Оглавление

Введение

Настоящая книга является коллекцией избранных материалов из первого модуля Подписки – обновляемых в режиме реального времени материалов по применению классических методов машинного обучения в различных промышленных задачах, которые автор делает вместе с коллегами и учениками.

Автор благодарит Дмитрия Ларько за помощь в подготовке раздела по конструированию признаков в третьей части книги, Уилла Керсена за предоставленные материалы к первому разделу пятой части книги.

Во втором томе мы разберем собственно процесс предварительной подготовки данных, обсудим некоторые метрики качества, рассмотрим ряд полезных библиотек и фреймворков.

Часть 3

План предварительной подготовки данных

1. Введение

До этого момента мы знакомились с инструментами – основными питоновскими библиотеками, классами и функциями, необходимыми для предварительной подготовки данных и построения моделей машинного обучения. Мы брали относительно простые примеры, выполняли предварительную подготовку данных и строили модели машинного обучения без глубокого понимания, зачем нужна та или иная операция предварительной подготовки и что происходит «под капотом» этой операции. В реальной практике мы так действовать не можем, нам нужен четкий план действий и глубокое понимание каждого этапа.

План предварительной подготовки данных, как правило, будет состоять из двух этапов. Первый этап – операции, которые можно выполнить до разбиения на обучающую и тестовую выборки / до цикла перекрестной проверки. Второй этап – операции, которые можно выполнить только после разбиения на обучающую и тестовую выборки / внутри цикла перекрестной проверки.

Если используются операции, использующие статистики, например укрупнение редких категорий по порогу, импутация пропусков статистиками, стандартизация, биннинг и конструирование признаков на основе статистик (frequency encoding, likelihood encoding), они должны быть осуществлены после разбиения на обучающую и тестовую выборки или внутри цикла перекрестной проверки.

Если мы используем случайное разбиение на обучающую и тестовую выборки и выполняем перечисленные операции до разбиения, получается, что для вычисления среднего и стандартного отклонения по каждому признаку для стандартизации, правил биннинга, частот и вероятностей положительного класса зависимой переменной в категориях признака использовались все наблюдения набора, часть из которых потом у нас войдут в тестовую выборку (по сути, выборку новых данных).

Если мы используем перекрестную проверку и выполняем перечисленные операции до перекрестной проверки, получается, что в каждом проходе перекрестной проверки для вычисления среднего и стандартного отклонения по каждому признаку для стандартизации, правил биннинга, частот и вероятностей положительного класса зависимой переменной в категориях признака использовались

все наблюдения набора, часть из которых у нас теперь находится в тестовом блоке (по сути, выборке новых данных). В таких случаях в Python используем классы `ColumnTransformer` и `Pipeline`. Случайное разбиение на обучающую и тестовую выборки и перекрестная проверка используются для сравнения конвейеров базовых моделей со значениями гиперпараметров по умолчанию. При подборе гиперпараметров лучшей практикой является комбинированная проверка, сочетающая случайное разбиение на обучающую и тестовую выборки и перекрестную проверку.

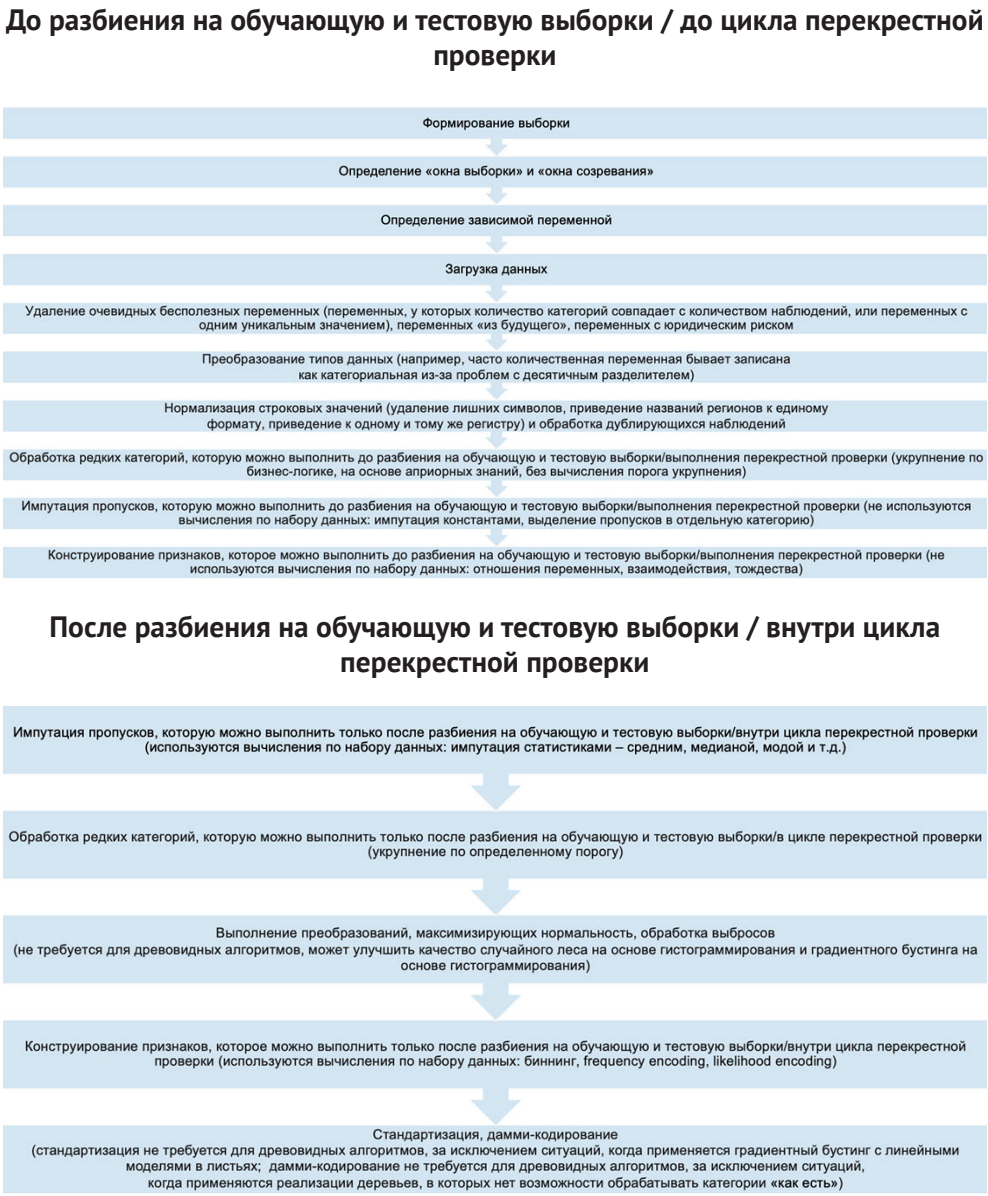


Рис. 1 План предварительной подготовки данных

2. Формирование выборки

2.1. Генеральная и выборочная совокупности

Генеральная совокупность, *или* **популяция (population)** – совокупность всех объектов (единиц), относительно которых предполагается делать выводы при изучении конкретной задачи.

Генеральная совокупность состоит из всех объектов, которые имеют качества, свойства, интересующие исследователя. Например, в исследованиях телезрительской аудитории, проводимых компанией Mediascope, генеральной совокупностью будет население России в возрасте 4 лет и старше, проживающее в городах от 100 000 и более. А в исследованиях читательской аудитории, проводимых этой же компанией, генеральной совокупностью будет уже население России в возрасте 16 лет и старше, проживающее в городах от 100 000 и более. В исследованиях политических предпочтений в преддверии президентских выборов генеральной совокупностью будет население России в возрасте 18 лет и старше (поскольку право голосовать гражданин получает с 18 лет). В банковском скоринге генеральной совокупностью считаются все потенциально возможные заемщики банка. В таких случаях принято считать, что объем генеральной совокупности у нас неизвестен.

Выборка, *или* **выборочная совокупность (sample)** – набор объектов, выбранных с помощью определенной процедуры из генеральной совокупности для участия в исследовании.

Цель проведения выборочных обследований – на основе выборки сформировать суждение обо всей генеральной совокупности.

Допустим, нам необходимо провести исследование, цель которого – убедиться, что груши на дереве созрели. Решение заключается в том, чтобы сорвать несколько груш с дерева и попробовать их. Генеральная совокупность – все груши на дереве. Выборочная совокупность – сорванные груши с дерева. Если все сорванные груши созрели, то с большой вероятностью можно сделать вывод, что и все остальные груши на дереве тоже созрели. Если необходимо узнать, все ли груши созрели во всем саду, то это будет уже другая генеральная совокупность – груши во всем саду. Исследование будет состоять в том, чтобы срывать и пробовать груши с разных деревьев.

2.2. Характеристики выборки

Перечень всех единиц наблюдения генеральной совокупности с базовой информацией представляет **основу выборки**. **Базовая информация** – набор характеристик, известных до проведения обследования для каждого элемента основы выборки (например, фамилия, имя и отчество респондента, адрес предприятия, регион проведения интервью и другие характеристики).

Элементы отбора при формировании выборочной совокупности называются **единицами отбора**. Объект, признаки которого подлежат регистрации, называется **единицей наблюдения**. Обычно единицей наблюдения в социологических опросах является конкретный человек, который будет отвечать на вопрос анкеты. Единица наблюдения может совпадать или не совпадать с единицей

отбора. При простой случайной выборке единицы отбора и единицы наблюдения совпадают. В случае использования многоступенчатой выборки сначала отбираются регионы, потом населенные пункты, затем предприятия или адреса проживания семей (все они и будут единицами отбора), и лишь на последнем этапе будут отобраны конкретные единицы наблюдения – респонденты.

Количество элементов выборки называется **объемом** (**размером**) выборки.

Соответствие характеристик выборки характеристикам популяции или генеральной совокупности в целом называется **репрезентативностью**. Репрезентативность определяет, насколько возможно обобщать результаты исследования с привлечением определенной выборки на всю генеральную совокупность, из которой она была отобрана. Корректный вывод обо всей генеральной совокупности можно сделать только на основании репрезентативной выборки. Поэтому при формировании выборки должен быть такой отбор элементов, чтобы выборка была репрезентативной.

В США одним из наиболее известных исторических примеров нерепрезентативной выборки считается случай, произошедший во время президентских выборов в 1936 году.

Журнал «Литрери Дайджест», успешно прогнозировавший события нескольких предшествующих выборов, ошибся в своих предсказаниях, разослав десять миллионов пробных бюллетеней своим подписчикам, а также людям, выбранным по телефонным книгам всей страны и людям из регистрационных списков автомобилей. В 25 % вернувшихся бюллетеней (почти 2,5 миллиона) голоса были распределены следующим образом:

57 % отдавали предпочтение кандидату-республиканцу Альфу Лэндону;

40 % выбрали действующего в то время президента-демократа Франклина Рузвельта.

На действительных же выборах, как известно, победил Рузвельт, набрав более 60 % голосов. Ошибка «Литрери Дайджест» заключалась в следующем: желая увеличить репрезентативность выборки, – так как им было известно, что большинство их подписчиков считают себя республиканцами, – они расширили выборку за счёт людей, выбранных из телефонных книг и регистрационных списков. Однако они не учли современных им реалий и в действительности набрали ещё больше республиканцев: во время Великой депрессии обладать телефонами и автомобилями могли себе позволить в основном представители среднего и высшего класса (то есть большинство республиканцев, а не демократов).

В нашем игрушечном примере, когда нам нужно было убедиться, что груши на дереве созрели, примером нерепрезентативной выборки были бы груши, сорванные только с одной, южной стороны дерева. А если бы нам необходимо было узнать, все ли груши созрели во всем саду, то примером нерепрезентативной выборки были бы груши, сорванные с деревьев, которые росли поблизости (допустим, мы поленились пройти в глубь сада).

Отклонение результатов оценки значений, полученных с помощью выборки, от истинных неизвестных значений в генеральной совокупности называется **ошибкой выборки**.

В выборочных обследованиях мы будем оперировать статистиками. **Статистика** – это некоторая функция от выборочных наблюдений, например минимальное значение, среднее арифметическое, стандартное отклонение и др. Например, минимальный вес груши, средний вес груши.

Исследование всех объектов генеральной совокупности называется **сплошным обследованием**. Наиболее точные оценки могут быть получены при сплошном наблюдении, однако могут быть сложности. Основные проблемы, возникающие при сплошном наблюдении: ограничение по времени, ограничение финансовых ресурсов, ограничение человеческих ресурсов (здесь речь идет о физических и интеллектуальных ресурсах как опрашивающих, так и опрошенных).

Понятно, что мы не можем для получения рейтингов кандидатов на пост Президента РФ физически опросить все население России в возрасте от 18 лет и старше. Однако даже если сплошное обследование можно организовать, оно не гарантирует получения надежных результатов. Примером, когда сплошное обследование потерпело неудачу, была сплошная перепись населения России 1897 г. Когда анализировалась численность населения по возрастам, то получалось, что максимальные численности (пики) имели возрасты, кратные 5 и в особенности кратные 10. Большая часть населения в те времена была неграмотна и свой возраст помнила только приблизительно, с точностью до пяти или до десяти лет. Чтобы все-таки узнать, каково было распределение по возрастам на самом деле, нужно было не увеличивать объем данных, а, наоборот, создать выборку из нескольких процентов населения и провести комплексное исследование, основанное на перекрестном анализе нескольких источников: документов, свидетельств и личных показаний. Это дало бы гораздо более точную картину, нежели сплошная перепись. Для решения проблем, возникающих при сплошном обследовании, как раз и используют выборочные обследования.

2.3. Детерминированные и вероятностные выборки

По способу отбора выборки делятся на:

- **детерминированные**;
- **вероятностные**.

Детерминированный отбор – выборочный метод, в котором не применяется процедура случайного отбора единиц генеральной совокупности. Этот метод основан на индивидуальных суждениях исследователя. Примерами являются экспертный отбор, квотный отбор, отбор методом «снежного кома».

Выборка по методу «снежного кома» строится следующим образом. У каждого респондента, начиная с первого, просятся контакты его друзей, коллег, знакомых, которые подходили бы под условия отбора и могли бы принять участие в исследовании. Таким образом, за исключением первого шага, выборка формируется с участием самих объектов исследования. Метод часто применяется, когда необходимо найти и опросить труднодоступные группы респондентов (например, респондентов, имеющих высокий доход, респондентов, принадлежащих к одной профессиональной группе, респондентов, имеющих какие-либо схожие хобби/увлечения и т. д.).

При квотной выборке генеральная совокупность сначала разделяется на непересекающиеся группы. Затем пропорционально из каждой группы выбираются единицы наблюдения на основании предпочтений отбирающего. Например, интервьюер может получить задание отобрать 200 женщин и 300 мужчин возрастом от 45 до 60 лет. Это значит, что внутри каждой квоты интервьюер отбирает респондентов по своим предпочтениям.

Описанный второй шаг формирования квотной выборки относит её к детерминированному типу. Отбор элементов в квотную выборку не является случайным и может быть ненадёжным. Например, интервьюеры могут в первую очередь пытаться опрашивать тех людей, которые выглядят наиболее отзывчивыми или живут поблизости. Соответственно, менее отзывчивые люди или репонденты, живущие в труднодоступных местах, криминогенных районах, в которых интервьер побоится опрашивать, имеют меньше шансов попасть в выборку.

Квотная выборка полезна, когда время ограничено, отсутствует основа для формирования вероятностной выборки, бюджет исследования небольшой или когда точность результатов не слишком важна.

Вероятностный отбор – выборочный метод, в котором состав выборки формируется случайным образом. В вероятностном отборе каждая единица генеральной совокупности имеет определенную вероятность включения в выборку. Нас будут интересовать вероятностные выборочные методы.

2.4. Виды, методы и способы вероятностного отбора

По виду отбора различают следующие вероятностные выборки:

- выборки с индивидуальным отбором;
- выборки с групповым отбором;
- выборки с комбинированным отбором.

Выборки с индивидуальным отбором осуществляют отбор из генеральной совокупности каждой единицы наблюдения в отдельности. Например, при обследовании удовлетворенности сотрудников предприятия размером заработной платы осуществляется отбор сотрудников.

Выборки с групповым отбором осуществляют отбор групп единиц. Например, при обследовании удовлетворенности сотрудников предприятия размером заработной платы осуществляется отбор отделов предприятия.

По методу отбора различают:

- выборки без возвращения (бесповторный отбор);
- выборки с возвращением (повторный отбор).

В выборках без возвращения (бесповторный отбор) отобранный элемент не возвращается в генеральную совокупность, из которой осуществлялся отбор. В выборках с возвращением (повторный отбор) отобранный объект возвращается в генеральную совокупность и имеет шанс быть отобранным повторно. Использование повторного метода дает бóльшую ошибку выборки, чем использование бесповторного.

По способам отбора различают:

- простой случайный отбор;
- систематический отбор;
- вероятностно-пропорциональный отбор;
- расслоенный случайный отбор;
- кластерный (серийный) отбор.

Более подробное обсуждение этих способов выходит за рамки книги, разберем здесь лишь процедуру простого случайного отбора.

При проведении простого случайного отбора каждая единица генеральной совокупности имеет известную и равную вероятность отбора. В простом случайном отборе каждая единица отбирается независимо от другой. Для отбора используется таблица случайных чисел или компьютерная программа.

Здесь отметим, что к повторному отбору приравнивается простой случайный отбор из генеральной совокупности, объем которой неизвестен. При вычислении необходимого объема выборки для построения моделей банковского скоринга как раз предполагают, что имеет место повторный отбор.

2.5. Подходы к определению необходимого объема выборки

Необходимый объем выборки может быть известен по результатам предыдущих аналогичных исследований. Если же объем выборки неизвестен, его необходимо рассчитать.

2.5.1. Определение объема выборки согласно теории выборочных обследований

Согласно теории выборочных обследований объем необходимой выборки зависит от задаваемой точности оценки параметров, дисперсии оцениваемых параметров и способа отбора. Общее правило следующее: чем больше дисперсия оцениваемых параметров, тем больший объем выборки необходим для того, чтобы обеспечить требуемую точность. Поэтому предварительно по отобранным данным необходимо рассчитать дисперсию оцениваемых переменных. В зависимости от величины надежности выбирают значение стандартного нормального распределения.

В банковском скоринге для построения качественной модели данные о «хороших» и «плохих» клиентах максимально должны отражать поток клиентов с улицы.

Предположим, мы хотим быть уверенными на 95 %, что соотношение «хороших» и «плохих» заемщиков в обучающей выборке отражает генеральную совокупность заемщиков. В таких случаях обычно используют следующую формулу определения объема выборки для оценки генеральной доли при повторном случайном отборе (при этом предполагается, что выборка значительно меньше генеральной совокупности):

$$n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2},$$

где:

n – минимальный объем выборки;

z_γ – значение стандартного нормального распределения, определяемое в зависимости от выбранного доверительного уровня (доверительной вероятности);

w – доля «плохих» на предварительной выборке (может быть получена, исходя из опыта имеющихся априорных знаний);

Δ_w – максимально допустимая предельная ошибка оценки доли «плохих» заемщиков (предельная ошибка выборки).

Доверительный уровень (доверительная вероятность) – это вероятность того, что генеральная доля лежит в границах полученного доверительного интервала: выборочная доля (w) ± ошибка выборки (Δ_w). Доверительный уровень устанавливает сам исследователь в соответствии со своими требованиями к надежности полученных результатов. Чаще всего применяются доверительные уровни, равные 0,95 или 0,99.

Допустим, среди 700 клиентов предварительной выборки 50 оказались «плохими». Оценка доли «плохих» клиентов, по имеющимся данным, для построения модели составила около 0,07, или 7 %. При таком значении оценки доли предположим, мы хотим ошибиться не более чем на 10 %, что будет соответствовать допустимой предельной ошибке оценки доли 0,007, т. е. (50 / 700) * 0,1. При этом задаем 95%-ную доверительную вероятность. В этом случае z-значение стандартного нормального закона распределения составит около 1,96. Вычисляем минимальный объем выборки: $n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2} = \frac{3,84 \cdot 0,07 \cdot 0,93}{0,000049} = 5102.$

На практике чаще всего нет возможности сформировать предварительную выборку и значение w неизвестно. В таком случае w принимается за 0,5 (самый консервативный сценарий). При этом значении размер ошибки выборки будет максимален.

Допустим, оценка доли «плохих» клиентов неизвестна, принимаем ее за 0,5. При таком значении оценки доли предположим, мы хотим ошибиться не более чем на 10 %. При этом задаем 95%-ную доверительную вероятность. Получаем

$$n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2} = \frac{3,84 \cdot 0,5 \cdot 0,5}{0,000049} = 19592.$$

На собеседованиях часто просят рассчитать объем выборки с определенной предельной ошибкой. Например, рассчитайте объем выборки, предельная ошибка которой составит 4 %. При этом мы принимаем, что доверительный уровень равен 95 %, а генеральная совокупность значительно больше выборки.

Применяем знакомую формулу $n = \frac{z_\gamma^2 w(1-w)}{\Delta_w^2} = \frac{3,84 \cdot 0,5 \cdot 0,5}{0,0016} = 600.$

При этом не забывайте, вам помимо обучающей выборки еще нужно отложить наблюдения для проверки, здесь, соответственно, нужно выделить ситуацию, когда вы строите и проверяете базовую модель (вам нужен тестовый набор), и ситуацию, когда вы строите модели, настраивая гиперпараметры (вам нужен валидационный набор для настройки гиперпараметров, роль валидационного набора могут выполнять проверочные блоки перекрестной проверки и тестовый набор для итоговой оценки качества).

Когда предполагается, что выборка сопоставима с генеральной совокупностью (обычное явление при опросах организаций в B2B-исследованиях), формула определения объема выборки для оценки генеральной доли будет выглядеть несколько иначе, в ней будет фигурировать показатель объема генеральной совокупности.

$$n = \frac{\frac{z_\gamma^2 w(1-w)}{\Delta_w^2}}{1 + \frac{\frac{z_\gamma^2 w(1-w)}{\Delta_w^2} - 1}{N}}.$$

2.5.2. Определение объема выборки согласно правилу NEPV

В практике банковского скоринга для ответа на вопрос об объеме выборки часто используют правило «Number of Events Per Variable» (количество событий на одну переменную, NEPV), сформулированное Фрэнком Харреллом.

Для задачи бинарной классификации оно связывает минимальный объем выборки с количеством «событий» – наблюдений в миноритарной (наименьшей по размеру) категории зависимой переменной и количеством признаков, поданным на вход модели. Согласно этому правилу, необходимо взять количество наблюдений в обучающей выборке, относящихся к миноритарной категории зависимой переменной (в кредитном скоринге это «плохие» заемщики). Это число наблюдений нужно разделить на количество заданных признаков. Для логистической регрессии на один параметр должно приходиться не менее 20 событий, при построении дерева решений CHAID на один признак должно приходиться не менее 50 событий, а для модели случайного леса, градиентного бустинга, SVM и нейронной сети на одну независимую переменную должно приходиться не менее 200 событий.

Для задачи регрессии мы просто берем количество наблюдений и делим на количество признаков, и для линейной регрессии на один параметр должно приходиться не менее 20 наблюдений, для дерева решений CHAID (в тех случаях, когда реализация алгоритма позволяет решать задачу регрессии) на один признак должно приходиться не менее 50 наблюдений, для случайного леса и других сложных моделей на один признак должно приходиться не менее 200 наблюдений. По мнению Фрэнка Харрелла, для дерева решений CART правило NEPV невозможно сформулировать из-за высокой нестабильности метода и склонности к переобучению.

Если правило выполняется для обучающей выборки, то объем выборки для обучения является достаточным. В противном случае необходимо либо увеличить объем выборки, либо сократить количество признаков, подаваемых на вход модели. Затем вся та же самая процедура применяется к тестовой выборке, если правило выполняется, объем выборки для проверки достаточен.

Мы решаем задачу классификации, и у нас есть общая выборка объемом 4424 клиента, классифицированных на два класса: класс *Остается* (2492 клиента) и класс *Уходит* (1932 клиента). Мы разбили выборку на обучающую и тестовую и получили следующее распределение классов в выборках: *Остается* (1746 клиентов) и *Уходит* (1334 клиента).

Выясняем, достаточен ли объем выборки для обучения. У нас 1746 оставшихся клиентов, 1334 ушедших клиента и 9 независимых переменных. Ми-

норитарный класс – класс *Уходит*. Проверяя выполнение правила NEPV, мы получаем 1334 / 9 = 148,2. Наша выборка обеспечивает достаточное количество событий на одну переменную, и мы можем использовать эту выборку для обучения.

Выясняем, достаточен ли объем выборки для проверки. У нас 746 оставшихся клиентов, 598 ушедших клиентов. Проверяя выполнение правила NEPV, мы получаем 598 / 9 = 66,4. Наша выборка обеспечивает достаточное количество событий на одну переменную, и мы можем использовать эту выборку для проверки.

Если выполняется перекрестная проверка, роль обучающей выборки выполняет набор обучающих блоков, а роль тестовой выборки выполняет тестовый блок.

Опять же напомним: данная схема работает для построения базовой модели без подбора гиперпараметров.

2.5.3. Определение объема выборки с помощью кривых обучения и валидации

Кроме того, необходимый объем выборки можно определить с помощью кривых обучения и валидации. Их можно построить с помощью функции `learning_curve()`. Она запускает перекрестную проверку на наборах данных разного объема. Генератор перекрестной проверки разбивает весь набор данных *k* раз на обучающую выборку и тестовую выборку. В итоге для набора соответствующего размера мы получаем метрику для обучающей выборки, усредненную по *k* проходам, и метрику для тестовой выборки, усредненную по *k* проходам.

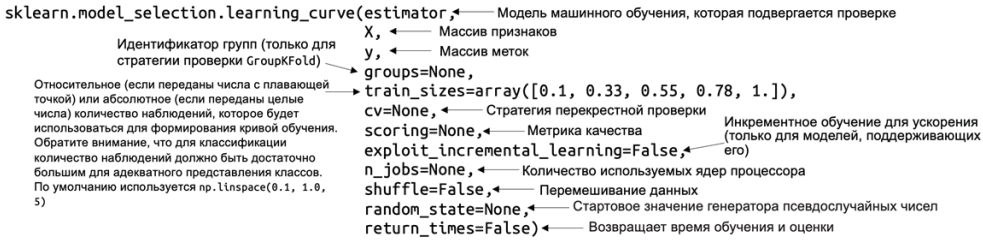


Рис. 2 Параметры функции learning_curve()

В результате функция `learning_curve()` возвращает:

- `train_sizes_abs` – количество наблюдений, использованное для построения кривой обучения;
- `train_scores` – значения метрики на обучающих выборках перекрестной проверки;
- `test_scores` – значения метрики на тестовых выборках перекрестной проверки;
- `fit_times` – время, затраченное на обучение, в секундах;
- `score_times` – время, затраченное на оценку качества, в секундах.

Давайте загрузим необходимые библиотеки, классы и функции.

```
# импортируем библиотеки, классы и функции
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
```

Теперь загрузим данные. Данные записаны в файле *Response.csv*. Исходная выборка содержит записи о 30 259 клиентах, классифицированных на два класса: 0 – отклика нет (17 170 клиентов) и 1 – отклик есть (13 089 клиентов).

По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- категориальный признак *Ипотечный кредит [mortgage]*;
- категориальный признак *Страхование жизни [life_ins]*;
- категориальный признак *Кредитная карта [cre_card]*;
- категориальный признак *Дебетовая карта [deb_card]*;
- категориальный признак *Мобильный банк [mob_bank]*;
- категориальный признак *Текущий счет [curr_acc]*;
- категориальный признак *Интернет-доступ к счету [internet]*;
- категориальный признак *Индивидуальный займ [perloan]*;
- категориальный признак *Наличие сбережений [savings]*;
- категориальный признак *Пользование банкоматом за последнюю неделю [atm_user]*;
- категориальный признак *Пользование услугами онлайн-маркетплейса за последний месяц [markpl]*;
- количественный признак *Возраст [age]*;
- количественный признак *Давность клиентской истории [cus_leng]*;
- категориальная зависимая переменная *Отклик на предложение новой карты [response]*.

```
# загружаем данные
data = pd.read_csv('Data/Response.csv', sep=';')
data.head(3)
```

	mortgage	life_ins	cre_card	deb_card	mob_bank	curr_acc	internet	perloan	savings	atm_user	markpl	age	cus_leng	response
0	No	No	No	No	No	No	No	No	No	No	No	18.0	less than 3 years	No
1	Yes	Yes	NaN	NaN	Yes	No	NaN	NaN	NaN	Yes	No	18.0	NaN	Yes
2	Yes	Yes	NaN	Yes	No	No	No	No	No	No	Yes	NaN	from 3 to 7 years	Yes

Формируем массив меток и массив признаков, создаем списки переменных и трансформеры, с помощью класса `ColumnTransformer` сопоставляем транс-

формеры со списками переменных, создаем конвейер для логистической регрессии и конвейер для градиентного бустинга, каждой модели машинного обучения будет соответствовать своя последовательность моделей предварительной подготовки данных.

```
# создаем массив меток и массив признаков
y = data.pop('response')

# создаем списки категориальных
# и количественных столбцов
categorical_features = data.select_dtypes(
    include='object').columns.tolist()
numeric_features = data.select_dtypes(
    exclude='object').columns.tolist()

# создаем трансформеры
numeric_transformer_logreg = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

numeric_transformer_boost = Pipeline([
    ('imputer', SimpleImputer(strategy='median'))
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='constant',
                              fill_value='missing')),
    ('onehot', OneHotEncoder(sparse=False,
                             handle_unknown='ignore'))
])

# сопоставляем трансформеры спискам переменных
# для логистической регрессии
preprocessor_logreg = ColumnTransformer([
    ('num', numeric_transformer_logreg, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

# сопоставляем трансформеры спискам переменных
# для градиентного бустинга
preprocessor_boost = ColumnTransformer([
    ('num', numeric_transformer_boost, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

# формируем итоговый конвейер
pipe_logreg = Pipeline([
    ('preprocessor', preprocessor_logreg),
    ('logreg', LogisticRegression(solver='lbfgs', max_iter=400))
])

pipe_boost = Pipeline([
    ('preprocessor', preprocessor_boost),
                     ('boost', GradientBoostingClassifier(
                         random_state=42))])
```

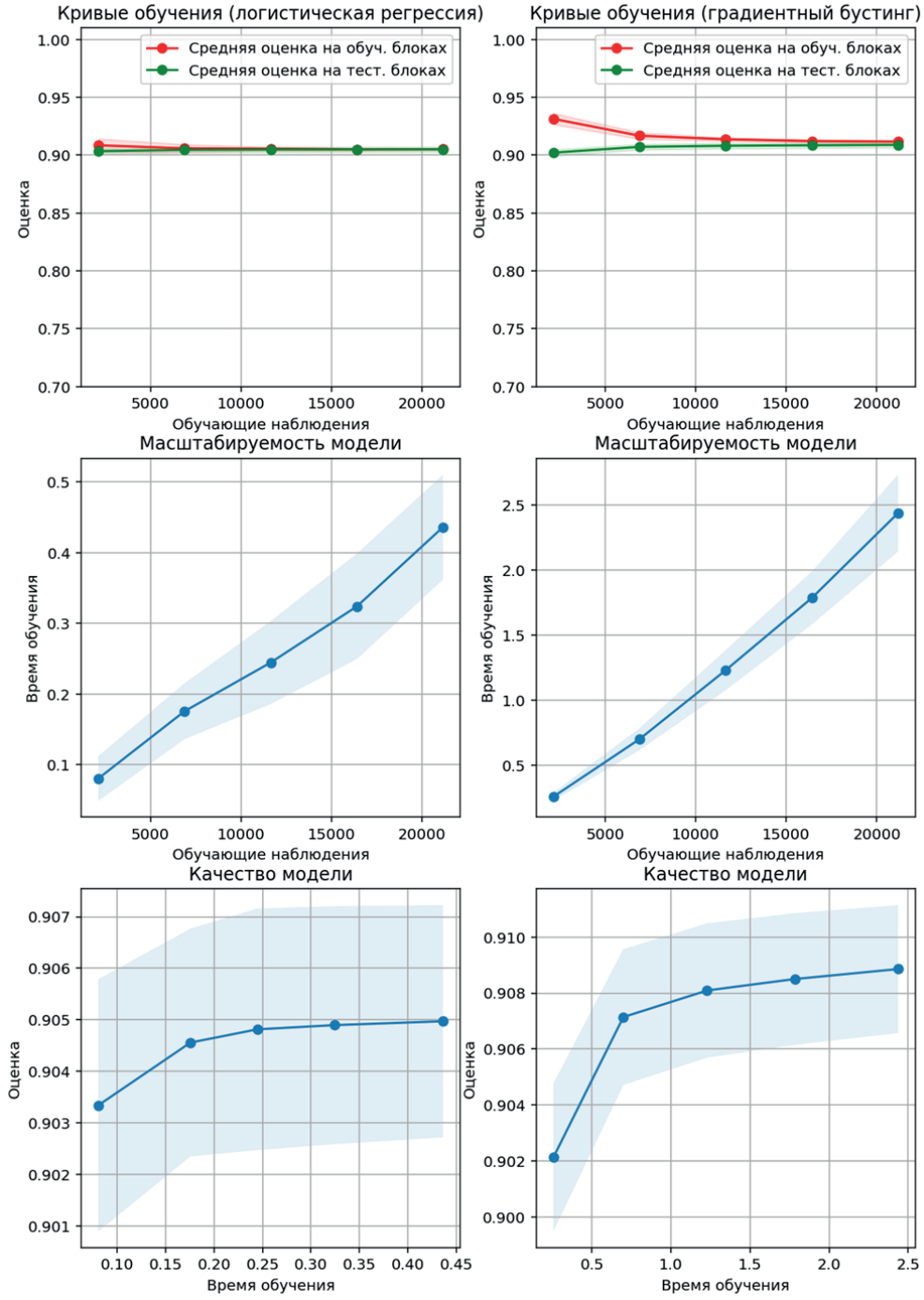
Теперь пишем функцию, которая будет строить графики на основе результатов, возвращенных функцией `learning_curve()`.

```
# пишем функцию, которая строит графики
# по результатам функции learning_curve()
def plot_learning_curve(estimator,
                        title,
                        X,
                        y,
                        axes=None,
                        ylim=None,
                        cv=None,
                        n_jobs=None,
                        train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Строит 3 графика: кривые обучения и валидации, кривую зависимости между
    объемом обучающих данных и временем обучения, кривую зависимости между
    временем обучения и оценкой качества.

    Параметры
    ----------
    estimator : модель машинного обучения для проверки.
    title : заголовок диаграммы.
    X : массив признаков.
    y : массив меток.
    axes : задаем область рисования (Axes) для построения кривых.
    ylim : задает минимальное и максимальное значения по оси y.
    cv : стратегия перекрестной проверки.
    n_jobs : количество используемых ядер процессора.
    train_sizes : абсолютное или относительное количество наблюдений.
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))
    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Обучающие наблюдения")
    axes[0].set_ylabel("Оценка")
    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv,
                       scoring='roc_auc', n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # строим кривые обучения и валидации
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
```

```
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Средняя оценка на обуч. блоках")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Средняя оценка на тест. блоках")
    axes[0].legend(loc="best")

    # строим график зависимости между объемом
    # обучающих данных и временем обучения
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Обучающие наблюдения")
    axes[1].set_ylabel("Время обучения")
    axes[1].set_title("Масштабируемость модели")
    # строим график зависимости между временем
    # обучения и оценкой качества
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("Время обучения")
    axes[2].set_ylabel("Оценка")
    axes[2].set_title("Качество модели")
    return plt
```

А сейчас будем строить графики кривых обучения и валидации.

```
# задаем сетку и размеры графиков
fig, axes = plt.subplots(3, 2, figsize=(10, 15))
# задаем стратегию перекрестной проверки
cv = ShuffleSplit(n_splits=20, test_size=0.3, random_state=42)

# задаем заголовок
title = "Кривые обучения (логистическая регрессия)"
# строим графики для логистической регрессии
plot_learning_curve(pipe_logreg, title, data, y, axes=axes[:, 0],
                    ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

# задаем заголовок
title = "Кривые обучения (градиентный бустинг)"
# строим графики для градиентного бустинга
plot_learning_curve(pipe_boost, title, data, y, axes=axes[:, 1],
                    ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

# выводим графики
plt.show()
```



Для каждой модели машинного обучения мы выводим по три графика: кривые обучения и валидации, кривую зависимости между объемом обучающих

данных и временем обучения, кривую зависимости между временем обучения и оценкой качества.

График кривых обучения и валидации для логистической регрессии показывает, что независимо от размера обучающего набора оценка на обучении практически совпадает с оценкой на тесте.

График кривых обучения и валидации для градиентного бустинга показывает, что по мере увеличения размера обучающего набора оценка на обучении приближается к оценке на тесте (уменьшается гэп между ними), т. е. происходит уменьшение переобучения. При этом мы видим, что обе модели практически эквивалентны по качеству (как метрика качества у нас используется AUC) и увеличения оценки на тесте по мере роста объема данных практически не происходит, что может говорить о достаточном объеме данных, и, скорее всего, улучшение качества можно добиться не за счет увеличения данных, а за счет тщательно продуманного конструирования признаков.

Графики кривой зависимости между объемом обучающих данных и временем обучения показывают, что обучение градиентного бустинга занимает больше времени. В том случае, когда мы используем набор размером 5000 наблюдений, обучение логистической регрессии занимает в среднем 0,12 секунды, а обучение градиентного бустинга – 0,5 секунды.

Графики кривой зависимости между временем обучения и оценкой качества показывают, сколько времени обучения требуется для получения соответствующего качества на тесте. Например, мы можем принять, что разница между AUC 0,905 и AUC 0,908 не столь критична, чтобы обучаться 2,5 секунды вместо 0,4 секунды.

Теперь возьмем другой набор данных. Данные записаны в файле *StateFarm_missing.csv*. Исходная выборка содержит записи о 8293 клиентах, классифицированных на два класса: 0 – отклика нет на предложение автостраховки (7462 клиента) и 1 – отклик есть на предложение автостраховки (831 клиент). По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- количественный признак *Пожизненная ценность клиента* [*Customer Lifetime Value*];
- категориальный признак *Вид страхового покрытия [Coverage]*;
- категориальный признак *Образование [Education];*
- категориальный признак *Тип занятости [EmploymentStatus]*;
- категориальный признак *Пол [Gender]*;
- количественный признак *Доход клиента* [*Income*];
- количественный признак *Размер ежемесячной автостраховки [Monthly Premium Auto]*;
- количественный признак *Количество месяцев со дня подачи последнего страхового требования [Months Since Last Claim]*;
- количественный признак *Количество месяцев с момента заключения страхового договора [Months Since Policy Inception]*;
- количественный признак *Количество открытых страховых обращений [Number of Open Complaints]*;
- количественный признак *Количество полисов [Number of Policies]*;
- категориальная зависимая переменная *Отклик на предложение автостраховки [Response]*.

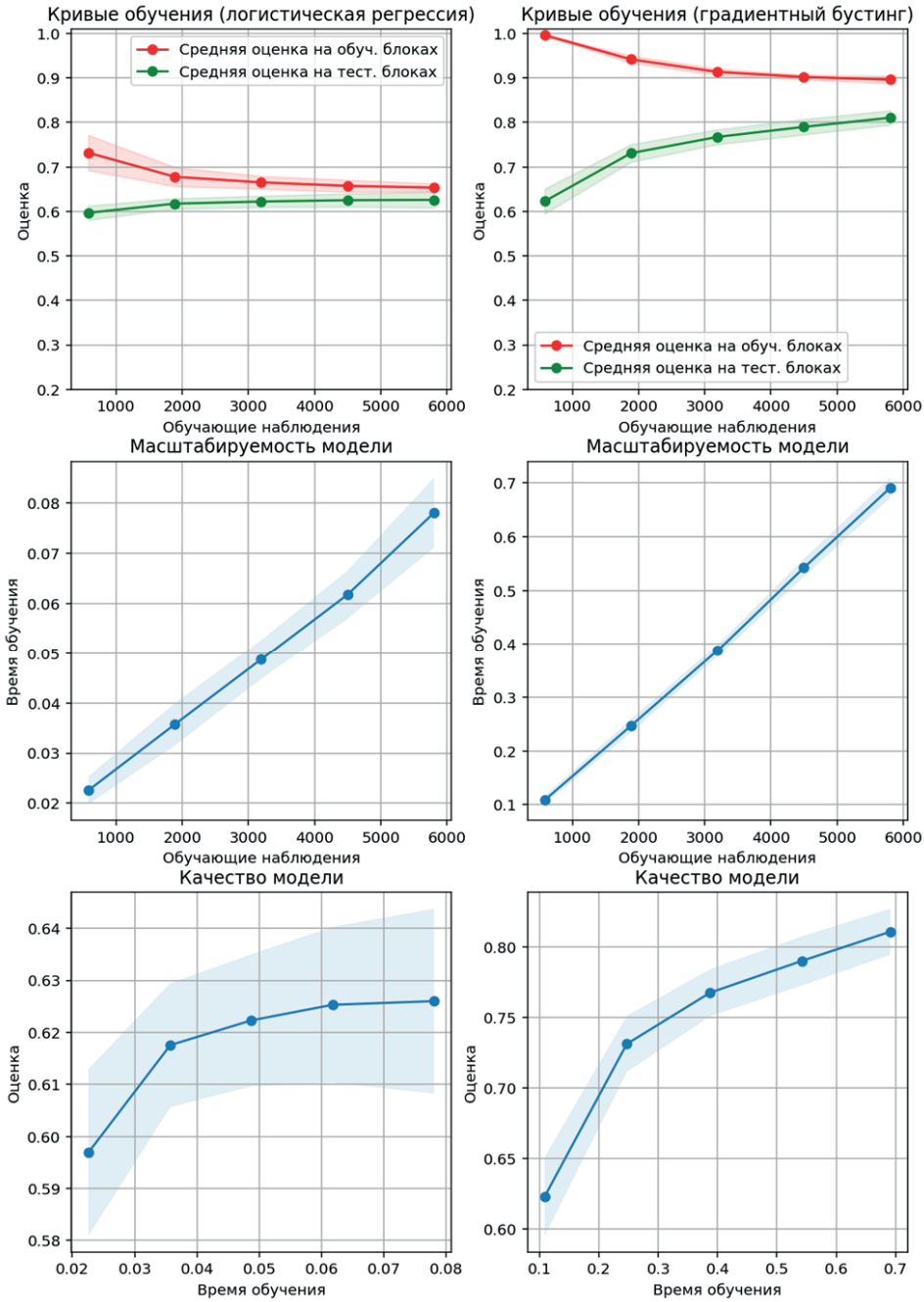
Давайте загрузим данные, опять создадим конвейеры и построим графики кривых обучения и валидации.

```
# загружаем данные
data = pd.read_csv('Data/StateFarm_missing.csv', sep=';')
data.head(3)
```

	Customer Lifetime Value	Coverage	Education	EmploymentStatus	Gender	Income	Monthly Premium Auto	Months Since Last Claim	Months Since Policy Inception	Number of Open Complaints	Number of Policies	Response
0	2763.519279	Basic	Bachelor	Employed	F	56274.0	NaN	32.0	5.0	NaN	1.0	No
1	NaN	NaN	Bachelor	Unemployed	F	0.0	NaN	13.0	42.0	NaN	NaN	No
2	NaN	NaN	NaN	Employed	F	48767.0	108.0	NaN	38.0	0.0	NaN	No

```
# формируем массив меток и массив признаков
y = data.pop('Response')

# создаем списки количественных
# и категориальных столбцов
cat_features = data.select_dtypes(
    include='object').columns.tolist()
num_features = data.select_dtypes(
    exclude='object').columns.tolist()

# сопоставляем трансформеры спискам переменных
# для логистической регрессии
preprocessor_lr = ColumnTransformer([
    ('num', numeric_transformer_logreg, num_features),
    ('cat', categorical_transformer, cat_features)
])

# сопоставляем трансформеры спискам переменных
# для градиентного бустинга
preprocessor_bst = ColumnTransformer([
    ('num', numeric_transformer_boost, num_features),
    ('cat', categorical_transformer, cat_features)
])

# формируем итоговый конвейер
pipe_lr = Pipeline([
    ('preprocessor', preprocessor_lr),
    ('logreg', LogisticRegression(solver='lbfgs',
                                  max_iter=400))
])

pipe_bst = Pipeline([
    ('preprocessor', preprocessor_bst),
    ('boost', GradientBoostingClassifier(
        random_state=42))
])

# задаем сетку и размеры графиков
fig, axes = plt.subplots(3, 2, figsize=(10, 15))

# задаем заголовок
title = "Кривые обучения (логистическая регрессия)"
# строим графики для логистической регрессии
plot_learning_curve(pipe_lr, title, data, y, axes=axes[:, 0],
                    ylim=(0.2, 1.01),
                    cv=cv, n_jobs=4)

# задаем заголовок
title = "Кривые обучения (градиентный бустинг)"
# строим графики для градиентного бустинга
```

```
plot_learning_curve(pipe_bst, title, data, y, axes=axes[:, 1],
                    ylim=(0.2, 1.01),
                    cv=cv, n_jobs=4)

# выводим графики
plt.show()
```

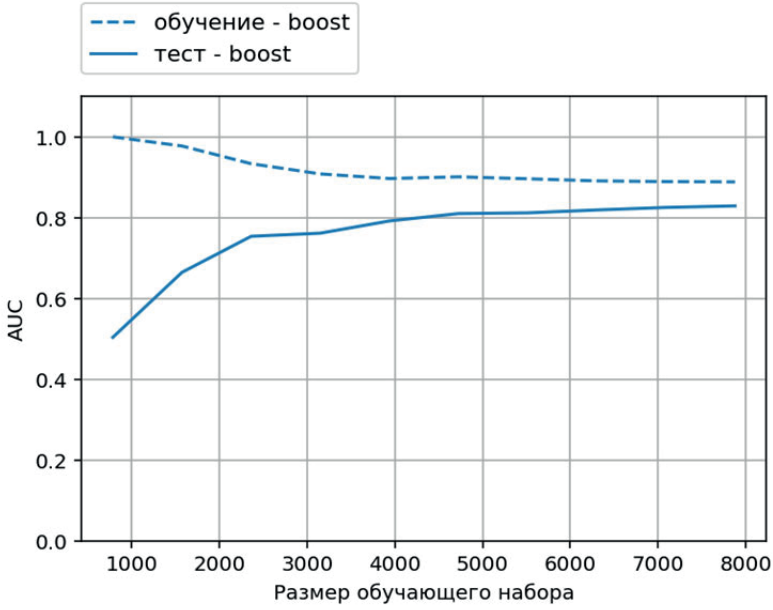


Вновь для каждой модели машинного обучения мы выводим по три графика: кривые обучения и валидации, кривую зависимости между объемом обучающих данных и временем обучения, кривую зависимости между временем обучения и оценкой качества.

Здесь уже графики кривых обучения и валидации для логистической регрессии и бустинга показывают, что по мере увеличения размера обучающего набора оценка на обучении сближается с оценкой на тесте. При этом если для логистической регрессии оценка на обучении практически сближается с оценкой на тесте, то для градиентного бустинга гэп хоть и уменьшается, но все еще остается значительным, что может говорить о недостаточном объеме данных, и, скорее всего, улучшения качества можно добиться за счет увеличения данных. При этом мы видим, что с точки зрения оценки на тесте модель градиентного бустинга существенно лучше модели логистической регрессии.

На практике часто пользуются упрощенной версией функции `plot_learning_curve_simple()`.

```
# импортируем класс KFold
from sklearn.model_selection import KFold

# пишем упрощенную версию plot_learning_curve()
def plot_learning_curve_simple(est, X, y):
    # получаем наборы для обучения, значения метрик
    training_set_size, train_scores, test_scores = learning_curve(
        est, X, y, train_sizes=np.linspace(.1, 1, 10), scoring='roc_auc',
        cv=KFold(20, shuffle=True, random_state=1))
    # извлекаем имя последнего этапа итогового конвейера -
    # название модели машинного обучения
    estimator_name = est.steps[-1][0]
    # строим кривые обучения и валидации
    line = plt.plot(training_set_size, train_scores.mean(axis=1), '--',
                    label="обучение - " + estimator_name)
    plt.plot(training_set_size, test_scores.mean(axis=1), '-',
             label="тест - " + estimator_name, c=line[0].get_color())
    # задаем координатную сетку
    plt.grid()
    # подписываем ось x
    plt.xlabel("Размер обучающего набора")
    # подписываем ось y
    plt.ylabel("AUC")
    # задаем пределы значений оси y
    plt.ylim(0, 1.1)
    # задаем расположение легенды
    plt.legend(loc=(0, 1.05), fontsize=11)
# применяем упрощенную версию plot_learning_curve()
plot_learning_curve_simple(pipe_bst, data, y)
```


обучение - boost
тест - boost
AUC
1.0
0.8
0.6
0.4
0.2
0.0
1000
2000
3000
4000
5000
6000
7000
8000
Размер обучающего набора

3. Определение «окна выборки» и «окна созревания»

Прогнозные модели разрабатываются, исходя из предположения «прошлое отражает будущее». На основе этого предположения мы анализируем поведение прошлых клиентов, чтобы спрогнозировать поведение будущих клиентов. Для того чтобы корректно выполнить этот анализ, нужно собрать необходимые данные о клиентах за определенный период времени, а затем осуществить мониторинг клиентов в течение другого определенного периода времени, оценив, были они «хорошими» или «плохими». Собранные данные (независимые переменные) наряду с соответствующей классификацией (зависимой переменной, которая принимает значение *Хороший* или *Плохой*) составят основу для разработки прогнозной модели. Ключевыми терминами здесь будут «окно выборки» и «окно созревания».

«Окно выборки» – это период времени, в течение которого те или иные клиенты отбираются для анализа (попадают в выборку).

«Окно созревания» – это период времени, в течение которого клиент, собственно говоря, имел возможность себя проявить, и мы присваиваем клиенту соответствующий класс зависимой переменной.

Допустим, сделано предположение, что новый клиент получил кредит в определенный период времени (например, 1 января 2014 г.). В некоторый момент времени в будущем (например, через 90 дней) нам нужно определить, был этот клиент «хорошим» или «плохим» (чтобы классифицировать поведение).

Если мы возьмем все кредиты, выданные в январе 2014 года, и посмотрим на их качество с момента открытия до декабря 2015 года, окном выборки будет январь 2014 года, а окном созревания – 24 месяца, период с января 2014 года по декабрь 2015 года.

В некоторых случаях, таких как мошенничество и банкротство, временной период уже известен или предопределен. Но тем не менее вышеописанный анализ полезно выполнить для того, чтобы определить идеальное окно созревания.

Мы можем попробовать несколько подходов к определению окна выборки и окна созревания.

В ряде случаев окно созревания определяется требованиями регулирующих органов, т. е. горизонтом прогнозирования модели. Например, Базель II требует 12-месячное окно созревания, поэтому вероятность дефолта в моделях, построенных в соответствии с требованиями Базель II, определяется по 12-месячному окну созревания. Более поздние инициативы, такие как МСФО (IFRS) 9.3, предлагают использовать более длительный горизонт прогнозирования убытков, вплоть до срока действия кредита.

Второй подход сопоставляет окно созревания со сроком кредита. Например, если срок автокредита составляет четыре года, оценка заявок по этому кредиту должна основываться на четырехлетнем окне созревания. Логично, что отношения, в которые вступает кредитор, продолжаются четыре года, поэтому риск должен оцениваться в течение четырехлетнего периода. Этот подход хорошо работает для срочных кредитов. Если срок займа очень большой (скажем, более 8–10 лет), то можно применить третий подход, основанный на опреде-

лении окна созревания по данным винтажного анализа, чтобы получить более короткое окно созревания и использовать более свежие данные.

Для револьверных кредитов (например, возобновляемых кредитных карт) и кредитов с длительным сроком (например, ипотека) лучше рассмотреть третий подход. Мы берем ежемесячные или ежеквартальные отчеты по когортному или винтажному анализу, имеющиеся в любом отделе кредитных рисков, анализируем динамику по платежам или просрочкам и строим график появления «плохих» случаев (случаев просрочки 90+, отказа от услуг) с течением времени.

Классический пример винтажного анализа для просроченной задолженности свыше 90 дней и 12-квартального (3-летнего) окна созревания приведен на рисунке ниже. Данные, выделенные жирным шрифтом, показывают текущий статус просрочки платежа на определенный отчетный период времени.

«Плохой» = 90 дней просрочки		Срок кредита (в кварталах)											
Дата открытия кредита		1 Qtr	2 Qtr	3 Qtr	4 Qtr	5 Qtr	6 Qtr	7 Qtr	8 Qtr	9 Qtr	10 Qtr	11 Qtr	12 Qtr
	Q1 13	0.00%	0.05%	1.10%	2.40%	2.80%	3.20%	3.50%	3.70%	3.80%	3.85%	3.85%	**3.86%**
	Q2 13	0.00%	0.06%	1.20%	2.30%	2.70%	3.00%	3.30%	3.50%	3.60%	3.60%	**3.60%**	
	Q3 13	0.00%	0.03%	0.90%	2.80%	3.20%	3.60%	4.10%	4.30%	4.40%	**4.45%**		
	Q4 13	0.00%	0.03%	1.00%	2.85%	3.20%	3.50%	3.80%	4.00%	**4.10%**			
	Q1 14	0.00%	0.04%	1.00%	2.20%	2.40%	2.70%	2.90%	**4.10%**				
	Q2 14	0.00%	0.05%	1.20%	2.50%	2.90%	3.30%	**3.50%**					
	Q3 14	0.00%	0.04%	1.30%	2.60%	3.00%	**3.35%**						
	Q4 14	0.00%	0.08%	1.40%	2.60%	**3.00%**							
	Q1 15	0.00%	0.02%	0.09%	**2.20%**								
	Q2 15	0.00%	0.08%	**1.50%**									
	Q3 15	0.00%	**0.05%**										
	Q4 15	**0.00%**											

Рис. 3 Пример когортного/винтажного анализа

Таблица имеет достаточно простую интерпретацию. Так, на первой строчке 2,8 % заемщиков, получивших кредит в первом квартале 2013 г., выпали в просрочку более 90 дней через 5 кварталов.

Несмотря на то что показатели просрочек схожи, мы видим, что в некоторых когортах показатели просрочек выше при одинаковой зрелости. Это нормальное явление, поскольку маркетинговые кампании, экономические циклы, изменения кредитной политики и другие факторы могут влиять на качество кредитов.

У нас есть несколько сценариев для построения кривой созревания просрочек по представленным данным.

Первый сценарий заключается в использовании значений по диагонали, выделенных жирным шрифтом. Здесь показаны самые последние показатели просроченной задолженности. В портфелях, винтажность которых может отличаться по качеству, это может привести к появлению кривых, которые не очень полезны (винтажная кривая не будет плавно расти, поскольку могут быть «провалы»), и, следовательно, это лучший вариант для продуктов, в которых качество заявителя и кредитного счета довольно стабильно, например ипотека.

В случае колебания показателей есть два дополнительных сценария, которые могут помочь сгладить числа и дать нам более реалистичную диаграмму роста

просрочек 90+ с течением времени. Например, мы можем использовать средние значения по последним четырем когортам, как показано овалами в рисунке, или мы можем выбрать одну отдельную когорту, например кредитные счета, открытые в первом квартале 2013 года, как показано прямоугольной рамкой.

«Плохой» = 90 дней просрочки	Срок кредита (в кварталах)											
Дата открытия кредита	1 Qtr	2 Qtr	3 Qtr	4 Qtr	5 Qtr	6 Qtr	7 Qtr	8 Qtr	9 Qtr	10 Qtr	11 Qtr	12 Qtr
Q1 13	0.00%	0.05%	1.10%	2.40%	2.80%	3.20%	3.50%	3.70%	3.80%	3.85%	3.85%	**3.86%**
Q2 13	0.00%	0.06%	1.20%	2.30%	2.70%	3.00%	3.30%	3.50%	3.60%	3.60%	**3.60%**	
Q3 13	0.00%	0.03%	0.90%	2.80%	3.20%	3.60%	4.10%	4.30%	4.40%	**4.45%**		
Q4 13	0.00%	0.03%	1.00%	2.85%	3.20%	3.50%	3.80%	4.00%	**4.10%**			
Q1 14	0.00%	0.04%	1.00%	2.20%	2.40%	2.70%	2.90%	**4.10%**				
Q2 14	0.00%	0.05%	1.20%	2.50%	2.90%	3.30%	**3.50%**					
Q3 14	0.00%	0.04%	1.30%	2.60%	3.00%	**3.35%**						
Q4 14	0.00%	0.08%	1.40%	2.60%	**3.00%**							
Q1 15	0.00%	0.02%	0.09%	**2.20%**								
Q2 15	0.00%	0.08%	**1.50%**									
Q3 15	0.00%	**0.05%**										
Q4 15	**0.00%**											

Рис. 4 Пример когортного/винтажного анализа: подходы к построению кривой созревания

Ниже показан график накопленного уровня просрочек 90+ для двух когорт: кредитных счетов, открытых в 1-м квартале 2013 года, и кредитных счетов, открытых в 1-м квартале 2014 года.

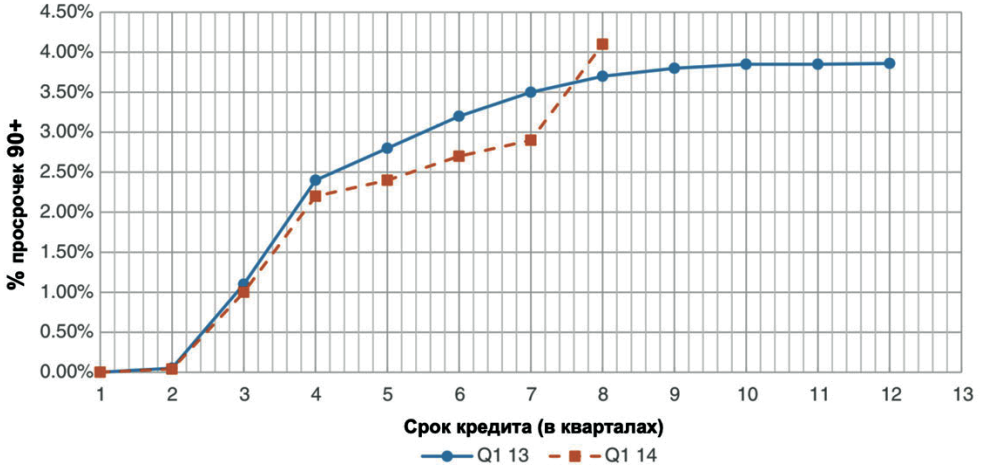


Рис. 5 Кривая винтажей

Перед нами пример типичного портфеля кредитных карт, где просрочки 90+ начинаются уже в самом начале и их количество растет с течением времени. Для счетов, открытых в первом квартале 2013 года, мы видим, что уровень просрочек 90+ быстро рос в первые несколько кварталов, а затем стабилизировался, когда срок кредитного счета стал приближаться к 10 кварталам. Для счетов, открытых во втором квартале 2014 года, мы видим, что уровень просрочек 90+ активно растет и о стабилизации говорить еще рано.

Мы сформируем выборку по такому периоду, когда можно считать, что уровень «плохих» случаев стабилизируется или когорта стала зрелой (т. е. когда накопленный уровень «плохих» случаев начинает выравниваться). В примере на рисунке выше хорошим окном выборки станут кредитные счета, которые были открыты 10–12 кварталов назад и дают 11-квартальное окно созревания.

Проектирование выборки по созревшей когорте осуществляется для того, чтобы минимизировать вероятность неправильной классификации клиентов (мы предоставляем всем клиентам достаточно времени, чтобы они могли стать «плохими») и убедиться в том, что определение «плохого» клиента, полученное по нашей выборке, не будет недооценивать итоговый ожидаемый процент «плохих» случаев. Например, если применительно к нашему примеру выборка будет спроектирована по кредитным счетам, открытым 4 квартала назад, мы увидим, что 2,4 % клиентов классифицированы как «плохие», однако уровень просрочек 90+ по-прежнему растет.

Поэтому некоторые кредитные счета, которые на самом деле являются «плохими», будут ошибочно помечены как «хорошие», если выборка будет спроектирована по 4-квартальному периоду.

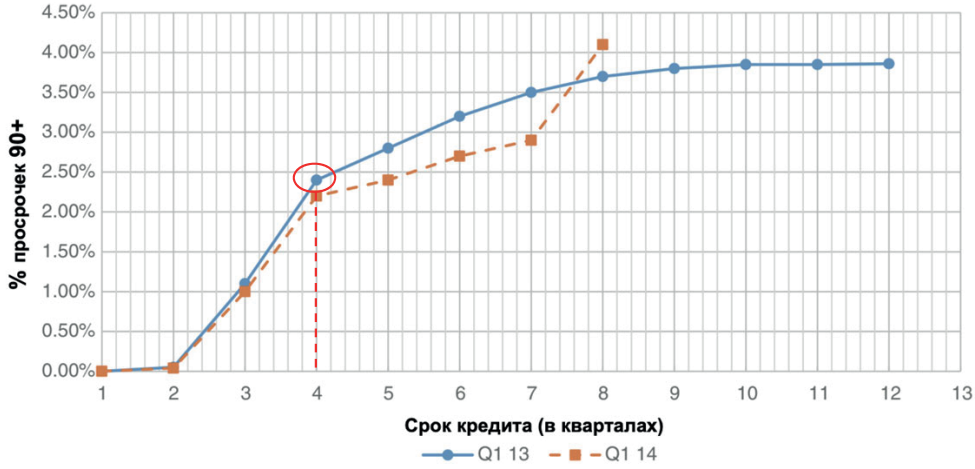


Рис. 6 Кривая винтажей: слишком короткое окно созревания

Временной горизонт «созревания» зависит от продукта, определения «плохого» клиента и конкурентной среды. Счета по кредитным картам считаются «зрелыми» после 12–18 месяцев, счета по автокредитам – обычно через 2–4 года, в то время как счета сроком от 4 до 5 лет считаются минимально допустимыми для разработки скоринговой карты по ипотечным кредитам. Поведенческие скоринговые карты предусматривают окно созревания из расчета 6–12 месяцев. Коллекторские модели строятся, как правило, на данных одного месяца, но все чаще компании строят такие карты для более коротких временных интервалов – до двух недель, чтобы облегчить разработку более подходящих способов взыскания долгов. По очевидным причинам счета по просрочкам 30+ будут становиться зрелыми быстрее, чем счета по просрочкам 90+. В условиях нестабильной макроэкономической ситуации, внутренних социально-экономических кризисов, как правило, счета по просрочкам также становятся зрелыми быстрее.

4. Определение зависимой переменной

Выбор зависимой переменной определяется целью построения прогнозной модели. Цели могут быть общими, например сокращение потерь по новым кредитным счетам, и конкретными, например сокращение числа дефолтов по одобренным заявкам в течение 6 месяцев после принятия положительного решения.

Зависимая переменная может быть количественной и категориальной.

В скоринге заявок зависимая переменная будет категориальной: погасит заемщик кредит («хороший») или не погасит («плохой»). Обычно к категории «плохой» относят клиентов, имеющих просроченную задолженность 90 дней и более. Этот период определяется требованиями банковского надзора. В соответствии с соглашением Базель II дефолт должника считается произошедшим, когда имело место одно или оба из следующих событий: банк считает, что должник не в состоянии полностью погасить свои кредитные обязательства перед банком без принятия банком таких мер, как реализация обеспечения (если таковое имеется); должник более чем на 90 дней просрочил погашение любых существенных кредитных обязательств перед банком.

Разумеется, банк может строить различные скоринговые карты с разными значениями зависимой переменной, вводя дополнительные критерии определения «плохого» и «хорошего» заемщика, а также меняя срок просрочки платежей. Примерами зависимой переменной могут быть наличие просроченной задолженности более 30 дней, 60 дней, 90 дней и более по одному кредиту на текущий момент или худший статус за все время кредитной истории, размер просроченной задолженности, глубина просрочки.

5. Загрузка данных из CSV-файлов и баз данных SQL

Загрузка CSV-файлов выполняется с помощью функции `pd.read_csv()`, а загрузку XLS-файлов можно выполнить с помощью функции `pd.read_excel()`.

Часто при загрузке CSV-файлов у нас возникают проблемы с кодировкой. Питоновский пакет chardet помогает определить тип кодировки прочитываемого файла.

```
# импортируем класс UniversalDetector библиотеки chardet
from chardet.universaldetector import UniversalDetector

# создаем экземпляр класса UniversalDetector
detector = UniversalDetector()

# определяем кодировку файла Credit_OTP.csv
with open('Data/Credit_OTP.csv', 'rb') as fh:
    for line in fh:
        detector.feed(line)
        if detector.done:
            break
    detector.close()
print(detector.result)
```

```
{'encoding': 'windows-1251', 'confidence': 0.8897840802899516, 'language': 'Russian'}
```

Библиотека pandas может считать данные из любой базы данных SQL, которая поддерживает адаптеры данных Python в рамках интерфейса Python DB-API. Чтение выполняется с помощью функций `pandas.read_sql()` и `pandas.read_sql_query()`, а запись в базу данных SQL выполняется с помощью метода `.to_sql()` объекта DataFrame.

Для иллюстрации программный код, приведенный ниже, считывает данные о котировках акций из файлов *msft.csv* и *aapl.csv*. Затем он подключается к файлу базы данных SQLite3. Если файл не существует, он создается «на лету». Затем программный код записывает данные MSFT в таблицу под названием STOCK_DATA. Если таблица не существует, она также будет создана. Если она уже существует, все данные заменяются данными о котировках акций MSFT. Наконец, программный код добавляет в эту таблицу данные о котировках акций AAPL.

```
# импортируем библиотеки pandas и SQLite
import pandas as pd
import sqlite3

# считываем данные о котировках акций из CSV-файла
msft = pd.read_csv('Data/msft.csv') msft['Symbol']='MSFT'
aapl = pd.read_csv('Data/aapl.csv') aapl['Symbol']='AAPL'

# создаем подключение
connection = sqlite3.connect('Data/stocks.sqlite')
# .to_sql() создаст базу SQL для хранения датафрейма в указанной таблице.
```

```
# параметр if_exists задает действие, которое нужно выполнить в том случае,
# если таблица уже существует ('fail' - выдать ошибку ValuError, 'replace' -
# удалить таблицу перед вставкой новых значений, 'append' - вставить
# новые значения в существующую таблицу)
msft.to_sql('STOCK_DATA', connection, if_exists='replace') aapl.to_sql('STOCK_DATA', con-
nection, if_exists='append')

# подтверждаем отправку данных в базу и закрываем подключение
connection.commit()
connection.close()
```

Чтобы убедиться в создании данных, можно открыть файл базы данных с помощью такого инструмента, как SQLite Database Browser (доступен по адресу *https://github.com/sqlitebrowser/sqlitebrowser*). Рисунок ниже показывает несколько записей в файле базы данных.

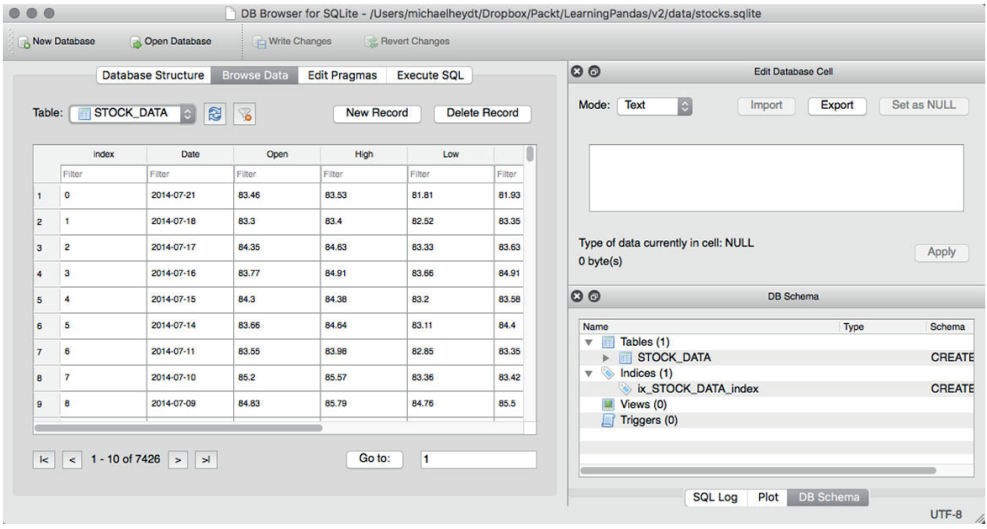


Рис. 7 SQLite Database Browser

Данные из базы данных SQL можно прочитать с помощью функции `pandas.read_sql()`. Следующий программный код демонстрирует выполнение запроса к файлу stocks.sqlite с помощью SQL и сообщает об этом пользователю.

```
# подключаемся к файлу базы данных
connection = sqlite3.connect('Data/stocks.sqlite')

# запрос всех записей в STOCK_DATA # возвращает датафрейм
# index_col задает столбец, который нужно сделать # индексом датафрейма
stocks = pd.read_sql("SELECT * FROM STOCK_DATA;",
   connection, index_col='index')

# закрываем подключение
connection.close()

# выводим первые 5 наблюдений в извлеченных данных
stocks[:5]
```

index	Date	Open	High	Low	Close	Volume	Symbol
0	7/21/2014	83.46	83.53	81.81	81.93	2359300	MSFT
1	7/18/2014	83.30	83.40	82.52	83.35	4020800	MSFT
2	7/17/2014	84.35	84.63	83.33	83.63	1974000	MSFT
3	7/16/2014	83.77	84.91	83.66	84.91	1755600	MSFT
4	7/15/2014	84.30	84.38	83.20	83.58	1874700	MSFT

Кроме того, для отбора столбцов еще можно использовать условие WHERE в SQL. Чтобы продемонстрировать это, следующий программный код отбирает записи, в которых количество проторгованных акций MSFT превышает 29 200 100.

```
# открываем подключение
connection = sqlite3.connect('Data/stocks.sqlite')
# создаем строку-запрос
query = "SELECT * FROM STOCK_DATA WHERE Volume>29200100 AND Symbol='MSFT';"
# выполняем подключение
items = pd.read_sql(query, connection, index_col='index')

# выводим результат запроса
items
```

index	Date	Open	High	Low	Close	Volume	Symbol
1081	5/21/2010	42.22	42.35	40.99	42.00	33610800	MSFT
1097	4/29/2010	46.80	46.95	44.65	45.92	47076200	MSFT
1826	6/15/2007	89.80	92.10	89.55	92.04	30656400	MSFT
3455	3/16/2001	47.00	47.80	46.10	45.33	40806400	MSFT
3712	3/17/2000	49.50	50.00	48.29	50.00	50860500	MSFT

Для этой же операции можно воспользоваться функцией `pandas.read_sql_query()`.

```
# можно воспользоваться функцией pandas.read_sql_query()
items2 = pd.read_sql_query(
    "SELECT * FROM STOCK_DATA WHERE Volume>29200100 AND Symbol='MSFT';",
    connection,
    index_col='index')
# закрываем подключение
connection.close()

# выводим результат запроса
items2
```

index	Date	Open	High	Low	Close	Volume	Symbol
1081	5/21/2010	42.22	42.35	40.99	42.00	33610800	MSFT
1097	4/29/2010	46.80	46.95	44.65	45.92	47076200	MSFT
1826	6/15/2007	89.80	92.10	89.55	92.04	30656400	MSFT
3455	3/16/2001	47.00	47.80	46.10	45.33	40806400	MSFT
3712	3/17/2000	49.50	50.00	48.29	50.00	50860500	MSFT

Итоговым моментом является то, что большая часть программного кода в этих примерах была программным кодом SQLite3. Библиотека pandas в этих примерах используется лишь тогда, когда нужно применить метод `.to_sql()`, функции `pandas.read_sql()` и `pandas.read_sql_query()`. Они принимают объект подключения, который может быть любым адаптером данных, совместимым с интерфейсом Python DB-API, поэтому вы можете работать с любой информацией базы данных, просто создав соответствующий объект подключения. Программный код на уровне pandas остается неизменным для любой поддерживаемой базы данных.