторов, имеет тенденцию минимизировать ошибку классификации мажоритарного класса, мы должны ввести более высокий штраф за ошибочную классификацию наблюдений миноритарного класса. Каждому классу мы присваиваем вес, при этом меньшему по размеру классу присваивается больший вес (т. е. задаем более высокую стоимость ошибочной классификации). Веса классов в алгоритме случайного леса используются два раза. В ходе построения дерева веса классов используются при взвешивании критерия Джини для поиска точек расщепления. В терминальных узлах каждого дерева веса классов вновь принимаются во внимание. Прогнозирование класса в каждом терминальном узле осуществляется взвешенным большинством голосов, где взвешенная доля класса – это вес класса, умноженный на количество наблюдений из этого класса в терминальном узле. Итоговый прогноз определяется путем агрегирования взвешенных долей классов, взятых по каждому отдельному дереву. Вес класса – это необходимый параметр настройки для достижения желаемого качества. Для подбора весов может использоваться контрольная выборка или контрольные блоки перекрестной проверки.

4.9.3. Присвоение весов и семплинг в современных реализациях случайного леса для R и Python

Необходимо отметить, что в современных реализациях случайного леса для R и Python методы сбалансированного случайного леса и взвешенного случайного леса не были реализованы полностью. Это обусловлено тем, что к каждой конкретной задаче с дисбалансом классов нужно подходить индивидуально и здесь не может быть универсального метода.

В пакете R `randomForest` мы можем выполнить семплинг с помощью параметров `strata` и `samplesize`. Параметр `strata` задает переменную-фактор (обычно зависимая переменная), которая используется для стратифицированного семплинга, то есть классы зависимой переменной и будут стратами. Параметр `samplesize` задает количество случайно извлекаемых наблюдений для каждого класса. Допустим, у нас есть 200 наблюдений, 150 наблюдений относится к классу *NoResponse* и 50 наблюдений относится классу *Response*. Мы задаем следующий программный код:

```
model<-randomForest(response ~., development,
                    strata=development$response, sampsize=c(100, 50))
```

или

```
model<-randomForest(response ~., development,
                    strata=development$response,
                    sampsize=c('NoResponse'=100, 'Response'=50))
```

Это означает, что переменной-стратой будет наша зависимая переменная *response* и мы случайным образом с возвращением отбираем 100 примеров из класса *NoResponse* и 50 примеров из класса *Response* для построения каждого дерева. Поскольку отбор с возвращением, наблюдения каждого класса могут повторяться. Отбор с возвращением задается значением `TRUE` для параметра `replace` (значение `TRUE` используется по умолчанию). Если для параметра `replace` задать значение `FALSE`, будет осуществлен отбор без возвращения. Отбор без возвращения в ряде случаев может дать лучшее качество, потому что при отборе с возвращением примеры миноритарного класса будут повторяться чаще и миноритарный класс может быть по-прежнему недопредставлен. Если данный подход ведет к получению выборок небольшого размера, можно увеличить значение mtry, иногда даже доведя его до общего количества предикторов.

Обратите внимание, `sampsize` не может быть больше частоты класса. Поэтому этот параметр чаще всего используют для андерсемплинга, когда количество наблюдений в мажоритарном классе уменьшают, а

количество наблюдений в миноритарном классе оставляют без изменений.

Веса классов в пакете R `randomForest` можно задать с помощью параметра `classwt`. Значением параметра будет вектор с весами классов. Веса равны априорным вероятностям классов. Допустим, у нас есть два класса *Нет дефолта* и *Есть дефолт* и мы задаем следующий программный код:

```
set.seed(152)
model<-randomForest(default ~., development,
                                classwt = c(0.75, 0.25))
```

Это обозначает, что вес класса *Нет дефолта* равен 0,75, а вес класса *Есть дефолт* равен 0,25. Априорные вероятности классов равны 0,75 и 0,25. Мы исходим из того, что если наш пример содержит 1000 наблюдений, 750 наблюдений принадлежат классу *Нет дефолта*, а 250 наблюдений принадлежат классу *Есть дефолт*. Обратите внимание, что веса вовсе не обязательно должны быть в сумме равны 1, поскольку алгоритмом предусмотрена нормализация.

В пакете R `ranger` специальных параметров для семплинга не предусмотрено. Веса классов настраиваются с помощью параметра `case.weights`. Сначала задают вектор весов, у которого длина будет равна количеству наблюдений, а затем для каждого класса задают вес, который определяет вероятность отбора в выборку с возвращением или без возвращения.

```
weights <- rep(NA, nrow(development))
weights[development$response == "NoResponse"] <- 0.7
weights[development$response == "Response"] <- 0.3
model <- ranger(response~., case.weights = weights, development,
               probability=TRUE, seed=152)
```

Как и в пакете `randomForest`, в пакете `ranger` с помощью параметра `replace` можно задать отбор с возвращением (значение `TRUE`, используется по умолчанию) и отбор без возвращения (значение `FALSE`).

В питоновской библиотеке `scikit-learn` семплинг выполняется с помощью различных классов библиотеки `imbalanced-learn`. Если вы используете дистрибутив Anaconda for Python, эту библиотеку можно установить в Anaconda Promt с помощью команды `conda install -c conda-forge imbalanced-learn`. Например, у нас 10656 наблюдений, 9426 наблюдений принадлежат классу *Не откликнулся*, 1230 наблюдений принадлежат классу *Откликнулся* и нам нужно выполнить случайный андерсемплинг.

Для начала нам нужно импортировать класс `RandomUnderSampler` библиотеки `imbalanced-learn`.

```
# импортируем класс RandomUnderSampler
from imblearn.under_sampling import RandomUnderSampler
```

Затем создаем экземпляр класса `RandomUnderSampler`, настроив параметры. Главным параметром является `ratio`. Параметр `ratio` задает соотношение числа объектов в миноритарном и мажоритарном классах. Для него задают значение `majority` и тогда выполняется случайное удаление примеров мажоритарного класса, еще можно передать словарь, ключами будут классы, а значениями – количество наблюдений. Например, если задать `ratio={0:9000, 1:1230}`, то применительно к нашему случаю мы оставляем все примеры миноритарного класса (1230), а из исходных 9426 наблюдений мажоритарного класса удаляем 426 наблюдений, оставив 9000 наблюдений. Как правило, наилучшее качество модели дает тонкая настройка андерсемплинга с помощью словаря. Также не забываем про параметр `random_state`, который задает стартовое значение генератора случайных чисел для воспроизводимости результатов.

```
# создаем экземпляр класса RandomUnderSampler, в рамках нашей модели
# мы случайно удалим 426 наблюдений мажоритарного класса
rus = RandomUnderSampler(ratio={0:9000, 1:1230}, random_state=42)
```

Затем подгоняем нашу модель и выполняем андерсемплинг с помощью метода `fit_sample`, создав новый массив признаков и новый массив меток для обучения.

```
# подгоняем модель RandomUnderSampler и выполняем андерсемплинг,
# создав новый массив признаков и новый массив меток для
# обучения
X_undersampled, y_undersampled = rus.fit_sample(X_train, y_train)
```

И вот на этих новых массивах мы обучаем модель случайного леса и проверяем ее на контрольной выборке, контрольных блоках перекрестной проверки и т.д. В данном случае для проверки используется оценка AUC на контрольной выборке.

```
# создаем экземпляр класса RandomForestClassifier (модель случайного леса)
# и подгоняем модель на новых массивах признаков и меток
under_forest = RandomForestClassifier(random_state=42).fit(X_undersampled,
                                                           y_undersampled)
# импортируем функцию roc_auc_score
from sklearn.metrics import roc_auc_score
# вычисляем и печатаем значение AUC для обучающей выборки
print("AUC на обучающей выборке: {:.3f}".
      format(roc_auc_score(y_undersampled,
                           under_forest.predict_proba(X_undersampled)[:, 1])))
# вычисляем и печатаем значение AUC для контрольной выборки
print("AUC на контрольной выборке: {:.3f}".
      format(roc_auc_score(y_test,
                           under_forest.predict_proba(X_test)[:, 1])))
```

Если мы хотим выполнить не просто случайный андерсемплинг, а андерсемплинг по определенным правилам, можно воспользоваться классом `TomekLinks`. Здесь также есть параметр `ratio`, задающий соотношение числа объектов в миноритарном и мажоритарном классах. Параметр `random_state` задает стартовое значение генератора случайных чисел для воспроизводимости результатов. Поскольку вычисление мажоритарных примеров, входящих в связи Томека, является затратной процедурой с вычислительной точки зрения, то с помощью параметра `n_jobs` можно задать количество ядер процессора для распараллеливания вычислений.

```
# создаем экземпляр класса TomekLinks, в рамках нашей
# модели мы удалим мажоритарные примеры, входящие
# в связи Томека
tomek = TomekLinks(ratio="majority", random_state=42, n_jobs=-1)
# подгоняем модель TomekLinks и выполняем андерсемплинг,
# создав новый массив признаков и новый массив меток
# для обучения
X_tomek, y_tomek = tomek.fit_sample(X_train, y_train)
# создаем экземпляр класса RandomForestClassifier (модель случайного леса)
# и подгоняем модель на новых массивах признаков и меток
tomek_forest = RandomForestClassifier(random_state=42).fit(X_tomek,
                                                           y_tomek)
# вычисляем и печатаем значение AUC для обучающей выборки
print("AUC на обучающей выборке: {:.3f}".
      format(roc_auc_score(y_tomek,
                           tomek_forest.predict_proba(X_tomek)[:, 1])))
# вычисляем и печатаем значение AUC для контрольной выборки
print("AUC на контрольной выборке: {:.3f}".
      format(roc_auc_score(y_test,
                           tomek_forest.predict_proba(X_test)[:, 1])))
```

Для выполнения случайного оверсемплинга нужно воспользоваться классом `RandomOverSampler`. Главным является параметр `ratio`. Для него задают значение `minority` и тогда выполняется случайное дублирование примеров миноритарного класса, еще можно передать словарь, ключами будут классы, а значениями – количество наблюдений. Например, если задать `ratio={0:9426, 1:1300}`, то применительно к нашему случаю мы оставляем все примеры мажоритарного класса (9426), а из исходных 1230 наблюдений миноритарного класса создаем 1300 наблюдений путем дублирования. Наилучшее качество обычно дает тонкая настройка с помощью словаря. Не забываем про параметр `random_state` для получения воспроизводимых результатов.

```
# создаем экземпляр класса RandomOverSampler, в рамках нашей модели
# мы случайно создадим 70 новых наблюдений мажоритарного класса
ros = RandomOverSampler(ratio={0:9426, 1:1300}, random_state=42)
# подгоняем модель RandomOverSampler и выполняем андерсемплинг,
# создав новый массив признаков и новый массив меток для
# обучения
X_oversampled, y_oversampled = ros.fit_sample(X_train, y_train)
```

```
# создаем экземпляр класса RandomForestClassifier (модель случайного леса)
# и подгоняем модель на новых массивах признаков и меток
over_forest = RandomForestClassifier(random_state=42).fit(X_oversampled,
                                                          y_oversampled)
# вычисляем и печатаем значение AUC для обучающей выборки
print("AUC на обучающей выборке: {:.3f}".
      format(roc_auc_score(y_oversampled,
                           over_forest.predict_proba(X_oversampled)[:, 1])))
# вычисляем и печатаем значение AUC для контрольной выборки
print("AUC на контрольной выборке: {:.3f}".
      format(roc_auc_score(y_test,
                           over_forest.predict_proba(X_test)[:, 1])))
```

Наконец, мы можем применить SMOTE. Нам понадобится класс `SMOTE`. Здесь помимо параметров `ratio`, `random_state` используется ряд дополнительных параметров. Параметр `kind` определяет тип алгоритма SMOTE. Можно задать значения `regular` (по умолчанию), `borderline1`, `borderline2`, `svm`. Чаще всего значения `borderline1` и `borderline2` дают лучшее качество. Параметр `k_neighbors` задает количество ближайших соседей для создания синтетических примеров (по умолчанию используется значение 5). Параметр `m_neighbors` задает количество ближайших соседей, которое требуется для идентификации миноритарного примера как «опасного». Параметр `m_neighbors` используется только со значениями `kind={'borderline1', 'borderline2', 'svm'}`. С помощью параметра `n_jobs` можно задать количество ядер процессора для распараллеливания вычислений. Применительно к нашему примеру мы используем Borderline-SMOTE2.

```
# создаем экземпляр класса SMOTE
smote = SMOTE(random_state=152, kind='borderline2', k_neighbors=5, n_jobs=-1)
# подгоняем модель SMOTE и выполняем SMOTE,
# создав новый массив признаков и новый массив меток для
# обучения
X_smote, y_smote = smote.fit_sample(X_train, y_train)
# создаем экземпляр класса RandomForestClassifier (модель случайного леса)
# и подгоняем модель на новых массивах признаков и меток
smote_forest = RandomForestClassifier(random_state=42).fit(X_smote,
                                                           y_smote)
# вычисляем и печатаем значение AUC для обучающей выборки
print("AUC на обучающей выборке: {:.3f}".
      format(roc_auc_score(y_smote,
                           smote_forest.predict_proba(X_smote)[:, 1])))
# вычисляем и печатаем значение AUC для контрольной выборки
print("AUC на контрольной выборке: {:.3f}".
      format(roc_auc_score(y_test,
                           smote_forest.predict_proba(X_test)[:, 1])))
```

Для экземпляра класса `RandomForestClassifier` питоновской библиотеки `scikit-learn` веса классов можно задать с помощью параметра `class_weight`. В качестве значения могут выступать словарь, список из словарей, значение `'balanced'`, `'balanced_subsample'` или `None`. По умолчанию используется значение `None`. Вес класса берется из словаря `{метка_класса: вес}`. Если словарь не задан, вес каждого класса равен 1. Допустим, у нас есть два класса *Не откликнулся* и *Откликнулся* и мы задаем следующий программный код:

```
forest=RandomForestClassifier(n_estimators=100,
                              class_weight={0:0.80, 1:0.20},
                              random_state=152)
forest.fit(X_train, y_train)
```

Это обозначает, что вес класса *Не откликнулся* равен 0,80, а вес класса *Откликнулся* равен 0,20. Вновь обратите внимание, что веса вовсе не обязательно должны быть в сумме равны 1, поскольку алгоритмом предусмотрена нормализация.

Режим `'balanced'` использует значения зависимой переменной `y`, чтобы автоматически настроить веса, обратно пропорциональные частотам встречаемости класса, используя формулу $w_j = \frac{n}{kn_j}$,

где:

w_j – вес класса j;

n – это общее количество наблюдений;

n_j – это количество наблюдений в классе j;

k – это общее количество классов.

Например, у нас 10656 наблюдений, 9426 наблюдений принадлежат классу *Не откликнулся*, 1230 наблюдений принадлежат классу *Откликнулся*. Если задать режим `'balanced'`, получаем вес класса *Не откликнулся* $\frac{10656}{2\times9426} = 0{,}565$ и вес класса Откликнулся $\frac{10656}{2\times1230} = 4{,}332$.

Режим `'balanced_subsample'` идентичен режиму `'balanced'`, за исключением того, что веса вычисляются для каждой бутстреп-выборки, по которой строилось дерево.

Отметим, что изменение пропроций классов с помощью весов и семплинга – это самый последний инструмент в арсенале средств аналитика. Проблему дисбаланса классов нужно попытаться решить сперва за счет конструирования новых признаков, обогащения данными из других источников, настройки параметров и ансамблирования моделей. И только когда все традиционные способы улучшения качества модели уже испробованы, можно применить специальные техники устранения дисбаланса классов.

Лекция 4.10. Преимущества и недостатки случайного леса

Как уже говорилось в самом начале этой главы, использование ансамбля по сравнению с одиночным деревом решений дает более лучшее качество модели за счет усреднения результатов по-разному переобучающихся деревьев. Случайный лес легко настраивается, для получения модели хорошего качества требуется лишь настроить количество деревьев и количество случайно отбираемых переменных для разбиения. Другим безусловным преимуществом метода является тот факт, что случайный лес способен более точно оценить вклад и поведение каждого предиктора, даже когда эффект одного предиктора ослаблен более значимыми предикторами, что характерно для регрессионных моделей. Как и деревья решений, случайный лес не требует процедуры предварительной подготовки данных (в частности, не нужно масштабировать данные). Случайный лес можно легко распараллелить между несколькими ядрами процессора в компьютере. С помощью случайного леса можно решать задачи подготовки данных и отбора переменных, например, определить наиболее важные переменные для включения в модель логистической регрессии или импутировать пропущенные значения.

Вместе с тем случайный лес является моделью «черного ящика», мы не можем сразу взглянуть на несколько сотен деревьев и интерпретировать их. Он не дает непосредственной информации о том, как меняется значение отклика в зависимости от значения того или иного предиктора. Судить о взаимосвязях между предиктором и зависимой переменной можно лишь по графикам частной зависимости, приняв во внимание ряд ограничений. Случайный лес плохо работает на данных очень высокой размерности, разреженных данных, примером которых являются текстовые данные. Для подобного рода данных линейные модели подходят больше. Как и дерево решений, случайный лес не умеет экстраполировать данные. Случайный лес склонен к переобучению при работе с сильно зашумленными данными. Кроме того, нельзя не отметить большой размер получаемых моделей случайного леса.