Артем Груздев



	CUSTOMER LIFETIME VALUE	COVERAGE	EDUCATION	EMPLOYMENT STATUS	GENDER	INCOME	MONTHLY PREMIUM AUTO
0	2763.519279	Basic	Bachelor	Employed	F	56274.0	NaN
1	NaN	NaN	Bachelor	Unemployed	F	0.0	NaN
2	NaN	NaN	NaN	Employed	F	48767.0	108.0
3	7645.861827	Basic	Bachelor	NaN	NaN	0.0	106.0
4	2813.692575	Basic	Bachelor	NaN	M	43836.0	73.0

Предварительная подготовка данных в Python

Том 1. Инструменты и валидация

ИЦ «Гевисста»

ДМК ПРЕСС ИЗДАТЕЛЬСТВО

УДК 004.04Python
ББК 32.372
Г90

Груздев А. В.
Г90 Предварительная подготовка данных в Python: Том 1. Инструменты и валидация. – М.: ДМК Пресс, 2023. – 816 с.: ил.

ISBN 978-5-93700-156-6

В двухтомнике представлены материалы по применению классических методов машинного обучения в различных промышленных задачах. Первый том посвящен инструментам Python – основным библиотекам, классам и функциям, необходимым для предварительной подготовки данных, построения моделей машинного обучения, выполнения различных стратегий валидации. В конце первого тома разбираются задачи с собеседований по SQL, Python, математической статистике и теории вероятностей.

Издание рассчитано на специалистов по анализу данных, а также может быть полезно широкому кругу специалистов, интересующихся машинным обучением.

УДК 004.04Python
ББК 32.372

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-5-93700-156-6

© Груздев А. В., 2023
© Оформление, издание, перевод, ДМК Пресс, 2023

Оглавление

Введение 10

ЧАСТЬ 1. НЕМНОГО МАТЕМАТИКИ 11

1.1. Функция 11
1.2. Производная 12
1.3. Дифференцирование сложных функций 15
1.4. Частная производная 16
1.5. Градиент 17
1.6. Функция потерь и градиентный спуск 18

Часть 2. Инструменты 23

1. Введение 23

1.1. Структуры данных 23
1.1.1. Кортеж (tuple) 23
1.1.2. Список (list) 24
1.1.3. Словарь (dictionary) 27
1.1.4. Множество (set) 31
1.2. Функция 34
1.3. Полезные встроенные функции 35
1.3.1. Функция enumerate() 35
1.3.2. Функция sorted() 36
1.3.3. Функция zip() 36
1.4. Класс 38
1.5. Знакомство с Anaconda 43

2. IPython и Jupyter Notebook 44

3. NumPy 50

3.1. Создание массивов NumPy 50
3.2. Обращение к элементам массива 55
3.3. Получение краткой информации о массиве 57
3.4. Изменение формы массива 58
3.5. Конкатенация массивов 61
3.6. Функции математических операций, знакомство с правилами транслирования 65
3.7. Обработка пропусков 70
3.8. Функция np.linspace() 72
3.9. Функция np.logspace() 74

3.10. Функция np.digitize()....75
3.11. Функция np.searchsorted()....76
3.12. Функция np.bincount()....78
3.13. Функция np.apply_along_axis()....79
3.14. Функция np.insert()....80
3.15. Функция np.repeat()....81
3.16. Функция np.unique()....82
3.17. Функция np.take_along_axis()....84
3.18. Функция np.array_split()....86

4. Библиотеки Numba, datatable, bottleneck для ускорения вычислений....88

4.1. Numba....88
4.2. Datatable....94
4.3. Bottleneck....98

5. SciPy....99

6. pandas....111

6.1. Почему pandas?....111
6.2. Библиотека pandas построена на NumPy....111
6.3. pandas работает с табличными данными....111
6.4. Объекты DataFrame и Series....111
6.5. Задачи, выполняемые pandas....113
6.6. Кратко о типах данных....113
6.7. Представление пропусков....114
6.8. Какую версию pandas использовать?....115
6.9. Подробно знакомимся с типами данных....115
6.9.1. Типы данных для работы с числами и логическими значениями....115
6.9.2. Типы данных для работы со строками....126
6.10. Чтение данных....136
6.11. Получение общей информации о датафрейме....137
6.12. Изменение настроек вывода с помощью функции get_options()....139
6.13. Знакомство с индексаторами [], loc и iloc....140
6.14. Фильтрация данных....147
6.14.1. Одно условие....147
6.14.2. Несколько условий....148
6.14.3. Несколько условий в одном столбце....148
6.14.4. Использование метода .query()....149
6.15. Агрегирование данных....151
6.15.1. Группировка и агрегирование с помощью одного столбца....151
6.15.2. Группировка и агрегирование с помощью нескольких столбцов 153
6.15.3. Группировка с помощью сводных таблиц....156
6.16. Анализ частот с помощью таблиц сопряженности....166
6.17. Выполнение SQL-запросов в pandas....169

7. scikit-learn.....179

7.1. Основы работы с классами, строящими модели предварительной подготовки данных и модели машинного обучения.....179
7.2. Строим свой первый конвейер моделей.....198
7.3. Разбираемся с дилеммой смещения–дисперсии и знакомимся с бутстрепом.....210
7.4. Обработка пропусков с помощью классов MissingIndicator и SimpleImputer.....228
7.5. Выполнение дамми-кодирования с помощью класса OneHotEncoder и функции get_dummies(), знакомство с разреженными матрицами.....235
7.6. Автоматическое построение конвейеров моделей с помощью класса Pipeline.....246
7.7. Знакомство с классом ColumnTransformer.....250
7.8. Класс FeatureUnion.....263
7.9. Выполнение перекрестной проверки с помощью функции cross_val_score(), получение прогнозов перекрестной проверки с помощью функции cross_val_predict(), сохранение моделей перекрестной проверки с помощью функции cross_validate().....264
7.10. Виды перекрестной проверки для данных формата «один объект – одно наблюдение» (отсутствует ось времени).....273
7.10.1. Обычная нестратифицированная *k*-блочная перекрестная проверка с помощью класса KFold.....274
7.10.2. Обычная стратифицированная *k*-блочная перекрестная проверка с помощью класса StratifiedKFold.....281
7.10.3. Повторная нестратифицированная *k*-блочная перекрестная проверка с помощью класса RepeatedKFold.....283
7.10.4. Повторная стратифицированная *k*-блочная перекрестная проверка с помощью класса RepeatedStratifiedKFold.....286
7.10.5. ***k***-кратное случайное разбиение на обучающую и тестовую выборки (перекрестная проверка Монте-Карло).....288
7.10.6. Перекрестная проверка со случайными перестановками при разбиении с помощью класса ShuffleSplit.....294
7.10.7. Стратифицированная перекрестная проверка со случайными перестановками при разбиении с помощью класса StratifiedShuffleSplit.....296
7.10.8. Перекрестная проверка с исключением по одному с помощью класса LeaveOneOut.....297
7.10.9. Перекрестная проверка с исключением *p* наблюдений с помощью класса LeavePOut.....299
7.11. Виды перекрестной проверки для данных формата «один объект – несколько наблюдений» и стратифицированных данных (отсутствует ось времени).....301
7.11.1. Перекрестная проверка, учитывающая группы связанных наблюдений, с помощью классов GroupKFold.....301
7.11.2. Перекрестная проверка, учитывающая группы связанных наблюдений с исключением из обучения одной группы, с помощью класса LeaveOneGroupOut.....302

7.11.3. Перекрестная проверка, учитывающая группы связанных наблюдений с исключением из обучения ***p*** групп, с помощью класса LeavePGroupsOut 304
7.11.4. Перекрестная проверка, учитывающая группы связанных наблюдений и распределение классов, с помощью класса StratifiedGroupKFold 305
7.11.5. Перекрестная проверка со случайными перестановками при разбиении и учитывающая группы связанных наблюдений с помощью класса GroupShuffleSplit 307
7.12. Обычный и случайный поиск наилучших гиперпараметров по сетке с помощью классов GridSearchCV и RandomizedSearchCV 309
7.12.1. Обычный поиск оптимальных значений гиперпараметров моделей предварительной подготовки и модели машинного обучения 312
7.12.2. Обычный поиск оптимальных значений гиперпараметров моделей предварительной подготовки и модели машинного обучения с добавлением строки прогресса 318
7.12.3. Случайный поиск оптимальных значений гиперпараметров моделей предварительной подготовки и модели машинного обучения 320
7.12.4. Обычный поиск оптимальных значений гиперпараметров для CatBoost при обработке категориальных признаков «как есть» (заданы индексы категориальных признаков) 321
7.12.5. Отбор оптимальной модели предварительной подготовки данных в рамках отдельного трансформера 324
7.12.6. Отбор оптимального метода машинного обучения среди разных методов машинного обучения (перебор значений гиперпараметров с отдельной предобработкой данных под каждый метод машинного обучения) 329
7.13. Вложенная перекрестная проверка 335
7.14. Классы PowerTransformer, KBinsDiscretizer и FunctionTransformer 341
7.15. Написание собственных классов предварительной подготовки для применения в конвейере 350
7.16. Модификация классов библиотеки scikit-learn для работы с датафреймами 375
7.17. Полный цикл построения конвейера моделей в scikit-learn 381
7.17.1. Первая задача 381
7.17.2. Вторая задача 393
7.18. Калибровка модели 404
7.18.1. Актуальность калибровки 404
7.18.2. Функция calibration_curve() 406
7.18.3. Оценка Брайера 413
7.18.4. Оценка качества калибровки моделей до применения калибратора 415
7.18.5. Класс CalibratedClassifierCV 420
7.18.6. Оценка качества калибровки моделей после применения калибратора 421

7.18.7. Оценка качества калибровки моделей после применения калибратора с уже обученным классификатором....423
7.18.8. Калибровка на основе сплайнов....426
7.19. Полезные классы CountVectorizer и TfidfVectorizer для работы с текстом....436
7.20. Сравнение моделей, полученных в ходе поиска по сетке, с помощью статистических тестов....450
7.20.1. Простое сравнение всех построенных моделей....451
7.20.2. Сравнение двух моделей: частотный подход....454
7.20.3. Сравнение двух моделей: байесовский подход....458
7.20.4. Попарное сравнение всех моделей: частотный подход....463
7.20.5. Попарное сравнение всех моделей: байесовский подход....465
7.20.6. Итоговые выводы....467
7.21. Разбиение на обучающую, проверочную и тестовую выборки с учетом временной структуры для валидации временных рядов....468
7.22. Виды перекрестной проверки для данных формата «один объект – одно наблюдение» (присутствует ось времени)....521
7.22.1. Перекрестная проверка расширяющимся окном....525
7.22.2. Перекрестная проверка скользящим окном....542
7.22.3. Перерестная проверка расширяющимся/скользящим окном с гэпом....552
7.23. Перекрестная проверка для данных формата «один объект – несколько наблюдений» (присутствует ось времени)....563
7.24. Многоклассовая классификация: подходы «один против всех», «один против одного» и «коды, исправляющие ошибки»....567
7.24.1. Подход «один против остальных» или «один против всех» («one versus rest», «one versus all»)....568
7.24.2. Подход «один против одного» («one versus one»)....573
7.24.3. Подход «коды, исправляющие ошибки» («error-correcting output codes»)....592

ЧАСТЬ 3. ДРУГИЕ ПОЛЕЗНЫЕ БИБЛИОТЕКИ....602

1. Библиотеки визуализации matplotlib, seaborn и plotly....602

1.1. Matplotlib....602
1.2. Seaborn....621
1.3. Plotly....629

2. Библиотека прогнозирования временных рядов ETNA....634

2.1. Общее знакомство....634
2.2. Создание объекта TSDataset....641
2.3. Визуализация рядов объекта TSDataset....645
2.4. Получение сводки характеристик по объекту TSDataset....646
2.5. Модель наивного прогноза....647
2.6. Модель скользящего среднего....654

2.7. Модель сезонного скользящего среднего 658
2.8. Модель SARIMAX 662
2.9. Модель Хольта–Винтерса (модель тройного экспоненциального сглаживания, модель ETS) 671
2.10. Модель Prophet 677
2.11. Модель CatBoost 689
2.12. Модель линейной регрессии с регуляризацией «эластичная сеть» 709
2.13. Объединение процедуры построения модели, оценки качества и визуализации прогнозов в одной функции 714
2.14. Перекрестная проверка нескольких моделей 717

2.15. Ансамбли 722
2.16. Стекинг 724
2.17. Создание собственных классов для обучения моделей 725
2.18. Импутация пропусков 741
2.19. Работа с трендом и сезонностью 751
2.20. Обработка выбросов 766
2.21. Собираем все вместе 772
2.22. Модели нейронных сетей 787
2.23. Оптимизация гиперпараметров с помощью Optuna от разработчиков 789

Ответы на вопросы с собеседований 794

Часть 1

Немного математики

1.1. Функция

В жизни мы часто встречаемся с зависимостями между различными величинами. Для простоты возьмем зависимость между двумя величинами. Например, мы знаем, что площадь круга зависит от радиуса, площадь квадрата зависит от его стороны. Переменную x, значения которой выбираются произвольно, называют независимой переменной, а переменную y, значения которой определяются выбранными значениями x, называют зависимой переменной. Давайте взглянем на рисунок. На нем изображен график температуры воздуха в течение суток.

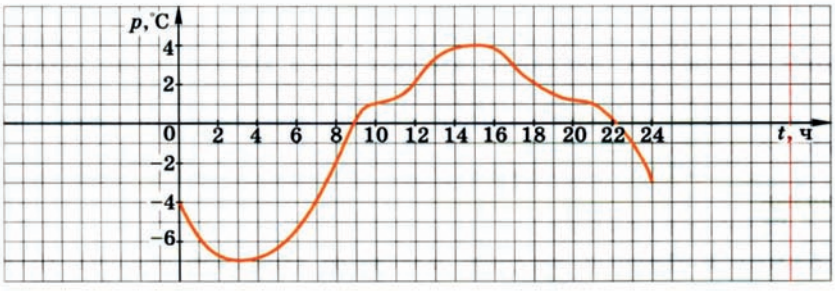


Рис. 1 График температуры воздуха в течение суток

С помощью этого графика для каждого момента времени t (в часах), где $0 \leqslant t \leqslant 24$, можно найти соответствующую температуру p (в градусах Цельсия). Например:

- если $t = 7$, то $p = -4$;
- если $t = 12$, то $p = 2$;
- если $t = 17$, то $p = 3$;
- если $t = 22$, то $p = 0$.

Здесь t является независимой переменной, а p – зависимой переменной. В рассмотренном примере каждому значению независимой переменной соответствует единственное значение зависимой переменной. Такую зависимость одной переменной от другой называют *функциональной зависимостью*, или *функцией*. Например, когда мы пишем $y = f(x)$ (читается «как y, равное f от x»),

мы как раз и имеем в виду эту идею зависимости: переменная *y* зависит от переменной *x* по определенному закону (предписанию, правилу). Закон этот обозначен буквой *f*.

Независимую переменную иначе называют *аргументом*, а о зависимой переменной говорят, что она является *функцией* от этого аргумента. Путь, пройденный автомобилем с постоянной скоростью, является функцией от времени движения. Например, если автомобиль движется с постоянной скоростью 60 км/ч, зависимость пути от времени можно задать формулой $s = 60t$, где s – пройденный путь (в километрах), t – время (в часах).

Значения зависимой переменной называют *значениями функции*. Все значения, которые принимает независимая переменная, образуют *область определения функции*. Значения функции в точке максимума (минимума) функции называются, соответственно, *максимумом* и *минимумом* функции. Минимальное или максимальное значения функции на заданном множестве называют экстремумом функции. Минимум и максимум функции может быть *локальным* и *глобальным*.



Рис. 2 Минимум и максимум функции

1.2. Производная

Теперь выясним, что такое *производная* функции. Для объяснения производной воспользуемся открытыми материалами Игоря Яковлева[1].

Представьте, вы едете на автомобиле и спидометр показывает 60 км/ч. Что это значит? Ответ простой: если автомобиль будет ехать так в течение часа, то он проедет 60 км.

Допустим, что автомобиль вовсе не собирается ехать так целый час. Например, водитель разгоняет автомобиль с места, давит на газ, в какой-то момент бросает взгляд на спидометр и видит стрелку на отметке 60 км/ч. В следующий момент стрелка уползет еще выше. Как же понимать, что в данный момент времени скорость равна 60 км/ч?

Давайте выясним это на примере. Предположим, что путь s, пройденный автомобилем, зависит от времени t следующим образом:

$$s(t) = t^2,$$

[1] *https://mathus.ru/math/der.pdf.*

где путь измеряется в метрах, а время – в секундах. То есть при $t = 0$ путь равен нулю, к моменту времени $t = 1$ пройденный путь равен $s(1) = 1$, к моменту времени $t = 2$ пройденный путь равен $s(2) = 4$, к моменту времени $t = 3$ пройденный путь равен $s(3) = 9$ и так далее.

Видно, что идет разгон – автомобиль набирает скорость с течением времени. Действительно: за первую секунду пройдено расстояние 1; за вторую секунду пройдено расстояние $s(2) - s(1) = 4 - 1 = 3$; за третью секунду пройдено расстояние $s(3) - s(2) = 9 - 4 = 5$, и далее по нарастающей.

А теперь вопрос. Пусть, например, через три секунды после начала движения наш водитель взглянул на спидометр. Что покажет стрелка? Иными словами, какова *мгновенная* скорость автомобиля в момент времени $t = 3$?

Просто поделить путь на время не получится: привычная формула $v = s/t$ работает только для *равномерного* движения (то есть когда стрелка спидометра застыла в некотором фиксированном положении). Но именно эта формула лежит в основе способа, позволяющего найти мгновенную скорость.

Идея способа такова. Отсчитаем от нашего момента $t = 3$ небольшой промежуток времени Δt, найдем путь Δs, пройденный автомобилем за этот промежуток, и поделим Δs на Δt. Чем меньше будет Δt, тем точнее мы приблизимся к искомой величине мгновенной скорости.

Давайте посмотрим, как эта идея реализуется. Возьмем для начала $\Delta t = 1$. Тогда

$$\Delta s = s(4) - s(3) = 4^2 - 3^2 = 16 - 9 = 7,$$

и для скорости (измеряется в м/с) получаем:

$$\frac{\Delta s}{\Delta t} = \frac{7}{1} = 7. \tag{1}$$

Будем уменьшать промежуток Δt. Берем $\Delta t = 0{,}1$:

$$\Delta s = s(3{,}1) - s(3) = 3{,}1^2 - 3^2 = 9{,}61 - 9 = 0{,}61,$$
$$\frac{\Delta s}{\Delta t} = \frac{0{,}61}{0{,}1} = 6{,}1. \tag{2}$$

Теперь берем $\Delta t = 0{,}01$:

$$\Delta s = s(3{,}01) - s(3) = 3{,}01^2 - 3^2 = 9{,}0601 - 9 = 0{,}0601,$$
$$\frac{\Delta s}{\Delta t} = \frac{0{,}601}{0{,}01} = 6{,}01. \tag{3}$$

Наконец, возьмем $\Delta t = 0{,}001$:

$$\Delta s = s(3{,}001) - s(3) = 3{,}001^2 - 3^2 = 9{,}006001 - 9 = 0{,}006001,$$
$$\frac{\Delta s}{\Delta t} = \frac{0{,}006001}{0{,}001} = 6{,}001. \tag{4}$$

Глядя на значения (1)–(4), мы понимаем, что величина $\Delta s/\Delta t$ приближается к числу 6. Это обозначает, что мгновенная скорость автомобиля в момент времени $t = 3$ составляет 6 м/с.

Таким образом, при безграничном уменьшении Δt путь Δs также стремится к нулю, но отношение $\Delta s/\Delta t$ стремится к некоторому пределу v, который и называется мгновенной скоростью в данный момент времени t:

$$v = \lim_{\Delta t \to 0} \frac{\Delta s}{\Delta t}. \tag{5}$$

Можно написать и так:

$$v(t) = \lim_{\Delta t \to 0} \frac{s(t + \Delta t) - s(t)}{\Delta t}. \tag{6}$$

Давайте вернемся к нашему примеру с $s(t) = t^2$ и проделаем в общем виде те выкладки, которые выше были выполнены с числами. Итак:

$$\Delta s = s(t + \Delta t) - s(t) = (t + \Delta t)^2 - t^2 = t^2 + 2t\Delta t + \Delta t^2 - t^2 = \Delta t(2t + \Delta t),$$

и для мгновенной скорости имеем:

$$v(t) = \lim_{\Delta t \to 0} \frac{\Delta s}{\Delta t} = \lim_{\Delta t \to 0} \frac{\Delta t(2t + \Delta t)}{\Delta t} = \lim_{\Delta t \to 0} (2t - \Delta t) = 2t. \tag{7}$$

В частности, при $t = 3$ формула (7) дает: $v(3) = 2 \times 3 = 6$, как и было получено выше. Скорость бывает не только у автомобиля. Мы можем говорить о скорости изменения чего угодно – например, физической величины или экономического показателя. И производная как раз и служит обобщением понятия мгновенной скорости на случай абстрактных математических функций.

Рассмотрим функцию $y = f(x)$. Напомним, что x называется аргументом данной функции. Отметим на оси X некоторое значение аргумента x, а на оси Y – соответствующее значение функции $f(x)$.

Дадим аргументу x некоторое *приращение*, обозначаемое Δx. Попадем в точку $x + \Delta x$. Обозначим ее на рисунке вместе с соответствующим значением функции $f(x + \Delta x)$. Величина $f(x + \Delta x) - f(x)$ называется *приращением функции*, которое отвечает данному приращению аргумента Δx.

Видите сходство с примером, когда мы вычисляли мгновенную скорость автомобиля? Приращение аргумента Δx есть абстрактный аналог промежутка времени Δt, а соответствующее приращение функции Δf – это аналог пути Δs, пройденного за время Δt. Производная – это в точности аналог мгновенной скорости.

Давайте дадим строгое определение производной. Производная $f'(x)$ функции $f(x)$ в точке x – это предел отношения приращения функции к приращению аргумента, когда приращение аргумента стремится к нулю:

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

Сравните с формулами (5) и (6). По сути, написано одно и то же. Можно сказать, что производная – это мгновенная скорость изменения функции.

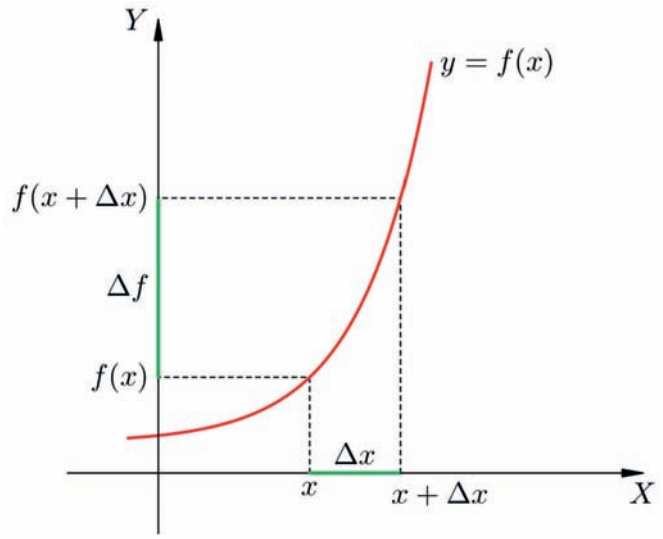


Рис. 3 Приращение аргумента и приращение функции

Для производной используются обозначения: $f'(x)$ (читается как «f штрих от x»), y' (читается как «y штрих»), $\frac{dy}{dx}$ (читается как «dy по dx»).

Функцию, имеющую конечную производную (в некоторой точке), называют дифференцируемой (в данной точке).

1.3. Дифференцирование сложных функций

Процесс вычисления производной называется *дифференцированием*. Производные простых функций можно легко вычислить с помощью формулы (7).

В машинном обучении вам часто придется дифференцировать степенную функцию – функцию вида $f(x) = x^a$. Формула производной степенной функции выглядит так:

$$(x^a)' = ax^{a-1}, a \in \mathbb{R}.$$

Производную степенной функции относят к табличным производным, которые нужно знать наизусть.

Еще чаще в машинном обучении нам придется находить производные сложных функций. Дифференцирование сложной функции происходит следующим образом. Сначала находим производную второй («внешней») функции и затем умножаем ее на производную первой («внутренней») функции. Эту процедуру мы еще называем правилом цепочки. Зная небольшое число табличных производных и располагая правилами дифференцирования, можно вычислять производные огромного количества функций.

Например, нужно вычислить производную функции $y = (\theta - 5)^2$.

Функция $(x - 5)^2$ является композицией $f(g(x))$ двух функций: $f(u) = u^2$ и $u = g(x) = x - 5$.

Применяем правило цепочки $(f(g(x)))' = \frac{d}{du} f(u) \cdot \frac{d}{dx} g(x)$:

$$((x-5)^2)' = \frac{d}{du}(u^2)\frac{d}{dx}(x-5).$$

Применяем правило дифференцирования степенной функции:

$$\frac{d}{du}(u^n) = n \cdot u^{n-1} \text{ с } n = 2:$$

$$\frac{d}{du}(u^2)\frac{d}{dx}(x-5) = (2u^{2-1})\frac{d}{dx}(x-5) = 2u\frac{d}{dx}(x-5).$$

$$2u\frac{d}{dx}(x-5) = 2(x-5)\frac{d}{dx}(x-5)$$

Производная от суммы (разности) равна сумме (разности) производных функций:

$$2(x-5)\frac{d}{dx}(x-5) = 2(x-5)\left(\frac{d}{dx}(x) - \frac{d}{dx}(5)\right) = (2x-10)\left(\frac{d}{dx}(x) - \frac{d}{dx}(5)\right).$$

Применяем правило дифференцирования степенной функции:

$$\frac{d}{dx}(x^n) = n \cdot x^{n-1} \text{ с } n = 1, \text{ другими словами, } \frac{d}{dx}(x) = 1:$$

$$(2x-10)\left(\frac{d}{dx}(x) - \frac{d}{dx}(5)\right) = (2x-10)\left((1) - \frac{d}{dx}(5)\right).$$

Производная константы равна 0:

$$(2x-10)\left(1 - \frac{d}{dx}(5)\right) = (2x-10)(1-(0)).$$

Таким образом, $((x-5)^2)' = 2x - 10$.

1.4. Частная производная

Встречаются зависимости не только от одной, но и от нескольких переменных. Например, площадь прямоугольника можно записать как $S = xy$. Значения S будут определяться совокупностью значений x и y. Это уже будет функция двух переменных. Объем V прямоугольного параллелепипеда с ребрами x, y и z выражается формулой $V = xyz$, т.е. значения V зависят от трех переменных. Это уже будет функция трех переменных.

Обобщением понятия производной на случай функции нескольких переменных будет *частная производная*.

Частная производная – это предел отношения приращения функции по выбранной переменной к приращению этой переменной, при стремлении этого приращения к нулю.

Возьмем функцию двух переменных $f(x, y)$. Часто бывает важно знать, с какой скоростью меняются значения функции, когда мы перемещаемся по плоскости xy.

Частная производная по x от функции $f(x, y)$ обозначается $\frac{df}{dx}$. Таким образом, по определению, частной производной по x от функции $f(x, y)$ будет предел отношения частного приращения $\Delta_x f$ по x к приращению Δx при стремлении Δx к нулю:

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{\Delta_x f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}.$$

Частная производная по y от функции обозначается $\frac{df}{dy}$. Частной производной по y от функции $f(x, y)$ будет предел отношения частного приращения $\Delta y f$ по y к приращению Δy при стремлении Δy к нулю:

$$\frac{\partial f}{\partial y} = \lim_{\Delta y \to 0} \frac{\Delta_y f}{\Delta y} = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y}.$$

С помощью этих частных производных мы можем вычислить скорость изменения функции вдоль координатных осей x и y.

При вычислении частной производной $\frac{df}{dx}$ мы воспринимаем переменную y как константу. Аналогично, вычисляя $\frac{df}{dy}$, мы считаем x константой. Из этого ясно, что правила вычисления частных производных совпадают с правилами, указанными для функций одной переменной, и только требуется каждый раз помнить, по какой переменной ищется производная.

Теперь перейдем к градиенту.

1.5. Градиент

Градиентом функции многих переменных в данной точке называется вектор, координаты которого равны частным производным по соответствующим аргументам, вычисленным в данной точке.

Рассмотрим функцию двух переменных $f(x, y)$. Градиентом функции $f(x, y)$ будет вектор $\text{grad}f = \nabla f = \left(\frac{df}{dx}, \frac{df}{dy} \right)$. Производные $\frac{df}{dx}$ и $\frac{df}{dy}$ вычисляются в каждой точке (x, y). Таким образом, градиент задан в каждой точке и будет меняться от точки к точке.

Чем интересен градиент? Он интересен тем, что в данной точке он будет показывать нам направление наибольшего возрастания функции. Например, если взять высоту поверхности Земли над уровнем моря, то ее градиент в каж-

дой точке поверхности будет показывать направление «в горку». Модуль градиента совпадает с максимальной скоростью возрастания функции в данной точке. Когда у нас есть две переменные, то наш двухкомпонентный градиент может указать направление наибольшего возрастания функции на плоскости. Аналогично с тремя переменными – градиент может указать направление наибольшего возрастания функции в трехмерном пространстве.

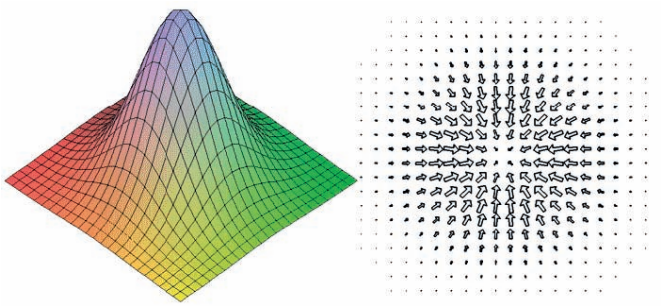


Рис. 4 Визуальная интерпретация градиента. Операция градиента преобразует холм (слева), если смотреть на него сверху, в поле векторов (справа). Видно, что векторы направлены «в горку» и чем длиннее, тем круче наклон. Источник: Википедия

1.6. Функция потерь и градиентный спуск

Функция, для которой мы будем искать экстремум, в машинном обучении называется **целевой функцией (objective function)**. Задача по нахождению экстремума функции называется задачей оптимизации. Если речь идет о поиске минимума, то употребляют термины **функция стоимости (cost function)**, **функция потерь (loss function)**, **функция ошибок (error function)**.

Если функция дифференцируема, то найти точки, подозрительные на экстремум, можно с помощью необходимого условия экстремума: все частные производные должны равняться нулю, а значит, вектор градиента – нулевому вектору. Но не всегда задачу можно решать аналитически. В таком случае используется численная оптимизация. Наиболее простым в реализации из всех методов численной оптимизации является метод градиентного спуска, тесно связанный с понятием градиента.

Градиентный спуск – итерационный метод. Основная идея градиентного спуска состоит в том, чтобы двигаться к минимуму в направлении наиболее быстрого убывания функции потерь, которое определяется антиградиентом. В ходе градиентного спуска мы итеративно применяем следующее правило обновления:

$$w^t = w^{t-1} - \eta \nabla Q(w^{t-1}),$$

где $\nabla Q(w^{t-1})$ – это градиент функции потерь, которую мы пытаемся минимизировать, а η — размер шага градиентного спуска, называемый *темпом обучения*, или *скоростью обучения* (*learning rate*).

Мы выбираем каким-либо способом начальную точку, вычисляем в ней градиент рассматриваемой функции и делаем небольшой шаг в обратном, антиградиентном направлении. В результате приходим в точку, в которой значение функции будет меньше первоначального. В новой точке повторяем процедуру: снова вычисляем градиент функции и делаем шаг в обратном направлении. Продолжая этот процесс, мы будем двигаться в сторону убывания функции. Можно представить это как движение вниз по холму – сделав шаг вниз, текущая позиция будет ниже, чем предыдущая. Таким образом, на каждом следующем шаге высота будет как минимум не увеличиваться. Поэтому этот метод и называется спуском. Важно, чтобы наша функция была выпуклой и гладкой. Гладкой или непрерывно дифференцируемой функцией называют функцию, имеющую непрерывную производную на всем множестве определения. Выпуклой (или выпуклой вниз) функцией называют функцию, для которой отрезок между любыми двумя точками ее графика в векторном пространстве лежит не ниже соответствующей дуги графика. Выпуклость гарантирует существование лишь одного минимума, а гладкость – существование вектора градиента в каждой точке.



Рис. 5 Градиентный спуск

Допустим, необходимо минимизировать функцию вида $y = (\theta - 5)^2$, т.е. нам надо найти, при каком значении θ наша функция принимает минимальное значение. Нужно выполнить следующие действия:

1) необходима производная по θ: $\frac{dy}{d\theta} = 2(\theta - 5) = 2\theta - 10$;

2) установим начальное значение $\theta = 0$;

3) установим скорость обучения γ равной 0,2;

4) 20 раз подряд применим формулу $\theta^t = \theta^{t-1} - \gamma \frac{dy}{d\theta}$. У нас только один неизвестный параметр, поэтому будет одна формула.

```
n_iter = 20 # количество итераций
learning_rate = 0.2  # скорость обучения
def func(x):
    return (x - 5) ** 2
def func_derivative(x):
    return 2 * (x - 5)
previous_x, current_x = 0, 0
```

```
for i in range(n_iter):
    current_x = previous_x - learning_rate * func_derivative(previous_x)
    previous_x = current_x
    print("theta", format(current_x, ".6f"),
          "function value=", format(func(current_x), ".6f"),
          "derivative=", format(func_derivative(current_x), ".6f"))
```

```
theta 2.000000 function value= 9.000000 derivative= -6.000000
theta 3.200000 function value= 3.240000 derivative= -3.600000
theta 3.920000 function value= 1.166400 derivative= -2.160000
theta 4.352000 function value= 0.419904 derivative= -1.296000
theta 4.611200 function value= 0.151165 derivative= -0.777600
theta 4.766720 function value= 0.054420 derivative= -0.466560
theta 4.860032 function value= 0.019591 derivative= -0.279936
theta 4.916019 function value= 0.007053 derivative= -0.167962
theta 4.949612 function value= 0.002539 derivative= -0.100777
theta 4.969767 function value= 0.000914 derivative= -0.060466
theta 4.981860 function value= 0.000329 derivative= -0.036280
theta 4.989116 function value= 0.000118 derivative= -0.021768
theta 4.993470 function value= 0.000043 derivative= -0.013061
theta 4.996082 function value= 0.000015 derivative= -0.007836
theta 4.997649 function value= 0.000006 derivative= -0.004702
theta 4.998589 function value= 0.000002 derivative= -0.002821
theta 4.999154 function value= 0.000001 derivative= -0.001693
theta 4.999492 function value= 0.000000 derivative= -0.001016
theta 4.999695 function value= 0.000000 derivative= -0.000609
theta 4.999817 function value= 0.000000 derivative= -0.000366
```

В итоге находим значение параметра θ, при котором функция $y = (\theta - 5)^2$ принимает минимальное значение.

Очень важно при использовании метода градиентного спуска правильно подбирать шаг. Каких-либо конкретных правил подбора шага не существует, выбор шага – это искусство, но существует несколько полезных закономерностей. Если длина шага слишком мала, то метод будет не спеша, но верно шагать в сторону минимума. Если же взять размер шага очень большим, появляется риск, что метод будет перепрыгивать через минимум. Более того, есть риск того, что градиентный спуск не сойдется.

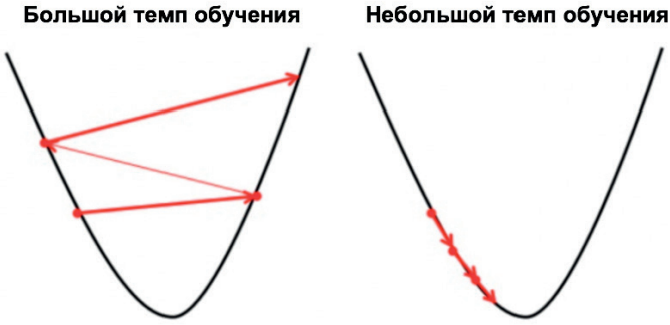


Рис. 6 Большой и маленький темпы обучения

В методе обычного градиентного спуска градиент вычисляется по всем наблюдениям обучающей выборки, формулу обновления для обычного градиентного спуска еще можно записать так:

$$w^t = w^{t-1} - \eta \nabla Q(w^{t-1}, X),$$

где X – обучающая выборка.

В этом и состоит основной недостаток метода градиентного спуска – в случае большой выборки даже одна итерация метода градиентного спуска будет осуществляться долго.

В методе стохастического градиентного спуска градиент функции качества вычисляется только на одном случайно выбранном объекте обучающей выборки:

$$w^t = w^{t-1} - \eta \nabla Q(w^{t-1}, \{x_i\}),$$

где $\{x_i\}$ – случайно отобранное наблюдение обучающей выборки.

Это позволяет обойти вышеупомянутый недостаток обычного градиентного спуска.

Показательно посмотреть на графики сходимости градиентного спуска и стохастического градиентного спуска. В обычном градиентном спуске на каждом шаге уменьшается суммарная ошибка на всех элементах обучающей выборки. График в таком случае обычно получается монотонным. В стохастическом градиентном спуске параметры меняются таким образом, чтобы максимально уменьшить ошибку для одного случайно выбранного объекта. Это приводит к тому, что график выглядит пилообразным, то есть на каждой конкретной итерации полная ошибка может как увеличиваться, так и уменьшаться. Но в итоге с ростом номера итерации значение функции уменьшается.

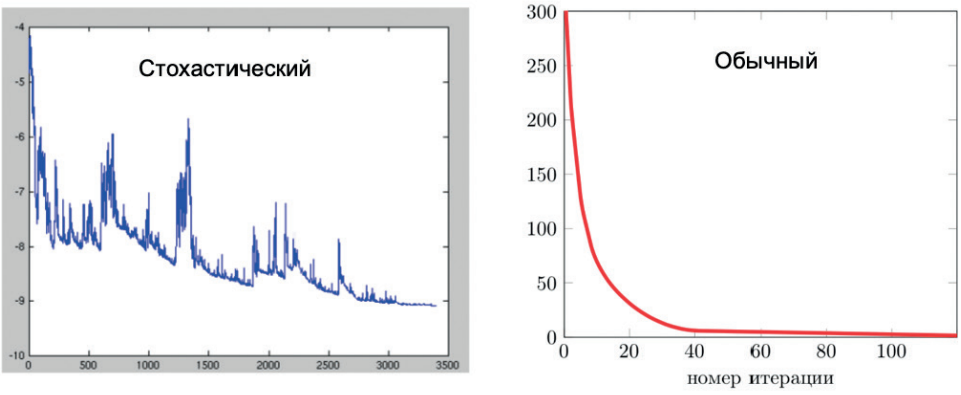


Рис. 7 Зависимость функции потерь от номера итерации в обычном градиентном спуске и стохастическом градиентном спуске

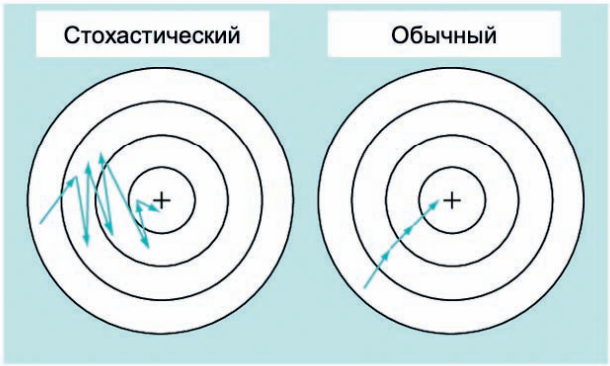


Рис. 8 Процесс обучения в обычном градиентном спуске и стохастическом градиентном спуске

Задача с собеседования (математика)

1. Идут три охотника на охоту. У одного – 3 стакана крупы, у другого – 5 стаканов крупы, у третьего – 8 патронов. Сварили кашу, и все поели поровну. Третий охотник решил отблагодарить двух и отдать им все патроны. Как поделить патроны справедливо?

Часть 2

Инструменты

1. ВВЕДЕНИЕ

Python стал одним из самых популярных языков, применяемых в машинном обучении для выполнения научных и коммерческих проектов. На момент написания книги экосистема Python предлагает широчайшие возможности для предварительной обработки данных и построения моделей машинного обучения.

Давайте вспомним основные структуры данных в Python. Нам понадобятся библиотека NumPy и модуль collections.

```
# импортируем библиотеку NumPy
# и модуль collections
import numpy as np
import collections
```

1.1. Структуры данных

1.1.1. Кортеж (tuple)

Кортеж – это *неизменяемая* последовательность с упорядоченными элементами. Мы можем создать кортеж, записав последовательность значений через запятую.

```
# создаем кортеж
tup = 4, 5, 6
# смотрим кортеж
tup
```

```
(4, 5, 6)
```

Мы можем создать кортеж кортежей.

```
# создаем кортеж кортежей
nested_tup = (4, 5, 6), (7, 8)
# смотрим кортеж кортежей
nested_tup
```

```
((4, 5, 6), (7, 8))
```

Любую последовательность или итератор можно преобразовать в кортеж с помощью функции `tuple()`.

```
# получаем кортеж
tup = tuple([3, 5, 2])
tup

(3, 5, 2)

# создаем кортеж
tup = tuple('кортеж')
tup

('к', 'о', 'р', 'т', 'е', 'ж')
```

К элементам кортежа, как и к элементам большинства других последовательностей Python, можно обращаться с помощью квадратных скобок []. Нумерация элементов последовательностей в Python начинается с нуля.

```
# обращаемся к первому элементу кортежа
tup[0]

'к'
```

1.1.2. Список (list)

Список – это *изменяемая* последовательность с упорядоченными элементами, т.е. в отличие от кортежей содержимое списков можно модифицировать. Список определяется с помощью квадратных скобок `[]` или функции `list()`.

```
# создаем список
lst = [3, 4, 5]
lst

[3, 4, 5]

# создаем список
lst = list('список')
lst

['с', 'п', 'и', 'с', 'о', 'к']
```

Для добавления элемента в конец списка служит метод `.append()`.

```
# создаем список
lst = ['a', 'b', 'c']
# добавляем элемент в конец списка
lst.append('d')
lst

['a', 'b', 'c', 'd']
```

Метод `.insert()` позволяет вставить элемент в указанную позицию списка. Мы должны указать позицию в списке и собственно сам элемент.

```
# создаем список
lst = [2, 4, 6, 9]
# вставляем 7 третьим элементом (нумерация с 0)
lst.insert(2, 7)
lst
```

```
[2, 4, 7, 6, 9]
```

Методом, обратным к `.insert()`, является `.pop()`, он удаляет из списка элемент, находившийся в указанной позиции, и возвращает его.

```
# удаляем третий элемент из списка
lst.pop(2)
7

# смотрим результат
lst
```

```
[2, 4, 6, 9]
```

Метод `.remove()` находит и удаляет из списка первый элемент с указанным значением.

```
# создаем список
lst = ['house', 'pop', 'techno', 'pop', 'trance']
# удаляем первый элемент с указанным значением
lst.remove('pop')
lst
```

```
['house', 'techno', 'pop', 'trance']
```

Для проверки, содержит ли список некоторое значение, используется оператор `in`.

```
# проверим, входит ли 'hardcore' в список
'hardcore' in lst
False
```

Проверка вхождения значения в случае списка занимает гораздо больше времени, чем в случае словаря или множества, потому что Python должен просматривать список от начала до конца, а это требует линейного времени, тогда как поиск в других структурах занимает постоянное время.

Операция сложения списков конкатенирует списки (то же самое будет с кортежами).

```
# складываем списки
['a', 'b'] + ['c', 'd']
```

```
['a', 'b', 'c', 'd']
```

```
# складываем кортежи
('a', 'b') + ('c', 'd')
```

```
('a', 'b', 'c', 'd')
```

Для добавления в конец имеющегося списка нескольких элементов можно использовать метод `.extend()`.

```
# создаем список
lst = ['house', 'pop', 'techno', 'pop', 'trance']
# добавляем в конец списка несколько элементов
lst.extend(['hardcore', 'speedcore'])
lst
```

```
['house', 'pop', 'techno', 'pop', 'trance', 'hardcore', 'speedcore']
```

С помощью метода `.clear()` можно очистить список.

```
# очищаем список
lst.clear()
lst
```

```
[]
```

Список можно отсортировать на месте (без создания нового объекта), вызвав его метод `.sort()`.

```
# создаем список
a = [7, 2, 5, 1, 3]
# сортируем
a.sort()
a
[1, 2, 3, 5, 7]
```

Метод `.count()` позволяет вычислить количество элементов с заданным значением.

```
# создаем список
lst = ['house', 'pop', 'techno', 'pop', 'trance']
# смотрим количество элементов с заданным значением
lst.count('pop')
```

```
2
```

Часто возникает потребность вычислить абсолютные частоты элементов списка. Это можно сделать с помощью функции библиотеки NumPy `np.unique()`. Сначала с ее помощью получаем массив уникальных значений и массив абсолютных частот, с помощью функции `zip()` «сшиваем» их и с помощью функции `dict()` записываем результат в словарь.

```
# вычисляем абсолютные частоты элементов с помощью np.unique()

# получаем массив уникальных значений и массив
# абсолютных частот с помощью np.unique
unique, counts = np.unique(lst, return_counts=True)
# с помощью zip() "сшиваем" два массива, с помощью
# dict() преобразовываем в словарь
freq = dict(zip(unique, counts))
```

```
freq
```

```
{'house': 1, 'pop': 2, 'techno': 1, 'trance': 1}
```

Кроме того, можно вычислить абсолютные частоты элементов с помощью класса `Counter` модуля `collections`.

```
# вычисляем абсолютные частоты элементов с помощью
# класса Counter модуля collections
freq = dict(collections.Counter(lst))
freq
```

```
{'house': 1, 'pop': 2, 'techno': 1, 'trance': 1}
```

От абсолютных частот легко перейти к относительным.

```
# теперь получим относительные частоты
freq = dict(zip(unique, counts * 100 / len(lst)))
freq
```

```
{'house': 20.0, 'pop': 40.0, 'techno': 20.0, 'trance': 20.0}
```

Метод `.reverse()` разворачивает список.

```
# создаем список
lst = [2, 4, 6, 8]
# разворачиваем список
lst.reverse()
lst
```

```
[8, 6, 4, 2]
```

Важно понимать, в каких ситуациях лучше использовать списки, а в каких – кортежи:

- списки содержат однородные данные – они должны быть одного типа, иначе с таким списком будет тяжеловато работать;
- кортеж, напротив, может содержать разнородные данные, которые в зависимости от позиции могут иметь тот или иной тип;
- список нужно использовать, когда длина заранее неизвестна либо она переменна, например список пользователей;
- кортеж удобно использовать, когда длина данных известна заранее и строго фиксирована, например как в записи из сформированной электронной таблицы.

1.1.3. Словарь (dictionary)

Словарь – одна из самых важных встроенных в Python структур данных. Его также называют *хешем, отображением или ассоциативным массивом*. Он представляет собой коллекцию пар ключ-значение переменного размера, в которой и ключ, и значение – объекты Python. Создать словарь можно с помощью фигурных скобок `{}`, отделяя ключи от значений двоеточием.