Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Д. В. ВИНОГРАДОВ

ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ

Учебное пособие

Владимир 2021

УДК 004.9
ББК 65.291.2
В49

Рецензенты:
Кандидат экономических наук, доцент
зав. кафедрой экономики и финансов Финансового университета
при Правительстве Российской Федерации
(Владимирский филиал)
Д. В. Кузнецов

Генеральный директор ООО «Индустриябетон»
Д. А. Кравченко

Виноградов, Д. В.
В49 Интеллектуальный анализ данных : учеб. пособие / Д. В. Виноградов ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2021. – 260 с.
ISBN 978-5-9984-1452-7

Рассмотрена совокупность эффективных подходов, инструментов и методов, направленных на обнаружение в исходных данных за приемлемое время и без потери точности ранее неизвестных, нетривиальных, практически полезных и доступных интерпретаций знаний, необходимых для принятия управленческих решений. Изложены современные методы прикладной математической статистики и интеллектуального анализа данных, методы обработки различных информационных объектов, методология проектирования и разработки программного обеспечения с использованием python-библиотек: NumPy, Pandas, matplottib, Scikit-Learn.

Предназначено для студентов направления подготовки бакалавров 38.03.05 – «Бизнес-информатика» по профилю «Информационно-аналитическое обеспечение предпринимательской деятельности» всех форм обучения; руководителей организаций и аналитических служб, а также специалистов, занимающихся вопросами интеллектуального анализа данных.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 158. Табл. 7. Библиогр.: 45 назв.

УДК 004.9
ББК 65.291.2

ISBN 978-5-9984-1452-7

ВВЕДЕНИЕ

Рассмотрена совокупность эффективных подходов, инструментов и методов, направленных на обнаружение в исходных данных за приемлемое время и без потери точности ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия управленческих решений.

Изложены современные методы прикладной математической статистики и интеллектуального анализа данных (задачи классификации, кластеризации, снижения размерности, прогнозирования и т. д.), методы обработки различных информационных объектов, методология проектирования и разработки программного обеспечения с использованием python-библиотек: NumPy, Pandas, matplottib, Scikit-Learn и TensorFlow.

На примерах рассмотрены особенности применения некоторых алгоритмов операционной аналитики: сбора, подготовки, исследования и моделирования данных.

В результате изучения материалов пособия у студента должны быть сформированы следующие профессиональные компетенции:

1) способность осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач;

2) способность использовать информацию, методы и программные средства ее сбора, обработки и анализа для информационно-аналитической поддержки принятия управленческих решений.

В результате освоения дисциплины студент должен демонстрировать следующие результаты образования:

знать:

1) основные методы сбора, подготовки, исследования и моделирования данных;

2) возможности различных методов интеллектуального анализа данных;

3) классы задач, для которых целесообразно использовать методы интеллектуального анализа данных;

4) варианты постановки и решения ключевых задач в бизнесе;

уметь:

1) понимать основные проблемы, возникающие при анализе данных, и пути их решения;

2) выбрать методы интеллектуального анализа данных, адекватные решаемой бизнес-задаче;

3) использовать python-библиотеки: NumPy, Pandas, matplottib, Scikit-Learn и TensorFlow;

владеть:

1) навыками анализа бизнес-данных различной природы;

2) методами сбора, подготовки, исследования и моделирования данных;

3) способами интеллектуального анализа данных применительно к конкретной задаче в бизнесе.

Теоретической основой при работе над учебным пособием послужили современные концепции, категории и понятия, ведущие мировые практики и стандарты, используемые в области интеллектуального анализа данных.

Учебное пособие выступает как основа воспитания экономического мышления, понимания современных задач и методов в области применения интеллектуального анализа данных и машинного обучения.

Глава 1. СУЩНОСТЬ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ И МАШИННОГО ОБУЧЕНИЯ

В главе рассматриваются следующие вопросы:

1. Сущность интеллектуального анализа данных

2. Интеллектуальный анализ данных в бизнесе

3. Технологии интеллектуального анализа

1.1. Сущность интеллектуального анализа данных

Стремительное развитие информационно-коммуникационных технологий и их повсеместное внедрение в деловую практику позволяет эффективно накапливать огромные массивы данных в различных сферах предпринимательской деятельности.

Обострение конкурентной борьбы на фоне высокой сложности внешней среды бизнеса делают крайне востребованными подходы к экспертному использованию накопленных данных для повышения эффективности принимаемых управленческих решений.

Размер накопленных данных, как правило, с одной стороны, превосходит возможности типичных баз данных по занесению, хранению, управлению и анализу информации, а с другой стороны – не позволяет эффективно применять стандартный набор средств прикладной статистики, машинного обучения (machine learning) и информационного поиска (information retrieval) для их обработки из-за проявления «проклятия размерности».

В последние годы усилия в области использования накопленных данных были направлены:

1) на разработку методов хранения и извлечения информации в больших массивах (концепция Big Data);

2) на создание специализированных алгоритмов, способных обнаруживать в исходных данных за приемлемое время и без потери точности ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия управленческих решений (концепция Data Mining).

Концепция Big Data предполагает использование совокупности эффективных подходов, инструментов и методов обработки непрерывно прирастающих данных огромных объёмов, которые распреде-

лены по многочисленным узлам вычислительной сети, с целью получения воспринимаемых человеком результатов.

С точки зрения информационных технологий Big Data – совокупность подходов и инструментов массово-параллельной обработки неопределённо структурированных данных с помощью разнообразных информационно-технологических решений (например, системами управления базами данных категории NoSQL, алгоритмами MapReduce и реализующими их программными каркасами и библиотеками проекта Hadoop) [1].

Data Mining является ключевым элементом Big Data и представляет собой совокупность подходов, инструментов и методов обнаружения в больших массивах данных, накапливающихся в информационных системах компаний, ранее неизвестных, нетривиальных, практически полезных и доступных для интерпретации знаний, необходимых для принятия управленческих решений [2].

Интеллектуальный анализ данных расширяет концепцию Data Mining за счет использования подходов, инструментов и методов извлечения данных из различных источников, их консолидации, профайлинга, трансформации, предобработки, очистки и обогащения.

В широком смысле интеллектуальный анализ данных – это современная концепция анализа данных, предполагающая, что [2]:

1) данные могут быть неточными, неполными (содержать пропуски), противоречивыми, разнородными, косвенными, и при этом иметь гигантские объёмы; поэтому понимание данных в конкретных приложениях требует значительных интеллектуальных усилий;

2) сами алгоритмы анализа данных могут обладать «элементами интеллекта», в частности, способностью обучаться по прецедентам, то есть делать общие выводы на основе частных наблюдений;

3) процессы переработки сырых данных в информацию, а информации в знания уже не могут быть выполнены по старинке вручную, и требуют нетривиальной автоматизации.

1.2. Интеллектуальный анализ данных в бизнесе

Наибольший интерес к технологиям интеллектуального анализа данных, в первую очередь, проявляют хозяйствующие субъекты, работающие в условиях высокой конкуренции и имеющие четкую груп-

пу потребителей (розничная торговля, банковская и страховая деятельность, связь) [2].

Возможность повышения эффективности собственного бизнеса данные участники рынка видят в выявлении связи между факторами внутренней и внешней среды и подготовки на основе выявленных закономерностей рекомендаций для принятия управленческих решений. При этом такие рекомендации не должны избыточно «погружать» лицо, принимающего решение, в детали базовых данных или промежуточной аналитики.

Для розничной торговли среди прикладных задач интеллектуального анализа данных выделяют (современные предприятия розничной торговли собирают сведения о каждой покупке каждого своего покупателя, используя карты клиента, а также компьютеризованные системы контроля и оплаты):

1) анализ портрета покупателя и формирование портрета целевой аудитории (позволяют сделать индивидуализированные предложения отдельным покупателям или их группам, а также определить способы продвижения товаров);

2) анализ покупательской корзины и формирование предложений относительной дальнейших покупок (позволяет выработать стратегии создания запасов товаров, а также определить способы их раскладки в торговых залах).

Для банковской деятельности среди прикладных задач интеллектуального анализа данных выделяют:

1) анализ платежных транзакций и выявление мошенничества с банковскими картами;

2) анализ портрета клиента и их сегментирование (выявление потенциально неблагонадежных заёмщиков, формирование портрета целевой аудитории и т.д.).

Для деятельности в области телекоммуникаций среди прикладных задач интеллектуального анализа данных выделяют (предприятия данного сектора экономики собирают информацию о параметрах использования связи, доступа в интернет, геолокации и др.):

1) анализ портрета клиента и формирование портрета целевой аудитории (позволяют сделать индивидуализированные предложения отдельным покупателям или их группам, определить способы про-

движения услуг, а также использовать при оказании рекламных услуг);

2) анализ поведения клиента и формирование предложений по повышению уровня его лояльности.

Для деятельности в области страхования среди прикладных задач интеллектуального анализа данных выделяют (предприятия данного сектора экономики собирают информацию о социальном и имущественном положении своих клиентов и страховых случаях):

1) анализ портрета клиента и формирование портрета целевой аудитории (позволяют сделать индивидуализированные предложения отдельным потребителям или их группам, определить способы продвижения услуг);

2) анализ страховых событий и выявление мошенничества с возмещением вреда по страховкам;

3) анализ риска и формирование страховых продуктов, отвечающих требованиям доходности.

Технологии интеллектуального анализа данных применяются и в других сферах, в которых возникают схожие задачи:

1) анализ рыночных корзин;

2) управление взаимоотношениями с клиентами;

3) анализ текстовой информации;

4) анализ информации, порождаемой в сети Интернет;

5) анализ клиентских сред;

6) маркетинговые исследования.

1.3. Технологии интеллектуального анализа

Технологии интеллектуального анализа предполагают не только обнаружение в больших массивах данных, накапливающихся в информационных системах компаний, ранее неизвестных, нетривиальных, практически полезных и доступных для интерпретации знаний, необходимых для принятия управленческих решений ответственным лицом, но и предусматривают возможность принятие управленческих решений автоматически.

Операционная аналитика – это интегрированные автоматические процессы принятия решений, предписывающие и реализующие действия в пределах «времени принятия решения».

Операционная аналитика интегрирует аналитику в бизнес-процессы и автоматизирует принятие решений без участия человека. Например, правильно настроенный рекомендательный алгоритм на сайте интернет-магазина гораздо лучше любого человека-продавца умеет предлагать покупателю дополнительные сервисы и покупки.

Операционная аналитика подразумевает исследование сверхбольших разнородных массивов цифровой информации в неразрывной связи с информационными технологиями, обеспечивающими их обработку и применение в практической деятельности.

В сравнении с деятельностью в области проектирования и работы с базами данных, где предполагается предварительное проектирование модели данных, отражающей взаимосвязи предметной области и последующее исследование загруженных данных относительно простыми (арифметическими) методами, в операционной аналитике предполагается опора на аппарат математической статистики, искусственного интеллекта, машинного обучения, зачастую без предварительной загрузки данных в модели.

Типичный процесс операционной аналитики состоит из нескольких выполняемых шагов (организация этих процессов во времени зависит от решаемых практических задач и может быть последовательной, итерационной или гибкой): назначение цели, сбор данных, подготовка данных, исследование данных, моделирование данных, автоматизация бизнес-процессов.

Основным процессам операционной аналитики предшествует процедура создания проектного задания, в котором как минимум определяются следующие элементы [3]:

1) цель операционной аналитики; предназначение и содержание операционной аналитики;

2) предварительное описание методики анализа;

3) перечень требуемых ресурсов;

4) доказательство практической реализуемости проекта (или возможности проверки концепции);

5) предполагаемые результаты и критерий успеха; календарный план.

Первым основным шагом операционной аналитики является сбор данных, который, как правило, производится автоматически из

внутренних и внешних источников в целом (пакетом) или по мере поступления.

В качестве таких источников могут выступать системы управления основными и вспомогательными процессами предприятия (например, данные о заключённых сделках, информация о бракованной продукции, документы, информация о динамике цен, обращений и т.д.) или специализированные базы данных, предлагаемые на рынке платно или предоставляемые безвозмездно, в т.ч. в форме открытых данных.

Результатом реализации данного шага является аккумулированный набор данных, как правило, разнородных, неполных и содержащих ошибки и неточности.

Следующим шагом является подготовка данных для того, что бы их можно было использовать в дальнейшем анализе. Данный шаг предполагает выявление и исправление данных, которые в каком-либо смысле не удовлетворяют определённым критериям качества (очистка данных), а также преобразование данных (оптимизация данных).

Качество данных – обобщенное понятие, отражающее степень их пригодности к решению определенной задачи. Основными критериями качества данных являются: доступность, точность, взаимосвязанность, полнота, непротиворечивость, однозначность, релевантность, надежность и своевременность.

Соответственно, очистка данных включает обработку пропущенных значений, дубликатов, противоречий, аномальных значений и выбросов, шумов, фиктивных значений и ошибок ввода данных [4].

Для обработки пропущенных значений используются следующие методы: исключение пропущенных значений, присваивание значения null, присваивание статического значения (например, 0 или среднего арифметического), вычисление значения на основании предполагаемого или теоретического распределения и независимое моделирование значения. Применение каждого из методов зависит от конкретной практической ситуации.

Для обработки дубликатов используется метод исключения, так как дубликаты обедняют обучающее множество в информационном плане. Однако, в некоторых случаях, для повышения эффективности

обучения модели дубликаты целесообразно оставлять или, более того, специально добавлять.

Для обработки противоречий используют следующие методы: исключение противоречивых значений (применяется в том случае, если причина противоречия вызвано ошибкой) и объединение записей с агрегированием числовых значений выходных атрибутов (применяется в том случае, если данные отражают реальные события).

Аномальные значения и выбросы являются следствием ошибок в данных, воздействия случайных и не поддающихся прогнозированию факторов, а также присутствия объектов «других» выборок. Для обработки аномальных значений и выбросов используют следующие методы: исключение аномальных значений и выбросов, подавление аномальных значений и выбросов.

Для обработки шумов (обусловленных воздействием случайных факторов флуктуаций значений признаков) используют метод их описания с помощью различных математических моделей в соответствии с их временной, спектральной и пространственной структурой с последующим исключением из основного набора данных.

Кроме этого, в процессе очистки восстанавливаются нарушения структуры, полноты и целостности данных, преобразуются некорректные форматы.

Оптимизация данных как элемент предобработки включает снижение размерности, а также выявление и исключение незначащих признаков [4]. Наличие в исходном наборе данных избыточных, неинформативных или слабо информативных признаков может понизить эффективность последующего анализа, а после такого преобразования она упрощается [5].

Уменьшение размерности может быть осуществлено методами выбора признаков или выделения признаков.

Методы выбора признаков оставляют некоторое подмножество исходного набора признаков, избавляясь от признаков избыточных и слабо информативных.

К методам выбора признаков относят:

1) методы фильтров (измеряют релевантность признаков на основе функции μ, и затем решают по правилу κ, какие признаки оставить в результирующем множестве);

2) оберточные методы (находят подмножество искомых признаков последовательно, используя некоторый классификатор в качестве источника оценки качества выбранных признаков);

3) встроенные методы (для выбора признаков используется непосредственно структуру некоторого классификатора) и другие.

Методы выделения признаков составляют из уже исходных признаков новые, все также полностью описывающие пространство набора данных, но уменьшая его размерность и теряя в репрезентативности данных, т.к. становится непонятно, за что отвечают новые признаки.

К методам выбора признаков относят:

1) метод главных компонент;

2) метод разложения по усеченным сингулярным значениям;

3) метод неотрицательного матричного разложения;

4) метода линейного дискриминантного анализа и другие.

Более подробно данные методы будет рассмотрены далее.

На четвертом этапе выполняется исследование данных с помощью различных визуальных и описательных методов с целью выявления скрытых закономерностей (например, корреляцию и отклонения), которые могут присутствовать в данных.

При наличии подготовленных данных и хорошем их понимании переходят к следующему этапу – моделированию данных. Для моделирования данных используют методы машинного обучения, анализа данных и статистики.

Машинное обучение – класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач. В общем, задача обучения рассматривает набор из n выборок данных, а затем пытается предсказать свойства неизвестных данных.

Распространенной практикой в машинном обучении является оценка эффективности работы алгоритма машинного обучения путем разделения набора данных на два: обучающий (предназначен для изучения свойств данных) и тестовый (предназначен для проверки изученных свойств).

Задачи машинного обучения делятся на несколько категорий:

1) обучение с учителем (обучение осуществляется принудительно с помощью примеров «стимул-реакция»);

2) обучение без учителя (обучение осуществляется спонтанно без вмешательства извне);

3) обучение с подкреплением (обучение осуществляется при взаимодействии с некоторой средой).

Все задачи, решаемые с помощью машинного обучения, относятся к одной из следующих категорий:

1) регрессия (является частным случаем задач прогнозирования, выполняется с помощью обучения с учителем на этапе тестирования);

2) классификация (выполняется с помощью обучения с учителем на этапе собственно обучения);

3) кластеризация (выполняется с помощью обучения без учителя);

4) понижение размерности данных (выполняется с помощью обучения без учителя);

5) выявление аномалий (выполняется с помощью обучения без учителя).

Моделирование является итеративным процессом. Процесс построения большинства моделей состоит из следующих шагов:

1) выбор метода моделирования и переменных для включения в модель;

2) выполнение модели;

3) диагностика и сравнение моделей;

4) выбор лучшей модели на основании ряда критериев (метрик, которые различаются между собой при решении различных аналитических задач).

Заключительным этапом является автоматизация анализа и принятия управленческих решений на его основе, суть которого заключается в адаптации процесса анализа для повторного использования и интеграции его в существующие бизнес-процессы предприятия.

Например, автоматизация анализа данных о покупках всех пользователей в интернет-магазине и формирование рекомендаций отдельным пользователям относительно возможных покупок.

Вопросы для обсуждения

1. Области человеческой деятельности наиболее и наименее подходят для их анализа методами интеллектуального анализа

2. Проблемы организации процессов операционной аналитики

3. Практические аспекты постановки и решения задачи машинного обучения с учителем

4. Практические аспекты постановки и решения задачи машинного обучения без учителя

5. Практические аспекты постановки и решения задачи машинного обучения с подкреплением

6. Организация различных способов сбора данных и их программная реализация

7. Проблемы оценки качества собранных данных

8. Целесообразность применения и выбор методов очистки и оптимизации данных при решении различных категорий задач машинного обучения

9. Подходы к тестированию разработанных моделей машинного обучения

Практические задания

Задание 1.1.

Сформулируйте задачи, возникающие в заданном по вариантам виде деятельности, которые можно было бы решить с использованием машинного обучения (необходимо выделить как минимум по одной задачи регрессии, классификации и кластеризации).

Опишите каждую задачу по следующей схеме: сущность задачи, класс задачи, состав признаков (наименование, тип данных, ограничения на значения), состав меток (наименование, тип данных, ограничения на значения).

Варианты видов деятельности:

1) банковская деятельность;
2) электронная коммерция;
3) риэлтерская деятельность;
4) информационная безопасность;
5) розничная торговля;
6) сельское хозяйство;
7) транспортные услуги;
8) туристические услуги;
9) услуги страхования;
10) бытовые услуги.

Библиографический список

1. И.Б. Тесленко, А.М. Губернаторов, В.Е. Крылов. Введение в Big Data : учеб. пособие / И.Б. Тесленко; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2021.

2. Замятин А. В. Введение в интеллектуальный анализ данных: учебное пособие / А. В. Замятин; Нац. исслед. Том. гос. ун-т. – Томск: Издательский Дом Томского государственного университета, 2016. URL:http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000529594

3. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

4. Предобработка данных // Аналитическая платформа Loginom: [Электронный ресурс]. – Режим доступа: https://wiki.loginom.ru/articles/data-preprocessing.html

5. Уменьшение размерности // Университет ИТМО: [Электронный ресурс]. – Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Уменьшение_размерности

6. Лесковец, Ю. Анализ больших наборов данных / Лесковец Ю., Раджараман А., Джеффри Д. Ульман - Москва : ДМК Пресс, 2016. - 498 с. - ISBN 978-5-97060-190-7. - Текст : электронный // ЭБС "Консультант студента": [сайт]. – URL: https://www.studentlibrary.ru/book/ISBN9785970601907.html

7. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

8. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

9. Машинное обучение // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Машинное_обучение

Глава 2. ОСНОВЫ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ С ПОМОЩЬЮ PYTHON

В данной главе рассматриваются следующие вопросы:

1. Обзор программных решений для интеллектуального анализа данных

2. Основные программные конструкции языка Python

2.1. Обзор программных решений для интеллектуального анализа данных

В настоящее время существует много разных инструментариев и инфраструктур для больших данных. Экосистема больших данных может быть разбита на группы по технологиям с похожими целями и функциональностью: распределенная файловая система, инструменты планирования, инструменты развертывания системы, инструменты программирования служб, безопасность, базы данных NoSQL и NewSQL, инструменты интеграции данных, инструменты интеллектуального анализа данных (машинного обучения) и прочее службы и инструменты.

В качестве инструментария осуществления машинного обучения в Big Data применяются следующие решения: Mahout, MLib, ML Grid, SAMOA, MADLib, библиотеки языка R, библиотеки .NET Framework, библиотеки языка Python и другие.

Mahout (Apache Mahout**)** – это проект Apache Software Foundation по созданию бесплатных реализаций распределенных или иным образом масштабируемых алгоритмов машинного обучения, ориентированных в первую очередь на линейную алгебру. Apache Mahout содержит алгоритмы для обработки данных, такие как фильтрация, классификация и кластеризация.

В прошлом многие реализации использовали платформу Apache Hadoop, однако сегодня она в основном ориентирована на Apache Spark. Mahout также предоставляет библиотеки языков Java и Scala для общих математических операций (ориентированных на линейную алгебру и статистику) и примитивных коллекций Java. Mahout находится в стадии разработки и в настоящий момент реализован только ряд алгоритмов.

MLib (Apache Spark MLib) библиотека алгоритмов машинного обучения, адаптированных к модели MapReduce, которые могут выполняться на платформе Apache Spark.

Apache Spark MLib включает: 1) алгоритмы классификации, регрессии, кластеризации и фильтрации; 2) алгоритмы извлечения признаков, преобразования, уменьшения размерности и выбор признаков; 3) алгоритмы построения, оценки и настройки конвейеров машинного обучения; 4) алгоритмы линейная алгебра, статистики, обработки данных и др.

MLlib включен в API Spark и взаимодействует с библиотеками языков Python и R, позволяя создавать высокоуровневые приложения, связанные с обработкой и анализом больших данных, а также операционной аналитикой, при этом использовав все возможности распределенных вычислений.

Apache Spark MLib позволяет использовать любой источник данных Hadoop (например, HDFS, HBase или локальные файлы), что упрощает подключение к рабочим процессам Hadoop. Apache Spark MLib легко интегрируется с другими компонентами Spark, такими как Spark SQL, Spark Streaming и DataFrames.

ML Grid библиотека алгоритмов машинного обучения, входящих в состав Apache Ignite, которая является горизонтально масштабируемой отказоустойчивой распределенной вычислительной платформой с открытым исходным кодом, предназначенная для хранения и вычисления больших объемов данных в кластере узлов. Обработка больших массивов данных осуществляется в оперативной памяти в режиме реального времени.

Преимуществом Apache Ignite ML Grid является возможность создания простых, масштабируемых и эффективных инструментов, которые позволяют создавать модели машинного обучения без дорогостоящей передачи данных.

Apache Ignite ML Grid включает все основные алгоритмы машинного обучения: классификации, регрессии, кластеризации, рекомендации и предварительной обработки.

API ML Grid позволяет взаимодействовать с языками высокого уровнями, такими как Java, C#, C++, Scala, Python, PHP и другие.

SAMOA (Apache SAMOA) – это распределенная среда, которая содержит алгоритмы распределенного потокового машинного обуче-

ния. SAMOA имеет адаптеры для выполнения алгоритмов на различных платформах, реализующих модель MapReduce. SAMOA позволяет разработчикам интегрировать существующие алгоритмы из библиотеки MOA.

MADlib (Apache MADlib) – это библиотека с открытым исходным кодом на основе SQL для масштабируемой аналитики в базах данных. Данный инструмент обеспечивает параллельную реализацию математических, статистических, графических методов и методов машинного обучения для структурированных и неструктурированных данных.

Apache MADlib включает все основные алгоритмы машинного обучения: классификации, регрессии, кластеризации, рекомендации и предварительной обработки. Поддерживает интеграцию с Python и C ++.

Все перечисленные выше инструменты машинного обучения используют различные платформы распределенных вычислений, основанных на модели MapReduce. Это даёт серьёзные преимущества в организации распределенных вычислений. Однако, адаптация алгоритма к данной модели сложна и трудоемка. В результате: количество алгоритмов, в том числе инновационных, не значительно.

Выход из такой ситуации заключается в создании библиотек на языках высокого уровня или использование технологий облачных вычислительных сред.

R – язык программирования для статистической обработки данных и работы с графикой. Представляет собой свободную программную среду для вычислений с открытым исходным кодом. Наиболее сильная сторона языка как раз и заключается в возможности неограниченного расширения за счет создания новых пакетов (библиотек).

В R в библиотеках caret, mlr3, h2o, pipelearner реализованы практически все актуальные средства машинного обучения. Разработано ряд библиотек, которые позволяют объединить мощные функции анализа данных и визуализации с возможностями, представляемыми Hadoop, MapReduce, HDFS и HBase.

Однако у языка есть серьёзные недостаток в плане применения в области операционной аналитики. Язык R не предназначен для создания бизнес-приложений. Поэтому результаты анализа должны быть интегрированы с внешними информационными системами (реализо-

ванными на других языках высокого уровня, таких как, например, C#, Java и Python) тем или минным образом, что не может не сказаться на производительности.

NET Framework – программная платформа, основанная на общеязыковой среде исполнения Common Language Runtime (CLR). Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду (C#, Visual Basic .NET, JScript .NET, F# и J#).

В состав NET Framework включен пакет ML.NET, который позволяет добавлять в приложения .NET возможности машинного обучения в автономном и подключенном режимах. В ML.NET реализованы практически все актуальные средства машинного обучения.

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой набор полезных функций. При этом за счет поддержки разбиения программ на модули позволяет их объединять в пакеты (библиотеки) и распространять среди разработчиков [1].

Python – активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года

Удобство, гибкость и простота работы, а также широкий выбор библиотек, обширная поддержка и качественная документация привила к тому, что Python вошел к настоящему моменту в число наиболее популярных языков программирования.

Существует множество библиотек языка Python с надежными реализациями широкого диапазона алгоритмов машинного обучения, которые можно условно разделить на три группы: 1) библиотеки, предназначенные для использования Python с технологиями больших данных (PySpark, PyDoop, Hadoopy); 2) библиотеки, предназначенные для оптимизации кода (PyCUDA, Cython, Blaze, PP, Dispy, IPCluster и другие); 3) библиотеки, предназначенные для работы с данными (Pandas, matplottib, NumPy, SymPy, SciPy, Scikit-Learn, TensorFlow, Stats-Models, RPy2, NLTK и другие). Краткий обзор основных пакетов приведен в табл. 2.1

Таблица 2.1

Краткий обзор основных пакетов Python, предназначенных для машинного обучения

Наименование библиотеки	Описание библиотеки
Pydoop Hadoopy	Связывают Python с Hadoop, популярной инфраструктурой больших данных
PySpark	Связывает Python и Spark, инфраструктуру для работы с большими данными в памяти
PyCUDA	Позволяет писать код, который будет выполняться на графическом процессоре вместо центрального процессора, а следовательно, идеально подходит для интенсивных вычислений
Cython	Открывает возможность использования языка программирования C из кода Python
Blaze	Предоставляет структуры данных, размер которых может превышать объем памяти компьютера, и позволяет работать с большими объемами данных.
PP	Позволяет выполнять вычисления на одном компьютере или в кластерах параллельно
Dispy IPCluster	Позволяет писать код, который может распределяться в компьютерном кластере
Pandas	Высокопроизводительный, но простой в использовании пакет обработки данных. Позволяет использовать концепцию фреймов данных
matplottib	Предоставляет возможность визуализации данных (2D-график, ограниченная 3D-функциональность)
NumPy	Предоставляет доступ к мощным функциям массивов и функциям линейной алгебры
SymPy	Предоставляет возможность решения задач символической математики и компьютерной алгебры
SciPy	Библиотека, интегрирующая фундаментальные пакеты, часто используемые в научных вычислениях (такие, как NumPy, matplotlib, Pandas и SymPy)
Scikit-Learn	Предоставляет возможность работы с алгоритмами машинного обучения
TensorFlow	Предоставляет возможность решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов
StatsModels	Предоставляет возможность работы с широким набором статистических методов и алгоритмов
RPy2	Позволяет вызывать функции R из кода Python
NLTK	Предоставляет возможность анализа текстовых данных

Пакет Scikit-Learn – один из самых известных и надежных инструментов, позволяющий работать с широким диапазоном алгоритмов машинного обучения. Особенностью этого пакета является аккуратный, продвинутый и единообразный API, которые позволяет легко осваивать работу с различными методами машинного обучения.

Scikit-learn предоставляет совокупность алгоритмов, которые предназначены для предварительной обработки, уменьшения размерности, анализа и моделирования данных, а также тестирования разработанных моделей.

Scikit-learn не поддерживает нейронные сети; самоорганизующиеся карты (сети Кохонена); обучение ассоциативным правилам и обучения с подкреплением.

Далее мы рассмотрим реализацию основных этапов интеллектуального анализа больших данных с применением python-библиотек: NumPy, Pandas, matplottib, Scikit-Learn, TensorFlow.

2.2. Основные программные конструкции языка Python

2.2.1. Общие сведения о языке Python

Python – интерпретируемый язык. Интерпретатор Python исполняет программу по одному предложению за раз. Это обуславливает зачастую более низкую скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на C или C++.

Python является полностью объектно-ориентированным языком. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации.

Программа на языке Python представляет собой обычный текстовый файл с инструкциями.

Каждая инструкция располагается на отдельной строке (концом инструкции является либо конец строки, либо точка с запятой (это вариант не рекомендуется использовать)):

Каждая инструкция (кроме инструкций внутри блоков) должна начинаться строго с начала строки, иначе будет выведено сообщение об ошибке (такая ситуация продемонстрирована на рис. 2.1):

Рис. 2.1. Расположение инструкций на языке Python

Каждая инструкция внутри блоков должна начинаться строго с одинакового количества пробелов с начала строки (именно так в языке Python выделяются инструкции образующие тело блоков, для выделения инструкций внутри блока принято использовать четыре пробела). Расположение инструкций внутри блоков на языке Python приведено на рис. 2.2.

Рис. 2.2. Расположение инструкций внутри блоков на языке Python

Если блок состоит из одной инструкции, то допустимо поместить ее на одной строке с основной инструкцией.

Если инструкция является слишком длинной, то ее можно перенести на следующую строку, одним из следующих способов:

1) в конце строки поставить символ «\», после которого должен следовать перевод строки;
2) поместить выражение внутри круглых скобок.

Для вставки пояснений в текст программы на языке Python допускается использовать комментарии. Внутри комментария может

располагаться любой текст, включая инструкции, которые выполнять не следует (интерпретатор полностью их игнорирует).

В языке Python присутствует только однострочный комментарий. Он начинается с символа «#».

2.2.2. Переменные, типы и коллекции данных

Все данные в языке Python представлены объектами. Каждый объект имеет тип данных и значение. Для доступа к объекту предназначены переменные. При инициализации в переменной сохраняется ссылка на объект (адрес объекта в памяти компьютера). Благодаря этой ссылке можно в дальнейшем изменять объект из программы.

Каждая переменная должна иметь уникальное имя, состоящее из латинских букв, цифр и знаков подчеркивания, причем имя переменной не может начинаться с цифры. Кроме того, следует избегать указания символа подчеркивания в начале имени, поскольку идентификаторам с таким символом определено специальное назначение.

В качестве имени переменной нельзя использовать ключевые слова (список всех ключевых слов можно получить запустив следующий код – import keyword; keyword.kwlist). При указании имени переменной важно учитывать регистр букв: x и X – разные переменные.

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем».

Тип данных в языке Python – это характеристика объекта, а не переменной. Переменная всегда содержит только ссылку на объект.

В Python имеются встроенные типы: булевый, строка, Unicode-строка, целое число произвольной точности, число с плавающей запятой, комплексное число и некоторые другие.

Из коллекций в Python встроены: список, кортеж (неизменяемый список), словарь, множество и другие. Все значения являются объектами, в том числе функции, методы, модули, классы.

Список представляет собой упорядоченную последовательность элементов. Он очень гибкий и является одним из самых используемых типов в Python. Элементы списка не обязательно должны быть одного типа.

Объявление и инициализация списка осуществляется помещением разделённых запятой элементов списка внутрь квадратных скобок. Для извлечения элемента списка по индексу или диапазона элементов (среза) используется оператор «[]» (в Python индексация начинается с нуля).

Объявление и инициализация списка, а также операция извлечения элемента списка по индексу или диапазона приведена на рис. 2.3.

```
В [36]: a = [1, 3.9, 'python', True]
        print ('Второй элемент списка -', a[1])
        print ('Элементы списка с первого до второго -', a[1:3])
        print ('Все элементы списка, начиная со второго -', a[1:])
        print ('Все элементы списка, до третьего -', a[:2])

        Второй элемент списка - 3.9
        Элементы списка с первого до второго - [3.9, 'python']
        Все элементы списка, начиная со второго - [3.9, 'python', True]
        Все элементы списка, до третьего - [1, 3.9]
```

Рис. 2.3. Работа со списками данных на языке Python

Списки являются изменяемым типом, т.е. значения его элементов можно изменить.

Кортеж, так же как и список, является упорядоченной последовательностью элементов. Вся разница заключается в том, что кортежи неизменяемы.

Кортежи используются для защиты данных от перезаписи и обычно работают быстрее, чем списки, т.к. их нельзя изменять.

Объявление и инициализация кортежа осуществляется помещением разделённых запятой элементов списка внутрь круглых скобок.

Множество является неупорядоченной уникализированной последовательностью. Над множествами можно выполнять такие операции, как объединение и пересечение. Так как элементы во множестве должны быть уникальны, они автоматически удаляют дубликаты. Поскольку множество является неупорядоченной последовательностью, оператор «[]» с с множествами не работает.

Объявление и инициализация кортежа осуществляется помещением разделённых запятой элементов списка внутрь фигурных скобок.

Словари – неупорядоченные наборы пар ключ-значение. Значение может быть любого типа, а вот ключ – только неизменяемого.

Данный тип данных используется, когда нужно сопоставить каждому из ключей значение и иметь возможность быстро получать доступ к значению, зная ключ. Словари оптимизированы для извлечения данных. Чтобы извлечь значение, нужно знать ключ.

Объявление и инициализация словаря осуществляется помещением разделённых запятой пар «ключ:значение» внутрь фигурных скобок. Объявление и инициализация словаря, а также операция извлечения элемента списка приведена на рис. 2.4.

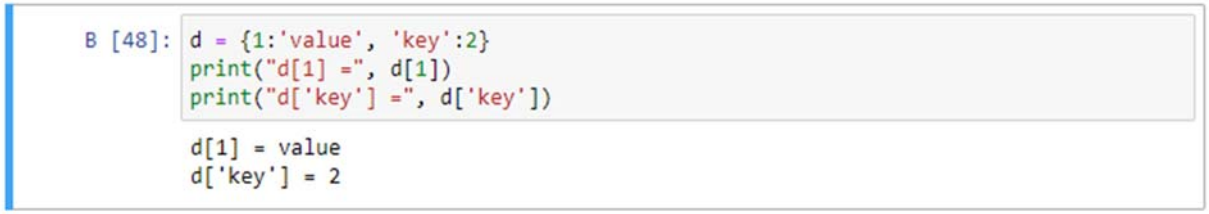


Рис. 2.4. Работа со словарем данных на языке Python

Диапазон представляет собой неизменяемую последовательность целых чисел от одного какого-то значения до другого.

Используется в циклах for для выполнения определенного количества итераций на основе значения, которое уменьшается или увеличивается на значение шага, пока не будет достигнуто граничное значение. Значение по умолчанию для шага не может быть 0 (по умолчанию – 1).

Объявление и инициализация диапазона осуществляется с применением функции range(), в качестве параметров которой передается информация о диапазоне: начальное значение, конечное значение и шаг. Создание диапазона приведено на рис. 2.5.

Рис. 2.5. Работа с диапазоном данных на языке Python

Присваивание значения переменной (или имени) в Python приводит к созданию ссылки на объект, стоящий в правой части присваивания.

В Python есть несколько встроенных функций, которые позволяют преобразовать данные из одного типа в другой (перечень функций преобразования не полный):

1) bool ([<объект>]) – преобразует объект в логический тип данных;

2) int ([<Объект> [, «Система счисления>]]) – преобразует объект в число (во втором параметре можно указать систему счисления (значение по умолчанию – 10));

3) float ([<Число или строка>]) – преобразует целое число или строку в вещественное число;

4) str ([<Объект>]) – преобразует объект в строку;

5) list (<Последовательность>) – преобразует элементы последовательности в список;

6) tuple (<Последовательность>) – преобразует элементы последовательности в кортеж.

Пример использования функций преобразования типов приведен на рис. 2.6

```
B [65]: print (bool(0), bool(1))
        print (int(1.303), int("777"))
        print (float("1.303"), float(777))
        print (str(125), str({12, 78, 79}))
        print (list ("12345"), list ((1, 2, 3, 4, 5)))
        print (tuple ("1234567"), tuple ((1, 2, 3, 4, 5, 6)))

        False True
        1 777
        1.303 777.0
        125 {12, 78, 79}
        ['1', '2', '3', '4', '5'] [1, 2, 3, 4, 5]
        ('1', '2', '3', '4', '5', '6', '7') (1, 2, 3, 4, 5, 6)
```

Рис. 2.6. Работа с функциями преобразования типа на языке Python

Операторы позволяют произвести с данными определенные действия. Например, операторы присваивания служат для сохранения данных в переменной, математические операторы позволяют выполнить арифметические вычисления, а оператор конкатенации строк служит для соединения двух строк в одну.

В Python существуют следующие типы операторов:

1) арифметические операторы («+», «-», «*», «/», «%», «**», «//»);

2) операторы сравнения («==», «!=», «<>», «<», «>», «<=», «>=»);

3) операторы присваивания («=», «+=», «-=», «*=», «/=», «%=», «**=», «//=»);

4) побитовые операторы («&», «|», «^», «~», «<<», «>>»);

5) операторы членства («in», «not in»);

6) логические операторы («and», «or», «not»);

7) операторы тождественности («is», «is not»).

2.2.3. Оператор ветвления и операторы цикла

Оператор ветвления «if.. ..else» позволяет в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнить его. Оператор имеет следующий формат:

if <Логическое выражение>:
<Блок, выполняемый, если условие истинно>
[elif <Логическое выражение>:
<Блок, выполняемый, если условие истинно>
]
[else:
<Блок, выполняемый, если все условия ложны>
]

Если блок состоит из одной инструкции, эту инструкцию можно разместить на одной строке с заголовком (следует иметь в виду, что подобная конструкция нарушает стройность кода и ухудшает его сопровождение в дальнейшем):

if <Логическое выражение>: <Блок, выполняемый, если условие истинно>

Оператор ветвления «if.. ..else» позволяет проверить сразу несколько условий с помощью инструкции «elif».

Пример использования оператора ветвления «if.. ..else» в различных форматах приведен на рис. 2.7.

```
В [13]: # Использование первого формата
        x = int (input ("Введите число: "))
        if x % 2 == 0:
            print (x, " - четное число (1 формат)")
        else:
            print (x, " - нечетное число (1 формат)")
        # Использование второго формата
        if x % 2 == 0: print (x, " - четное число (2 формат)")
        else: print (x, " - нечетное число (2 формат)")
        # Использование третьего формата
        print (x, " - четное число (3 формат)" if x % 2 == 0 \
               else " - нечетное число (31 формат)")

        Введите число: 4
        4  - четное число (1 формат)
        4  - четное число (2 формат)
        4  - четное число (3 формат)
```

```
В [17]: x = int (input ("Введите целое число от 1 до 3: "))
        if x == 1:
            print ("Вы ввели число 1")
        elif x == 2:
            print ("Вы ввели число 2")
        elif x == 3:
            print ("Вы ввели число 3")
        else:
            print ("Вы ввели ошибочное число")

        Введите целое число от 1 до 3: 1
        Вы ввели число 1
```

Рис. 2.7. Пример использования оператора ветвления «if.. ..else»

Операторы цикла позволяют выполнять блок кода заданное число раз, либо до тех пор, пока не выполнится заданное условие.

Оператор цикл «for» применяется для перебора элементов последовательности и имеет такой формат:

for <Текущий элемент> in <Последовательность>:
<Инструкции внутри цикла>
[else:
<Блок, выполняемый, если не использовался оператор break>
]

В качестве <Последовательность> указывается любой объект, поддерживающий механизм итерации: строка, список, кортеж, диапазон, словарь и др. <Текущий элемент> представляет собой перемен-

ную, через которую на каждой итерации цикла доступен очередной элемент последовательности или ключ словаря.

Если внутри цикла не использовался оператор «break», то после завершения выполнения цикла будет выполнен блок в инструкции «else».

Пример использования оператора цикла «for» для различных видов последовательностей приведен на рис. 2.8.

```
В [27]: # Диапазон
        for i in range(1, 4): print(i, end=",")
        # Список
        for i in (19, 3, 75): print(i, end=";")
        # Строка
        for i in "Привет":
            print(i, end=" ")
        else:
            print("\nЦикл выполнен")

        1,2,3,19;3;75;П р и в е т
        Цикл выполнен
```

Рис. 2.8. Пример использования оператора цикла «for

Выполнение инструкций в цикле оператора «while» продолжается до тех пор, пока логическое выражение истинно. Цикл «while» имеет следующий формат:

<Установка начального значения для переменной-счетчика>
while <Условие>:
<Инструкции>
<Приращение значения в переменной-счетчике>
[else:
<Блок, выполняемый, если не использовался оператор break>
]

Последовательность работы цикла «while»:

1) переменной-счетчику присваивается начальное значение;

2) проверяется условие, и если оно истинно, то выполняются инструкции внутри цикла, иначе выполнение цикла завершается;

3) переменная-счетчик изменяется на величину, указанную в параметре <приращение>;

4) переход к пункту 2;

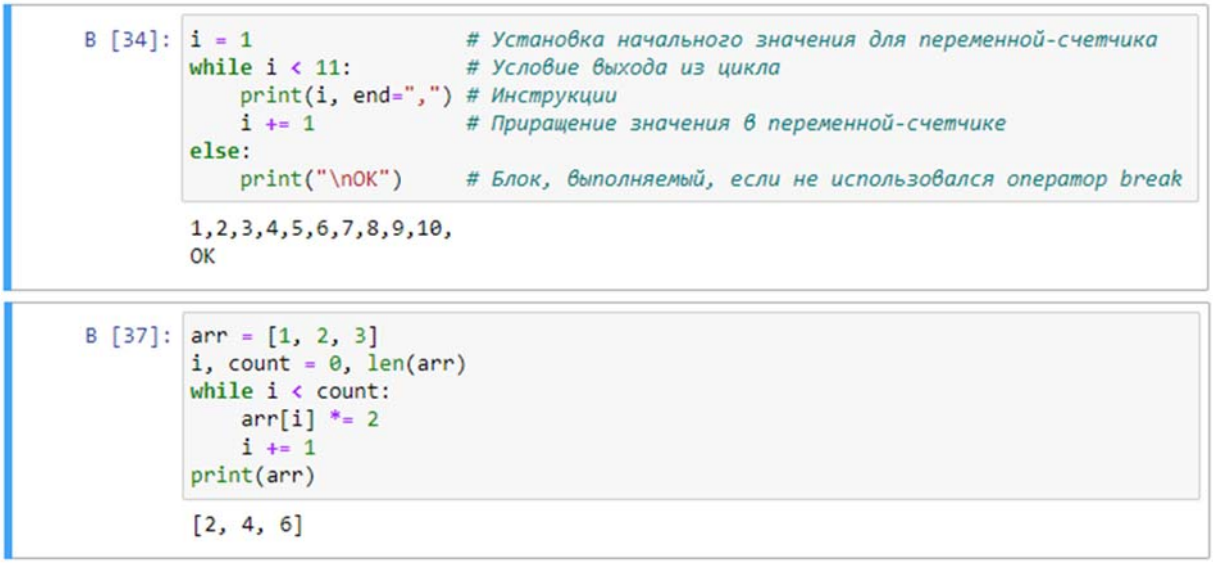
5) если внутри цикла не использовался оператор «break», то после завершения выполнения цикла будет выполнен блок в инструкции «else». Этот блок не является обязательным.

С помощью цикла «while» можно перебирать и элементы различных структур.

Оператор «continue» начинает следующий проход цикла, минуя оставшееся тело цикла («for» или «while»)

Оператор «break» досрочно прерывает цикл.

Пример использования оператора цикла «while» приведен на рис. 2.9.



```
В [34]: i = 1                  # Установка начального значения для переменной-счетчика
        while i < 11:          # Условие выхода из цикла
            print(i, end=",") # Инструкции
            i += 1             # Приращение значения в переменной-счетчике
        else:
            print("\nOK")      # Блок, выполняемый, если не использовался оператор break

        1,2,3,4,5,6,7,8,9,10,
        OK
```

```
В [37]: arr = [1, 2, 3]
        i, count = 0, len(arr)
        while i < count:
            arr[i] *= 2
            i += 1
        print(arr)

        [2, 4, 6]
```

Рис. 2.9. Пример использования оператора цикла «while»

2.2.4. Пользовательские функции

Функция – это фрагмент кода, который можно вызвать из любого места программы. Функция создается (или, как говорят программисты, определяется) с помощью ключевого слова def в следующем формате:

def <Имя функции> ([<Параметры>]):
<Тело функцию>
[return <Результат>]

Имя функции должно быть уникальным идентификатором, состоящим из латинских букв, цифр и знаков подчеркивания (имя функции не может начинаться с цифры). Регистр символов в имени функции также имеет значение.

После имени функции в круглых скобках через запятую можно указать один или несколько параметров, а если функция не принимает параметры, указываются только круглые скобки. После круглых скобок ставится двоеточие.

Тело функции представляет собой составную конструкцию. Если тело функции не содержит инструкций, то внутри нее необходимо разместить оператор «pass», который не выполняет никаких действий.

Необязательная инструкция «return» позволяет вернуть из функции какое-либо значение в качестве результата. После исполнения этой инструкции выполнение функции будет остановлено, и последующие инструкции никогда не будут выполнены.

Определение функции должно быть расположено перед вызовом функции. При вызове функции значения ее параметров указываются внутри круглых скобок через запятую. Если функция не принимает параметров, оставляются только круглые скобки (количество параметров в определении функции должно совпадать с количеством параметров при вызове):

Чтобы сделать некоторые параметры функции необязательными, следует в определении функции присвоить этому параметру начальное значение

Если перед параметром в определении функции указать символ «*», то функции можно будет передать произвольное количество параметров. Все переданные параметры сохраняются в кортеже.

Одну функцию можно вложить в другую функцию, причем уровень вложенности не ограничен. При этом вложенная функция получает свою собственную локальную область видимости и имеет доступ к переменным, объявленным внутри функции, в которую она вложена.

Пример использования функции приведен на рис. 2.10.

```
В [40]: def print_Yes():
            """Пример функции без параметров"""
            print("Да")
        def print_No(str):
            """Пример функции с параметром"""
            print(str)
        def summa (x, y):
            """Пример функции с параметрами, возвращающей результат"""
            return (x + y)

        # Основная программа
        s1=5
        s2=10
        print_Yes()
        print_No("Нет")
        print ('Сумма - ', summa(s1, s2))

        Да
        Нет
        Сумма -  15
```

```
В [41]: def summa (x, y=10):
            return (x + y)
        print ('Сумма - ', summa(5))
        print ('Сумма - ', summa(20, 30))

        Сумма -  15
        Сумма -  50
```

Рис. 2.10. Пример использования функции

2.2.5. Модули и пакеты

Модулем в языке Python называется любой файл с программным кодом. Каждый модуль может импортировать другой модуль, получая, таким образом, доступ к атрибутам (переменным, функциям и классам), объявленным внутри импортированного модуля. Следует заметить, что импортируемый модуль может содержать программу не только на Python – можно импортировать скомпилированный модуль, написанный на языке C.

Импортировать модуль позволяет инструкция «import», которая имеет следующий формат:

import <Название модуля 1> [as <Псевдоним 1 >] [, . . . , <Название модуля N> [as <Псевдоним N>]]

После ключевого слова import указывается название модуля. При именовании модулей необходимо учитывать, что операция им-

порта создает одноименный идентификатор, – это означает, что название модуля должно полностью соответствовать правилам именования переменных. За один раз можно импортировать сразу несколько модулей, записав их через запятую.

После импортирования модуля его название становится идентификатором, через который можно получить доступ к атрибутам, определенным внутри модуля. Доступ к атрибутам модуля осуществляется с помощью точечной нотации.

Если название модуля слишком длинное и его неудобно указывать каждый раз для доступа к атрибутам модуля, то можно создать псевдоним. Псевдоним задается после ключевого слова «as» и в дальнейшем обращение к атрибутам модуля осуществляется через указание псевдонима.

Для импортирования только определенных идентификаторов из модуля можно воспользоваться инструкцией «from».

Пример работы с модулем приведен на рис. 2.11.

Рис. 2.11. Пример работы с модулем

Пакетом называется каталог с модулями. Пакеты позволяют распределить модули по каталогам. При использовании инструкции «import» путь к модулю должен включать не только имена каталогов, но и название модуля без расширения.

Вопросы для обсуждения

1. Генезис развития теории и инструментария машинного обучения

2. Правовые проблемы применения инструментов машинного обучения

3. Преимущества и недостатки наиболее востребованных инструментов машинного обучения

4. Mahout: функциональные возможности, преимущества и недостатки

5. MLib: функциональные возможности, преимущества и недостатки

6. SAMOA: функциональные возможности, преимущества и недостатки

7. MADLib: функциональные возможности, преимущества и недостатки

8. Библиотеки языка R: функциональные возможности, преимущества и недостатки

9. Библиотека ML.NET: функциональные возможности, преимущества и недостатки

10. Библиотеки языка Python: функциональные возможности, преимущества и недостатки

Практические задания

Задание 2.1.

Вычислить значения на интервале [0, 1] с шагом 0.01 с точностью 0.01 для следующих функций и вывести на экран только положительные результаты (по вариантам).

Варианты функций:

1) $F(x) = x - \sin x$;
2) $F(x) = x \cdot \cos x - 1$;
3) $F(x) = x \cdot \sin x - 1$;
4) $F(x) = 2 \cdot \sin^2 x + 1$;
5) $F(x) = -\cos x + 1$;
6) $F(x) = 2 \cdot \cos 2x$;
7) $F(x) = (x - \sin x)^2$;
8) $F(x) = \sqrt{x} \cdot \cos^2 x$;
9) $F(x) = \sin^2 x - \cos 2x$;
10) $F(x) = \sin x + 0.5 \cdot \cos x$

Библиографический список

1. Python // Википедия: [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Python

2. Основы Data Science и Big Data. Python и наука о данных: [текст] / Дэви Силен, Арно Мейсман, Мохамед Али; [пер. с англ. Е. Матвеев]. - Санкт-Петербург [и др.] : Питер, 2017. - 334 с.: ил. - (Библиотека программиста). - Пер. изд. : Introducing data science. Big data, machine learning, and more, using python tools / Davy Cielen, Arno D. B. Meysman, Mohamed Ali. - Greenwich, 2016. - 12+. - ISBN 978-5-496-02517-1

3. Плас, Дж. Вандер. Python для сложных задач наука о данных: и машинное обучение : монография / Дж. В. Плас ; [пер. с англ. И. Пальти]. - Санкт-Петербург [и др.] : Питер, 2018. - 572 с. : рис., граф. - Пер. изд.: Python Data Science Hazndbook. Essential Tools for Working with Data / J. V. Plas. - Beijing [et al.], 2017. - ISBN 978-5-496-03068-7

4. Мюллер, Андреас. Введение в машинное обучение с помощью Python: руководство для специалистов по работе с данными / А. Мюллер, С. Гвидо. - Москва [и др.]: Диалектика, 2017. - 472 с. : цв. ил. - Предм. указ.: с. 465-472. - Пер. изд. : Introduction to Machine Leaning with Python / A. C. Muller, S. Guido. - Beijing [et al.], 2017. - ISBN 978-5-9908910-8-1

5. Маккинни, У. Маккинли, У. Python и анализ данных / Уэс Маккинли ; пер. с англ. А.А. Слинкина. - Москва : ДМК Пресс, 2015. - 482 с. - ISBN 978-5-97060-315-4. - Текст : электронный. - URL: https://znanium.com/catalog/product/1027796

6. Гуриков, С. Р. Основы алгоритмизации и программирования на Python: учебное пособие / С.Р. Гуриков. – Москва : ФОРУМ : ИНФРА-М, 2020. – 343 с. – (Высшее образование: Бакалавриат). - ISBN 978-5-00091-487-8. - Текст: электронный. - URL: https://znanium.com/catalog/product/1206074

Глава 3. NUMPY: ОБРАБОТКА И АНАЛИЗ НАБОРОВ ОДНОРОДНЫХ ДАННЫХ

В данной главе рассматриваются следующие вопросы:

1. Функциональные возможности python-библиотеки NumPy

2. Объект «ndarray» как контейнер для хранения больших наборов данных

3. Обзор основных операций с наборами однородных данных

3.1. Функциональные возможности NumPy

Numerical Python, или сокращенно NumPy [1] – краеугольный пакет для высокопроизводительных научных расчетов и анализа данных, который лежит в основе большинства библиотек представления данных и машинного обучения.

В большинстве приложений для анализа данных основной интерес представляет следующая функциональность:

1) быстрые векторные операции для переформатирования и очистки данных, выборки подмножеств и фильтрации, преобразований и других видов вычислений;

2) стандартные алгоритмы работы с массивами, например: фильтрация, удаление дубликатов и теоретико-множественные операции;

3) эффективная описательная статистика, агрегирование и обобщение данных;

4) выравнивание данных и реляционные операции объединения и соединения разнородных наборов данных;

5) описание условной логики в виде выражений-массивов вместо циклов с ветвлением if-elif-else;

6) групповые операции с данными (агрегирование, преобразование, применение функции).

Пакет NumPy принято импортировать, используя в качестве его псевдонима «np»

3.2. Объект «ndarray» как контейнер для хранения больших наборов данных

Одна из ключевых особенностей NumPy – **объект «ndarray»** для представления N-мерного массива; это быстрый и гибкий контейнер для хранения больших наборов данных в Python.

Объект «ndarray» – это обобщенный многомерный контейнер для однородных данных, т. е. в нем могут храниться только элементы одного типа. У любого массива есть атрибут «shape» – кортеж, описывающий размер по каждому измерению, и атрибут «dtype» – объект, описывающий тип данных в массиве.

Как и для одномерных объектов (наподобие списков Python), для объектов «ndarray» можно формировать срезы.

3.2.1. Создание массива

Существует огромное количество вариантов создания массива: создание массива на основе существующих данных (array, fromfile, fromstring, loadtxt и др.), генерация массива с заполнением его нулями и/или единицами (empty, eye, ones, zeros и др.), генерация массива без инициализации или с заполнением его значениями с заданной функцией распределения (random.random, random.normal, random.randint и др.), генерация массива с заполнением его заданными интервальными значениями (arange, linspace, logspace, geomspace и др.)[2].

Наиболее распространённым способом создания массива на основе существующих данных является использование функции «numpy.array», описание которой выглядит следующим образом:

numpy.array (object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0, like=None)

Основные аргументы функции:

object (тип данных – array_like) – массив, любой объект, предоставляющий интерфейс массива или любая (вложенная) последовательность (списки, кортежи, массивы и т.д.);

dtype (тип данных – data-type, необязательный аргумент) – тип данных создаваемого массива [3] (если атрибут не определен явно, то

функция пытается самостоятельно определить подходящий тип данных для создаваемого массива).

Функция «numpy.array» возвращает объект массива.

Пример создания массива на основе существующих данных с использованием функции «numpy.array» приведен на рис. 3.1:

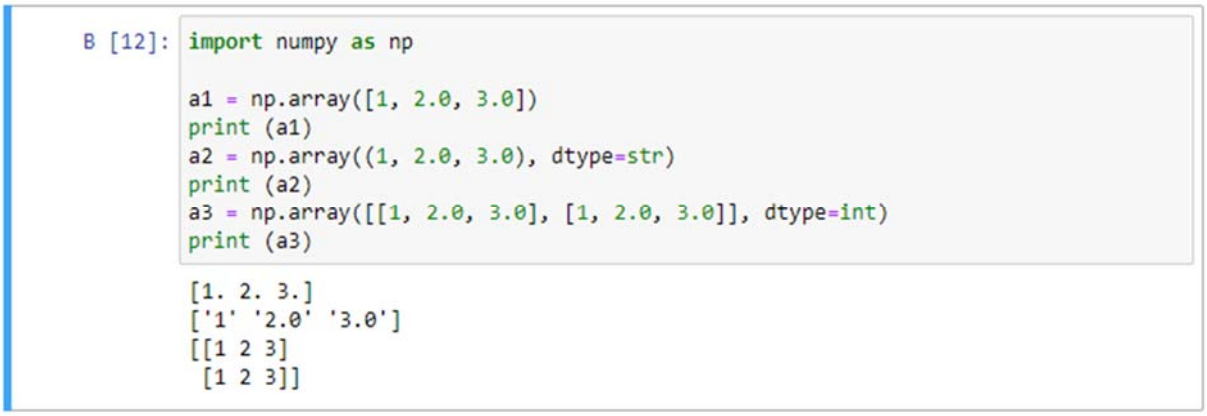


Рис. 3.1. Создание массива на основе существующих данных

Создание массива из данных, содержащихся в текстовом или двоичном файле можно осуществить с помощью функции «numpy.fromfile», описание которой выглядит следующим образом:

numpy.fromfile (file, dtype=float, count=-1, sep='', offset=0, *, like=None)

Основные аргументы функции:

file (тип данных – file или str) – файловый объект или путь к файлу;

dtype (тип данных – data-type, необязательный аргумент) – тип данных создаваемого массива (значение по умолчанию – float);

count (тип данных – int, необязательный аргумент) – количество элемента для чтения (значение по умолчанию – -1 соответствует чтению всех элементов в файле);

sep (тип данных – str, необязательный аргумент) – разделитель между элементами в текстовом файле (значение по умолчанию – «» соответствует двоичному файлу).

Для записи массива в файл используется функция «numpy.tofile», описание которой выглядит следующим образом:

ndarray.tofile (fid, sep="", format="%s")

Основные аргументы функции:

fid (тип данных – file или str) – файловый объект или путь к файлу;

sep (тип данных – str, необязательный аргумент) – разделитель между элементами при записи в текстовой файл (значение по умолчанию – «» соответствует записи в двоичный файл).

Пример чтения и записи массива из файла с использованием функций «numpy.fromfile» и «numpy.tofile» приведен на рис. 3.2.

```
В [26]: import numpy as np

        # Формируем исходный массив
        arr = np.array([[1, 2.0, 3.0], [11, 12, 14]], dtype=int)
        print ("Исходный массив:\n ", arr)

        # Сохраняем исходный массив на диск
        arr.tofile ("Z:/1.txt", sep=";")

        # Загружаем массив с диска в объект
        new_arr = np.fromfile("Z:/1.txt", sep=";", dtype = np.int)
        print ("Загруженный массив:\n ", new_arr)

        Исходный массив:
          [[ 1  2  3]
         [11 12 14]]
        Загруженный массив:
          [ 1  2  3 11 12 14]
```

Рис. 3.2. Создание массива из файла

Данный способ хранения и загрузки данных имеет существенный недостаток – при записи массива, теряется информация о его форме. Поэтому применение функции «numpy.fromfile» целесообразно только для первичной загрузки данных с последующей обработкой полученного одномерного массива (при необходимости воссоздания его формы).

Рекомендуемый способ хранения и загрузки данных предполагает использование функций «numpy.save» и «numpy.load». Программная реализация данного метода приведена на рис. 3.3.

```
В [28]: import numpy as np
        fname = "Z:/1.npy"

        # Формируем исходный массив
        arr = np.array([[1, 2.0, 3.0], [11, 12, 14]], dtype=int)
        print ("Исходный массив:\n ", arr)

        # Сохраняем исходный массив на диск
        np.save(fname, arr)

        # Загружаем массив с диска в объект
        new_arr = np.load(fname)
        print ("Загруженный массив:\n ", new_arr)

        Исходный массив:
          [[ 1  2  3]
         [11 12 14]]
        Загруженный массив:
          [[ 1  2  3]
         [11 12 14]]
```

Рис. 3.3. Рекомендуемый способ хранения и загрузки данных

Для загрузки данных из текстового файла может использоваться также функция «numpy.loadtxt». Для работы этой функции необходимо, что бы каждая строка в текстовом файле имело одинаковое количество значений.

Пример генерации массива либо без инициализации, либо с заполнением его нулями и/или единицами с использованием функций «numpy.empty», «numpy.eye», «numpy.ones», «numpy.zeros» приведен на рис. 3.4.

Наиболее распространённым способом создания массива с заполнением его заданными интервальными значениями является использование функции «numpy.arange» (функция возвращает равномерно распределенные значения в заданном интервале), описание которой выглядит следующим образом:

numpy.arange ([start,]stop, [step,]dtype=None, *, like=None)

Основные аргументы функции:

start (тип данных – integer или real, необязательный аргумент) – начало интервала включительно (значение по умолчанию – 0);

stop (тип данных – integer или real) – окончание интервала (значение не включается в результирующий массив);

step (тип данных – integer или real, необязательный аргумент) – расстояние между значениями (значение по умолчанию – 1);

dtype (тип данных – data-type, необязательный аргумент) – тип данных создаваемого массива.

```
В [61]: import numpy as np

arr1 = np.empty([2, 2])
print ("Массив без инициализации записей:\n ", arr1)

arr2 = np.ones([3, 3], dtype=float)
print ("Массив, заполненный единицами:\n ", arr2)

arr3 = np.zeros([3, 3])
print ("Массив, заполненный нулями:\n ", arr3)

arr4 = np.eye(3, dtype=int)
print ("Двумерный массив с единицами по диагонали "\
       "и нулями на других позициях:\n", arr3)
```

```
Массив без инициализации записей:
  [[3.23786e-319 4.94066e-324]
 [0.00000e+000 0.00000e+000]]
Массив, заполненный единицами:
  [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Массив, заполненный нулями:
  [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Двумерный массив с единицами по диагонали и нулями на других позициях:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Рис. 3.4. Генерации массива без инициализации
или с заполнением нулями и единицами

Создание массива с заполнением его заданными интервальными значениями осуществляется с использованием функции «numpy.arange».

Создание массива из заданного количества значений, равномерно располагающихся между начальным и конечным значением интервала, осуществляется с использованием функции «numpy.-linspace».

Примеры использования функции «numpy.arange» и numpy.linspace» приведены на рис. 3.5.

```
В [75]: import numpy as np

arr1 = np.arange(3)
print ("Указан только параметр stop:\n", arr1)
arr2 = np.arange(3,7)
print ("Указаны параметры: start, stop:\n", arr2)
arr3 = np.arange(3.2,6.7,0.5)
print ("Указаны параметры: start, stop, step:\n", arr3)

Указан только параметр stop:
 [0 1 2]
Указаны параметры: start, stop:
 [3 4 5 6]
Указаны параметры: start, stop, step:
 [3.2 3.7 4.2 4.7 5.2 5.7 6.2]
```

```
В [76]: import numpy as np

arr1 = np.linspace(2.1, 3.1, num=5)
print ("Указаны параметры: start, stop, num:\n", arr1)

Указаны параметры: start, stop, num:
 [2.1  2.35 2.6  2.85 3.1 ]
```

Рис. 3.5.Создание массива с заполнением его заданными интервальными значениями

Для создания массивов со случайными элементами служит модуль «numpy.random».

Для создания массива со случайными равномерно распределёнными элементами на интервале [0, 1) используются следующие функции:

1) равномерно распределёнными элементами – функция «numpy.random. random_sample» (для целых чисел – «numpy.random.randint»),

2) нормально распределёнными элементами – функция «numpy.random.normal»,

3) экспоненциально распределёнными элементами – функция «numpy.random.exponential»,

4) логистически распределёнными элементами – функция «numpy.random.exponential»

5) и многие другие [4].

Пример создания массивов со случайными элементами приведен на рис. 3.6.

```
В [1]: import numpy as np

print ("random.random_sample():\n", np.random.random_sample())
print ("random.random_sample((3, 3)):\n", np.random.random_sample((3, 3)))
print ("np.random.normal(0, 1, (3, 3)):\n", np.random.normal(0, 1, (3, 3)))

random.random_sample():
 0.11437691825221452
random.random_sample((3, 3)):
 [[0.53648031 0.89214924 0.38461169]
 [0.70480575 0.28028272 0.5156373 ]
 [0.24667236 0.51562317 0.08710617]]
np.random.normal(0, 1, (3, 3)):
 [[ 1.31846695 -1.04546127 -1.1805438 ]
 [-0.07859687  0.22836167  1.06404399]
 [-0.28486387  0.78805151 -0.91966274]]
```

Рис. 3.6. Создание массива с заполнением его случайными элементами

3.2.2. Доступ к отдельным элементам и выборка подмножеств

Доступ к отдельным элементам объекта ndarray осуществляется аналогично доступу к элементам стандартных списков языка Python: в одномерном массиве обратиться к i-му (считая с 0) значению можно, указав требуемый индекс в квадратных скобках, а к элементам в многомерном массиве – с помощью разделенных запятыми кортежей индексов.

Синтаксис срезов библиотеки NumPy также соответствует аналогичному синтаксису для стандартных списков языка Python:

x [начало : конец : шаг]

Значения по умолчанию: начало = 0, конец = размер соответствующего измерения массива, шаг = 1).

Многомерные срезы задаются схожим образом, с разделением срезов запятыми.

Предоставление доступа к отдельным строкам или столбцам объекта ndarray осуществляются путем комбинации индексации и среза, с помощью пустого среза, задаваемого двоеточием (:), который при желании можно игнорировать.

Пример доступа к отдельным элементам объекта «ndarray», а также работы со срезами, столбцами и строками приведен на рис. 3.7.

```
В [43]: import numpy as np
        arr=np.around(100*np.random.random_sample((3, 3)))
        print ("Исходный массив:\n", arr)
        print ("Элемент массива (0, 0):\n", arr[0,0])
        print ("Элемент массива (1, 2):\n", arr[1,2])
        print ("Строки, начиная со строки 1:\n", arr[1:,:])
        print ("Строки, со строки 0 до строки 2:\n", arr[0:2,:])
        print ("Столбец, крайний справа:\n", arr[:,-1])
        print ("Все строки, каждый второй столбец 2:\n", arr[:3, ::2])

Исходный массив:
 [[18. 88. 69.]
 [44. 44. 54.]
 [18. 92. 30.]]
Элемент массива (0, 0):
 18.0
Элемент массива (1, 2):
 54.0
Строки, начиная со строки 1:
 [[44. 44. 54.]
 [18. 92. 30.]]
Строки, со строки 0 до строки 2:
 [[18. 88. 69.]
 [44. 44. 54.]]
Столбец, крайний справа:
 [69. 54. 30.]
Все строки, каждый второй столбец 2:
 [[18. 69.]
 [44. 54.]
 [18. 30.]]
```

Рис. 3.7. Доступ к отдельным элементам массива и выборка подмножеств

3.3. Обзор операций с наборами однородных данных

3.3.1. Векторные операции

Вектор является одним из основополагающих понятий линейной алгебры. При использовании наиболее общего определения векторами оказываются практически все изучаемые в линейной алгебре объекты, в том числе матрицы и тензоры.

Объект «ndarray» рассматривается как вектор и к нему применимы все применяемые к векторам поэлементные операции: операции со скалярными величинами, сложение, вычитание, умножение, деление и другие.

Примеры векторных вычислений с элементами объектов «ndarray» приведены на рис. 3.8.

```
В [17]: import numpy as np
arr1 = np.array([[5, 4, 3], [9, 7, 5]], dtype=int)
arr2 = np.array([[1, 2, 3], [1, 2, 3]], dtype=int)
print ("Исходный массив №1:\n ", arr1, "\nИсходный массив №2:\n ", arr2)

print ("Операции со скалярными величинами:\n", arr1*10 + 1000)
print ("Сложение:\n", arr1+arr2)
print ("Вычитание:\n", arr1-arr2)
print ("Произведение:\n", arr1*arr2)

Исходный массив №1:
  [[5 4 3]
 [9 7 5]]
Исходный массив №2:
  [[1 2 3]
 [1 2 3]]
Операции со скалярными величинами:
 [[1050 1040 1030]
 [1090 1070 1050]]
Сложение:
 [[ 6  6  6]
 [10  9  8]]
Вычитание:
 [[4 2 0]
 [8 5 2]]
Произведение:
 [[ 5  8  9]
 [ 9 14 15]]
```

Рис. 3.8. Векторные вычисления с элементами массивов

Функции, которые выполняют поэлементные операции над данными, хранящимися в объектах «ndarray», называются универсальными или u-функциями.

Различают 2 вида u-функций:

1) унарные (в качестве аргумента принимают один массив)

2) бинарные (в качестве аргумента принимают два массива).

Краткий обзор основных унарных и бинарных u-функций приведен в табл. 3.1.

Таблица 3.1

Основные u-функции

Функция	Описание
Унарные u-функции	
abs	Позволяет вычислить абсолютное значение элементов массива.
ceil	Позволяет вычислить для каждого элемента наименьшее целое число, не меньшее его
floor	Позволяет вычислить для каждого элемента наибольшее целое число, не большее его
exp	Позволяет вычислить экспоненту e^x каждого элемента
sqrt	Позволяет вычислить квадратный корень из каждого элемента

Окончание табл. 3.1

Функция	Описание
square	Позволяет вычислить квадрат каждого элемента
cos, cosh, tan, tanh, sin, sinh, arcsin, arctanh	Позволяет вычислить обычные и гиперболические тригонометрические функции
arccos, arccosh, arcsinh, arctan	Позволяет вычислить обратные тригонометрические функции
log, log10, log2	Позволяет вычислить натуральный, десятичный и двоичный логарифм
sign	Позволяет вычислить знак каждого элемента: 1 (для положительных чисел), 0 (для нуля) или -1 (для отрицательных чисел)
rint	Позволяет округлить элементы до ближайшего целого с сохранением dtype
Бинарные u-функции	
add	Позволяет сложить соответственные элементы массивов
subtract	Позволяет вычесть элементы второго массива из соответственных элементов первого
multiply	Позволяет перемножить элементы массивов
divide, floor_divide	Позволяет произвести деление и деление с отбрасыванием остатка
power	Позволяет возвести элементы первого массива в степени, указанные во втором массиве
maximum	Позволяет вычислить поэлементный максимум.
minimum	Позволяет вычислить поэлементный минимум.
greater, equal, greater_ less, less_equal, not_equal	Позволяет произвести поэлементное сравнение
logical_and, logical_or, logical_xor	Позволяет вычислить логическое значение истинности логических операций.

Примеры использования унарных и бинарных u-функций приведены на рис. 3.9.

3.3.2. Преобразование массивов

К операциям преобразования массивов относят: копирование массива, изменение его формы, слияние массивов и разбиение массива.

Для копирования объекта «ndarray» или его части используется функция «copy». Пример применения указанной функции приведен на рис. 3.10.

```
В [10]: import numpy as np
arr1 = np.array([[5, 0, 3], [-9, 7, -5]], dtype=int)
arr2 = np.array([[1, 2, 3], [1, 0, 3]], dtype=int)
print ("Исходный массив №1:\n ", arr1, "\nИсходный массив №2:\n ", arr2)
print ("Абсолютное значение элементов массива №1:\n", np.sign(arr1))
print ("Сложение элементов массива №1 и №2:\n", np.add(arr1, arr2))
print ("Поэлементный максимум №1 и №2:\n", np.maximum(arr1, arr2))

Исходный массив №1:
  [[ 5  0  3]
 [-9  7 -5]]
Исходный массив №2:
  [[1 2 3]
 [1 0 3]]
Абсолютное значение элементов массива №1:
 [[ 1  0  1]
 [-1  1 -1]]
Сложение элементов массива №1 и №2:
 [[ 6  2  6]
 [-8  7 -2]]
Поэлементный максимум №1 и №2:
 [[5 2 3]
 [1 7 3]]
```

Рис. 3.9. Использование унарных и бинарных u-функций

```
В [50]: import numpy as np
arr=np.around(100*np.random.random_sample((3, 3)))
print ("Исходный массив:\n", arr)
arr2=arr.copy()
print ("Скопированный массив:\n", arr2)
arr3=arr[:2,:2].copy()
print ("Скопированная часть массива:\n", arr3)

Исходный массив:
 [[25. 67. 91.]
 [63. 30. 25.]
 [97.  1. 77.]]
Скопированный массив:
 [[25. 67. 91.]
 [63. 30. 25.]
 [97.  1. 77.]]
Скопированная часть массива:
 [[25. 67.]
 [63. 30.]]
```

Рис. 3.10. Копирование массива и его части

Изменение формы массива без изменения его данных осуществляется функцией «reshape». Пример применения которой приведен на рис.13.11.

Слияние, или объединение, двух массивов в библиотеке NumPy может быть выполнено с помощью функций «concatenate», «vstack» и «hstack».

```
В [74]: import numpy as np
arr=np.around(100*np.random.random_sample((8)))
print ("Исходный массив:\n", arr)
arr2=arr.reshape((4, 2))
print ("Массив с измененной формой:\n", arr2)

Исходный массив:
 [69. 79. 10. 29.  6. 90. 24. 42.]
Массив с измененной формой:
 [[69. 79.]
 [10. 29.]
 [ 6. 90.]
 [24. 42.]]
```

Рис. 3.11. Изменение формы массива без изменения его данных

Для слияния массивов вдоль одной из его осей используется функция «concatenate»:

numpy.concatenate ((a1, a2, ...), axis=0, dtype=None

Основные аргументы функции:

(a1, a2, ...) – последовательность подобных массиву объектов;

axis (тип данных – integer) – ось, вдоль которой соединяются массивы (по умолчанию axis=0, что соответствует первой оси);

Пример слияния массивов приведен на рис. 3.12.

```
В [82]: import numpy as np
arr1=np.around(100*np.random.random_sample((2,2)))
print ("Исходный массив №1:\n", arr1)
arr2=np.around(100*np.random.random_sample((2,2)))
print ("Исходный массив №2:\n", arr2)
arr3=np.concatenate ((arr1, arr2))
print ("Объединенный по первой оси массив:\n", arr3)
arr4=np.concatenate ((arr1, arr2),axis=1)
print ("Объединенный по второй оси массив №2:\n", arr4)

Исходный массив №1:
 [[ 3. 47.]
 [19. 92.]]
Исходный массив №2:
 [[11.  5.]
 [99. 97.]]
Объединенный по первой оси массив:
 [[ 3. 47.]
 [19. 92.]
 [11.  5.]
 [99. 97.]]
Объединенный по второй оси массив №2:
 [[ 3. 47. 11.  5.]
 [19. 92. 99. 97.]]
```

Рис. 3.12. Слияние массивов вдоль одной из его осей

Для работы с массивами с различающимися измерениями используют функции «vstack» (вертикальное объединение, т.е. присоединять данные по вертикальной оси) и «hstack» (горизонтальное объединение, т.е. присоединять данные по горизонтальной оси).

Примеры вертикального и горизонтального слияния массивов приведены на рис. 3.13.

```
В [88]: import numpy as np
        arr1=np.around(100*np.random.random_sample((2,2)))
        print ("Исходный массив №1:\n", arr1)
        arr2=np.around(100*np.random.random_sample((2,2)))
        print ("Исходный массив №2:\n", arr2)
        arr3=np.vstack ([arr1, arr2])
        print ("Вертикальное объединение:\n", arr3)
        arr4=np.hstack ([arr1, arr2])
        print ("Горизонтальное объединение:\n", arr4)

        Исходный массив №1:
         [[52. 70.]
         [82. 15.]]
        Исходный массив №2:
         [[32.  5.]
         [ 4. 56.]]
        Вертикальное объединение:
         [[52. 70.]
         [82. 15.]
         [32.  5.]
         [ 4. 56.]]
        Горизонтальное объединение:
         [[52. 70. 32.  5.]
         [82. 15.  4. 56.]]
```

Рис. 3.13. Разбиение массива с помощью функций «hstack» (горизонтальное разбиение) и «vstack» (вертикальное разбиение)

Разбиение массива в библиотеке NumPy может быть выполнено с помощью функций «split», «hsplit» и «vsplit»:

Для разделения массива по одной из его осей используется функция «split», описание которой выглядит следующим образом:

numpy.split (ary, indices_or_sections, axis=0)

Основные аргументы функции:

ary (тип данных – ndarray) – подлежащий разделению массив;

indices_or_sections (тип данных – integer или одномерный массив) – если index_or_sections является целым числом N, томассив будет разделен на N равных массивов по оси, если

index_or_sections представляет собой одномерный массив отсортированных целых чисел, записи указывают, где именно вдоль оси массив разбивается

axis (тип данных – integer) – ось, по которой следует разделить массив (по умолчанию axis=0, что соответствует первой оси).

Пример слияния массивов с помощью функции «split» приведен на рис. 3.14.

```
В [109]: import numpy as np
arr1=np.around(100*np.random.random_sample((6)))
print ("Исходный массив:\n", arr1)
arr2,arr3,arr4=np.split (arr1, 3)
print ("Массив №1:\n", arr2)
print ("Массив №2:\n", arr3)
print ("Массив №3:\n", arr4)
arr5,arr6,arr7=np.split (arr1, (1, 3))
print ("Массив №4:\n", arr5)
print ("Массив №5:\n", arr6)
print ("Массив №6:\n", arr7)

Исходный массив:
 [73. 75.  1. 50. 34. 70.]
Массив №1:
 [73. 75.]
Массив №2:
 [ 1. 50.]
Массив №3:
 [34. 70.]
Массив №4:
 [73.]
Массив №5:
 [75.  1.]
Массив №6:
 [50. 34. 70.]
```

Рис. 3.14. Слияние массивов с помощью функции «split»

Разбиение массива может быть также выполнено с помощью функций «hsplit» (горизонтальное разбиение) и «vsplit» (вертикальное разбиение). Пример разбиения массива с помощью указанных функций приведен на рис. 3.15.

3.3.3. Операции линейной алгебры

К операциям линейной алгебры относят умножение и разложение матриц, вычисление определителей и некоторые другие операции.

```
В [122]: import numpy as np
arr1=np.around(100*np.random.random_sample((3,3)))
print ("Исходный массив:\n", arr1)
arr2,arr3 =np.hsplit (arr1, [2])
print ("Массив №1:\n", arr2)
print ("Массив №2:\n", arr3)
arr4,arr5 =np.vsplit (arr1, [2])
print ("Массив №1:\n", arr4)
print ("Массив №2:\n", arr5)

Исходный массив:
 [[28. 97. 83.]
 [31. 60. 24.]
 [ 1. 79. 85.]]
Массив №1:
 [[28. 97.]
 [31. 60.]
 [ 1. 79.]]
Массив №2:
 [[83.]
 [24.]
 [85.]]
Массив №1:
 [[28. 97. 83.]
 [31. 60. 24.]]
Массив №2:
 [[ 1. 79. 85.]]
```

Рис. 3.15. Разбиение массива с помощью функций «hsplit» (горизонтальное разбиение) и «vsplit» (вертикальное разбиение)

Для перемножения матриц используется функция «dot» (в виде функции в пространстве имен numpy).

В модуле «numpy.linalg» имеет стандартный набор алгоритмов, в частности, умножение и разложение матриц, нахождение обратной матрицы и вычисление определителя.

Список основных функций линейной алгебры, включенных в модуль «numpy.linalg» приведен в табл. 3.2.

Таблица 3.2

Основные функции линейной алгебры

Функция	Описание
diag	Возвращает диагональные элементы квадратной матрицы в виде одномерного массива
dot	Вычисляет произведение матриц
trace	Вычисляет сумму диагональных элементов
det	Вычисляет определитель матрицы
eig	Вычисляет собственные значения и собственные векторы квадратной матрицы
inv	Вычисляет обратную матрицу

Пример использования функций линейной алгебры из модуля «numpy.linalg» приведен на рис. 3.16.

```
B [20]: import numpy as np
        arr1=np.around(10*np.random.random_sample((3,2)))
        print ("Исходная матрица №1:\n", arr1)
        arr2=np.around(100*np.random.random_sample((2,2)))
        print ("Исходная матрица №2:\n", arr2)
        print ("Умножение матриц:\n", np.dot (arr1, arr2))
        print ("Сингулярно разложение исходной матрицы №1:\n", np.linalg.svd (arr1))
        print ("Обратная матрица для исходной матрицы №2:\n", np.linalg.inv (arr2))

        Исходная матрица №1:
         [[2. 0.]
         [2. 6.]
         [7. 9.]]
        Исходная матрица №2:
         [[83. 51.]
         [36. 76.]]
        Умножение матриц:
         [[ 166.  102.]
         [ 382.  558.]
         [ 905. 1041.]]
        Сингулярно разложение исходной матрицы №1:
         (array([[-0.08656425,  0.66391452, -0.74278135],
                [-0.47013902, -0.68456152, -0.55708601],
                [-0.87833703,  0.30098676,  0.37139068]]), array([12.95289435,  2.494499
        53]), array([[-0.56062881, -0.82806723],
                [ 0.82806723, -0.56062881]]))
        Обратная матрица для исходной матрицы №2:
         [[ 0.01699463 -0.01140429]
         [-0.00805009  0.01855993]]
```

Рис. 3.16. Пример использования функций линейной алгебры

3.3.4. Сортировка элементов в массивах

Для сортировки массивов используется функция «sort», для определения индексов отсортированных элементов – функция «argsort».

По умолчанию в данных функциях используется алгоритм быстрой сортировки, однако доступны для использования также алгоритмы сортировки слиянием и пирамидальной сортировки.

Примеры сортировки элементов массива по различным осям, а также определение индексов отсортированных элементов приведены на рис. 3.17.

При необходимости использования сортировки по нескольким осям для указания порядка сортировки используется параметр «order».

```
В [34]: import numpy as np
        arr=np.around(10*np.random.random_sample((4,4)))
        print ("Исходный массив:\n", arr)
        print ("Отсортированный массив по оси 1:\n", np.sort(arr,0))
        print ("Отсортированный массив по оси 2:\n", np.sort(arr,1))
        print ("Индексы отсортированного по оси 2 массива:\n", np.argsort(arr,1))

        Исходный массив:
         [[4. 4. 3. 3.]
         [7. 7. 6. 4.]
         [6. 3. 9. 7.]
         [1. 2. 9. 1.]]
        Отсортированный массив по оси 1:
         [[1. 2. 3. 1.]
         [4. 3. 6. 3.]
         [6. 4. 9. 4.]
         [7. 7. 9. 7.]]
        Отсортированный массив по оси 2:
         [[3. 3. 4. 4.]
         [4. 6. 7. 7.]
         [3. 6. 7. 9.]
         [1. 1. 2. 9.]]
        Индексы отсортированного по оси 2 массива:
         [[2 3 0 1]
         [3 2 0 1]
         [1 0 3 2]
         [0 3 1 2]]
```

Рис. 3.17. Пример сортировки массивов

3.3.5. Теоретико-множественные операции

Операцией над множеством или над совокупностью множеств, расположенных в некотором пространстве, называется всякая функция, ставящая в соответствие этому множеству или этой совокупности множеств (независимых переменных) некоторое множество (результат операции), лежащее в том же пространстве [5].

В состав теоретико-множественных операций входят:

1) объединение множеств элементов массива;

2) пересечение множеств элементов массива;

3) разность множеств элементов массива;

4) декартово произведение множеств элементов массива.

В библиотеку NumPy включен ряд функций, позволяющих осуществлять с элементами массива теоретико-множественные операции. Список указанных функций приведен в табл. 3.3.

Функция «np.unique» позволяет устранить дубликаты значений в массиве, что является полезным при подготовке данных к дальнейшему анализу.

Таблица 3.3

Основные теоретико-множественные функции

Функция	Описание
unique (x)	Вычисляет отсортированное множество уникальных элементов
intersectld (x, y)	Вычисляет отсортированное множество элементов, общих для x и y
unionld (x, y)	Вычисляет отсортированное объединение элементов
inld (x, y)	Вычисляет булев массив, показывающий, какие элементы x встречаются в y
setdiffld (x, y)	Вычисляет разность множеств, т. е. элементы, принадлежащие x, но не принадлежащие y
setxorld (x, y)	Симметрическая разность множеств; элементы, принадлежащие одному массиву, но не обоим сразу

Пример осуществления теоретико-множественных операций с элементами массивов приведен на рис. 3.18.

```
В [50]: import numpy as np
        arr1=np.around(10*np.random.random_sample((3,3)))
        print ("Исходный массив №1:\n", arr1)
        arr2=np.around(10*np.random.random_sample((3,3)))
        print ("Исходный массив №2:\n", arr2)
        print ("Отсортированный массив уникальных значений:\n", np.unique(arr1))
        print ("Отсортированный массив общих элементов для массивов №1 и №2:\n",\
               np.intersect1d(arr1, arr2))

        Исходный массив №1:
         [[7. 5. 6.]
         [5. 9. 8.]
         [2. 2. 5.]]
        Исходный массив №2:
         [[3. 4. 2.]
         [7. 4. 6.]
         [4. 2. 8.]]
        Отсортированный массив уникальных значений:
         [2. 5. 6. 7. 8. 9.]
        Отсортированный массив общих элементов для массивов №1 и №2:
         [2. 6. 7. 8.]
```

Рис. 3.18. Теоретико-множественные операции с элементами массивов

3.3.6. Расчет сводных статистических показателей

В библиотеку NumPy включены ряд функций, которые позволяют вычислить ряд сводных статистических показателей: среднее значение, стандартное отклонение, сумму, произведение, медиану, минимум и максимум, квантили и т. д.

Пример использования некоторых статистических функций приведен на рис. 3.19.

```
B [65]: import numpy as np
arr=np.around(10*np.random.random_sample((3,3)))
print ("Исходный массив:\n", arr)
print ("Сумма элементов массива:", np.sum(arr))
print ("Сумма элементов массива по оси 0:", np.sum(arr,0))
print ("Сумма элементов массива по оси 1:", np.sum(arr,1))
print ("Значение минимального элемента массива:", np.min(arr))
print ("Значение максимального элемента массива по оси 1:", np.max(arr,1))
print ("Среднее значение элементов массива по оси 0:", np.mean(arr, 0))
print ("Стандартное отклонение значений элементов массива:", np.std(arr))
print ("Дисперсия значений элементов массива:", np.var(arr))

Исходный массив:
 [[6. 1. 8.]
 [6. 5. 2.]
 [6. 3. 4.]]
Сумма элементов массива: 41.0
Сумма элементов массива по оси 0: [18.  9. 14.]
Сумма элементов массива по оси 1: [15. 13. 13.]
Значение минимального элемента массива: 1.0
Значение максимального элемента массива по оси 1: [8. 6. 6.]
Среднее значение элементов массива по оси 0: [6.         3.         4.66666667]
Стандартное отклонение значений элементов массива: 2.1140330656044943
Дисперсия значений элементов массива: 4.469135802469136
```

Рис. 3.19. Расчет сводных статистических показателей

Вопросы для обсуждения

1. Роль библиотеки NumPy в интеллектуальном анализе данных
2. Сходства и отличия библиотеки NumPy от MatLab
3. Представление основных объектов анализа и геометрии средствами NumPy
4. Программный интерфейс управления объектами средствами NumPy
5. Общие подходы к организации вычислений с использованием библиотеки NumPy

Практические задания

Задание 3.1.

Выполните следующие действия:

1. создайте массив №1 на основе данных, полученных при решении задания 2.1 (при создании массива применить функцию «numpy.array», тип данных массива – int);

2. создайте массив №2, размер которого соответствует размеру массива №1 и заполните его интервальными значениями (при создании массива применить функцию «numpy.arange» или функцию «numpy.linspace», диапазоны интервала и шаг (или количество значений) задать самостоятельно, тип данных массива – float);

3. создайте массив №3, размер которого соответствует размеру массива №1, и заполните его случайными элементами, распределёнными по заданному закону (при создании массива применить функцию, включенную в состав пакета «numpy.random» (по вариантам), тип данных массива – float);

4. сделать детальное описание используемой в п.3 функции, т.е. привести теоретическое описание, привести общий вид функции и описать состав её аргументов и возвращаемое значение;

5. осуществите следующие векторные операции (тип данных результирующих массивов – float):

- сложить массивы №1, №2, №3 и умножить результат на 5 (скаляр);
- вычесть из массива №1 массив №2;
- умножить массивы №1, №3;
- разделить массив №1 на массив №2;
- возвести элементы массива №1 в степень 3.

Задание по вариантам (функции распределения)

№	Функция распределения	№	Функция распределения	№	Функция распределения
1	beta	11	laplace	21	pareto
2	binomial	12	logistic	22	poisson
3	chisquare	13	lognormal	23	power
4	dirichlet	14	logseries	24	rayleigh
5	exponential	15	multinomial	25	standard_cauchy
6	f	16	multivariate_normal	26	standard_exponential
7	gamma	17	negative_binomial	27	standard_gamma
8	geometric	18	noncentral_chisquare	28	standard_normal
9	gumbel	19	noncentral_f	29	standard_t
10	hypergeometric	20	normal	30	triangular

Задание 3.2.

На основе данных об изменении стоимости активов рассчитайте экономико-статистические показатели с целью дальнейшей оценки

рисков инвестирования: минимальное значение; максимальное значение; математическое ожидание; дисперсию; стандартное отклонение; среднеквадратическое отклонение и коэффициент вариации.

Для выполнения данного задания необходимо выполнить ниже перечисленные действия:

1. создать массив на основании официальных статистических данных о курсе акций за предыдущий месяц (тип данных массива – str, источник данных – https://ru.investing.com/equities/, эмитент акций – по вариантам, применить функции (на выбор): «numpy.loadtxt», «numpy.fromfile»);

2. создать массив, который должен содержать только информацию о ценах за каждый торговый день: ценах открытия, минимальных ценах, максимальных ценах и ценах закрытия (тип данных массива – float, источник данных – созданный в п.1 задания массив, эмитент акций – по вариантам, применить функции (на выбор): «numpy.split», «numpy.split», «numpy.hsplit», использование среза совместно с «numpy.array»);

3. создать массив (вектор-столбец), который должен содержать информацию о средней цене актива за каждый торговый день (применить функции (совместно): «numpy.mean», «numpy.Reshape»);

4. осуществить слияние массивов, созданных на шаге 2 (слева) и шаге 3 (справа) (применить функции (на выбор): «numpy.concatenate», «numpy.hstack»);

5. рассчитать минимальное значение, максимальное значение, математическое ожидание, дисперсию, стандартное отклонение, среднеквадратическое отклонение и коэффициент вариации по всем видам цен активов. Для расчетов показателей использовать следующие функции и способы расчета:

- математическое ожидание («numpy.mean»);
- дисперсия («numpy.mean»);
- стандартное отклонение («numpy.std»);
- среднеквадратическое отклонение (корень квадратный из дисперсии);
- коэффициент вариации (среднеквадратическое отклонение/математическое ожидание));

6. запишите полученный массив на диск (применить функцию: «numpy.save»)

Задание по вариантам

№	Эмитент акций	№	Эмитент акций	№	Эмитент акций
1	ВТБ	11	РУСАЛ	21	Ростелеком
2	ФСК ЕЭС ОАО	12	АК АЛРОСА	22	НОВАТЭК
3	РусГидро	13	Система	23	Детский мир
4	Интер РАО ЕЭС ОАО	14	ММК ОАО	24	ЛУКОЙЛ
5	МКБ	15	НЛМК	25	ТМК
6	Сбербанк	16	Роснефть	26	Северсталь
7	Газпром	17	М.видео	27	ФосАгро
8	Магнит	18	Московская биржа	28	Lenta Ltd
9	Сургутнефтегаз	19	Татнефть	29	X5 Retail Group
10	Аэрофлот	20	МТС	30	Яндекс

Библиографический список

1. NumPy Reference // NumPy: [Электронный ресурс] – Режим доступа: https://numpy.org/doc/stable/reference/

2. Array creation routines // NumPy: [Электронный ресурс] – Режим доступа: https://numpy.org/doc/stable/reference/routines.array-creation.html

3. Numpy.dtype // NumPy: [Электронный ресурс] – Режим доступа: https://numpy.org/doc/stable/reference/generated/numpy.dtype.html

4. Random sampling (numpy.random) // NumPy: [Электронный ресурс]. – Режим доступа: https://numpy.org/doc/1.16/reference/routines.random.html

5. Очан Ю. С., Теория операций над множествами [Электронный ресурс] – Режим доступа: http://www.mathnet.ru/links/f83bb6ad2964894b49fb7ab23989cf21/rm7999.pdf

7. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

8. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

9. Маккинни, У. Маккинли, У. Python и анализ данных / Уэс Маккинли ; пер. с англ. А.А. Слинкина. - Москва : ДМК Пресс, 2015. - 482 с. - ISBN 978-5-97060-315-4. - Текст: электронный. - URL: https://znanium.com/catalog/product/1027796.

Глава 4. PANDAS: ОБРАБОТКА И АНАЛИЗ НАБОРОВ РАЗНОРОДНЫХ ДАННЫХ

В данной главе рассматриваются следующие вопросы:

1. Функциональные возможности python-библиотеки pandas

2. Основные структуры данных

3. Обзор основных операций с наборами однородных данных

4.1. Функциональные возможности python-библиотеки pandas

Pandas – программная библиотека на языке Python, ориентированная на анализ разнородных данных, представленных в структурированном виде. Название происходит от эконометрического термина «панельные данные» (panel data), используемого для описания многомерных структурированных наборов информации.

Pandas предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами, в т.ч. дает возможность:

- иметь удобный доступ к табличным данным;
- осуществлять сбор данных из различных источников и форматов файлов;
- осуществлять первичную оценку данных по общим показателям;
- осуществлять объединение наборов данных;
- осуществлять подготовку и очистку данных (заполнение, замену, обработку пустых значений, обнаружение и фильтрация выбросов);
- выполнять агрегирование и группировку данных;
- строить сводные таблицы;
- осуществлять работу с временными рядами;
- осуществлять векторизованные операции над строками.

Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Пакет pandas принято импортировать, используя в качестве его псевдонима «pd».

4.2. Основные структуры данных

Основными структурами данных, которые предоставляет python-библиотека pandas, являются следующие объекты:

1) «DataFrame» (примерно соответствует таблице в реляционной базе данных – со строками и поименованными колонками)

2) «Series» (соответствует отдельной поименованной колонке объекта DataFrame);

3) «Index» (соответствует меткам вдоль осей).

4.2.1. Объект «Series»

Объект «Series» – это одномерный массив, предназначенный для хранения любых типов данных (целые числа, строки, числа с плавающей запятой, объекты Python и т. д.) и имеющий метки, которые именуются индексами.

Основной метод создания объекта Series – вызов его конструктора:

pandas.Series (data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)

Основные аргументы функции:

data (тип данных – array_like (любой объект, предоставляющий интерфейс массива), итератор, словарь или скаляр) – содержит данные, хранящиеся в серии;

index (тип данных – array_like (любой объект, предоставляющий интерфейс массива), необязательный аргумент) – содержит метки данных, хранящихся в серии (по умолчанию заполняется значениями от 0 до значения размера data, допускаются неуникальные значения индекса);

dtype (тип данных – str, необязательный аргумент) – имя присваиваемое создаваемому объекту;

name (тип данных – str, numpy.dtype или ExtensionDtype, необязательный аргумент) – тип данных создаваемого объекта.

В качестве источника данных и наименования меток для объекта «Series» может выступать любой объект, предоставляющий интерфейс массива, в т.ч. объект «numpy». Пример создания объекта «Series» на основе объекта «numpy» приведен на рис. 4.1.

```
В [45]: import numpy as np
        import pandas as pd

        arr=np.random.random(3)
        ind=["а", "б", "в"]
        s1 = pd.Series(arr)
        s2 = pd.Series(arr, ind)
        s3 = pd.Series(arr, ind, dtype="string")
        print ("Создание объекта Series на основе объекта numpy")
        print ("-- без указания меток и типа возвращаемых данных\n",s1)
        print ("-- с указанием меток, но без указания типа возвращаемых данных\n",s2)
        print ("-- с указанием меток и типа возвращаемых данных\n",s3)

        Создание объекта Series на основе объекта numpy
        -- без указания меток и типа возвращаемых данных
         0    0.093957
        1    0.909203
        2    0.388927
        dtype: float64
        -- с указанием меток, но без указания типа возвращаемых данных
         а    0.093957
        б    0.909203
        в    0.388927
        dtype: float64
        -- с указанием меток и типа возвращаемых данных
         а    0.09395712919414834
        б    0.9092028098402184
        в    0.3889265247963012
        dtype: string
```

Рис. 4.1. Создание объекта «Series» на основе объекта «numpy»

Представление массива данных и индекса объекта «Series» можно получить помощью атрибутов «values» и «index». Пример отображения информации о структуре объекта «Series приведен на рис. 4.2.

```
В [15]: import numpy as np
        import pandas as pd
        arr=np.random.random(3)
        ind=["а", "б", "в"]
        s1 = pd.Series(arr, ind)
        print ("Массив меток объекта Series:\n", s1.index)
        print ("Массив данных объекта Series:\n", s1.values)

        Массив меток объекта Series:
         Index(['а', 'б', 'в'], dtype='object')
        Массив данных объекта Series:
         [0.98069255 0.18241729 0.19087464]
```

Рис. 4.2. Представление данных и индекса объекта «Series»

Также в качестве источника данных для объекта «Series» может выступать словарь или скалярное значение. Пример создания объекта «Series» на основе словаря и скалярного значения, а также получение значения элемента данных по его индексу и присвоение имени объекту приведен на рис. 4.3.

```
В [28]: import numpy as np
        import pandas as pd
        s4 = pd.Series({"А": 1, "Б": 0, "В": 2})
        print ("Создание объекта Series на основе словаря:\n",s4)
        s5 = pd.Series(2, index=["а", "б", "в"])
        print ("Создание объекта Series на основе скаляра:\n",s5)
        print ("Получение значения элемента данных по индексу:",s4 ['Б'])
        s5.name = "Series_s5"
        print ("Присвоение объекту Series имени:\n",s5)

        Создание объекта Series на основе словаря:
         А    1
        Б    0
        В    2
        dtype: int64
        Создание объекта Series на основе скаляра:
         а    2
        б    2
        в    2
        dtype: int64
        Получение значения элемента данных по индексу: 0
        Присвоение объекту Series имени:
         а    2
        б    2
        в    2
        Name: Series_s5, dtype: int64
```

Рис. 4.3. Создание объекта «Series» на основе словаря и скалярного значения

Так как объекты «Series» основаны на объектах «ndarray», то к ним применимы большинство функций библиотеки NumPy.

Вследствие того, что по своей сути объекты «Series» представляют коллекцию из двух массивов (один используется для хранения данных, а второй – для хранений меток), то ряд операций будут применяться не только к значениям, но и к индексам. При этом такие операции позволяют сохранить связь между индексом и значением.

Пример операций с объектами «Series» с использованием функций библиотеки NumPy приведен на рис. 4.4.

В отличие от обычного массива NumPy, для выборки одного или нескольких элементов из объекта «Series» можно использовать значения индекса.

```
В [21]: import numpy as np
        import pandas as pd
        s1 = pd.Series(np.random.random(3), ["а", "б", "в"])
        s2 = pd.Series(np.random.random(3), ["а", "б", "г"])
        print ("Исходный объект Series (s1):\n",s1)
        print ("Исходный объект Series (s2):\n",s2)
        print ("Сумма s1 и s2:\n",s1+s2)
        print ("Все элементы s1 с <б> до последнего:\n",s1[1:])
        print ("Все элементы s1 с нулевого до <в> (включительно)\n",s1[:2])
        print ("Сумма элементов s1:", np.sum(s1))
        print ("Максимальное значение s1 - ", np.max(s1))
        print ("Дисперсия элементов s1 - ", np.var(s1))

        Исходный объект Series (s1):
         а    0.598982
        б    0.843023
        в    0.884373
        dtype: float64
        Исходный объект Series (s2):
         а    0.177694
        б    0.184607
        г    0.034660
        dtype: float64
        Сумма s1 и s2:
         а    0.776676
        б    1.027630
        в         NaN
        г         NaN
        dtype: float64
        Все элементы s1 с <б> до последнего:
         б    0.843023
        в    0.884373
        dtype: float64
        Все элементы s1 с нулевого до <в> (включительно)
         а    0.598982
        б    0.843023
        dtype: float64
        Сумма элементов s1: 2.32637877540567
        Максимальное значение s1 -  0.8843732739808202
        Дисперсия элементов s1 -  0.015857112195383893
```

Рис. 4.4. Операции с объектами «Series» с использованием функций библиотеки NumPy

Следует также отметить, что объект «Series», по своей сути, является упорядоченным словарём фиксированной длины, поскольку он позволяет сопоставлять индекс и данные. Это даёт возможность передавать объект «Series» ожидающим получить словарь в качестве аргумента функциям.

4.2.2. Объект «DataFrame»

Объект «DataFrame» – табличная структура данных (фрейм данных), состоящая из упорядоченной коллекции столбцов, причем типы

значений (числовой, строковый, булев и т. д.) в разных столбцах могут различаться.

Основной метод создания объекта «DataFrame» – вызов его конструктора:

pandas.DataFrame (data=None, index=None, columns=None, dtype=None, copy=False)

Основные аргументы функции:

data (тип данных – массив (ndarray), итератор, словарь или другой DataFrame) – содержит данные фрейма данных ;

index (тип данных – array_like (любой объект, предоставляющий интерфейс массива), необязательный аргумент) – содержит метки данных, хранящихся в серии (по умолчанию заполняется значениями от 0 до значения размера data, допускаются неуникальные значения индекса);

columns (тип данных – array_like (любой объект, предоставляющий интерфейс массива), необязательный аргумент) – содержит значения меток столбцов фрейма данных;

dtype (тип данных – dtype, необязательный аргумент) – тип данных создаваемого объекта.

Пример создания объекта «DataFrame» на основе объекта «numpy» приведен на рис. 4.5.

```
В [8]: import numpy as np
       import pandas as pd
       df1= pd.DataFrame(np.arange(8).reshape((2,4)),
          index=['строка-1', 'строка-2'],
          columns=['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'])
       df1
```

Out[8]:

	столбец-1	столбец-2	столбец-3	столбец-4
строка-1	0	1	2	3
строка-2	4	5	6	7

Рис. 4.5. Создание объекта «DataFrame» на основе объекта «numpy»

В объекте «DataFrame» хранятся два индекса: по строкам и по столбцам. Внутри объекта данные хранятся в виде одного или не-

скольких двумерных блоков, а не в виде списка, словаря или еще какой-нибудь коллекции одномерных массивов.

Также в качестве источника данных для объекта «DataFrame» может выступать словарь. Пример создания объекта «DataFrame» на основе словаря, содержащего в качестве данных объекты «Series», приведен на рис. 4.6.

```
В [37]: import pandas as pd
        d = {
             "первый": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
             "второй": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
             "третий": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "f"]),
            }
        pd.DataFrame(d)
```

Out[37]:

	первый	второй	третий
a	1.0	1.0	1.0
b	2.0	2.0	2.0
c	3.0	3.0	3.0
d	NaN	4.0	NaN
f	NaN	NaN	4.0

Рис. 4.6. Создание объекта «DataFrame» на основе словаря

Еще один способ создание объекта «DataFrame» – объединить нескольких объектов «Series» приведен на рис. 4.7.

```
В [60]: import numpy as np
        import pandas as pd
        s1 = pd.Series(np.random.random(3), ["a", "б", "в"])
        s2 = pd.Series(np.random.random(3), ["a", "б", "г"])
        s3 = pd.Series(np.random.random(4), ["a", "б", "в","г"])
        df1= pd.DataFrame([s1, s2, s3])
        df1
```

Out[60]:

	а	б	в	г
0	0.899033	0.182250	0.527971	NaN
1	0.424019	0.918540	NaN	0.665953
2	0.674199	0.738781	0.119863	0.264713

Рис. 4.7. Создание объекта «DataFrame» на основе нескольких объектов «Series»

При необходимости, добавление нового столбца может быть осуществлено посредством указания в качестве индекса у объекта «DataFrame» имени создаваемого столбца и присвоении ему объекта

«Series». Для добавления нового столбца в объект «DataFrame» также может быть использован метод «assign», который возвращает новый объект со всеми исходными столбцами в дополнение к новым. Пример добавления нового столбца приведен на рис. 4.8.

```
В [147]: import numpy as np
         import pandas as pd
         df1= pd.DataFrame(np.arange(9).reshape((3,3)),
            index=['строка-1', 'строка-2', 'строка-3'],
            columns=['ст-1', 'ст-2', 'ст-3'])
         print ("Исходный DataFrame:\n",df1)
         df1 ["Сумма"] = df1["ст-1"]+df1["ст-2"]+df1["ст-3"]
         df1=df1.assign(Произведение=df1["ст-1"] * df1["ст-2"]* df1["ст-3"])
         print ("\nИтоговый DataFrame №2:\n",df1)

         Исходный DataFrame:
                    ст-1  ст-2  ст-3
         строка-1     0     1     2
         строка-2     3     4     5
         строка-3     6     7     8

         Итоговый DataFrame №2:
                    ст-1  ст-2  ст-3  Сумма  Произведение
         строка-1     0     1     2      3             0
         строка-2     3     4     5     12            60
         строка-3     6     7     8     21           336
```

Рис. 4.8 Добавления нового столбца в объект «DataFrame»

4.2.3. Объект Index

Как объект «Series», так и объект «DataFrame» содержат явный индекс, обеспечивающий возможность ссылаться на данные и модифицировать их. Для хранения меток вдоль осей и прочих метаданных используются объекты «Index», которые можно рассматривать или как неизменяемый массив, или как упорядоченное множество.

Объекты «Index» являются неизменяемыми, то есть их нельзя модифицировать стандартными средствами. Неизменяемость делает безопаснее совместное использование индексов несколькими объектами «DataFrame» и «Series», исключая возможность побочных эффектов в виде случайной модификации индекса по неосторожности.

4.2.4. Доступ к отдельным элементам объекта «DataFrame»

Аналогично объекту «Series» у объекта «DataFrame» имеется атрибут «index», обеспечивающий доступ к меткам строк (индекса), атрибут «values», обеспечивающий доступ к данным, а также есть атрибут «columns», обеспечивающий доступ к меткам столбцов. Пример использования указанных атрибутов для доступа к данным объекта «DataFrame» приведен на рис. 4.9.

```
В [73]: import numpy as np
        import pandas as pd
        df1= pd.DataFrame(np.arange(16).reshape((4,4)),
           index=['строка-1', 'строка-2', 'строка-3', 'строка-4'],
           columns=['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'])
        print ("Метки строк:\n",df1.index)
        print ("Данные:\n",df1.values)
        print ("Метки столбцов:\n",df1.columns)

        Метки строк:
         Index(['строка-1', 'строка-2', 'строка-3', 'строка-4'], dtype='object')
        Данные:
         [[ 0  1  2  3]
         [ 4  5  6  7]
         [ 8  9 10 11]
         [12 13 14 15]]
        Метки столбцов:
         Index(['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'], dtype='object')
```

Рис. 4.9. Доступ к данным и меткам строк и столбцов объекта «DataFrame»

Объект «DataFrame» во многом ведет себя аналогично двумерному или структурированному массиву, а также словарю объектов «Series» с общим индексом.

К отдельным объектам «Series», составляющим столбцы объекта «DataFrame», можно обращаться посредством такой же индексации, как и для словарей, по имени столбца (для доступа к строкам используется функция «loc»).

Пример доступа, к отдельным объектам «Series», составляющим столбцы объекта «DataFrame», а также к отдельным элементам данных приведен на рис. 4.10.

Также можно обращаться к данным объекта «DataFrame» с помощью атрибутов, используя в их качестве строковые имена столбцов.

```
В [125]: import numpy as np
         import pandas as pd
         df1= pd.DataFrame(np.arange(16).reshape((4,4)),
            index=['строка-1', 'строка-2', 'строка-3', 'строка-4'],
            columns=['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'])
         s1=df1["столбец-2"]
         print ("Доступ к значению элементов в столбце №2:\n",s1)
         print ("Доступ к значению элемента в строке №2 (столбец 2):",s1["строка-2"])
         print ("Доступ к значению элементов в строке №2:\n",df1.loc["строка-2"])

         Доступ к значению элементов в столбце №2:
          строка-1     1
         строка-2     5
         строка-3     9
         строка-4    13
         Name: столбец-2, dtype: int32
         Доступ к значению элемента в строке №2 (столбец 2): 5
         Доступ к значению элементов в строке №2:
          столбец-1    4
         столбец-2    5
         столбец-3    6
         столбец-4    7
         Name: строка-2, dtype: int32
```

Рис. 4.10. Доступ к данным объекта «DataFrame»

К отдельным элементам данных объекта «DataFrame» можно также обращаться посредством указания индексов в атрибуте «values». Пример использования атрибута «values» с указанием индексов для обращения к отдельным элементам данных объекта «DataFrame» приведен на рис. 4.11.

```
В [101]: import numpy as np
         import pandas as pd
         df1= pd.DataFrame(np.arange(16).reshape((4,4)),
            index=['строка-1', 'строка-2', 'строка-3', 'строка-4'],
            columns=['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'])
         print (df1)
         print ("Доступ к значению элементов в строке №2:\n",df1.values[1])
         print ("Доступ к значению элементов в столбце №2:\n",df1.values[:,1])
         print ("Доступ к значению элемента в строке №2 и столбце №2:",df1.values[1,1])

                   столбец-1  столбец-2  столбец-3  столбец-4
         строка-1          0          1          2          3
         строка-2          4          5          6          7
         строка-3          8          9         10         11
         строка-4         12         13         14         15
         Доступ к значению элементов в строке №2:
          [4 5 6 7]
         Доступ к значению элементов в столбце №2:
          [ 1  5  9 13]
         Доступ к значению элемента в строке №2 и столбце №2: 5
```

Рис. 4.11. Доступ к данным объекта «DataFrame» с использованием атрибута «values»

4.2.5. Загрузка данных в объект «DataFrame»

Создание объекта «DataFrame» из данных, содержащихся в текстовом или бинарном файле можно осуществить с помощью ряда функций, которые содержаться в библиотеке pandas и предназначенные для чтения файлов различных форматов: read_csv, read_json, read_html, read_clipboard, read_excel и другие [1].

При загрузке данных приходится сталкиваться со следующими проблемами:

1) необходимость выбора имени столбцов и строк, а также порядка столбцов (проблема индексирования);

2) необходимость преобразования значений в необходимый тип данных, а при отсутствии значений введение маркеров отсутствующих данных (проблема выведения типа и преобразование данных);

3) необходимость работы с большими файлами (проблема итерирования);

4) необходимость работы с неполными, избыточными или ошибочными данными (проблема «грязных данных»).

Поэтому у функций чтения данных, как правило, много аргументов, которые помогают справиться с широким разнообразием файловых форматов при загрузке данных в объект «DataFrame».

Создание объекта «DataFrame» из данных, содержащихся в файле формата «csv» осуществляется с помощью функции «pandas.read_csv», описание которой выглядит следующим образом:

pandas.read_csv(filepath_or_buffer, sep=<object object>, delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, es-

capechar=None, comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None)

Основные аргументы функции:

filepath_or_buffer (тип данных – str, объект path, файловый объект) – файловый объект или путь к файлу;

sep или delimiter (тип данных – str, необязательный аргумент) – разделитель между элементами в текстовом файле (значение по умолчанию – «,»).

Пример использования основных аргументов функции «pandas.read_csv» приведен на рис. 4.12.

```
B [7]: import pandas as pd
       df1=pd.read_csv('Z:/test.csv', sep =",")
       df1.head(4)
```

Out[7]:

	Дата	Цена	Откр.	Макс.	Мин.	Объём	Изм. %
0	24.03.2021	7,067	6,988	7,076	6,957	44,77M	1,19%
1	23.03.2021	6,984	7,025	7,046	6,967	94,57M	-0,58%
2	22.03.2021	7,025	7,083	7,125	7,003	49,31M	-0,80%
3	19.03.2021	7,082	6,920	7,120	6,792	373,10M	2,06%

Рис. 4.12. Использование основных аргументов функции «pandas.read_csv» для загрузки данных в объект «DataFrame»

Аргументы функции, решающие проблему индексации:

header (тип данных – int (или список int), необязательный аргумент) – определяет номера строк для использования в качестве имен столбцов и начала данных (значение по умолчанию – «infer»: имена столбцов выводятся из первой строки файла)

names (тип данных – массив, необязательный аргумент) – определяет список имен столбцов результирующего объекта (значение по умолчанию – None, задается, если header=None);

index_col (тип данных – int, str или их последовательность, необязательный аргумент) – определяет номера или имена столбцов, трактуемых как индекс строк в результирующем объекте (значение по умолчанию – None);

usecols (тип данных – список значений, необязательный аргумент) – определяет подмножество столбцов для возврата в результирующем объекте (значение по умолчанию – None: возвращаются все столбцы).

Пример использования аргументов функции «pandas.read_csv», решающих проблему индексации приведен на рис. 4.13.

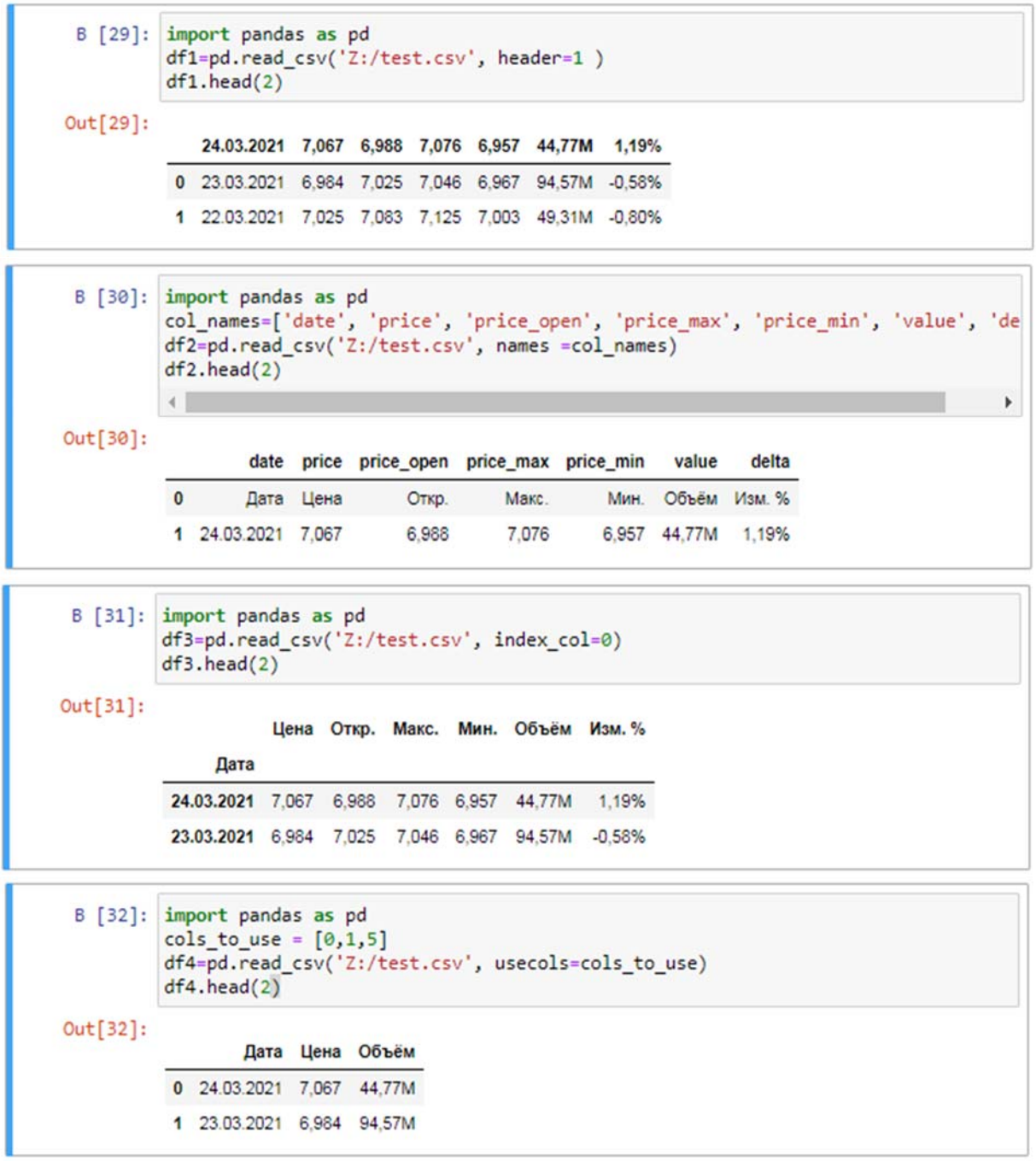


Рис. 4.13. Использование аргументов функции «pandas.read_csv», решающих проблему индексации, для загрузки данных в объект «DataFrame»

Аргументы функции, решающие проблему выведения типа и преобразование данных:

dtype (тип данных – словарь соотнесенных с метками столбцов типов данных, необязательный аргумент) – определяет тип данных для значений, содержащихся в определённом столбце (значение по умолчанию – None, при невозможности корректного преобразования значений в необходимый тип данных выдаётся ошибка);

converters (тип данных – словарь соотнесенных с метками столбцов функций преобразования данных, необязательный аргумент) – определяет функции для преобразования значений в определенных столбцах (значение по умолчанию – None);

skiprows (тип данных – int (или список int), необязательный аргумент) – определяет количество игнорируемых (или список номеров) начальных строк (нумерация начинается с 0) (значение по умолчанию – None);

skipfooter (тип данных – int (или список int), необязательный аргумент) – определяет количество (или список номеров) игнорируемых конечных строк (нумерация начинается с 0) (значение по умолчанию – None);

na_values (тип данных – скалярная величина, строка, список или словарь, необязательный аргумент) – определяет последовательность значений, интерпретируемых как маркеры отсутствующих данных (по умолчанию следующие значения интерпретируются как NaN: ‘’, ‘#N/A’, ‘#N/A N/A’, ‘#NA’, ‘-1.#IND’, ‘-1.#QNAN’, ‘-NaN’, ‘-nan’, ‘1.#IND’, ‘1.#QNAN’, ‘<NA>’, ‘N/A’, ‘NA’, ‘NULL’, ‘NaN’, ‘n/a’, ‘nan’, ‘null’);

skip_blank_lines (тип данных – boolean, необязательный аргумент) – определяет, необходимость пропуска пустых строк без интерпретации их как значения NaN (значение по умолчанию – True: пустые строки игнорируются).

Пример использования аргументов функции «pandas.read_csv», решающих проблему выведения типа и преобразование данных приведен на рис. 4.14 и 4.15.

```
B [71]: import numpy as np
        import pandas as pd
        def convert_date(series):
            return pd.to_datetime(series, format="%d.%m.%Y")
        def convert_price(series):
            return pd.to_numeric(series.replace(',','.'))
        w = lambda x: (x.replace('M',''))

        cols_to_use = [0,1,5,6]
        col_names=['date', 'price', 'value', 'delta']
        df=pd.read_csv(
            'Z:/test.csv',
            usecols=cols_to_use,
            names=col_names,
            skiprows=1,
                converters={'date':convert_datetime,'value':w,'price':convert_price},
            dtype={'delta': str}
        )
        df.info()
        df.head(3)

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 20 entries, 0 to 19
        Data columns (total 4 columns):
         #   Column  Non-Null Count  Dtype
        ---  ------  --------------  -----
         0   date    20 non-null     datetime64[ns]
         1   price   20 non-null     float64
         2   value   20 non-null     object
         3   delta   20 non-null     object
        dtypes: datetime64[ns](1), float64(1), object(2)
        memory usage: 768.0+ bytes
```

Out[71]:

	date	price	value	delta
0	2021-03-24	7.067	44,77	1,19%
1	2021-03-23	6.984	94,57	-0,58%
2	2021-03-22	7.025	49,31	-0,80%

Рис. 4.14. Использование аргументов функции «pandas.read_csv», решающих проблему выведения типа и преобразование данных, для загрузки данных в объект «DataFrame» (часть 1)

Аргументы функции, решающие проблему «грязных данных»:

error_bad_lines (тип данных – boolean, необязательный аргумент) – определяет, необходимость удаления строки со слишком большим количеством значений (значение по умолчанию – True: пустые строки игнорируются);

decimal (тип данных – str, необязательный аргумент) – определяет знак, который нужно распознать как десятичную точку (значение по умолчанию – ‘.’);

thousands (тип данных – str, необязательный аргумент) – определяет знак, который нужно распознать как разделитель тысяч (значение по умолчанию – None).

```
Содержимое файла test.csv:
"Дата","Цена","Откр.","Макс.","Мин.","Объём","Изм. %"
"24.03.2021","7.067","6,988","7,076","6,957","nan","1,19%"

"23.03.2021","6.984","7,025","7,046","*","94,57M","-0,58%"
"22.03.2021","7.025","7,083","","7,003","49,31M","-0,80%"
```

```
In [91]: import pandas as pd
df=pd.read_csv(
    'Z:/test.csv',
    na_values=["*","@"],
    skip_blank_lines=False,
    nrows=4
)
df
```

Out[91]:

	Дата	Цена	Откр.	Макс.	Мин.	Объём	Изм. %
0	24.03.2021	7.067	6,988	7,076	6,957	NaN	1,19%
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	23.03.2021	6.984	7,025	7,046	NaN	94,57M	-0,58%
3	22.03.2021	7.025	7,083	NaN	7,003	49,31M	-0,80%

Рис. 4.15. Использование аргументов функции «pandas.read_csv», решающих проблему выведения типа и преобразование данных, для загрузки данных в объект «DataFrame» (часть 2)

4.3. Обзор операций с наборами разнородных данных

Список операций с наборами разнородных данных, которые предоставляет возможность осуществлять библиотека pandas широк и разнообразен. Приведем и далее более подробно рассмотрим ключевые возможности pandas, которые могут быть задействованы при проведении интеллектуального анализа данных:

- проведение сортировки и ранжирования данных;
- применение стандартных математических и статистических методов;
- осуществление обработки отсутствующих данных;
- осуществление объединения наборов данных;
- осуществление группировки, агрегирования, фильтрации, преобразования данных и формирование сводных таблиц.

4.3.1. Сортировка и ранжирование данных

Сортировка – последовательное расположение элементов данных объекта «DataFrame» в зависимости от выбранного критерия.

Объект «DataFrame» можно сортировать по индексу меток строк или столбцов, а также по значениям в одном или нескольких столбцах.

В первом случае используется функция «sort_index», во втором случае – функция «sort_values». Пример сортировки элементов данных объекта «DataFrame» с помощью указанных функций приведен на рис. 4.16.

```
В [124]: import pandas as pd
df = pd.DataFrame({
    'col4': ['a',  'D', 'F'],
    'col1': ['A',  np.nan, 'C'],
    'col3': [0, 9,  3],
    'col2': [2, 9,  4]
})
print ("Исходный набор данных:\n",df)
print ("Сортировка по значениям в 1 и 2 столбце (прямая сортировка):\n",
       df.sort_values(by=['col1', 'col2']))
print ("Сортировка по значениям в 1 столбце (обратная сортировка):\n",
       df.sort_values(by=['col1'], ascending=False))
print ("Сортировка по меткам строк:\n",df.sort_index(axis=0))
print ("Сортировка по меткам столбцов:\n",df.sort_index(axis=1))

Исходный набор данных:
   col4 col1  col3  col2
0    a    A     0     2
1    D  NaN     9     9
2    F    C     3     4
Сортировка по значениям в 1 и 2 столбце (прямая сортировка):
   col4 col1  col3  col2
0    a    A     0     2
2    F    C     3     4
1    D  NaN     9     9
Сортировка по значениям в 1 столбце (обратная сортировка):
   col4 col1  col3  col2
2    F    C     3     4
0    a    A     0     2
1    D  NaN     9     9
Сортировка по меткам строк:
   col4 col1  col3  col2
0    a    A     0     2
1    D  NaN     9     9
2    F    C     3     4
Сортировка по меткам столбцов:
   col1  col2  col3 col4
0    A     2     0    a
1  NaN     9     9    D
2    C     4     3    F
```

Рис. 4.16. Сортировка элементов данных

Ранжирование – присваивание рангов (от единицы до числа присутствующих в наборе данных элементов).

Для ранжирования применяется метод «rank» объектов «Series» и «DataFrame»; по умолчанию «rank» обрабатывает связанные ранги, присваивая каждой группе средний ранг.

Пример ранжирования элементов данных объекта «DataFrame» приведен на рис. 4.17.

```
В [129]: import pandas as pd
         df = pd.DataFrame(data={'Animal': ['кошка', 'цапля', 'собака',
                                            'паук', 'змея'],
                                 'Number_legs': [4, 2, 4, 8, np.nan]})
         df
```

Out[129]:

	Animal	Number_legs
0	кошка	4.0
1	цапля	2.0
2	собака	4.0
3	паук	8.0
4	змея	NaN

```
В [131]: df['default_rank'] = df['Number_legs'].rank()
         df['max_rank'] = df['Number_legs'].rank(method='max')
         df['NA_bottom'] = df['Number_legs'].rank(na_option='bottom')
         df['pct_rank'] = df['Number_legs'].rank(pct=True)
         df
```

Out[131]:

	Animal	Number_legs	default_rank	max_rank	pct_rank	NA_bottom
0	кошка	4.0	2.5	3.0	0.625	2.5
1	цапля	2.0	1.0	1.0	0.250	1.0
2	собака	4.0	2.5	3.0	0.625	2.5
3	паук	8.0	4.0	4.0	1.000	4.0
4	змея	NaN	NaN	NaN	NaN	5.0

Рис. 4.17. Ранжирование элементов данных

4.3.2. Стандартные математические и статистические методы

Объекты «Series» и «DataFrame» оснащены набором стандартных математических и статистических методов. Особенностью данных методов являются то, что при их использовании игнорируются отсутствующие значения в указанных объектах.

Первая часть методов относится к категории редукций, или сводных статистик – методов, которые вычисляют единственное значение (например, метод «sum», «mean» и т.д.) для объекта «Series» или объект «Series» – для строк либо столбцов объекта «DataFrame».

Вторая часть методов относится к категории аккумулирующих методов, например метод «cumsum», который возвращает кумулятивные суммы по осям объекта «DataFrame» или объекта «Series».

Существуют также методы, не относящиеся ни к редуцирующим, ни к аккумулирующим. Примером может служить метод «describe», который возвращает несколько сводных статистических показателей за одно обращение.

Пример получения сводной статистики с применением метода «describe» объекта «DataFrame» приведен на рис. 4.18.

```
B [144]: import pandas as pd
         df=pd.read_csv('Z:/test.csv', decimal=",")
         df.describe()
```

Out[144]:

	Откр.	Макс.	Мин.
count	30.000000	30.000000	30.000000
mean	6.837700	6.917467	6.789933
std	0.125392	0.096122	0.118759
min	6.520000	6.754000	6.488000
25%	6.753000	6.841250	6.712000
50%	6.883000	6.923000	6.824000
75%	6.920750	7.000000	6.880250
max	6.989000	7.120000	6.947000

Рис. 4.18. Получение сводной статистики с применением метода «describe» объекта «DataFrame»

Некоторые сводные статистики, например корреляция и ковариация, вычисляются по парам аргументов.

Метод «corr» объекта «Series» вычисляет корреляцию перекрывающихся, отличных от NA, выровненных по индексу значений в двух объектах «Series». Соответственно, метод «cov» вычисляет ковариацию.

Пример расчета корреляции и ковариации двух перекрывающихся объектах «Series» приведен на рис. 4.19.

```
B [155]: import pandas as pd
         df=pd.read_csv('Z:/test.csv', decimal=",")
         df.cov()
```

Out[155]:

	Откр.	Макс.	Мин.
Откр.	0.015723	0.010020	0.013966
Макс.	0.010020	0.009239	0.009493
Мин.	0.013966	0.009493	0.014104

```
B [154]: df.corr()
```

Out[154]:

	Откр.	Макс.	Мин.
Откр.	1.000000	0.831354	0.937863
Макс.	0.831354	1.000000	0.831633
Мин.	0.937863	0.831633	1.000000

Рис. 4.19. Расчет корреляции и ковариации двух перекрывающихся объектах «Series»

С помощью метода corrwith объекта DataFrame можно вычислить попарные корреляции между столбцами или строками DataFrame и другим объектом Series или DataFrame.

4.3.3. Обработка отсутствующих данных

Обработка отсутствующих данных является стандартной задачей при работе с наборами реальных данных при проведении их интеллектуального анализа.

Существует несколько методов обработки пустых значений:

1) удаление содержащих пустые значения строк;
2) заполнение пропусков нулями;
3) заполнение медианой;
4) заполнение средним арифметическим значением;
5) введение индикаторных переменных.

Удаление содержащих пустые значения строк или столбцов может быть осуществлено с помощью функции «dropna».

По умолчанию функция «dropna» отбрасывает все строки, в которых присутствует хотя бы одно пустое значение, а установка параметра «axis» в значение равное единицы отбрасывает все столбцы,

содержащие хотя бы одно пустое значение. Пример применения функции «dropna» приведен на рис. 4.20.

```
B [160]: df = pd.DataFrame( [
             [1, np.nan, 2],
             [2, 3, 5],
             [np.nan, 4, 6]
         ])
         df1=df.dropna ()
         df1

Out[160]:
```

	0	1	2
1	2.0	3.0	5

```
B [161]: df2=df.dropna (axis=1)
         df2

Out[161]:
```

	2
0	2
1	5
2	6

Рис. 4.20. Удаление содержащих пустые значения строк или столбцов

С помощью параметров «how» и «thresh» функции «dropna» можно установить допустимое количество пустых значений в строке или столбце с целью обеспечения контроля удаления строк или столбцов.

Заполнение пропусков нулями является возможным, но не лучшим решением обработки отсутствующих данных, так как такая замена оказывает существенное влияние на показатели распределения значений, приводя к существенному их искажению. Это в полной мере относится и к заполнению пропусков медианой и средним арифметическим значением.

Для заполнения пустых значений в наборе данных фиксированным значением (нулём, медианой и средним арифметическим значением) может быть использована функция «fillna» с указанием значения параметра «values».

Пример заполнения пустых элементов набора данных средним арифметическим значением, рассчитанным по существующим в столбцах данным, приведен на рис. 4.21.

```
В [167]: import pandas as pd
         df = pd.DataFrame( [
             [1, np.nan, 2],
             [2, 3, 5],
             [np.nan, 4, 6]
         ])
         df.fillna (df.mean())
```

Out[167]:

	0	1	2
0	1.0	3.5	2
1	2.0	3.0	5
2	1.5	4.0	6

Рис. 4.21. Заполнение пустых элементов набора данных средним арифметическим значением

Восполнение отсутствующих данных методом «fillna» можно рассматривать как частный случай более общей замены значений с помощью метода «replace».

Устранение дубликатов осуществляется с помощью функции «drop_duplicates», который по умолчанию оставляет первую встретившуюся строку с данной комбинацией значений. Пример данной операции приведен на рис. 4.22.

```
В [173]: df = pd.DataFrame({
             'name': ['Петров П.П.', 'Иванов И.И.', 'Петров П.П.', 'Сидоров М.Ю.'],
             'group': ['БИ-121', 'БИ-122', 'БИ-121', 'БИ-123'],
             'rating': [4.2, 4.7, 4.2, 5.0]
         })
         df.drop_duplicates()
```

Out[173]:

	name	group	rating
0	Петров П.П.	БИ-121	4.2
1	Иванов И.И.	БИ-122	4.7
3	Сидоров М.Ю.	БИ-123	5.0

Рис. 4.22. Устранение дубликатов

4.3.4. Объединение наборов данных

При решение практических задач по анализу данных, как правило, требуется объединение данных из различных источников. Объекты «Series» и «DataFrame» созданы в расчете на подобные операции, и

библиотека pandas содержит следующие возможности по объединение данных из различных источников:

- конкатенация и добавление данных в конец существующего набора;

- слияние и соединение.

Конкатенация – добавление данных в конец других данных. Для простой конкатенации объектов «Series» или «DataFrame» используется функция «concat». Пример данной операции приведен на рис. 4.23.

```
В [24]: import pandas as pd
        ser1 = pd.Series(['A', 'B'], index=[1, 2])
        ser2 = pd.Series(['C', 'D'], index=[3, 4])
        ser3 = pd.concat([ser1, ser2])
        ser3

Out[24]: 1    A
         2    B
         3    C
         4    D
         dtype: object
```

```
В [23]: import numpy as np
        import pandas as pd
        df1= pd.DataFrame(np.arange(8).reshape((2,4)),
           index=['строка-1', 'строка-2'],
           columns=['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'])
        df2= pd.DataFrame(np.arange(7,15,1).reshape((2,4)),
           index=['строка-3', 'строка-4'],
           columns=['столбец-1', 'столбец-2', 'столбец-3', 'столбец-4'])
        print(df1); print(df2); print(pd.concat([df1, df2]))

                  столбец-1  столбец-2  столбец-3  столбец-4
        строка-1          0          1          2          3
        строка-2          4          5          6          7
                  столбец-1  столбец-2  столбец-3  столбец-4
        строка-3          7          8          9         10
        строка-4         11         12         13         14
                  столбец-1  столбец-2  столбец-3  столбец-4
        строка-1          0          1          2          3
        строка-2          4          5          6          7
        строка-3          7          8          9         10
        строка-4         11         12         13         14
```

Рис. 4.23. Простая конкатенация объектов «DataFrame»

Функция «concat» позволяет указывать ось, по которой будет выполняться конкатенация с помощью параметра «axis» (по умолчанию в объекте «DataFrame» конкатенация происходит построчно, т.е axis=0). При значении параметра «axis» равном единице конка-

тенация осуществляется по столбцам. Пример данной операции приведен на рис. 4.24.

```
В [20]: import numpy as np
        import pandas as pd
        df1= pd.DataFrame(np.arange(4).reshape((2,2)),
           index=['строка-1', 'строка-2'],
           columns=['столбец-1', 'столбец-2'])
        df2= pd.DataFrame(np.arange(4,8,1).reshape((2,2)),
           index=['строка-1', 'строка-2'],
           columns=['столбец-3', 'столбец-4'])
        print(df1); print(df2); print(pd.concat([df1, df2], axis=1))
```

```
          столбец-1  столбец-2
строка-1          0          1
строка-2          2          3
          столбец-3  столбец-4
строка-1          4          5
строка-2          6          7
          столбец-1  столбец-2  столбец-3  столбец-4
строка-1          0          1          4          5
строка-2          2          3          6          7
```

Рис. 4.24. Конкатенация объектов «DataFrame» по столбцам

Функции «concat» за счет широкого набора опций позволяет обеспечить широкий диапазон возможных поведений при соединении двух наборов данных (обработка дублирования индексов: игнорирование индекса, добавление ключей мультииндекса, перехват повторов как ошибки; обработка дублирования столбцов и многие другие операции).

Например, при использовании параметра «join» в значении «inner» можно добиться включение в результирующий набор всех общих столбцов (строк). Пример данной операции приведен на рис. 4.25.

Для возможности объединения наборов данных на основе реляционного подхода используется функция «merge», которая позволяет реализовать следующие типы соединений: «один-к-одному», «многие-к-одному» и «многие-ко-многим».

Пример соединения набора данных по типу «один-к-одному» (наличие в наборах данных двух ключевых столбцов с уникальными значениями) приведен на рис. 4.26.

Пример соединения набора данных по типу «многие-к-одному» (наличие в наборах данных двух ключевых столбцов, один из которых

содержит уникальные значения, а второй – дублирующийся) приведен на рис. 4.27.

```
В [44]: import numpy as np
        import pandas as pd
        df1= pd.DataFrame(np.arange(4).reshape((2,2)),
           index=['строка-1', 'строка-2'],
           columns=['столбец-1', 'столбец-2'])
        df2= pd.DataFrame(np.arange(4,8,1).reshape((2,2)),
           index=['строка-3', 'строка-4'],
           columns=['столбец-2', 'столбец-4'])
        print(df1); print(df2); print(pd.concat([df1, df2], join='inner'))

                  столбец-1  столбец-2
        строка-1          0          1
        строка-2          2          3
                  столбец-2  столбец-4
        строка-3          4          5
        строка-4          6          7
                  столбец-2
        строка-1          1
        строка-2          3
        строка-3          4
        строка-4          6
```

Рис. 4.25. Конкатенация объектов «DataFrame» с включением в результирующий набор всех общих столбцов (строк)

```
В [55]: import pandas as pd
        df1 = pd.DataFrame({'студент': ['Иванов', 'Петров', 'Сидоров'],
        'группа': ['БИ-120', 'БИ-121', 'БИ-122']})
        df2 = pd.DataFrame({'студент': ['Петров', 'Иванов', 'Сидоров'],
        'год_рождения': [2004, 2003, 2004]})
        print(df1); print(df2); print(pd.merge(df1, df2))

           студент  группа
        0   Иванов  БИ-120
        1   Петров  БИ-121
        2  Сидоров  БИ-122
           студент  год_рождения
        0   Петров          2004
        1   Иванов          2003
        2  Сидоров          2004
           студент  группа  год_рождения
        0   Иванов  БИ-120          2003
        1   Петров  БИ-121          2004
        2  Сидоров  БИ-122          2004
```

Рис. 4.26. Соединение объектов «DataFrame» по типу «один-к-одному»

По умолчанию функция «merge» осуществляет в двух входных объектах поиск столбцов с одинаковыми названиями и использует найденное в качестве ключа.

```
В [59]: import pandas as pd
df1 = pd.DataFrame({'студент': ['Иванов', 'Петров', 'Сидоров'],
'группа': ['БИ-120', 'БИ-121', 'С-121']})
df2 = pd.DataFrame({'группа': ['С-121', 'БИ-121', 'БИ-120'],
'институт': ['ИАСиЭ', 'ИЭиМ', 'ИЭиМ']})
print(df1); print(df2); print(pd.merge(df1, df2))

   студент  группа
0   Иванов  БИ-120
1   Петров  БИ-121
2  Сидоров   С-121
   группа институт
0   С-121    ИАСиЭ
1  БИ-121     ИЭиМ
2  БИ-120     ИЭиМ
   студент  группа институт
0   Иванов  БИ-120     ИЭиМ
1   Петров  БИ-121     ИЭиМ
2  Сидоров   С-121    ИАСиЭ
```

Рис. 4.27. Соединение объектов «DataFrame» по типу «многие-к-одному»

Ключевые столбцы можно задать принудительно с помощью ключевого поля «on», в котором указывается название или список названий столбцов (применяется в том случае, если ключевые столбцы в двух наборах данных имеют одинаковые названия). Пример данной операции приведен на рис. 4.28.

```
В [60]: import pandas as pd
df1 = pd.DataFrame({'студент': ['Иванов', 'Петров', 'Сидоров'],
'группа': ['БИ-120', 'БИ-121', 'С-121']})
df2 = pd.DataFrame({'группа': ['С-121', 'БИ-121', 'БИ-120'],
'институт': ['ИАСиЭ', 'ИЭиМ', 'ИЭиМ']})
print(df1); print(df2); print(pd.merge(df1, df2, on ='группа' ))

   студент  группа
0   Иванов  БИ-120
1   Петров  БИ-121
2  Сидоров   С-121
   группа институт
0   С-121    ИАСиЭ
1  БИ-121     ИЭиМ
2  БИ-120     ИЭиМ
   студент  группа институт
0   Иванов  БИ-120     ИЭиМ
1   Петров  БИ-121     ИЭиМ
2  Сидоров   С-121    ИАСиЭ
```

Рис. 4.28. Соединение объектов «DataFrame» по типу «многие-к-одному» с принудительным заданием ключевых столбцов, имеющих одинаковые имена

При слиянии двух наборов данных с различными именами столбцов необходимо использовать ключевые слова «left_on» и

«right_on» для указания названий двух ключевых столбцов, по которым будет осуществляться слияние.

Пример использования ключевых слов «left_on» и «right_on» при слиянии двух наборов данных с различными именами столбцов приведен на рис. 4.29.

```
В [65]: import pandas as pd
        df1 = pd.DataFrame({'студент': ['Иванов', 'Петров', 'Сидоров'],
        'st_group': ['БИ-120', 'БИ-121', 'С-121']})
        df2 = pd.DataFrame({'group': ['С-121', 'БИ-121', 'БИ-120'],
        'институт': ['ИАСиЭ', 'ИЭиМ', 'ИЭиМ']})
        print(df1); print(df2);
        print(pd.merge(df1, df2, left_on ='st_group', right_on ='group'))

          студент st_group
        0   Иванов   БИ-120
        1   Петров   БИ-121
        2  Сидоров    С-121
           group институт
        0  С-121    ИАСиЭ
        1  БИ-121    ИЭиМ
        2  БИ-120    ИЭиМ
          студент st_group   group институт
        0   Иванов   БИ-120  БИ-120     ИЭиМ
        1   Петров   БИ-121  БИ-121     ИЭиМ
        2  Сидоров    С-121   С-121    ИАСиЭ
```

Рис. 4.29. Соединение объектов «DataFrame» по типу «многие-к-одному» с принудительным заданием ключевых столбцов, имеющих различающиеся имена

Функции «merge» за счет широкого набора опций позволяет обеспечить широкий диапазон возможных поведений при соединении двух наборов данных (слияние по индексу, осуществление операций над множествами для соединений: внутреннее соединение, внешнее соединение, левое соединение, правое соединение и многие другие операции).

4.3.5. Группировка, агрегирование, фильтрация, преобразование данных и формирование сводных таблиц

Группировка – это распределение множества строк исследуемого набора данных по группам в соответствии с существенным для данной группы признаком.

Метод группировки позволяет обеспечивать первичное обобщение данных, представление их в более упорядоченном виде.

Благодаря группировке можно соотнести сводные показатели по набору данных в целом со сводными показателями по группам. Появляется возможность сравнивать, анализировать причины различий между группами, изучать взаимосвязи между признаками. Группировка формирует основу для последующей сводки и анализа данных [2].

Признаки, по которым проводится группировка, называют группировочными признаками. Если для построения группировки используется только один признак, то такую группировку называются простой, если группировка проводится по нескольким признакам, ее называют сложной.

Сложная группировка бывает или комбинационная (группы, выделенные по одному признаку, затем выделяются в подгруппы по другому признаку, которые, в свою очередь, могут выделяться по следующему другому признаку), или многомерная (осуществляется по комплексу признаков одновременно).

Для реализации группировки данных используется метод «groupby» объекта «DataFrame». В качестве параметра данному методу передаются имена группировочных признаков (столбцов набора данных), в качестве результата данный метод возвращает объект «DataFrameGroupBy» – специальное представление объекта «DataFrame», готовое к группировке, но не выполняющее никаких фактических вычислений до этапа применения операций агрегирования.

Агрегирование – процесс преобразования данных с высокой степенью детализации к более обобщенному представлению. Список агрегирующих функций объекта «DataFrameGroupBy» приведен в табл. 4.1.

Пример группировки данных с применением операции агрегирования приведен на рис. 4.30.

Агрегирование может быть проведено также с использованием функции «aggregate», в качестве аргумента в которую может быть передан словарь данных, связывающих имена столбцов с операциями, которые должны быть применены к этим столбцам. Пример агрегирования с использованием функции «aggregate» приведен на рис. 4.31.

Таблица 4.1

Агрегирующие функции объекта «DataFrameGroupBy»

Функция	Описание
count()	Общее количество элементов
first().last()	Первый и последний элементы
mean(), median()	Среднее значение и медиана
min(), max()	Минимум и максимум
std(),var()	Стандартное отклонение и дисперсия
mad()	Среднее абсолютное отклонение
prod()	Произведение всех элементов
sum()	Сумма всех элементов

```
В [19]: import pandas as pd
        df1 = pd.DataFrame({
        'студент': ['Иванов', 'Петров', 'Сидоров', 'Алексеева', 'Дмитриева'],
        'группа': ['БИ-120', 'БИ-121', 'С-121', 'БИ-121', 'С-121'],
        'стипендия': [5000, 3000, 2000, 10000, 2500]
        })
        print("Исходный набор данных:\n",df1)
        print("Сумма стипендий по группам:\n",df1.groupby(['группа']).sum())

        Исходный набор данных:
              студент  группа  стипендия
        0     Иванов  БИ-120       5000
        1     Петров  БИ-121       3000
        2    Сидоров   С-121       2000
        3  Алексеева  БИ-121      10000
        4  Дмитриева   С-121       2500
        Сумма стипендий по группам:
                 стипендия
        группа
        БИ-120        5000
        БИ-121       13000
        С-121         4500
```

Рис. 4.30. Группировка данных с применением агрегирующей функции «sum»

Операция фильтрации дает возможность исключать из результирующего набора данные в зависимости от характеристик группы.

Для осуществления фильтрации используется функция «filter» объекта «DataFrameGroupBy», которая возвращает булево значение, определяющее, прошла ли группа фильтрацию.

Пример группировки данных с применением операции агрегирования и фильтрации приведен на рис. 4.32.

```
В [22]: import pandas as pd
df1 = pd.DataFrame({
'студент': ['Иванов', 'Петров', 'Сидоров', 'Алексеева', 'Дмитриева'],
'группа': ['БИ-120', 'БИ-121', 'С-121', 'БИ-121', 'С-121'],
'стипендия': [5000, 3000, 2000, 10000, 2500],
'средний_балл': [95, 89, 65, 99, 73]
})
print("Исходный набор данных:\n",df1)
print("Группировка и агрегирование:\n",
      df1.groupby(['группа']).aggregate(
      {'стипендия': 'sum','средний_балл': 'max'}))
```

```
Исходный набор данных:
      студент  группа  стипендия  средний_балл
0     Иванов  БИ-120       5000            95
1     Петров  БИ-121       3000            89
2    Сидоров   С-121       2000            65
3  Алексеева  БИ-121      10000            99
4  Дмитриева   С-121       2500            73
Группировка и агрегирование:
         стипендия  средний_балл
группа
БИ-120       5000            95
БИ-121      13000            99
С-121        4500            73
```

Рис. 4.31. Группировка данных с применением функции «aggregate»

```
В [78]: import pandas as pd
df1 = pd.DataFrame({
'студент': ['Иванов', 'Петров', 'Сидоров', 'Алексеева', 'Дмитриева'],
'группа': ['БИ-120', 'БИ-121', 'С-121', 'БИ-121', 'С-121'],
'стипендия': [5000, 3000, 2000, 10000, 2500]
})
grouped_filter = df1.groupby('группа')
grouped_filter=grouped.filter(lambda x: x['стипендия'].max() >4900)
print ("Результирующий набор данных:\n",grouped_filter.groupby('группа').sum())
```

```
Результирующий набор данных:
         стипендия
группа
БИ-120       5000
БИ-121      13000
```

Рис. 4.32. Группировка данных с применением функции агрегирования и функции фильтрации

Операция преобразования предполагает возврат полного набора данных, преобразованного ради дальнейшей его перекомпоновки. Операция преобразования осуществляется с помощью функции «transform» объекта «DataFrameGroupBy».

Пример использования операции преобразования для центрирования данных путем вычитания среднего значения по группам приведен на рис. 4.33.

```
В [81]: import pandas as pd
df1 = pd.DataFrame({
'студент': ['Иванов', 'Петров', 'Сидоров', 'Алексеева', 'Дмитриева'],
'группа': ['БИ-120', 'БИ-121', 'С-121', 'БИ-121', 'С-121'],
'стипендия': [5000, 3000, 2000, 10000, 2500]
})
df1.groupby('группа').transform(lambda x: x - x.mean())
```

Out[81]:

	стипендия
0	0
1	-3500
2	-250
3	3500
4	250

Рис. 4.33. Преобразование для центрирования данных путем вычитания среднего значения по группам

Сводная таблица – операция, предполагающая группировку записей в двумерную таблицу, обеспечивающую многомерное представление данных.

Для создания сводных таблиц используется функция «pivot_table» объекта «DataFrame», основными параметрами которой являются:

values – перечень столбцов для агрегирования;

index – перечень ключей для группировки по индексу сводной таблицы;

columns – перечень ключей для группировки по столбцу сводной таблицы;

aggfunc – список функций для агрегирования данных;

fill_value – значение, на которое нужно заменить отсутствующие значения в итоговой сводной таблице после агрегирования;

dropna – значение, логического типа, указывающее на необходимость включения в сводную таблицу отсутствующих значений.

Пример создания сводной таблицы (зависимость размера стипендии от среднего балла по учебным группам) приведен на рис. 4.34.

```
В [90]: import numpy as np
        import pandas as pd
        df1 = pd.DataFrame({
        'студент': ['Иванов', 'Петров', 'Сидоров', 'Алексеева', 'Дмитриева'],
        'группа': ['БИ-120', 'БИ-121', 'С-121', 'БИ-121', 'С-121'],
        'стипендия': [5000, 3000, 5000, 5000, 3000],
        'средний_балл': [95, 89, 97, 93, 85]})
        print("Исходный набор данных:\n",df1)
        print("Сводная таблица:\n",
              pd.pivot_table(df1, values=['средний_балл'], index=['группа'],
                             columns=['стипендия'], aggfunc=np.mean))
```

```
Исходный набор данных:
      студент  группа  стипендия  средний_балл
0     Иванов  БИ-120       5000            95
1     Петров  БИ-121       3000            89
2    Сидоров   С-121       5000            97
3  Алексеева  БИ-121       5000            93
4  Дмитриева   С-121       3000            85
Сводная таблица:
           средний_балл
стипендия         3000  5000
группа
БИ-120             NaN  95.0
БИ-121            89.0  93.0
С-121             85.0  97.0
```

Рис. 4.34. Создание сводной таблицы

Вопросы для обсуждения

1. Роль библиотеки pandas в интеллектуальном анализе данных
2. Сходства и отличия библиотеки pandas от NumPy
3. Общие подходы к организации вычислений с использованием библиотеки pandas
4. Способы обработки отсутствующих данных с применением библиотеки pandas
5. Способы обнаружения и фильтрации выбросов с применением библиотеки pandas
6. Дискретизация непрерывных данных с применением библиотеки pandas
7. Иерархическая индексация с применением библиотеки pandas
8. Работа с временными рядами с применением библиотеки pandas
9. Увеличение производительности библиотеки pandas
10. Разбор веб-страниц с помощью библиотеки pandas

Практические задания

Задание 4.1.

Выполните следующие действия:

1. создайте объект Series №1, содержащий фамилии пяти студентов (тип данных созданного объекта – str, создание объекта осуществить с использованием конструктора, в качестве меток использовать номер студенческого билета, укажите имя объекта);

2. создайте объект Series №2, содержащий имена пяти студентов (тип данных созданного объекта – str, создание объекта осуществить с использованием конструктора, в качестве меток использовать номер студенческого билета, укажите имя объекта);

3. создайте объект Series №3, содержащий среднюю оценку успеваемости пяти студентов (тип данных созданного объекта – float, в качестве меток использовать номер студенческого билета);

4. создайте объект Series №4, который содержит сведения о фамилии и имени студентов;

5. осуществите следующие операции надо объектом Series №3 (тип данных результирующих объектов – float): определите максимальную оценку, определите минимальную оценку, определите среднюю оценку.

Задание 4.2

Выполните следующие действия:

1. создайте объект DataFrame №1, содержащий следующую информацию о пяти студентах: отчество, дата рождения, учебная группа (создание объекта осуществить с использованием конструктора, в качестве меток использовать номер студенческого билета);

2. создайте объект DataFrame №2 на основе объединения нескольких объектов Series: Series №1, Series №2, Series №3;

3. создайте объект DataFrame №3 на основе объединения нескольких объектов DataFrame: DataFrame №1, DataFrame №2;

4. осуществите вывод на экран для объекта DataFrame №3 списка меток строк, списка меток столбцов, массива данных (по отдельности);

5. осуществите вывод на экран для объекта DataFrame №3 следующих элементов: а) первые две строки; б) строки со второй по чет-

вертую; в) столбец, содержащий информацию о фамилии студента; г) столбец, содержащий информацию о фамилии студента; д) фамилию студента по заданному номеру студенческого билета; е) имя студента по заданному порядковому номеру.

Задание 4.3

Выполните следующие действия:

1. На основании официальных статистических данных о курсе акции, которые были использованы при решении задания 3.2 (предварительно необходимо заменить информацию о ценах для нескольких строк на следующие символы «-», « », «н/д»: всего заменить не более 3-4 значений, при этом в одной из строк должно быть не менее 2 таких символов), используя функцию pandas.read_csv, создайте набор данных, в котором решите следующие проблемы:

а) проблему индексирования: определите имена столбцов либо по данным, содержащимся в первой строке исходного файла (аргумент header) или задайте имена столбцов самостоятельно (аргумент names), а также исключите из результирующего набора данных столбец с данными об изменении цен (аргумент usecols);

б) проблему выведения типа и преобразование данных: определите тип данных для значений, содержащихся в столбцах (для столбца с данными о датах установить тип данных datetime64, для цен и объёмов продаж – float64), при необходимости конвертируя значения (аргументы dtype и converters совместно с пунктом б); определите последовательность значений, интерпретируемых как маркеры отсутствующих данных (аргумент na_values);

в) проблему «грязных данных»: определите символ, который нужно распознать как десятичную точку (аргумент decimal).

2. На основании набора данных, полученного в п.1, осуществите:

а) сортировку значений по столбцу с данными об объёмах продаж по убыванию (sort_values ());

б) ранжирование значений по столбцу с данными о цене акции на момент открытия (rank ());

в) получение сводной статистики по динамике цен на акцию (describe());

г) вычисление корреляции и ковариации по цене закрытия и объёму продаж (corr() и cov());

д) обработку отсутствующих данных:

- исключите из набора данных строки, в которых отсутствует 2 и более значений (dropna() с параметром how);

- заполните пустые значения в наборе данных средним арифметическим значением (для ценовых данных – по данным строк, по данным об объёмах продаж – по данным столбца) (dropna() с параметром values);

2. На основании официальных статистических данных о курсе акции, которые были использованы при решении задания 3.2, а также данных о курсе акции за другой период (например, за прошлый год) осуществите:

а) соединение двух наборов данных в один используя конкатенацию (concat());

б) соединение двух наборов данных в один используя реляционный подход (merge()).

3. На основании официальных статистических данных о курсе акций, которые были использованы при решении задания 3.2, а также данных о курсе акции за перекрывающий период (например, за текущий год) осуществите:

а) соединение двух наборов данных в один, используя конкатенацию, не допуская дублирования данных (concat());

б) соединение двух наборов данных в один используя реляционный подход, не допуская дублирования данных (merge()).

4. На основании официальных статистических данных о курсе акции, которые были использованы при решении задания 3.2, а также данных о курсе любой другой акции за аналогичный период осуществите:

а) соединение двух наборов данных в один, используя конкатенацию (concat());

б) соединение двух наборов данных в один используя реляционный подход (merge ()).

Библиографический список

1. IO tools (text, CSV, HDF5, …) [Электронный ресурс] – Режим доступа: https://pandas.pydata.org/docs/user_guide/io.html

2. Васнев С.А. Статистика: Учебное пособие [Электронный ресурс] – Режим доступа: http://www.hi-edu.ru/e-books/xbook096/01/part-003.htm

3. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

4. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

5. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

6. Маккинни, У. Маккинли, У. Python и анализ данных / Уэс Мак-кинли ; пер. с англ. А.А. Слинкина. - Москва : ДМК Пресс, 2015. - 482 с. - ISBN 978-5-97060-315-4. - Текст : электронный. - URL: https://znanium.com/catalog/product/1027796

Глава 5. MATPLOTLIB: ВИЗУАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ ДАННЫХ

В данной главе рассматриваются следующие вопросы:

1. Общие сведения о библиотеке Matplotlib

2. Построение двумерных графиков и диаграмм

3. Построение трехмерных графиков

5.1. Общие сведения о библиотеке Matplotlib

Matplotlib – программная библиотека на языке Python, ориентированная на визуализацию данных двумерной (2D) графикой и трехмерной (3D) графикой.

Условно структуру Matplotlib можно представить как совокупность компонентов образующих строгую иерархическую структуру (уровни): Backend, Artist и Scripting.

Backend Layer – верхний уровень, содержащий три абстрактных класса, реализация которых осуществляется на нижележащих уровнях:

1) «FigureCanvas»: определяет область, в которой рисуется фигура;

2) «Renderer»: определяет инструментарий для рисования в указанной области «FigureCanvas»;

3) «Event»: определяет реакцию пользователя на действия пользователя.

Artist Layer – средний уровень, который содержит абстрактный базовый класс для объектов, которые отображаются в FigureCanvas. Как правило, все видимые элементы графики являются подклассами «Artist».

Структура класса «Artist» приведена на рис. 5.1.

Ключевыми интерактивным элементом класса «Artist» является «Figure» (Рис.), интерактивный объект самого верхнего уровня, на котором располагаются другие элементы, такие как:

- области рисования («Axes»),

- элементы рисунка (координатные оси («Axis»), текст («Text»), плоские линии («Line2D») и другие элементы).

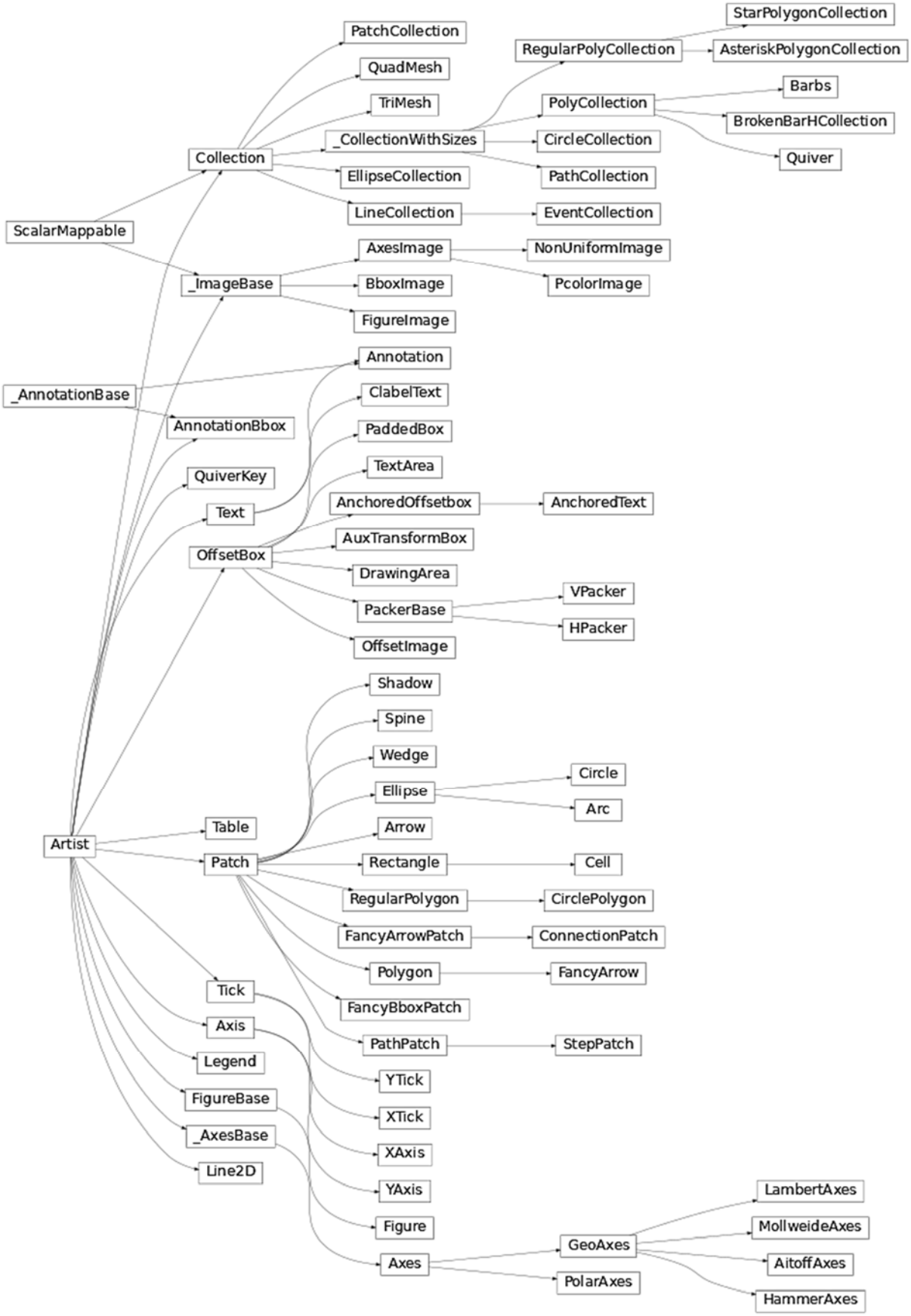


Рис. 5.1. Структура класса Artist [2]

Scripting Layer – нижний уровень, который автоматизирует и упрощает процесс сборки графики. В состав пакета Matplotlib входит класс, предназначенный для построения простых интерактивных графиков – «pyplot», который реализует абстрактный класс «Artist». Данный пакет принято импортировать, используя в качестве его псевдонима «plt».

Создание графики в «pyplot» начинается с создания контейнеров, которые будут включать в себя все остальные интерактивные элементы:

1) «figure» (Рис.);

2) «axes» (область рисования).

Объект «figure» создается с помощью конструктора одноименного класса:

pyplot.figure (num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs)

Основные аргументы:

num (тип данных – integer, string или Figure, необязательный аргумент) – устанавливает уникальный идентификатор фигуры (значение по умолчанию – None; если фигура с указанным идентификатором уже существует, то метод возвращает ссылку на существующий объект Figure);

figsize (тип данных – (float, float)) – устанавливает ширину и высоту в дюймах. (значение по умолчанию – None, соответствует [6.4, 4.8]);

facecolor (тип данных – color) – устанавливает цвет фона. (значение по умолчанию – None, соответствует ‘white’);

edgecolor (тип данных – color) – устанавливает цвет рамки (значение по умолчанию – None, соответствует ‘white’).

Добавление области рисования (axes) к существующей фигуре осуществляется с помощью метода «add_subplot», в который в качестве аргумента передается позиция области рисования (в формате: строка, колонка, порядковый номер).

Пример добавления области рисования (axes) к существующей фигуре приведен на рис. 5.2.

```
B [61]: import matplotlib.pyplot as plt
        fig = plt.figure(facecolor='ivory', figsize=[5.8,1.0] )
        ax1 = fig.add_subplot(2,2,1)
        ax4 = fig.add_subplot(224)
```

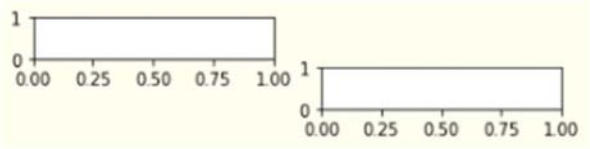


Рис. 5.2. Добавления области рисования

Создать объект «figure» с несколькими областями единовременно можно с помощью метода «pyplot.subplots» (в качестве аргументов среди прочих задается размер сетки, т.е. количество строк и столбцов (по умолчанию сетка имеет одну строку и один столбец)).

Пример создания объекта «figure» с несколькими областями единовременно приведен на рис. 5.3.

```
B [65]: import matplotlib.pyplot as plt
        fig, axs = plt.subplots(1, 2, facecolor='ivory', figsize=[5.8,3.0])
```

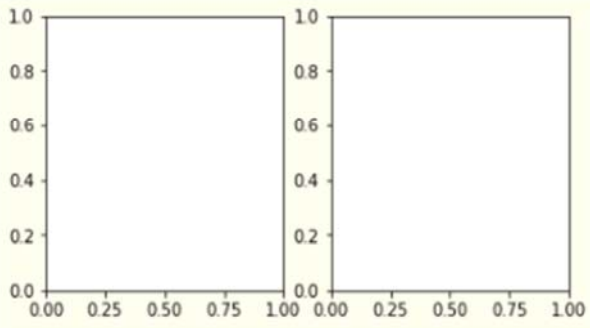


Рис. 5.3. Создания объекта «figure» единовременно с несколькими областями

Следующим шагом является построение графиков и диаграмм в заданной области рисования. Рассмотрим возможности библиотеки Matplotlib при построении двумерных и трехмерных графиков и диаграмм.

5.2. Построение двумерных графиков и диаграмм

Библиотека Matplotlib поддерживает большое количество видов двумерных графиков и диаграмм, неполный список которых выглядит следующим образом:

1) линейные графики;
2) стержневые диаграммы;
3) диаграммы разброса;
4) столбчатые диаграммы и гистограммы;
5) круговые диаграммы;
6) контурные графики;
7) поля градиентов;
8) спектральные диаграммы

5.2.1. Линейные графики

Для построения линейных графиков используется метод «pyplot.plot», описание которого выглядит следующим образом:

pyplot.plot (*args, scalex=True, scaley=True, data=None, **kwargs

Основные аргументы:

*args – список параметров, определяющих координаты точек графика (x, y), а также внешний вид маркеров и линий (fmt):

x, y (тип данных – массив или скалярная величина) – устанавливает горизонтальные и вертикальные координаты точек данных (значения x являются необязательными и по умолчанию равны диапазону (len (y));

fmt (тип данных – строка, необязательный аргумент) – определяет свойства линий и маркеров графика (представляет собой специальную строку формата '[marker][line][color]', с помощью которой кодируется, соответственно, внешний вид маркера, внешний вид линии и их цвет, например: «^r:»: треугольные маркеры красного цвета, соединенные пунктирной линией)[3];

scalex, scaley (тип данных – bool, необязательный аргумент) – определяет адаптивность области просмотра данных к их значениям, соответственно по осям x и y (значение по умолчанию - True).

**kwargs – группа аргументов, определяющих свойства элементов графика (marker, linestyle, linewidth, markersize и т.д.).

Для настройки цвета используется ключевое слово «color», которому ставится в соответствие строковый аргумент, задающий практически любой цвет.

Если цвет не задан, библиотека Matplotlib будет автоматически перебирать по циклу набор цветов по умолчанию при наличии на графике нескольких линий. Стиль линий можно настраивать и с помощью ключевого слова «linestyle», толщину – с помощью «linewidth».

Пример построения линейного графика приведен на рис. 5.4.

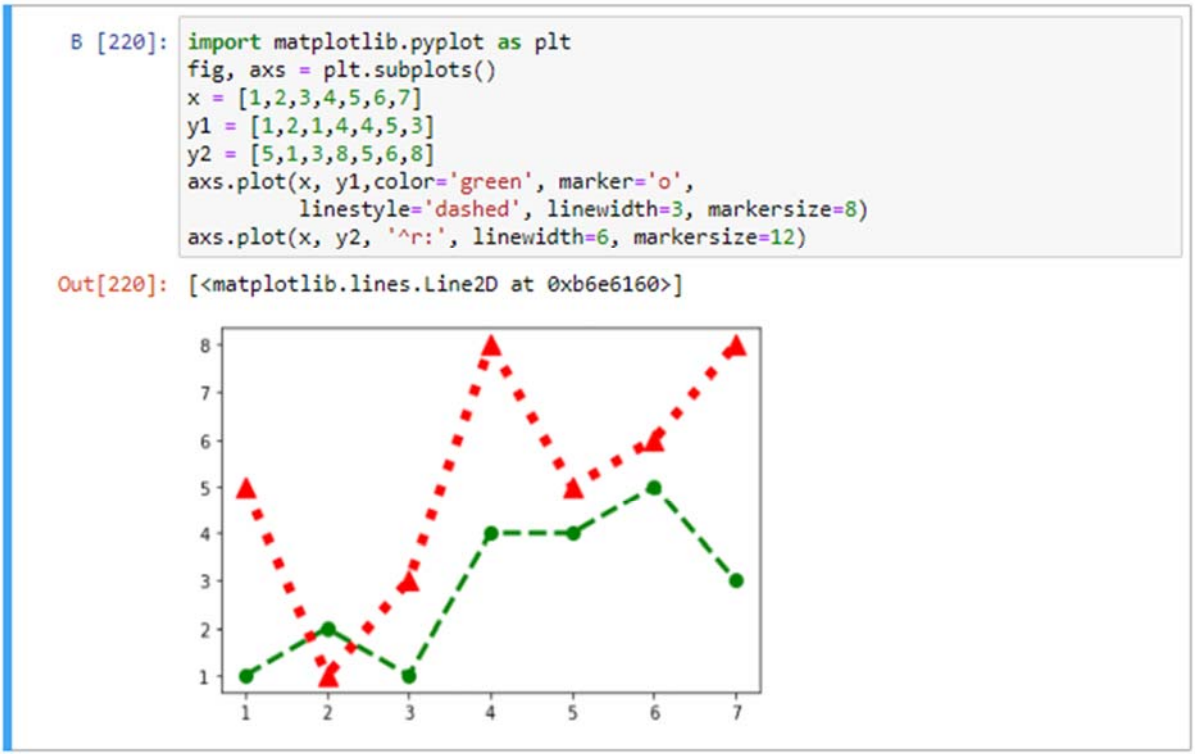


Рис. 5.4. Построение линейного графика

Графики могут быть маркированы, т.е. на них нанесены названия, метки осей координат и простые легенды.

Название графика устанавливается с помощью метода «title», метки осей – с помощью методов «xlabel» и «ylabel», соответственно, для оси x и y.

В случае отображения нескольких линий в одной области удобно создать легенду для графика, на которой бы отмечался каждый тип линии. Для создания легенды используется метод «plt.legend».

Пример применения маркировки линейных графиков приведен на рис. 5.5.

```
B [251]: import matplotlib.pyplot as plt
         fig, axs = plt.subplots()
         x = np.linspace(0, 10, 1000)
         axs.plot(x, np.sin(x), 'b-', label='F(x)=sin(x)')
         axs.plot(x, np.cos(x), 'r:', label='F(x)=cos(x)')
         plt.title ('Линейный график')
         plt.xlabel('x')
         plt.ylabel('F(x)')
         plt.grid()
         plt.legend()

Out[251]: <matplotlib.legend.Legend at 0xe532ca0>
```

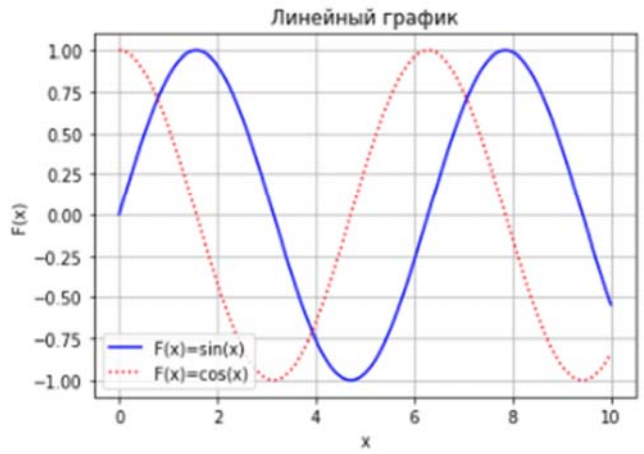


Рис. 5.5. Маркировка линейных графиков

5.2.2. Стержневые диаграммы

На стержневой диаграмме линии рисуются перпендикулярно базовой линии, начиная с оси координат и заканчивая маркерами данных. Для вертикальных стержневых графиков (по умолчанию) местоположениями являются положения x, а заголовками - значения y. Для горизонтальных стволовых диаграмм - это позиции y, а заголовки - x значения.

Для построения линейных графиков может использоваться метод «pyplot.stem», описание которого выглядит следующим образом:

pyplot.stem (*args, linefmt=None, markerfmt=None, basefmt=None, bottom=0, label=None, use_line_collection=True, orientation='vertical', data=None)

Пример построения стержневой диаграммы приведен на рис. 5.6.

```
B [426]: import matplotlib.pyplot as plt
         import numpy as np
         x = np.linspace(0, 10, 50)
         y = np.cos(x)
         plt.stem(x, y)

Out[426]: <StemContainer object of 3 artists>
```

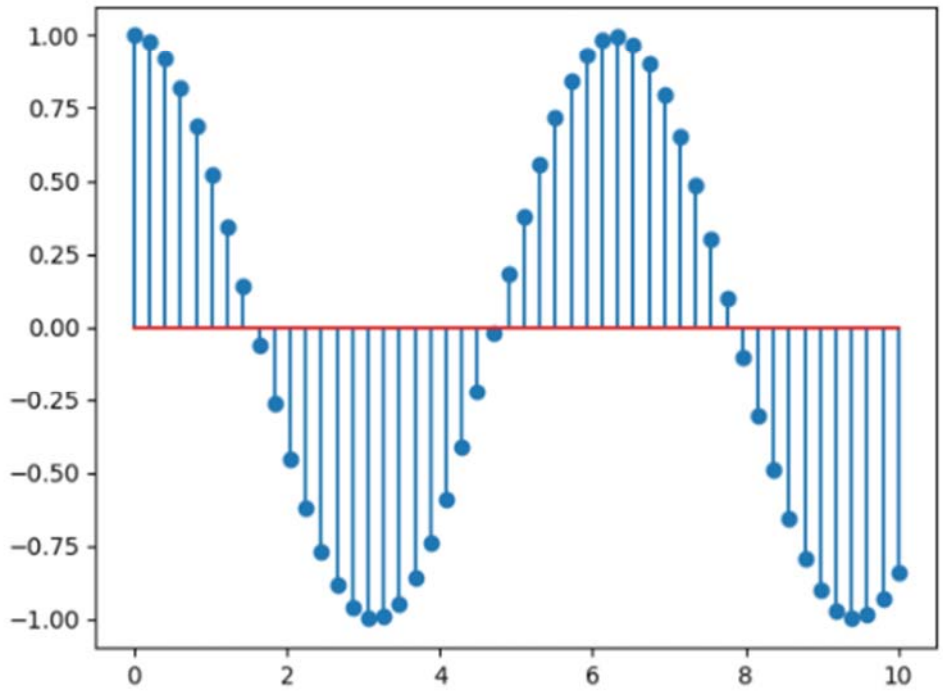


Рис. 5.6. Построение стержневой диаграммы

5.2.3. Диаграммы разброса

Диаграмма разброса – это точечная диаграмма, получаемая путем нанесения на график экспериментальных данных в виде точек. Координаты точек на графике соответствуют значениям показателя качества (y) и влияющего на него фактора (x).

Диаграмма разброса позволяет сформулировать и проверить гипотезу о наличии или отсутствии корреляционной связи между двумя признаками (характеристика качества и влияющий на нее фактор).

Пример построения диаграммы разброса приведен на рис. 5.7.

```
В [269]: import numpy as np
         import matplotlib.pyplot as plt

         np.random.seed(19680801)
         N = 50
         x = np.random.rand(N)
         y = np.random.rand(N)
         colors = np.random.rand(N)
         area = (30 * np.random.rand(N))**2
         plt.scatter(x, y, s=area, c=colors, alpha=0.5)
         plt.title ('Диаграмма разброса')
         plt.xlabel('x')
         plt.ylabel('F(x)')
         plt.grid()
```

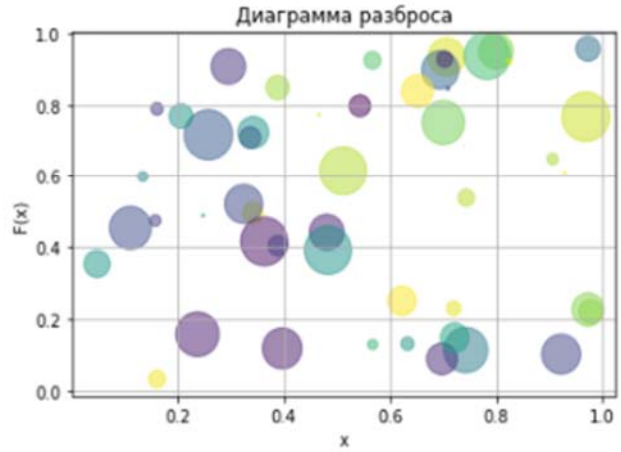


Рис. 5.7. Построение диаграммы разброса

Для построения диаграмм разброса используется метод «pyplot.scatter», описание которого выглядит следующим образом:

pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, *, edgecolors=None, plotnonfinite=False, data=None, **kwargs)

Основные аргументы:

x, y (тип данных – массив или скалярная величина) – устанавливает горизонтальные и вертикальные координаты точек данных;

s (тип данных – массив или скалярная величина) – устанавливает размер маркера;

c (тип данных – массив, список цветовых решений) – устанавливает цвет маркера;

alpha (тип данных – float) – устанавливает степень прозрачности маркера (принимает значение: 0 (прозрачный) and 1 (непрозрачный).

5.2.4. Столбчатые диаграммы (гистограммы)

Столбчатая диаграмма (гистограмма) – диаграмма, представленная прямоугольными зонами (столбцами), высоты или длины которых пропорциональны величинам, которые они отображают. Прямоугольные зоны могут быть расположены вертикально или горизонтально. Столбчатая диаграмма отображает сравнение нескольких дискретных категорий.

Для построения столбчатых диаграмм с вертикальным расположением прямоугольных зон используется метод «pyplot.hist», описание которого выглядит следующим образом:

pyplot.hist (x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)

Основные аргументы:

x (тип данных – массив или последовательность массивов) – определяет входные данные для построения диаграммы;

bins (тип данных – integer, или string) – определяет количество интервалов одинаковой ширины в диапазоне, по которым строятся столбцы гистограммы (значение по умолчанию – None, соответствует 10) или определяет стратегию определения указанных интервалов (доступны следующие варианты: 'auto', 'fd', 'doane', 'scott', 'stone', 'rice', 'sturges', 'sqrt');

histtype (тип данных – string) – определяет тип гистограммы (доступны следующие варианты: 'bar', 'barstacked', 'step', 'stepfilled', значение по умолчанию – 'bar'): 'bar' – это традиционная гистограмма в виде столбиков, располагаемых рядом друг с другом; 'barstacked' – это гистограмма в виде столбцов, которые накладываются друг на друга; 'step' – это гистограмма в виде линейного графика; 'stepfilled' – это гистограмма в виде линейного графика с заполнением;

rwidth (тип данных – float) – определяет относительную ширину столбиков (значение по умолчанию – None, предполагает автоматический подбор ширины 10);

alpha (тип данных – float) – устанавливает степень прозрачности столбиков (принимает значение: 0 (прозрачный) and 1 (непрозрачный).

Пример построения вертикальной гистограммы с использованием метода «pyplot.hist» приведен на рис. 5.8.

```
B [336]: import numpy as np
         import matplotlib.pyplot as plt

         np.random.seed(19680801)
         mu_x = 200
         sigma_x = 25
         x = np.random.normal(mu_x, sigma_x, size=1000)
         fig, axs = plt.subplots()
         axs.hist(x, 20, density=True, histtype='barstacked', rwidth=0.93,
                  color='g', alpha=0.75)
         axs.set_title('Гистограмма')

Out[336]: Text(0.5, 1.0, 'Гистограмма')
```

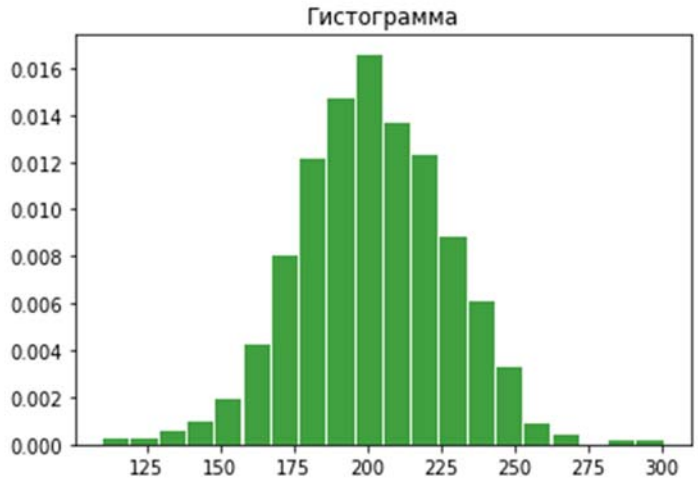


Рис. 5.8. Построение вертикальной гистограммы с использованием метода «pyplot.hist»

Для построения диаграмм с вертикальным расположением прямоугольных зон используется также метод «pyplot.bar», описание которого выглядит следующим образом:

pyplot.bar (x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)

Основные аргументы:

y (тип данных – float или массив) – определяет входные данные для построения диаграммы;

height (тип данных – float или массив) – определяет высоту полос диаграммы;

width (тип данных – float или массив, необязательный параметр) – определяет ширину полос диаграммы (значение по умолчанию width =0.8).

Пример построения вертикальной гистограммы с использованием метода «pyplot.bar» приведен на рис. 5.9.

```
B [12]: import matplotlib.pyplot as plt
        import numpy as np
        np.random.seed(19680801)
        fig, ax = plt.subplots()
        country = ('Индия', 'Китай', 'США', 'Индонезия', 'Бразилия')
        pup = [1487,1424,331,272,213]
        ax.bar(country, pup)
        ax.set_xticklabels(country)
        ax.invert_yaxis()
        ax.set_ylabel('млн. чел.')
        ax.set_title('Население стран мира')
```

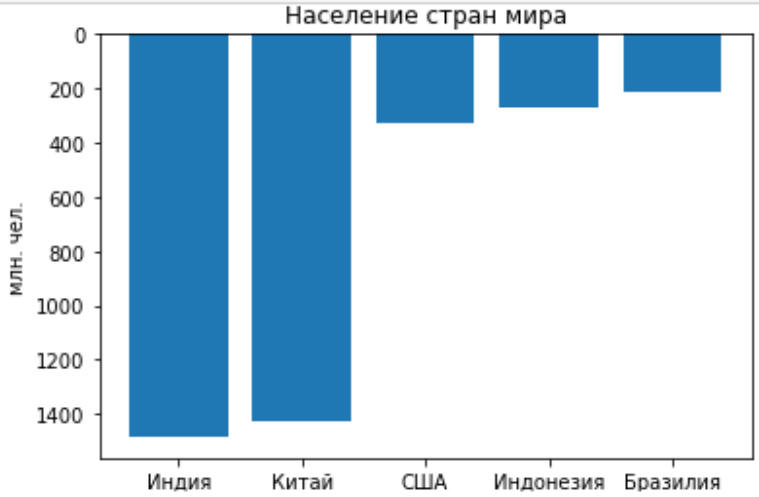


Рис. 5.9. Построение вертикальной гистограммы («pyplot.bar»)

Данный метод также поддерживает возможность построения диаграмм с накоплением. Пример построения вертикальной гистограммы с накоплением приведен на рис. 5.10.

```
В [14]: import matplotlib.pyplot as plt
        import numpy as np
        np.random.seed(19680801)
        fig, ax = plt.subplots()
        country = 'Индия', 'Китай', 'США', 'Индонезия', 'Бразилия'
        pup = [1487,1424,331,272,213]
        pup_plus = [487,424,31,72,13]
        ax.bar(country, pup)
        ax.bar(country, pup_plus)
        ax.invert_yaxis()
        ax.set_ylabel('млн. чел.')
        ax.set_title('Население стран мира')
```

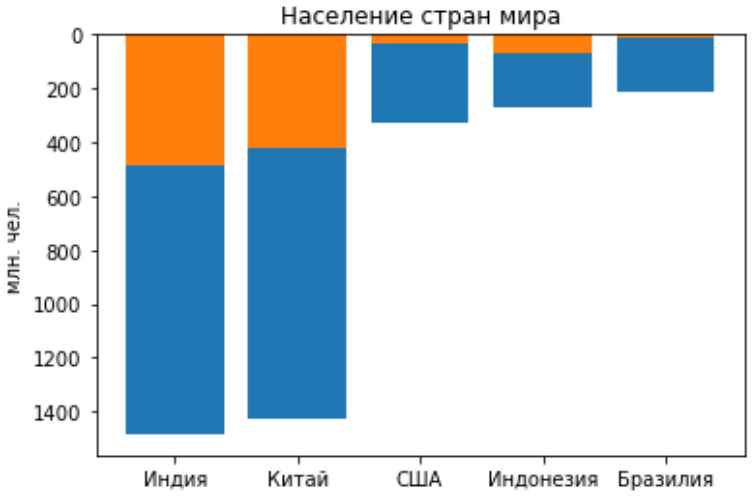


Рис. 5.10. Построение вертикальной гистограммы с накоплением

Для построения диаграмм с горизонтальным расположением прямоугольных зон используется метод «pyplot.barh», описание которого выглядит следующим образом:

pyplot.barh (y, width, height=0.8, left=None, *, align='center', **kwargs)

Основные аргументы:

y (тип данных – float или массив) – определяет входные данные для построения диаграммы;

width (тип данных – float или массив) – определяет ширину полос диаграммы;

height (тип данных – float или массив, необязательный параметр) – определяет высоту полос диаграммы (значение по умолчанию height=0.8).

Данный метод также поддерживает возможность построения диаграмм с накоплением. Пример построения горизонтальной гистограммы приведен на рис. 5.11.

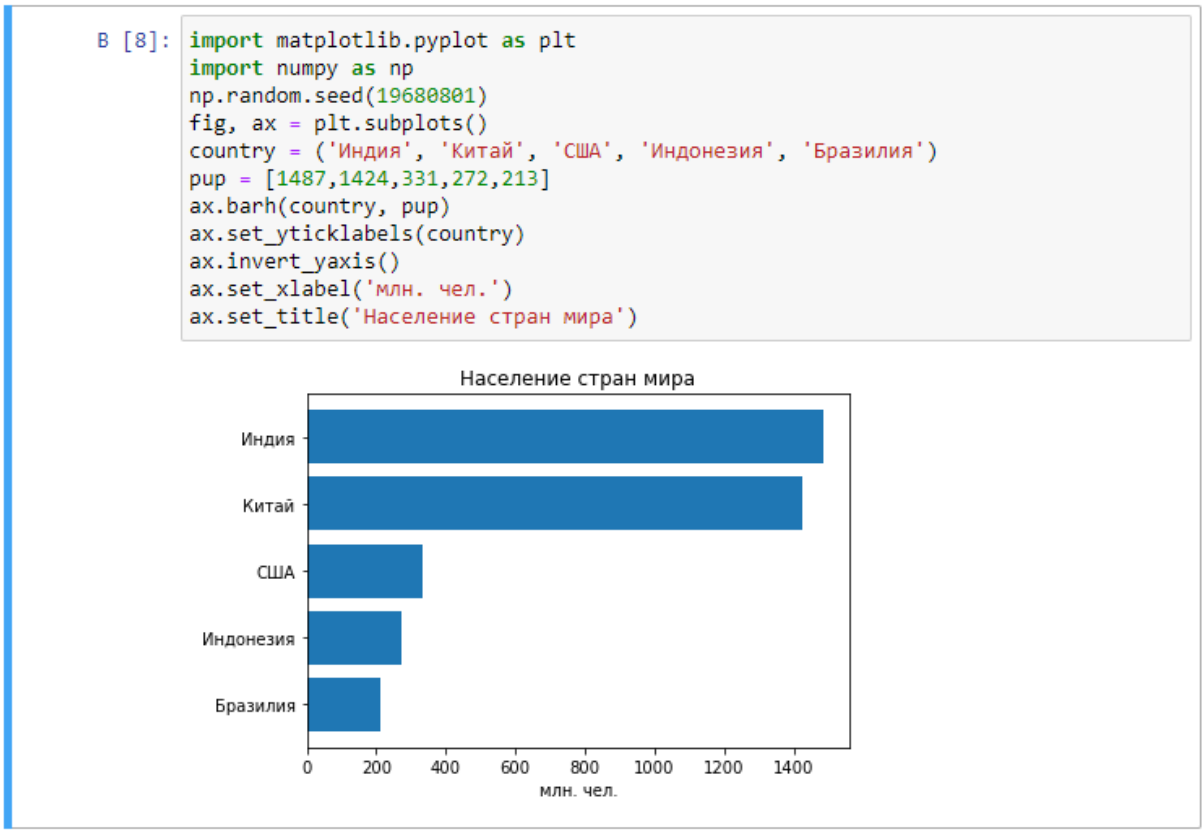


Рис. 5.11. Построение горизонтальной гистограммы

5.2.5. Круговые диаграммы

Круговые диаграммы – это круглые графики, поделенные на секторы, каждый из которых представляет размер какой-либо связанной части данных.

Для построения круглых диаграмм используется метод «pyplot.pie», описание которого выглядит следующим образом:

pyplot.pie (x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1, counterclock=True, wedgeprops=None, textprops=None, center=0, 0, frame=False, rotatelabels=False, *, normalize=None, data=None)

Основные аргументы:

x (тип данных – массив) – определяет входные данные для построения диаграммы;

explode (тип данных – массив) – определяет размер смещения сектора относительно соседних секторов (по умолчанию – None, соответствует отсутствию смещения);

labels (тип данных – список) – определяет метки для каждого сектора (по умолчанию – None, соответствует отсутствию меток);

autopct (тип данных – string) – определяет формат числовых меток для каждого сектора;

startangle (тип данных – float) – определяет угол поворота начала круговой диаграммы против часовой стрелки от оси x.

Пример построения круговой диаграммы приведен на рис. 5.11.

```
В [15]: import matplotlib.pyplot as plt
        labels = ('Индия', 'Китай', 'США', 'Индонезия', 'Бразилия')
        pup = [1487,1424,331,272,213]
        explode = (0.1, 0.1, 0, 0, 0)
        fig1, ax1 = plt.subplots()
        ax1.pie(pup, explode=explode, labels=labels, autopct='%1.1f%%',
                shadow=True, startangle=90)
        ax1.axis('equal')
```

Рис. 5.11. Построение круговой диаграммы

5.3. Построение трехмерных графиков и диаграмм

Входящий в состав пакета Matplotlib класс pyplot предназначен для построения простых интерактивных 2D-графиков и его нельзя использовать для работы с 3D-графикой.

Для указанных целей используют библиотеку mpl_toolkits.mplot3d, предоставляющую некоторые базовые инструменты работы с 3D-графикой. Данная библиотека поставляется с Matplotlib и, не обладая высокой скоростью решения задач и исчерпывающим набором функций, тем не менее, может быть использована для решения множества практических задач.

Использование mpl_toolkits.mplot3d соответствует принципам применения Matplotlib и поэтому построение 3D-графиков является задачей подобной построению 2D-графиков с дополнительным измерением (данную задачу реализуют с помощью объекта Axes3D, который создается с использованием ключевого слова projection = ‘3d’).

Пример 3D-графика с линейным графиком и точечной диаграммой приведен на рис. 5.12.

```
B [24]: import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        import numpy as np
        theta=np.linspace(0, 2*np.pi)
        x = np.cos(theta - np.pi/2)
        y = np.sin(theta - np.pi/2)
        z = theta
        fig, ax = plt.subplots(subplot_kw=dict(projection='3d'))
        ax.scatter(x, y, z)
        ax.plot(x+1, y+1, z+1)
```

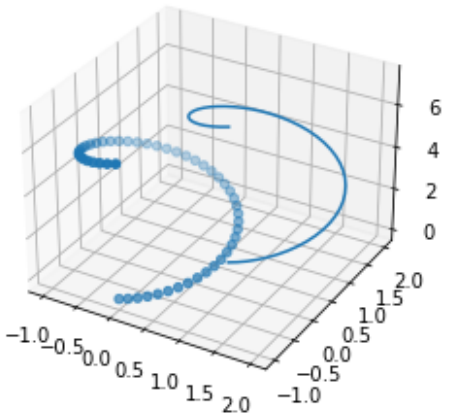


Рис. 5.12. Построения трехмерных графиков и диаграмм

Для построения каркасной и сплошной поверхности используются соотвественно функции plot_wireframe() и plot_surface(), описание которых выглядит следующим образом:

Axes3D.plot_wireframe (self, X, Y, Z, *args, **kwargs)
Axes3D.plot_surface(self, X, Y, Z, *args, norm=None, vmin=None, vmax=None, lightsource=None, **kwargs)

Основные аргументы:

X, Y, Z (тип данных – 2-ух мерный массив) – определяет входные данные для построения диаграммы;

Пример 3D-графика с каркасной и сплошной поверхностью приведен на рис. 5.13.

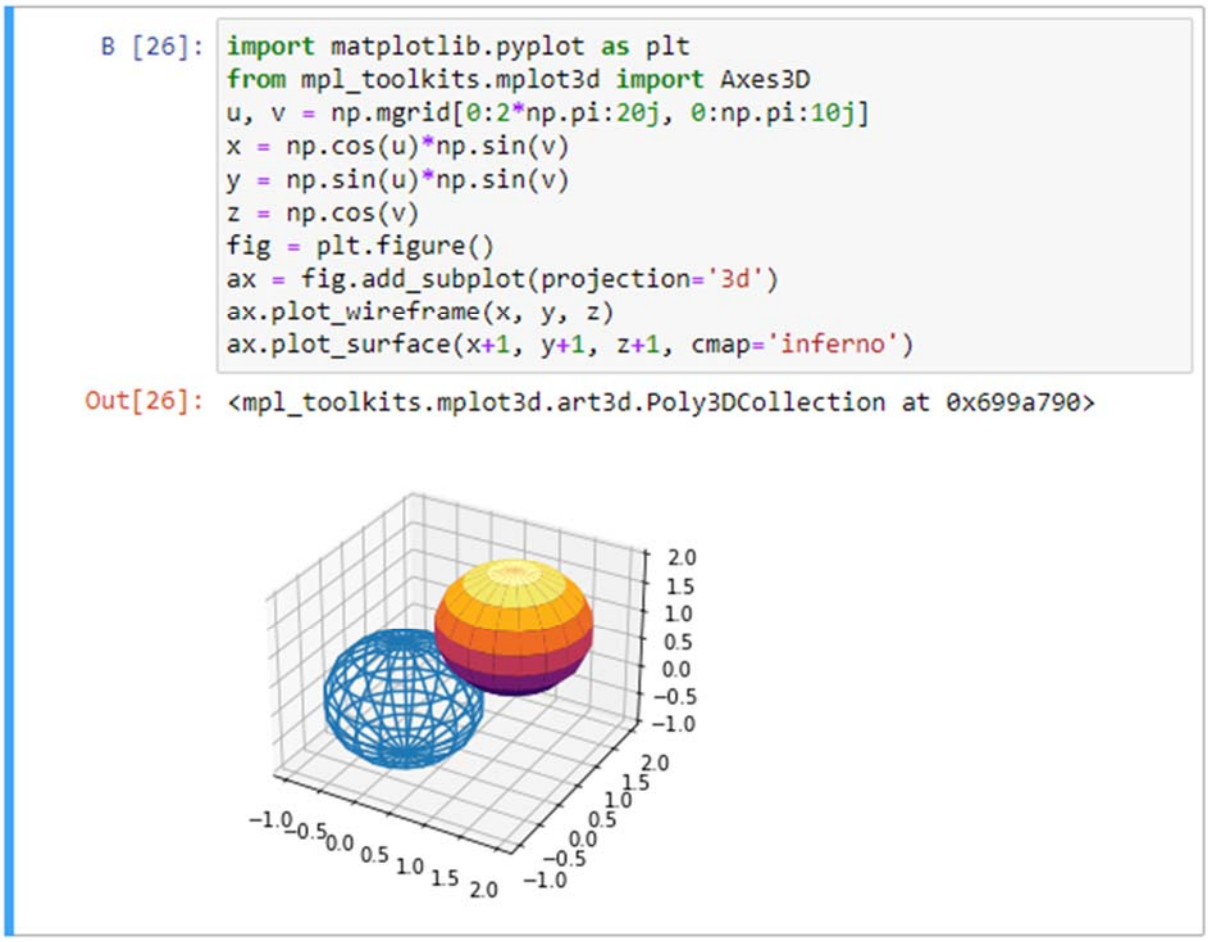


Рис. 5.13. Построение трехмерных графиков с каркасной и сплошной поверхностью

Вопросы для обсуждения

1. Роль библиотеки Matplotlib в интеллектуальном анализе данных
2. Интерфейса библиотеки Matplotlib
3. Визуализация погрешностей с помощью библиотеки Matplotlib
4. Использование пользовательских настроек шкал цветов
5. Использование пользовательских настроек легенд на графиках
6. Использование поясняющих надписей на графиках
7 Картографические проекции
8. Использование прочих библиотек языка Python для визуализации данных (Seaborn, Bokeh, Plotly, Vispy, Vega)
9. Будущее средств визуализации

Практические задания

Задание 5.1

Выполните следующие действия:

а) создайте Рис. (объект «figure») с двумя областями рисования («объект axes»), расположенными друг под другом;

б) на основании официальных статистических данных о курсе акции, которые были использованы при решении задания 3.2 в верхней области постройте с помощью линейных графиков («pyplot.plot») ценовой канал по данным ежедневной максимальной и минимальной цены;

б) на основании официальных статистических данных о курсе акции, которые были использованы при решении задания 3.2 в нижней области отобразите с помощью стержневых графиков («pyplot.stem») динамику ежедневных объёмов торгов.

Задание 5.2

Выполните следующие действия:

а) создайте Рис. (объект «figure») с двумя областями рисования (объект «axes»), расположенными рядом друг с другом;

б) на основании данных, полученных при решении задания 3.1 (п.1 и п.2) в левой области постройте диаграмму разброса («pyplot.scatter»);

в) на основании данных, полученных при решении задания 3.1 (п.3) в правой области постройте гистограмму («pyplot.hist»).

Задание 5.3

Выполните следующие действия:

а) создайте Рис. (объект «figure») с одной областью рисования (объект «axes»);

б) на основании данных о продажах за месяц (по вариантам) постройте круговую диаграмму («pyplot.pie»);

Варианты:

Задание по вариантам:

№	Выручка, млн руб.	№	Выручка, млн руб.
1	яблоки - 1; бананы - 12; груши - 44	11	Яблоки - 11; бананы - 32; груши - 34
2	Яблоки - 2; бананы - 14; груши - 43	12	Яблоки - 12; бананы - 34; груши - 33
3	Яблоки - 3; бананы - 16; груши - 42	13	Яблоки - 13; бананы - 36; груши - 32
4	Яблоки - 4; бананы - 18; груши - 41	14	Яблоки - 14; бананы - 38; груши - 31
5	Яблоки - 5; бананы - 20; груши - 40	15	Яблоки - 15; бананы - 40; груши - 30

Библиографический список

1. Matplotlib: Visualization with Python [Электронный ресурс] – Режим доступа: https://matplotlib.org/

2. Matplotlib.artist [Электронный ресурс] – Режим доступа: https://matplotlib.org/stable/api/artist_api.html

3. Matplotlib.pyplot.plot [Электронный ресурс] – Режим доступа: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

4. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для со-

здания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

5. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

6. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

7. Маккинни, У. Маккинли, У. Python и анализ данных / Уэс Мак-кинли ; пер. с англ. А.А. Слинкина. - Москва : ДМК Пресс, 2015. - 482 с. - ISBN 978-5-97060-315-4. - Текст : электронный. - URL: https://znanium.com/catalog/product/1027796

Глава 6. PYTHON-БИБЛИОТЕКА SCIKIT-LEARN КАК ИНСТРУМЕНТ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ

В данной главе рассматриваются следующие вопросы:

1. Общие сведения о библиотеке Scikit-learn

2. Обзор основных способов загрузки данных

3. Реализация подготовки данных к анализу

4. Моделирование данных и анализ качества моделей

6.1. Общие сведения о библиотеке Scikit-learn

Как было отмечено, в предыдущих главах, пакет Scikit-Learn – один из самых известных и надежных инструментов, позволяющий работать с широким диапазоном алгоритмов машинного обучения.

Рассмотрим применение Python-библиотеки Scikit-learn в контексте основных этапов операционной аналитики: сбор данных, подготовка данных, исследование данных и моделирование данных.

Для ввода данных в Scikit-learn используется одним из самых распространённых в машинном обучении способов: признаковое описание объекта, т.е. вектор, который составлен из значений, соответствующих некоторому набору признаков для данного объекта.

Признак – результат измерения некоторой характеристики объекта. Значения признаков могут быть различного, не обязательно числового, типа.

Множество всех признаков всех объектов предметной области традиционно обозначают через X и называют матрицей объектов-признаков. Строку такой матрицы называют выборкой.

Помимо матрицы объектов-признаков X в задачах машинного обучения приходится иметь дело с набором зависимых от признаков переменных, которые называют метками. Прогнозирование значений меток является целью в большинстве моделей машинного обучения.

Множество всех меток всех объектов предметной области в Scikit-learn традиционно обозначают через **y** и называют матрицей объектов-меток (объектов-целей).

Для хранения данных о признаках в Scikit-learn использует массивы библиотеки из NumPy, объекты DataFrame из библиотеки

Pandas, разреженные матрицы из библиотеки SciPy. Для хранения данных о метках в Scikit-learn - использует массивы библиотеки из NumPy и объекты DataFrame из библиотеки Pandas.

Загрузка данных в X и y осуществляется средствами указанных библиотек. Например, пакет pandas.io предоставляет инструменты для чтения данных из распространенных форматов (CSV, Excel, JSON и SQL и других); пакет scipy.io специализируется на двоичных форматах, часто используемых в контексте научных вычислений (таких как файлы в формате MATLAB и формате Weka).

6.2. Обзор основных способов загрузки данных

Для возможности тестирования алгоритмов в Scikit-learn существует три основных возможности получения наборов данных в рамках библиотеки:

1) использование небольших стандартных наборов данных;

2) использование наборов реальных данных;

3) создание (генерация) набора данных. Так же существует возможность получения наборов данных из внешних депозитариев.

Загрузка небольших стандартных наборов данных реализована в пакете sklearn.datasets с помощью следующих функций:

а) load_boston (данные о ценах на жилье в Бостоне);

б) load_iris (данные о цветках ириса);

в) load_diabetes (данные о больных диабетом);

г) load_digits (данные о графических изображениях рукописных цифр);

д) load_linnerud (данные о посетителях фитнес-клуба);

е) load_wine (данные о составе вина);

ж) load_breast_cancer (данные о больных раком);

Эти наборы не соответствуют реальным задачам машинного обучения и поэтому, как правило, используются только для оперативной демонстрации поведения различных алгоритмов.

Пример загрузки данных о ценах на жилье в Бостоне приведен на рисунках 6.1 (программный код алгоритма) и 6.2 (результаты работы алгоритма).

```
# Загружаем набор данных (цены на жилье в Бостоне)
from sklearn.datasets import load_boston
X, y = load_boston(return_X_y=True)

# Выводим на экран размер матрицы объектов-признаков
print('Размер матрицы объектов-признаков:', X.shape)

# Выводим на экран значения признаков для первого объекта
print ('Значения признаков для первого объекта:\n', X[0][:])

# Выводим на экран значения метки для первого объекта
print ('Значение метки для первого объекта:\n',y[0])
```

Рис. 6.1. Программный код алгоритма загрузки данных о ценах на жилье в Бостоне

```
Размер матрицы объектов-признаков: (506, 13)
Значения признаков для первого объекта:
 [6.320e-03 1.800e+01 2.310e+00 0.000e+00 5.380e-01 6.575e+00 6.520e+01
 4.090e+00 1.000e+00 2.960e+02 1.530e+01 3.969e+02 4.980e+00]
Значение метки для первого объекта:
 24.0
```

Рис. 6.2. Результаты выполнения алгоритма загрузки данных о ценах на жилье в Бостоне

Загрузка наборов реальных данных реализована также в пакете sklearn.datasets с помощью следующих функций:

а) fetch_olivetti_faces, fetch_lfw_people, fetch_lfw_pair (набор данных об изображениях лиц);

б) fetch_20newsgroups, fetch_20newsgroups_vectorized, fetch_rcv1 (наборы данных о новостях);

в) fetch_covtype (наборы данных о лесных участках);

г) fetch_kddcup99 (наборы данных о вредоносных атаках на компьютерные системы);

д) fetch_california_housing (наборы данных о ценах на жилье в Калифорнии).

Пример загрузки данных о ценах на жилье в Калифорнии приведен на рисунках 6.3 (программный код алгоритма) и 6.4 (результаты работы алгоритма).

Scikit-learn включает в себя различные генераторы случайных выборок, которые можно использовать для создания искусственных наборов данных контролируемого размера и сложности.

```python
# Загружаем набор данных (цены на жилье в Калифорнии)
from sklearn.datasets import fetch_california_housing
X, y = fetch_california_housing(return_X_y=True)

# Выводим на экран размер матрицы объектов-признаков
print('Размер матрицы объектов-признаков:', X.shape)

# Выводим на экран значения признаков для первого объекта
print ('Значения признаков для первого объекта:\n', X[0][:])

# Выводим на экран значения метки для первого объекта
print ('Значение метки для первого объекта:\n',y[0])
```

Рис. 6.3. Программный код алгоритма загрузки данных о ценах на жилье в Калифорнии

```
Размер матрицы объектов-признаков: (20640, 8)
Значения признаков для первого объекта:
 [   8.3252       41.            6.98412698    1.02380952  322.
    2.55555556   37.88       -122.23      ]
Значение метки для первого объекта:
 4.526
```

Рис. 6.4. Результаты выполнения алгоритма загрузки данных о ценах на жилье в Калифорнии

б) datasets.make_circles (генерирует вложенные окружности);

в) datasets.make_classification (генерирует случайные данные для классификации);

г) datasets.make_gaussian_quantiles (генерирует изотропные гауссовские образцы);

д) datasets.make_low_rank_matrix (генерирует матрицу в основном низкого ранга с колоколообразными сингулярными значениями);

е) datasets.make_moons (генерирует два чередующихся полукруга);

ж) datasets.make_multilabel_classification (генерирует случайные данные для многоцелевой классификации);

з) datasets.make_regression (генерирует данные для регрессии);

и) datasets.make_s_curve (генерирует набор данных S-образной кривой);

к) и другие

Пример генерации наборов данных с применением datasets.make_blobs, datasets.make_circles и datasets.make_moons приведен на рисунках 6.5 (программный код алгоритма) и 6.6 (результаты работы алгоритма).

```python
def datasets_visualization(X, y, plt, m, n):
    plt.subplot(320+n)
    plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], 'gx', markersize=10)
    plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'b1', markersize=10)
    plt.plot(X[:, 0][y == 2], X[:, 1][y == 2], 'r+',markersize=10)
    plt.title("Пример "+str(n)+"\n "+ m)

import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=3, n_samples=100, random_state=150)
# визуализация набора данных
datasets_visualization(X, y, plt, "make_blobs (1)", 1)

# Пример 2
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=100, random_state=150)
transformation = [[0.61, -0.64], [-0.4, 0.8]]
X = np.dot(X, transformation)
# визуализация набора данных
datasets_visualization(X, y, plt, "make_blobs (2)", 2)

# Пример 3
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=100,cluster_std=[1.0, 2.5, 0.5],random_state=150)
# визуализация набора данных
datasets_visualization(X, y, plt, "make_blobs (3)", 3)

# Пример 4
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=1000, random_state=150)
X = np.vstack((X[y == 0][:75], X[y == 1][:20], X[y == 2][:5]))
y = np.concatenate([np.tile(0, 75),np.tile(1, 20),np.tile(2, 5)])
# визуализация набора данных
datasets_visualization(X, y, plt, "make_blobs (4)", 4)

# Пример 5
# генерация набора данных
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=100, factor=.5, noise=.05)
# визуализация набора данных
datasets_visualization(X, y, plt, "make_circles", 5)

# Пример 6
# генерация набора данных
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=.05)
# визуализация набора данных
datasets_visualization(X, y, plt, "make_moons", 6)

plt.show()
```

Рис. 6.5. Программный код алгоритма генерации данных

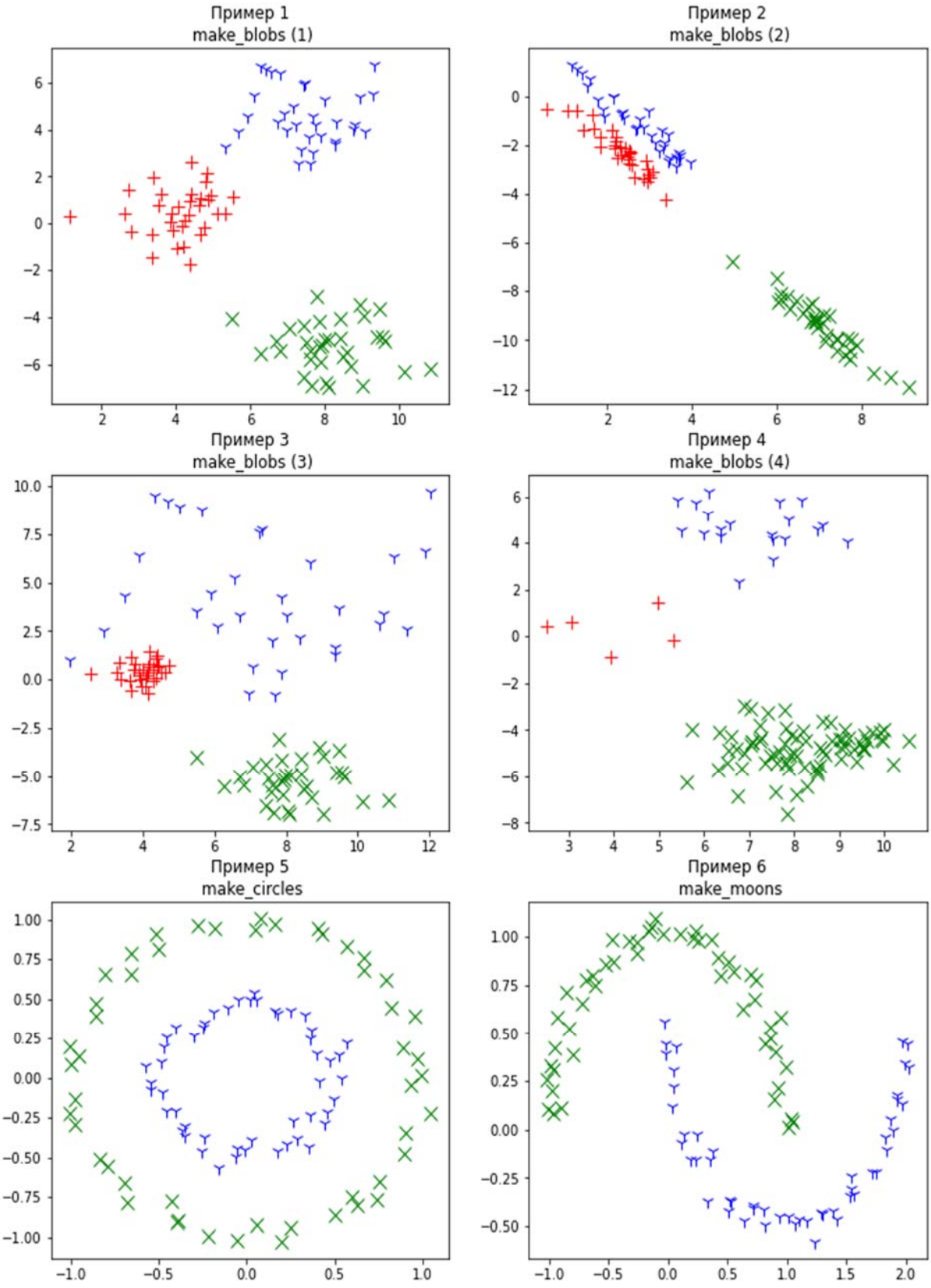


Рис. 6.6. Результаты выполнения алгоритма генерации данных с применением datasets.make_blobs, datasets.make_circles и datasets.make_moons

Scikit-learn предоставляет возможность загружать данные из внешних репозитариев. Так, например, пакет sklearn.datasets содержит специальную функцию sklearn.datasets.fetch_openml, позволяющую загружать наборы данных из репозитария openml.org – общедоступного источника данных для машинного обучения и экспериментов. Для загрузки набора данных из openml.org необходимо указать его имя и номер версии.

Пример загрузки набора данных для оценки кредитных рисков (name="credit-g", id="31", version="1") из репозитария openml.org с применением sklearn.datasets.fetch_openml приведен на рисунках 6.7 (программный код алгоритма) и 6.8 (результаты работы алгоритма).

```
# Зарузка набора данных
from sklearn.datasets import fetch_openml
credit_g = fetch_openml(name="credit-g")
X = credit_g.data
y =credit_g.target

# Вывод информации о наборе данных
print ('Версия набора - ', credit_g.details['version'])
print ('ID набора     - ', credit_g.details['id'])
#print (credit_g.DESCR)
print ('Признаки      - ',credit_g.feature_names)
print ('Первая выборка')
print (' X =', X[0])
print (' y =', y[0])
```

Рис. 6.7. Программный код алгоритма загрузки набора данных для оценки кредитных рисков из репозитария openml.org

```
Версия набора -  1
ID набора     -  31
Признаки      -  ['checking_status', 'duration', 'credit_history', 'purpose',
'credit_amount', 'savings_status', 'employment', 'installment_commitment', 'per
sonal_status', 'other_parties', 'residence_since', 'property_magnitude', 'age',
'other_payment_plans', 'housing', 'existing_credits', 'job', 'num_dependents',
'own_telephone', 'foreign_worker']
Первая выборка
 X = [0.000e+00 6.000e+00 4.000e+00 3.000e+00 1.169e+03 4.000e+00 4.000e+00
 4.000e+00 2.000e+00 0.000e+00 4.000e+00 0.000e+00 6.700e+01 2.000e+00
 1.000e+00 2.000e+00 2.000e+00 1.000e+00 1.000e+00 0.000e+00]
 y = good
```

Рис. 6.8. Результаты выполнения алгоритма загрузки набора данных для оценки кредитных рисков из репозитария openml.org

6.3. Реализация подготовки данных к анализу

Для возможности подготовки наборов данных к моделированию в Scikit-learn включен пакет sklearn.preprocessing, который представляет широкий перечень общих классов и методов масштабирования, центрирования, нормализации и бинаризации. Рассмотрим для примера несколько из них.

При использовании функции scale и класса StandardScaler масштабированные данные имеют для каждого признака по всем выборкам нулевое среднее значение и единичную дисперсию. Отличие класса от функции заключается в возможности использовать дополнительные алгоритмы к масштабированным данным, в т.ч. предоставляется возможность повторного применения выполненного преобразования к новым выборкам.

Пример масштабирования набора данных с помощью функции scale и класса StandardScaler приведен на рисунках 6.9 (программный код алгоритма) и 6.10 (результаты работы алгоритма).

```
from sklearn import preprocessing
import numpy as np

# Формируем исходный набор данных
X = np.array([[ 1., -1.,  2.],
              [ 2.,  0.,  0.],
              [ 0.,  1., -1.]])

# Осуществляем масштабирование с помощью функции preprocessing.scale
X_scaled = preprocessing.scale(X)
print ('Масштабирование с помощью функции preprocessing.scale: \n',X_scaled)
print ('Средние значения по столбцам признаков: \n',X_scaled.mean(axis=0))
print ('Дисперсия по столбцам признаков: \n',X_scaled.std(axis=0))

# Осуществляем масштабирование с помощью класса preprocessing.StandardScaler
scaler= preprocessing.StandardScaler().fit(X)
X_scaled = scaler.transform(X)
print ('\nМасштабирование с помощью StandardScaler: \n',X_scaled)
print ('Средние значения по столбцам признаков: \n',X_scaled.mean(axis=0))
print ('Дисперсия по столбцам признаков: \n',X_scaled.std(axis=0))

# Преобразование новых выборок в масштаб существующего набора данных
X_new = np.array([[ 1., -1.,  2.],
                  [ 0.,  1., -1.]])
print ('\nПреобразование: \n', scaler.transform(X_new))
```

Рис. 6.9. Программный код алгоритма масштабирования с помощью функции scale и класса StandardScaler (пакет sklearn.preprocessing)

```
Масштабирование с помощью функции preprocessing.scale:
 [[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
Средние значения по столбцам признаков:
 [0. 0. 0.]
Дисперсия по столбцам признаков:
 [1. 1. 1.]

Масштабирование с помощью класса preprocessing.StandardScaler:
 [[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
Средние значения по столбцам признаков:
 [0. 0. 0.]
Дисперсия по столбцам признаков:
 [1. 1. 1.]

Преобразование новых выборок в масштаб существующего набора данных:
 [[ 0.         -1.22474487  1.33630621]
 [-1.22474487  1.22474487 -1.06904497]]
```

Рис. 6.10. Результаты выполнения алгоритма масштабирования с помощью функции scale и класса StandardScaler (пакет sklearn.preprocessing)

При использовании функций maxabs_scale, minmax_scale и классов MaxAbsScaler и MinMaxScaler масштабированные данных осуществляется в определенном диапазоне, который задаётся соответственно абсолютными максимальными или минимальными и максимальными значениями в столбце признаков.

Пример масштабирования набора данных с помощью классов MaxAbsScaler и MinMaxScaler приведен на рисунках 6.11 (программный код алгоритма) и 6.12 (результаты работы алгоритма).

```python
from sklearn import preprocessing
import numpy as np

# Формируем исходный набор данных
X = np.array([[ 1., -1.,  2.],[ 2.,  0.,  0.],[ 0.,  1., -2.]])

# Осуществляем масштабирование с помощью класса preprocessing.MaxAbsScaler
X_scaled = preprocessing.MaxAbsScaler().fit_transform(X)
print ('\nМасштабирование с помощью класса MaxAbsScale: \n',X_scaled)

# Осуществляем масштабирование с помощью класса preprocessing.MinMaxScaler
X_scaled = preprocessing.MinMaxScaler().fit_transform(X)
print ('\nМасштабирование с помощью класса MinMaxScaler: \n',X_scaled)
```

Рис. 6.11. Программный код алгоритма масштабирования с помощью классов MaxAbsScaler и MinMaxScaler (пакет sklearn.preprocessing)

```
Масштабирование с помощью класса preprocessing.MaxAbsScale:
 [[ 0.5  -1.    1.  ]
 [ 1.    0.    0.  ]
 [ 0.    0.25 -1.  ]]

Масштабирование с помощью класса preprocessing.MinMaxScaler:
 [[0.5 0.  1. ]
 [1.  0.8 0.5]
 [0.  1.  0. ]]
```

Рис. 6.12. Результаты выполнения алгоритма масштабирования с помощью классов MaxAbsScaler и MinMaxScaler (пакет sklearn.preprocessing)

Такой подход к масштабированию применяется в том случае, когда необходима устойчивость к очень небольшим стандартным отклонениям функций и сохранение нулевых записей в разреженных данных. Если данные содержат много выбросов, масштабирование с использованием среднего значения и дисперсии данных будет не целесообразным.

В такой ситуации используют функцию robust_scale и класс, которые используют более надежные оценки для центра и диапазона набора данных: медианное значение удаляется и данные масштабируются в соответствии с заданным квантильным диапазоном (квантиль в математической статистике – значение, которое заданная случайная величина не превышает с фиксированной вероятностью).

Пример масштабирования набора данных с помощью класса RobustScaler приведен на рисунках 6.13 (программный код алгоритма) и 6.14 (результаты работы алгоритма).

```
1 from sklearn import preprocessing
2 import numpy as np
3
4 # Формируем исходный набор данных
5 X = np.array([[ 1., -1.,  2.],
6               [ 100.,  0.,  0.],
7               [ 0.,  1., -2.]])
8
9 # Осуществляем масштабирование с помощью класса preprocessing.RobustScaler
10 X_scaled = preprocessing.RobustScaler().fit_transform(X)
11 print ('\nМасштабирование с помощью класса RobustScaler: \n',X_scaled)
```

Рис. 6.13. Программный код алгоритма масштабирования с помощью класса RobustScaler (пакет sklearn.preprocessing)

```
Масштабирование с помощью класса preprocessing.RobustScaler:
 [[ 0.    -1.    1.  ]
 [ 1.98  0.    0.  ]
 [-0.02  1.   -1.  ]]
```

Рис. 6.14. Результаты выполнения алгоритма масштабирования с помощью класса RobustScaler (пакет sklearn.preprocessing)

В Scikit-learn доступны также два вида нелинейных преобразований наборов данных: квантильные преобразования и степенные преобразования, которые основаны на монотонных преобразованиях характеристик и, таким образом, сохраняют ранг значений по каждой из них.

Класс QuantileTransformer и функция quantile_transform обеспечивают непараметрическое преобразование данных в равномерное распределение со значениями от 0 до 1. Класс PowerTransformer обеспечивает параметрическое преобразование для придания данным гауссовского типа: осуществляется стабилизация дисперсии и минимизации асимметрии данных.

В Scikit-learn доступно осуществление нормализации данных. Нормализация – это процесс масштабирования, при котором значения признаков во входном векторе приводятся к некоторому заданному диапазону, например, [0…1] или [-1…1]. Для нормализации используют функцию normalize и класс Normalizer.

Пример нормализации набора данных с помощью класса Normalizer приведен на рисунках 6.15 (программный код алгоритма) и 6.16 (результаты работы алгоритма).

```
from sklearn import preprocessing
import numpy as np

# Формируем исходный набор данных
X = np.array([[ 1., -1.,  2.],[ 1.,  0.,  0.],[ 0.,  1., -2.]])

# Осуществляем нормализацию с помощью класса Normalizer
X_scaled = preprocessing.Normalizer().fit_transform(X)
print ('\nНормализация с помощью класса Normalizer: \n',X_scaled)
```

Рис. 6.15. Программный код алгоритма нормализации с помощью класса Normalizer (пакет sklearn.preprocessing)

```
Нормализация с помощью класса preprocessing.Normalizer:
 [[ 0.40824829 -0.40824829  0.81649658]
 [ 1.          0.          0.        ]
 [ 0.          0.4472136  -0.89442719]]
```

Рис. 6.16. Результаты выполнения алгоритма нормализации с помощью класса Normalizer (пакет sklearn.preprocessing)

Для преобразования категориальных признаков (например, данных о поле человека, о городе его проживания и т.д.) в целочисленные коды используется класс OrdinalEncoder. С помощью этого класса осуществляется преобразование каждого значения категориального признака в новое целочисленное значение от 0 до n_categories – 1 (где n_categories – общее количество категорий).

По умолчанию OrdinalEncoder определяет категории на основе уникальных значений в каждой столбце признаков, однако, есть возможность указать категории вручную. Пример преобразования категориальных признаков с помощью класса OrdinalEncoder Normalizer (пакет sklearn.preprocessing) приведен на рисунках 6.17 (программный код алгоритма) и 6.18 (результаты работы алгоритма).

```
from sklearn import preprocessing
import numpy as np

# Формируем исходный набор данных
X = np.array([['женский',  'директор'],
              ['женский',  'экономист'],
              ['мужской',  'юрист']])

# Осуществляем преобразования категориальных признаков
# с помощью класса preprocessing.OrdinalEncoder
X_scaled = preprocessing.OrdinalEncoder  ().fit_transform(X)
print ('\nПреобразования категориальных признаков')
print ('помощью класса preprocessing.OrdinalEncoder : \n', X_scaled)
```

Рис. 6.17. Программный код алгоритма преобразования категориальных признаков с помощью класса Normalizer (пакет sklearn.preprocessing)

```
Преобразования категориальных признаков
с помощью класса preprocessing.OrdinalEncoder :
 [[0. 0.]
 [0. 1.]
 [1. 2.]]
```

Рис. 6.18. Результаты выполнения алгоритма преобразования категориальных признаков с помощью класса Normalizer (пакет sklearn.preprocessing)

Для дискретизации, т.е. разделении непрерывных функций на дискретные значения используется класс KBinsDiscretizer.

Для усложнения модели с учетом нелинейных особенностей входных данных применяют простой и распространенный метод пре-

образования набора данных – полиномиальные функции, которые в Scikit-learn реализованы в классе PolynomialFeatures.

Заполнение пропущенных значений реализовано в Scikit-learn в пакете sklearn.impute, в который включены следующие классы:

а) SimpleImputer (пропущенные значения определяются либо напрямую пользователем (strategy=mean), либо на основании статистической обработки значений в столбце признака на основании среднего (strategy=median), медианного (strategy=most_frequent) или наиболее частого встречающегося значения (strategy=most_frequen);

б) IterativeImputer (пропущенные значения определяются на основании моделирования каждой функции с пропущенными значениями как функции других функций в циклическом режиме);

в) MissingIndicator (пропущенные значения определяются как бинарные).

г) KNNImputer (пропущенные значения определяются на основании метода k-ближайших соседей).

Пример преобразования категориальных признаков с помощью класса SimpleImputer (пакет sklearn.impute) приведен на рисунках 6.19 (программный код алгоритма) и 6.20 (результаты работы алгоритма).

```
from sklearn.impute import SimpleImputer
import numpy as np

# Формируем исходный набор данных
X = np.array([[ 1., -1.,  2.],
              [ np.nan, np.nan,  0.],
              [ 0.,  1., -2.]])

# Осуществляем заполнение пропущенных значений с помощью класса SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
X_imputed = imp.fit_transform(X)
print ('\nЗаполнение пропущенных значений')
print ('с помощью класса SimpleImpute: \n',X_imputed)
```

Рис. 6.19. Программный код алгоритма заполнения пропущенных значений с помощью класса SimpleImputer (пакет sklearn.impute)

```
Заполнение пропущенных значений с помощью класса SimpleImpute:
 [[ 1.  -1.   2. ]
 [ 0.5  0.   0. ]
 [ 0.   1.  -2. ]]
```

Рис. 6.20. Результаты выполнения алгоритма заполнения пропущенных значений с помощью класса SimpleImputer (пакет sklearn.impute)

Производительность и качество машинного обучения при росте числа функциональных зависимостей улучшается, но только до определенного момента, после которого повышается вероятность переобучения, за счет увеличения размерности данных.

Существует несколько подходов, которые используют для борьбы с переобучением, из которых одним из самых эффективных является уменьшение размерности данных. Суть этого подхода заключается в выборе наиболее важных компонентов пространства признаков, сохранении их и отбрасывании других компонентов.

Уменьшение размерности в машинном обучении используется не только для контроля переобучения, а также для борьбы с вычислительными затратами, для визуализации и интерпретации высокомерных наборов данных.

Снижение размерности может использоваться как в контексте обучения с учителем, так и без него. В случае обучения без учителя уменьшение размерности часто используется для предварительной обработки данных путем выбора или извлечения признаков.

Основными методами, используемыми для уменьшения размерности при обучении без учителя, являются метод анализа главных компонентов (PCA) и метод разложения по сингулярным значениям (SVD).

В случае обучения с учителем уменьшение размерности может использоваться для упрощения функциональных зависимостей, вводимых в классификатор машинного обучения. Наиболее распространенными методами, используемыми для уменьшения размерности для задач контролируемого обучения, являются метод неотрицательного матричного разложения (NMF), метод дискриминантного анализа (LDA) и PCA, которые можно использовать для прогнозирования новых случаев.

В Scikit-learn для работы с моделями понижения размерности включен модуль sklearn.decomposition, который предоставляет возможность работать с широким перечнем основных методов понижения размерности, включая:

а) метод главных компонент (класс decomposition.PCA);

б) метод разложения по усеченным сингулярным значениям (decomposition.TruncatedSVD);

в) метод неотрицательного матричного разложения (NMF) – класс decomposition.NMF.

Для возможности работы с методом дискриминантного анализа в Scikit-learn включен модуль sklearn.discriminant_analysis, содержащий реализацию метода линейного дискриминантного анализа (discriminant_analysis.LinearDiscriminantAnalysis).

Метод главных компонент (Principal Component Analysis PCA) – это статистический метод, который создает новые функции или характеристики данных путем анализа характеристик набора данных. По сути, характеристики данных суммируются или объединяются.

PCA выбирает «основные» или наиболее важные характеристики набора данных и создает на их основе функции. При выборе только тех функций, которые больше всего влияют на набор данных, размерность уменьшается. PCA сохраняет корреляции между переменными при создании новых функций. Основные компоненты, созданные с помощью этого метода, представляют собой линейные комбинации исходных переменных, вычисленные с помощью концепций, называемых собственными векторами. Предполагается, что новые компоненты ортогональны или не связаны друг с другом.

Пример применения метода PCA (пакет sklearn.decomposition) для понижения размерности приведен на рисунках 6.21 (результаты работы алгоритма) и 6.22 (программный код алгоритма).

```
import numpy as np
from sklearn.decomposition import PCA
X = np.array([[1, 5, 3, 5], [8, 1, 3, 5], [3, 2, 2, 4], [1, 1, 1, 5],
              [2, 1, 0, 5], [3, 2, 1, 3]])
print ("Исходная выборка\n", X)
pca = PCA(n_components=2)
pca.fit(X)
X = pca.transform(X)
print ("Выборка после снижения размерности\n", X)
X = pca.inverse_transform(X)
print ("Восстановленная исходная выборка\n", X)
```

Рис. 6.21. Понижение размерности с помощью метода PCA

```
Исходная выборка
 [[1 5 3 5]
 [8 1 3 5]
 [3 2 2 4]
 [1 1 1 5]
 [2 1 0 5]
 [3 2 1 3]]
Выборка после снижения размерности
 [[-2.44727463  2.98473216]
 [ 5.24356852  0.57337098]
 [ 0.05627525  0.17815511]
 [-1.7494869  -1.36542869]
 [-0.9775529  -1.87966077]
 [-0.12552934 -0.49116878]]
Восстановленная исходная выборка
 [[0.98054573 4.97067317 3.07119191 4.64830155]
 [8.031767   1.06937761 2.94559057 4.55815178]
 [3.07139982 2.12414774 1.78623878 4.50990245]
 [1.20241845 1.39485419 0.51742631 4.41726157]
 [1.88262042 0.79100199 0.33731155 4.39352966]
 [2.83124858 1.6499453  1.34224087 4.47285298]]
```

Рис. 6.22. Результаты выполнения алгоритма понижения размерности с помощью метода PCA (пакет sklearn.decomposition)

6.4. Моделирование данных и анализ качества моделей

В Scikit-learn представлен широкий выбор методов обучения, которых объединяет единый способ использования

Для всех классов моделей доступны следующие методы:

а) model.fit () – настройка на данные (обучение): для обучения с учителем в качестве параметров передается матрица объектов-признаков и матрица объектов-меток (X, y); для обучения без учителя в качестве параметров передается только матрица объектов-признаков (X);

б) model.predict () – для обучения с учителем предсказывает значения целевой переменной (в качестве параметров должен выступать тестовый набор данных – X_test), для обучения без учителя получает матрицы объектов-меток (не во всех моделях);

в) model.predict_proba () – для обучения с учителем рассчитывает «степень уверенности» в ответе (вероятность) (не во всех моделях), для обучения без учителя рассчитывает вероятности принадлежности к компонентам для каждой точки;

г) model.transform () – для обучения с учителем осуществляет отбор признаков либо на основании сжатия «обучающей» матрицы, либо на основе выделения наиболее информативных признаков, для обучения без учителя преобразует данные;

д) model. score () – для обучения с учителем оценивает качество модели, для обучения без учителя оценивает «правдоподобие» модели (насколько данные соответствуют модели).

Переобучение (переподгонка) в машинном обучении – явление, когда построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении (на примерах из тестовой выборки).

Это связано с тем, что при построении модели («в процессе обучения») в обучающей выборке обнаруживаются некоторые случайные закономерности, которые отсутствуют в генеральной совокупности. Иными словами - модель запоминает огромное количество всех возможных примеров вместо того, чтобы научиться подмечать особенности [1].

В Scikit-learn реализован один из самых эффективных методов предотвращения переобучения: метод перекрёстной проверки. В качестве его основы выступает случайное разбиение на обучающие и тестовые наборы с помощью вспомогательной функции train_test_split, а также метрика перекрестной проверкой с помощью функции cross_val_score (обе функции включены в sklearn.model_selection).

В Scikit-learn для обеспечения возможности оценки ошибки прогнозирования моделей машинного обучения включен ряд функций, объединённых в пакет sklearn.metrics. Данные метрики являются уникальными для каждого класса решаемых задач (классификации, регрессии и кластеризации) и будут нами рассмотрены далее в контексте использования конкретным методов машинного обучения.

Вопросы для обсуждения

1. Роль библиотеки Scikit-learn в интеллектуальном анализе данных
2. Функциональные возможности библиотеки Scikit-learn
3. Признаковые описания объекта.

2. Гипотеза компактности.
3. Метрики, виды метрик.
4. Нормализация признаков.
5. Стандартизация признаков
6. Весовая Евклидова метрика
7. Метрика Минковского.
6. Масштабирование признаков.
7. Кросс-валидация.
8. Алгоритм выполнения кросс-валидации по блокам.

Практические задания

Задание 6.1

На основании вариантов видов деятельности, приведённых по вариантам в задании 1.1, найдите во внешних открытых репозитариях несколько наборов данных им соответствующих и сделайте их описание по следующей схеме:

а) наименование репозитария с указанием его интернет-адреса,

б) краткое и полное наименование набора данных,

в) идентификатор набора данных,

г) краткая характеристика (решаемые задачи, первоисточник данных и т.д.),

д) количественные параметры набора данных (размер выборки),

е) описание признаков и меток.

Задание 6.2

На основании решения задания 6.1 напишите программный код с использованием Python-библиотеки Scikit-learn, который позволял бы автоматизировать загрузку и формирование описания одного из найденных наборов данных.

Задание 6.3.

На основании решения задания 6.2 проведите анализ данных в наборе на доступность, точность, взаимосвязанность, полноту, непротиворечивость, однозначность, релевантность, надежность и своевременность. Для исследования данных примените Python-библиотеку Seaborn.

Изучите каждый признак и его характеристики: имя, тип данных, процент отсутствующих значений; зашумленность и тип шума (стохастический, выбросы, ошибки округления и т.д.) и тип распределения (гауссово, равномерное, логарифмическое и т.д.).

Предложите способы очистки данных (обработки пропущенных значений, дубликатов, противоречий, аномальных значений и выбросов, шумов, фиктивных значений и ошибок ввода данных) и оптимизации (снижение размерности, исключение незначащих признаков).

Напишите программный код с использованием Python-библиотеки Scikit-learn, в котором реализуйте предложенные способы очистки и оптимизации данных.

Задание 6.4

Используя функцию datasets.make_regression или datasets.make_multilabel_classification Python-библиотеки Scikit-learn сгенерируйте случайные данные для использования в задачах регрессии (по вариантам: размер выборки, количество признаков (независимых переменных), количество информативных признаков, количество целей (зависимых переменных), смещение относительно базовой линейной модели): 1) 500, 5, 3, 1, 0.05; 2) 600, 4, 3, 2, 0.11; 3) 700, 7, 1, 0.15; 4) 500, 6, 4, 2, 0.21; 5) 600, 5, 3, 1, 0.25; 6) 700, 4, 3, 2, 0.31; 7) 500, 4, 3, 1, 0.35; 8) 600, 5, 4, 2, 0.41; 9) 700, 6, 4, 1, 0.45; 10) 500, 7, 5, 2, 0.51.

Задание 6.5

Используя функцию datasets.make_classification Python-библиотеки Scikit-learn сгенерируйте случайные данные для использования в задачах классификации (по вариантам: размер выборки, количество признаков, количество информативных признаков, количество классов): 1) 500, 8, 5, 4; 2) 600, 9, 4, 2; 3) 700, 10, 7, 5; 4) 500, 11, 6, 3; 5) 600, 12, 5, 3; 6) 700, 11, 4, 5; 7) 500, 10, 4, 2; 8) 600, 9, 5, 4; 9) 700, 8, 6, 5; 10) 500, 9, 7, 6

Задание 6.6.

Используя функцию datasets.make_blobs Python-библиотеки Scikit-learn сгенерируйте случайные данные для использования в задачах кластеризации (по вариантам: размер выборки, количество при-

знаков, количество кластерных центров, разброс данных от центра кластера (стандартное отклонение)): 1) 500, 5, 3, 1.05; 2) 600, 4, 2, 1.11; 3) 700, 4, 4, 1.15; 4) 500, 6, 2, 1.21; 5) 600, 5, 3, 1.25; 6) 700, 4, 2, 1.31; 7) 500, 4, 3, 1.35; 8) 600, 5, 2, 1.41; 9) 700, 6, 4, 1.45; 10) 500, 4, 2, 1.51.

Библиографический список

1. Переобучение // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Переобучение

2. Замятин А. В. Введение в интеллектуальный анализ данных: учебное пособие / А. В. Замятин; Нац. исслед. Том. гос. ун-т. - Томск: Издательский Дом Томского государственного университета, 2016. URL: http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000529594

3. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

4. Лесковец, Ю. Анализ больших наборов данных / Лесковец Ю., Раджараман А. , Джеффри Д. Ульман - Москва : ДМК Пресс, 2016. - 498 с. - ISBN 978-5-97060-190-7. - Текст : электронный // ЭБС "Консультант студента": [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN9785970601907.html

5. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

6. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

7. Машинное обучение // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Машинное_обучение

Глава 7. SCIKIT-LEARN: МОДЕЛИ РЕГРЕССИИ

В данной главе рассматриваются следующие вопросы:

1. Регрессионный анализ как задача машинного обучения

2. Линейные модели регрессии

3. Нелинейные модели регрессии

7.1. Регрессионный анализ как задача машинного обучения

Регрессия – односторонняя стохастическая зависимость, устанавливающая соответствие между случайными переменными, то есть математическое выражение, отражающее связь между зависимой переменной (критериальной переменной, y) и независимыми переменными (предикторами, X) при условии, что это выражение будет иметь статистическую значимость .

Терминология зависимых и независимых переменных отражает лишь математическую зависимость переменных, а не причинно-следственные отношения.

Регрессионный анализ – одна из важнейших областей машинного обучения, представляющий собой набор статистических методов исследования регрессии, позволяющих:

1) определить степень детерминированности вариации критериальной переменной предикторами;

2) предсказать значение критериальной переменной с помощью предикторов;

3) определить вклад отдельных предикторов в вариацию критериальной переменной называют.

Регрессионный анализ используется, например, для выявления зависимости размера заработной платы сотрудников, от как параметров как опыт, уровень образования, должность, город, в котором они работают, и так далее.

Аналогичным образом регрессионный анализ может быть использован для установления математической зависимости цен на дома от их площади, количества спален, расстояния до центра города и так далее, а также во множестве других похожих ситуациях.

Задачи регрессии обычно имеют одну непрерывную и неограниченную критериальную переменную. Предикторы, однако, могут

быть непрерывными, дискретными или даже категориальными данными, такими как пол, национальность, бренд и так далее.

С точки зрения машинного обучения регрессия является частным случаем задач прогнозирования и выполняется с помощью обучения с учителем на этапе тестирования: параметры модели подгоняются под наблюдаемый набор данных.

Различают линейные и нелинейные модели регрессии. Рассмотрим данные модели более подробно и продемонстрируем работу с ними с применением Python-библиотека Scikit-learn.

7.2. Линейные модели регрессии

Линейная регрессия – используемая в статистике регрессионная модель зависимости одной (объясняемой, зависимой) переменной y от другой или нескольких других переменных (факторов, регрессоров, независимых переменных) X с линейной функцией зависимости.

Общее уравнение для линейной модели выглядит следующим образом:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{p} x_j w_j = \mathbf{x}^\top \mathbf{w}, \quad \mathbf{x} = (1, x_1, \ldots, x_p)$$

где $x^\top$ – вектор входов

На основании вектора входов $x_\top$ возможно предсказать на горизонте прогнозирования выход y.

Примером применения линейной регрессии является выявление трендов, сезонности и раскладки (смены модели ряда) при анализе временных рядов рыночных цен, объёмов продаж и потребления товаров, работ и услуг и др.

Например, линейная регрессия используется для предсказания скидки на продукты на основе поведения покупателей в прошлом; застройщики при помощи данного метода могут предсказать, сколько домов он продаст в ближайшие месяцы и по какой цене и т.д.

Свойства линейной регрессии:

1) легкость моделирования;

2) высокая полезность при создании не очень сложной зависимости, а также при небольшом количестве данных;

3) интуитивно-понятна;

4) чувствительность к выбросам.

В Scikit-learn для работы с линейными моделями регрессии включен пакет linear_mode, который представляет возможность работы со следующими методами построения линейной регрессии:

а) методом наименьших квадратов (linear_model. LinearRegression);

б) методом наименьших квадратов с регуляризацией (linear_model.Ridge);

в) методом риджевой регрессии со встроенной перекрестной проверкой (linear_model.RidgeCV);

г) построение линейной регрессии на основе использования метода минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска (linear_model. SGDRegressor).

Для построения линейной регрессии методом наименьших квадратов используется класс linear_model. LinearRegression.

LinearRegression соответствует линейной модели с коэффициентами w = (w1,…, wp) для минимизации разности суммы квадратов между наблюдаемыми результатами в тренировочном наборе данных и результатами, предсказанными с помощью линейной аппроксимации.

Описание класса модели:

class sklearn.linear_model.LinearRegression (*, fit_intercept, normalize, copy_X, n_jobs)

Параметры класса модели:

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w0 (в случае fit_intercept=False w0 (intercept_ = 0);

normalize (тип данных – bool; значение по умолчанию – False) – указывает алгоритму необходимость нормализации данных (нормализация – это изменение масштаба данных из исходного диапазона в масштаб, при котором все значения параметра находились бы в диапазоне от 0 до 1);

copy_X (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость копирования данных без их перезаписи;

n_jobs (тип данных – int; значение по умолчанию – None) указывает алгоритму на количество процессоров (или их ядер), которые он может использовать для своей работы (-1 означает использование всех процессоров, None – одного процессора).

Основные атрибуты класса модели:

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения рассчитанных коэффициентов wtarget, j для задачи линейной регрессии (при наличии одной целевой функции одномерный массив размерностью [n_features];

intercept_ (тип данных – float или одномерный массив размерностью [n_targets]) – предназначен для хранения значения независимого параметра w0 или wtarget,0. При fit_intercept = False intercept_ = 0.0.

Основные методы класса модели:

fit(X, y[, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples, n_features] или [n_samples]) – зависимые переменные модели, sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода наименьших квадратов для построения линейной регрессии приведен на рисунках 7.1 (результаты работы алгоритма) и 7.2 (программный код алгоритма).

Для иллюстрации работы алгоритмов построения линейной регрессии мы для наглядности использовали парную регрессию. Однако, всё, что выше изложено применимо для построения множественной линейной регрессии.

```
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error, r2_score

# генерация набора данных
X, y = datasets.make_regression(n_samples=100, n_features=1, noise=11,
                                shuffle=True)

#разбиение набора данных на тренировочный и тестовый
train_size = 88
test_size = 12
X_train = X[:-train_size]
X_test = X[-test_size:]
y_train = y[:-train_size]
y_test = y[-test_size:]

# построение и обучение модели:
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# определение и вывод параметров построенной модели
w0 = str(regr.intercept_)
w1 = str(regr.coef_[0])
y_func =  'y=' + w0 +('+' if w1>0 else '') + str(w1)+'x'
print('w0 = '+ w0)
print('w1 =  '+ w1)
print(y_func)

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
        % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
        % r2_score(y_test, y_pred))
print('или (1 - идеальное предсказание): %.3f' % regr.score(X_test, y_test))

# вывод прогноза на основании тестового набора данных
plt.scatter(X_test, y_test, color='red', s=15)
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Рис. 7.1. Программный код алгоритма построения линейной регрессии с использованием метода наименьших квадратов

Пример использования метода наименьших квадратов для построения множественной линейной регрессии приведен на рисунках 7.3 (программный код алгоритма) и 7.4 (результаты работы алгоритма).

```
w0 = -0.447496965902
w1 =  84.997960708
y=-0.447496965902+84.997960708x
Среднеквадратическое отклонение: 112.433
Коэффициент детерминации (R-квадрат): 0.983
или (1 - идеальное предсказание): 0.983
```

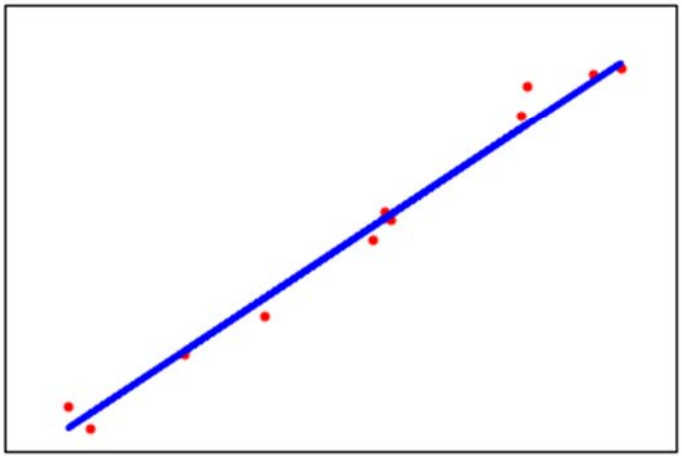


Рис. 7.2. Результаты выполнения алгоритма построения линейной регрессии с использованием метода наименьших квадратов

У регрессии по методу наименьших квадратов есть один очень серьёзный недостаток: она проявляет тенденции к переобучению, то есть сильно зависит от обучающих данных.

В качестве подхода, направленного на снижение склонности к переобучению, используют метод регуляризация, который заключается в наложении дополнительных ограничений на искомые параметры, которые могут предотвратить излишнюю сложность модели.

Смысл процедуры заключается в «стягивании» в ходе настройки вектора коэффициентов ω таким образом, чтобы они в среднем оказались несколько меньше по абсолютной величине, чем это было бы при оптимизации по методу наименьших квадратов. Параметр регуляризации (α) имеет смысл штрафа за сложность.

При значении параметра регуляризации α=0, регрессия сводится к обычному методу наименьших квадратов, а при увеличении α формируемая модель становится все более «лаконичной» за счет снижения размерности коэффициентов ω (или обнуления части из них).

```python
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# генерация набора данных
X, y = datasets.make_regression(n_samples=10000, n_features=2, noise=15,
                                shuffle=True)

#разбиение набора данных на тренировочный и тестовый
train_size = 88
test_size = 12
X_train = X[:-train_size][:-train_size]
X_test = X[-test_size:][-test_size:]
y_train = y[:-train_size][:-train_size]
y_test = y[-test_size:][-test_size:]

# построение и обучение модели:
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# определение и вывод параметров построенной модели
w0 = regr.intercept_
w1 = regr.coef_[0]
w2 = regr.coef_[1]
print("w0 = {:.8f}".format(w0))
print("w1 = {:.8f}".format(w1))
print("w2 = {:.8f}".format(w2))
print("y = {:.8f} + {:.8f}x1 + {:.8f}x2".format(w0, w1,w2))

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
      % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
      % r2_score(y_test, y_pred))
print('или (1 - идеальное предсказание): %.3f' % regr.score(X_test, y_test))

# вывод прогноза на основании тестового набора данных
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_test[:,0], X_test[:,1], y_pred, marker='.', color='red')
ax.set_xlabel("x1")
ax.set_ylabel("x2")
ax.set_zlabel("y")
xs = np.tile(np.arange(6)-3, (1,1))
ys = np.tile(np.arange(6)-3, (1,1)).T
zs = w0+xs*w1+ys*w2
ax.plot_surface(xs,ys,zs, alpha=0.1)
plt.show()
```

Рис. 7.3. Программный код алгоритма построения множественной линейной регрессии с использованием метода наименьших квадратов

```
w0 = 0.14497476
w1 = 2.21330690
w2 = 90.42679075
y = 0.14497476 + 2.21330690x1 + 90.42679075x2
Среднеквадратическое отклонение: 205.798
Коэффициент детерминации (R-квадрат): 0.984
или (1 - идеальное предсказание): 0.984
```

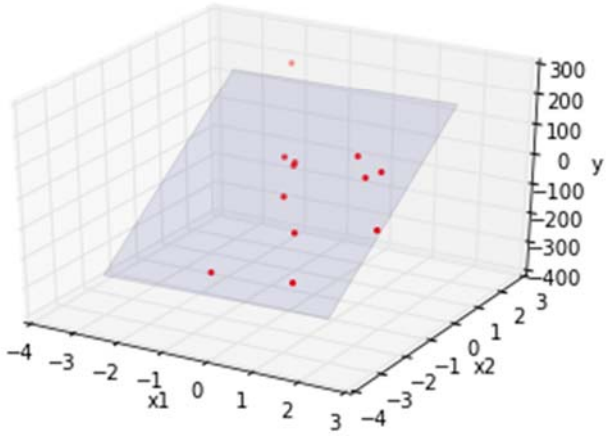


Рис. 7.4. Результаты выполнения алгоритма построения линейной регрессии с использованием метода наименьших квадратов

Оптимальная величина α находится с использованием перекрестной проверки, т.е. ей соответствует минимальная ошибка прогноза (коэффициент детерминации) на наблюдениях, не участвовавших в построении самой модели.

Один из методов понижения размерности коэффициентов ω является гребневая регрессия (ridge regression), который часто применяют для борьбы с избыточностью данных, когда независимые переменные коррелируют друг с другом (т.е. имеет место мультиколлинеарность) .

Для построения линейной регрессии методом наименьших квадратов с регуляризацией используется класс linear_model.Ridge

В рамках данного метода регуляризация задается L2-нормой. Норма L2 вычисляет расстояние векторной координаты от начала векторного пространства.

Описание класса модели:

class sklearn.linear_model.Ridge(alpha, *, fit_intercept, normalize, copy_X, max_iter, tol, solver, random_state)

Основные параметры класса модели:

alpha (тип данных – float или одномерный массив размерностью [n_targets]; значение по умолчанию – 1.0; ограничение – больше 0) – указывает алгоритму значение параметра регуляризации α (величину штрафа за сложность модели);

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w0 (в случае fit_intercept=False w0 (intercept_ = 0));

normalize (тип данных – bool; значение по умолчанию – False) – указывает алгоритму необходимость нормализации данных;

copy_X (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость копирования данных без их перезаписи;

max_iter (тип данных – int; значение по умолчанию – None) – указывает алгоритму максимальное количество итераций, None соответствует значению 1000;

tol (тип данных – int; значение по умолчанию – 0.001)) – указывает алгоритму точность решения;

solver (тип данных – str; значение по умолчанию – ‘auto’) – указывает алгоритму метод решения (возможные варианты: {‘auto’, ‘svd’, ‘cholesky’, ‘lsqr’, ‘sparse_cg’, ‘sag’, ‘saga’}, например при значении параметра ‘sag’ алгоритм использует стохастический средний градиентный спуск), при установке ‘auto’ метод выбирается автоматически в зависимости от типа данных;

random_state (тип данных – int; значение по умолчанию – None; ограничение – solver={‘sag’|‘saga’}) указывает алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения рассчитанных коэффициентов wtarget,j для задачи линейной регрессии (при наличии одной целевой функции одномерный массив размерностью [n_features];

intercept_ (тип данных – float или одномерный массив размерностью [n_targets]) – предназначен для хранения значения независимого параметра w0 или wtarget,0. При fit_intercept = False intercept_ = 0.0.

Основные методы класса модели:

fit(X, y[, sample_weight]) – построение модели на основании обучающего набора данных, где:

X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели,

y (тип данных – массив размерностью [n_samples, n_features] или [n_samples]) – зависимые переменные модели,

sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода наименьших квадратов с регуляризацией для построения линейной регрессии приведен на рисунках 7.5 (результаты работы алгоритма) и 7.6 (программный код алгоритма)

Рис. 7.5. Результаты выполнения алгоритма построения линейной регрессии с использованием метода наименьших квадратов с регуляризацией

```
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error

# генерация набора данных
X, y = datasets.make_regression(n_samples=100, n_features=1, noise=11,
                                shuffle=True)

#разбиение набора данных на тренировочный и тестовый
train_size = 88
test_size = 12
X_train = X[:-train_size]
X_test = X[-test_size:]
y_train = y[:-train_size]
y_test = y[-test_size:]

# построение и обучение модели:
regr = linear_model.Ridge(alpha=.99)
regr.fit(X_train, y_train)

# определение и вывод параметров построенной модели
w0 = str(regr.intercept_)
w1 = str(regr.coef_[0])
y_func =  'y=' + w0 +('+' if w1>0 else '') + str(w1)+'x'
print('w0 = '+ w0)
print('w1 =  '+ w1)
print(y_func)

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
        % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
        % regr.score(X_test, y_test))

# вывод прогноза на основании тестового набора данных
plt.scatter(X_test, y_test, color='red', s=15)
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Рис. 7.6. Программный код алгоритма построения линейной регрессии с использованием метода наименьших квадратов с регуляризацией

Алгоритм нахождения оптимальной величины параметра регуляризации α на основе поиска минимальной ошибки прогноза (максимального значения коэффициента детерминации) на наблюдениях, не участвовавших в построении самой модели, приведены на рисунках 7.7 (программный код алгоритма) и 7.8 (результаты работы алгоритма

```
from sklearn import datasets,linear_model
import numpy as nm

# генерация набора данных
X, y = datasets.make_regression(n_samples=100, n_features=1, noise=11,
                                shuffle=True)

#разбиение набора данных на тренировочный и тестовый
train_size = 88
test_size = 12
X_train = X[:-train_size]
X_test = X[-test_size:]
y_train = y[:-train_size]
y_test = y[-test_size:]

# построение и обучение модели
# формирование и оценка прогноза
# при разных значениях коэфициента регуляризациии
maxScoreAlpha=0
maxScore=-1
maxScoreW0=0
maxScoreW1=0

for k_alpha in nm.arange(0.25, 5, 0.25):
    regr = linear_model.Ridge(alpha=k_alpha)
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    currentScore=regr.score(X_test, y_test)
    print('alpha =: %.2f' % k_alpha,'R^2 =: %.5f' % currentScore)
    if maxScore<=currentScore:
        maxScoreAlpha=k_alpha
        maxScore=currentScore
        maxScoreW0=str(regr.intercept_)
        maxScoreW1=str(regr.coef_[0])

# Вывод параметров оптимальной модели
y_func = 'y=' + maxScoreW0 +('+' if maxScoreW1>0 else '') + str(maxScoreW1)+'x'
print('alpha =: %.2f' % maxScoreAlpha)
print('w0 = '+ maxScoreW0)
print('w1 =  '+ maxScoreW1)
print(y_func)
```

Рис. 7.7. Программный код алгоритма поиска оптимального значения коэффициента регуляризации при построении линейной регрессии с использованием метода наименьших квадратов с регуляризацией

Для поиска оптимального значения коэфициента регуляризации при построении линейной регрессии с использованием метода наименьших квадратов с регуляризацией может быть использован метод гребневой регрессии со встроенной перекрестной проверкой (linear_model.RidgeCV).

```
('alpha =: 0.25', 'R^2 =: 0.97405')
('alpha =: 0.50', 'R^2 =: 0.97742')
('alpha =: 0.75', 'R^2 =: 0.97954')
('alpha =: 1.00', 'R^2 =: 0.98056')
('alpha =: 1.25', 'R^2 =: 0.98061')
('alpha =: 1.50', 'R^2 =: 0.97980')
('alpha =: 1.75', 'R^2 =: 0.97825')
('alpha =: 2.00', 'R^2 =: 0.97604')
('alpha =: 2.25', 'R^2 =: 0.97325')
('alpha =: 2.50', 'R^2 =: 0.96994')
('alpha =: 2.75', 'R^2 =: 0.96620')
('alpha =: 3.00', 'R^2 =: 0.96206')
('alpha =: 3.25', 'R^2 =: 0.95758')
('alpha =: 3.50', 'R^2 =: 0.95280')
('alpha =: 3.75', 'R^2 =: 0.94777')
('alpha =: 4.00', 'R^2 =: 0.94251')
('alpha =: 4.25', 'R^2 =: 0.93706')
('alpha =: 4.50', 'R^2 =: 0.93144')
('alpha =: 4.75', 'R^2 =: 0.92568')
alpha =: 1.25
w0 = -0.880698981511
w1 =  78.5591500931
y=-0.880698981511+78.5591500931x
```

Рис. 7.8. Результаты выполнения алгоритма поиска оптимального значения коэффициента регуляризации при построении линейной регрессии с использованием метода наименьших квадратов с регуляризацией

```
('alpha =: 0.25', 'R^2 =: 0.97405')
('alpha =: 0.50', 'R^2 =: 0.97742')
('alpha =: 0.75', 'R^2 =: 0.97954')
('alpha =: 1.00', 'R^2 =: 0.98056')
('alpha =: 1.25', 'R^2 =: 0.98061')
('alpha =: 1.50', 'R^2 =: 0.97980')
('alpha =: 1.75', 'R^2 =: 0.97825')
('alpha =: 2.00', 'R^2 =: 0.97604')
('alpha =: 2.25', 'R^2 =: 0.97325')
('alpha =: 2.50', 'R^2 =: 0.96994')
('alpha =: 2.75', 'R^2 =: 0.96620')
('alpha =: 3.00', 'R^2 =: 0.96206')
('alpha =: 3.25', 'R^2 =: 0.95758')
('alpha =: 3.50', 'R^2 =: 0.95280')
('alpha =: 3.75', 'R^2 =: 0.94777')
('alpha =: 4.00', 'R^2 =: 0.94251')
('alpha =: 4.25', 'R^2 =: 0.93706')
('alpha =: 4.50', 'R^2 =: 0.93144')
('alpha =: 4.75', 'R^2 =: 0.92568')
alpha =: 1.25
w0 = -0.880698981511
w1 =  78.5591500931
y=-0.880698981511+78.5591500931x
```

Рис. 7.9. Результаты выполнения алгоритма поиска оптимального значения коэффициента регуляризации при построении линейной регрессии методом наименьших квадратов с регуляризацией

Описание класса модели:

class sklearn.linear_model.RidgeCV(alphas=), *, fit_intercept, normalize, scoring, cv, gcv_mode, store_cv_values=False)

Основные параметры класса модели:

alphas (тип данных – одномерный массив размерностью [n_alphas]; значение по умолчанию – (0.1, 1.0, 10.0)) – указывает алгоритму возможные значения параметров регуляризации α;

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w0 (в случае fit_intercept=False w0 (intercept_ = 0);

normalize (тип данных – bool; значение по умолчанию – False) – указывает алгоритму необходимость нормализации данных;

gcv_mode (тип данных – str; значение по умолчанию – ‘auto’) – указывает алгоритму метод обобщенной перекрестной проверки (возможные варианты: {‘auto’, ‘svd’, ‘eigen’}, соответственно выбор метода автоматически, метод сингулярного разложения, метод собственного разложения.

Основные атрибуты класса модели:

cv_values_ (тип данных – трехмерный массив размерностью [n_samples, n_targets, n_alphas]) – предназначен для хранения расчета параметра регуляризации α (доступно, только если store_cv_values = True и cv = None);

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения расчитанных коэффициентов wtarget,j для задачи линейной регрессии (при наличии одной целевой функции одномерный массив размерностью [n_features];

intercept_ (тип данных – float или одномерный массив размерностью [n_targets]) – предназначен для хранения значения независимого параметра w0 или wtarget,0. При fit_intercept = False intercept_ = 0.0;

alpha_ (тип данных – float) – предназначен для хранения значения полученного параметра регуляризации α;

best_score_ (тип данных – float) – предназначен для хранения значения полученного параметра минимальной ошибки прогноза (коэффициента детерминации).

Основные методы класса модели:

fit(X, y[, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples, n_features] или [n_samples]) – зависимые переменные модели, sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода наименьших квадратов с регуляризацией для построения линейной регрессии приведен на рисунках 7.10 (результаты работы алгоритма) и 7.11 (программный код алгоритма).

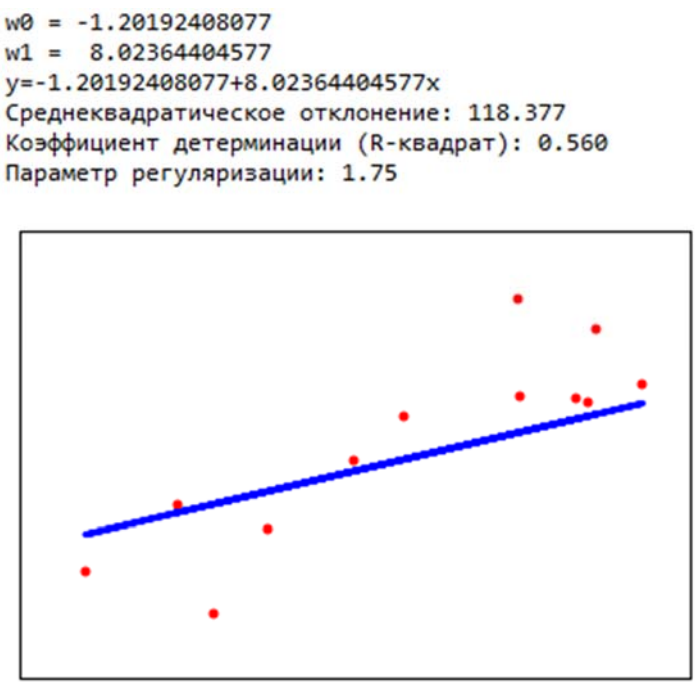


Рис. 7.10. Результаты выполнения алгоритма построения линейной регрессии с использованием метода наименьших квадратов с регуляризацией

В последние годы по причине своей простоты и эффективности очень популярным методом оптимизации в машинном обучении и управлении стал метод стохастического градиентного спуска.

```
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error
import numpy as nm

# генерация набора данных
X, y = datasets.make_regression(n_samples=100, n_features=1, noise=11,
                                shuffle=True)

#разбиение набора данных на тренировочный и тестовый
train_size = 88
test_size = 12
X_train = X[:-train_size]
X_test = X[-test_size:]
y_train = y[:-train_size]
y_test = y[-test_size:]

# построение и обучение модели:
alphas = nm.arange(0.25, 5.0, 0.25)
regr = linear_model.RidgeCV(alphas=alphas)
regr.fit(X_train, y_train)

# определение и вывод параметров построенной модели
w0 = str(regr.intercept_)
w1 = str(regr.coef_[0])
y_func =  'y=' + w0 +('+' if w1>0 else '') + str(w1)+'x'
print('w0 = '+ w0)
print('w1 =  '+ w1)
print(y_func)

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
        % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
        % regr.score(X_test, y_test))
print('Расчетный параметр регуляризации: %.2f'
         % regr.alpha_)

# вывод прогноза на основании тестового набора данных
plt.scatter(X_test, y_test, color='red', s=15)
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Рис. 7.11. Программный код алгоритма построения линейной регрессии с использованием метода наименьших квадратов с регуляризацией

Стохастический градиентный спуск – это метод итерации для оптимизации целевой функции с подходящими свойствами гладкости (например, дифференцируемость или субдифференцируемость) [1].

Данный метод можно расценивать как стохастическую аппроксимацию оптимизации методом градиентного спуска, поскольку он заменяет реальный градиент (вычисленный из полного набора данных) путём оценки такового (вычисленного из случайно выбранного подмножества данных). Такой подход является особенно востребованным в приложениях обработки больших данных, что сокращает вычислительные ресурсы, достигая более быстрые итерации в обмен на более низкую скорость сходимости.

Для построения линейной регрессии методом минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска используется класс linear_model. SGDRegressor.

Описание класса модели:

class sklearn.linear_model.SGDRegressor(loss, *, penalty, alpha, l1_ratio, fit_intercept, max_iter, tol, shuffle, verbose, epsilon1, random_state, learning_rate, eta0, power_t, early_stopping, validation_fraction, n_iter_no_change, warm_start, average)

Основные параметры класса модели:

loss (тип данных – str; значение по умолчанию – ‘squared_loss’) – указывает алгоритму вид функции потерь, который необходимо использовать (возможные варианты: {‘squared_loss’, ‘huber’, ‘epsilon_insensitive’, ‘squared_epsilon_insensitive’};

penalty (тип данных – str; значение по умолчанию – ‘l2’) – указывает алгоритму метод расчета штрафа за сложность модели, который необходимо использовать (возможные варианты: {‘l1’, ‘l2’, ‘elasticnet’};

alpha (тип данных – float; значение по умолчанию – 0.0001) – указывает алгоритму значение константы к величине штрафа за сложность модели (также при learning_rate равным ‘optimal’ используется для вычисления скорости обучения);

l1_ratio (тип данных – float; значение по умолчанию – 0.15; ограничение – параметр penalty должен быть равен ‘elasticnet’) – указывает алгоритму границу между использованием методов расчета

штрафа за сложность модели при penalty равным ‘elasticnet’ (0 <= l1_ratio <= 1, 0 соответствует штрафу l1, 1 соответствует штрафу l2)

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w0 (в случае fit_intercept=False w0 (intercept_ = 0);

max_iter (тип данных – int; значение по умолчанию – 1000) – указывает алгоритму максимальное количество проходов (итераций) по обучающим данным;

shuffle (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость перетасовки обучающих данных после каждого прохода по обучающим данным;

random_state (тип данных – int; значение по умолчанию – None; ограничение – параметр shuffle должен быть равен True) указывает алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

coef_ (тип данных – одномерный массив размерностью [n_features]) – предназначен для хранения рассчитанных коэффициентов wj для задачи линейной регрессии;

intercept_ (тип данных – float) – предназначен для хранения значения независимого параметра w0 или wtarget,0. При fit_intercept = False intercept_ = 0.0;

n_iter_ (тип данных – int) – фактическое количество итераций до достижения критерия остановки;

Основные методы класса модели:

fit(X, y[, coef_init, intercept_init, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples]) – зависимые переменные модели; coef_init (тип данных – массив размерностью [n_features]) – начальные значения коэффициентов wj для задачи линейной регрессии для быстрого старта оптимизации; intercept_init (тип данных – массив размерностью [n_features]) – начальные значения коэффициента w0 для задачи линейной регрессии для быстрого старта оптимизации; sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

partial_fit(X, y[, sample_weight]) – выполнение одного прохода метдом стохастического градиентного спуска на заданном образцах обучающем наборе;

densify() – преобразование разряженной матрицы (массива) коэффициентов (coef_) в формат плотной матрицы (разрежённая матрица – это матрица с преимущественно нулевыми элементами, в противном случае, если большая часть элементов матрицы ненулевые, матрица считается плотной);

sparsify() – преобразование плотной матрицы (массива) коэффициентов (coef_) в формат разряженной матрицы (массива);

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода наименьших квадратов с регуляризацией для построения линейной регрессии приведен на рисунках 7.12 (программный код алгоритма) и 7.13 (результаты работы алгоритма).

7.3. Нелинейные модели регрессии

Нелинейная регрессия – это вид регрессионного анализа, в котором экспериментальные данные моделируются функцией, являющейся нелинейной комбинацией параметров модели и зависящей от одной и более независимых переменных

Различают два класса нелинейных регрессий:

а) регрессии, нелинейные относительно включенных в анализ объясняющих переменных, но линейные по оцениваемым параметрам (например, полиномы различных степеней);

б) регрессии, нелинейные по оцениваемым параметрам (например, логарифмическая, показательная, степенная функции).

```python
import matplotlib.pyplot as plt
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error

# генерация набора данных
X, y = datasets.make_regression(n_samples=100, n_features=1, noise=11,
                                shuffle=True)

#разбиение набора данных на тренировочный и тестовый
train_size = 88
test_size = 12
X_train = X[:-train_size]
X_test = X[-test_size:]
y_train = y[:-train_size]
y_test = y[-test_size:]

# построение и обучение модели:
regr = linear_model.SGDRegressor(loss='squared_epsilon_insensitive',
                                 penalty ='elasticnet',
                                 alpha = 0.005)
regr.fit(X_train, y_train)

# определение и вывод параметров построенной модели
w0 = str(regr.intercept_)
w1 = str(regr.coef_[0])
y_func =  'y=' + w0 +('+' if w1>0 else '') + str(w1)+'x'
print('w0 = '+ w0)
print('w1 =  '+ w1)
print(y_func)

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
        % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
        % regr.score(X_test, y_test))

# вывод прогноза на основании тестового набора данных
plt.scatter(X_test, y_test, color='red', s=15)
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Рис. 7.12. Программный код алгоритма построения линейной регрессии с использованием метода минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска

Некоторые задачи нелинейной регрессии могут быть сведены к линейным путём подходящего преобразования формулировки модели.

```
w0 = 0.688971210934
w1 =  27.7300775811
y=0.688971210934+27.7300775811x
Среднеквадратическое отклонение: 157.672
Коэффициент детерминации (R-квадрат): 0.859
```

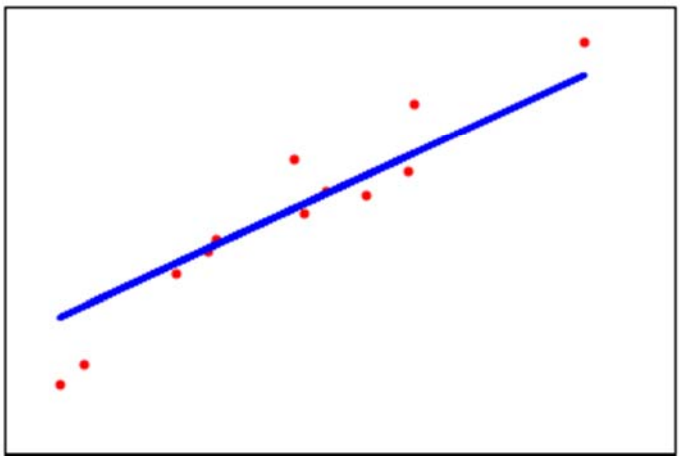


Рис. 7.13. Результаты выполнения алгоритма построения линейной регрессии с использованием метода минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска

Для того чтобы можно было использовать линейную регрессию для нелинейного набора данных необходимо преобразовать их к линейному виду (т.е провести линеаризацию). Например, двумерный набор данных, соответствующий полиному второй степеней (y=a+bx+cx2), необходимо преобразовать набор параметров [a, b, c] к следующему виду: [1, a, b, c, a^2, b^2, c^2, ab, bc, ca].

В библиотеке sklearn ряд таких преобразований реализован в специальных классах, таких как, например PolynomialFeatures, который применяется для линеаризации полиномов различных степеней и FunctionTransformer, который применяется для линеаризации степенных, показательных и логарифмических функций, основные методы которой приведены в табл. 7.1.

Таблица 7.1

Основные методы линеаризации

Функция	Пример	Иходная форма	Линейная форма
Логарифмическая	log(x)	y = A * log(B*x) + C	y = C + A * (log(B) + log(x))
Показательная	2x, ex	y = A * exp(B*x) + C	log(y-C) = log(A) + B * x
Степенная	x2	y = B * x**N + C	log(y-C) = log(B) + N * log(x)

В машинном обучении широкое применение находят параболы второй степени и полином третьего порядка. С использованием полиномов более высоких степеней существуют определенные проблемы: из-за роста изгибов кривой полинома с ростом его порядка уменьшается менее однородна совокупность по результативному признаку.

Пример использования метода наименьших квадратов для построения регрессии на основе полинома третей степеней приведен на рисунках 7.14 (результаты работы алгоритма) и 7.15 (программный код алгоритма).

```
w0 = -1.301
w1 = 0.632
w2 = -0.179
w3 = 0.228
y=-1.30107898926+0.631536720606x-0.178860267163x^2+0.228385953311x^3
Среднеквадратическое отклонение: 2.232
Коэффициент детерминации (R-квадрат): 0.811
```

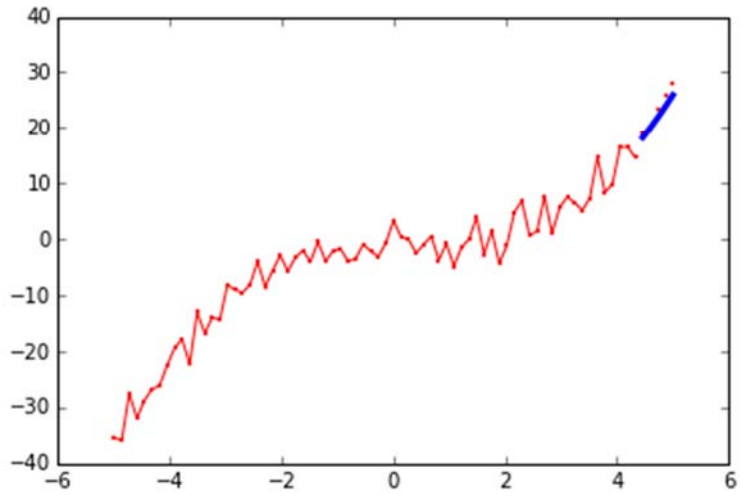


Рис. 7.14 Результаты выполнения алгоритма построения линейной регрессии на основе полинома третей степеней

Пример использования метода наименьших квадратов для построения регрессии на основе логарифмической модели приведен на рисунках 7.15 (программный код алгоритма) и 7.16 (результаты работы алгоритма).

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# генерация набора данных
X = np.linspace(-5, 5, num=75)[:, None]
y = -0.7 + 1.2*X - 0.2*X**2 +0.2*X**3 + 2.5*np.random.randn(75, 1)

# приведение полиноминальной модели к линейному виду:
poly = PolynomialFeatures(degree=3, include_bias=False)
X_new = poly.fit_transform(X)

#разбиение набора данных на тренировочный и тестовый
train_size = 70
test_size = 5
X_train = X[:train_size]
y_train = y[:train_size]
X_test = X[-test_size:]
y_test = y[-test_size:]
X_train_poly = X_new[:train_size]
X_test_poly = X_new[-test_size:]

# вывод тренировочного набора данных
plt.plot(X_train, y_train, color='red',linewidth=1)

# построение и обучение модели:
regr = LinearRegression()
regr.fit(X_train_poly, y_train)

# определение и вывод параметров построенной модели
w0 = regr.intercept_[0]
w1 = regr.coef_[0,0]
w2 = regr.coef_[0,1]
w3 = regr.coef_[0,2]
print('w0 = %.3f'  % w0)
print('w1 = %.3f'  % w1)
print('w2 = %.3f'  % w2)
print('w3 = %.3f'  % w3)
y_func =  'y=' + str(w0) +('+' if w1>0 else '')
y_func =  y_func + str(w1)+'x'+ ('+' if w2>0 else '') + str(w2)
y_func =  y_func +'x^2'+('+' if w3>0 else '') + str(w3)+'x^3'
print(y_func)

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test_poly)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
        % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
        % r2_score(y_test, y_pred))

# вывод прогноза на основании тестового набора данных
plt.scatter(X, y, color='red', s=1)
plt.plot(X_test, y_pred, color='blue', linewidth=3)
```

Рис. 7.15. Программный код алгоритма построения линейной регрессии на основе полинома третей степеней

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# генерация набора данных
X = np.linspace(1, 6, num=100)[:, None]
y= 2.5 * np.log(1.2 * X) + 5.7 + 0.2*np.random.randn(100, 1)

# приведение логарифмической модели к линейному виду:
transformer = FunctionTransformer(func=np.log, validate=True)
X_new = transformer.fit_transform(X)

#разбиение набора данных на тренировочный и тестовый
train_size = 80
test_size = 20
X_train = X[:train_size]
y_train = y[:train_size]
X_test = X[-test_size:]
y_test = y[-test_size:]
X_train_poly = X_new[:train_size]
X_test_poly = X_new[-test_size:]

# вывод тренировочного набора данных
plt.plot(X_train, y_train, color='red',linewidth=1)

# построение и обучение модели:
regr = LinearRegression()
regr.fit(X_train_poly, y_train)

# определение и вывод параметров построенной модели
w0 = regr.intercept_
w1 = regr.coef_[0]
print('w0 = %.3f'  % w0)
print('w1 = %.3f'  % w1)

print('y= %.3f * log(x)+ %.3f' % (w1, w0))

# формирование прогноза на основании тестового набора данных
y_pred = regr.predict(X_test_poly)

# Оценка прогноза
print('Среднеквадратическое отклонение: %.3f'
        % mean_squared_error(y_test, y_pred))
print('Коэффициент детерминации (R-квадрат): %.3f'
        % r2_score(y_test, y_pred))

# вывод прогноза на основании тестового набора данных
plt.scatter(X, y, color='red', s=1)
plt.plot(X_test, y_pred, color='blue', linewidth=3)
```

Рис. 7.16. Программный код алгоритма построения линейной регрессии на основе логарифмической функции

```
w0 = 6.062
w1 = 2.559
y= 2.559 * log(x)+ 6.062
Среднеквадратическое отклонение: 0.064
Коэффициент детерминации (R-квадрат): 0.257
```

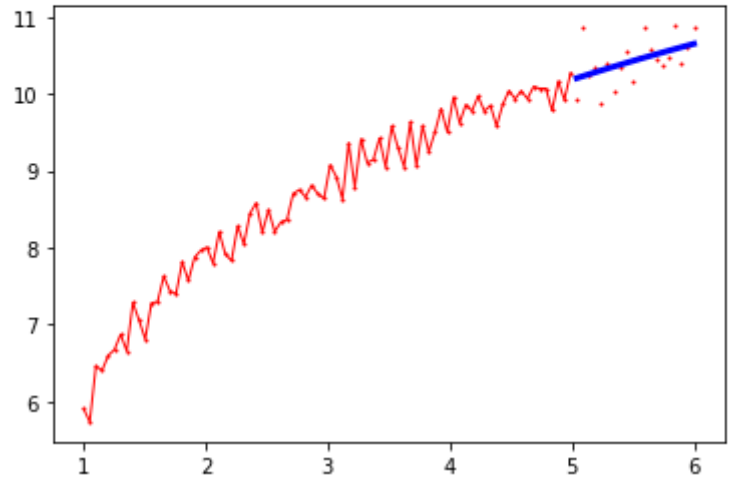


Рис. 7.17. Результаты выполнения алгоритма построения линейной регрессии на основе логарифмической функции

Вопросы для обсуждения

1. Проблемы практического применения регрессионных моделей
2. Методы отбора признаков в задачах линейной регрессии
3. Значимость коэффициентов линейной регрессии
4. Методы борьба с переобучением в различных методах линейной регрессии
5. Вероятностная постановка задач регрессии
6. Взаимосвязь между сложностью регрессионной модели и размером набора данных
7. Вычисление оптимальных параметров логистической регрессии
8. Проблема определения оптимальных параметров линейной регрессии
9. Метрики качества регрессии

Практические задания

Задание 7.1.

На основании учебных данных о стоимости жилья в г. Бостоне (sklearn.datasets.load_boston) необходимо предсказать стоимость дома со следующими характеристиками: CRIM=0.05; ZN=0; INDUS=11.9; CHAS=0; NOX=0.6; RM=6.1; AGE=80; DIS=2.5; RAD=1; TAX=3; PTRATIO=21; B=397; LSTAT=7.98.

Задание 7.2

На основании набора данных, полученного при решении задания 5, приведенного в предыдущей теме, проведите исследование и сравнительные эксперименты для следующих методов построения моделей регрессии (по вариантам):

1) наименьших квадратов;

2) наименьших квадратов с регуляризацией;

3) риджевой регрессии со встроенной перекрестной проверкой;

4) минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска.

При проведении экспериментов использовать различные допустимые комбинации параметры методов. Сделайте выводы.

Задание 7.3

На основании данных о производстве конфет в США с января 1972 по настоящий момент необходимо спрогнозировать индустриальный продуктовый индекс (IPG3113N – универсальный индекс уровня производства, который измеряется как % от уровня производства 2012 года).

Горизонт прогнозирования выбран: временной интервал в 24 месяца.

Источник данных: https://fred.stlouisfed.org/series/IPG3113N

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать метод построения модели регрессии (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование модели регрессии с использованием различных комбинаций её параметров;

7) оценить качество различных по параметрам вариантов модели по выбранной метрике;

8) спрогнозировать целевой показатель на основании варианта модели с лучшим значением метрики на заданном горизонте.

Задание 7.4.

На основании данных о розничных продаж в электронной торговле США необходимо спрогнозировать индекс ECOMPCTSA (доля розничных продаж в в электронной торговле в общем объеме продаж).

Горизонт прогнозирования выбран: временной интервал в 12 месяцев

Источник данных: https://fred.stlouisfed.org/series/ECOMPCTSA

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать несколько методов построения модели регрессии (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование моделей регрессии с использованием различных комбинаций их параметров;

7) оценить качество различных по параметрам вариантов моделей по выбранной метрике;

8) спрогнозировать целевой показатель на основании варианта модели с лучшим значением метрики на заданном горизонте.

Задание 7.5.

На основании данных о рынке недвижимости США спрогнозировать среднюю стоимость новых домов.

Горизонт прогнозирования выбран: временной интервал в 24 месяца

Источник данных: https://fred.stlouisfed.org/series/MSPUS

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать несколько методов построения модели регрессии (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование моделей регрессии с использованием различных комбинаций их параметров;

7) оценить качество различных по параметрам вариантов моделей по выбранной метрике;

8) спрогнозировать целевой показатель на основании варианта модели с лучшим значением метрики на заданном горизонте.

Задание 7.6.

На основании данных о первичном рынке недвижимости любого региона Российской Федерации спрогнозировать среднюю стоимость 1 м2 общей площади квартир.

Горизонт прогнозирования выбран: временной интервал в 24 месяца

Источник данных: https://showdata.gks.ru/report/277332/?

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать несколько методов построения модели регрессии (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование моделей регрессии с использованием различных комбинаций их параметров;

7) оценить качество различных по параметрам вариантов моделей по выбранной метрике;

8) спрогнозировать целевой показатель на основании варианта модели с лучшим значением метрики на заданном горизонте.

Задание 7.7.

На основании данных о вторичном рынке недвижимости любого региона (или города) Российской Федерации предсказать стоимость самостоятельно выбранного объекта недвижимости (квартиры, загородного дома, земли).

Источник данных: любая база данных объявлений о продажи объектов недвижимости, размещенная в сети Интернет.

Необходимо:

1) сформировать набор данных на основании любой базы данных объявлений о продажи объектов недвижимости, размещенной в сети Интернет;

2) провести исследование данных;

3) выполнить предварительную обработку данных (особое внимание уделить наличию дубликатов, неполным сведениям и неинформативным признакам);

4) выбрать несколько методов построения модели регрессии (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование моделей регрессии с использованием различных комбинаций их параметров;

7) оценить качество различных по параметрам вариантов моделей по выбранной метрике;

8) спрогнозировать целевой показатель на основании варианта модели с лучшим значением метрики на заданном горизонте.

Библиографический список

1. Стохастический градиентный спуск // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Стохастический_градиентный_спуск

2. Замятин А. В. Введение в интеллектуальный анализ данных: учебное пособие / А. В. Замятин; Нац. исслед. Том. гос. ун-т. - Томск: Издательский Дом Томского государственного университета, 2016. URL: http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000529594

3. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

4. Лесковец, Ю. Анализ больших наборов данных / Лесковец Ю., Раджараман А. , Джеффри Д. Ульман - Москва : ДМК Пресс, 2016. - 498 с. - ISBN 978-5-97060-190-7. - Текст : электронный // ЭБС "Консультант студента": [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN9785970601907.html

5. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

6. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

7. Машинное обучение // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Машинное_обучение

Глава 8. SCIKIT-LEARN: МОДЕЛИ КЛАССИФИКАЦИИ

В данной главе рассматриваются следующие вопросы:
1. Классификация как задача машинного обучения
2. Линейные модели классификации
3. Модели классификации с нелинейными разделяющими поверхностями

8.1. Классификация как задача машинного обучения

Классификация – это область машинного обучения с учителем, которая пытается предсказать, к какому классу или категории принадлежит тот или иной объект, на основе его характеристик.

Например, анализ сотрудников компании с целью установить взаимосвязь между различными их характеристиками, такими как уровень образования, количество лет на текущей должности, возраст, зарплата, шансы на продвижение по службе и т. д. является задачей классификации.

Характеристики (или переменные) в рамках проводимого анализа могут принимать одну из двух форм:

1) либо они являются независимыми от других (входные данные, предикторы);

2) либо они являются зависимыми от других характеристик (выходные данные, отклики).

В приведенном выше примере, при анализе сотрудников, можно предположить, что уровень образования, время на текущей должности и возраст являются взаимнонезависимыми и их необходимо рассматривать как входные данные. Заработная плата и шансы на продвижение по службе могут быть результатами, которые зависят от входных данных.

Как правило, классификацию можно разбить на две области:

а) бинарная классификация, в которой требуется сгруппировать результат в одну из двух групп (0 или 1, истинное или ложное, положительное или отрицательное);

б) мультиклассовая классификация, где требуется сгруппировать результат в одну из нескольких (более двух) групп.

Область применения моделей классификации: выявление спама, разделение положительных и отрицательных комментариев (с точки зрения различных критериев) в социальных сетях и на торговых интернет-площадках, определение кредитного рейтинга клиентов банка, а также распознавание изображений и т.д.

Далее основное внимание будет уделено использованию различных алгоритмов классификации без детального рассмотрения теории, которая лежит в их основе. Для целей классификации будем использовать библиотеки Python scikit-learn для моделей построения машинного обучения.

8.2. Линейные модели классификации

В области машинного обучения целью статистической классификации является использование характеристик объекта для определения того, к какому классу (или группе) он принадлежит. Линейный классификатор достигает этого путем принятия решения классификации, основанным на значении в линейной комбинации характеристик. Иными словами, линейный классификатор – способ решения задач классификации, когда решение принимается на основании линейного оператора над входными данными [1].

Общее уравнение для линейной модели классификации выглядит следующим образом:

$$y = f(\vec{\omega} \cdot \vec{x}) = f(\omega_0 + \sum_j \omega_j \cdot x_j)$$

где $\vec{x}$ – вектор входных независимых параметров модели; $\vec{\omega}$ – вектор весов, значения которых определяются в ходе машинного обучения на подготовленных образцах.

При этом функция f либо простая пороговая функция, отделяющая один класс объектов от другого, либо имеет смысл вероятности того или иного решения.

Операция бинарной линейной классификации может быть представлена как отображение объектов в многомерном пространстве на гиперплоскость, в которой те объекты, которые попали по одну сто-

рону разделяющей линии, относятся к первому классу («да»), а объекты по другую сторону - ко второму классу («нет»)).

В Scikit-learn для работы с линейными моделями классификации включен класс linear_model, который предоставляет возможность работы со следующими методами построения линейной классификации:

а) методом логистической регрессии (linear_model. LogisticRegression);

б) методом логистической регрессии со встроенной перекрестной проверкой (linear_model.LogisticRegressionCV);

в) методом риджевой регрессии (linear_model.RidgeClassifier);

г) методом риджевой регрессии со встроенной перекрестной проверкой (linear_model.RidgeClassifierCV);

д) методом минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска (linear_model. SGDClassifier).

е) методом, использующим персептроны (linear_model. Perceptron);

д) методом, использующим пассивно-агрессивные алгоритмы классификации (linear_model.Passive-AggressiveClassifier).

Рассмотрим несколько методов поподробнее.

Логистическая регрессия – это фундаментальный метод классификации, который применяется в основном для решения бинарных задач. Он принадлежит к группе линейных классификаторов и чем-то похож на полиномиальную и линейную регрессию. Логистическая регрессия выполняется быстро, она относительно не сложна и удобна для интерпретации результатов. Логистическая регрессия требует довольно больших размеров выборки.

Логистическая регрессия имеет следующие допущения: 1) для двоичной логистической регрессии требуется, чтобы зависимая переменная была двоичной; 2) для бинарной регрессии значение зависимой переменной равное 1 должно представлять желаемый результат; 3) в модель следует включать только значимые переменные; 4) входные переменные должны быть независимыми друг от друга

Для построения линейной классификации методом логистической регрессии используется класс LogisticRegression

Описание класса модели:

class sklearn.linear_model.LogisticRegression(penalty, *, dual, tol, C, fit_intercept, intercept_scaling, class_weight, random_state, solver, max_iter, multi_class, verbose, warm_start, n_jobse, l1_ratio)

Основные параметры класса модели:

penalty (тип данных – str; значение по умолчанию – ‘l2’) – указывает алгоритму метод расчета штрафа за сложность модели, который необходимо использовать (возможные варианты: {‘l1’, ‘l2’, ‘elasticnet’, ‘none’};

dual (тип данных – bool; значение по умолчанию – False) – указывает алгоритму тип задачи оптимизации (прямая или дуальная);

tol (тип данных – int; значение по умолчанию – 0.001) – указывает алгоритму точность решения;

C (тип данных – float; значение по умолчанию – 1.0, ограничение – положительное значение) – параметр регуляризации (меньшее значение обеспечивает более сильную регуляризацию);

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w_0 (в случае fit_intercept=False w_0 (intercept_ = 0);

intercept_scaling (тип данных – float; значение по умолчанию – 1) – указывает алгоритму коэффициент масштабирования выходных параметров (применяется только при solver = ‘liblinear’);

solver (тип данных – str; значение по умолчанию – ‘'lbfgs') – указывает алгоритму метод решения (возможные варианты: {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, для небольших наборов данных быстрее работают алгоритмы ’liblinear’, ’sag’; для многоклассовых задач – ‘newton-cg’ и ‘lbfgs’, которые поддерживают только L2-регуляризацию и работают быстрее.

class_weight (тип данных – str или ‘balanced’; значение по умолчанию – None) указывает алгоритму соответствие определенных весов определенным классам выходных переменных (запись осуществляется по форме: {class_label: weight}), если параметр не указан, все веса предполагаются равными 1, а если указан параметр ’balanced’, то осуществляется автоматическая регулировка весов, обратно пропор-

циональных частотам классов во входных данных следующим образом: n_samples / (n_classes * np.bincount(y);

max_iter (тип данных – int; значение по умолчанию – None) – указывает алгоритму максимальное количество итераций, None соответсвует значению 1000;

random_state (тип данных – int; значение по умолчанию – None; ограничение – solver={‘sag’|‘saga’}) указывает алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

classes_ (тип данных – двухмерный массив размерностью [n_classes,]) – список меток классов, известных классификатору;

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features], для бинарной классификации – [1, n_features]) – предназначен для хранения рассчитанных коэффициентов $w_{target,j}$ для задачи линейной классификации;

intercept_ (тип данных – двумерный массив размерностью [n_classes,] или [1,] – для бинарной классификации) – предназначен для хранения значения независимого параметра w_0 или $w_{target,0}$. При fit_intercept = False intercept_ = 0.0.

Основные методы класса модели:

fit(X, y, sample_weight) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples,]) – зависимые переменные модели, sample_weight (тип данных – массив размерностью или [n_samples,]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1);

densify() – преобразовывает матрицу коэффициентов в формат плотного массива;

sparsify() – преобразовывает матрицу коэффициентов в разреженный формат.

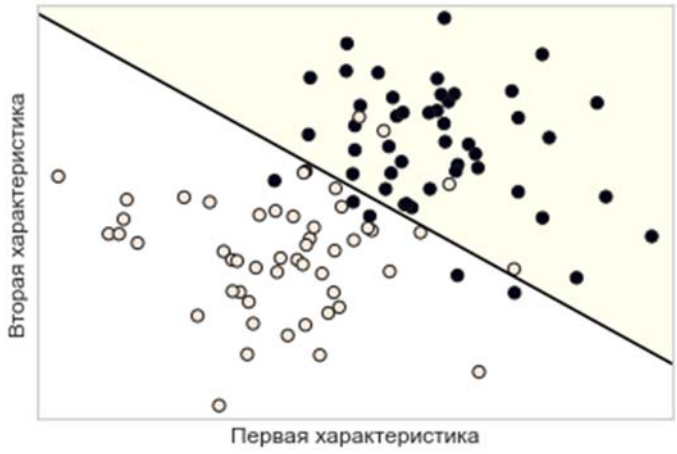
decision_function(X) – прогнозирует оценки достоверности для образцов, т.е расстояние со знаком от этого образца до гиперплоскости (возвращает массив (n_samples,) для бинарной классификации или массив (n_samples, n_classes) для многоклассовой).

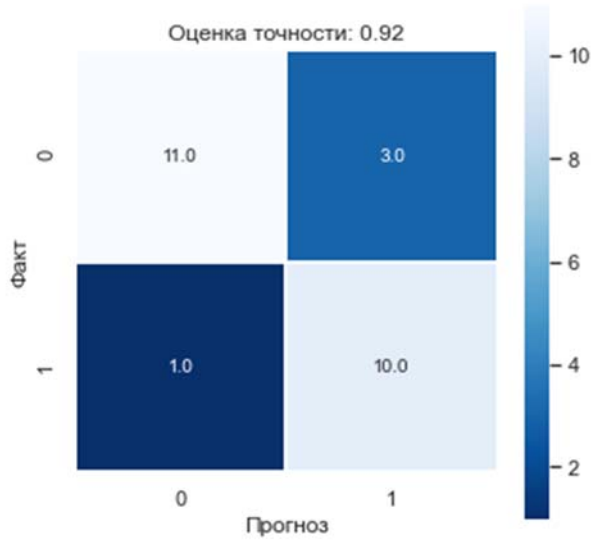
Пример использования метода логистической регрессии для линейной классификации приведен на рисунках 8.1 (программный код алгоритма), 8.2 (результаты работы алгоритма) и 8.3 (программный код визуализации модели).

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
# краткая форма - print (confusion_matrix(y_test, y_pred))
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=1.9, square = True,
            cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.1. Программный код алгоритма построения линейной классификации с использованием метода линейной регресии

Для иллюстрации работы алгоритмов построения классификации мы для наглядности использовали только два параметра и только бинарную классификацию. Однако, всё, что выше изложено применимо как для построения классификации для наборов данных с несколькими независимыми параметрами, так и для мультиклассовой классификации.





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.92	0.79	0.85	14
1	0.77	0.91	0.83	11
accuracy			0.84	25
macro avg	0.84	0.85	0.84	25
weighted avg	0.85	0.84	0.84	25

Рис. 8.2. Результаты выполнения алгоритма линейной классификации с использованием метода линейной регресии

```
def plot_2d_separator(classifier, X, fill=False, line=True, ax=None, eps=None):
    if eps is None:
        eps = 1.0 #X.std() / 2.
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy = np.linspace(y_min, y_max, 100)
    X1, X2 = np.meshgrid(xx, yy)
        X_grid = np.c_[X1.ravel(), X2.ravel()]
    try:
        decision_values = classifier.decision_function(X_grid)
        levels = [0]
        fill_levels = [decision_values.min(), 0, decision_values.max()]
    except AttributeError:
        # no decision_function
        decision_values = classifier.predict_proba(X_grid)[:, 1]
        levels = [.5]
        fill_levels = [0, .5, 1]
    if ax is None:
        ax = plt.gca()
    if fill:
        ax.contourf(X1, X2, decision_values.reshape(X1.shape),
        levels=fill_levels, colors=['ivory', 'white'])
    if line:
        ax.contour(X1, X2, decision_values.reshape(X1.shape), levels=levels,
        colors="black")
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())
```

Рис. 8.3. Программный код алгоритма визуализации модели

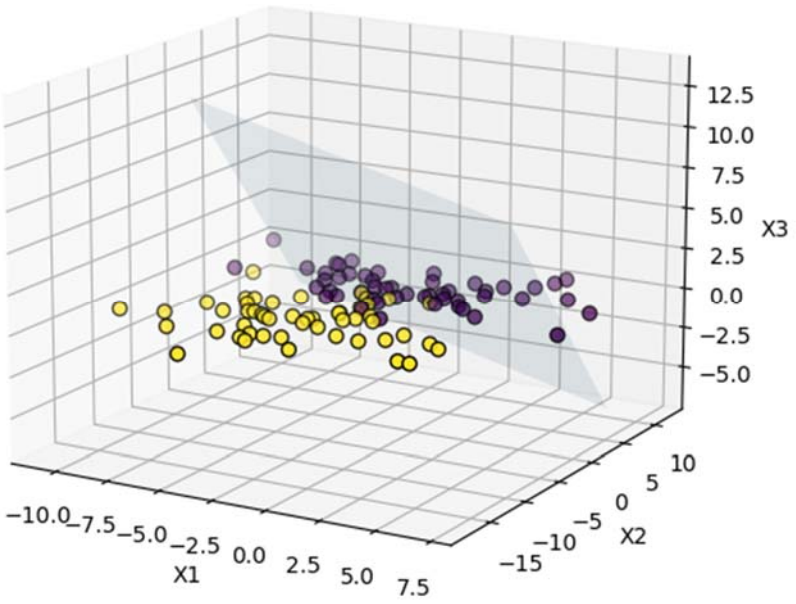
Пример использования метода логистической регрессии для множественной линейной классификации (по трех независимым параметрам) приведен на рисунках 8.4 (программный код алгоритма) и 8.5 (результаты работы алгоритма).

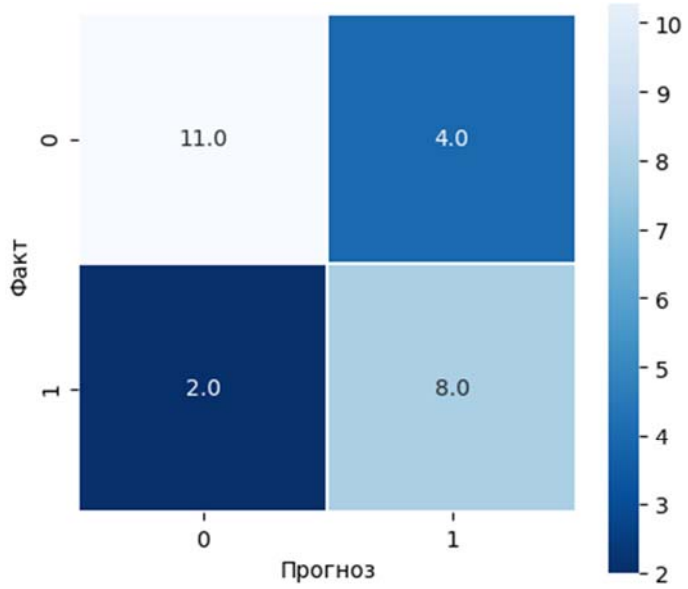
Пример использования метода логистической регрессии для мультиклассовой линейной классификации (по трех классам) приведен на рисунках 8.6 (программный код алгоритма) и 8.7 (результаты работы алгоритма).

Метод логистической регрессии, используемый для проведения линеной классификации, с целью повышения скорости получения результата может быть дополнен алгоритмом встроенной перекрестной проверки.

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_features=3,centers=2, random_state=1,cluster_std=4)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
prediction = classifier.predict(X_test)

# визуализация модели
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure(1,figsize=(7, 7))
ax = Axes3D(fig, elev=15, azim=-60)
ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("X3")
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=70, edgecolor='k')
w0=classifier.intercept_
w1=classifier.coef_[0,0]
w2=classifier.coef_[0,1]
w3=classifier.coef_[0,2]
xs = np.tile(np.arange(16)-8, (1,1))
ys = np.tile(np.arange(16)-8, (1,1)).T
zs = (w0+xs*w1+ys*w2)/w3
ax.plot_surface(xs,ys,zs, alpha=0.1)

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(2,figsize=(7,7))
sns.heatmap(cm, annot=True,
            fmt=".1f",
            linewidths=0.9,
            square = True,
            cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_test, y_test)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.4. Программный код алгоритма построения множественной линейной классификации с использованием метода линейной регресии (окончание)





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.85	0.73	0.79	15
1	0.67	0.80	0.73	10
accuracy			0.76	25
macro avg	0.76	0.77	0.76	25
weighted avg	0.77	0.76	0.76	25

Рис. 8.5. Результаты выполнения алгоритма множественной линейной классификации с использованием метода линейной регресии

```
# генерация набора данных (3 параметра)
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:, :2]
Y = iris.target

# обучение модели и прогнозирование
import numpy as np
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X, Y)
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))
Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])

# визуализация модели
from sklearn.linear_model import LogisticRegression
Z = Z.reshape(xx.shape)
plt.figure(figsize=(7, 7))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Pastel2)
plt.scatter(X[:, 0], X[:, 1], c=Y,  s=90, edgecolors='k', cmap=plt.cm.ocean)
plt.xlabel('X1')
plt.ylabel('X2')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.show()
```

Рис. 8.6. Программный код алгоритма построения мультиклассовой линейной классификации с использованием метода линейной регресии

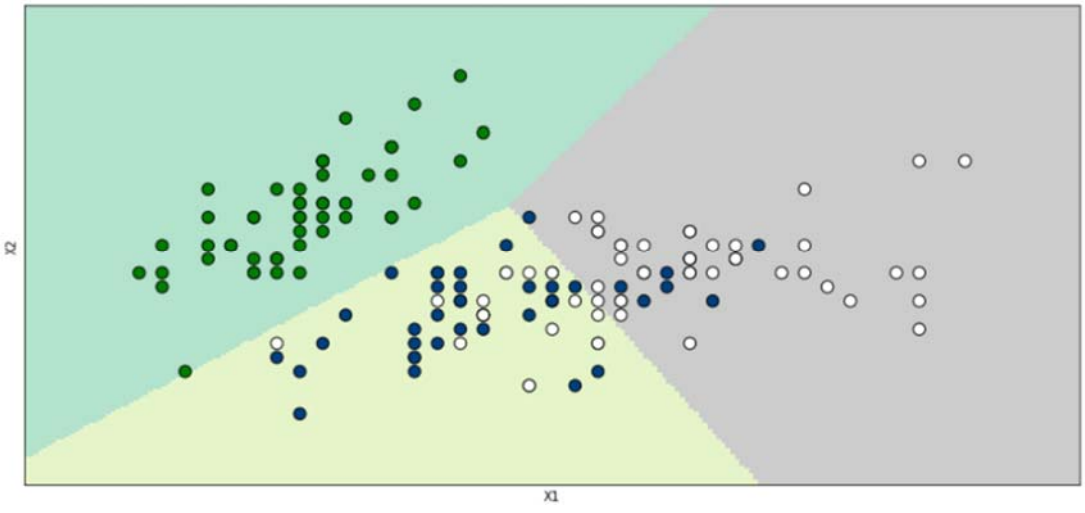


Рис. 8.7. Результаты выполнения алгоритма мультиклассовой линейной классификации с использованием метода линейной регресии

Перекрёстная проверка – метод оценки аналитической модели и её поведения на независимых данных. При оценке модели имеющиеся

в наличии данные разбиваются на k частей. Затем на k−1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных

Для построения линейной классификации методом логистической регрессии с перекрёстной проверкой используется класс LogisticRegressionCV.

Описание класса модели:

class sklearn.linear_model.LogisticRegressionCV(*, Cs, fit_intercept, cv, dual, penalty, scoring, solver, tol, max_iter, class_weight, n_jobs, verbose, refit=True, intercept_scaling, multi_class, random_state, l1_ratios)

Основные параметры класса модели, которые отличают её от класса LogisticRegressionCV являются:

Cs (тип данных – int; значение по умолчанию – 10) – каждое из значений данного параметра описывает обратную силу регуляризации (меньшие значения указывают на более сильную регуляризацию);

cv (тип данных – int (или sklearn.model_selection); значение по умолчанию – None) – указывает алгоритму перекрестной проверки количество блоков, на которые необходимо разбить исходный набор данных (по умолчанию – model_selection.StratifiedKFold, в это случае cv – количество блоков разбиения);

Пример использования метода логистической регрессии с перекрёстной проверкой для линейной классификации приведен на рисунках 8.8 (программный код алгоритма) и 8.2 (результаты работы алгоритма аналогичны применению класса LogisticRegressionCV).

Для построения линейной классификации методом гребневой регрессии используется класс RidgeClassifier. Данный классификатор сначала преобразует целевые значения в {-1, 1}, а затем рассматривает проблему классификации как задачу регрессии, особенности которой были описаны в предыдущей главе

```
 # генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import LogisticRegressionCV
classifier = LogisticRegressionCV()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=1.9,
            square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.8. Программный код алгоритма построения линейной классификации с перекрестной проверкой с использованием метода линейной регресии

Описание класса модели:

class sklearn.linear_model.RidgeClassifier (alpha, *, fit_intercepte, normalize, copy_X, max_iter, tol, class_weight, solver, random_state)

Основные параметры класса модели:

alpha (тип данных – float или одномерный массив размерностью [n_targets]; значение по умолчанию – 1.0; ограничение – больше 0) – указывает алгоритму значение параметра регуляризации α (величину штрафа за сложность модели);

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w_0 (в случае fit_intercept=False w_0 (intercept_ = 0);

normalize (тип данных – bool; значение по умолчанию – False) – указывает алгоритму необходимость нормализации данных;

copy_X (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость копирования данных без их перезаписи;

max_iter (тип данных – int; значение по умолчанию – None) – указывает алгоритму максимальное количество итераций, None соответствует значению 1000;

tol (тип данных – int; значение по умолчанию – 0.001)) – указывает алгоритму точность решения;

solver (тип данных – str; значение по умолчанию – ‘auto’) – указывает алгоритму метод решения (возможные варианты: {‘auto’, ‘svd’, ‘cholesky’, ‘lsqr’, ‘sparse_cg’, ‘sag’, ‘saga’}, например, при значении параметра ‘sag’ алгоритм использует стохастический средний градиентный спуск), при установке ‘auto’ метод выбирается автоматически в зависимости от типа данных;

random_state (тип данных – int; значение по умолчанию – None; ограничение – solver={‘sag’ | ‘saga’}) указывает алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения рассчитанных коэффициентов $w_{target,j}$ для задачи линейной регрессии (при наличии одной целевой функции одномерный массив размерностью [n_features];

intercept_ (тип данных – float или одномерный массив размерностью [n_targets]) – предназначен для хранения значения независимого параметра w_0 или $w_{target,0}$. При fit_intercept = False intercept_ = 0.0.

Основные методы класса модели:

fit(X, y[, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples, n_features] или [n_samples]) – зависимые переменные модели, sample_weight (тип

данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода гребневой регрессии для линейной классификации приведен на рис. 8.9 (программный код алгоритма) и 8.10 (фрагмент результата работы алгоритма).

Для нахождения оптимальной величины параметра регуляризации α на основе поиска минимальной ошибки прогноза (максимального значения коэффициента детерминации) на наблюдениях, не участвовавших в построении самой модели, может быть использован алгоритм аналогичный тому, программный код которого приведен на рис. 13.7.

Для поиска оптимального значения коэффициента регуляризации при линейной классификации с использованием гребневой регрессии может быть использован метод гребневой регрессии со встроенной перекрестной проверкой (linear_model.RidgeClassifierCV).

Описание класса модели:

class sklearn.linear_model.RidgeClassifierCV(alphas, fit_intercept, normalize, scoring, cv, class_weight, store_cv_values)

alphas (тип данных – одномерный массив размерностью [n_alphas]; значение по умолчанию – (0.1, 1.0, 10.0)) – указывает алгоритму возможные значения параметров регуляризации α;

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w_0 (в случае fit_intercept=False w_0 (intercept_ = 0);

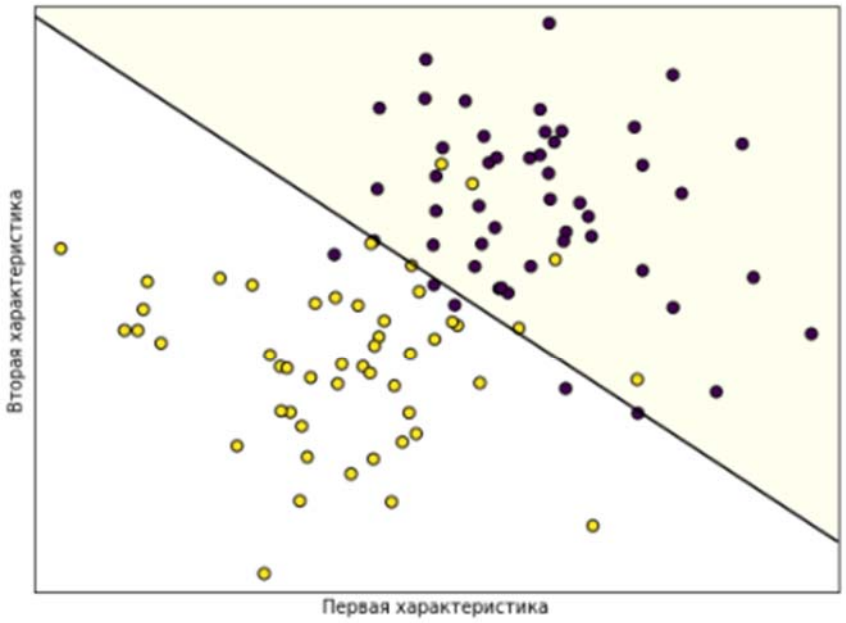
cv (тип данных – int (или sklearn.model_selection); значение по умолчанию – None) – указывает алгоритму перекрестной проверки количество блоков, на которые необходимо разбить исходный набор данных (по умолчанию – model_selection.StratifiedKFold, в это случае cv – количество блоков разбиения);

normalize (тип данных – bool; значение по умолчанию – False) – указывает алгоритму необходимость нормализации данных;

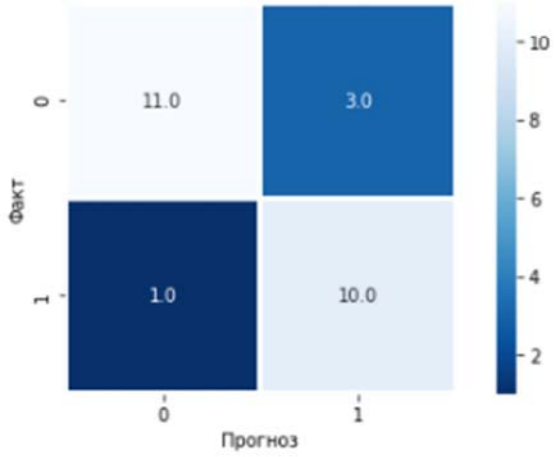
gcv_mode (тип данных – str; значение по умолчанию – ‘auto’) – указывает алгоритму метод обобщенной перекрестной проверки (возможные варианты: {‘auto’, ‘svd’, ‘eigen’}, соответственно выбор метода автоматически, метод сингулярного разложения, метод собственного разложения.

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import RidgeClassifier
classifier = RidgeClassifier(alpha=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
plt.figure(figsize=(8,6))
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,4))
sns.heatmap(cm, annot=True, fmt=".1f",
            linewidths=1.9, square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_train, y_train)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.9. Программный код алгоритма линейной классификации с использованием метода гребневой регрессии



Оценка точности: 0.9066666666666666



Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.92	0.79	0.85	14
1	0.77	0.91	0.83	11
accuracy			0.84	25
macro avg	0.84	0.85	0.84	25
weighted avg	0.85	0.84	0.84	25

Рис. 8.10. Результаты выполнения алгоритма построения линейной регрессии с использованием метода метод гребневой регрессии

Основные атрибуты класса модели:

cv_values_ (тип данных – трехмерный массив размерностью [n_samples, n_targets, n_alphas]) – предназначен для хранения расчета

параметра регуляризации α (доступно, только если store_cv_values = True и cv = None);

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения рассчитанных коэффициентов $w_{target,j}$ для задачи линейной регрессии (при наличии одной целевой функции одномерный массив размерностью [n_features];

intercept_ (тип данных – float или одномерный массив размерностью [n_targets]) – предназначен для хранения значения независимого параметра w_0 или $w_{target,0}$. При fit_intercept = False intercept_ = 0.0;

alpha_ (тип данных – float) – предназначен для хранения значения полученного параметра регуляризации α;

best_score_ (тип данных – float) – предназначен для хранения значения полученного параметра минимальной ошибки прогноза (коэффициента детерминации).

Основные методы класса модели:

fit(X, y[, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples, n_features] или [n_samples]) – зависимые переменные модели, sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода гребневой регрессии со встроенной перекрестной проверкой для линейной классификации приведен на рисунках 8.11 (программный код алгоритма) и 8.10 (результаты работы алгоритма аналогичны применению класса RidgeCV).

Для линейной классификации методом минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска используется класс linear_model. SGDClassifier.

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import RidgeClassifierCV
classifier = RidgeClassifierCV(alphas=[1e-3, 1e-2, 1e-1, 1])
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
plt.figure(figsize=(8,6))
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,4))
sns.heatmap(cm, annot=True, fmt=".1f",
            linewidths=1.9, square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_train, y_train)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.11. Программный код алгоритма построения линейной классификации с использованием метода гребневой регрессии со встроенной перекрестной проверкой

Описание класса модели:

class sklearn.linear_model.SGDClassifier (loss, *, penalty, alpha, l1_ratio, fit_intercept, max_iter, tol, shuffle, verbose, epsilon, n_jobs, random_state, learning_rate, eta0, power_t, early_stopping, validation_fraction, n_iter_no_change, class_weight, warm_start, average)

Основные параметры класса модели:

loss (тип данных – str; значение по умолчанию – ‘hinge’) – указывает алгоритму вид функции потерь, который необходимо использовать (возможные варианты: { ‘hinge’, ‘log’, ‘modified_huber’, ‘squared_hinge’, ‘perceptron’, ‘squared_loss’, ‘epsilon_insensitive’, ‘huber’, ‘squared_epsilon_insensitive’};

penalty (тип данных – str; значение по умолчанию – ‘l2’) – указывает алгоритму метод расчета штрафа за сложность модели, который необходимо использовать (возможные варианты: {‘l1’, ‘l2’, ‘elasticnet’};

alpha (тип данных – float; значение по умолчанию – 0.0001) – указывает алгоритму значение константы к величине штрафа за сложность модели (также при learning_rate равным ‘optimal’ используется для вычисления скорости обучения);

l1_ratio (тип данных – float; значение по умолчанию – 0.15; ограничение – параметр penalty должен быть равен ‘elasticnet’) – указывает алгоритму границу между использованием методов расчета штрафа за сложность модели при penalty равным ‘elasticnet’ (0 <= l1_ratio <= 1, 0 соответствует штрафу l1, 1 соответствует штрафу l2)

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость включения в модель независимого параметра w0 (в случае fit_intercept=False w0 (intercept_ = 0));

max_iter (тип данных – int; значение по умолчанию – 1000) – указывает алгоритму максимальное количество проходов (итераций) по обучающим данным;

shuffle (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость перетасовки обучающих данных после каждого прохода по обучающим данным;

random_state (тип данных – int; значение по умолчанию – None; ограничение – параметр shuffle должен быть равен True) указывает алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

coef_ (тип данных – одномерный массив размерностью [n_features]) – предназначен для хранения рассчитанных коэффициентов w_j для задачи линейной регрессии;

intercept_ (тип данных – float) – предназначен для хранения значения независимого параметра w_0 или $w_{target,0}$. При fit_intercept = False intercept_ = 0.0;

n_iter_ (тип данных – int) – фактическое количество итераций до достижения критерия остановки;

Основные методы класса модели:

fit(X, y[, coef_init, intercept_init, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples]) – зависимые переменные модели; coef_init (тип данных – массив размерностью [n_features]) – начальные значения коэффициентов w_j для задачи линейной регрессии для быстрого старта оптимизации; intercept_init (тип данных –массив размерностью [n_features]) – начальные значения коэффициента w_0 для задачи линейной регрессии для быстрого старта оптимизации; sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

partial_fit(X, y[, sample_weight]) – выполнение одного прохода метдом стохастического градиентного спуска на заданном образцах обучающем наборе;

densify() – преобразование разряженной матрицы (массива) коэффициентов (coef_) в формат плотной матрицы (разрежённая матрица – это матрица с преимущественно нулевыми элементами, в противном случае, если большая часть элементов матрицы ненулевые, матрица считается плотной);

sparsify() – преобразование плотной матрицы (массива) коэффициентов (coef_) в формат разряженной матрицы (массива);

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спус-

ка для линейной классификации приведен на рисунках 8.12 (программный код алгоритма) и 8.13 (результаты работы алгоритма).

Персептрон – еще один простой алгоритм классификации, подходящий для крупномасштабного обучения.

Персептрон является простейшим видом искусственных нейронных сетей. В основе персептрона лежит математическая модель восприятия информации мозгом. В самом общем своем виде он представляет систему из элементов трех разных типов: сенсоров, ассоциативных элементов и реагирующих элементов: поступающие от датчиков сигналы передаются ассоциативным элементам, а затем реагирующим элементам (это позволяет создать набор «ассоциаций» между входными стимулами и необходимой реакцией на выходе)[2].

Логическая схема персептрона с тремя выходами приведена на рис. 8.12.

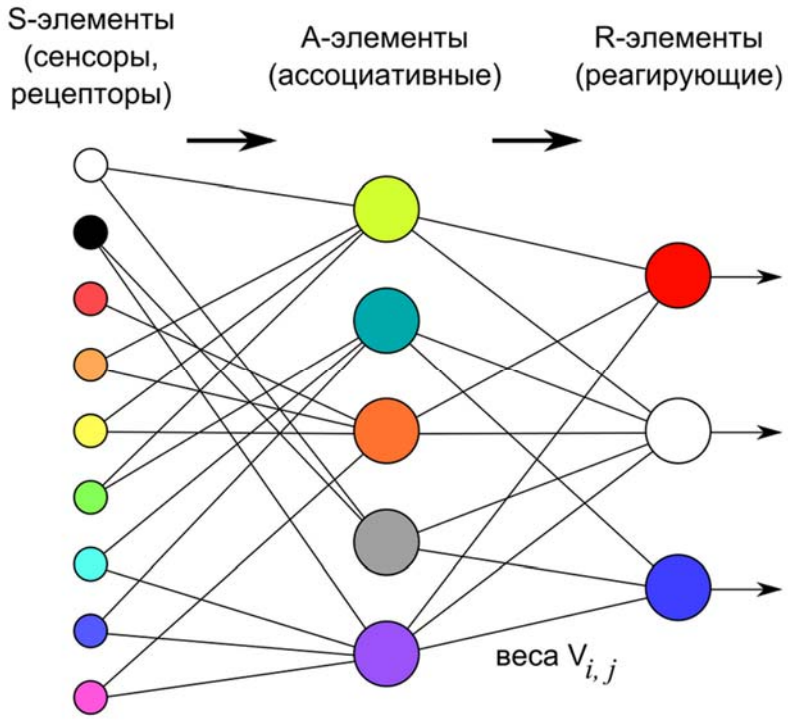


Рис. 8.12. Логическая схема персептрона с тремя выходами

Одним из основных методов обучения персептрона является метод коррекции ошибки, который представляет собой такой метод обучения, при котором вес связи не изменяется до тех пор, пока текущая реакция персептрона остается правильной (при появлении неправильной реакции вес (величина подкрепления) изменяется на единицу, а

знак подкрепления (+/-) определяется противоположным от знака ошибки)[3].

Существует следующие модификации метода коррекции ошибок, которые отличаются между собой в зависимости способом выбора величины и знака подкрепления:

а) метод коррекции ошибок без квантования;

б) метод коррекции ошибок с квантованием;

в) метод коррекции ошибок со случайным знаком подкрепления;

г) метод коррекции ошибок со случайными возмущениями.

Для линейной классификации методом, использующим перцептроны, применяется класс linear_model.Perceptron. Данный алгоритм классификации использует ту же базовую реализацию, что и класс SGDClassifier.

Фактически, данный класс без параметров класс (Perceptron()) эквивалентен классу SGDClassifier со следующими параметрами: loss=’perceptron’, eta0=1, learning_rate=’constant’, penalty=None.

Описание класса модели:

class sklearn.linear_model.Perceptron(*, penalty, alpha, fit_intercept, max_iter, tol, shuffle, verbose, eta0, n_jobs, random_state, early_stopping, validation_fraction, n_iter_no_change, class_weight, warm_start)

Основные параметры класса модели:

penalty (тип данных – str) – указывает алгоритму метод расчета штрафа за сложность модели, который необходимо использовать (возможные варианты: {‘l1’, ‘l2’, ‘elasticnet’};

alpha (тип данных – float; значение по умолчанию – 0.0001) – указывает алгоритму значение константы к величине штрафа за сложность модели;

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость центрирования модели, т.е. включения в модель независимого параметра w0;

max_iter (тип данных – int; значение по умолчанию – 1000) – указывает алгоритму максимальное количество проходов (итераций) по обучающим данным;

random_state (тип данных – int; значение по умолчанию – None; ограничение – параметр shuffle должен быть равен True) указывает

алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения рассчитанных коэффициентов (весов) $w_{target,j}$ для задачи линейной классификации (при наличии одной целевой функции одномерный массив размерностью [1, n_features];

intercept_ (тип данных – float) – предназначен для хранения значения независимого параметра w_0 или $w_{target,0}$.;

n_iter_ (тип данных – int) – фактическое количество итераций до достижения критерия остановки;

t_ (тип данных – int) – количество обновлений весов, выполненных во время тренировки до достижения критерия остановки.

Основные методы класса модели:

fit(X, y[, coef_init, intercept_init, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples]) – зависимые переменные модели; coef_init (тип данных – массив размерностью [n_features]) – начальные значения коэффициентов w_j для задачи линейной регрессии для быстрого старта оптимизации; intercept_init (тип данных –массив размерностью [n_features]) – начальные значения коэффициента w_0 для задачи линейной регрессии для быстрого старта оптимизации; sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

partial_fit(X, y[, sample_weight]) – выполнение одного прохода методом стохастического градиентного спуска на заданном образцах обучающем наборе;

densify() – преобразование разряженной матрицы (массива) коэффициентов (coef_) в формат плотной матрицы (разрежённая матрица – это матрица с преимущественно нулевыми элементами, в противном случае, если большая часть элементов матрицы ненулевые, матрица считается плотной);

sparsify() – преобразование плотной матрицы (массива) коэффициентов (coef_) в формат разряженной матрицы (массива);

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

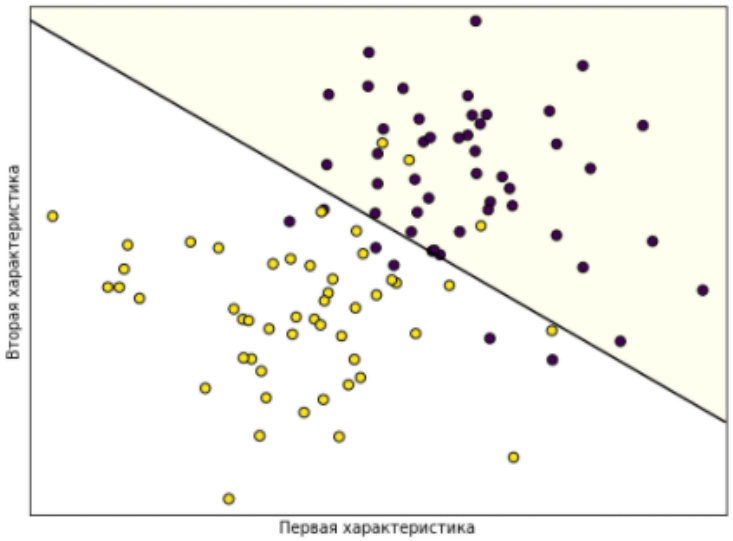
Пример использующего персептроны метода для линейной классификации приведен на рисунках 8.13 (программный код алгоритма) и 8.14 (результат работы алгоритма).

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import PassiveAggressiveClassifier
classifier =  PassiveAggressiveClassifier(C=0.2,
                                          max_iter=3000,
                                          fit_intercept=True,
                                          random_state=1,
                                          average=False,
                                          tol=None)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,4))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=1.9,
            square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_train, y_train)
```

Рис. 8.13. Программный код алгоритма линейной классификации с использованием метода, использующего персептроны (начало)

```
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.13. Программный код алгоритма линейной классификации с использованием метода, использующего персептроны (окончание)




```
Отчет по итогу проведенной классификации:
              precision    recall  f1-score   support

           0       0.92      0.79      0.85        14
           1       0.77      0.91      0.83        11

    accuracy                           0.84        25
   macro avg       0.84      0.85      0.84        25
weighted avg       0.85      0.84      0.84        25
```

Рис. 8.14. Результаты выполнения алгоритма линейной классификации с использованием метода, использующего персептроны

Пассивно-агрессивные алгоритмы – это семейство алгоритмов, используемых для крупномасштабного обучения. Их особенность заключается в том, что обучение осуществляется не на основе одновременно поступающего на вход пакета данных, а на основе входных данных, которые поступают в последовательном порядке (модель машинного обучения обновляется шаг за шагом по мере поступления новых данных).

Это очень полезно в ситуациях, когда имеется огромный объем данных, а обучение всего набора данных с вычислительной точки зрения невозможно из-за их огромного размера.

Пассивно-агрессивные алгоритмы чем-то похожи на модель персептрона в том смысле, что они не требуют скорости обучения. Однако они включают параметр регуляризации.

Название своё данное семейство алгоритмов получила из-за двоякой реакции на результаты проверки соответствия поступающих в модель новых данных и сделанного ранее прогноза: пассивная реакция в ситуации верного прогноза (модель в данном случае не изменяется) и агрессивная (модель корректируется с целью исправления выявленного не соответствия).

Для линейной классификации использующим пассивно-агрессивные алгоритмы методом, применяется класс linear_model.PassiveAggressiveClassifier.

Описание класса модели:

class sklearn.linear_model.PassiveAggressiveClassifier (*, C, fit_intercept, max_iter, tol, early_stopping, validation_fraction, n_iter_no_change, shuffle, verbose, loss, n_jobs, random_state, warm_start, class_weight, average)

Основные параметры класса модели:

C (тип данных – float; значение по умолчанию – 1.0, ограничение – положительное значение) – параметр регуляризации (меньшее значение обеспечивает более сильную регуляризацию);

fit_intercept (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость центрирования модели, т.е. включения в модель независимого параметра w0;

max_iter (тип данных – int; значение по умолчанию – 1000) – указывает алгоритму максимальное количество проходов (итераций) по обучающим данным;

tol (тип данных – int; значение по умолчанию – 0.001)) – указывает алгоритму точность решения;

early_stopping (тип данных – bool; значение по умолчанию – False) – указывает алгоритму необходимость преждевременной остановки в случае отсутствия улучшений по крайней мере на величину tol для последних n_iter_no_change последовательных эпох;

validation_fraction (тип данных – float; значение по умолчанию – 0.1, ограничение – значение от 0 до 1, используется при early_stopping=True) – указывает алгоритму доля обучающих данных, которые нужно отложить в качестве проверочного набора для преждевременной остановки;

n_iter_no_change (тип данных – int; значение по умолчанию – 5) – указывает алгоритму количество итераций без улучшений, требующихся для преждевременной остановки обучения;

shuffle (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость перетасовки обучающих данных после каждого прохода по обучающим данным;

random_state (тип данных – int; значение по умолчанию – None; ограничение – параметр shuffle должен быть равен True) указывает алгоритму параметр генератора случайных чисел для перемешивания данных;

warm_start (тип данных – bool; необязательный параметр) – указывает алгоритму необходимость использования решения предыдущего вызова (при warm_start=True)

class_weight (тип данных – str или ‘balanced’; значение по умолчанию – None) указывает алгоритму соответствие определенных весов определенным классам выходных переменных (запись осуществляется по форме: {class_label: weight}), если параметр не указан, все веса предполагаются равными 1, а если указан параметр ’balanced’, то осуществляется автоматическая регулировка весов, обратно пропорциональных частотам классов во входных данных следующим образом: n_samples / (n_classes * np.bincount(y).

Основные атрибуты класса модели:

coef_ (тип данных – двухмерный массив размерностью [n_targets, n_features]) – предназначен для хранения рассчитанных коэффициентов (весов) $w_{target,j}$ для задачи линейной классификации (при наличии одной целевой функции одномерный массив размерностью [1, n_features];

intercept_ (тип данных – одномерный массив float) – предназначен для хранения значения независимого параметра w_0 или $w_{target,0}$.;

n_iter_ (тип данных – int) – фактическое количество итераций до достижения критерия остановки;

t_ (тип данных – int) – количество обновлений весов, выполненных во время тренировки до достижения критерия остановки.

Основные методы класса модели:

fit(X, y[, coef_init, intercept_init, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples]) – зависимые переменные модели; coef_init (тип данных – массив размерностью [n_features]) – начальные значения коэффициентов w_j для задачи линейной регрессии для быстрого старта оптимизации; intercept_init (тип данных –массив размерностью [n_features]) – начальные значения коэффициента w_0 для задачи линейной регрессии для быстрого старта оптимизации; sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

partial_fit(X, y[, sample_weight]) – выполнение одного прохода методом стохастического градиентного спуска на заданном образцах обучающем наборе;

densify() – преобразование разряженной матрицы (массива) коэффициентов (coef_) в формат плотной матрицы (разрежённая матрица – это матрица с преимущественно нулевыми элементами, в противном случае, если большая часть элементов матрицы ненулевые, матрица считается плотной);

sparsify() – преобразование плотной матрицы (массива) коэффициентов (coef_) в формат разряженной матрицы (массива);

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример классификации с использованием основанном на пассивно-агрессивных алгоритмах методе приведен на рисунках 13.15 (программный код алгоритма) и 13.16 (результат работы алгоритма).

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.linear_model import PassiveAggressiveClassifier
classifier =  PassiveAggressiveClassifier(C=0.1,
                                          max_iter=3000,
                                          fit_intercept=True,
                                          random_state=1,
                                          average=False,
                                          tol=None)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

 # визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')
# вывод матрицы ошибок модели
# краткая форма - print (confusion_matrix(y_test, y_pred))
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,4))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=1.9, square = True,
            cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_train, y_train)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.15. Программный код алгоритма линейной классификации с использованием основанном на пассивно-агрессивных алгоритмах методе

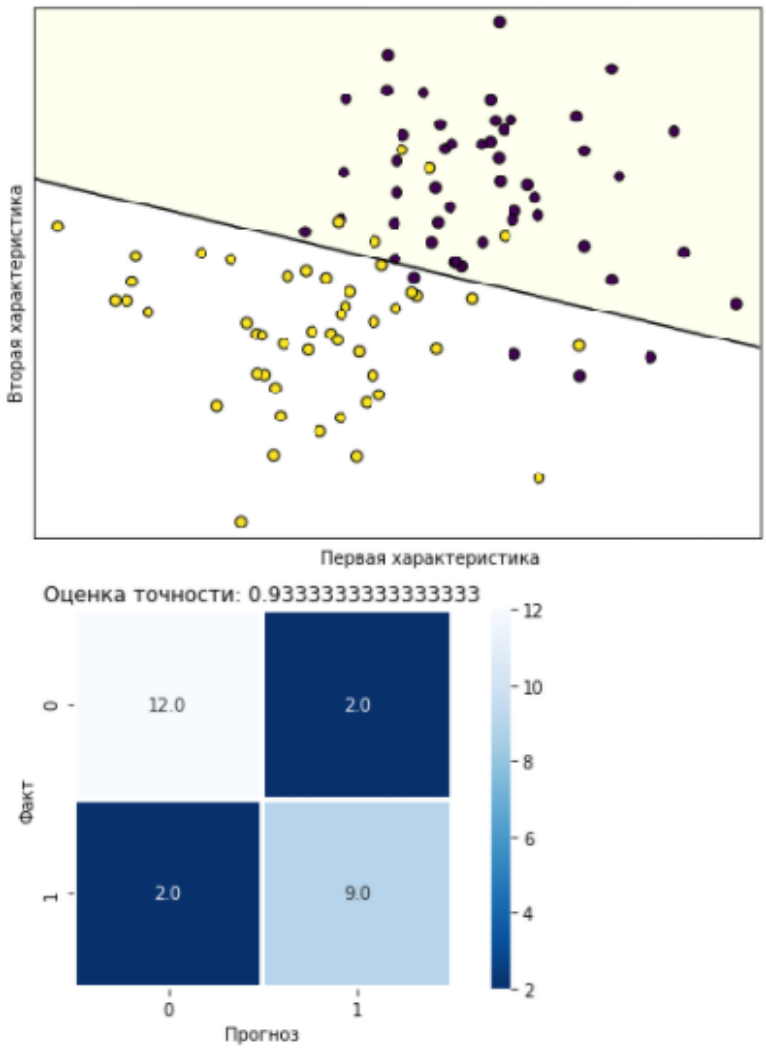


Рис. 8.16. Результаты выполнения алгоритма линейной классификации

8.3. Модели классификации с нелинейными разделяющими поверхностями

При наличии нелинейной связи между входными и выходными данными качество линейных классификаторов часто может оказываться неудовлетворительным, т.е. построение разделяющей прямой (или линейной поверхности) не эффективно. В этом случае рекомендуется использовать модели классификации с нелинейными разделяющими поверхностями, которые, как правило, для учета нелинейности расширяют пространство переменных за счет различных функциональных преобразований исходных данных (полиномы, экспоненты и проч.).

В Scikit-learn для работы с моделями классификации с нелинейными разделяющими поверхностями созданы несколько классов, которые реализуют следующие методы классификации:

а) метод k-ближайших соседей (sklearn.neighbors.KNeighborsClassifie);

б) группа методов опорных векторов (sklearn.svm);

в) метод дерева решений (sklearn.tree.DecisionTreeClassifier)

г) метод случайного леса (sklearn.ensemble. RandomForestClassifier);

д) наивный байесовский метод (sklearn.naive_bayes.GaussianNB).

Рассмотрим несколько методов поподробнее

Метод k-ближайших соседей – метрический алгоритм для автоматической классификации объектов, при котором объект присваивается тому классу, который является наиболее распространённым среди k соседей данного элемента, классы которых уже известны [4].

К особенностям данного метода относят: а) не требуется осуществления обучения перед выполнением прогнозов в реальном времени (в результате метод является легкореализуемым, позволяет добавлять новые данные, имеет высокую скорость выполнения); б) может быть применим к выборкам с большим количеством атрибутов (многомерным); в) перед применением необходимо определить функцию расстояния (метрику).

К недостаткам метода относят: а) высокую стоимость прогнозирования для больших наборов данных (при большом количестве измерений алгоритму становится сложно вычислять расстояние в каждом измерении); б) высокую стоимость прогнозирования для моделей с категориальными признаками.

Для классификации методом k-ближайших соседей используется класс sklearn.neighbors. KNeighborsClassifie. Описание класса модели:

class sklearn.neighbors.KNeighborsClassifier (n_neighbors, *, weights, algorithm, leaf_size, p, metric, metric_params, n_jobs, **kwargs)

Основные параметры класса модели:

n_neighbors (тип данных – int; значение по умолчанию – 5) – указывает алгоритму количество ближайших соседей, необходимых для проведения классификации;

weights (тип данных – str, значение по умолчанию – ‘uniform’) указывает алгоритму функцию для расчета весов либо предопределенную (возможные варианты: {‘uniform’, ‘distance’}) либо определяемую пользователем функция, которая принимает массив расстояний и возвращает массив той же формы, содержащий веса;

algorithm (тип данных – str, значение по умолчанию – ‘auto) указывает алгоритму способ вычисления ближайших соседей: (возможные варианты: {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’});

p (тип данных – int; значение по умолчанию – 2) указывает алгоритму степенный параметр для метрики Минковского (при p = 1 эквивалентно использованию manhattan_distance (l1), при p = 2 – euclidean_distance (l2), при всех остальных значениях p используется minkowski_distance (l_p));

metric (тип данных – str, значение по умолчанию – ‘minkowski’) указывает алгоритму метрику расстояния, которую необходимо использовать.

Основные атрибуты класса модели:

classes_ (тип данных – массив размерностью [n_classes,]) предназначен для хранения меток классов, известных классификатору;

effective_metric_ (тип данных – str) предназначен для хранения информации об используемой метрики расстояния;

Основные методы класса модели:

fit(X, y[, sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features] или [n_samples, n_samples]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples, n_outputs] или [n_samples]) – зависимые переменные модели;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

kneighbors(X, n_neighbors, return_distance) – возвращает индексы и расстояния до соседей указанной точки или группы точек, где X (тип данных – двухмерный массив размерностью [n_queries, n_features] или [n_queries, n_indexed]) – точка запроса (если данный параметр не указан, возвращаются соседи каждой точки), n_neighbors (тип данных – int) – число соседних точек (по умолчанию значение, переданное конструктору класса), return_distance (тип данных – bool; по умолчанию – True);

kneighbors_graph(X, n_neighbors, mode) – вычисляет взвешенный граф до соседей указанной точки или группы точек, где X (тип данных – двухмерный массив размерностью [n_queries, n_features] или [n_queries, n_indexed]) – точка запроса (если данный параметр не указан, возвращаются соседи каждой точки), n_neighbors (тип данных – int) – число соседних точек (по умолчанию значение, переданное конструктору класса), mode (тип данных – str, значение по умолчанию – ‘connectivity’) – способ представления графа в виде матрицы («connectivity» – в виде матрицы смежности, «distance» – в виде матрицы, содержащей евклидово расстояние между точками.

score (X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования k-ближайших соседей для классификации приведен на рисунках 8.17 (программный код алгоритма) и 8.18 (результаты работы алгоритма).

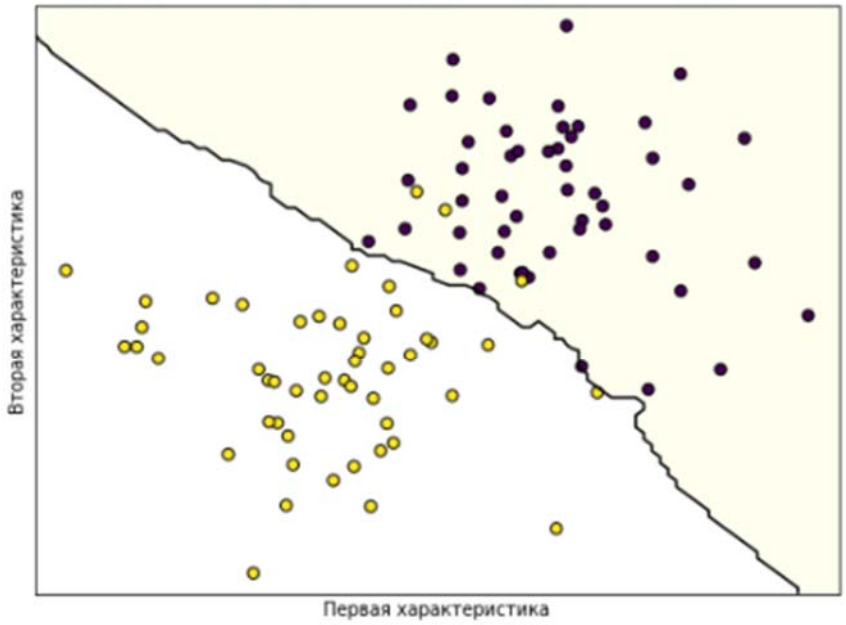
Методы опорных векторов – набор схожих алгоритмов для автоматической классификации объектов, при котором осуществляется перевод исходных векторов в пространство более высокой размерности и осуществляется поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве [5].

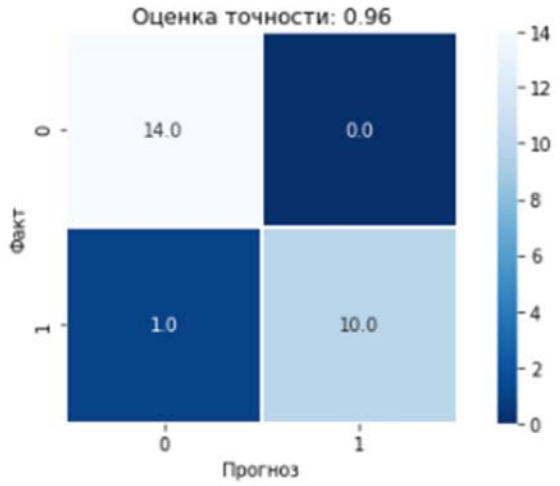
```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=4)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=9)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True,
            fmt=".1f",
            linewidths=0.9,
            square = True,
            cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_test, y_test)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.17. Программный код алгоритма классификации
с использованием метода k-ближайших соседей

Основная идея классификатора на опорных векторах заключается в том, чтобы строить разделяющую поверхность с использованием только небольшого подмножества точек, лежащих в зоне, критической для разделения, тогда как остальные верно классифицируемые наблюдения обучающей выборки вне этой зоны игнори-

руются (точнее, являются «резервуаром» для оптимизационного алгоритма) [6].





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.93	1.00	0.97	14
1	1.00	0.91	0.95	11
accuracy			0.96	25
macro avg	0.97	0.95	0.96	25
weighted avg	0.96	0.96	0.96	25

Рис. 8.18. Результаты выполнения алгоритма классификации с использованием метода k-ближайших соседей

К недостаткам методов опорных векторов относят: а) сложность интерпретации параметров модели; б) отсутствие возможности калибровки вероятности попадания объекта в определенный класс.

В Scikit-learn для возможности осуществления классификации включены несколько классов, которые реализуют следующие методы опорных векторов:

а) линейный классификатор (sklearn.svm.LinearSVC);

б) классификатор с поддержкой управления количеством опорных векторов (sklearn.svm.NuSVC);

в) классификатор с поддержкой регуляризации (sklearn.svm.SVC);

Рассмотрим поподробнее классификатор с поддержкой регуляризации. Для классификации данным методом используется класс sklearn.svm.SVC. Описание класса модели:

class sklearn.svm.SVC(*, C, kernel, degree, gamma, coef0, shrinking, probability, tol, cache_size, class_weight, verbose, max_iter, decision_function_shape, break_ties, random_state)

Основные параметры класса модели:

C (тип данных – float; значение по умолчанию – 1.0, ограничение – положительное значение) – параметр регуляризации (меньшее значение обеспечивает более сильную регуляризацию);

kernel (тип данных – str; значение по умолчанию – ‘rbf’) указывает алгоритму тип ядра, который необходимо использовать (возможные варианты: {‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’});

degree (тип данных – int; значение по умолчанию – 3) – указывает алгоритму степень полиномиальной функции ядра (используется только при kernel=‘poly’);

gamma (тип данных – str или float; значение по умолчанию – ‘scale’) указывает алгоритму способ расчета коэффициента ядра или его значения (возможные варианты способ расчета: {‘scale’, ‘auto’}, используется только при kernel=‘poly’|‘poly’|‘sigmoid’);

coef0 (тип данных – float; значение по умолчанию – 0.0) указывает алгоритму способ расчета коэффициента ядра или его значения (возможные варианты способ расчета: {‘scale’, ‘auto’}, используется только при kernel=‘poly’|‘poly’|‘sigmoid’);

tol (тип данных – int; значение по умолчанию – 0.001)) – указывает алгоритму точность решения;

class_weight (тип данных – str или ‘balanced’; значение по умолчанию – None) указывает алгоритму соответствие определенных весов определенным классам выходных переменных (запись осуществляется по форме: {class_label: weight}), если параметр не указан, все веса предполагаются равными 1, а если указан параметр ’balanced’, то осуществляется автоматическая регулировка весов, обратно пропорциональных частотам классов во входных данных следующим образом: n_samples / (n_classes * np.bincount(y).

max_iter (тип данных – int; значение по умолчанию – -1) – указывает алгоритму максимальное количество проходов (итераций) по обучающим данным (-1 соответствует не ограниченному количеству);

shuffle (тип данных – bool; значение по умолчанию – True) – указывает алгоритму необходимость перетасовки обучающих данных после каждого прохода по обучающим данным;

random_state (тип данных – int; значение по умолчанию – None; ограничение – параметр shuffle должен быть равен True) указывает алгоритму параметр генератора случайных чисел для перемешивания данных.

Основные атрибуты класса модели:

support_ (тип данных – двухмерный массив размерностью [n_SV,]) – предназначен для хранения индексов опорных векторов;

support_vectors_ (тип данных – двухмерный массив размерностью [n_SV, n_features]) – предназначен для хранения информации об опорных векторах;

n_support_ (тип данных – двухмерный массив размерностью [n_class,]) – предназначен для хранения информации о количестве опорных векторов для каждого класса;

coef_ (тип данных – двухмерный массив размерностью [n_class * (n_class-1) / 2, n_features]) – предназначен для хранения рассчитанных коэффициентов (весов) $w_{target,j}$ для задачи линейной классификации (ограничение: kernel=‘linear’);

intercept_ (тип данных – двухмерный массив размерностью [n_class * (n_class-1) / 2,]) – предназначен для хранения значений констант в решающие функции;

classes_ (тип данных – двухмерный массив размерностью [n_classes,]) – предназначен для хранения меток классов.

Основные методы класса модели:

fit(X, y[,sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – двухмерный массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples,]) – зависимые переменные модели; sample_weight (тип данных – массив размерностью или [n_samples]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода опорных векторов с поддержкой регуляризации для построения классификации приведен на рисунках 8.19 (результаты работы) и 8.20 (программный код алгоритма).

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=4)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.svm import SVC
classifier = SVC(C=10,
                 kernel='poly',
                 degree=7,
                 coef0=0.02)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')
```

Рис. 8.19. Программный код алгоритма классификации с использованием метода опорных векторов с регуляризацией (начало)

```
# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,4))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=0.9,
            square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_test, y_test)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.title(all_sample_title, size = 12);
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.19. Программный код алгоритма классификации
с использованием метода опорных векторов с регуляризацией (окончание)

Метод дерева решений – алгоритм классификации объектов, при котором осуществляется построение древовидной структуры, состоящий из решающих правил вида «если ..., то ...», которые генерируются автоматически в процессе обучения на основании обобщения множества отдельных наблюдений [7]. Пример структуры дерева решений приведен на рис. 8.21[8].

Преимущества метода дерева решений:

а) простота понимания и интерпретации (интуитивно понятная классификационная модель, правила на естественном языке);

б) быстрый процесс обучения;

в) отсутствие требований к подготовке данных;

г) надёжность;

д) возможность построения непараметрических моделей;

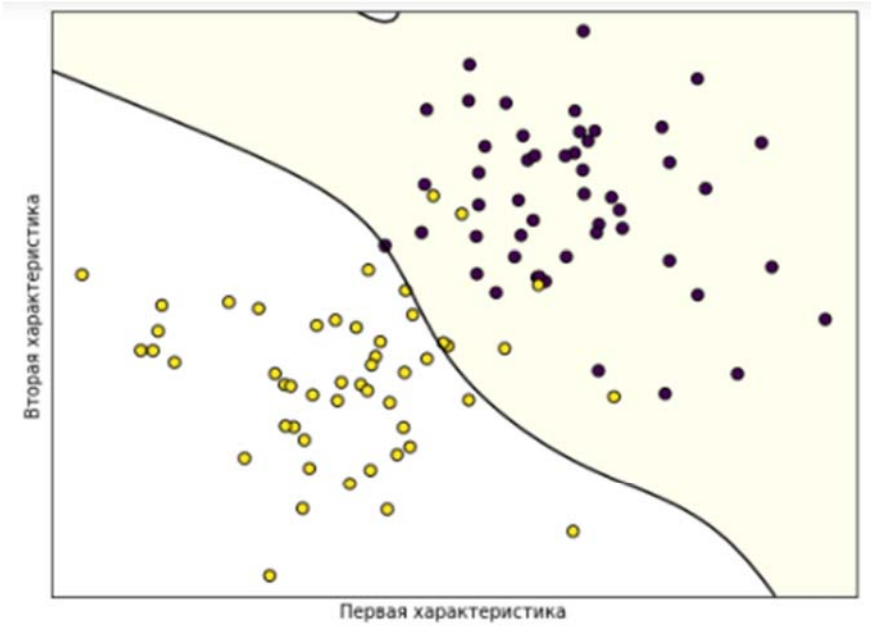
е) высокая точность предсказания.

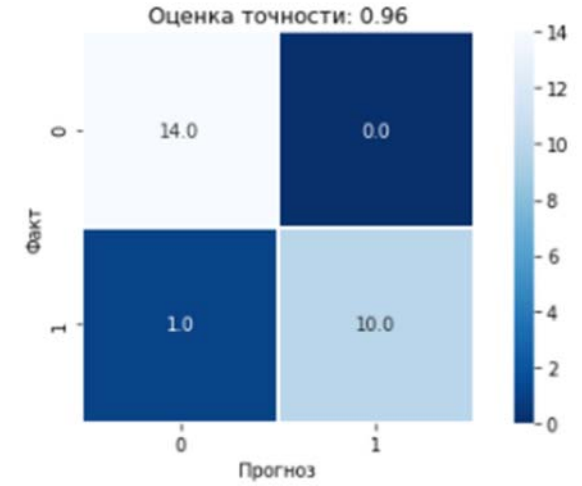
Недостатки метода дерева решений:

а) неустойчивость к изменению обучающих данных;

б) отсутствие гарантии построения оптимального дерева;

в) склонность к переобучению.





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.93	1.00	0.97	14
1	1.00	0.91	0.95	11
accuracy			0.96	25
macro avg	0.97	0.95	0.96	25
weighted avg	0.96	0.96	0.96	25

Рис. 8.20. Результаты выполнения алгоритма классификации
с использованием метода опорных векторов с поддержкой регуляризации

Для классификации методом дерева решений используется класс sklearn.tree.DecisionTreeClassifier.

Описание класса модели:

class sklearn.tree.DecisionTreeClassifier(*, criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, random_state, max_leaf_nodes, min_impurity_decrease0, min_impurity_split, class_weight, presort, ccp_alpha)

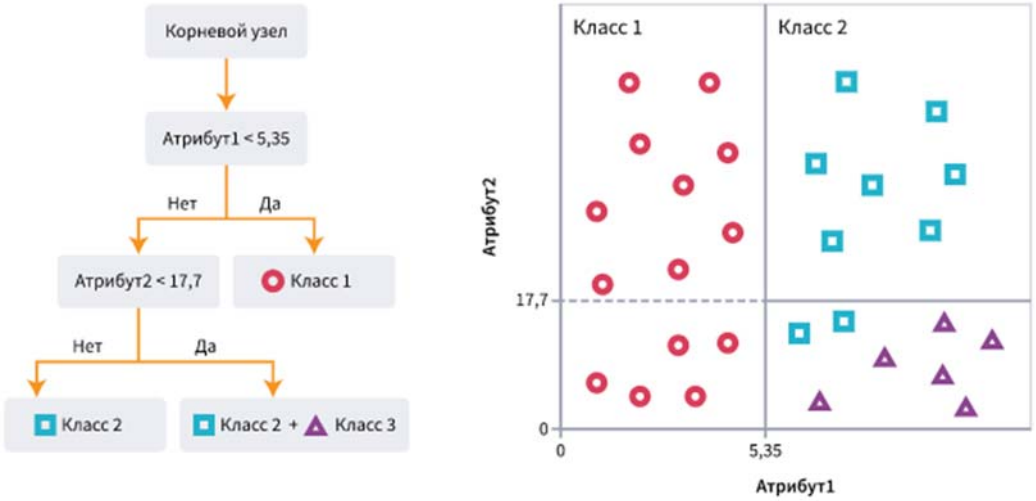


Рис. 8.21. Структура дерева решений при выборе

Основные параметры класса модели:

criterion (тип данных – str; значение по умолчанию – ‘gini’) указывает алгоритму кретерий измерения качества разделения (возможные варианты: ‘gini’, ‘entropy’});

splitter (тип данных – str; значение по умолчанию – ‘best’) указывает алгоритму стратегию выбора разделения на каждом (возможные варианты: ‘best’, ‘random’});

max_depth (тип данных – int) – указывает алгоритму максимальную глубину дерева (если данный параметр не указывается, то разделение идет до тех пор, пока не будет однозначного решения или пока не будет достигнуто значение параметра min_samples_split);

min_samples_split (тип данных – int или float; значение по умолчанию – 2) – указывает алгоритму минимальное количество данных, необходимое для разделения промежуточного узла;

min_samples_leaf (тип данных – int или float; значение по умолчанию – 1) – указывает алгоритму минимальное количество данных, необходимое для разделения конечного узла;

max_features (тип данных – str; значение по умолчанию – ‘auto’) указывает алгоритму максимальное количество выходных признаков (возможные варианты: “auto”, “sqrt”, “log2”});

random_state (тип данных – int; значение по умолчанию – None) указывает алгоритму параметр генератора случайных чисел при осуществлении разделения данных;

ccp_alpha (тип данных – float; значение по умолчанию – 1.0, ограничение – положительное значение) – параметр сложности модели, используемый для обрезки дерева.

Основные атрибуты класса модели:

classes_ (тип данных – массив размерностью [n_classes,] или список таких массивов) – список меток классов (модель с одним выходом) или список массивов меток классов (модель с несколькими выходами), известных классификатору;

feature_importances_ (тип данных – массив размерностью [n_classes,]) – значения функции разделения (индекса Джини);

n_classes_(тип данных int или список int) – количество классов (модель с одним выходом) или список, содержащий количество классов для каждого выхода (модель с несколькими выходами).

Основные методы класса модели:

apply(X, check_input=True) – возвращает индексы конечных узлов, где X (тип данных – массив размерностью [n_samples, n_features]) – независимые переменные модели;

decision_path(X, check_input=True) – возвращает путь решения в дереве, где X (тип данных – массив размерностью [n_samples, n_features]) – независимые переменные модели;

feature_importances_ – возвращает общие значения функции разделения (индекса Джини);

fit(X, y[,sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples,] или [n_samples, n_outputs]) – зависимые переменные модели (классы) в виде целых чисел или строк.; sample_weight (тип данных – массив размерностью [n_samples,]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

get_depth() – возвращает глубину дерева решений;

get_n_leaves() – возвращает количество конечных узлов (листьев) дерева решений;

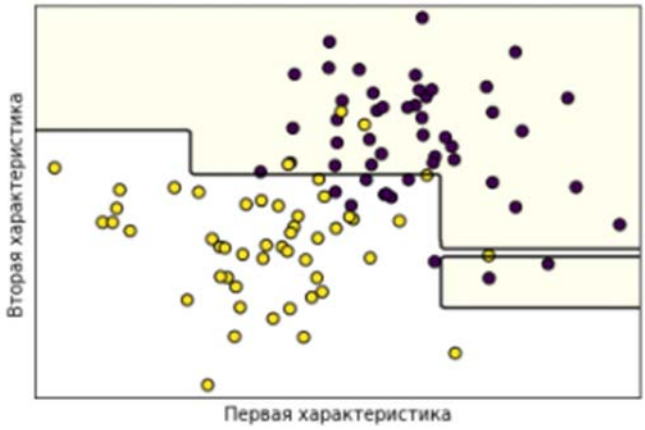
predict(X) – прогнозирование параметров модели на основании тестового набора данных;

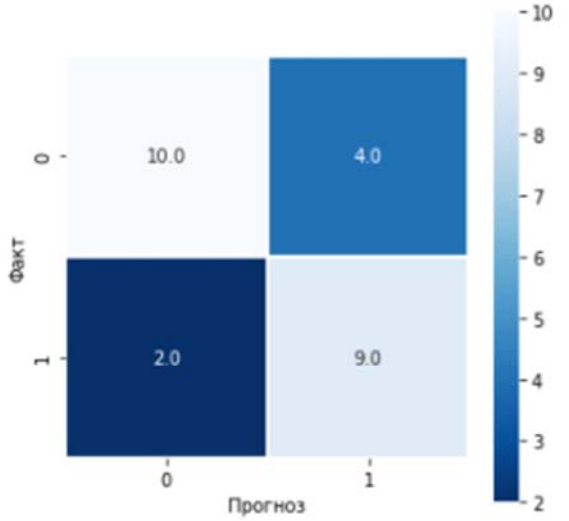
score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая возможная оценка равна 1).

Пример использования метода дерева решений для построения классификации приведен на рисунках 8.22 (результаты работы алгоритма) и 8.23 (программный код алгоритма).

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=4)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(splitter='random', max_depth=6)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=0.9,
            square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_test, y_test)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.22. Программный код алгоритма классификации
с использованием метода дерева решений





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.83	0.71	0.77	14
1	0.69	0.82	0.75	11
accuracy			0.76	25
macro avg	0.76	0.77	0.76	25
weighted avg	0.77	0.76	0.76	25

Рис. 8.23. Результаты выполнения алгоритма классификации
с использованием метода дерева решений

Метод случайного леса – алгоритм автоматической классификации, который предполагает использование большого ансамбля решающих деревьев, каждое из которых само по себе даёт результат очень невысокого качества, но за счёт их большого количества качество результата повышается до приемлемого уровня [9].

К достоинствам данного метода относят:

а) отсутствие чувствительности к масштабированию;

б) высокий уровень реализации параллельных вычислений;

в) высокая эффективность работы с большим числом признаков и классов;

г) высокая эффективность работы с непрерывными и дискретными признаками, а также с пропущенными значениями признаков.

Ещё одним достоинством является то, что данный алгоритм имеет внутреннюю оценку способности модели к обобщению (например, тест Out-of-Bag позволяет проводить оценку базовых алгоритмов на тех данных, на которых модель не обучалась).

Ключевым недостатком метода случайного леса является большой размер получающихся моделей.

Для классификации методом случайного леса используется класс sklearn.ensemble. RandomForestClassifier.

Описание класса модели:

class sklearn.ensemble.RandomForestClassifier(n_estimators, *, criterion, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, min_impurity_decrease0, min_impurity_split, bootstrap, oob_score, n_jobs, random_state, verbose, warm_start, class_weight, ccp_alpha, max_samples)

Основные параметры класса модели:

n_estimators (тип данных – int, значение по умолчанию – 100) – указывает алгоритму число деревьев в «лесу»;

criterion (тип данных – str; значение по умолчанию – ‘gini’) указывает алгоритму критерий измерения качества разделения (возможные варианты: ‘gini’, ‘entropy’});

max_depth (тип данных – int) – указывает алгоритму максимальную глубину дерева (если данный параметр не указывается, то разделение идет до тех пор пока не будет однозначного решения или пока не будет достигнуто значение параметра min_samples_split);

min_samples_split (тип данных – int или float; значение по умолчанию – 2) – указывает алгоритму минимальное количество данных, необходимое для разделения промежуточного узла;

min_samples_leaf (тип данных – int или float; значение по умолчанию – 1) – указывает алгоритму минимальное количество данных, необходимое для разделения конечного узла;

max_features (тип данных – str; значение по умолчанию – ‘auto’) – указывает алгоритму максимальное количество выходных признаков (возможные варианты: “auto”, “sqrt”, “log2”});

bootstrap (тип данных – bool; значение по умолчанию – True) – указывает алгоритму на необходимость использовать образцы начальной загрузки при построении деревьев (при значении False, весь набор данных используется для построения каждого дерева);

oob_scorebool (тип данных – bool; значение по умолчанию – False) – указывает алгоритму на необходимость использовать нестандартные образцы для оценки точности обобщения (т.е. на необходимость проведения теста Out-of-Bag);

ccp_alpha (тип данных – float; значение по умолчанию – 0.0, ограничение – положительное значение) – параметр сложности модели, используемый для обрезки дерева.

Основные атрибуты класса модели:

estimators_ – коллекция подобранных оценок;

classes_ (тип данных – массив размерностью [n_classes,] или список таких массивов) – список меток классов (модель с одним выходом) или список массивов меток классов (модель с несколькими выходами), известных классификатору;

n_classes_ (тип данных int или список int) – количество классов (модель с одним выходом) или список, содержащий количество классов для каждого выхода (модель с несколькими выходами).

oob_score_ (тип данных float) – оценка набора обучающих данных, полученная с использованием нестандартной оценки (возвращает значение только при oob_score установленным в значение True).

Основные методы класса модели:

apply(X, check_input=True) – возвращает индексы конечных узлов, где X (тип данных – массив размерностью [n_samples, n_features]) – независимые переменные модели;

decision_path(X, check_input=True) – возвращает путь решения в дереве, где X (тип данных – массив размерностью [n_samples, n_features]) – независимые переменные модели;

feature_importances_ – возвращает общие значения функции разделения (индекса Джини);

fit(X, y[,sample_weight]) – построение модели на основании обучающего набора данных, где X (тип данных – массив размерностью [n_samples, n_features]) – независимые переменные модели, y (тип данных – массив размерностью [n_samples,] или [n_samples, n_outputs]) – зависимые переменные модели (классы) в виде целых чисел или строк.; sample_weight (тип данных – массив размерностью [n_samples,]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

get_depth() – возвращает глубину дерева решений;

get_n_leaves() – возвращает количество конечных узлов (листьев) дерева решений;

predict(X) – прогнозирование параметров модели на основании тестового набора данных;

score(X, y, sample_weight=None) – оценка качества прогноза на основании тестового набора данных (возвращает коэффициент детерминации при сравнении прогноза и данных тестовой выборки, наилучшая оценка равна 1).

Пример использования метода случайного леса для построения классификации приведен на рисунках 8.24 (результаты работы алгоритма) и 8.25 (программный код алгоритма).

Наивный байесовский метод – алгоритм классификации, который предполагает использование теоремы Байеса со строгими (наивными) предположениями о статистической независимости признаков [10].

Классификация проводится в следующем порядке: для классифицируемого объекта вычисляются функции правдоподобия каждого из классов; по функциям правдоподобия каждого из классов вычисляются апостериорные вероятности классов; объект относится к тому классу, для которого апостериорная вероятность максимальна.

К достоинствам данного метода относят:

а) высокую скорость выполнения;

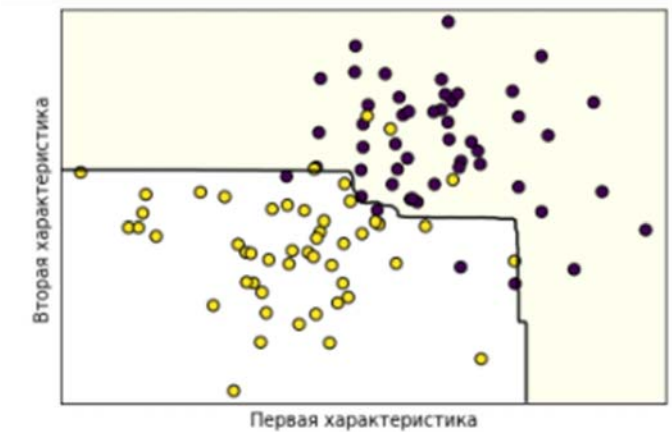
б) низкие требования к объёму обучающих данных;

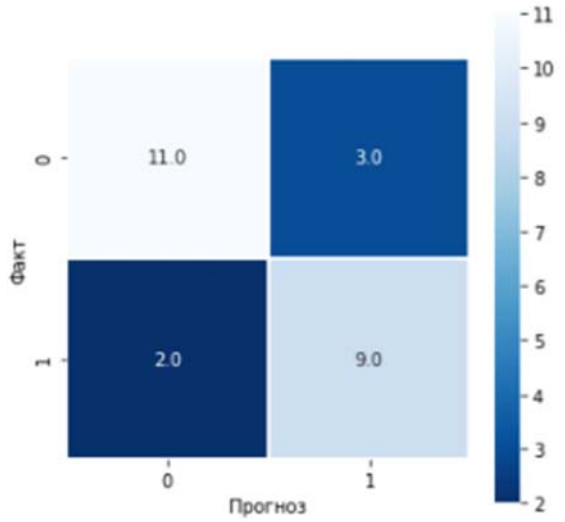
в) высокую эффективность работы с признаками-категориями.

Ключевым недостатком наивного байесовского метода является возможность появления так называемой «нулевой частоты», когда не встречающемуся в обучающем наборе данных, но присутствующем в тестовом признаку при формировании прогноза присваивается нулевая вероятность и прогноз становится неполным. Данную проблему принято решать с помощью дополнительной процедуры – сглаживания.

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=200,
                                    max_depth=2,
                                    random_state=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".1f", linewidths=0.9,
            square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_test, y_test)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Рис. 8.24. Программный код алгоритма классификации
с использованием метода случайного леса





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.85	0.79	0.81	14
1	0.75	0.82	0.78	11
accuracy			0.80	25
macro avg	0.80	0.80	0.80	25
weighted avg	0.80	0.80	0.80	25

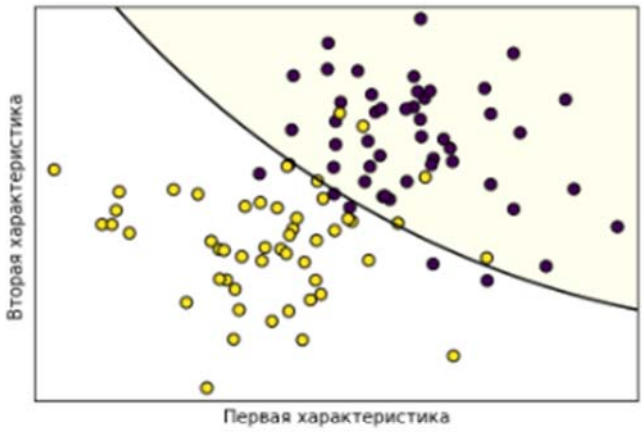
Рис. 8.25. Результаты выполнения алгоритма классификации
с использованием метода случайного леса

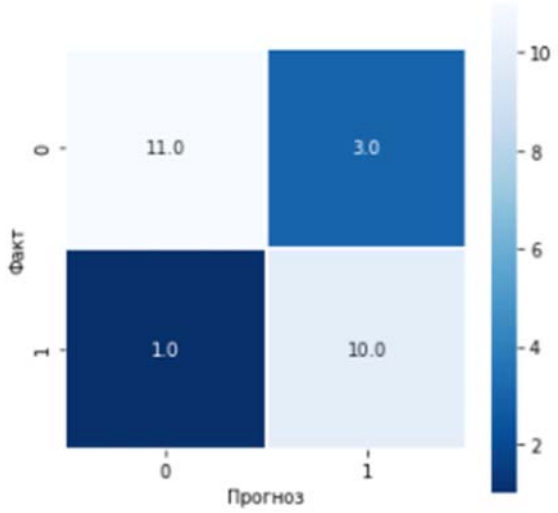
Для классификации наивный байесовский метод используется класс sklearn.naive_bayes. GaussianNB. Описание класса модели:

class sklearn.naive_bayes.GaussianNB(*, priors, var_smoothing)

```
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=1,cluster_std=5)

# разбиение набора данных на тренировочный и тестовый
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# обучение модели и прогнозирование
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB(var_smoothing =0.001)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# визуализация модели
import numpy as np
import matplotlib.pyplot as plt
plt.xlabel("Первая характеристика")
plt.ylabel("Вторая характеристика")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=40, edgecolor='k')

# вывод матрицы ошибок модели
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".1f",
            linewidths=0.9, square = True, cmap = 'Blues_r');
plt.ylabel('Факт');
plt.xlabel('Прогноз');
score=classifier.score(X_test, y_test)
all_sample_title = 'Оценка точности: {0}'.format(score)
plt.show();

# вывод отчета по итогу проведенной классификациии
print ('Отчет по итогу проведенной классификации:')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

# вывод прочих данных
print (classifier.class_count_)
print (classifier.class_prior_)
print (classifier.classes_)
print (classifier.epsilon_)
print (classifier.sigma_)
print (classifier.theta_)
```

Рис. 8.26. Программный код алгоритма классификации с использованием наивного байесовского метода





Отчет по итогу проведенной классификации:

	precision	recall	f1-score	support
0	0.92	0.79	0.85	14
1	0.77	0.91	0.83	11
accuracy			0.84	25
macro avg	0.84	0.85	0.84	25
weighted avg	0.85	0.84	0.84	25

Рис. 8.27. Результаты выполнения алгоритма классификации с использованием наивного байесовского метода

Основные параметры класса модели:

priors (тип данных – массив размерностью [n_classes,], значение по умолчанию – None) – указывает алгоритму пользовательские значения априорных вероятностей классов (если параметр не указан – априорные значения расчитываются при выполнении алгоритма);

var_smoothing (тип данных – float; значение по умолчанию – 1e-09) – указывает алгоритму долю наибольшей дисперсии всех функций, добавляемой к дисперсии для стабильности расчетов.

Основные атрибуты класса модели:

class_count_ (тип данных – массив размерностью [n_classes,] – количество обучающих выборок, наблюдаемых в каждом классе;

class_prior_ (тип данных – массив размерностью [n_classes,] – априорные вероятности для каждого класса;

classes_ (тип данных – массив размерностью [n_classes,] – метки классов, известные классификатору.

Пример использования наивного байесовского метода для классификации приведен на рисунках 8.24 (результаты работы алгоритма) и 8.25 (программный код алгоритма).

Вопросы для обсуждения

1. Проблемы практического применения классификационных моделей

2. Сравнение методов классификации

3. Метрики качества разбиения в задачах классификации и регрессии.

4. Композиции алгоритмов, применяемые в задачах классификации

5. Вычисление оптимальных параметров моделей классификации

6. Методы борьба с переобучением в различных методах классификации

7. Проблема определения оптимальных параметров классификации

8. Метрики качества классификации: доля правильных ответов, матрица ошибок, точность и полнота, AUC, индекс Джини, чувствительность к размеру классов.

9. Чувствительность и специфичность метода классификации

Практические задания

Задание 8.1.

На основании учебных данных о параметрах вина (sklearn.datasets.load_wine) необходимо классифицировать напиток со следующим параметрами: alcohol – 4.25; malic acid – 1.68; ash – 2.51; alcalinity of ash – 15.3; magnesium – 125.0; total phenols – 2.70; flavanoids – 3.02; nonflavanoid phenols – 0.25; proanthocyanins – 2.27; color intensity –5.61; hue – 1.08; od280/od315 of diluted wines –3.82; proline –1065.0.

Задание 8.2

На основании набора данных, полученного при решении задания 6, приведенного в теме 11, проведите исследование и сравнительные эксперименты для следующих методов построения моделей линейной классификации (по вариантам): 1) логистической регрессии; 2) логистической регрессии со встроенной перекрестной проверкой; 3) риджевой регрессии; 4) риджевой регрессии со встроенной перекрестной проверкой; 5) минимизации регуляризованных эмпирических потерь с помощью стохастического градиентного спуска; 6) использующим персептроны; 7) использующим пассивно-агрессивные алгоритмы классификации.

При проведении экспериментов использовать различные допустимые комбинации параметры методов. Сделайте выводы.

Задание 8.3

На основании набора данных, полученного при решении задания 6, приведенного в теме 8, проведите исследование и сравнительные эксперименты для следующих методов построения моделей нелинейной классификации (по вариантам): 1) k-ближайших соседей; 2) опорных векторов; 3) дерева решений; 4 случайного леса; 5) наивный байесовский метод.

При проведении экспериментов использовать различные допустимые комбинации параметры методов. Сравните с решением задания 2. Сделайте выводы.

Задание 8.4

На основании данных о транзакциях по кредитным картам необходимо выявить те из них, которые являются мошенническими.

Источник данных: https://www.kaggle.com/mlg-ulb/creditcardfraud

В наборе данных признаки V1, V2,… V28 – это основные компоненты, значения которых в целях безопасности были преобразованы; признак «Time» содержит секунды, прошедшие между каждой транзакцией и первой транзакцией в наборе данных; признак «Amount» – это сумма транзакции; признак «Class» – принадлежность транзакции к одному из двух классов (1 –мошенничество).

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать метод построения модели классификации (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение (на случайной выборке в размере 90%.) и тестирование (на случайной выборке в размере 10%.) модели классификации с использованием различных комбинаций её параметров;

7) оценить качество различных по параметрам вариантов модели по выбранной метрике;

8) осуществить классификацию на основании варианта модели с лучшим значением метрики.

Задание 8.5

На основании данных о клиентах банка осуществите их классификацию на надежных и ненадежных (задача кредитного скоринга).

Источник данных: https://www.openml.org/d/31

В наборе данных присутствуют 20 признаков, характеризующих клиента банка, а также признак «Class» – принадлежность клиента банка к одному из двух классов кредитоспособности («Bad» или «Good»).

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать метод построения модели классификации (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение (на случайной выборке в размере 90%.) и тестирование (на случайной выборке в размере 10%.) модели классификации с использованием различных комбинаций её параметров;

7) оценить качество различных по параметрам вариантов модели по выбранной метрике;

8) осуществить классификацию на основании варианта модели с лучшим значением метрики

Задание 8.6

На основании данных об изображении рукописных цифр в оттенках серого цвета размером 28 пикселей на 28 пикселей произвести их распознание.

Источник данных: https://www.openml.org/d/554

Необходимо:

1) загрузить набор данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) в качестве метода классификации выбрать метод опорных векторов;

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение (на случайной выборке в размере 50000 наблюдений) и тестирование (на случайной выборке в размере 10000 наблюдений.) модели классификации с использованием различных комбинаций её параметров;

7) оценить качество различных по параметрам вариантов модели по выбранной метрике;

8) осуществить классификацию на основании варианта модели с лучшим значением метрики

Задание 8.7

На основании данных об эффективности телефонных предложений о заключении договора на банковский депозит произвести разделение новых обращений на две группы: перспективные и неперспективные.

Источник данных: https://www.openml.org/d/1461

В наборе данных присутствуют 16 признаков, характеризующих потенциального клиента банка, а также признак «Class», характеризующий заключение контракта («Да» или «Нет»).

Необходимо:

1) загрузить набор данных и провести его исследование;

2) выполнить предварительную обработку данных (при необходимости);

3) выбрать метод построения модели классификации и метрику оценки качества модели (решение обосновать);

4) осуществить обучение модели классификации с использованием различных комбинаций её параметров (размер обучающвыборок выбрать самостоятельно);

5) оценить качество различных по параметрам вариантов модели по выбранной метрике.

6) осуществить классификацию на основании варианта модели с лучшим значением метрики

Библиографический список

1. Линейный классификатор // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Линейный _классификатор

2. Персептрон // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ Перцептрон

3. Метод коррекции ошибки // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ метод_коррекции_ошибки

4. Метод k-ближайших соседей // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Метод_k-ближайших_соседей

5. Метод опорных векторов // Википедия: [Электронный ресурс]. – Режим доступа: ttps://ru.wikipedia.org/wiki/Метод_ опорных_векторов

6. Классификация, регрессия и другие алгоритмы Data Mining с использованием R / Шитиков В. К., Мастицкий С. Э. - https://ranalytics.github.io/data-mining/062-SVM.html

7. Дерево решений // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Дерево_решений

8.Деревья решений: общие принципы // Loginom [Электронный ресурс]. – Режим доступа: https://wiki.loginom.ru/articles/decision-trees.html

9. Random forest // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Random_forest

10. Наивный байесовский классификатор // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ Наивный_байесовский_классификатор

11. Замятин А. В. Введение в интеллектуальный анализ данных: учебное пособие / А. В. Замятин; Нац. исслед. Том. гос. ун-т. - Томск: Издательский Дом Томского государственного университета, 2016. URL: http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000529594

12. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

13. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

Глава 9. SCIKIT-LEARN: МОДЕЛИ КЛАСТЕРИЗАЦИИ

В данной главе рассматриваются следующие вопросы:

1. Кластерный анализ как задача машинного обучения

2. Модели итеративной кластеризации

3. Модели иерархической кластеризации

4. Модели плотностной кластеризации

9.1. Кластерный анализ как задача машинного обучения

Кластерный анализ – многомерная статистическая процедура, выполняющая сбор данных, содержащих информацию о выборке объектов, и затем упорядочивающая объекты в сравнительно однородные группы [1]. Задача кластеризации относится к классу задач машинного обучения без учителя.

Целями кластеризации в зависимости от решаемой задачи являются:

1) расширение знаний о предметной области;

2) сжатие данных (замена множества данных, входящим в кластер, одним представителем);

3) обнаружение новизны (данных, которые невозможно отнести ни к одному из кластеров).

Кластерный анализ используется, например, для сегментации клиентов на группы со схожими историями покупок, которые затем компании могут использовать для создания целевых рекламных кампаний.

Кластерный анализ также используется в исследованиях рынка при работе с многомерными данными из опросов и тестовых панелей. Исследователи рынка используют кластерный анализ для того, чтобы сегментировать рынок и лучше понять отношения между различными группами потребителей (потенциальными клиентами), а также для позиционирования продукции, разработки новых продуктов и выбора тестовых рынков.

При изучении социальных сетей кластеризация с целью продвижения продукции может использоваться для распознавания сообществ внутри больших групп людей.

Существует около 100 разных алгоритмов кластеризации, однако, наиболее часто используемыми являются алгоритмы, которые можно отнести к одной из следующих категорий: итеративная кластеризация, иерархическая кластеризация, плотностная кластеризация и прочие.

Итеративная кластеризация предполагает объединение объектов в кластеры на основе их «близкого» (по некоторой метрике, например на основании евклидова расстояния, манхэттенского расстояния, расстояния Чебышева и т.д.) расположения относительно образовавшегося центра.

Итеративная кластеризация разделяет объекты данных на неперекрывающиеся группы. Другими словами, ни один объект не может быть членом более чем одного кластера, и каждый кластер должен иметь хотя бы один объект.

Эти методы требуют явного указания количества кластеров, к которым следует отнести объекты, и предполагают итеративный процесс назначения подмножеств точек данных в каждом из них. Примерами методов разделенной кластеризации являются: метод k-средних (k-means), метод k-медиан (k-medians) и метод k-медоидов (k-medoids), а также их вариации.

Методы итеративной кластеризации характеризуются неопределённостью и непредсказуемостью результатов. К достоинствам итеративной кластеризации относят: получение высоких результатов на кластерах сферической формы; хорошая масштабируемость. Разделенная кластеризация неэффективна для кластеров сложной формы и разных размеров, а также разной плотности.

Иерархическая кластеризация предполагает упорядочивание данных, направленных на создание древовидной структуры вложенных кластеров. Иерархическая кластеризация является детерминированным процессом.

В зависимости от способа построения иерархии выделяют два класса методов иерархической кластеризации [2]:

1) агломеративные методы (реализуют восходящий подход): новые кластеры создаются путем объединения объектов и более мелких кластеров;

2) дивизионные методы (реализуют нисходящий подход): новые кластеры создаются путем деления более крупных кластеров на более мелкие и в конечном итоге на объекты.

Примерами методов иерархической кластеризации являются: агломеративный метод и метод сбалансированного итеративного сокращения и кластеризации с помощью иерархий (BIRCH).

К сильным сторонам методов иерархической кластеризации относят: раскрытие более тонких деталей взаимосвязей между объектами данных, а также низкую сложность интерпретации результатов, представленных в форме дендрограммы.

К недостаткам иерархических методов кластеризации можно отнести следующее: высокие требования к вычислительным ресурсам, а также чувствительность к шуму и выбросам.

Плотностная кластеризация предполагает оценку плотности расположения объектов в пространстве и формирование кластеров в областях пространства с максимальной плотностью с разделением их между собой в областях с минимальной плотностью.

В отличие от других видов кластеризации методы плотностной кластеризации не требует указания количества кластеров (вместо этого задаётся пороговый параметр близости объектов для отнесения их к одному кластеру).

Примерами методов плотностной кластеризации являются: метод сдвига среднего значения, метод пространственной кластеризации для приложений с шумам (DBSCAN), метод упорядочения точек для обнаружения кластерной структуры (OPTICS) и метод распространения близости.

К сильным сторонам плотностной кластеризации относят: получение высоких результатов на кластерах несферической формы; хорошая устойчивость к выбросам. К недостаткам относят: наличие проблем с идентификацией скоплений разной плотности; получение низких результатов в многомерном пространстве.

9.2. Модели итеративной кластеризации

В Scikit-learn для работы с моделями итеративной кластеризации включен класс sklearn.cluster, который предоставляет возможность работы со следующими методами её осуществления:

а) методом k-средних (cluster.KMeans);

б) методом k-средних с применением мини-пакетов (cluster.MiniBatchKMeans).

Метод k-средних относится к группе методов разделённой кластеризации и предполагает минимизацию суммарного квадратичного отклонения точек кластеров от центров этих кластеров (главных точек).

Основная идея алгоритма k-средних заключается в том, что он осуществляется итерационно: на каждом расчётном шаге ранее рассчитанные значения центра масс каждого кластера вычисляются заново, а затем объекты заново разбиваются на кластеры в зависимости от того какой из новых центров оказался ближе (на первом шаге разбиение осуществляется произвольно). Алгоритм завершается в тот момент, когда на какой-то итерации не происходит изменения внутрикластерного расстояния.

Описание класса, реализующего в scikit-learn метод k-средних:

class sklearn.cluster.KMeans(n_clusters, *, init, n_init, max_iter, tol, precompute_distances, verbose, random_state, copy_x, n_jobs, algorithm)

Основные параметры класса модели:

n_clusters (тип данных – int; значение по умолчанию – 8) – указывает алгоритму количество кластеров, которые нужно сформировать (количество центроидов, которые необходимо создать);

init (тип данных – str; значение по умолчанию – ‘k-means++’) – указывает алгоритму метод инициализации (возможные варианты: {‘random’, ‘ndarray’, ‘callable’}, при ‘k-means++’: начальные центры кластеров для ускорения сходимости выбираются с k-средним; ‘random’: начальные центры кластеров выбираются случайным образом; ‘ndarray’: начальные центры кластеров передаются в массиве ndarray размерностью [n_clusters, n_features]; ‘callable’y’: начальные центры кластеров задаются с помощью функции пользователя);

n_init (тип данных – int; значение по умолчанию – 10) – указывает алгоритму количество подходов к определению начальных центров кластеров;

max_iter (тип данных – int; значение по умолчанию – None) – указывает алгоритму максимальное количество итераций, None соответствует значению 300;

tol (тип данных – int; значение по умолчанию – 0.0001) – указывает алгоритму точность решения.

random_state (тип данных – int; значение по умолчанию – None) указывает алгоритму параметр генератора случайных чисел для определения начальных центров кластеров;

Основные атрибуты класса модели:

cluster_centers_ (тип данных – массив размерностью [n_clusters, n_features]) – предназначен для хранения рассчитанных координат кластерных центров;

labels_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения меток каждого объекта;

inertia_ (тип данных – float) – предназначен для хранения суммы квадратов расстояний от образцов до ближайшего центра кластера;

n_iter_ (тип данных – int) – предназначен для хранения информации о количестве выполненных итераций.

Основные методы класса модели:

fit(X, y[,sample_weight]) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – неиспользуется; sample_weight (тип данных – массив размерностью [n_samples,]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – вычисление кластерных центров и прогнозирование параметров модели для выборки данных X.

Пример использования метода k-средних для кластеризации приведен на рисунках 9.1 (программный код алгоритма) и 9.2 (результаты работы алгоритма).

Метод k-средних с применением мини-пакетов (Mini-Batch K-Means) является вариацией метод k-средних и предполагает использование для сокращения времени вычислений подмножеств входных данных, которые выбираются случайным образом на каждой итерации обучения.

Для кластеризации методом k-средних с применением мини-пакетов в scikit-learn используется класс sklearn.cluster. MiniBatchKMeans.

```python
# Расчет оценок качества
def get_metrics(y1, y2):
    from sklearn import metrics
    ARI=metrics.adjusted_rand_score(y1, y2)
    AMI=metrics.adjusted_mutual_info_score(y1, y2)
    Homogenity=metrics.homogeneity_score(y1, y2)
    Completeness=metrics.completeness_score(y1, y2)
    V_measure=metrics.v_measure_score(y1, y2) #V-measure
    Silhouette=metrics.v_measure_score(y1, y2)
    return "ARI",ARI,"AMI",AMI,"Homogenity",Homogenity,"Completeness",
    Completeness,"V-measure",V_measure,"Silhouette",Silhouette

# Визуализация модели
def model_visualization(y_pred, y, labels, plt, m, n):
    plt.subplot(320+n)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, marker='o', s=120, edgecolor='k')
    plt.title("Пример "+str(n)+"\n "+m[0]+"="+str(m[1]))

from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)

# Пример 2
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
transformation = [[0.61, -0.64], [-0.4, 0.8]]
X = np.dot(X, transformation)
# Обучение модели и прогнозирование
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации и визуализация модели
m=get_metrics (y, model.labels_)
model_visualization (y_pred, y, model.labels_, plt, m, 2)

```

Рис. 9.1. Программный код алгоритма классификации с использованием метода k-средних (начало)

```
# Пример 3
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200,cluster_std=[1.0, 2.5, 0.5],random_state=150)
# Обучение модели и прогнозирование
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации и визуализация модели
m=get_metrics (y, model.labels_)
model_visualization (y_pred, y, model.labels_, plt, m, 3)

# Пример 4
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=1000, random_state=150)
X = np.vstack((X[y == 0][:150], X[y == 1][:40], X[y == 2][:10]))
y = np.concatenate([np.tile(0, 150),np.tile(1, 40),np.tile(2, 10)])
# Обучение модели и прогнозирование
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации и визуализация модели
m=get_metrics (y, model.labels_)
model_visualization (y_pred, y, model.labels_, plt, m, 4)

# Пример 5
# генерация набора данных
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=200, factor=.5, noise=.05)
# Обучение модели и прогнозирование
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации и визуализация модели
m=get_metrics (y, model.labels_)
model_visualization (y_pred, y, model.labels_, plt, m, 5)

# Пример 6
# генерация набора данных
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=200, noise=.05)
# Обучение модели и прогнозирование
model = KMeans(n_clusters=3, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации и визуализация модели
m=get_metrics (y, model.labels_)
model_visualization (y_pred, y, model.labels_, plt, m, 6)

plt.show()
```

Рис. 9.1. Программный код алгоритма классификации с использованием метода k-средних (окончание)

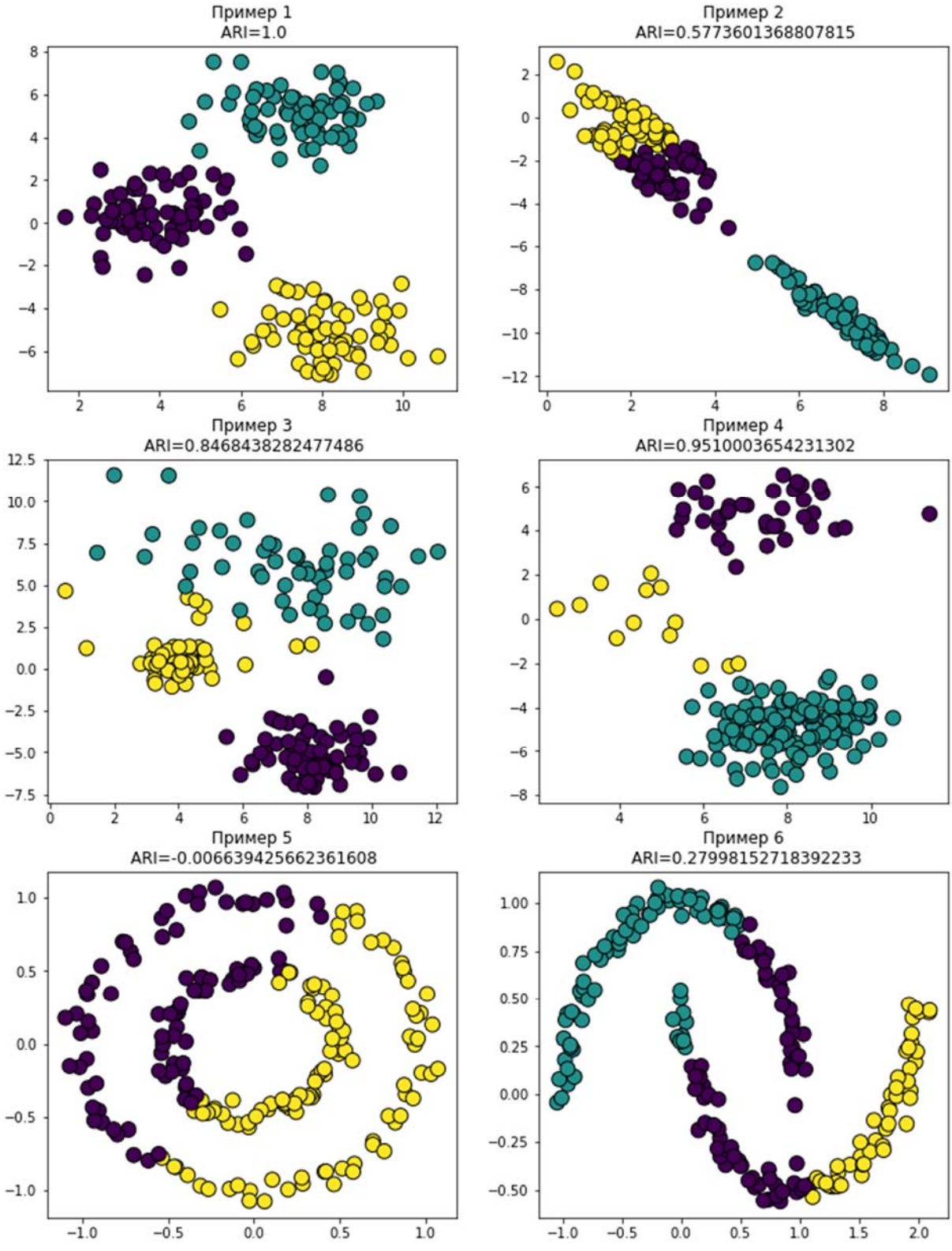


Рис. 9.2. Результаты выполнения алгоритма кластеризации
с использованием метода k-средних

Описание класса модели:

class sklearn.cluster.MiniBatchKMeans(n_clusters, *, init, max_iter, batch_size, verbose, compute_labels, random_state, tol, max_no_improvement, init_size, n_init, reassignment_ratio)

Основные параметры класса модели:

n_clusters (тип данных – int; значение по умолчанию – 8) – указывает алгоритму количество кластеров, которые нужно сформировать (количество центроидов, которые необходимо создать);

init (тип данных – str; значение по умолчанию – ‘k-means++’) – указывает алгоритму метод инициализации (возможные варианты: {‘random’, ‘ndarray’, ‘callable’}, при ‘k-means++’: начальные центры кластеров для ускорения сходимости выбираются с k-средним; ‘random’: начальные центры кластеров выбираются случайным образом; ‘ndarray’: начальные центры кластеров передаются в массиве ndarray размерностью [n_clusters, n_features]; ‘callable’у’: начальные центры кластеров задаются с помощью функции пользователя);

n_init (тип данных – int; значение по умолчанию – 3) – указывает алгоритму число подходов к определению начальных центров кластеров;

max_iter (тип данных – int; значение по умолчанию – 100) – указывает алгоритму максимальное количество итераций;

batch_size (тип данных – int; значение по умолчанию – 100) – указывает алгоритму размер мини-пакетов;

compute_labels (тип данных – bool; значение по умолчанию – True) – указывает алгоритму на необходимость вычисления результатов кластеризации для всего набора данных после схождения алгоритма (True) или в процессе выполнения каждой итерации (False);

tol (тип данных – int; значение по умолчанию – 0.0) – указывает алгоритму точность решения;

random_state (тип данных – int; значение по умолчанию – None) указывает алгоритму параметр генератора случайных чисел для определения начальных центров кластеров.

Основные атрибуты класса модели:

cluster_centers_ (тип данных – массив размерностью [n_clusters, n_features]) – предназначен для хранения рассчитанных координат кластерных центров;

labels_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения меток каждого объекта (compute_labels);

inertia_ (тип данных – float) – предназначен для хранения суммы квадратов расстояний от образцов до ближайшего центра кластера.

Основные методы класса модели:

fit(X, y[,sample_weight]) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – не используется (включен для совместимости); sample_weight (тип данных – массив размерностью [n_samples,]) – необязательный аргумент, определяющий индивидуальные веса данных в выборке;

predict(X) – вычисление кластерных центров и прогнозирование параметров модели для выборки данных X.

Пример использования метода k-средних с применением мини-пакетов для кластеризации при n_clusters=3, batch_size=6, random_state=0 приведен на рисунках 9.3 (программный код алгоритма – для примера 1 распределения данных) и 9.4 (результаты работы алгоритма – для всех примеров).

```
from sklearn.cluster import MiniBatchKMeans
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model = MiniBatchKMeans(n_clusters=3, batch_size=6, random_state=0)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)
plt.show()
```

Рис. 9.3. Программный код алгоритма классификации с использованием метода k-средних с применением мини-пакетов

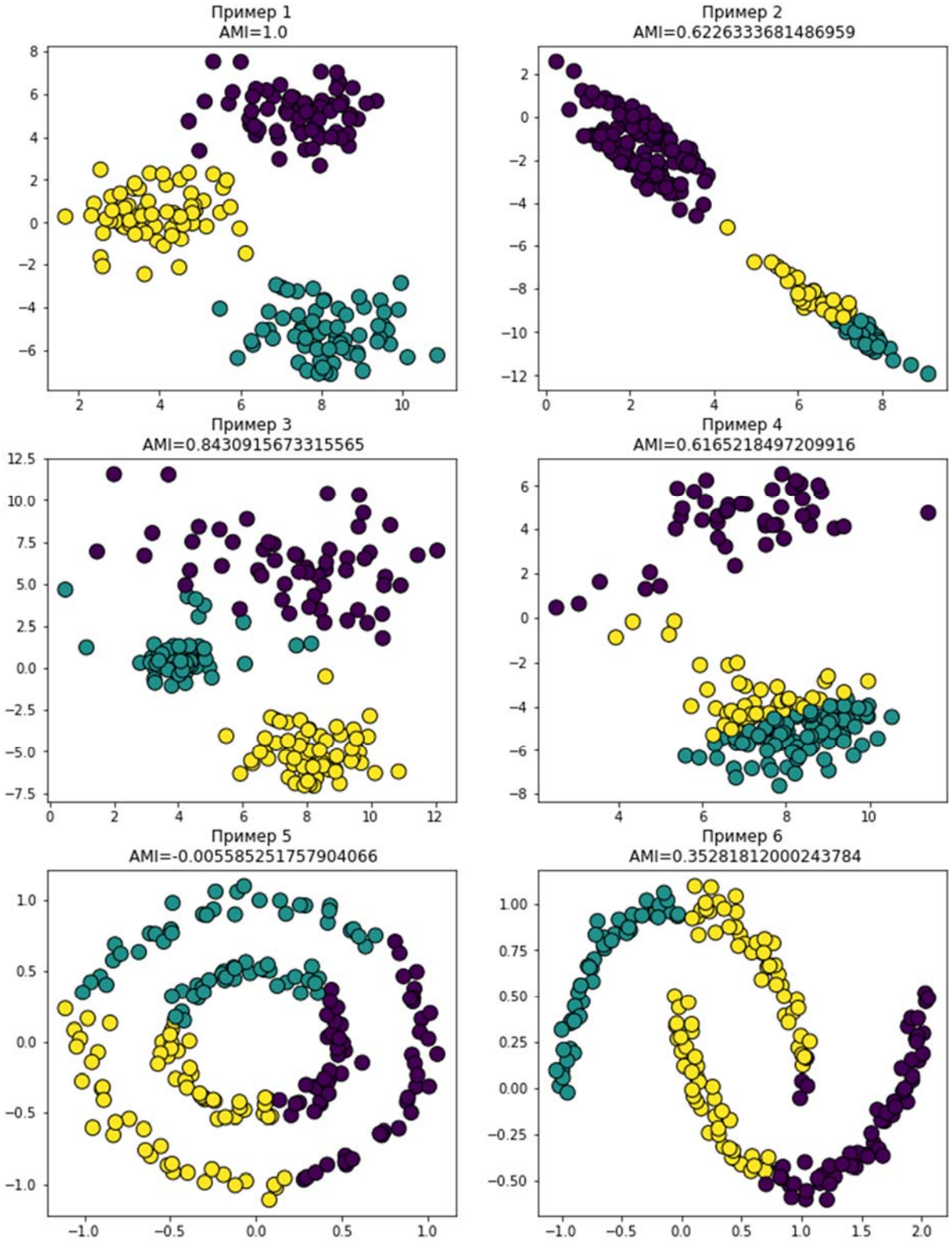


Рис. 9.4. Результаты выполнения алгоритма кластеризации с использованием метода k-средних с применением мини-пакетов

9.3. Модели иерархической кластеризации

В Scikit-learn для работы с моделями иерархической кластеризации включен класс sklearn.cluster, который предоставляет возможность работы со следующими методами её осуществления:

а) агломеративным методом (cluster.AgglomerativeClustering);

б) методом сбалансированного итеративного сокращения и кластеризации с помощью иерархий (cluster.Birch).

Агломеративный метод выполняет иерархическую кластеризацию, используя восходящий подход: каждое наблюдение начинается в собственном кластере, и кластеры последовательно объединяются.

В качестве критерия объединения используются различные метрики, определяющие стратегию слияния: расстояние Уорда (минимум суммы квадратов разностей во всех кластерах), минимум максимального расстояния между парами кластеров, минимум среднего расстояние между всеми парами кластеров и минимум расстояния между ближайшими парами кластеров.

Для кластеризации агломеративным методом в scikit-learn используется класс sklearn.cluster.AgglomerativeClustering s

Описание класса модели:

class sklearn.cluster.AgglomerativeClustering(n_clusters, *, affinity, memory, connectivity, compute_full_tree, linkage, distance_threshold)

Основные параметры класса модели:

n_clusters (тип данных – int; значение по умолчанию – 2) – указывает алгоритму количество кластеров, которые нужно сформировать;

affinity (тип данных – str; значение по умолчанию – ‘euclidean’) – указывает алгоритму метрику, которую необходимо использовать для вычисления связи между кластерами (возможные варианты: {‘euclidean’, ‘l1’, ‘l2’, ‘manhattan’, ‘cosine’, ‘precomputed’};

connectivity (тип данных – str; значение по умолчанию – None) – указывает алгоритму матрицу смежности, которая определяет для каждой выборки соседние выборки, следующие заданной структуре данных (по умолчанию алгоритм иерархической кластеризации не структурирован);

compute_full_tree (тип данных – str или bool; значение по умолчанию – ‘auto’) – указывает алгоритму на необходимость построение дерева в n_clusters (True);

linkage (тип данных – str; значение по умолчанию – ‘ward’) – указывает алгоритму критерий объединений пары кластеров (возможные варианты: {‘ward’, ‘complete’, ‘average’, ‘single’}, где ‘ward’ – минимум суммы квадратов разностей во всех кластерах (расстояние Уорда), ‘complete’ – минимум максимального расстояния между парами кластеров, ‘average’ – минимум среднего расстояние между всеми парами кластеров, ‘single’ – минимум расстояния между ближайшими парами кластеров);

distance_threshold (тип данных – float; значение по умолчанию – None) – указывает алгоритму пороговое значение расстояния между кластерами, выше которого кластеры не будут объединены.

Основные атрибуты класса модели:

n_clusters_(тип данных – int) – количество кластеров, найденных алгоритмом. Если distance_threshold = None, n_clusters_= n_clusters;

labels_ (тип данных – массив размерностью [n_samples]) – предназначен для хранения меток каждого объекта;

n_leaves_ (тип данных – int) – количество листьев в иерархическом дереве;

n_connected_components_ (тип данных – int) – предполагаемое количество связанных компонентов на графике;

children_ (тип данных – массив размерностью [n_samples-1, 2]) – предназначен для хранения информации о потомках каждого не являющегося листом узла.

Основные методы класса модели:

fit(X, y) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – не используется (включен для совместимости);

fit_predict (X, y) – построение модели кластеризации и прогнозирование параметров модели для выборки данных X.

Пример использования агломеративного метода для кластеризации при n_clusters=3, affinity= 'l2', linkage='complete' приведен на рисунках 9.5 (результаты работы алгоритма – для всех примеров) и 9.6 (программный код алгоритма – для примера 1 распределения данных).

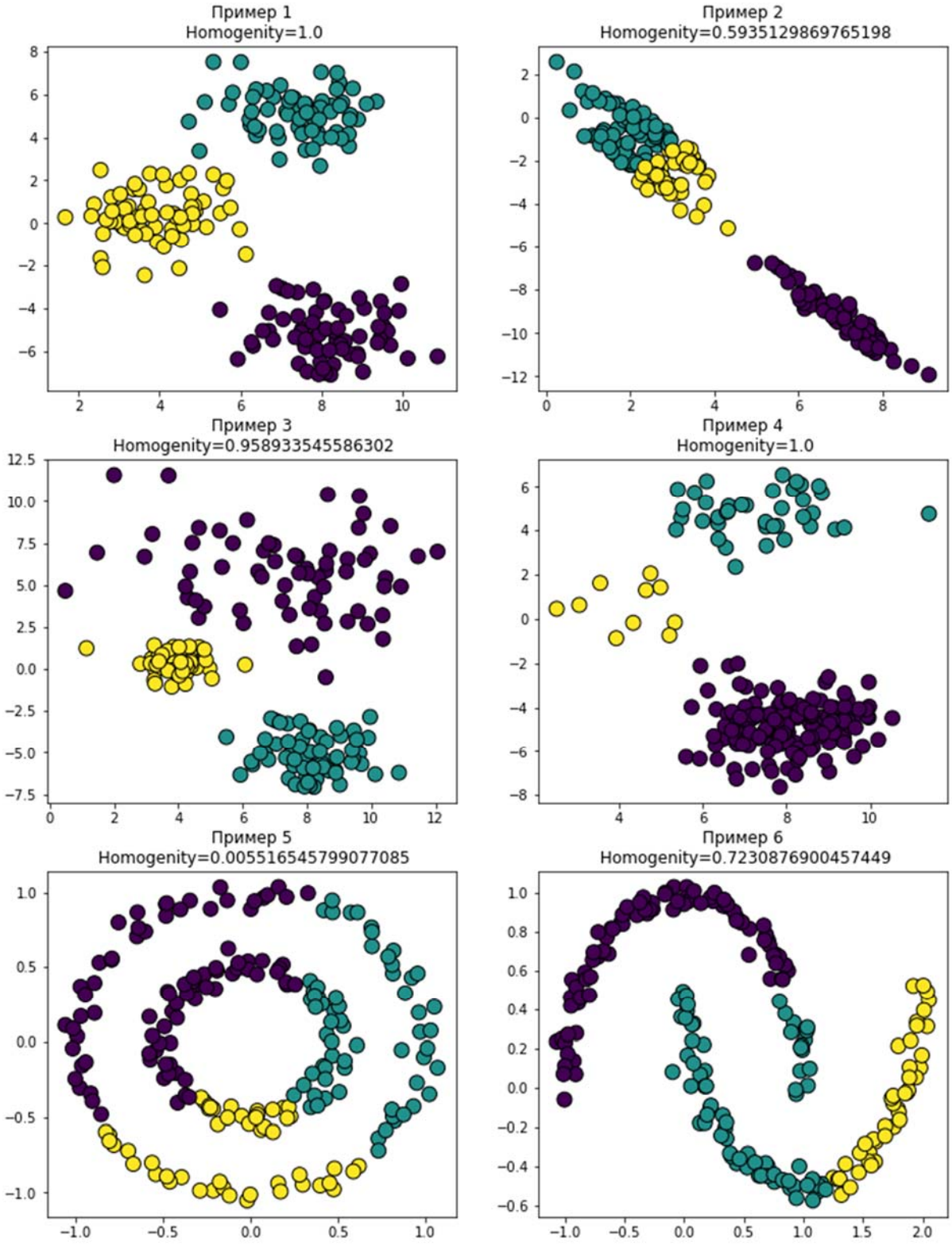


Рис. 9.5. Результаты выполнения алгоритма кластеризации с использованием агломеративного метода

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model = AgglomerativeClustering(n_clusters=3,
                                affinity='l2',
                                linkage='complete')
model.fit(X)
y_pred=model.fit_predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)
```

Рис. 9.6. Программный код алгоритма классификации с использованием агломеративного метода

Для визуализации результатов иерархической кластеризации используют дендрограммы, которые показывают степень близости отдельных объектов и кластеров, а также наглядно демонстрируют в графическом виде последовательность их объединения или разделения.

Алгоритм построения кластеризации с использованием класса AgglomerativeClustering и доступного в библиотеке scipy метода cluster.hierarchy приведен на рис. 9.8, результат работы алгоритма – на рис. 9.7.

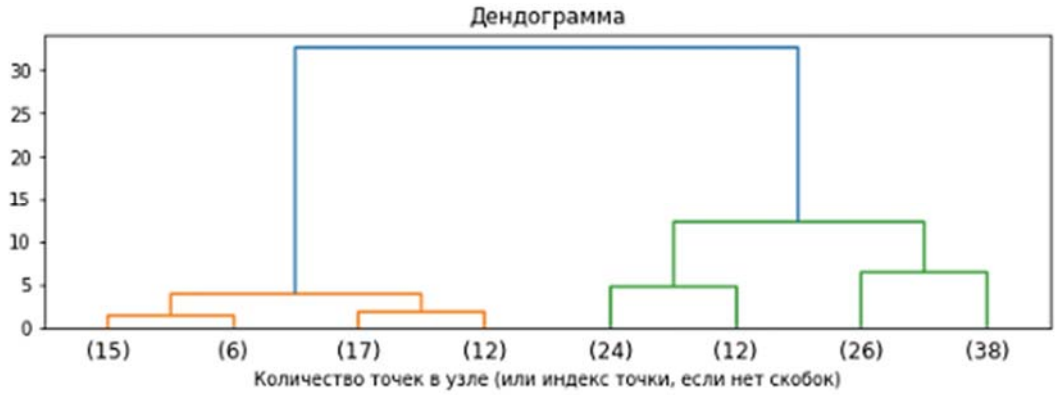


Рис. 9.7. Результаты кластеризации в форме дендограммы

```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering

def plot_dendrogram(model, **kwargs):
    # Создание матрицу смежности и ипостроение дендограммы
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count
    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
    dendrogram(linkage_matrix, **kwargs)

# генерация набора данных
iris = load_iris()
X = iris.data

# Обучение модели и прогнозирование
# для вычисления полного дерева устанавливаем distance_threshold = 0
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X)

# Визуализация (2 верхних уровня дендрограммы)
plt.figure(figsize=(10, 6))
plt.title('Дендограмма')
plot_dendrogram(model, truncate_mode='level', p=2)
plt.xlabel("Количество точек в узле (или индекс точки, если нет скобок)")
plt.show()
```

Рис. 9.8. Программный код алгоритма кластеризации с представлением результатов в форме дендограммы

Метод сбалансированного итеративного сокращения и кластеризации с помощью иерархий (BIRCH) предполагает задание глобальной функции оптимизации, при этом добавление нового объекта в кластер осуществляется на основе предыдущего значения глобальной функции, характеристик нового объекта и так называемых кластерных характеристик.

Достоинства метода BIRCH: двухступенчатая кластеризация, кластеризация больших объемов данных, работает на ограниченном объеме памяти, является локальным алгоритмом, может работать при одном сканировании входного набора данных, использует тот факт, что данные неодинаково распределены по пространству, и обрабаты-

вает области с большой плотностью как единый кластер. К недостаткам относят: работа с только числовыми данными, хорошо выделяет только кластеры сферической формы, есть необходимость в задании пороговых значений [3]. Для кластеризации методом сбалансированного итеративного сокращения и кластеризации с помощью иерархий в scikit-learn используется класс sklearn.cluster.Birch.

Описание класса модели:

class sklearn.cluster.Birch(*, threshold, branching_factor, n_clusters, compute_labels, copy)

Основные параметры класса модели:

threshold (тип данных – float; значение по умолчанию – 0.5) – указывает алгоритму пороговое значение радиуса подкластера, полученного путем слияния новой выборки и ближайшего подкластера (установка очень низкого значения способствует разделению кластеров, и очень высокого – слиянию);

branching_factor (тип данных – int; значение по умолчанию – 50) – указывает алгоритму максимальное количество подкластеров в каждом узле (если новые образцы поступают так, что количество подкластеров превышает branching_factor, то этот узел разделяется на два узла с перераспределением подкластеров в каждом, а родительский подкластер этого узла удаляется, и два новых подкластера добавляются в качестве родителей 2-х разбитых узлов);

n_clusters (тип данных – int; значение по умолчанию – 3) – указывает алгоритму количество кластеров, которые нужно сформировать после последнего шага кластеризации, на котором подкластеры из листьев рассматриваются как новые выборки;

compute_labels (тип данных – bool; значение по умолчанию – True) – указывает алгоритму на необходимость вычисления результатов кластеризации для всего набора;

copy (тип данных – bool; значение по умолчанию – True) – указывает алгоритму на необходимость вычисления результатов кластеризации для всего набора.

Основные атрибуты класса модели:

root_ (тип данных – _CFNode) – предназначен для хранения информации о корне построенного дерева;

dummy_leaf_ (тип данных – _CFNode) – предназначен для хранения информации о начальных указателях на все листья;

subcluster_centers_– предназначен для хранения информации о центроидах всех подкластеров;

subcluster_ labels_– предназначен для хранения информации о метках всех подкластеров;

labels_ (тип данных – массив размерностью [n_samples]) – предназначен для хранения меток каждого объекта.

Основные методы класса модели:

fit(X, y) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – не используется (включен для совместимости);

predict(X) – прогнозирование данных с использованием информации о центроидах подкластеров.

Пример использования метода BIRCH для кластеризации при n_clusters=None, threshold=3, branching_factor=100 приведен на рисунках 9.8 (программный код алгоритма – для примера 1 распределения данных) и 9.9 (результаты работы алгоритма – для всех примеров).

```
# Визуализация модели
def model_visualization(y_pred, y, labels, plt, m, n):
    plt.subplot(320+n)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, marker='o', s=120, edgecolor='k')
    plt.title("Пример "+str(n)+"\n "+m[0]+"="+str(m[1]))

from sklearn.cluster import Birch
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model =  Birch(n_clusters=None, threshold =3, branching_factor=100)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)
```

Рис. 9.8. Программный код алгоритма классификации
с использованием метода BIRCH

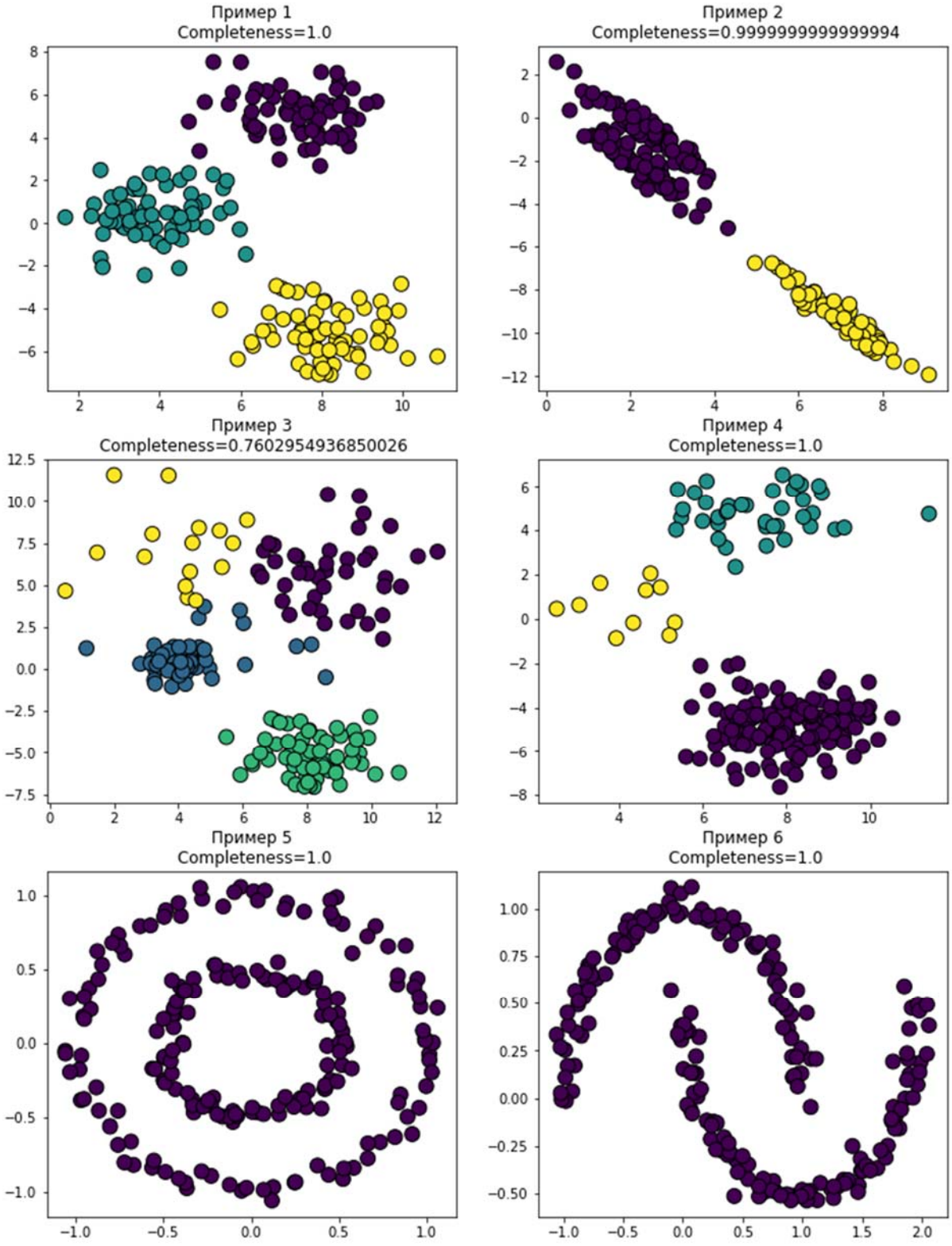


Рис. 9.9. Результаты выполнения алгоритма кластеризации с использованием метода BIRCH

9.4. Модели плотностной кластеризации

В Scikit-learn для работы с моделями плотностной кластеризации включен класс sklearn.cluster, который предоставляет возможность работы со следующими методами её осуществления:

а) методом сдвига среднего значения (cluster.MeanShift);

б) методом пространственной кластеризации для приложений с шумами (cluster.DBSCAN);

в) методом упорядочения точек для обнаружения кластерной структуры (cluster.OPTICS).

Метод сдвига среднего значения предполагает кластеризацию на основе построения непараметрической оценки плотности объектов и отнесения их к кластерам за счет смещения в направлении наивысшей плотности объектов, то есть центроида.

Плотность оценивается как суммарное влияние объектов выборки, при этом вклад каждого элемента описывается колоколообразной функцией (ядром), зависящей от расстояния до этого объекта и значения параметра сглаживания (ширины пропускания).

Объекты относятся к кластерам на основании расчета вектора «среднего сдвига», направление которого совпадает с направлением максимального роста плотности. Соответственно, кластеры соответствуют локальным максимумам функции оценки плотности (модам) [4].

Для кластеризации методом сдвига среднего значения в scikit-learn используется класс sklearn.cluster.MeanShift

Описание класса модели:

class sklearn.cluster.MeanShift(*, bandwidth, seeds, bin_seeding, min_bin_freq, cluster_all, n_jobs, max_iter)

Основные параметры класса модели:

bandwidth (тип данных – float; значение по умолчанию – None) – указывает алгоритму ширину пропускания для ядер;

seeds (массив размерностью [n_samples, n_features]; значение по умолчанию – None) – указывает алгоритму начальные положения ядер;

bin_seeding (тип данных – bool; значение по умолчанию – False) – указывает алгоритму на необходимость определения начального положения ядер на основе дискредитированной версии объектов;

cluster_all (тип данных – bool; значение по умолчанию – True) – указывает алгоритму на необходимость осуществлять кластеризацию всех объектов (при cluster_all = false); объектам, которые не прошли кластеризацию, не присваивается метка 1);

max_iter (тип данных – int; значение по умолчанию – 300) – указывает алгоритму максимальное количество итераций.

Основные атрибуты класса модели:

cluster_centers_ (тип данных – массив размерностью [n_clusters, n_features]) – предназначен для хранения расчитанных координат кластерных центров;

labels_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения меток каждого объекта;

n_iter_ (тип данных – int) – предназначен для хранения информации о количестве выполненных итераций.

Основные методы класса модели:

fit(X, y) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – не используется (включен для совместимости);

predict(X) – прогнозирование данных с использованием информации о центроидах подкластеров.

Пример использования метода сдвига среднего значения для кластеризации при bandwidth=3 (для примеров 1-4) и bandwidth=0.75 (для примеров 5, 6) приведен на рисунках 9.10 (результаты работы алгоритма – для всех примеров) и 9.11 (программный код алгоритма – для примера 1 распределения данных).

Метод пространственной кластеризации для приложений с шумами (DBSCAN) предполагает кластеризацию на основе построения оценки плотности объектов и отнесения их к кластерам на основании решения задачи поиска ближайшего соседа с фиксированным радиусом, при этом одиночные объекты, обособленные в областях с малой плотностью помечаются как выбросы [5].

Для кластеризации методом DBSCAN в scikit-learn используется класс sklearn.cluster.DBSCAN.

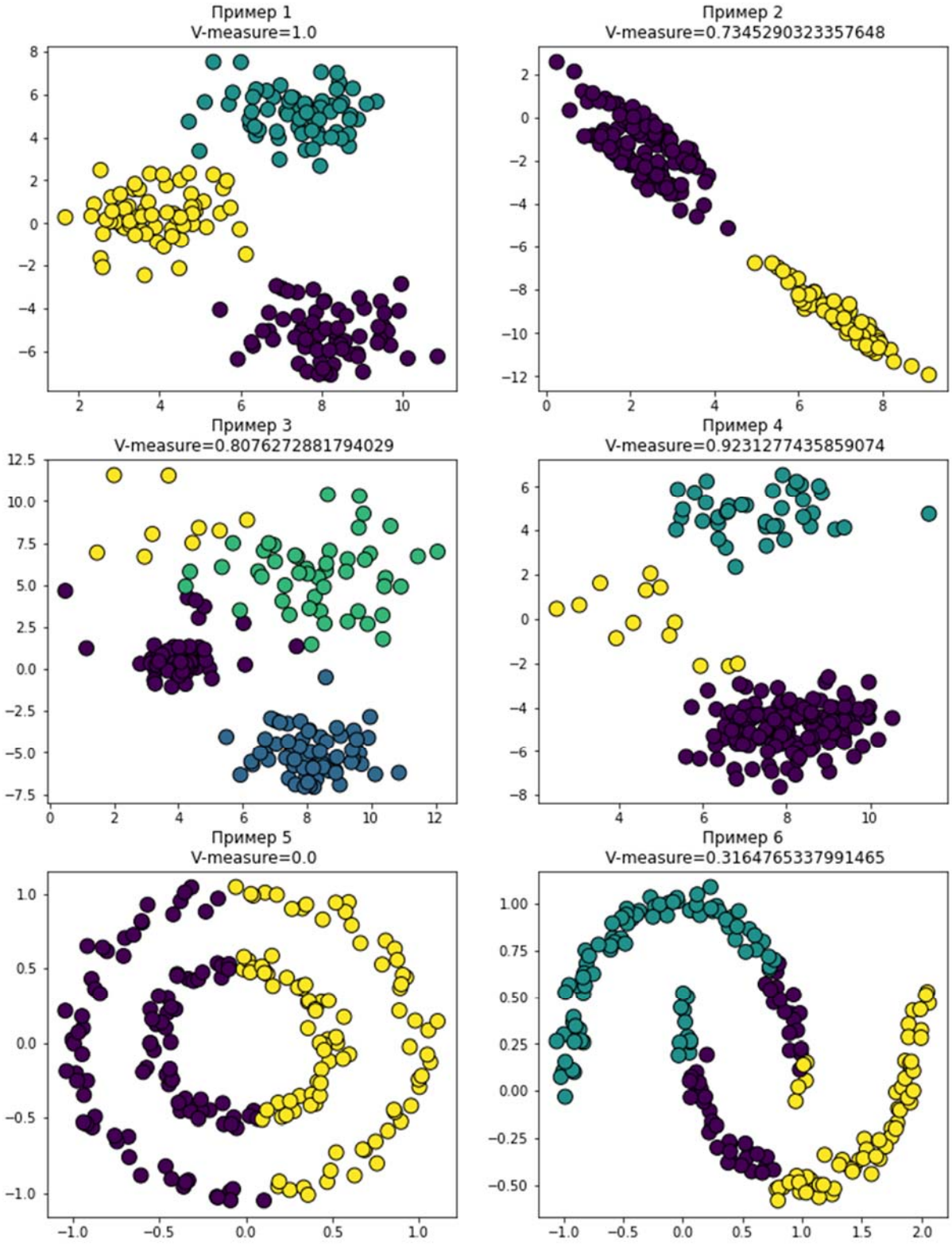


Рис. 9.10. Результаты выполнения алгоритма кластеризации
с использованием метода сдвига среднего значения

```
from sklearn.cluster import MeanShift
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model =  MeanShift(bandwidth=3)
model.fit(X)
y_pred=model.predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)
```

Рис. 9.11. Программный код алгоритма классификации
с использованием метода сдвига среднего значения

Описание класса модели:

class sklearn.cluster.DBSCAN(eps, *, min_samples, metric, metric_params, algorithm, leaf_size, p, n_jobs)

Основные параметры класса модели:

eps (тип данных – float; значение по умолчанию – 0.5) – указывает алгоритму максимальное расстояние между двумя объектами, чтобы один считался соседним с другим;

min_samples (тип данных – int; значение по умолчанию – 5) – указывает алгоритму минимальное количество объектов-соседей необходимых для их слияния в кластер (с учетом самого объекта);

metric (тип данных – str, значение по умолчанию – ‘euclidean’) указывает алгоритму метрику расстояния, которую необходимо использовать;

algorithm (тип данных – str, значение по умолчанию – ‘auto’) указывает алгоритму способ вычисления ближайших соседей: (возможные варианты: {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’});

Основные атрибуты класса модели:

core_sample_indices_ (тип данных – массив размерностью [n_core_samples,]) – предназначен для хранения индексов ядер;

components_ (тип данных – массив размерностью [n_core_samples, n_features]) – предназначен для хранения копии каждого ядра, найденного в процессе обучения;

labels_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения меток каждого объекта (выбросы (шумы) помечаются меткой -1);

Основные методы класса модели:

fit(X, y) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – не используется (включен для совместимости);

fit_predict (X, y) – построение модели кластеризации и прогнозирование параметров модели для выборки данных X.

Пример использования метода DBSCAN для кластеризации при eps=1.75, min_samples=20 (для примеров 1-4), eps=0.3, min_samples=20 (для примера 5) и eps=0.3, min_samples=10 (для примера 6) приведен на рисунках 9.12 (программный код алгоритма – для примера 1 распределения данных) и 9.13 (результаты работы алгоритма – для всех примеров).

Метод упорядочения точек для обнаружения кластерной структуры (OPTICS) обобщает метод DBSCAN, решая проблему обнаружения содержательных кластеров в данных, имеющих различные плотности, за счет упорядочивания объектов точек и расстояний достижимости таким образом, что пространственно близкие точки становятся соседними в упорядочении [6].

```
from sklearn.cluster import DBSCAN
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model =  DBSCAN(eps=1.75, min_samples=20)
y_pred=model.fit_predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)
```

Рис. 9.12. Программный код алгоритма классификации
с использованием метода DBSCAN

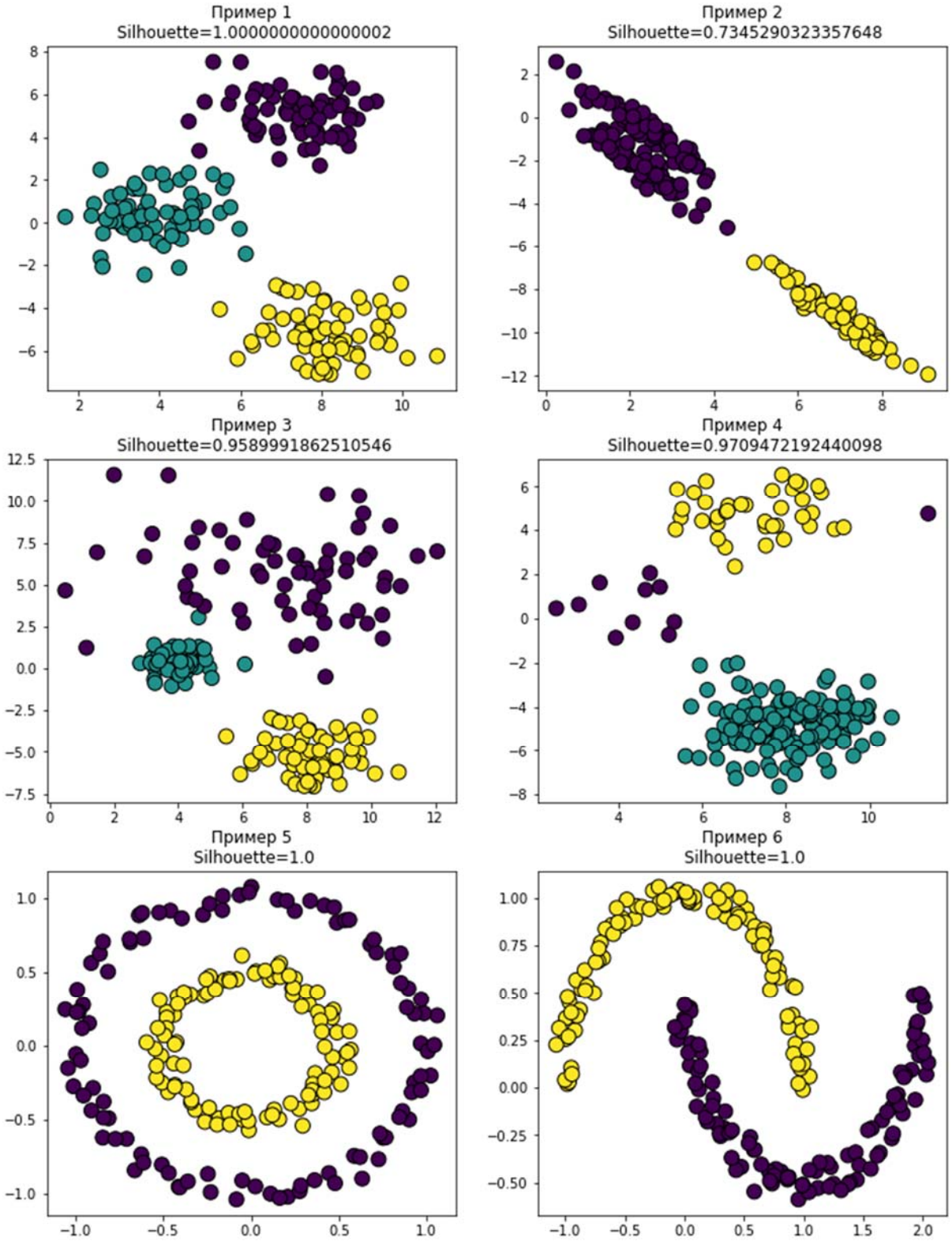


Рис. 9.13. Результаты выполнения алгоритма кластеризации с использованием метода DBSCAN

Для кластеризации методом OPTICS в scikit-learn используется класс sklearn. cluster.OPTICS

Описание класса модели:

class sklearn.cluster.OPTICS (*, min_samples, max_eps, metric, p, metric_params, cluster_method, eps, xi, predecessor_correction, min_cluster_size, algorithm, leaf_size, n_jobs)

Основные параметры класса модели:

min_samples (тип данных – int или float; значение по умолчанию – 5, ограничение – для int: $min\_samples \in (1;\infty)$, для float: $min\_samples \in [0;1]$) – указывает алгоритму минимальное количество объектов-соседей необходимых для их слияния в кластер (либо абсолютное число, либо доля от общего количества);

max_eps (тип данных – float; значение по умолчанию – 0.5) – указывает алгоритму максимальное расстояние между двумя объектами, чтобы один считался соседним с другим;

metric (тип данных – str, значение по умолчанию – ‘minkowski’) – указывает алгоритму метрику расстояния, которую необходимо использовать;

cluster_method (тип данных – str, значение по умолчанию – ‘xi’) – указывает алгоритму метод формирования кластеров (возможные варианты: {‘xi’, ‘dbscan’});

eps (тип данных – float; значение по умолчанию – None (eps= max_eps)) – указывает алгоритму максимальное расстояние между двумя объектами, чтобы один считался соседним с другим (используется при cluster_method=' dbscan ');;

xi (тип данных – int или float; значение по умолчанию – 0.05, ограничение – для int: $min\_samples \in (1;\infty)$, для float: $min\_samples \in [0;1]$) – указывает алгоритму определяющую границу кластера минимальную крутизну на графике достижимости (используется при cluster_method='xi');

min_cluster_size (тип данных – int или float; значение по умолчанию – None, ограничение – для int: $min\_samples \in (1;\infty)$, для float: $min\_samples \in [0;1]$) – указывает алгоритму минимальное количество объектов в кластере или их долю от общего количества (используется

при cluster_method='xi', при min_cluster_size =None используется значение min_samples);

algorithm (тип данных – str, значение по умолчанию – ‘auto) указывает алгоритму способ вычисления ближайших соседей: (возможные варианты: {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’});

Основные атрибуты класса модели:

labels_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения меток каждого объекта (выбросы (шумы) помечаются меткой -1);

ordering_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения списка индексов объектов в упорядоченном списке;

cluster_hierarchy_ (тип данных – массив размерностью [n_samples, 2]) – предназначен для хранения информации об кластерах в виде координат начального и конечного индекса упорядоченного списка объектов;

reachability_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения расстояний достижимости для каждого объекта в упорядоченом списке;

core_distances_ (тип данных – массив размерностью [n_samples,]) – предназначен для хранения в упорядоченном списке объектов расстояний от них до центральной точки;

Основные методы класса модели:

fit(X, y) – построение модели кластеризации, где X (тип данных – массив размерностью [n_samples, n_features]) – набор данных для обучения, y – не используется (включен для совместимости);

fit_predict (X, y) – построение модели кластеризации и прогнозирование параметров модели для выборки данных X.

Пример использования метода OPTICS для кластеризации при min_samples=20 для всех примеров приведен на рисунках 9.14 (результаты работы алгоритма – для всех примеров) и 9.15 (программный код алгоритма – для примера 1 распределения данных).

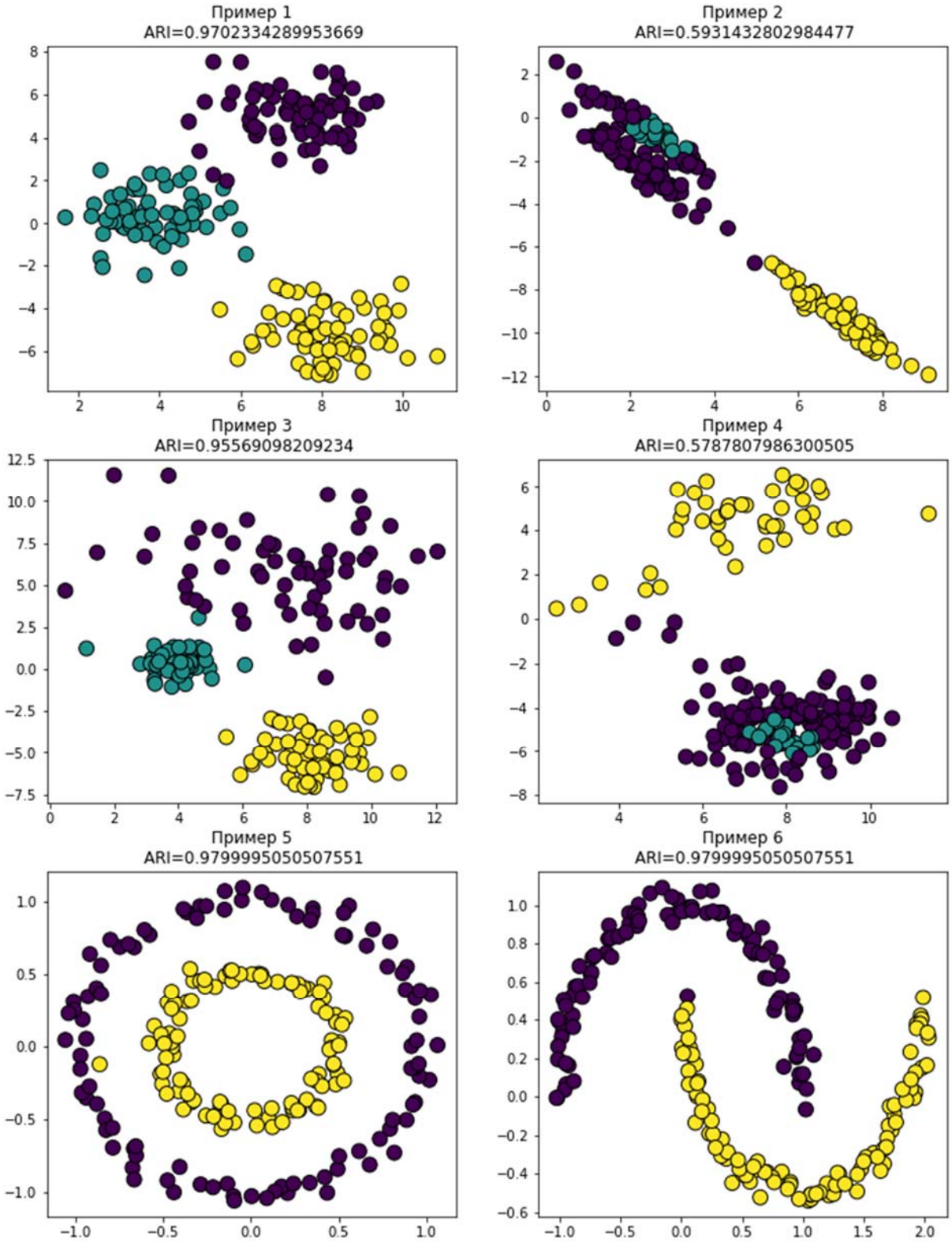


Рис. 9.14. Результаты выполнения алгоритма кластеризации с использованием метода OPTICS

```
# Визуализация модели
def model_visualization(y_pred, y, labels, plt, m, n):
    plt.subplot(320+n)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, marker='o', s=120, edgecolor='k')
    plt.title("Пример "+str(n)+"\n "+m[0]+"="+str(m[1]))

from sklearn.cluster import OPTICS
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 16))

# Пример 1
# генерация набора данных
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=200, random_state=150)
# Обучение модели и прогнозирование
model =  OPTICS(min_samples=20)
y_pred=model.fit_predict(X)
# Оценка качества кластеризации
m=get_metrics (y, model.labels_)
# Визуализация модели
model_visualization (y_pred, y, model.labels_, plt, m, 1)
```

Рис. 9.15. Программный код алгоритма классификации
с использованием метода OPTICS

Вопросы для обсуждения

1. Проблемы практического применения моделей кластеризации
2. Сравнение методов кластеризации. Подходы к классификации методов кластеризации
3. Методы итеративной кластеризации: преимущества и недостатки, область применения
4. Методы иерархической кластеризации: преимущества и недостатки, область применения
5. Методы плотностной кластеризации: преимущества и недостатки, область применения
6. Методы борьба с переобучением в различных методах кластеризации
7. Проблема определения оптимальных параметров кластеризации
8. Метрики качества классификации:

Практические задания

Задание 9.1.

На основании учебных данных о стоимости жилья в г. Бостоне (sklearn.datasets.load_boston) необходимо сегментировать объекты недвижимости по трех любым их признакам.

Кластеризацию провести одним из методов итеративной кластеризации, иерархической кластеризации и плотностной кластеризации. Сделать выводы об эффективности применения каждого из них.

Задание 9.2.

На основании учебных данных свойствах вина (sklearn.datasets.load_wine) необходимо разделить напитки на классы.

Кластеризацию провести одним из методов итеративной кластеризации, иерархической кластеризации и плотностной кластеризации. Включенные в набор метки классов использовать для проверки результатов. Сделать выводы об эффективности применения каждого из них.

Задание 9.3

На основании набора данных, полученного при решении задания 7, приведенного в теме 7, проведите исследование и сравнительные эксперименты для всех рассмотренных методов итеративной кластеризации.

При проведении экспериментов использовать различные допустимые комбинации параметры методов. Сделайте выводы.

Задание 9.4

На основании набора данных, полученного при решении задания 7, приведенного в теме 7, проведите исследование и сравнительные эксперименты для всех рассмотренных методов иерархической кластеризации.

При проведении экспериментов использовать различные допустимые комбинации параметры методов. Сделайте выводы.

Задание 9.5

На основании набора данных, полученного при решении задания 7, приведенного в теме 7, проведите исследование и сравнительные эксперименты для всех рассмотренных методов плотностной кластеризации.

При проведении экспериментов использовать различные допустимые комбинации параметры методов. Сделайте выводы.

Задание 9.6

На основании набора данных о покупках в розничном магазине, занимающемся продажей подарков и сувениров через интернет-сайт, осуществите кластеризацию заказов и покупателей.

Источник данных:
https://archive.ics.uci.edu/ml/datasets/online+retail,

1) загрузить наборы данных;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать несколько методов построения модели кластеризации (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование моделей кластеризации с использованием различных комбинаций их параметров;

7) оценить качество различных по параметрам вариантов моделей по выбранной метрике;

8) осуществить кластеризацию на основании варианта модели с лучшим значением метрики и сделать вывод

Задание 9.7

На основании статистических данных за последний год разделить все регионы Российской Федерации на группы в зависимости от уровня развития информационного общества.

Источник данных: https://showdata.gks.ru (группа показателей раздела «Информационное общество»)

Необходимо:

1) загрузить наборы данных и объединить их в один;

2) провести исследование данных;

3) выполнить предварительную обработку данных (при необходимости);

4) выбрать несколько методов построения модели кластеризации (решение обосновать);

5) выбрать метрику оценки качества модели (решение обосновать);

6) осуществить обучение и тестирование моделей кластеризации с использованием различных комбинаций их параметров;

7) оценить качество различных по параметрам вариантов моделей по выбранной метрике;

8) осуществить кластеризацию на основании варианта модели с лучшим значением метрики и сделать вывод.

Библиографический список

1. Кластерный анализ // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ Кластерный_анализ

2. Иерархическая кластеризация // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Иерархическая _кластеризация

3. Нейский И.М. Классификация и сравнение методов кластеризации. // Научно-образовательный кластер CLAIM. – Режим доступа: http://it-claim.ru/Persons/Neyskiy/Article2_Neiskiy.pdf

4. Рылов С.А. Непараметрический алгоритм кластеризации для сегментации изображений на основе комбинации сеточного подхода и процедуры среднего сдвига // Институт вычислительных технологий СО РАН, Новосибирск – Режим доступа: http://ceur-ws.org/Vol-2033/29_paper.pdf

5. DBSCAN // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/DBSCAN

6. Алгоритм кластеризации OPTICS // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Алгоритм_ кластеризации_OPTICS

7. Замятин А. В. Введение в интеллектуальный анализ данных: учебное пособие / А. В. Замятин; Нац. исслед. Том. гос. ун-т. - Томск:

Издательский Дом Томского государственного университета, 2016. URL: http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000529594

8. Силен Дэви, Мейсман Арно, Али Моха мед. Основы Data Science и Big Data. Python и наука о данных. – СПб.: Питер, 2017. – 336 с: ил. – (Серия «Библиотека программиста»). – ISBN 978-5-496-02517-1

9. Лесковец, Ю. Анализ больших наборов данных / Лесковец Ю., Раджараман А. , Джеффри Д. Ульман - Москва : ДМК Пресс, 2016. - 498 с. - ISBN 978-5-97060-190-7. - Текст : электронный // ЭБС "Консультант студента": [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN9785970601907.html

10. Жерон, Орельеа. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем.: Пер. с англ. – СпБ.; ООО "Альфа-книга", 2018. – 688 с.: ил. – Парал. тит. англ. – ISBN 978-5-9500296-2-2 (рус.)

11. Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение. – СПб.: Питер, 2018. – 576 с: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7

12. Машинное обучение // Википедия: [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Машинное_обучение

ЗАКЛЮЧЕНИЕ

Стремительная технологическая эволюция последних лет в сфере информационно-коммуникационных технологий позволила сформировать существенный задел в части развитой программно-аппаратной инфраструктуры, поддерживающей накопление и постоянное пополнение архивов данных различной природы и назначения.

Обостряющаяся конкурентная борьба в различных областях человеческой деятельности (бизнесе, медицине, корпоративном управлении и др.) и сложность внешней среды делают крайне востребованными подходы к экспертному использованию имеющихся данных для повышения обоснованности и оперативности принятия управленческих решений.

При этом не всегда сегодня возможно непосредственное эффективное применение хорошо проработанного и известного аппарата теории вероятности или математической статистики без учета особенностей конкретной предметной области, компьютерных наук (включая детали хранения и обработки данных, алгоритмов машинного обучения и т.п.), специфики современных информационных технологий.

Именно поэтому относительно недавно стала привлекать особое внимание область, связанная с высокопроизводительной интеллектуальной аналитической обработкой данных, направленная на то, чтобы оперативно извлекать из значительных массивов накопленных и поступающих данных ценные экспертные знания, поддерживая эффективную управленческую деятельность.

Учитывая междисциплинарный характер этой предметной области, ее глубину и ярко выраженную прикладную направленность, до сих пор существует определенный дефицит систематизированных представлений о ней, на устранение которых в некоторой степени направлен материал пособия.

ОГЛАВЛЕНИЕ

Учебное издание

ВИНОГРАДОВ Дмитрий Викторович

ИНТЕЛЛЕКТУАЛЬНЫЙ
АНАЛИЗ ДАННЫХ

Учебное пособие

Издается в авторской редакции

Подписано в печать 22.06.21.
Формат 60×84/16. Усл. печ. л. 15,11. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.