1. **Понятие машинного обучения. Отличие машинного обучения от других областей программирования.**

Машинное обучение – это наука о разработке алгоритмов и статистических моделей, которые компьютерные системы используют для выполнения задач без явных инструкций, полагаясь вместо этого на шаблоны и логические выводы. Компьютерные системы используют алгоритмы машинного обучения для обработки больших объемов статистических данных и выявления шаблонов данных. Таким образом, системы могут более точно прогнозировать результаты на основе заданного набора входных данных. Машинное обучение (ML) — это направление искусственного интеллекта (ИИ), сосредоточенное на создании систем, которые обучаются и развиваются на основе получаемых ими данных.

Основная идея состоит в том, что компьютер не просто может использовать заранее написанный алгоритм, но и способен учиться, выявляя закономерности в имеющихся данных и принимая решения с минимальным участием человека.

Что делает программист? Он разрабатывает алгоритм; пишет код, используя этот алгоритм; использует входные данные и получает результат, применив разработанный алгоритм. Чтобы создать алгоритм, нужно учесть множество параметров, обработать большой объем данных и сделать это вручную человеку просто невозможно. Таким образом, программирование позволяет создать примерный алгоритм, основанный на ограниченном наборе данных, которые способен обработать человек.

Что делает Data Scientist? Он собирает статистические данные, чтобы создать полуавтоматическую модель; вводит крупные наборы данных в различные алгоритмы машинного обучения; на выходе получает модель, которая способна создавать новые рекомендации. Дата-инженер может подключиться к коллеге, чтобы доработать обучающий алгоритм и получить разные модели. Выходит, разница еще и в том, что в отличие от программирования, с МО не нужно самостоятельно строить модель. Эта задача возлагается на алгоритмы МО, правки в которые может внести дата-инженер.

Также отличием является объем данных, который влияет на точность модели и подвергается значительным ограничениям в программировании, однако может быть использован во всей своей полноте в МО. Прибегая к МО, можно использовать столько параметров, сколько необходимо.

1. **Классификация задач машинного обучения. Примеры задач из различных классов.**

Задача машинного обучения — это тип прогноза или вывода, основанный на возникшей проблеме или на вопросе, а также доступных данных. Например, задача классификации назначает данные категориям, а задача кластеризации группирует данные в соответствии со сходством. Задачи машинного обучения полагаются на шаблоны в данных, а не на явное программирование.

**Двоичная классификация.** Задача контролируемого машинного обучения, которая прогнозирует распределение элементов данных по двум классам. На вход алгоритма классификации подается набор примеров с метками, каждая из которых представляет собой целое число 0 или 1. Результатом работы алгоритма двоичной классификации является классификатор, который умеет прогнозировать класс для новых экземпляров без метки.

* Распределение комментариев Twitter по тональности — позитивные или негативные.
* Диагностика пациента на наличие определенной болезни.
* Определение того, содержит ли фотография определенный элемент.

**Многоклассовая классификация.** Задача контролируемого машинного обучения, которая прогнозирует распределение экземпляров данных по нескольким классам. На вход алгоритма классификации подается набор примеров с метками. Каждая метка обычно запускается как текст. Затем она запускается через TermTransform, который преобразует ее в тип ключа (числовой). Результатом работы алгоритма классификации является классификатор, который умеет прогнозировать класс для новых экземпляров без метки.

* определение категорий рейсов, например "ранние", "вовремя" или "с опозданием";
* распределение отзывов о фильме по категориям "позитивный", "нейтральный" или "негативный";

**Регрессия.** Задача контролируемого машинного обучения, которая прогнозирует значение метки по набору связанных компонентов. Метка здесь может принимать любое значение, а не просто выбирается из конечного набора значений, как в задачах классификации. На вход алгоритма регрессии подается набор примеров с метками известных значений. Результатом работы алгоритма регрессии является функция, которая умеет прогнозировать значения метки для любого нового набора входных компонентов.

* прогнозирование цен на дома по таким атрибутам, как количество комнат, расположение и размер;
* прогнозирование продаж товара в зависимости от рекламного бюджета.

**Кластеризация.** Задача неконтролируемого машинного обучения, которая группирует отдельные экземпляры данных в кластеры со сходными характеристиками. Кластеризацию можно также использовать для определения в наборе данных связей, которые невозможно логически отследить просмотром или наблюдением данных. Входные и выходные данные для алгоритма кластеризации зависят от выбранного метода. Вы можете выбрать подход на основе распространения, центроида, возможности подключения или плотности.

* распределение посетителей гостиниц на сегменты, исходя из привычек и характеристик выбора гостиниц;
* определение сегментов и демографических характеристик для клиентов, чтобы создавать целевые рекламные кампании;
* определение категорий запасов по параметрам производства.

**Обнаружение аномалий.** Эта задача создает модель обнаружения аномалий с помощью анализа главных компонентов (PCA). Обнаружение аномалий на основе PCA позволяет создавать модели в сценариях, где легко получить данные для обучения из одного класса, такого как допустимые транзакции, однако получить достаточную выборку аномальных значений затруднительно. PCA выполняется путем анализа данных с несколькими переменными. Он выполняет поиск корреляции между переменными и определяет сочетание значений, которые лучше всего фиксируют различия результатов. Эти объединенные значения компонентов используются для создания более компактных компонентов, называемых главными компонентами.

* Выявление потенциально мошеннических транзакций.
* Изучение шаблонов, которые указывают, что произошли сетевые атаки.
* Поиск аномальных кластеров пациентов.
* Проверка значений, введенных в систему.

**Ранжирование.** Задача ранжирования создает средство ранжирования на основе набора примеров с метками. Этот набор примеров состоит из групп экземпляров, которые могут быть оценены с заданными критериями. Метки ранжирования для каждого экземпляра — { 0, 1, 2, 3, 4 }. Средство ранжирования обучается ранжировать новые группы экземпляров с неизвестными оценками для каждого экземпляра. Входные данные метки должны иметь тип key или Single.

**Прогнозирование.** Задача прогнозирования использует предыдущие данные временных рядов, чтобы делать прогнозы о будущем поведении. Сценарии, применимые к прогнозированию, включают прогнозирование погоды, сезонные прогнозы продаж и диагностическое обслуживание.

**Классификация изображений.** Задача контролируемого машинного обучения, которая прогнозирует распределение изображений по нескольким классам. Результатом работы алгоритма классификации изображений является классификатор, который можно использовать для прогнозирования класса новых изображений. Задача классификации изображений — это тип классификации по нескольким классам.

**Обнаружение объектов.** Задача контролируемого машинного обучения, которая используется для прогнозирования класса (категории) изображения, а также предоставляет ограничивающий прямоугольник для категории внутри изображения. Вместо классификации одного объекта в изображении, обнаружение объектов может определить несколько объектов в изображении. Примеры обнаружения объектов:

* обнаружение автомобилей, дорожных знаков или людей на изображениях дорог;
* обнаружение дефектов на изображениях продуктов;
* обнаружение проблемных участков на изображениях рентгеновских снимков.

1. **Основные понятия машинного обучения: набора данных, объекты, признаки, атрибуты, модели, параметры.**

Машинное обучение — это численная оптимизация параметрических моделей для описания определенного набора данных. Машинное обучение является частью более широкого понятия - искусственного интеллекта. Искусственный интеллект — это способ заставить компьютер решить определенную задачу без описания явного алгоритма решения задачи. Интеллектуальные технологии позволяют написать программу без явного алгоритма решения задач.

**Датасет** — это набор данных, используемый для обучения моделей. **Объекты** — это элементарные сущности, которые мы изучаем, объекты реального мира, измерения, наблюдения. (**строка датасета**). Каждый объект характеризуется набором атрибутов. В датасете у всех объектов одинаковый набор атрибутов, а значения - разные. Атрибут или переменная — это свойство объектов в датасете, **признак** — это **колонка данных**, которая подается на вход модели машинного обучения.

Машинное обучение позволяет решать такие задачи, которые считались невозможными в классическом программировании. Машинное обучение тесно связано с обработкой больших массивов данных. Частью машинного обучения является глубокое обучение - оно имеет дело с созданием многослойных искусственных нейронных сетей.

Предлагаются два типа наборов в зависимости от их использования в обучении — FileDatasets и TabularDatasets. Оба типа данных можно использовать в рабочих процессах Машинного обучения Azure, в которые включены оценщики, AutoML, hyperDrive и конвейеры. FileDataset ссылается на один или несколько файлов, размещенных в хранилищах данных или доступных по общедоступным URL-адресам. Если данные уже очищены и готовы к использованию в обучающих экспериментах, файлы можно загрузить или подключить в вычислительный процесс, как объект FileDataset. TabularDataset представляет данные в табличном формате путем синтаксического анализа предоставленного файла или списка файлов. Это позволяет материализовать данные в кадр данных Pandas или Spark, чтобы работать с привычными библиотеками подготовки данных и обучения, не покидая записной книжки. Объект TabularDataset можно создать из файлов .csv, .tsv, .parquet, .jsonl и из результатов SQL-запроса.

Объекты в машинном обучении описываются набором признаков. В машинном обучении признак — это индивидуальное измеримое свойство или характеристика наблюдаемого явления. Выбор информативных, отличительных и независимых признаков является критическим шагом для эффективных алгоритмов в распознавании образов, классификации и регрессии. Признаки могут быть числовыми или нечисловыми. Матрица расстояний между объектами. Каждый объект описывается расстояниями до всех остальных объектов обучающей выборки. С этим типом входных данных работают немногие методы, в частности метод ближайших соседей, метод парзеновского окна, метод потенциальных функций.

Моделью машинного обучения называется файл, который обучен распознаванию определенных типов закономерностей. Вы обучаете модель на основе набора данных, предоставляя ей алгоритм, который она может использовать для анализа и обучения на основе этих данных. Хотя существует довольно много моделей представления объектов в машинном обучении, наибольшее распространение получила так называемая модель пространства признаков. В ней каждый объект представляется в виде набора пар [атрибут, значение], которые и называются признаками. Например, кандидат на должность девелопера может иметь такие атрибуты как возраст, наличие высшего образования, количество лет использования платформы, запрашиваемая зарплата и т.д.

Параметр модели - это переменная конфигурации, которая является внутренней по отношению к модели и значение которой можно оценить на основе данных. Они требуются моделью при прогнозировании. Эти значения определяют умение модели по вашей проблеме. Они оцениваются или извлекаются из данных. Они часто сохраняются как часть изученной модели. Параметры являются ключевыми для алгоритмов машинного обучения. Они являются частью модели, которая извлекается из исторических данных обучения.

1. **Структура и представление данных для машинного обучения.**

Под данными для обучения понимается исходный набор данных, передаваемый модели машинного обучения, на котором обучается модель. Структура данных — это способ организации информации для более эффективного использования. Со структурой можно работать: добавлять данные, извлекать их и обрабатывать, например изменять, анализировать, сортировать. Структура данных определяется как основной строительный блок компьютерного программирования, который помогает нам организовывать, управлять и хранить данные для эффективного поиска. Другими словами, структура данных представляет собой набор «значений» типов данных, которые хранятся и организованы таким образом, чтобы обеспечить эффективный доступ и модификацию. Структура данных представляет собой упорядоченную последовательность данных, и она сообщает компилятору, как программист использует такие данные, как целое число, строка, логическое значение и т. д. Существует два разных типа структур данных: линейные и нелинейные структуры данных.

Линейная структура данных — это особый тип структуры данных, который помогает организовывать данные и управлять ими в определенном порядке, когда элементы присоединяются друг к другу. В основном существует 4 типа линейной структуры данных:

* Массив — одна из самых основных и распространенных структур данных, используемых в машинном обучении. Используется в линейной алгебре для решения сложных задач.
* Стеки основаны на концепции LIFO (последним пришел — первым вышел) или FILO (первым пришел — последним ушел). Он используется для бинарной классификации в глубоком обучении.
* Связный список — это тип коллекции, имеющий несколько отдельно выделенных узлов. Другими словами, список — это тип набора элементов данных, состоящих из значения и указателя, указывающего на следующий узел в списке.
* Очередь определяется как «FIFO» (первым пришел, первым вышел). Полезно прогнозировать сценарий очередей в программах реального времени.

В нелинейных структурах данных элементы не располагаются в какой-либо последовательности. Все элементы расположены и связаны друг с другом иерархическим образом, где один элемент может быть связан с одним или несколькими элементами.

* Бинарное дерево: понятие бинарного дерева очень похоже на связанный список, но разница только в узлах и их указателях. В связанном списке каждый узел содержит значение данных с указателем, указывающим на следующий узел в списке, тогда как в двоичном дереве каждый узел имеет два указателя на последующие узлы вместо одного. Подобно связному списку, двоичное дерево также может быть преобразовано в массив на основе сортировки дерева.
* Графы. Структура данных графа также очень полезна в машинном обучении для предсказания ссылок. Графы представляют собой направленные или ненаправленные концепции с узлами и упорядоченными или неупорядоченными парами.
* Карты — это популярная структура данных в мире программирования, которая в основном полезна для минимизации алгоритмов времени выполнения и быстрого поиска данных. Он хранит данные в виде пары (ключ, значение), где ключ должен быть уникальным; однако значение может дублироваться.

**Представление данных.** Основная цель машинного обучения — строить модели путем интерпретации данных. Для этого очень важно подавать данные таким образом, чтобы их мог прочитать компьютер. Чтобы передать данные в модель scikit-learn, они должны быть представлены в виде таблицы или матрицы требуемой размерности.

Таблицы данных. Большинство таблиц, используемых в задачах машинного обучения, являются двумерными, то есть содержат строки и столбцы. Обычно каждая строка представляет экземпляр, тогда как каждый столбец представляет характеристику (признак) каждого наблюдения.

Особенности и целевые матрицы. Для многих проблем с данными одна из характеристик набора данных будет использоваться в качестве метки. Это означает, что из всех других признаков именно этот является целью, до которой модель должна обобщать данные.

Матрица признаков: Матрица признаков содержит данные из каждого экземпляра для всех признаков, кроме целевого. Он может быть создан с использованием массива NumPy или Pandas DataFrame.

Целевая матрица: В отличие от матрицы признаков, целевая матрица обычно является одномерной, поскольку она содержит только одну функцию для всех экземпляров.

Подобно матрице функций, целевая матрица обычно создается в виде массива NumPy или серии Pandas. Значения целевого массива могут быть дискретными или непрерывными.

1. **Инструментальные средства машинного обучения.**

**Языки программирования.**

**Python**. Это высокоуровневый язык программирования, который имеет множество различных способов применений, включая науку о данных и внутреннюю веб-разработку. Он является мощным инструментом для анализа данных, широко используется в технологии больших данных. Особый статус ему придает многочисленное сообщество разработчиков МО, которое в основном сосредоточено на быстрорастущем ИИ-направлении. Благодаря активному сообществу для Python появилось множество готовых библиотек МО. Этот язык — платформенно независимый, поэтому его можно адаптировать практически к любой операционной системе.

**R**. Этот язык программирования широко применяется в анализе данных и, как правило, является целевым для решения общих задач МО, таких как регрессия, классификация и формирование дерева решений. R помимо прочего пользуется популярностью среди статистиков. Как и Python, R обладает открытым исходным кодом и широко известен как язык, пакеты для работы с которым относительно легко установить, настроить и применять.

**C++**. Это самый старый среди наиболее распространенных на сегодняшний день языков программирования. Он был создан как научно-исследовательский проект, направленный на расширение возможностей языка Си. Обладая возможностями одновременно как низкоуровневого, так и высокоуровневого языка программирования, в контексте МО C++ обеспечивает более высокий уровень контроля и эффективности, чем другие языки программирования. Гибкость языка хорошо подходит для ресурсоемких приложений, и подмножество программ МО здесь — не исключение. Учитывая, что C++ — статически типизированный язык, он может выполнять задачи с относительно высокой скоростью.

**Java**. Он изначально замышлялся как высокоуровневый и объектно-ориентированный язык программирования, который во многом напоминает по структуре C++. Обладая огромной популярностью, Java может похвастаться широким спектром алгоритмов, которые очень полезны для сообщества разработчиков софта МО. Во многом Java считается одним из самых безопасных языков программирования благодаря использованию байт-кода и песочниц.

**Инструменты для сбора, обработки и визуализации данных.** Здесь собирают данные с различных сайтов и создают датасет, который потом используют для обучения алгоритма. Сбор данных с сайтов ещё называют веб-скрейпингом. После того, как собрали данные, их нужно обработать, чтобы избавиться от ошибок, шума и несогласованностей. Это очень важно, так как от корректности данных будет зависеть точность результатов алгоритма. Визуализация поможет определить линейность структуры данных, существенные признаки и аномалии.

pandas: библиотека для обработки и анализа данных. Это группировки, сортировки, извлечения и трансформации. Для работы с файлами CSV, JSON и TSV pandas превращает их в структуру данных DataFrame со строками и столбцами. Выглядит, как обычная таблица в Excel, и работать с ней легче, чем с for-циклами для прохода по элементам списков и словарей.

Matplotlib: библиотека для построения 2D-графиков. Matplotlib в связке с библиотеками seaborn и ggplot позволяет строить разнообразные графики: гистограммы, диаграммы рассеяния, круговые и полярные диаграммы, и много других.

**Интерактивные среды разработки.** Эти инструменты часто используются для Data Science и машинного обучения. Веб-среда позволяет разработчикам на лету тестировать небольшие части кода, проверять функциональность и разные гипотезы. Тем не менее, при желании в ней можно поместить и целый проект.

Jupyter Notebook: интерактивное моделирование. Простая в использовании бесплатная интерактивная веб-оболочка.

Kaggle: сообщество Data Science. Kaggle также предоставляет интерактивную среду разработки. Здесь можно найти готовые датасеты, модели и даже программный код для решения разных задач.

**Фреймворки и библиотеки для общего машинного обучения.** Обучение модели делится на две большие категории: с учителем и без. В первом случае мы маркируем датасет, объясняя алгоритму машинного обучения, где правильный ответ, а где — нет. Так данные можно представить таблицей соответствий «элемент-категория». Во втором случае алгоритм сам вынужден искать признаки и закономерности, так как в датасете мы даём данные без уточняющей информации. Датасет представлен сплошным потоком данных нужного типа: текста, картинок и др.

NumPy: готовые вычислительные алгоритмы и линейная алгебра для машинного обучения. Данные в машинном обучении представлены числовыми массивами. Даже если мы работаем с картинками или естественной речью, они должны быть преобразованы в числовые массивы.

NLTK: разбираем естественный язык на части. Один из ведущих инструментов для обработки естественного языка. По аналогии с тем, как NumPy упрощает линейную алгебру, NLTK упрощает парсинг текста, анализ тональности, структуры предложений и всё, что с этим связано.

1. **Задача регрессии: постановка, математическая формализация.**

В задачах регрессии мы принимаем входные переменные и пытаемся подогнать выход на непрерывную ожидаемую функцию результата. Линейная регрессия с одной переменной также известна как «парная линейная регрессия». Одномерная линейная регрессия используется, когда вы хотите предсказать одно выходное значение *y*, зависящее от одного входного значения *x*. Мы проводим обучение с учителем, это означает, что мы уже имеем представление о том, какие значения выходной переменной соответствуют некоторым значениям входной переменной.

Задача регрессии (контролируемого машинного обучения), которая прогнозирует значение метки по набору связанных компонентов. Метка здесь может принимать любое значение, а не просто выбирается из конечного набора значений, как в задачах классификации. Алгоритмы регрессии моделируют зависимость меток от связанных компонентов, чтобы определить закономерности изменения меток при разных значениях компонентов. На вход алгоритма регрессии подается набор примеров с метками известных значений. Результатом работы алгоритма регрессии является функция, которая умеет прогнозировать значения метки для любого нового набора входных компонентов. Вот несколько примеров для сценария регрессии:

* прогнозирование цен на дома по таким атрибутам, как количество комнат, расположение и размер;
* прогнозирование будущей цены акций на основе исторических данных и текущих тенденций рынка;
* прогнозирование продаж товара в зависимости от рекламного бюджета.

**Постановка задачи регрессии:**

Регрессия — это задача машинного обучения с учителем, которая заключается в предсказании некоторой непрерывной величины. Для использования регрессионных моделей нужно, чтобы в датасете были характеристики объектов и “правильные” значения целевой переменной. Задача регрессии основывается на предположении, что значение целевой переменной зависит от значения признаков. Регрессионная модель принимает набор значений и выдает предсказание значения целевой переменной. В качестве регрессионных моделей часто берут аналитические функции, например, линейную.

### Общая постановка задачи:

Пусть теперь x1,…,xn∈Rd зафиксированы, а y1,…,yn являются случайными величинами. Мы предполагаем, что истинная связь между yi и xi является линейной, плюс некоторая случайная ошибка. А именно, существует такой вектор w∈Rd (вектор весов), что yi=⟨xi,w⟩+εi,

где ⟨xi,w⟩ — стандартное скалярное произведение  а все εi независимы в совокупности, имеют нулевое матожидание E[εi]=0 для всех i=1,…,n и одинаковую конечную дисперсию 

В этом случае процесс обучения линейной модели состоит в нахождении вектора  по имеющимся данным. Если бы модель была на 100% верна и ошибки отсутствовали (εi=0 для всех i=1,…,d), то уравнение было бы линейным уравнением на w, из которого можно было бы найти последний. В реальности же приходится использовать другие методы — в частности, метод максимального правдоподобия.

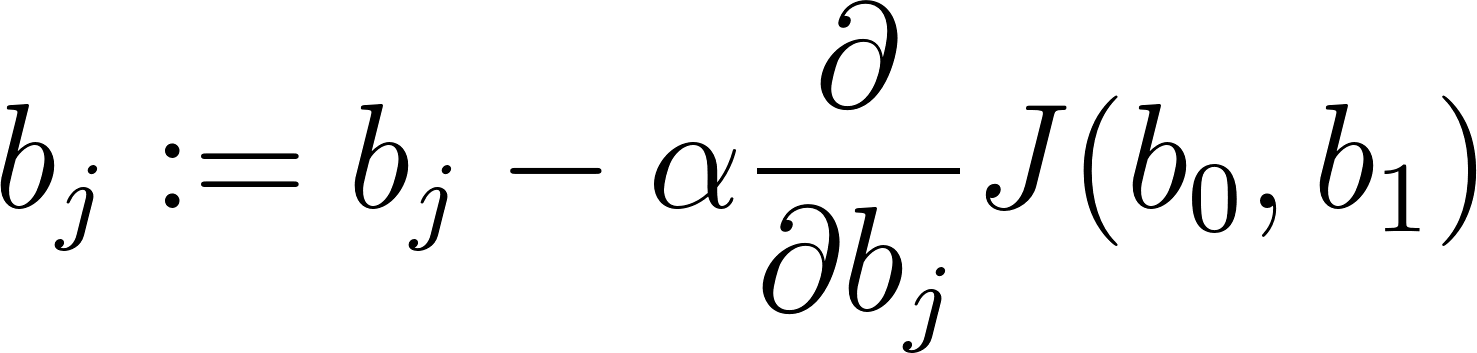
1. **Метод градиентного спуска для парной линейной регрессии.**

В статистике линейная регрессия — это линейный подход к моделированию взаимосвязи между зависимой переменной и одной или несколькими независимыми переменными.

Наиболее простой и понятный, вместе с тем часто используемый метод математического программирования для решения задач такого рода — метод градиентного спуска *(gradient descent).*Это итерационный алгоритм, на каждом шаге которого вектор весов *w* меняется в направлении наибольшего убывания целевого функционала, т.е. в направлении антиградиента:Изображение выглядит как текст

Автоматически созданное описание

Алгоритм градиентного спуска (повторять до сходимости):

[](about:blank)где j=0,1 - представляет собой индекс номера признака.

Алгоритм градиентного спуска для парной линейной регрессии (повторять до сходимости):Изображение выглядит как текст

Автоматически созданное описание

Метод градиентного спуска нужен, чтобы найти минимум функции, если мы не можем ее вычислить аналитически. Это численный итеративный алгоритм локальной оптимизации. Для запуска градиентного спуска нужно знать частную производную функции ошибки. Для начала мы берем произвольные значения параметров, затем обновляем их по данной формуле. Доказано, что этот метод сходится к локальному минимуму. Если функция ошибки достаточно сложная, то разные начальные точки дадут разный результат. Метод градиентного спуска имеет свой параметр - скорость обучения. Обычно его подстраивают автоматически. Метод градиентного спуска повторяют много раз до тех пор, пока функция ошибки не перестанет значимо изменяться.

Выбор параметра скорости обучения *(learning rate)* выполняется экспериментальным путем, зависит от природы данных, для которых подбирается значение этого параметра. Если минимизируемая функция достаточно гладкая, то параметр можно выбрать достаточно большим, вследствие чего время работы алгоритма обучения будет малым. Однако если целевая функция имеет сильную кривизну, то в случае больших значений скорости обучения возможна ситуация, при которой вектор весов *w* будет «перескакивать» свое оптимальное значение. Для реализации алгоритма обучения возможны два подхода:

1) пакетный *{batch) —* на каждой итерации алгоритма обучающая выборка просматривается целиком и после этого рассчитывается новое значение вектора *w;*

2) стохастический (*stochastic*) — на каждой итерации алгоритма случайным образом выбирается только один объект из обучающей выборки.

Пакетный метод реализовать просто — необходимо выполнять обновление весов при помощи усредненного значения антиградиента на всех примерах из обучающей выборки до тех пор, пока не стабилизируется оценка функционала *Q.*

Преимущество метода стохастического градиента в сравнении с пакетным заключается также в том, что данный алгоритм позволяет реализовать так называемое онлайн-обучение, и если в случае линейной регрессии эта особенность не является важной, то для решения задач классификации она становится интересной чертой, позволяющей алгоритму дообучаться в реальном времени.

Градиентный спуск — это метод, который мы позаимствовали из оптимизации. Очень простой, но мощный алгоритм, который можно использовать для поиска минимума функции. Этот метод найдет глобальный минимум, если функция выпуклая. В противном случае мы можем быть уверены только в том, что достигнем локальный минимум.

Градиентный спуск является одним из самых простых и широко используемых алгоритмов в машинном обучении главным образом потому, что его можно применять к любой функции для его оптимизации. Обучение это закладывает основу для овладения машинным обучением.

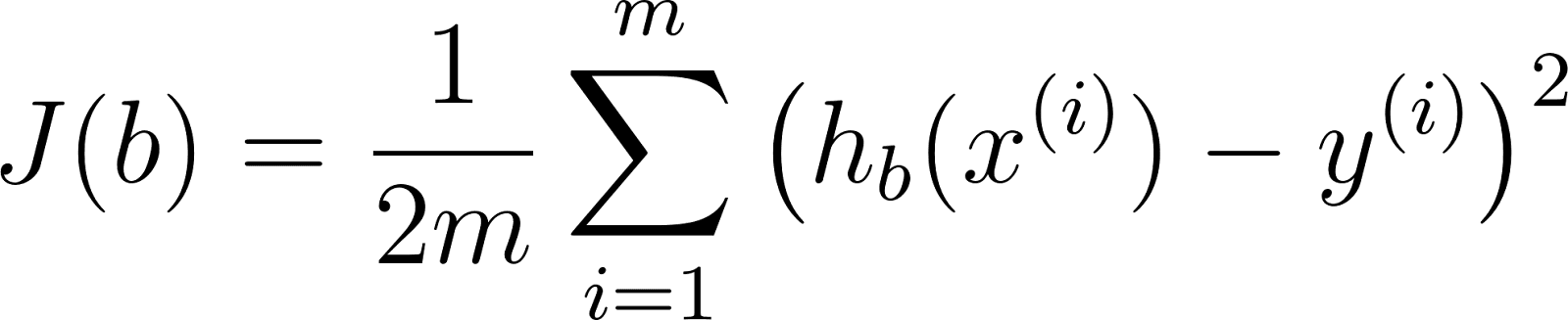
1. **Понятие функции ошибки: требования, использование, примеры.**

Для того, чтобы подобрать наилучшую модель, нам нужно средство измерения “точности” модели, некоторая функция, которая показывает, насколько модель хорошо или плохо соответствует имеющимся данным. Такая функция называется функцией ошибки (cost function). Она измеряет отклонения теоретических значений (то есть тех, которые предсказывает модель) от эмпирических (то есть тех, которые есть в данных). Чем выше значение функции ошибки, тем хуже модель соответствует имеющимся данным, хуже описывает их. Если модель полностью соответствует данным, то значение функции ошибки будет нулевым.

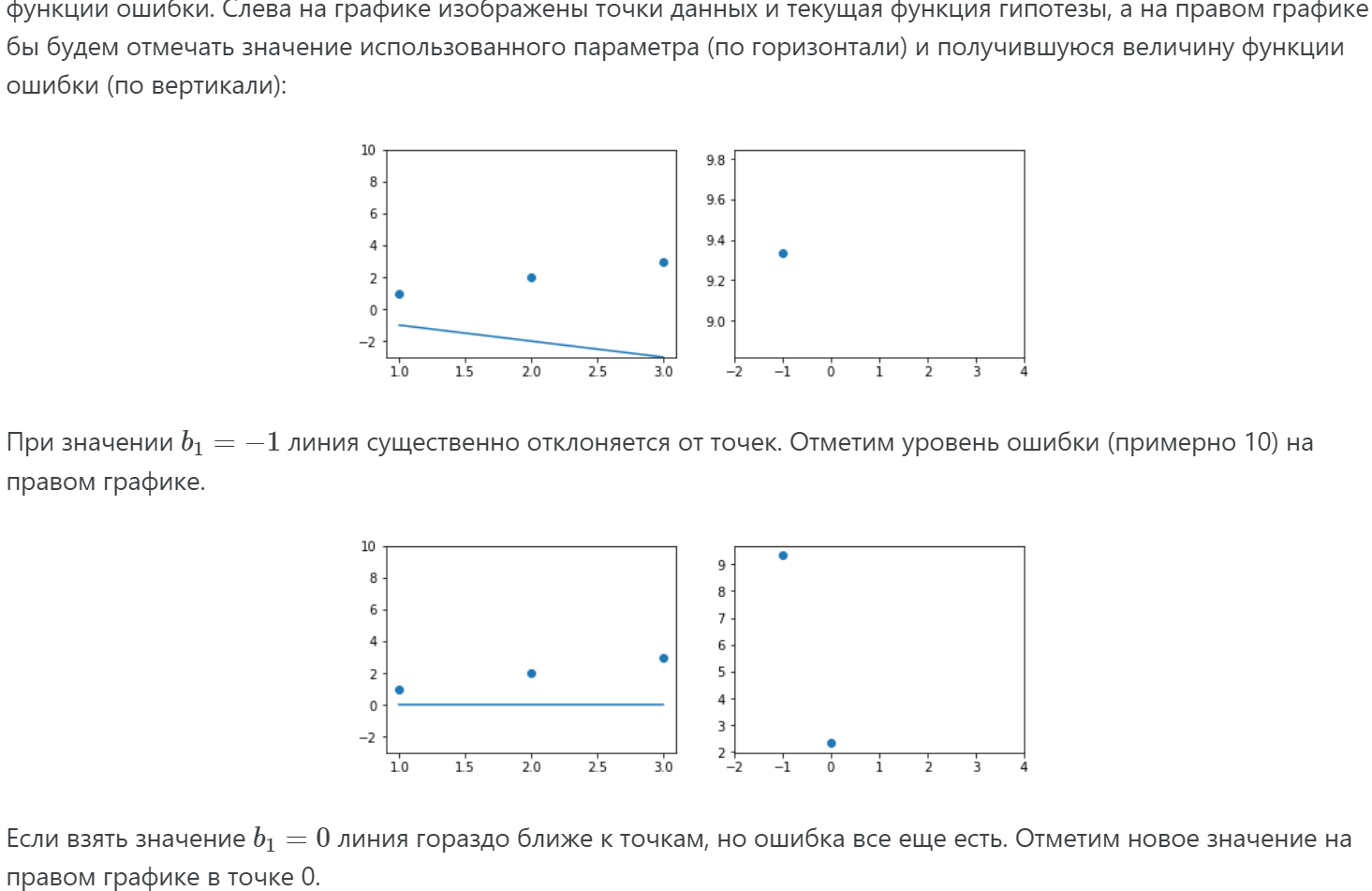
Мы можем измерить точность нашей функции гипотезы, используя функцию ошибки. Для этого требуется средняя всех результатов вычисления гипотезы с входами x по сравнению с фактическим выходом y.

В задачах регрессии в качестве функции ошибки чаще всего берут среднеквадратичное отклонение теоретических значений от эмпирических. То есть сумму квадратов отклонений, деленную на удвоенное количество измерений. По сути своей это половина среднего квадрата разницы между прогнозируемым и фактическим значением выходной переменной. Эту функцию называют «функцией квадрата ошибки» или «среднеквадратичной ошибкой» (mean squared error, MSE).

Теперь мы можем конкретно измерить точность нашей предсказывающей функции по сравнению с правильными результатами, которые имеем, чтобы могли предсказать новые, которых у нас нет. Если мы попытаемся представить это наглядно, наш набор данных обучения будет разбросан по плоскости x-y. Мы пытаемся подобрать прямую линию, которая проходит через этот разбросанный набор данных. Наша цель - получить наилучшую возможную линию. Лучшая линия будет такой, чтобы средние квадраты вертикальных расстояний рассеянных точек от линии были наименьшими. В лучшем случае линия должна проходить через все точки нашего набора данных обучения. В таком случае значение J будет равно 0. Для множественной регрессии функция ошибки от вектора параметров b выглядит следующим образом:

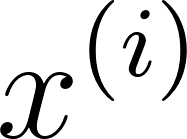
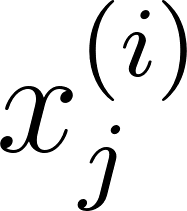
[](about:blank)

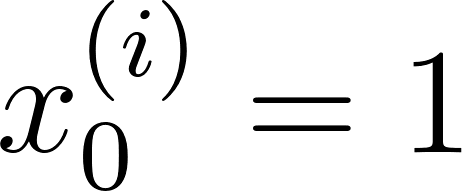
Проследим формирование функции ошибки на еще более простом примере. Возьмем упрощенную форму линейной модели - прямую пропорциональность. Она выражается формулой: y=hb(x)=b1x. Эта модель поможет нам, так как у нее всего один параметр. И функцию ошибки можно будет изобразить на плоскости. Возьмем фиксированный набор точек и попробуем несколько значений параметра для вычисления функции ошибки.



1. **Множественная и нелинейная регрессии.**

Линейная регрессия с несколькими переменными также известна как «множественная линейная регрессия». Модель множественной линейной регрессии — это практичная статистическая модель для оценки связей между непрерывной зависимой переменной и переменными-предикторами. Предикторы могут быть непрерывными, категориальными или производными полями, так что поддерживаются также нелинейные взаимосвязи. 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. Введем обозначения для уравнений, где мы можем иметь любое количество входных переменных:

[](about:blank) - вектор-столбец всех значений признаков i-го обучающего примера; [](about:blank) - значение j-го признака i-го обучающего примера; *m* - количество примеров в обучающей выборке; *n* - количество признаков; *X* - матрица признаков; *b* - вектор параметров регрессии.

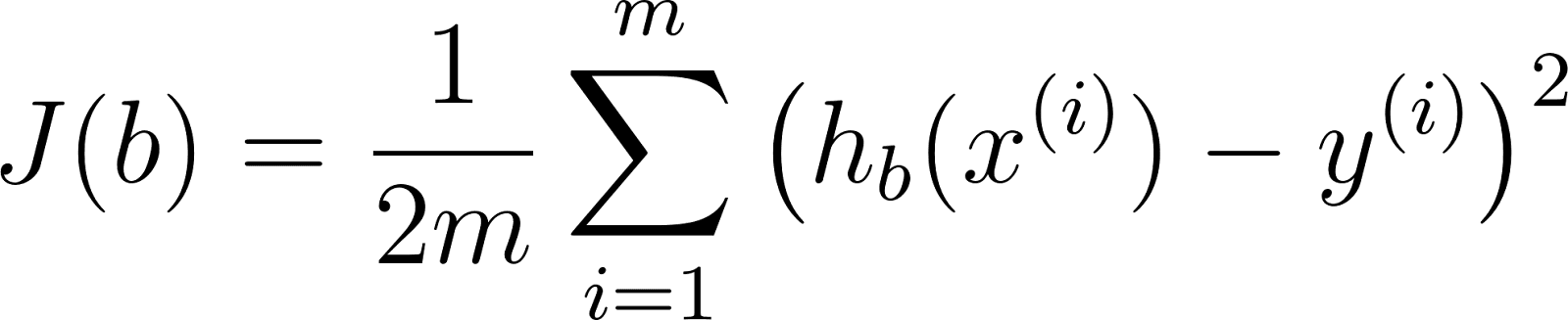
Заметим, что в будущем для удобства примем, что [](about:blank) для всех i. Другими словами, мы для удобства введем некий суррогатный признак, для всех наблюдений равный единице. это сильно упростит математические выкладки, особенно в матричной форме.

Теперь определим множественную форму функции гипотезы следующим образом, используя несколько признаков:

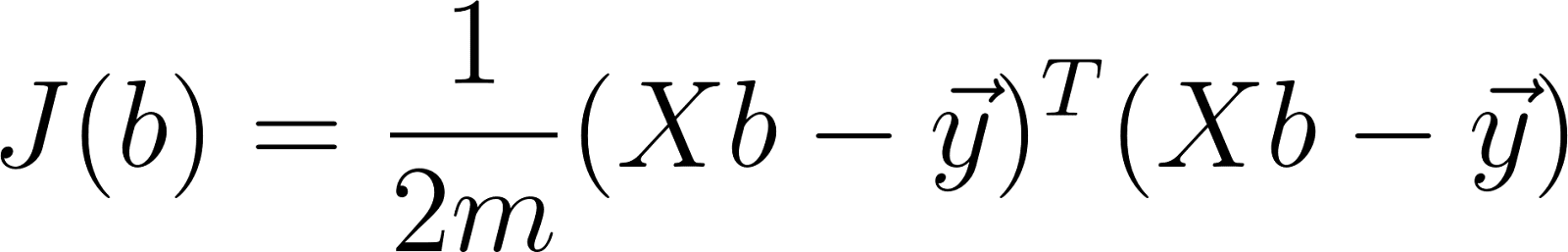
[](about:blank)

Используя определение матричного умножения, наша многопараметрическая функция гипотезы может быть кратко представлена в виде: *h(x) = B X*.

Для множественной регрессии функция ошибки от вектора параметров b выглядит следующим образом:

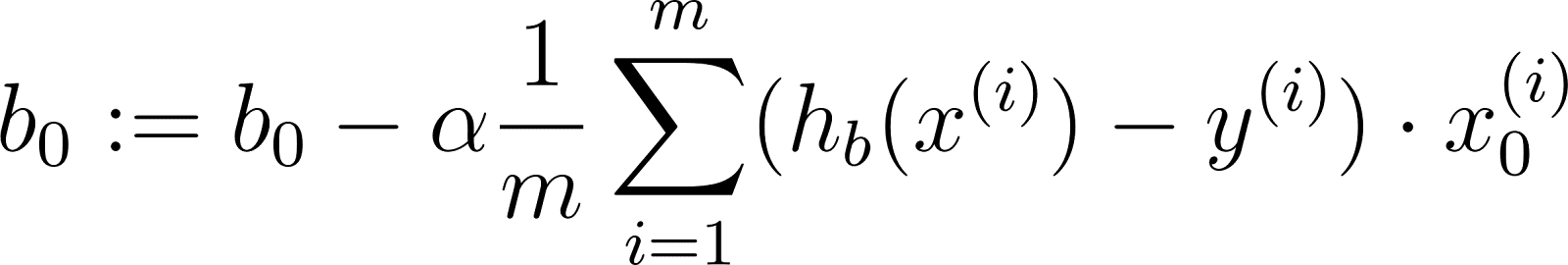
[](about:blank)

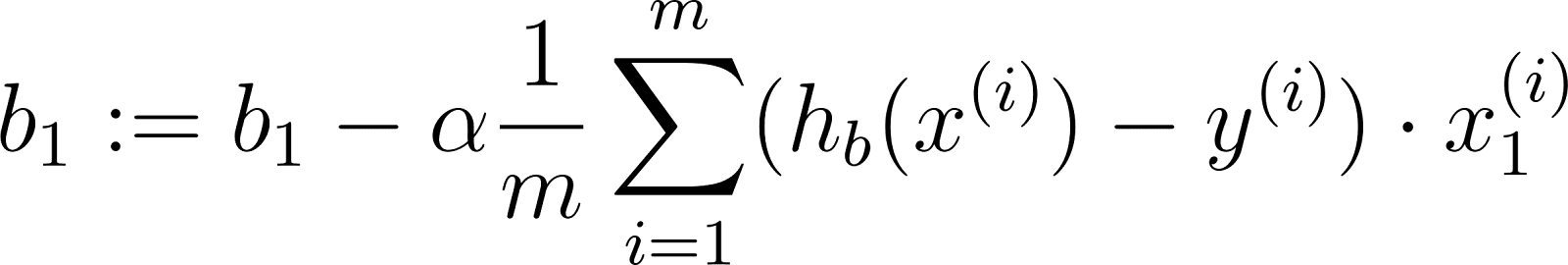
Или в матричной форме:

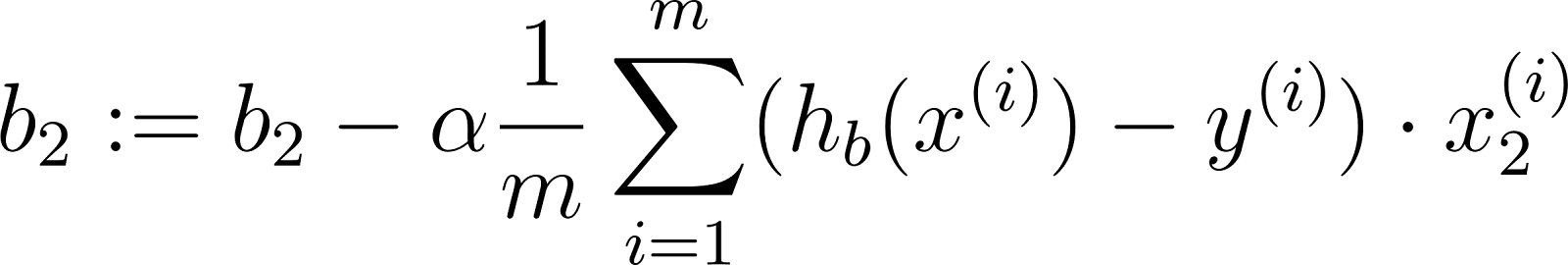
[](about:blank)

Метод градиентного спуска для множественной регрессии определяется следующими уравнениями:

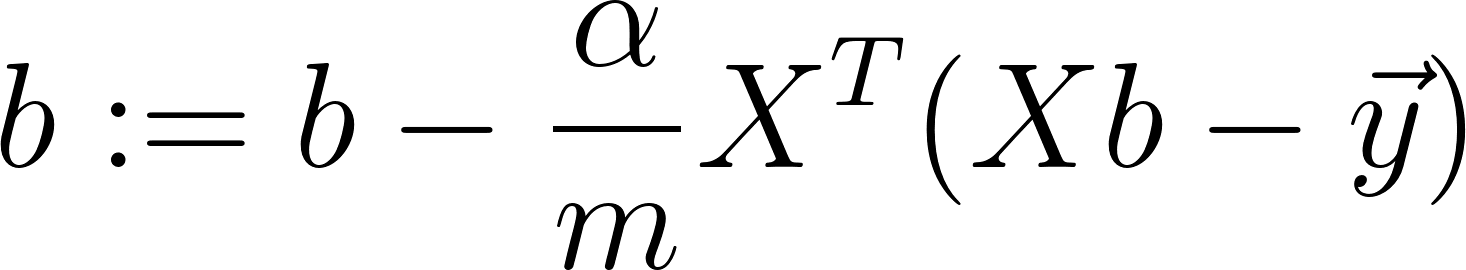
повторять до сходимости:

[](about:blank)

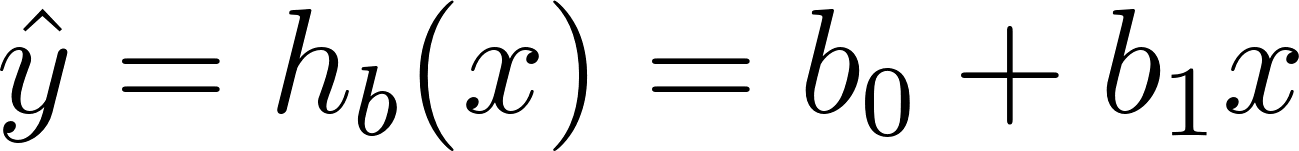
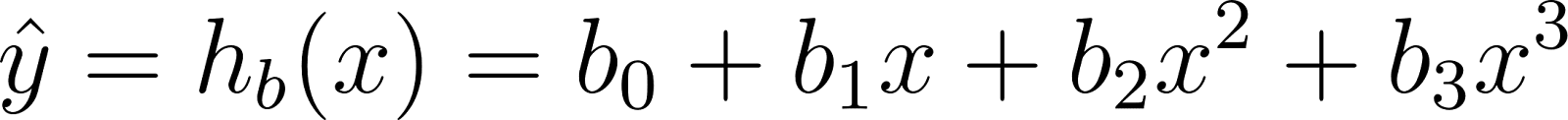
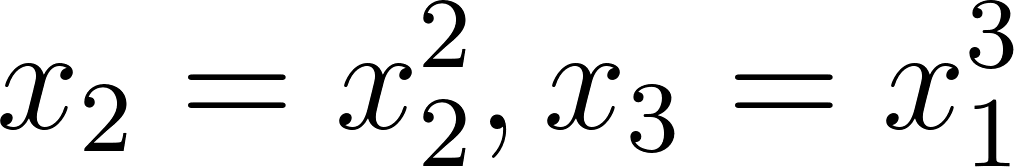
[](about:blank)

[](about:blank)

Или в матричной форме:

[](about:blank)

Наша функция гипотезы не обязательно должна быть линейной (прямой), если это не соответствует данным.

Мы можем изменить поведение или кривую нашей функции гипотезы, сделав ее квадратичной, кубической или квадратной корневой функцией (или любой другой формой). Например, если наша функция гипотезы [](about:blank), то мы можем добавить еще один признак, основанный на x1, получив квадратичную функцию [](about:blank) или кубическую функцию [](about:blank). В кубической функции мы, по сути, ввели два новых признака: [](about:blank).

1. **Нормализация признаков в задачах регрессии.**

Мы можем ускорить сходимость метода градиентного спуска, получив каждое из наших входных значений примерно в том же диапазоне. Это связано с тем, что b будет быстро сходиться на малых диапазонах и медленно на больших диапазонах, и поэтому будет колебаться неэффективно до оптимального, если переменные очень неравномерны.

Способ предотвратить это - изменить диапазоны наших входных переменных, чтобы они были примерно одинаковыми. В идеале  -1 ≤ x ≤ 1 или же -0,5 ≤ x ≤ 0,5. Это не точные требования, мы только пытаемся ускорить процесс. Цель состоит в том, чтобы получить все входные переменные в примерно один из этих диапазонов, дать или взять несколько.

Два метода для этого - масштабирование признаков (Минимаксная нормализация) и нормализация по среднему (стандартизация или z-оценки).

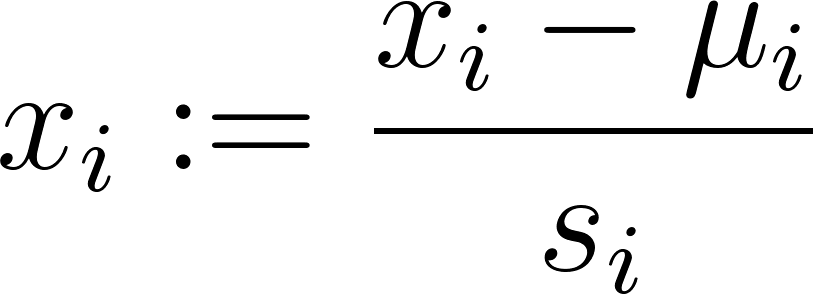
Масштабирование признаков заключается в делении входных значений на размах выборки (то есть максимальное значение минус минимальное значение) входной переменной, в результате чего новый диапазон составляет всего 1. После преобразования все значения будут лежать в диапазоне x∈[0;1].

Минимаксная нормализация: x′=(x−xmin)/(xmax−xmin)

Нормализация по среднему включает в себя вычитание среднего значения входной переменной из значений для этой входной переменной, в результате чего новое среднее значение для этой переменной равно нулю. В таком случае данный признак приводится к стандартному распределению, то есть такому, у которого среднее 0, а дисперсия - 1.

стандартизация или z-оценки: x′=(x−M[x])/σx

Чтобы реализовать оба этих метода, отрегулируйте свои входные значения, как показано в этой формуле:

[](about:blank) Где [](about:blank) - среднее значение признака i, а [](about:blank) - стандартное отклонение этого признака.

1. **Задача классификации: постановка, математическая формализация.**

Формально, задача классификации состоит в предсказании какого-то дискретного значения. Задача классификации сводится к отнесению конкретного объекта к одному из заранее известных классов. Вот эта “метка” - название класса - и выступает в роли целевой переменной. И, совершенно естественно, что классов может быть только конечное количество, поэтому целевая переменная - дискретная. Важно, что классы должны быть заранее известны. Мы должны знать сколько их всего, и к какому классу относится каждый объект обучающей выборки. Если идет речь о двух классах, то это так называемая бинарная классификации. Все эти задачи можно сформулировать как определение наличия или отсутствия какого-либо признака у объекта. Иногда случается такое, что классов больше двух. Тогда мы говорим о задаче множественной классификации. Кроме бинарной и множественной классификации еще выделяют одноклассовую и мультиклассовую. Одноклассовая — это когда один объект может принадлежать только одному классу. В мультиклассовой классификации каждый объект может принадлежать сразу нескольким классам. Отдельно выделяют нечеткую классификацию — это когда объект может принадлежать некоторым классам с разной принадлежностью или вероятностью. Примеры задач классификации - распознавание объектов, генерация текстов, подбор тематики текстов, идентификация объектов на изображениях, распознавание речи, машинный перевод и так далее. Итак, рассмотрим математическую формализацию задачи классификации. В такой задаче, так же, как и в регрессии, на вход модели подается вектор признаков . Как и ранее, введем искусственный признак . Он нужен для удобства представления многих моделей классификации. Можете представлять его как еще один столбец в датасете, в котором во всех строчках стоят единицы.

Функция гипотезы в таком случае будет иметь точно такой же вид: Изображение выглядит как текст

Автоматически созданное описание. Существенное отличие в том, что целевая переменная *y* будет принимать одно из конечного множества значений: , где *k* — это количество классов.

Изображение выглядит как текст

Автоматически созданное описание

1. **Метод градиентного спуска для задач классификации.**

Наша функция стоимости (ошибки) для логистической регрессии выглядит так:

[](about:blank), где

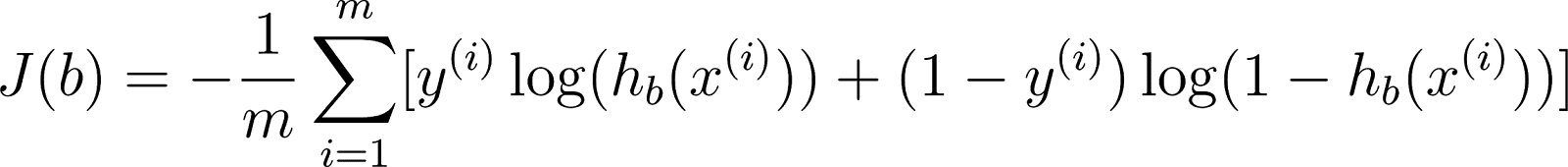
[](about:blank) [](about:blank)

Мы можем сжать два условных случая функции стоимости (ошибки) в один случай (в одно выражение), используя тот факт, что истинные значения y могут принимать только значения 0 или 1:

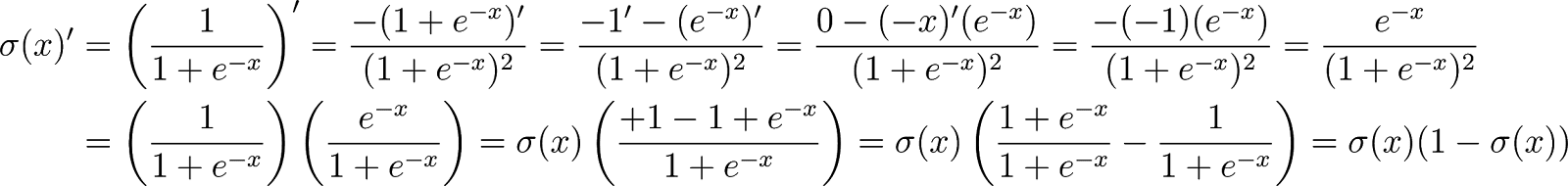
[](about:blank)

Стоит обратить внимание, что когда y равно 1, то второй член будет равен нулю и не повлияет на результат. Если y равно 0, то первый член будет равен нулю и не повлияет на результат.

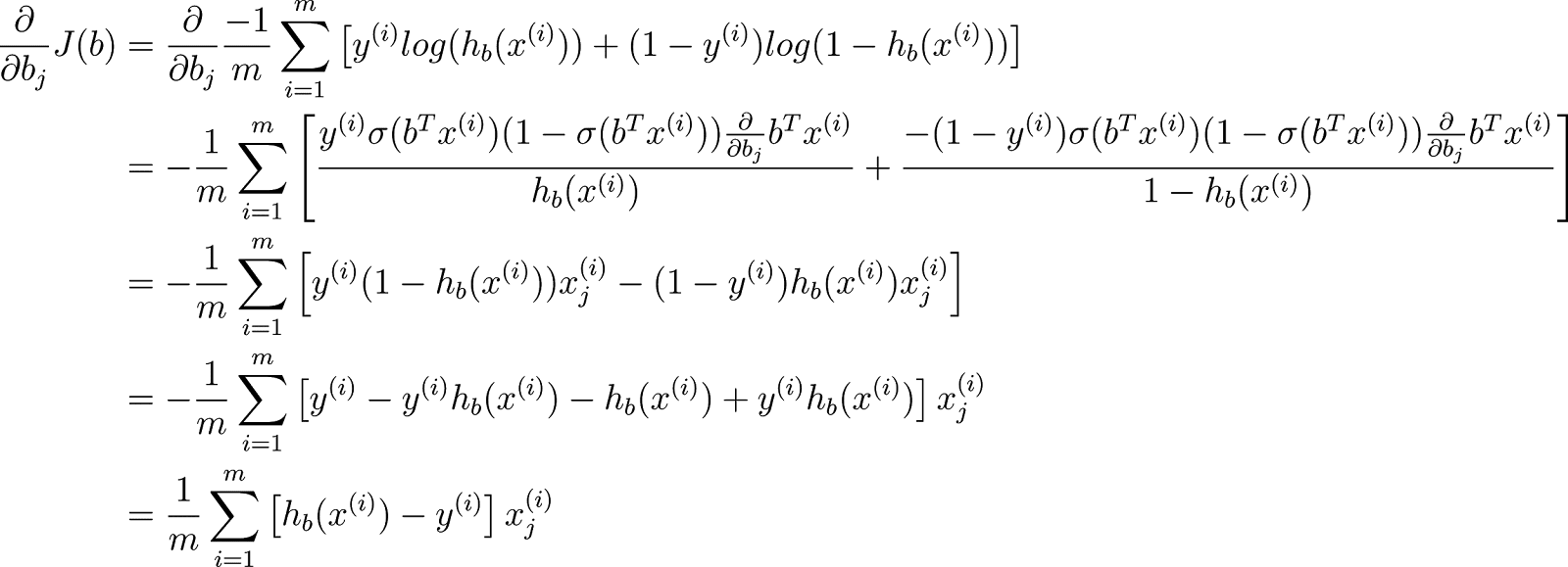
Мы можем полностью выписать всю нашу функцию затрат (ошибки) следующим образом:

[](about:blank)

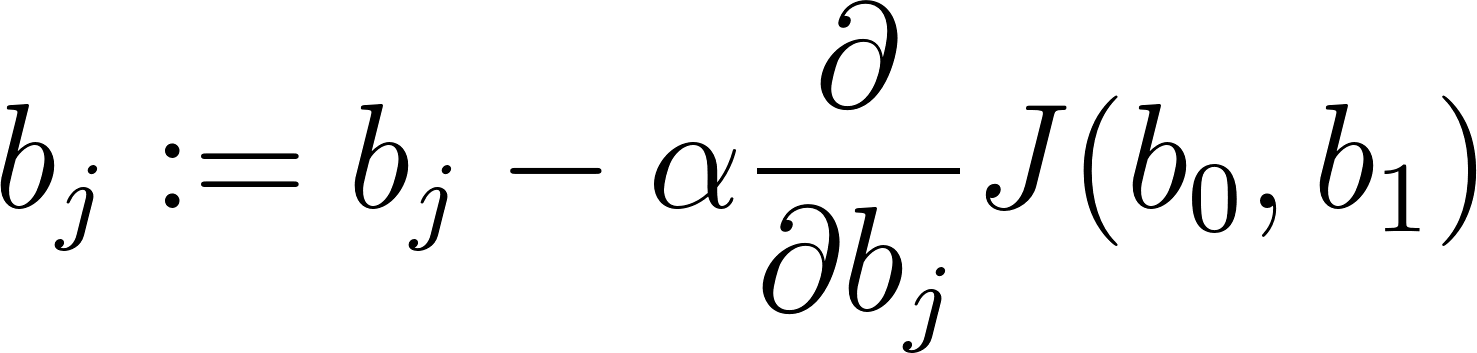
Частная производная от J (θ). Сначала вычислим производную от сигмоидной функции (она будет полезна при нахождении частной производной от J(θ)):

[](about:blank)

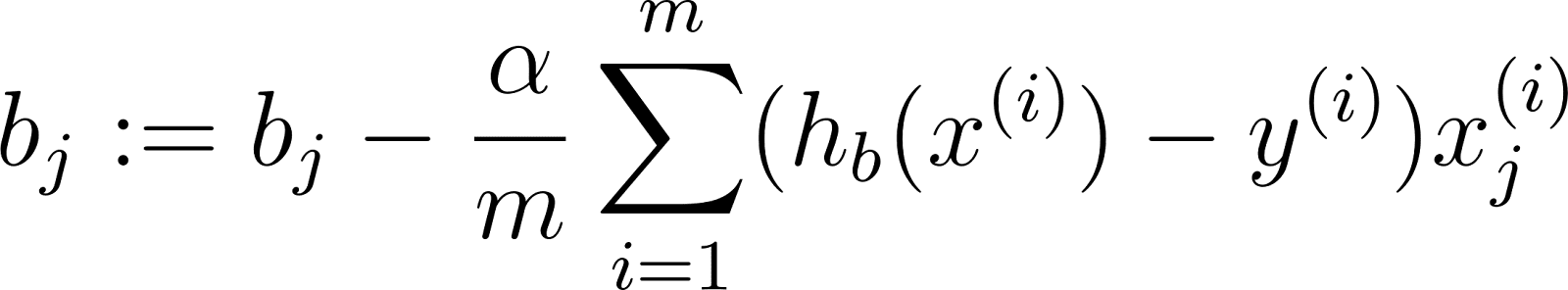
Теперь мы готовы найти полученную частную производную:

[](about:blank)

Помним, что общая форма градиентного спуска:

[](about:blank)

Мы можем разработать производную часть, используя вышеприведенное дифференцирование, чтобы получить:

[](about:blank)

Стоит обратить внимание, что этот алгоритм идентичен тому, который мы использовали в линейной регрессии. Мы все равно должны одновременно обновлять все значения в b.

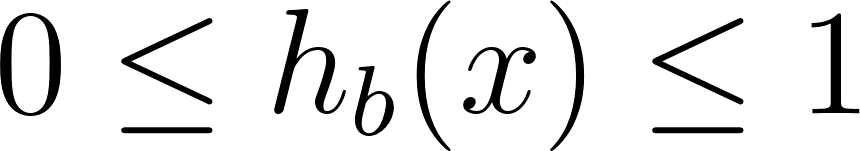
Сопряженный градиентный спуск, BFGS и L-BFGS — это более сложные и быстрые способы оптимизации b, которые можно использовать вместо градиентного спуска. Лучше не писать программные реализации этих более сложных алгоритмов самостоятельно, но вместо этого использовать библиотеки, поскольку они уже протестированы и сильно оптимизированы. Библиотека scikit-learn реализует многие эти алгоритмы.

1. **Логистическая регрессия в задачах классификации.**

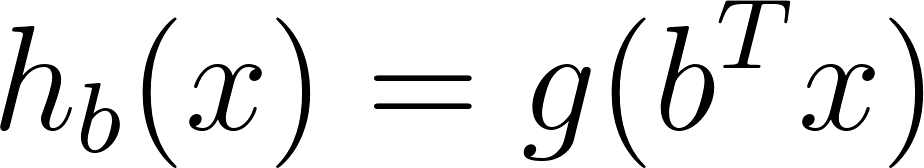
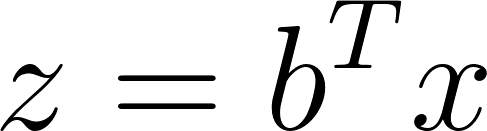
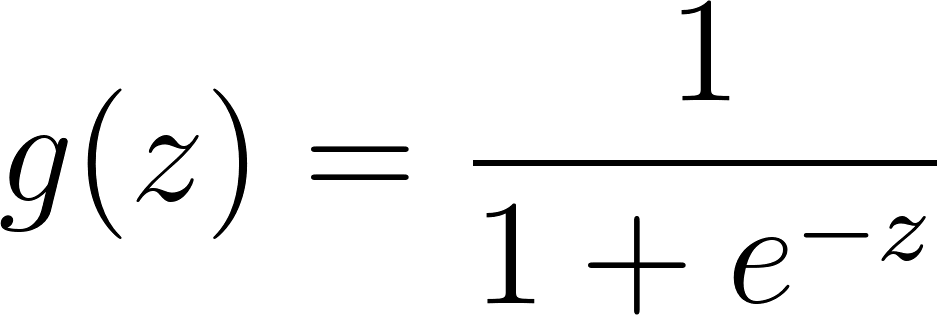
Одним из самых простых и распространенных алгоритмов классификации является логистическая регрессия. Метод назван таким образом по историческим причинам и на самом деле является подходом к проблемам классификации, а не регрессионным проблемам. Вместо того, чтобы наш выходной вектор *y* был непрерывным диапазоном значений, он будет равен только 0 или 1. y∈ {0,1}

Где 0 обычно принимается как «отрицательный класс» и 1 как «положительный класс», но вы можете назначить ему какое-либо представление. На данный момент мы занимаемся только двумя классами, называемыми проблемой двоичной или бинарной классификации.

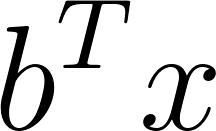
Один из методов - использовать линейную регрессию и отображать все предсказания, превышающие 0,5, как 1 и все меньше 0,5 в качестве 0. Этот метод не работает, потому что классификация на самом деле не является линейной функцией.

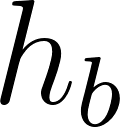
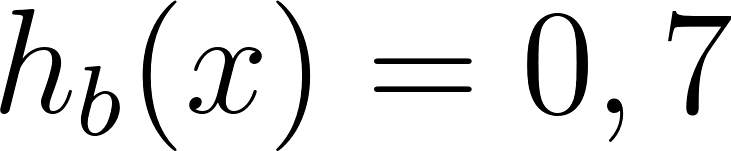
Наша гипотеза должна удовлетворять: [](about:blank)

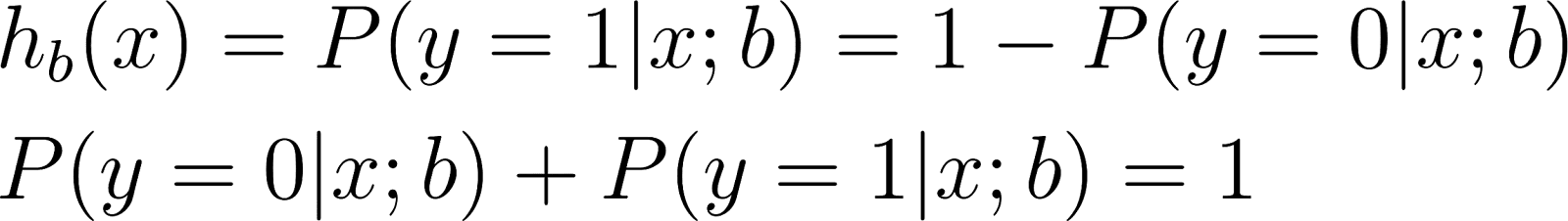
В этой новой форме задачи используется «сигмоидальная функция», также называемая «логистическая функция»:

[](about:blank) [ ](about:blank)

Функция g(z) отображает любое вещественное число на интервал (0, 1), что делает его полезным для преобразования произвольнозначной функции в функцию, более подходящую для классификации.

Начнем с нашей старой гипотезы (линейной регрессии), за исключением того, что мы хотим ограничить диапазон до 0 и 1. Это достигается путем подключения [](about:blank) в логистическую функцию.

[](about:blank) даст нам вероятность того, что наш результат равен 1. Например,  дает нам вероятность 70%, что наш выход равен 1.

[](about:blank)

Наша вероятность того, что наше предсказание равна 0, является просто дополнением нашей вероятности того, что она равна 1 (например, если вероятность того, что она равна 1, равна 70%, то вероятность того, что она равна 0, равна 30%).

Для четкой классификации обычно выбирают некоторое пороговое значение, обычно - 0,5.

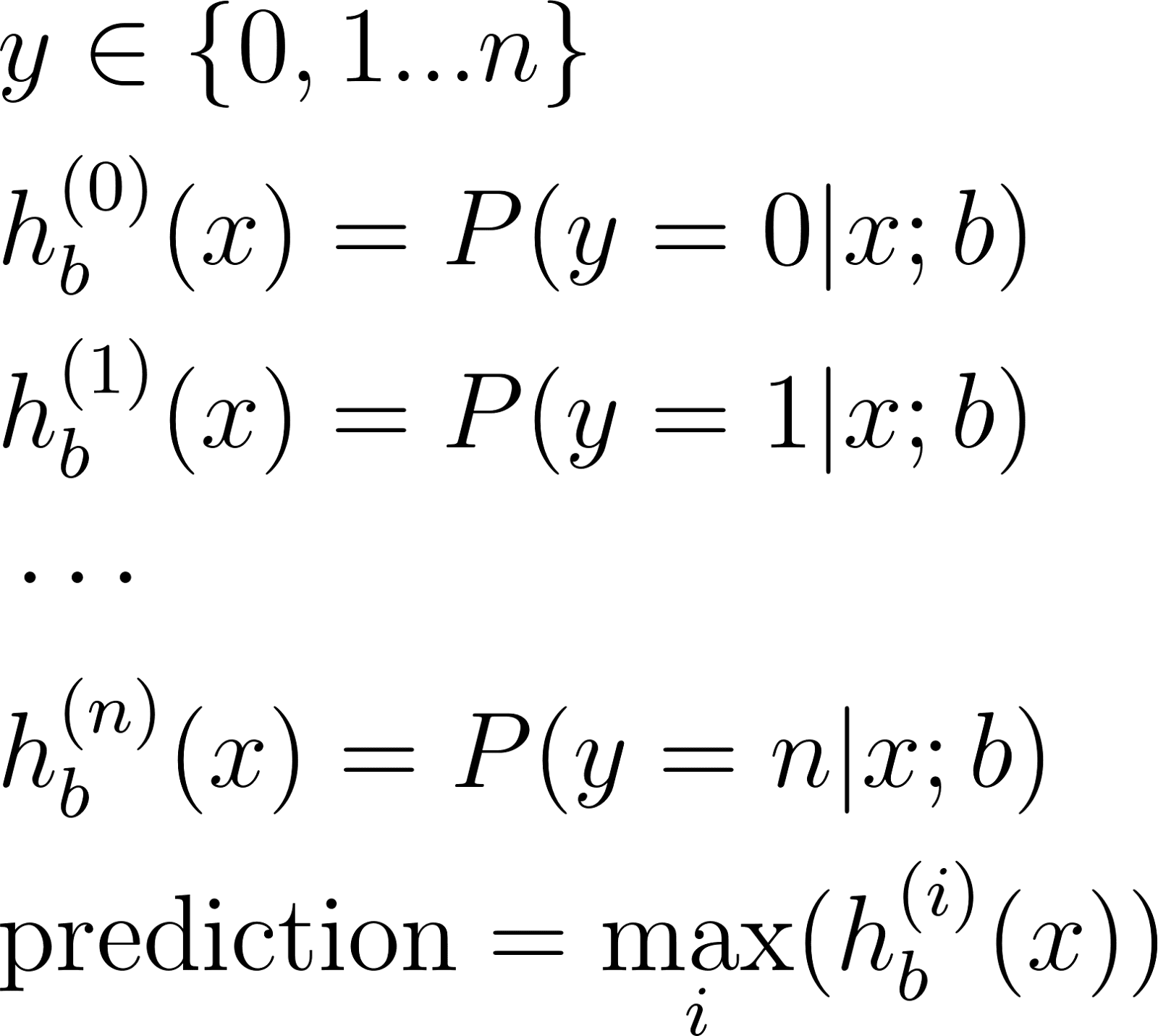
1. **Множественная и многоклассовая классификация. Алгоритм “один против всех”.**

Рассмотрим классификацию данных более чем в двух категориях. Вместо y = {0,1} мы расширим наше определение так, чтобы y = {0,1 ... n}.

Пример множественной классификации набора данных, содержащих два признака и три класса можно увидеть на рисунке:

Алгоритм классификации в данном случае очень прост. Мы берем последовательно каждый имеющийся класс в данных, делаем его “положительным”, а все остальные - “отрицательным”, и обучаем модель, которая стремится отделить данный класс от остальных.

В этом случае мы делим нашу задачу на n + 1 (+1, потому что индекс начинается с 0) двоичных задач классификации; в каждом из них мы прогнозируем вероятность того, что 'y' является членом одного из наших классов.

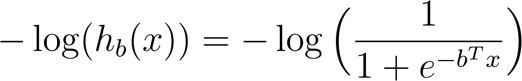
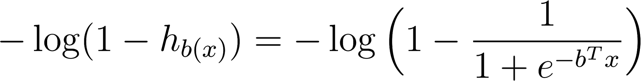
[](about:blank)

Мы в основном выбираем один класс, а затем объединяем всех остальных в один второй класс. Мы делаем это неоднократно, применяя двоичную логистическую регрессию к каждому случаю, а затем используем гипотезу, которая вернула наивысшее значение в качестве нашего прогноза.

Другими словами, построив несколько моделей бинарной классификации мы можем использовать их, чтобы получить оценки вероятности принадлежности любого объекта ко всем имеющимся классам. После этого для окончательной классификации выбирается тот класс, чья модель дала наивысший результат.

Данный метод называется “один против всех” (one vs all или one vs rest). Он нужен только для моделей, которые способны решать только бинарную классификацию, чтобы “приспособить” их для задач, где классов больше двух. Надо отметить, что во всех современных программных инструментах для машинного обучения, он уже реализован и встроен в существующие методы классификации, так что разработчику не придется программировать его специально.

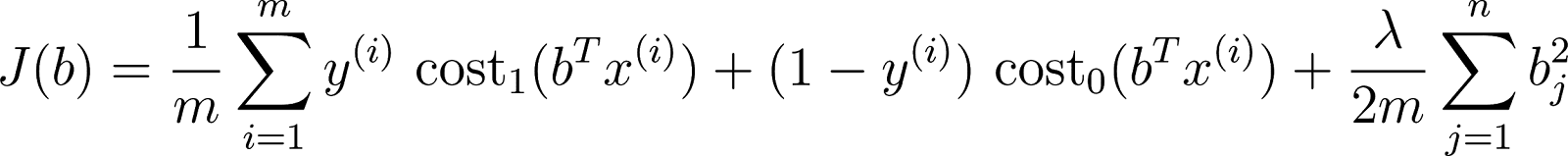
1. **Метод опорных векторов в задачах классификации.**

Метод опорных векторов (support vector machines, SVM) является еще одним типом алгоритма машинного обучения с учителем для задач классификации. Иногда он работает быстрее и точнее логистической регрессии. Чтобы использовать метод опорных векторов, мы модифицируем первый член функции стоимости [](about:blank), так что при [](about:blank) (здесь и далее будем обозначать эту величину как z) больше 1, оно выводит 0. Кроме того, для значений z, меньших 1, мы будем использовать прямую убывающую линию вместо сигмовидной кривой (в англоязычной литературе это называется функцией hinge loss). Аналогично, мы модифицируем второй член функции стоимости [](about:blank), так что, когда z меньше -1, алгоритм выдает 0. Мы также модифицируем его так, чтобы при значениях z больше -1, вместо сигмоидной кривой мы используем прямую растущую линию.

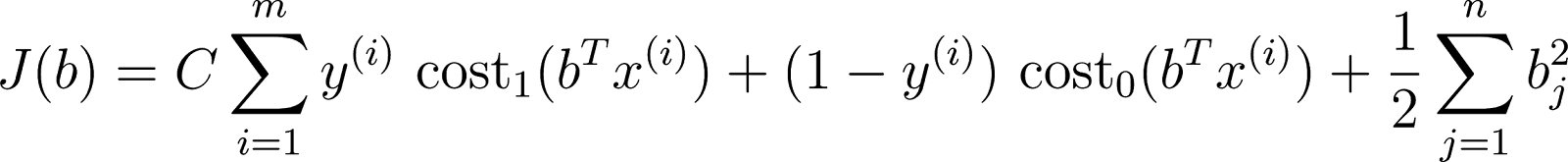
Мы будем обозначать соответствующие компоненты функции ошибки как [](about:blank) и [](about:blank) (соответственно [](about:blank) является стоимостью для классификации при y = 1 и [](about:blank) - это стоимость классификации при y = 0), и мы можем определить их следующим образом (где k - произвольная константа, определяющая величину наклона линии):

[](about:blank) [](about:blank) [](about:blank)

Вспомним полную функцию стоимости из регуляризованной логистической регрессии. Мы можем преобразовать ее в функцию стоимости для векторных машин поддержки, заменив cost0 и cost1.

[](about:blank)

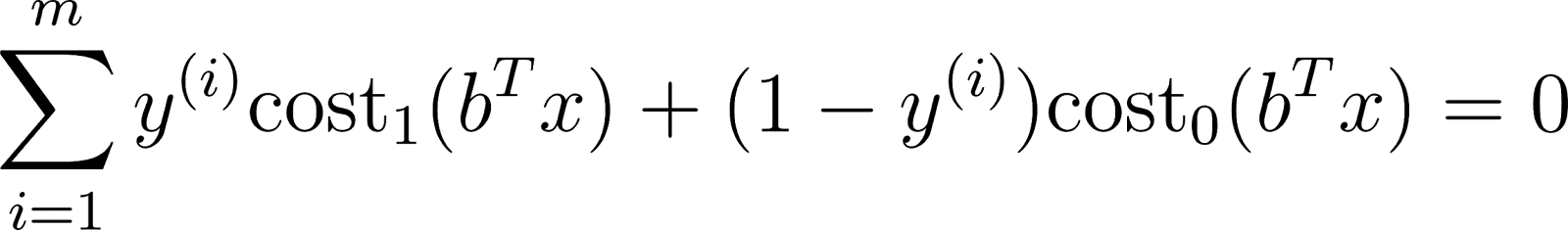
Кроме того, общепринятое соглашение диктует, что мы регуляризуем с использованием фактора C вместо λ, тогда функция ошибки будет выглядеть следующим образом:

[](about:blank)

Это эквивалентно умножению уравнения на [](about:blank) и, таким образом, приводит к тем же значениям при оптимизации. Теперь, когда мы хотим еще больше регуляризовать (то есть сократить возможность переобучения), мы уменьшаем C, и когда мы хотим регуляризовать меньше (то есть уменьшать возможность недообучения), мы увеличиваем C.

Наконец, обратим внимание, что значение функции гипотезы в методе опорных векторов не интерпретируется как вероятность того, что y будет равен 1 или 0 (как для гипотезы логистической регрессии). Вместо этого он выводит либо 1, либо 0 дискретно.

Полезный способ думать о методе опорных векторов - как о методе, максимизирующем расстояние между классами и границей принятия решения.  Когда мы установим нашу константу C в очень большое значение, наша оптимизирующая функция будет ограничивать b так, чтобы уравнение суммы ошибки каждого примера равна 0. Мы накладываем следующие ограничения на b: [](about:blank)

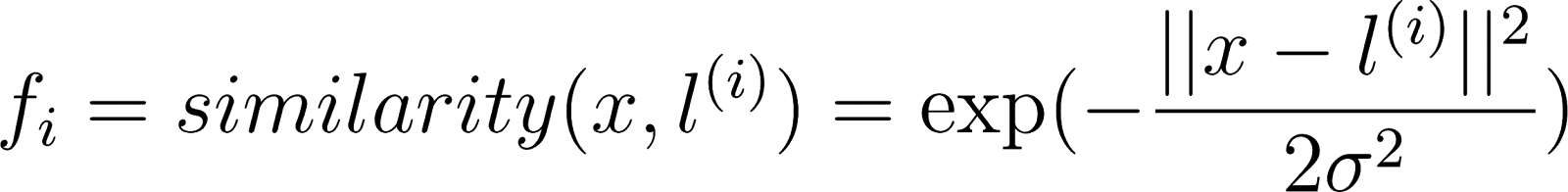
Если C очень велико, мы должны выбрать такие параметры b, что: [](about:blank)

Напомним, что граница принятия решения в логистической регрессии — это линия, отделяющая положительные и отрицательные примеры. В SVM граница решения имеет особое свойство, заключающееся в том, что она как можно дальше от положительного и отрицательного примеров. SVM будет отделять отрицательные и положительные примеры с большим отрывом. Этот большой запас достигается только тогда, когда C очень большой.

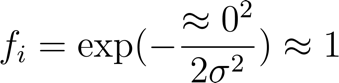
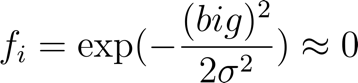
Данные называются линейно разделимыми, когда прямая линия, плоскости или гиперплоскость (в зависимости от размерности данных) может отделить положительные и отрицательные примеры. Увеличение и уменьшение C аналогично уменьшению и увеличению λ и может упростить вид границы принятия решения.

1. **Понятие ядра и виды ядер в методе опорных векторов.**

Ядра (kernels) позволяют нам создавать сложные нелинейные классификаторы с использованием метода опорных векторов. При данном x можно ввести новые признаки в зависимости от близости x к определенным заранее выбранным точкам (ориентирам), скажем [](about:blank). Для этого мы находим «подобие» x и некоторой точки  [](about:blank):

[](about:blank)

Эта функция «подобия» называется гауссовским ядром. Это конкретный пример ядра. Существует несколько свойств функции подобия:

* Если [](about:blank), тогда [](about:blank)
* Если x далек от [](about:blank), тогда [](about:blank)

Другими словами, если x и ориентир близки, то сходство будет близким к 1, и если x и ориентир находятся далеко друг от друга, сходство будет близко к 0.

Каждый ориентир дает нам набор признаков для нашей модели:

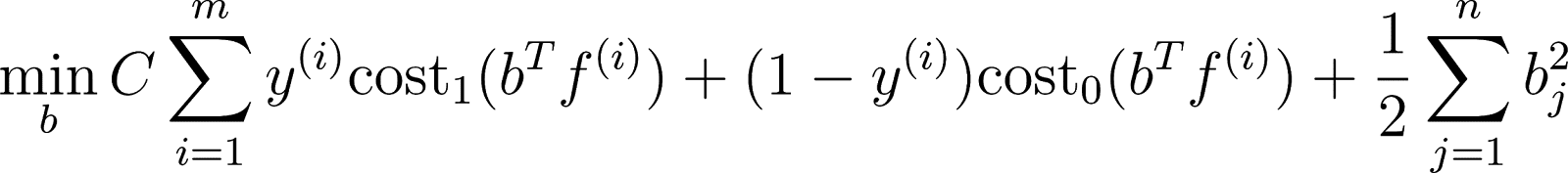
[](about:blank) [](about:blank) [](about:blank)

[](about:blank)

σ2 - параметр гауссовского ядра, и его можно модифицировать, чтобы увеличить или уменьшить область действия нашей функции fi. В сочетании с изменением значений внутри b, мы можем выбрать эти ориентиры, чтобы получить общую форму границы принятия решения.

Один из способов получить ориентиры - разместить их в тех же точках гиперпространства, что и все учебные примеры. Это дает нам набор ориентиров, с одним ориентиром на каждый пример обучения. Это дает нам «вектор функций» [](about:blank) для всех наших изначальных признаков, например [](about:blank). Мы можем также установить [](about:blank), чтобы соответствовать b0. Таким образом, данный пример обучения [](about:blank).

Теперь, чтобы получить параметры b, мы можем использовать алгоритм минимизации SVM, но с заменой [](about:blank) на [](about:blank):

[](about:blank)

Использование ядер для генерации f(i) не является исключительным для SVM и может также применяться к логистической регрессии. Однако из-за вычислительной оптимизации в алгоритмах SVM, ядра работают с SVM, намного быстрее, чем с другими алгоритмами, поэтому ядра почти всегда встречаются вместе только с SVM.

Выбор ядра (функция подобия)

* Нет ядра («линейное» ядро) - дает стандартный линейный классификатор. Выберем, когда n велико, а m мало
* Гауссовское ядро ​​(описанное выше) - нужно выбрать σ2. Выберите, когда n мало и m велико
* Полиномиальное: 
* Радиальная базисная функция: 
* Гауссова радиальная базисная функция: 
* Сигмоид: 

1. **Метод k ближайших соседей в задачах классификации.**

Метод K-ближайших соседей – это алгоритм [Машинного обучения (ML)](https://www.helenkapatsa.ru/mashinnoie-obuchieniie/), который используют для решения задач классификации и регрессии. Алгоритм [Контролируемого обучения (Supervised Learning)](https://www.helenkapatsa.ru/kontroliruiemoie-obuchieniie/), в отличие от Неконтролируемого (Unsupervised Learning), полагается на размеченные входные данные для получения соответствующего результата с новыми данными без [Ярлыков (Label)](https://www.helenkapatsa.ru/iarlyk/).

Алгоритм kNN состоит из трех последовательных этапов:

1) вычислить расстояние от целевого объекта (который необходимо классифицировать) до каждого из объектов обучающей выборки (уже маркированных каким-либо классом);

2) отобрать k объектов обучающей выборки, расстояния до которых минимальны (на первом этапе k выбирается произвольно, затем итеративно подбирается лучшее значение k на основе точности полученных прогнозов при каждом из выбранных k);

3) получить класс объекта на основе наиболее часто встречающегося среди k ближайших соседей (это может быть число или название класса в зависимости от того, как изначально были обозначены классы - например, в примере с беспилотниками это может быть "человек" или "бетонный блок").

Метод ближайших соседей заключается в предсказании значения исходя из значения ближайших к нему объектов обучающей выборки. Чаще всего используется евклидова метрика расстояния между объектами. Не имеет внутренних параметров. Можно считать, что параметры — это сама обучающая выборка. Нужно выбрать k - количество используемых соседей. Чем больше k, тем проще модель и больше вероятность недообучения. Самый простой алгоритм для классификации. Опирается на гипотезу компактности: схожие объекты чаще принадлежат одному классу, чем разным. Данные обязательно надо нормализовать. Алгоритм не чувствителен к выбросам. Алгоритм работает медленно при большом объеме обучающей выборки.

1. **Метод решающих деревьев в задачах классификации.**

**Деревья решений (DT)** — это непараметрический контролируемый метод обучения, используемый для [классификации](https://scikit-learn.ru/1-10-decision-trees/?ysclid=ld2odp9d2w671889961#tree-classification) и [регрессии](https://scikit-learn.ru/1-10-decision-trees/?ysclid=ld2odp9d2w671889961#tree-regression) . Цель состоит в том, чтобы создать модель, которая предсказывает значение целевой переменной, изучая простые правила принятия решений, выведенные из характеристик данных. Дерево можно рассматривать как кусочно-постоянное приближение.

Решающее дерево предсказывает значение целевой переменной с помощью применения последовательности простых решающих правил (которые называются предикатами). Этот процесс в некотором смысле согласуется с естественным для человека процессом принятия решений.

Хотя обобщающая способность решающих деревьев невысока, их предсказания вычисляются довольно просто, из-за чего решающие деревья часто используют как кирпичики для построения [ансамблей](https://ysda_trove.gitlab.io/ml-handbook/chapters/ensembles/intro) — моделей, делающих предсказания на основе агрегации предсказаний других моделей.

Дерево решений — это граф, в узлах которого содержатся некоторые условия (как правило, пороговые условия по признаку). Деревья решений являются одним из самых популярных методов классического машинного обучения. Используются как для построения модели, так и для предварительного анализа данных. Дерево решений просто построить, ему не требуются огромные вычислительные мощности. Деревья решений - интерпретируемый метод, его можно объяснить и понять, особенно при малых глубинах дерева. Деревья решений более лояльны к входным данным - могут работать с категориальными признаками. Деревья решений склонны к переобучению при достаточно больших глубинах. Дерево решений очень толерантно к лишним признакам в наборе данных. Существуют специальные виды распределений, которые сложно описываются деревьями решений.

Некоторые преимущества деревьев решений:

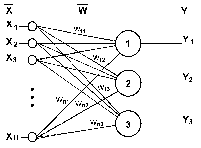
* Просто понять и интерпретировать. Деревья можно визуализировать.
* Требуется небольшая подготовка данных. Другие методы часто требуют нормализации данных, создания фиктивных переменных и удаления пустых значений. Однако обратите внимание, что этот модуль не поддерживает отсутствующие значения.
* Стоимость использования дерева (т. Е. Прогнозирования данных) является логарифмической по количеству точек данных, используемых для обучения дерева.
* Может обрабатывать как числовые, так и категориальные данные. Однако реализация scikit-learn пока не поддерживает категориальные переменные. Другие методы обычно специализируются на анализе наборов данных, содержащих только один тип переменных. См. [Алгоритмы](https://scikit-learn.ru/1-10-decision-trees/?ysclid=ld2odp9d2w671889961#tree-algorithms) для получения дополнительной информации.
* Способен обрабатывать проблемы с несколькими выходами.
* Использует модель белого ящика. Если данная ситуация наблюдаема в модели, объяснение условия легко объяснить с помощью булевой логики. Напротив, в модели черного ящика (например, в искусственной нейронной сети) результаты могут быть труднее интерпретировать.
* Возможна проверка модели с помощью статистических тестов. Это позволяет учитывать надежность модели.
* Работает хорошо, даже если его предположения несколько нарушаются истинной моделью, на основе которой были сгенерированы данные.

К недостаткам деревьев решений можно отнести:

* Обучающиеся дереву решений могут создавать слишком сложные деревья, которые плохо обобщают данные. Это называется переобучением. Чтобы избежать этой проблемы, необходимы такие механизмы, как обрезка, установка минимального количества выборок, необходимых для конечного узла, или установка максимальной глубины дерева.
* Деревья решений могут быть нестабильными, поскольку небольшие изменения в данных могут привести к созданию совершенно другого дерева. Эта проблема смягчается за счет использования деревьев решений в ансамбле.
* Как видно из рисунка выше, предсказания деревьев решений не являются ни гладкими, ни непрерывными, а являются кусочно-постоянными приближениями. Следовательно, они не годятся для экстраполяции.
* Известно, что проблема обучения оптимальному дереву решений является NP-полной с точки зрения нескольких аспектов оптимальности и даже для простых концепций. Следовательно, практические алгоритмы обучения дереву решений основаны на эвристических алгоритмах, таких как жадный алгоритм, в котором локально оптимальные решения принимаются в каждом узле. Такие алгоритмы не могут гарантировать возврат глобального оптимального дерева решений. Это можно смягчить путем обучения нескольких деревьев в учащемся ансамбля, где функции и образцы выбираются случайным образом с заменой.
* Существуют концепции, которые трудно изучить, поскольку деревья решений не выражают их легко, например проблемы XOR, четности или мультиплексора.
* Ученики дерева решений создают предвзятые деревья, если некоторые классы доминируют. Поэтому рекомендуется сбалансировать набор данных перед подгонкой к дереву решений.

1. **Однослойный перцептрон в задачах классификации.**

**Однослойный** персептрон (персептрон Розенблатта) - однослойная *нейронная сеть*, все нейроны которой имеют жесткую пороговую функцию активации. Однослойный *персептрон* имеет простой алгоритм обучения и способен решать лишь самые простые задачи. Эта модель вызвала к себе большой интерес в начале 1960-х годов и стала толчком к развитию искусственных *нейронных сетей*. Классический пример такой *нейронной сети* - однослойный трехнейронный *персептрон.*

Как уже отмечалось, однослойный персептрон может применяться для задач классификации образов, прогнозирования и т. д. В случае задачи классификации однослойный персептрон с пороговой или не прерывной, возрастающей, ограниченной функцией активации (например, сигмовидной) формирует линейную разделяющую поверхность, т. е. он способен решать только линейно разделимые задачи классификации. При использовании сигнальной функции активации он способен также решать некоторые линейно неразделимые задачи классификации, например «ИСКЛЮЧАЮЩЕЕ ИЛИ».

1. **Метрики эффективности и функции ошибки: назначение, примеры, различия.**

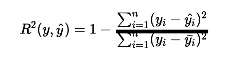
У каждой модели есть функция ошибки, которая показывает, на сколько модель соответствует эмпирическим значениям. Однако, использование функции ошибки не очень удобно для оценки именно “качества” уже построенных моделей. Ведь эта функция специально создается для единственной цели - организации процесса обучения. Поэтому для оценки уже построенных моделей используется не функция ошибки, а так называемые метрики эффективности - специальные функции, которые показывают, насколько эффективна уже готовая, обученная модель.

Функция ошибки нужна в первую очередь для формализации процесса обучения модели. То есть для того, чтобы подбирать параметры модели именно такими, чтобы модель как можно больше соответствовала реальным данным в обучающей выборке. Функция ошибки нужна, чтобы формализовать отклонения предсказанных моделью значений от реальных.

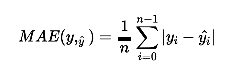
**Функция ошибки:** Используется для организации процесса обучения, Используется для нахождения оптимума, Должна быть быстро вычислимой, Должна конструироваться исходя их типа модели, Может быть только одна

**Метрика эффективности:** Используется для оценки качества полученной модели, Используется для сравнения моделей между собой, Должна быть понятной, Должна выбираться исходя из задачи, Может быть несколько

**Коэффициент детерминации (r-квадрат)**

Коэффициент детерминации показывает силу связи между двумя случайными величинами. Если модель всегда предсказывает точно, метрика равна 1. Для тривиальной модели - 0. Значение метрики может быть отрицательно, если модель предсказывает хуже, чем тривиальная. Это одна из немногих несимметричных метрик эффективности. Эта метрика не определена, если y=const. Надо следить, чтобы в выборке присутствовали разные значения целевой переменной.

**Средняя абсолютная ошибка (MAE)**

MAE показывает среднее абсолютное отклонение предсказанных значений от реальных. Чем выше значение MAE, тем модель хуже. У идеальной модели MAE=0. MAE очень легко интерпретировать - на сколько в среднем ошибается модель.

1. **Предварительный анализ данных: задачи, методы, цели.**

В ходе предварительного анализа определяют соответствие имеющихся данных требованиям, предъявляемым к ним математическими методами (объективности, сопоставимости, полноты, однородности и устойчивости). Подготовка данных для машинного обучения начинается с исследовательского анализа данных. В результате выявляют параметры данных, закономерности, ошибки, аномалии и пропуски. В зависимости от качества и формата исходных данных на этапе предварительной обработки стоят следующие задачи:

1. Очистить данные.

При очистке данных удаляют устаревшие данные, дубликаты, аномалии, пропуски и ошибки. Но не всегда можно просто убрать все некачественные данные. Иногда их так много, что удаление повлияет на результаты машинного обучения, поэтому данные придётся редактировать.

2. Редактировать данные.

Данные могут быть записаны с ошибками или в разных форматах, поэтому их нужно корректировать. Модель будет воспринимать схожие названия как разные значения, поэтому придётся привести записи к одному формату. Что касается числовых данных, то, чтобы привести их к единому формату, например, можно преобразовать значения в диапазон от 0 до 1.

3. Заполнить пропуски.

В работе с пропусками есть разные подходы. Их выбор зависит от видов и источников данных. Пропуски можно заполнить наиболее вероятным значением. Например, в форме для объявления о продаже автомобиля есть графа с информацией об участии в авариях. В графе есть два варианта ответа — «да» и «нет». Специалист, который проводит подготовку данных для машинного обучения, знает, что на основе статистики эту графу чаще пропускают в случае отсутствия аварий. Поэтому он может заполнить пропуски наиболее вероятным значением — «нет». Числовые показатели можно заменить, например, на усреднённые значения или построить алгоритм на основе взаимосвязей между показателями. С помощью такого алгоритма для каждого пропуска рассчитывается собственное значение.

4. Форматировать данные.

Инструменты машинного обучения, как правило, работают с данными в табличном формате. Поэтому набор чисел и текстовые записи преобразовывают в форматы .csv, .xls, .xlsx. Исходные данные в виде изображений тоже преобразовывают. Их можно конвертировать в один формат или сжать до определённого размера. К изображениям могут применять чёрно-белый или другой единый цветовой фильтр и обрезать. Например, если нужна информация из конкретного поля на скане документа, которое заполняется от руки, то все сканы можно автоматически обрезать на определённое количество пикселей со всех сторон.

5. Отобрать признаки данных.

Некоторые признаки могут быть сильно связаны между собой, поэтому приводят к утечке данных. Допустим, есть два признака — год рождения и возраст. Если выгрузка данных происходит в один день, то один из этих признаков стоит удалить. Отбор признаков также делают, чтобы снизить эффект шума. В этом случае удаляют те признаки, которые в наименьшей степени влияют на целевой показатель. Предположим, нужно сделать прогноз размера заработной платы. На этот показатель в большей степени повлияет сфера занятости и стаж работы, а вот день и месяц рождения сотрудника нет, значит, эти значения можно убрать.

1. **Понятие набора данных (датасета) в машинном обучении. Требования, представление. Признаки и объекты.**

**Датасет** — это набор данных, используемый для обучения моделей. **Объекты** — это элементарные сущности, которые мы изучаем, объекты реального мира, измерения, наблюдения. (**строка датасета**). Каждый объект характеризуется набором атрибутов. В датасете у всех объектов одинаковый набор атрибутов, а значения - разные. Атрибут или переменная — это свойство объектов в датасете, **признак** — это **колонка данных**, которая подается на вход модели машинного обучения.

Требования:

* Данные представлены в виде единой таблицы.
* Строки таблицы представляют собой измерение, точку данных, объект предметной области.
* Колонки таблицы представляют собой атрибуты объектов, признаки, переменные.
* Каждая таблица, файл представляет собой данные об одном виде наблюдений или экспериментов.
* Дополнительно: Все данные должны быть выражены в численном виде.
* Дополнительно: В данных не должно быть отсутствующих (пропущенных) значений.

**Представление данных.** Основная цель машинного обучения — строить модели путем интерпретации данных. Для этого очень важно подавать данные таким образом, чтобы их мог прочитать компьютер. Чтобы передать данные в модель scikit-learn, они должны быть представлены в виде таблицы или матрицы требуемой размерности.

Таблицы данных. Большинство таблиц, используемых в задачах машинного обучения, являются двумерными, то есть содержат строки и столбцы. Обычно каждая строка представляет экземпляр, тогда как каждый столбец представляет характеристику (признак) каждого наблюдения.

Особенности и целевые матрицы. Для многих проблем с данными одна из характеристик набора данных будет использоваться в качестве метки. Это означает, что из всех других признаков именно этот является целью, до которой модель должна обобщать данные.

Матрица признаков: Матрица признаков содержит данные из каждого экземпляра для всех признаков, кроме целевого. Он может быть создан с использованием массива NumPy или Pandas DataFrame.

Целевая матрица: В отличие от матрицы признаков, целевая матрица обычно является одномерной, поскольку она содержит только одну функцию для всех экземпляров.

Подобно матрице функций, целевая матрица обычно создается в виде массива NumPy или серии Pandas. Значения целевого массива могут быть дискретными или непрерывными.

1. **Шкалы измерения признаков. Виды шкал, их характеристика.**

Шкала измерения — это способ представления переменных (признаков, атрибутов) и их группировки в различные категории. Она определяет характер значений, присвоенных переменным в наборе данных.

**Описание шкалы измерения атрибутов****.** Шкалы атрибутов показывают, какие значения может принимать этот атрибут и как его можно интерпретировать и сравнивать.Все атрибуты подразделяются на численные (непрерывные) и категориальные (дискретные). Категориальные переменные потом придется преобразовать в численные. **Номинальная шкала** — это признак, для которого имеет смысл только равенство. Пример - метка класса.Номинальная шкала (категориальная) — это шкала измерения, которая используется для идентификации. Она присваивает номера атрибутам для удобства идентификации, но может использоваться только как метка. Единственный вид статистического анализа, который можно выполнить с использованием номинальной шкалы, это вычисление процентных долей и частот. Данные в номинальной шкале можно проанализировать графически с помощью гистограммы и круговой диаграммы. **Порядковая шкала (ординальная шкала)** — это когда наряду с равенством мы можем выстроить значения по порядку, который имеет смысл. Пример - класс обслуживания. Порядковая шкала (ординальная) — предполагает упорядочивание значений переменной в зависимости от масштабирования. Атрибуты в порядковой шкале обычно располагаются в порядке возрастания или убывания. Порядковая шкала может быть использована в исследованиях рынка, рекламы и опросов удовлетворенности клиентов. В порядковой шкале можно использовать для статистического анализа такие статистики как медиана, но не среднее значение.Ординальные признаки можно преобразовывать методом LabelEncoder. Номинальные - только OneHotEncoder. **Интервалая шкала** — это когда еще имеет смысл говорить о разнице между двумя значениями. Пример – даты. Интервальная шкала (разностей) — это шкала, в которой уровни упорядочены, а интервалы между ними равны. Интервальная шкала не только позволяет однозначно определить, какое значение больше (меньше), но и на сколько. Кроме того, в отличие от порядковой и номинальной шкал, в интервальной могут выполняться арифметические операции. Интервальную шкалу можно использовать при расчете среднего значения, медианы, моды, стандартного отклонения и других статистик. **Абсолютная шкала** — это когда еще можно говорить о том, во сколько раз одно значение больше другого. Пример - сумма денег. Она может рассматриваться как расширение интервальной шкалы, и следовательно, удовлетворяет четырем свойствам шкалы измерения: идентифицируемостью, величиной, равноинтервальностью и наличием абсолютного нуля. Абсолютная шкала совместима со всеми методами статистического анализа и может использовать как показатели центральной тенденции, так и разброса значения.

1. **Понятие чистых данных. Определение, очистка данных.**

**Понятие чистых данных.** Данные представлены в виде единой таблицы.Строки таблицы представляют собой измерение, точку данных, объект предметной области. Колонки таблицы представляют собой атрибуты объектов, признаки, переменные. Каждая таблица, файл представляет собой данные об одном виде наблюдений или экспериментов. Дополнительно: Все данные должны быть выражены в численном виде. Дополнительно: В данных не должно быть отсутствующих (пропущенных) значений.

**Очистка и преобразование данных.** Очистка данных – это важный процесс подготовки исходных данных для анализа моделей машинного обучения. Необработанные данные могут содержать многочисленные ошибки, которые повлияют на точность моделей машинного обучения.

**Удаление лишних признаков (feature selection):** Выбор признаков - важная часть преобразования данных, ведь чем точнее мы определим необходимую для моделирования информацию, тем эффективнее будет проходить обучение. Не стоит оставлять в датасете ненужные признаки — это повышает вариативность моделей и может приводить к переобучению. Для выбора признаком часто используют выводы из EDA, обучение простых моделей (DT, RF) или здравый смысл. Выбор признаков можно автоматизировать - применять алгоритм очередного удаления или добавления признаков. Выбор признаков менее актуален, если обучаются нейронные сети, они способны сами отбирать признаки. Регуляризация обычно работает лучше отбора признаков. Совсем лишние данные все равно надо убирать.

**Удаление непоказательных объектов:** Аномальные объекты часто удаляются из датасета, чтобы не искажать результаты обучения. Если можно исправить выброс, можно попытаться это сделать. Следует анализировать, показателен ли объект для предметной области. Если нет - можно смело удалять. После удаления множества объектов может понадобиться повторить анализ сбалансированности классов.

1. **Основные этапы проекта по машинному обучению.**

**Краткий сценарий анализа данных:**

* Интеграция и очистка данных.
* Подробное описание каждого признака, шкалы, вида распределения.
* Исследование отсутствующих значений, заполнение или удаление.
* Описание вида распределения каждого значимого признака и целевой переменной.
* Преобразования категориальных признаков в численные.

**Подробный сценарий анализа данных**

* Оценка источников и объемов данных
* Анализ репрезентативности и однородности разделения выборки.
* Гипотезы о виде распределения каждого признака и выявление аномалий, визуализация.
* Выявление и исправление несбалансированности классов.
* Корреляционная матрица, выявление мультиколлинеарности, важность признаков.
* Отбор признаков, инжиниринг признаков, возможная группировка значений.

Определение задачи — какую бизнес-проблему вы пытаетесь решить? Как это можно сформулировать в виде задачи машинного обучения?

Данные — какие данные у вас есть? Как это соответствует определению задачи? Данные структурированы или нет? Данные статические или потоковые?

Оценка — что определяет успех? Достаточно ли хороша модель машинного обучения с точностью 95 %?

Особенности — какие части данных будут использованы для модели? Как может то, что уже известно, повлиять на это?

Моделирование — какую модель выбрать? Как вы можете улучшить модель? Как вы сравниваете это с другими моделями?

Эксперименты — что ещё можно попробовать? Развёрнутая модель работает так, как ожидалось? Как другие шаги меняются в зависимости от того, что вы обнаружили?

1. **Проблема отсутствующих данных: причины, исследование, пути решения.**

Наличие пропусков в данных, так же, как и анализ только полных наблюдений (после исключения наблюдений с пропусками), может привести к получению смещенных результатов, и как следствие к искажению выводов, которые могут быть сделаны по результатам исследования и принятию неверных стратегических решений. **Заполнение отсутствующих значений.** При выборе стратегии борьбы с пропусками в данных следует проанализировать, сколько пропусков и где они расположены. Если по одному из признаков большинство значений пропущено, можно задуматься об удалении этого признака из датасета. Если наоборот, по одному объекту много неизвестных атрибутов, можно удалить объект. Следует следить, чтобы от датасета что-то осталось после массового удаления. Самый простой способ заполнить пропуски - заполнить их средним значением, но это сильно искажает форму распределения признака. Зачастую можно считать групповое среднее, более “индивидуальную” оценку атрибута данного объекта, с учетом значений других атрибутов. Иногда применяют заполнение случайным значением. Его лучше всего генерировать из распределения данного признака. Самое грамотное решение - введение нового признака или заполнение специальным значением (по категориальным признакам).

1. **Проблема несбалансированных классов: исследование, пути решения.**

**Проблема несбалансированных классов.** Сбалансированность классов - это равномерность распределения целевой переменной в задачах классификации. Сильно несбалансированные классы плохо сказываются на эффективности обучения, так как модель подстраивается под мажоритарный класс. Самый простой способ устранения - удаление случайной части объектов мажоритарного класса, но это приводит к кратному уменьшения датасета. Можно попробовать добавить объектов миноритарных классов, но это не всегда возможно. С несбалансированными классами неплохо справляются иерархические классификаторы. Можно попробовать модифицировать алгоритм обучения, придав классам веса. Если ничего не помогает, возможно следует пересмотреть постановку задачи, разбить мажоритарный класс, переформулировать проблему. Нужно проводить анализ ошибок. Метод NearMiss — это метод недостаточной выборки. Он пробует сбалансировать распределение классов путём случайного исключения наблюдений из бо́льших классов. Если экземпляры из двух разных классов очень похожи между собой, метод удаляет наблюдение из мажоритарного класса. Большинство вероятностных моделей слабо зависят от баланса классов. Проблемы обычно возникают, когда мы переходим к не-вероятностной или многоклассовой классификации.

1. **Понятие параметров и гиперпараметров модели. Обучение параметров и гиперпараметров. Поиск по сетке.**

**Параметр модели** — это переменная конфигурации, которая является внутренней по отношению к модели и значение которой можно оценить на основе данных. Они требуются моделью при прогнозировании. Эти значения определяют умение модели по вашей проблеме. Они оцениваются или извлекаются из данных. Они часто не устанавливаются вручную. Они часто сохраняются как часть изученной модели. **Гиперпараметры модели.** Гиперпараметр модели — это численное значение, которое влияет на работу модели, но не подбирается в процессе обучения. Примеры гиперпараметров - k в kNN, параметр регуляризации, степень полиномиальной регрессии, глубина дерева решения. У каждой модели множество гиперпараметров, которые можно посмотреть в документации. Гиперпараметры модели нужно задавать до начала обучения. Если значение гиперпараметра изменилось, то обучение надо начинать заново. Существуют скрытые гиперпараметры модели - степень полинома, количество нейронов и слоев, ядерная функция. Оптимизация гиперпараметров и задача выбора модели - одно и то же.Таблица различий между параметрами модели и гиперпараметрами:

|  |  |
| --- | --- |
| **Параметры** | **Гиперпараметры** |
| Они необходимы для прогнозирования | Они необходимы для оценки параметров модели |
| Они оцениваются алгоритмами оптимизации (Gradient Descnet, Adam, Adagrad) | Они оцениваются путем настройки гиперпараметров |
| Они не устанавливаются вручную | Они устанавливаются вручную |
| Окончательные параметры, найденные после обучения, будут определять, как модель будет работать с невидимыми данными | Выбор гиперпараметров определяет, насколько эффективно обучение. При градиентом спуске скорость обучения определяет, насколько эффективен и точен процесс оптимизации при оценке параметров |

**Поиск по сетке.** Поиск по сетке - полный перебор всех комбинаций значений гиперпараметров для поиска оптимальных значений. Для его организации надо задать список гиперпараметров и их конкретных значений. Непрерывные гиперпараметры надо дискретизировать. Поиск по сетке имеет экспоненциальную сложность. Чем больше параметров и значений задать, тем лучше модель, но дольше поиск. Можно задать критерии поиска - целевые метрики. **Случайный поиск.** Случайный поиск позволяет задать распределение гиперпараметра, в котором будет вестись поиск. Случайный поиск семплирует набор значений гиперпараметров из указанных распределений. Можно задать количество итераций поиска независимо от количества гиперпараметров. Добавление параметров не влияет на продолжительность поиска. Результат не гарантируется. Воспроизводимость можно настроить

1. **Понятие недо- и переобучения. Определение, пути решения.**

**Переобучение** (англ. overfitting) — негативное явление, возникающее, когда алгоритм обучения вырабатывает предсказания, которые слишком близко или точно соответствуют конкретному набору данных и поэтому не подходят для применения алгоритма к дополнительным данным или будущим наблюдениям. **Недообучение** (англ. underfitting) — негативное явление, при котором алгоритм обучения не обеспечивает достаточно малой величины средней ошибки на обучающей выборке. Недообучение возникает при использовании недостаточно сложных моделей.

**Обнаружение пере- и недообучения:**

При недообучении тестовая и обучающая эффективности будут достаточно близкими, но недостаточными. При переобучении тестовая и обучающая эффективности будут сильно различаться - тестовая будет значительно ниже. Пере- и недообучение — это относительные понятия. Более простые модели склонны к недообучению, более сложные - к переобучению. Диагностика пере- и недообучения очень важна, так как для повышения эффективности предпринимаются противоположные меры. Для построения можно использовать функцию ошибки, метрику эффективности или метрику ошибки, важна только динамика этих показателей. Диагностика моделей машинного обучения - это не точная наука, здесь нужно принимать в расчет и задачу, и выбор признаков и многие другие факторы.

**Методы борьбы с недообучением:**

1. Ввести в модель новые данные об объектах (атрибуты).
2. Уменьшение степени регуляризации модели.
3. Введение полиномиальных и других признаков.
4. В целом, инжиниринг признаков.
5. Использование более сложных моделей.

**Методы борьбы с переобучением:**

1. Ввести в модель данные о новых объектах, использовать большую выборку.
2. Убрать признаки из модели, использовать отбор признаков.
3. Увеличить степень регуляризации модели.
4. Использовать более простые модели.
5. Регуляризация обычно работает лучше уменьшения количества параметров.
6. **Диагностика модели машинного обучения. Методы, цели.**

Один из самых наглядных способов диагностики моделей машинного обучения - кривых обучения. Как могли заметить, мы нигде не говорим о четких критериях. Недо- и переобученной моделей — это вообще относительные понятия. И кривые обучения измеряют их только косвенно. Поэтому диагностика не сводится к оценке какой-либо метрики или статистики. Нужно оценить общую форму графика кривых обучения, что не является точной наукой.

Построение кривых обучения может быть проведено после разделения датасета на обучающую и тестовую выборки. Происходит это следующим образом. Тестовый набор фиксируется и каждый раз используется один и тот же. Из обучающего набора же сначала берут малую часть, скажем 10% от общего количества точек в нем. Обучают модель на этой малой части, а затем измеряют ее эффективность на этой части и на постоянной тестовой выборке. Первая оценка называется обучающая эффективность (training score), а вторая - тестовая эффективность (test score). Затем повторяют процесс с чуть большей частью обучающей выборки, например, 20%, затем еще с большей и так, пока мы не дойдем до полной обучающей выборки.

В частности, именно с помощью кривых обучения можно предположить пере- и недообучение модели, что является главной целью диагностики моделей машинного обучения.

Чтобы облегчить задачу диагностики модели очень часто эффективность данной модели рассматривают не абстрактно, а сравнивают с аналогами. Практически всегда выбор моделей осуществляется от простого к сложному - сначала строят очень простые модели. Их тестовая эффективность может задать некоторых базовый уровень, планку, по сравнению к которой уже можно готовить об улучшении эффективности у данной, более сложной модели, насколько это улучшение существенно и так далее. Кроме того, строя кривые обучения нескольких моделей можно получить сравнительное представление о том, как эти модели соотносятся между собой, какие из них более недообученные, какие - наоборот. Диагностика моделей нужна для поиска путей повышения ее эффективности.

1. **Проблема выбора модели машинного обучения. Сравнение моделей.**

Процесс выбора модели в машинном обучении — это поиск компромисса между точностью решающего правила на обучении и его «надежностью» на произвольных объектах генеральной совокупности.

**Выбор модели (model selection).** Просто потому, что алгоритм обучения хорошо подходит для тренировочного набора, это не значит, что это хорошая гипотеза. Ошибка вашей гипотезы, измеренная в наборе данных, с которым вы подготовили параметры, будет ниже, чем на любом другом наборе данных. Чтобы решить эту проблему, мы можем ввести третий набор, валидационный набор, чтобы использовать его как промежуточный набор для обучения гиперпараметров. Тогда наш тестовый набор даст нам точную, не оптимистичную ошибку. Метод выбора гипотезы с помощью валидационного набора:

1. Оптимизируйте параметры b, используя набор тренировок для каждой степени полинома.

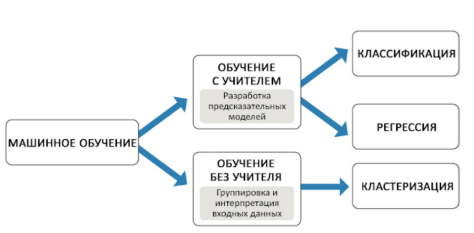
2. Найдите степень полинома d с наименьшей ошибкой, используя валидационный набор.

3. Оцените ошибку обобщения, используя тестовый набор с (где b - параметры от полинома с более низкой ошибкой);

Таким образом, степень полинома d не была обучена с использованием тестового набора.

Задача выбора класса модели для решения определенной задачи. Очень сложно сказать априори какой класс модели будет работать лучше на конкретных данных. Следует учитывать нефункциональные требования к задаче. Обычно начинают с самых простых моделей - они быстро считаются и дают базовый уровень эффективности. По результатам диагностики простых моделей принимают решение о дальнейших действиях. Можно провести поиск по разным классам моделей для определения самых перспективных. Выбор модели — это творческий и исследовательский процесс. Есть подходы автоматизации выбора модели (AutoML), но они пока несовершенны. В исследовательских задачах модели сравниваются со state-of-the-art.

**Сравнение эффективности моделей (валидационный набор).** При сравнении нескольких моделей между собой возникает проблема оптимистичной оценки эффективности. Поэтому для исследования выбранной модели нужно использовать третью часть выборки - валидационную. В терминах существует путаница, главное - три непересекающиеся части выборки. Обучающая (train) используется для оптимизации параметров (обучения) модели. Валидационная (validation) - для оптимизации гиперпараметров и выбора модели. Тестовая (test, holdout) - для итоговой оценки качества, представления результатов. Во многих случаях использование кросс-валидации автоматически разбивает выборку. Поэтому тестовая играет роль валидационной. Есть проблема глобального переобучения моделей на известных датасетах.



1. **Измерение эффективности работы моделей машинного обучения. Метрики эффективности.**

В задачах машинного обучения для оценки качества моделей и сравнения различных алгоритмов используются метрики, а их выбор и анализ — непременная часть машинного обучения.

Метрики эффективности — это способ показать, насколько точно модель отражает реальный мир. Метрики эффективность должны выбираться исходя из задачи, которую решает модель. Функция ошибки и метрика эффективности — это разные вещи, к ним предъявляются разные требования. В задаче можно применять несколько метрик эффективности. Наряду с метриками эффективности есть и другие характеристики моделей - скорость обучения, скорость работы, надежность, робастность, интерпретируемость. Метрики эффективности вычисляются как правило из двух векторов - предсказанных (теоретических) значений целевой переменной и эмпирических (реальных) значений. Обычно метрики устроены таким образом, что чем выше значение, тем модель лучше.

**Метрики эффективности для классификации.** Метрики эффективности классификации подсчитывают количество правильно распознанных объектов. В задачах классификации почти всегда надо применять несколько метрик одновременно. Тривиальной моделью в задачах классификации считается та, которая предсказывает случайный класс, либо самый популярный класс. Качество бинарной классификации при прочих равных почти всегда будет сильно выше, чем для множественной. Вообще, чем больше в задаче классов, тем ниже ожидаемые значения эффективности. Некоторые метрики работают с метрическими методами, другие - со всеми.

**Доля правильных ответов (accuracy).** Точность (accuracy) - самая простая метрика качества классификации, доля правильных ответов. Может быть выражена в процентах и в долях единицы. Идеальная модель дает точность 1.0, тривиальная - 0.5, самая худшая - 0.0. Тривиальная модель в множественной сбалансированной задаче классификации дает точность 1/m. Метрика точности очень чувствительная к несбалансированности классов.

>>> import numpy as np

>>> from sklearn.metrics import accuracy\_score

>>> y\_pred = [0, 2, 1, 3]

>>> y\_true = [0, 1, 2, 3]

>>> accuracy\_score(y\_true, y\_pred)

0.5

**Метрики эффективности для регрессии.** Метрики эффективности для регрессий обычно анализируют отклонения предсказанных значений от реальных. Большинство метрик пришло в машинное обучение из математической статистики. Результаты работы модели можно исследовать более продвинутыми статистическими методами. Обычно метрики сравнивают данную модель с тривиальной - моделью, которая всегда предсказывает среднее реальное значение целевой переменной. Модель могут быть точны на 100%, но плохи они могут быть без ограничений.

Средний квадрат ошибки (MSE). MSE показывает средний квадрат отклонений предсказанных значений от реальных. Чем выше значение MSE, тем модель хуже. У идеальной модели MSE=0. MSE больше учитывает сильные отклонения, но хуже интерпретируется, чем MAE.

>>> from sklearn.metrics import mean\_squared\_error

>>> y\_true = [3, -0.5, 2, 7]

>>> y\_pred = [2.5, 0.0, 2, 8]

>>> mean\_squared\_error(y\_true, y\_pred, squared=False)

0.612...

1. **Метрики эффективности моделей классификации. Виды, характеристика, выбор.**

Метрики эффективности для классификации. Метрики эффективности классификации подсчитывают количество правильно распознанных объектов. В задачах классификации почти всегда надо применять несколько метрик одновременно. Тривиальной моделью в задачах классификации считается та, которая предсказывает случайный класс, либо самый популярный класс. Качество бинарной классификации при прочих равных почти всегда будет сильно выше, чем для множественной. Вообще, чем больше в задаче классов, тем ниже ожидаемые значения эффективности. Некоторые метрики работают с метрическими методами, другие - со всеми.

**Метрики:**

**Доля правильных ответов (accuracy).** Точность (accuracy) - самая простая метрика качества классификации, доля правильных ответов. Может быть выражена в процентах и в долях единицы. Идеальная модель дает точность 1.0, тривиальная - 0.5, самая худшая - 0.0. Метрика точности очень чувствительная к несбалансированности классов.

>>> import numpy as np

>>> from sklearn.metrics import accuracy\_score

>>> y\_pred = [0, 2, 1, 3]

>>> y\_true = [0, 1, 2, 3]

>>> accuracy\_score(y\_true, y\_pred)

0.5

**Метрики классификации для неравных классов (precision, recall, F1).** Если классы в задаче не сбалансированы, то метрика точности не дает полного представления о качестве работы моделей. Для бинарной классификации подсчитывается количество истинно положительных, истинно отрицательных, ложно положительных и ложно отрицательных объектов. Precision - доля истинно положительных объектов во всех, распознанных как положительные. Precision характеризует способность модели не помечать положительные объекты как отрицательные (не делать ложно положительных прогнозов). Recall - для истинно положительных объектов во всех положительных. Recall характеризует способность модели выявлять все положительные объекты (не делать ложно отрицательных прогнозов). F1 - среднее гармоническое между этими двумя метриками. F1 — это частный случай. Вообще, семейство F-метрик — это взвешенное среднее гармоническое. Часто используют все вместе для более полной характеристики модели.

>>> from sklearn import metrics

>>> y\_pred = [0, 1, 0, 0]

>>> y\_true = [0, 1, 0, 1]

>>> metrics.precision\_score(y\_true, y\_pred)

1.0

>>> metrics.recall\_score(y\_true, y\_pred)

0.5

>>> metrics.f1\_score(y\_true, y\_pred)

0.66...

**Матрица классификации.** Матрица классификации, или матрица ошибок представляет собой количество объектов по двум осям - истинный класс и предсказанный класс. Обычно, истинный класс располагается по строкам, а предсказанный - по столбцам. Для идеальной модели матрица должна содержать ненулевые элементы только на главной диагонали. Матрица позволяет наглядно представить результаты классификации и увидеть, в каких случаях модель делает ошибки. Матрица незаменима при анализе ошибок, когда исследуется, какие объекты были неправильно классифицированы.

>>> from sklearn.metrics import confusion\_matrix

>>> y\_true = [2, 0, 2, 2, 0, 1]

>>> y\_pred = [0, 0, 2, 2, 0, 2]

>>> confusion\_matrix(y\_true, y\_pred)

array([[2, 0, 0],

[0, 0, 1],

[1, 0, 2]])

**Метрики множественной классификации.** Метрики для каждого класса рассчитываются, полагая данный класс положительным, а все остальные - отрицательными. Каждую метрику можно усреднить арифметически или взвешенно по классам. Весами выступают объемы классов. В модуле sklearn реализовано несколько алгоритмов усреднения они выбираются исходя их задачи. В случае средневзвешенного F1-метрика может получиться не между P и R. Отчет о классификации содержит всю необходимую информацию в стандартной форме. Отчет показывает метрики для каждого класса, а также объем каждого класса. Также отчет показывает средние и средневзвешенные метрики для всей модели. Отчет о классификации - обязательный элемент представления результатов моделирования. По отчету можно понять сбалансированность задачи, какие классы определяются лучше, какие - хуже.

>>> from sklearn.metrics import classification\_report

>>> y\_true = [0, 1, 2, 2, 0]

>>> y\_pred = [0, 0, 2, 1, 0]

>>> target\_names = ['class 0', 'class 1', 'class 2']

>>> print(classification\_report(y\_true, y\_pred, target\_names=target\_names))

precision recall f1-score support

class 0 0.67 1.00 0.80 2

class 1 0.00 0.00 0.00 1

class 2 1.00 0.50 0.67 2

accuracy 0.60 5

macro avg 0.56 0.50 0.49 5

weighted avg 0.67 0.60 0.59 5

**PR-AUC.** Кривая precision-recall используется для методов метрической классификации, которые выдают вероятность принадлежности объекта данному классу. Дискретная классификации производится при помощи порогового значения. Чем больше порог, тем больше объектов модель будет относить к отрицательному классу. Повышение порога в среднем увеличивает precision модели, но понижает recall. PR-кривая используется чтобы выбрать оптимальное значение порога. PR-кривая нужна для того, чтобы сравнивать и оценивать модели вне зависимости от выбранного уровня порога. PR-AUC - площадь под PR-кривой, у лучшей модели - 1.0, у тривиальной - 0.5, у худшей - 0.0.

**ROC\_AUC.** ROC-кривая показывает качество бинарной классификации при разных значениях порога. В отличие от PR-кривой, ROC-кривая монотонна. Площадь под графиком ROC-кривой, ROC\_AUC - одна из основных метрик качества классификационных моделей. ROC\_AUC можно использовать для сравнения качества разных моделей, обученных на разных данных. ROC чаще используют для сбалансированных и множественных задач, PR - для несбалансированных. Кривые для множественной классификации строятся отдельно для каждого класса. Метрика AUC считается по кривой средних значений.

**Топ k классов.** Эта метрика - обобщение точности для случая, когда модель выдает вероятности отнесения к каждому классу. Вычисляется как доля объектов, для которых правильный класс попадает в список k лучших предсказанных классов. Чем больше k, тем выше метрика, но бесполезнее результат. Эта метрика часто применяется в задачах с большим количеством классов. Применимость этой метрики сильно зависит от характера задачи.

>>> top\_k\_accuracy\_score(y\_true, y\_score, k=2)

0.75

Выбор метрики нужно делать с фокусом на предметную область, предварительно обрабатывая данные и, возможно, сегментируя.

1. **Метрики эффективности моделей регрессии. Виды, характеристика, выбор.**

**Метрики эффективности для регрессии.** Метрики эффективности для регрессий обычно анализируют отклонения предсказанных значений от реальных. Большинство метрик пришло в машинное обучение из математической статистики. Результаты работы модели можно исследовать более продвинутыми статистическими методами. Обычно метрики сравнивают данную модель с тривиальной - моделью, которая всегда предсказывает среднее реальное значение целевой переменной. Модель могут быть точны на 100%, но плохи они могут быть без ограничений.

**Коэффициент детерминации (r-квадрат).** Коэффициент детерминации показывает силу связи между двумя случайными величинами. Если модель всегда предсказывает точно, метрика равна 1. Для тривиальной модели - 0. Значение метрики может быть отрицательно, если модель предсказывает хуже, чем тривиальная. Это одна из немногих несимметричных метрик эффективности. Эта метрика не определена, если y=const. Надо следить, чтобы в выборке присутствовали разные значения целевой переменной.

from sklearn.metrics import r2\_score

def r2(y, y\_):

return 1 - ((y - y\_)\*\*2).sum() / ((y - y.mean())\*\*2).sum()

print(reg.score(X, Y))

print(r2\_score(Y, Y\_))

print(r2(Y, Y\_))

**Средняя абсолютная ошибка (MAE).** MAE показывает среднее абсолютное отклонение предсказанных значений от реальных. Чем выше значение MAE, тем модель хуже. У идеальной модели MAE=0. MAE очень легко интерпретировать - на сколько в среднем ошибается модель.

>>> from sklearn.metrics import mean\_absolute\_error

>>> y\_true = [3, -0.5, 2, 7]

>>> y\_pred = [2.5, 0.0, 2, 8]

>>> mean\_absolute\_error(y\_true, y\_pred)

0.5

**Средний квадрат ошибки (MSE).** MSE показывает средний квадрат отклонений предсказанных значений от реальных. Чем выше значение MSE, тем модель хуже. У идеальной модели MSE=0. MSE больше учитывает сильные отклонения, но хуже интерпретируется, чем MAE.

>>> from sklearn.metrics import mean\_squared\_error

>>> y\_true = [3, -0.5, 2, 7]

>>> y\_pred = [2.5, 0.0, 2, 8]

>>> mean\_squared\_error(y\_true, y\_pred, squared=False)

0.612...

**Среднеквадратичная ошибка (RMSE).** RMSE — это, по сути, корень из MSE. Выражается в тех же единицах, что и целевая переменная. Чаще применяется при статистическом анализа данных. Данная метрика очень чувствительна к аномалиям и выбросам.

>>> from sklearn.metrics import mean\_squared\_error

>>> y\_true = [3, -0.5, 2, 7]

>>> y\_pred = [2.5, 0.0, 2, 8]

>>> mean\_squared\_error(y\_true, y\_pred)

0.375

**Среднеквадратичная логарифмическая ошибка (MSLE).** MSLE это среднее отклонение логарифмов реальных и предсказанных данных. Так же, идеальная модель имеет MSLE=0. Данная метрика используется, когда целевая переменная простирается на несколько порядков величины. Еще эта метрика может быть полезна, если моделируется процесс в экспоненциальным ростом.

>>> from sklearn.metrics import mean\_squared\_log\_error

>>> y\_true = [3, 5, 2.5, 7]

>>> y\_pred = [2.5, 5, 4, 8]

>>> mean\_squared\_log\_error(y\_true, y\_pred)

0.039...

**Среднее процентное отклонение (MAPE).** Идея этой метрики — это чувствительность к относительным отклонениям. Данная модель выражается в процентах и имеет хорошую интерпретируемость. Идеальная модель имеет MAPE=0. Верхний предел - не ограничен. Данная метрика отдает предпочтение предсказанию меньших значений.

>>> from sklearn.metrics import mean\_absolute\_percentage\_error

>>> y\_true = [1, 10, 1e6]

>>> y\_pred = [0.9, 15, 1.2e6]

>>> mean\_absolute\_percentage\_error(y\_true, y\_pred)

0.2666...

**Абсолютная медианная ошибка.** Медианная абсолютная ошибка похожа на среднюю абсолютную, но более устойчива к аномалиям. Применяется в задачах, когда известно, что в данных присутствуют выбросы, аномальные , непоказательные значения. Эта метрика более робастная, нежели MAE.

>>> from sklearn.metrics import median\_absolute\_error

>>> y\_true = [3, -0.5, 2, 7]

>>> y\_pred = [2.5, 0.0, 2, 8]

>>> median\_absolute\_error(y\_true, y\_pred)

0.5

**Максимальная ошибка.** Максимальная ошибка показывает наихудший случай предсказания модели. В некоторых задачах важно, чтобы модель не ошибалась сильно, а небольшие отклонения не критичны. Зачастую эта метрика используется как вспомогательная совместно с другими.

>>> from sklearn.metrics import max\_error

>>> y\_true = [3, 2, 7, 1]

>>> y\_pred = [9, 2, 7, 1]

>>> max\_error(y\_true, y\_pred)

6

Выбор метрики нужно делать с фокусом на предметную область, предварительно обрабатывая данные

1. **Перекрестная проверка (кросс-валидация). Назначение, схема работы.**

Техника кросс-валидации или перекрестной проверки служит для разбиения набора данных. Процесс разбиения набора автоматизируется и становится более робастным.

**Кросс-валидация.** Разбиение выборки на обучающую и тестовую может внести случайные ошибки. Нужно повторить разбиение несколько раз, посчитать метрики и усреднить. Кросс-валидация разбивает выборку на k блоков, каждый из которых используется по очереди как тестовый. Сколько задать k, столько и будет проходов. Обычно берут 3 или 5. Чем больше k тем надежнее оценка, но дольше ее получение, так как модель каждый раз заново обучается. Использование кросс-валидации обязательно для получения робастных оценок. В библиотеке sklearn кросс-валидация (CV) встроена во многие функции.

>>> from sklearn.model\_selection import cross\_validate

>>> from sklearn.metrics import recall\_score

>>> scoring = ['precision\_macro','recall\_macro']

>>> clf = svm.SVC(kernel='linear', C=1, random\_state=0)

>>> scores = cross\_validate(clf, X, y, scoring=scoring)

>>> sorted(scores.keys())

['fit\_time', 'score\_time', 'test\_precision\_macro', 'test\_recall\_macro']

>>> scores['test\_recall\_macro']

array([0.96..., 1. ..., 0.96..., 0.96..., 1. ])

1. **Конвейеры в библиотеке sklearn. Назначение, использование.**

**Конвейер sklearn.** Конвейер — объединение трансформаторов и модели для последовательной обработки данных и предсказания на обработанных данных. Трансформатор в sklearn — класс, в котором определены методы transform и fit\_transform. Модель в sklearn — класс, в котором определен метод predict. Конвейер в sklearn представлен классом Pipeline из под-модуля sklearn.pipeline.

Цель конвейера состоит в том, чтобы собрать несколько шагов, которые могут быть перекрестно проверены вместе при установке различных параметров. Для этого он позволяет устанавливать параметры различных шагов, используя их имена и имя параметра, разделенные символом "\_\_", как в примере ниже. Оценщик шага может быть полностью заменен, установив параметр с его именем на другой оценщик, или преобразователь может быть удален, установив для него значение "passthrough " или "None".

Изображение выглядит как текст

Автоматически созданное описание

1. **Использование методов визуализации данных для предварительного анализа.**

**Визуализация распределения атрибутов.** На начальном этапе зачастую строят индивидуальное эмпирическое распределение каждого признака, что позволяет выдвинуть предварительную гипотезу о виде распределения соответствующей переменной в генеральной совокупности. Вид распределения может оказать влияние на выбор метода нормализации данных (по среднему, по разбросу или стандартизация), выявить проблему смещенных классов, особенно острую в распределении по целевой переменной, индицировать системные ошибки выборки. Анализ индивидуального распределения может также помочь выявить аномальные выбросы, ошибки измерений, или опечатки в данных.

**Оценка влияния атрибутов на целевую переменную.** В случае задачи обучения с учителем весьма полезно оценить форму совместного распределения целевой переменной с каждым признаком в отдельности. Это позволяет сделать первичное эмпирическое предположение о влиянии каждого фактора на результирующую переменную в случае обнаружения распределения, сильно отличающегося от равномерного.

Основные задачи:

• Выдвижение гипотез об относительной важности факторов;

• Обоснование введения суррогатных факторов.

**Построение корреляционной матрицы.** Построение корреляционной матрицы дает очень наглядное и полное представление о влиянии каждого атрибута как на целевую переменную, так и на другие факторы, поэтому многие исследователи используют ее как завершающий этап описательного анализа данных.

Основные задачи:

• Выдвижение гипотезы об относительной важности факторов

• Обнаружение мультиколлинеарности факторов

1. **Исследование коррелированности признаков: методы, цели, выводы.**

Пример: у нас есть две переменные - «время, затрачиваемое на беговую дорожку в считанные минуты» и «сжигаемые калории». Эти переменные сильно коррелируют, поскольку чем больше времени вы проводите на беговой дорожке, тем больше калорий вы будете сжигать. Следовательно, нет смысла хранить оба значения, поскольку только один из них делает то, что вам нужно. Исследование коррелированности признаков важно при предобработке данных, так как это помогает удалять избыточные признаки (как в примере).

Корреляция — это один из основных терминов теории вероятности, показывающий меру зависимости между двумя и более случайными величинами. Коррелограмма — это популярный инструмент визуализации, он дает много информации, а построить его просто. Корреляционная матрица дает представление о степени влияния каждого фактор на каждый и на целевую переменную. Корреляционная матрица по своей природе симметрична относительно главной диагонали, в которой стоят единицы. Коррелограмма может также дать информацию о мультиколлинеарности атрибутов. Стоит обращать внимание как на положительные, так и на отрицательные корреляции. Коррелограмма показывает только линейные зависимости. По необходимости еще строится матрица совместных распределений. Можно проводить и до и после преобразования данных, но, если атрибутов слишком много, матрица получится перегруженной. Построение корреляционной матрицы:

sns.heatmap(data.corr(), annot=True, cmap=’RdYlGn’, linewidth=0.2) #data.corr() 🡪 correlation matrix

fig=plt.gcf()

fig.set\_size\_inches(10,8)

plt.show()

1. **Решкалирование данных. Виды, назначение, применение. Нормализация и стандартизация данных.**

Виды:

**NORMALIZER: ПРИМЕНЕНИЕ НОРМАЛИЗАЦИИ К СТРОКАМ**

Normalizer необходим для нормализации не атрибутов (столбцов), а записей (строк) путем деления на p-норму. Общая формула выглядит так:

p\_norm = sum(X\*\*p) \*\* (1/p)

X = X / p\_norm

Единственным параметром в этом виде нормализации является p, причём:

* если p=1, то p-норма равна сумме значений каждой строки;
* если p=∞, то p-норма равна максимальному значению в каждой строке.

Normalizer можно применять после атрибутивного шкалирования, о которых пойдёт речь дальше.

**STANDARDSCALER: ПРИВЕДЕНИЕ К НОРМАЛЬНОМУ РАСПРЕДЕЛЕНИЮ**

StandardScaler подразумевает приравнивание среднего значения к нулю и/или приравнивание стандартного отклонения к единице. В отличие от Normalizer применяется к атрибутам, т.е. к столбцам, а не строкам. Данный вид шкалирования стремится привести данные к нормальному распределению. StandardScaler рассчитывается для каждого атрибута следующим образом:

z = (X – u) / s, где u — среднее значение или 0 при with\_mean=False, s — стандартное отклонение или 0 при with\_std=False.

**MINMAXSCALER: ПРИВЕДЕНИЕ К ДИАПАЗОНУ [0,1]**

MinMaxScaler применяется для шкалирования в диапазоне [min,max]. Рассчитывается как

X\_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X\_scaled = X\_std \* (max - min) + min

**MAXABSSCALER: ПРИВЕДЕНИЕ К ДИАПАЗОНУ [-1,1]**

С помощью MaxAbsScaler значения атрибутов приводятся к диапазону [-1,1] путем деления на максимальное абсолютное значение каждого атрибута.

**Нормализация и решкалирование признаков.**

Минимаксная нормализация — это изменение входных данных по следующей формуле: Изображение выглядит как стол

Автоматически созданное описание

После преобразования все значения будут лежать в диапазоне x∈[0;1]. Z-оценки или стандартизация производится по формуле: x′=(x−M[x])/σx. В таком случае данный признак приводится к стандартному распределению, то есть такому, у которого среднее 0, а дисперсия - 1.

Нормализация признаков нужна для ускорения обучения и сходимости градиентного спуска. Во многих реализациях моделей нормализация уже встроена и применяется по умолчанию. Основная идея – это сделать так, чтобы все признаки измерялись по одной шкале, то есть лежали в одних пределах. Подразумевает изменение диапазонов в данных без изменения формы распределения. **Нормализация** масштабирует набор данных таким образом, чтобы каждое значение находилось в диапазоне от 0 до 1.

Еще применяют стандартизацию - приведение к стандартному распределению. Стандартизация изменяет форму распределения данных (приводится к нормальному распределению). **Стандартизация** изменяет масштаб набора данных, чтобы иметь среднее значение 0 и стандартное отклонение 1.

В отдельных случая применяются более крутые алгоритмы решкалирования с автоматическим устранением выбросов.

1. **Преобразование категориальных признаков в числовые.**

Под категориальными данными мы понимаем данные, которые не имеют численного представления, они могут иметь, как и два уникальных значения (бинарные признаки), так и более. Напомним, что любой признак можно перевести в категориальный (category или object), если рассматриваемый признак принимает значения из фиксированного списка.

Самый простой способ – обычная нумерация значений (0, 1, 2, …). У данного подхода есть существенный недостаток. Обычно он ведет к плохому результату так как, алгоритмы начинают учитывать бессмысленную упорядоченность значений признаков. Однако данный метод имеет преимущество с точки зрения памяти. Метод реализован в классе sklearn.preprocessing.LabelEncoder:

import pandas

from sklearn.preprocessing import LabelEncoder

df = pandas.read\_excel('Пример данных.xls')

le = LabelEncoder()

le.fit(df['Категория'])

df['Категория\_le']=le.transform(df['Категория'])

df

Следующий способ – dummy-кодирование, также называемое — one-hot. Суть заключается в создании дополнительных N признаков (столбцов), где N – количество уникальных категорий. Новые признаки принимают значения 0 или 1 в зависимости от принадлежности к категории. One-hot encoder значительно увеличивает объем данных, что делает его неэффективным с точки зрения памяти, частично эту проблему решает применение разреженных матриц.

В Python-библиотеке Pandas имеется функция get\_dummies, которая конвертирует категориальные значения в числовые. На вход она принимает массив или объект DataFrame: pd.get\_dummies(data.room\_type).

Есть простейший кодировщий sklearn.preprocessing.LabelEncoder, который каждой категории сопоставляет некоторое целое число (собственно, номер категории). Даже если бы его не было, то такую кодировку несложно написать самому с помощью функции map. Для этого предварительно задаётся словарь, в котором указывается, что и чем кодировать.

1. **Проблема сбора и интеграции данных для машинного обучения.**

**Сбор данных для обучения.** Реляционная форма - объекты, атрибуты и признаки. Датасет - это набор данных, используемый для обучения моделей. Объекты — это элементарные сущности, которые мы изучаем, объекты реального мира, измерения, наблюдения. Каждый объект характеризуется набором атрибутов. В датасете у всех объектов одинаковый набор атрибутов, а значения - разные. Атрибут или переменная — это свойство объектов в датасете, признак — это колонка данных, которая подается на вход модели машинного обучения.

**Понятие чистых данных.** Данные представлены в виде единой таблицы. Строки таблицы представляют собой измерение, точку данных, объект предметной области. Колонки таблицы представляют собой атрибуты объектов, признаки, переменные. Каждая таблица, файл представляет собой данные об одном виде наблюдений или экспериментов. Все данные должны быть выражены в численном виде. В данных не должно быть отсутствующих значений.

**Оценка источников и объемов данных.** После постановки задачи машинного обучения первый этап моделирования - определение источников и объема данных, необходимых для эффективного обучения. Объем данных определяется порядком сложности задачи: чем больше данных, тем более сложные модели можно на них строить. В целом, чем больше данных можно собрать, тем лучше. Данные в датасете должны быть репрезентативной выборкой генеральной совокупности. Источники данных могут быть открытые и закрытые, платные, по подписке, внутренние и внешние. Данные могут быть пакетные и потоковые, с ними надо работать по-разному. Данные можно собирать, генерировать и модифицировать.

**Интеграция данных.** Интеграция данных - это процесс объединения данных из нескольких источников в единый датасет. Объединение датасетов может происходить по горизонтали и по вертикали. Вертикальное объединение датасетов — это, по сути, просто склеивание двух однотипных наборов данных в один. Горизонтальное объединение датасетов происходит обязательно через аналог операции JOIN по ключу. При объединении данных из разных источников необходимо особенно тщательно следить за обозначениями, соглашениями, датами. Интеграция данных - самая трудная часть работы с машинным обучением.

1. **Понятие чистых данных и требования к данным.**

Чистые данные (*clean data) —* это данные, приведенные к единому формату, которые могут быть легко подвержены анализу, а также любому преобразованию типа объединения нескольких наборов данных этого формата, выделения подмножеств и т.п. Процесс предварительной обработки данных является неотъемлемой частью решения задач машинного обучения на практике.

Выводы:

1. Данные представлены в виде единой таблицы.
2. Строки таблицы представляют собой измерение, точку данных, объект предметной области.
3. Колонки таблицы представляют собой атрибуты объектов, признаки, переменные.
4. Каждая таблица, файл представляет собой данные об одном виде наблюдений или экспериментов.
5. Дополнительно: Все данные должны быть выражены в численном виде.
6. Дополнительно: В данных не должно быть отсутствующих (пропущенных) значений.
7. **Основные задачи описательного анализа данных.**

Описательный/разведочный/дескриптивный анализ (exploratory data analysis - EDA) - анализ основных свойств данных, нахождение в них закономерностей, распределений, артефактов и аномалий, зачастую с использованием инструментов визуализации. Задачи:

* Очистка данных: выявление и удаление ошибок, выбросов и отсутствующих значений из набора данных.
* Исследование данных: изучение датасета, чтобы понять распределение и взаимосвязь между признаками и закономерности в данных.
* Преобразование данных: приведение к формату, с которым легче работать и который лучше подходит для анализа.
* Визуализация данных: построение графиков, гистограмм, точечных и столбчатых диаграмм, которые помогают идентифицировать закономерности в данных, понимать их распределение.
* Обобщение данных: создание сводных статистических данных, таких как среднее значение, медиана, мода, стандартное отклонение и процентили, для описания тенденции и дисперсии признаков.
* Сокращение данных: уменьшение размера набора данных, например, путем удаления нерелевантных переменных или путем объединения данных в меньшее количество фич.

1. **Методы визуализации данных для машинного обучения.**

*Диаграмма рассеяния (Scatter plot) в matplotlib*

Диаграмма рассеяния отображает пространство одних вещественных чисел в пространстве других вещественных чисел. Иными словами, каждая точка одного атрибута соответствует каждой точке другого. В matplotlib он имеет название scatter:

plt.xlabel('points')

plt.ylabel('price')

plt.scatter(x=data['points'], y=data['price'])

xlabel и ylabel служат для обозначения осей x и y, соответственно. В качестве аргументов принимает одномерные массивы x и y. Диаграмма рассеяния может служить для визуализации линейных моделей машинного обучения, такие как линейная регрессия или метод опорных векторов.

*Линейный график (plot) в matplotlib*

Линейный график необходим для построения линии от точки к точке. В matplotlib.pyplot он называется plot:

d = data.groupby('points').mean()

plt.xlabel('points')

plt.ylabel('Средняя цена')

plt.plot(d.index, d.values)

Вы можете увидеть подобный график в машинном обучении, например, как в нейронной сети изменяется точность модели с увеличением эпохи обучения.

*Барный график (bar plot)*

Барный график представляет собой столбчатую диаграмму, которая показывает количественное отношение категориального признака. В matplotlib барный график называется bar, принимающий в качестве аргументов x — массив категорий и height — массив значений этой категории:

countries = data['country'].value\_counts().head(7)

plt.xlabel('Cтрана')

plt.ylabel('Количество вин')

plt.bar(x=countries.index, height=countries.values)

*Гистограмма (hist plot)*

Для отображения частотного показателя анализируемого атрибута используется гистограмма. Гистограммы похожи на барный график за исключением того, что вместо категориальных признаков берутся числовые, поэтому используются диапазоны значений. Кроме того, с помощью гистограмм можно построить плотность распределения.

plt.xlabel('points')

plt.ylabel('Вероятность')

plt.hist(x=data['points'], bins=40, density=True)

В качестве аргумента он принимает x, как массив значений, bins — количество значений, разбиваемых на диапазоны, denisty, равное True, представляет данные в виде плотности распределения.

*Ящик с усами (box plot)*

Ящик с усами, он же диаграмма размаха, можно сравнить с плотностью распределения. Он тоже показывает диапазон значений, лежащих около среднего. Помимо прочего, с его помощью можно определить выбросы — те данные, которые находятся далеко от среднего. Удаление выбросов является важным шагом подготовки модели машинного обучения, что очень важно для Data Scientist’a. В matplotlib ящик с усами называется boxplot:

plt.boxplot(x=data['points'])

Основным аргументом является x, принимающий массив анализируемых числовых значений.

Нижняя и верхняя границы соответствуют 25% и 75% квартилям, соответственно; горизонтальная черта внутри ящика показывает среднее значение; а концы “усов” определяются, как края статистически значимой выборки.

Методы визуализации позволяют наиболее полно понять данные сквозь огромное количество цифр.

1. **Задача выбора модели. Оценка эффективности, валидационный набор.**

Обычно рекомендуется начинать с простых, интерпретируемых моделей, таких как линейная регрессия, и если результаты будут неудовлетворительными, то переходить к более сложным, но обычно более точным методам.

Линейная регрессия. Метод k-ближайших соседей. «Случайный лес». Градиентный бустинг. Метод опорных векторов.

Один из важнейших этапов подготовки данных – сокращение числа признаков для устранения избыточной информации. Если модель имеет встроенные механизмы оценки важности признаков, то можно понять, от каких признаков зависит выбранный. Более того, выбор модели позволяет искать определённые зависимости. Например,

• линейная регрессия – линейные,

• обобщённая линейная – полиномиальные и т.п.,

• ансамбль пней – несложные нелинейные и т.п.

Процесс выбора модели в машинном обучении — это поиск компромисса между точностью решающего правила на обучении и его «надежностью» на произвольных объектах генеральной совокупности. Под задачей выбора модели будем понимать проблему автоматической настройки всех структурных параметров для данного алгоритма машинного обучения. Настройку структурных параметров нельзя проводить, минимизируя ошибку на обучении, т.к. эти параметры по смыслу как раз должны ограничивать допустимые решающие правила, чтобы избежать эффекта переобучения

Вероятностная модель порождения данных предполагает, что выборка из генеральной совокупности формируется случайным образом. Если все ее элементы одинаково случайно и независимо друг от друга распределены по исходному множеству (генеральной совокупности), выборка называется простой. Простая выборка является математической моделью серии независимых опытов и, как правило, используется для машинного обучения. При этом для каждого этапа Machine Learning необходим свой набор данных:

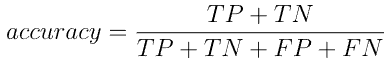
• для непосредственного обучения модели нужна обучающая выборка (training sample), по которой производится настройка (оптимизация параметров) алгоритма;

• для оценки качества модели используется тестовая (контрольная) выборка (test sample), которая, в идеальном случае, не должна зависеть от обучающей;

• для выбора наилучшей модели машинного обучения понадобится проверочная (валидационная) выборка (validation sample), которая также не должна пересекаться с обучающей.

Главное в формировании выборок – ни в коем случае не объединять обучающий датасет и с оценочными (тестовым и валидационным), поскольку это грозит переобучением модели Machine Learning.

Accuracy. Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy — доля правильных ответов алгоритма:



Эта метрика бесполезна в задачах с неравными классами, и это легко показать на примере. Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является AUC-ROC (или ROC AUC) — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve ). Данная кривая представляет из себя линию от (0,0) до (1,1) в координатах True Positive Rate (TPR) и False Positive Rate (FPR):

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

F-мера представляет собой гармоническое среднее между точностью и полнотой. Она стремится к нулю, если точность или полнота стремится к нулю.

Изображение выглядит как текст

Автоматически созданное описание

1. **Кривые обучения для диагностики моделей машинного обучения.**

Кривая обучения — это кривая эффективности обучения модели с учетом опыта или времени. Это широко используемый диагностический инструмент в машинном обучении для постепенного изучения алгоритмов на основе наборов обучающих данных. Модель может быть оценена на наборе обучающих данных и наборе данных проверки после каждого обновления обучения, а также может создать график производительности теста, чтобы показать кривую обучения.

Кривая обучения — это зависимость эффективности модели от размера обучающей выборки. Для построения кривых обучения модель обучают много раз, каждый раз с другим размером обучающей выборки. При малых объемах обучающая эффективность будет очень большой, а тестовая - очень маленькой. При увеличении объема обучающей выборки они будут сходиться, но обычно тестовая эффективность всегда ниже обучающей. Кривые обучения позволяют увидеть, как быстро модель учится, хватает ли ей данных, а также обнаруживать пере- и недообучение.

При недообучении тестовая и обучающая эффективности будут достаточно близкими, но недостаточными. При переобучении тестовая и обучающая эффективности будут сильно различаться - тестовая будет значительно ниже. Пере- и недообучение — это относительные понятия. Более простые модели склонны к недообучению, более сложные - к переобучению. Диагностика пере- и недообучения очень важна, так как для повышения эффективности предпринимаются противоположные меры. Для построения можно использовать функцию ошибки, метрику эффективности или метрику ошибки, важна только динамика этих показателей. Диагностика моделей машинного обучения — это не точная наука, здесь нужно принимать в расчет и задачу, и выбор признаков, и многие другие факторы.

1. **Регуляризация моделей машинного обучения. Назначение, виды, формализация.**

Регуляризация — это возможность добавить дополнительное ограничение в условиях решения задачи, чтобы найти решение задачи, если она поставлена с некорректными условиями. Регуляризация используется в машинном обучении, в статистике и в теории обратных задач.

## Регуляризация в машинном обучении:

В машинном обучении есть две проблемы, которые мешают качественному обучению модели:

1. Шум. Это те данные, которые косвенно связаны с данными для обучения. Они не являются важными. Их не нужно и не хочется учитывать в конечном результате. Это ненужный признак, который никак не влияет на расчеты, но все равно там присутствует.
2. Переобучение. Это процесс, при котором модель слишком интенсивно обучается. В конечном счете получается, что, как только вы в эту модель принесете новые данные для анализа — она покажет низкую эффективность.

## L1 и L2 регуляризация:

Регуляризация, может быть, двух основных видов:

1. L1-регуляризация — она же «манхэттенское расстояние» или «регрессия лассо». Ее идея заключается в том, чтобы сводить набор правил на наиболее важных функциях, которые влияют на конечный результат. Этот способ выглядит как способ выбора признаков. Выражается формулой: L1 = Σ(yi — y(ti))2 + λΣ|ai|.
2. L2-регуляризация — она же регуляризация Тихонова или «регрессия хребта». Этот вид регуляризации несколько похож на первый вид. По крайней мере, они выполняют одни и те же функции. Однако основная направленность деятельности этого метода — агрессивное применение штрафов. Практически получается, что этот метод не подходит для выбора признаков функции. В этом его основное отличие от L1. Выражается формулой: L2 = Σ(yi — y(ti))2 + λΣai2.
3. **Полиномиальные модели машинного обучения.**

Добавление полиномиальных признаков возможно как к регрессионным, так и к классификационным моделям.

Полиномиальная регрессия позволяет охватывать нелинейные зависимости атрибутов и целевой переменной. Полиномиальная классификация позволяет очерчивать нелинейные границы принятия решений.

Здесь и далее: атрибуты - характеристики объектов, данные в датасете; признаки - компоненты вектора, подающегося на вход модели машинного обучения.

Полиномиальные модели универсальны, но очень дороги при высоких порядках полинома. Полиномиальная регрессия – это алгоритм машинного обучения, который используется для обучения линейной модели на нелинейных данных. Порядок полиномиальной регрессии подбирается в качестве компромисса между качеством получаемой регрессии, и вычислительной сложностью. Ведь чем выше порядок полинома, тем более сложные зависимости он может аппроксимировать. И вообще, чем выше степень полинома, тем меньше будет ошибка при прочих равных. Если степень полинома на единицу меньше количества точек - ошибка будет нулевая. Но одновременно с этим, чем выше степень полинома, тем больше в модели параметров, тем она сложнее и занимает больше времени на обучение.

Полиномиальная регрессия может аппроксимировать любую функцию, нужно только подобрать степень полинома.

Чем больше степень полиномиальной регрессии, тем она сложнее и универсальнее, но вычислительно сложнее.

1. **Основные виды преобразования данных для подготовки к машинному обучению.**

Предварительная обработка и очистка данных должны проводиться до того, как набор данных будет использоваться для обучения модели. Необработанные данные зачастую искажены и ненадежны, и в них могут быть пропущены значения. Использование таких данных при моделировании может приводить к неверным результатам. Эти задачи являются частью процесса обработки и анализа данных группы и обычно подразумевают первоначальное изучение набора данных, используемого для определения и планирования необходимой предварительной обработки.

Главные задачи предварительной обработки данных:

·   Очистка данных: заполнение отсутствующих значений, обнаружение и удаление шумных данных и выбросов.

·   Преобразование данных: нормализация данных для уменьшения размеров и шума.

·   Сокращение данных. Примеры записей или атрибутов данных для упрощения обработки данных.

·   Дискретизация данных. Преобразуйте непрерывные атрибуты в категориальные атрибуты, чтобы упростить использование с определенными методами машинного обучения.

·   Очистка текста — удаление внедренных символов, которые могут нарушать выравнивание данных, например внедренных символов табуляции в файле с разделителем-табуляцией, внедренных новых линий, которые могут, например, разбивать записи.

Подготовительная обработка данных включает следующие шаги:

* удаление или заполнение отсутствующих значений;
* заполнение численных признаков - с помощью медианы(column.median())
* заполнение категориальных признаков - с помощью моды(column.mode())
* приведение всех признаков к бинарной либо числовой шкале (LabelEncoder, OneHotEncoder, dummy-признаки);
* удаление несущественных либо избыточных признаков
* нормализация и решкалирование (StandardScaler, normalize)
* удаление аномалий (выбросов)

1. **Задача выбора признаков в машинном обучении.**

Часто наборы данных, с которыми приходится работать, содержат большое количество признаков, число которых может достигать нескольких сотен и даже тысяч. При построении модели машинного обучения не всегда понятно, какие из признаков действительно для неё важны (т.е. имеют связь с целевой переменной), а какие являются избыточными (или шумовыми). Удаление избыточных признаков позволяет лучше понять данные, а также сократить время настройки модели, улучшить её точность и облегчить интерпретируемость. Иногда эта задача и вовсе может быть самой значимой, например, нахождение оптимального набора признаков может помочь расшифровать механизмы, лежащие в основе исследуемой проблемы. Это может быть полезным для разработки различных методик, например, банковского скоринга, поиска фрода или медицинских диагностических тестов. Методы отбора признаков обычно делят на 3 категории: фильтры (filter methods), встроенные методы (embedded methods) и обёртки (wrapper methods).

1)**Методы фильтрации**. Методы фильтрации применяются до обучения модели и, как правило, имеют низкую стоимость вычислений. К ним можно отнести визуальный анализ оценку признаков с помощью какого-нибудь статистического критерия (дисперсии, корреляции, X2 и др.) и экспертную оценку (удаление признаков, которые не подходят по смыслу, или признаков с некорректными значениями).

Простейшим способом оценки пригодности признаков является разведочный анализ данных (например, с библиотекой [pandas-profiling](https://newtechaudit.ru/uskoryaem-analiz-dannyh-c-pomoshhyu-pandas-profiling/)). Эту задачу можно автоматизировать с помощью библиотеки [feature-selector](https://github.com/WillKoehrsen/feature-selector), которая отбирает признаки по следующим параметрам:

* Количество пропущенных значений (удаляются признаки, у которых процент пропущенных значений больше порогового).
* Коэффициент корреляции (удаляются признаки, у которых коэффициент корреляции больше порогового).
* Вариативность (удаляются признаки, состоящие из одного значения).
* Оценка важности признаков с помощью lightgbm (удаляются признаки, имеющие низкую важность в модели lightgbm. Следует применять только если lightgbm имеет хорошую точность.)

2)**Встроенные методы**. Встроенные методы выполняют отбор признаков во время обучения модели, оптимизируя их набор для достижения лучшей точности. К этим методам можно отнести регуляризацию в линейных моделях (обычно L1) и расчёт важности признаков в алгоритмах с деревьями. Отметим, что для линейных моделей требуется масштабирование и нормализация данных.

3)**Методы обертки (wrapper methods)**. Особенность этих методов – поиск всех возможных подмножеств признаков и оценка их качества путем “прогонки” через модель. Процесс выбора функции основан на конкретном алгоритме машинного обучения, который мы используем. Он следует подходу жадного поиска, оценивая все возможные комбинации функций по определенному критерию. Методы оболочки обычно обеспечивают лучшую точность прогнозирования чем методы фильтрации.