ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Классификаторы k-NN и наивный Байес

Высшая школа цифровой культуры
Университет ИТМО
dc@itmo.ru

Содержание

1 Классификатор k-NN

1.1 Небольшое введение

Добрый день, уважаемые слушатели! В прошлой лекции мы познакомились с первой из двух веток обучения с учителем – задачей регрессии – задачей предсказания числа (точнее – непрерывной переменной) по известному набору входных данных. В данной лекции мы рассмотрим несколько подходов к решению задачи второй ветки обучения с учителем – задачи классификации.

Как уже отмечалось ранее, с задачей классификации мы встречаемся чуть ли не ежедневно: дети, собирая пирамидку, классифицируют ее элементы по размеру или по форме; взрослые перед стиркой раскладывают одежду по цвету и типу ткани; в магазине овощи можно найти в отделе овощей, молочные продукты – в холодильнике, а химию – на отдельных полках, расположенных, как правило, вдалеке от съестных товаров; нас классифицируют в банках при рассмотрении заявки на кредит (кредитоспособен или нет), при трудоустройстве на новое место работы (компетентен или нет), во время пандемии (в зоне риска или нет) и много-много где еще. Приведенные примеры – малая толика, скажем так, классификаций, которые каждый день с нами (или над нами) производят(ся).

Шутки шутками, но задача классификации – одна из важнейших задач и в производственной сфере, которая, конечно, активно влияет и на нашу жизнь. Все мы пользуемся электронной почтой, и наверняка нас периодически раздражает, что важное письмо теряется среди какой-то рекламы, которую мы считаем спамом. С другой стороны, папка со спамом тоже редко пустует. Как туда попадают письма? Почему иногда туда попадают и важные письма? Дело заключается, конечно, в алгоритмах работы и настройке так называемого спам-фильтра, который относительного каждого входящего письма принимает решение: пропустить письмо или отправить в спам – вот вам и двухклассовая классификация. Но спам – это хотя бы вполне безобидно. А что по поводу подозрительных финансовых транзакций?

Предположим, вы потеряли банковскую карту, но до сих пор этого не заметили. И с этой карты стали снимать через банкомат серии крупных сумм денег. Бьемся об заклад – банк позвонит вам чуть ли не сразу, чтобы уточнить: а вы ли совершаете эти операции. Если нет – карта будет моментально заблокирована. В частности из-за этого, для обеспечения безопасности, многие банки убедительно просят сообщать им информацию о примерных датах поездок за границу, глобальных тратах и многом другом. Иначе все нестандартные действия сразу становятся подозрительными, а подозрительные действия могут повлечь моментальную блокировку доступа к счету, что

за границей, согласитесь, весьма не к стати.

Итак, обосновав мотивировку решения задачи классификации, давайте приступим к изучению, наверное, самого простого и естественного метода метрической классификации – метода ближайших соседей.

1.2 Интуитивное представление об алгоритме k-NN

Прежде чем строго сформулировать алгоритм, давайте поговорим про саму идею и суть метода. Начнем же с формальной постановки задачи: разберемся с тем, что нам дано, и что мы хотим получить.

Итак, тренировочный набор данных в разделе обучения с учителем у нас стандартен: каждый объект набора обладает p предикторами $X_1, X_2, ..., X_p$ и некоторым откликом Y. При решении задачи классификации множество откликов Y состоит из так называемых меток классов – заранее известного, конечного набора каких-то элементов. На практике классы обычно удобно нумеровать числами.

Замечание 1.2.1 *В случае решения задачи двухклассовой классификации, множество Y часто полагается, состоящим из двух элементов: «+1» и «−1» (условно – положительный и отрицательный классы), т.е. $Y = \{-1, 1\}$.*

При решении задачи многоклассовой, скажем, M-классовой классификации (при $M > 2$), множество Y обычно полагается таким: $Y = \{1, 2, ..., M\}$, где каждый элемент Y взаимно однозначно (биективно) отвечает некоторому классу предметной области.

Пример 1.2.1 *Например, решая задачу 4-ех классовой классификации с возможными метками классов: «банан», «яблоко», «груша», «апельсин», взаимно однозначное соответствие может быть таким:*

$$1 \leftrightarrow \text{«банан»},\ 2 \leftrightarrow \text{«яблоко»},\ 3 \leftrightarrow \text{«груша»},\ 4 \leftrightarrow \text{«апельсин»}.$$

Нумерация, конечно, может быть и другой. Иногда она и вовсе не обязательна, все зависит от используемого алгоритма.

Ну что, с исходными данными разобрались. Теперь давайте обсудим логику работы модели, которая могла бы на основе имеющихся тренировочных данных строить какие-то предсказания. Давайте попробуем придумать алгоритм самостоятельно, а для этого обратимся к рисунку 1.

Будем считать, что рассматриваемые объекты описываются двумя числовыми предикторами X_1, X_2 и одним из двух возможных откликов, условно,

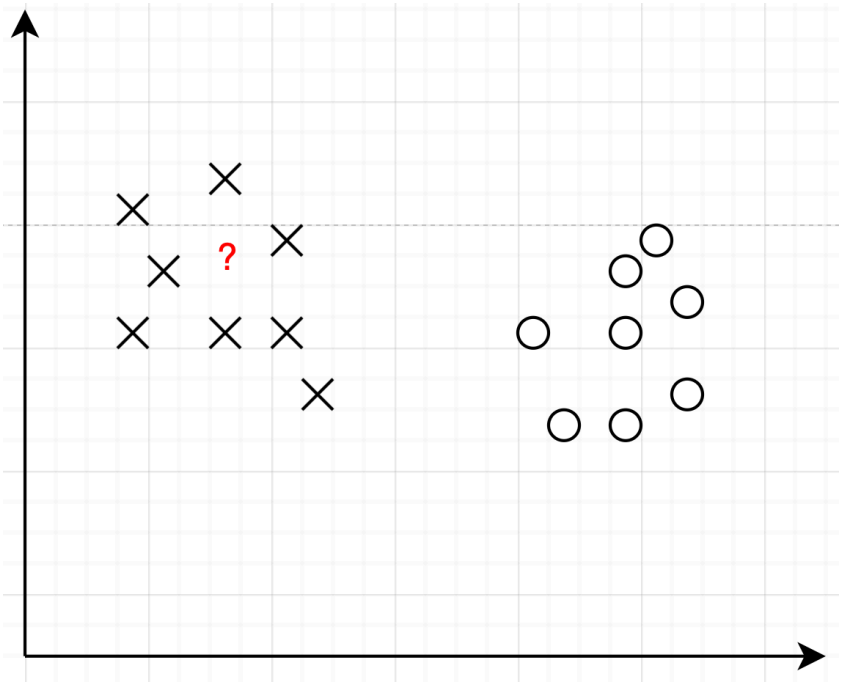

Рис. 1: Пример интуитивного подхода

«крестики» или «нолики» (двухклассовая классификация). Так как в качестве тренировочных данных используются объекты, принадлежность которых к классам заранее известна, мы сразу, для наглядности, изображаем их крестиками и ноликами (в зависимости от отклика).

Задача, конечно, состоит не в том, чтобы красиво изобразить тренировочные объекты, а в том, чтобы новому, тестовому наблюдению, обладающему двумя предикторами, назначить один из двух возможных классов. Посмотрите на появившийся красный объект. Как вы считаете, к какому классу его стоит отнести: крестики или нолики? Интуиция подсказывает, что новый объект – это, скорее всего, крестик. Почему мы так решили? Ну как... Мы видим, что рядом с новым объектом находятся в основном крестики, поэтому разумно считать, что он тоже крестик. Такое рассуждение находит отражение и в поговорке: «скажи мне кто твой друг, и я скажу кто ты». Впрочем, такая определенность бывает не всегда.

Рассмотрим менее очевидную ситуацию (рисунок 2 а)). К какому теперь классу отнести тестовый красный объект? Задумайтесь, какой бы вы сделали выбор, а затем как бы вы его обосновали? Правильного ответа на поставленный вопрос (да, собственно, как и на предыдущий), конечно, нет, но некоторой логики придерживаться все-таки можно. Скорее всего вы догадались, что самый простой и естественный путь – найти **ближайшего** представителя тренировочных данных и посмотреть на метку его класса. На глаз, или путем несложных расчетов мы видим, что этот ближайший представитель относится к классу «нолики», значит возникает идея и тестовому наблюдению

назначить класс «нолики». Все? Или не все?

Немного подумав, скорее всего возникает весьма резонный вопрос: а почему мы принимаем решение, основываясь лишь на метке класса одного тренировочного объекта? А вдруг этот представитель – выброс? Тогда мы точно ошибемся. И вообще, опираясь на нашу логику получается, что мы совершенно слепы: видим только то, что перед самым носом, и вообще не обращаем внимание на перспективу. Такой подход работает далеко не всегда и, конечно, не может считаться удовлетворительным. Что же делать? Решение, по большому счету, лежит на поверхности: нужно немного «открыть глаза» нашему алгоритму – рассмотреть побольше ближайших тренировочных данных. Давайте в нашем примере рассмотрим отклики не одного, а, например, трех «ближайших соседей». Что будет в этом случае (рисунок 2 б))?

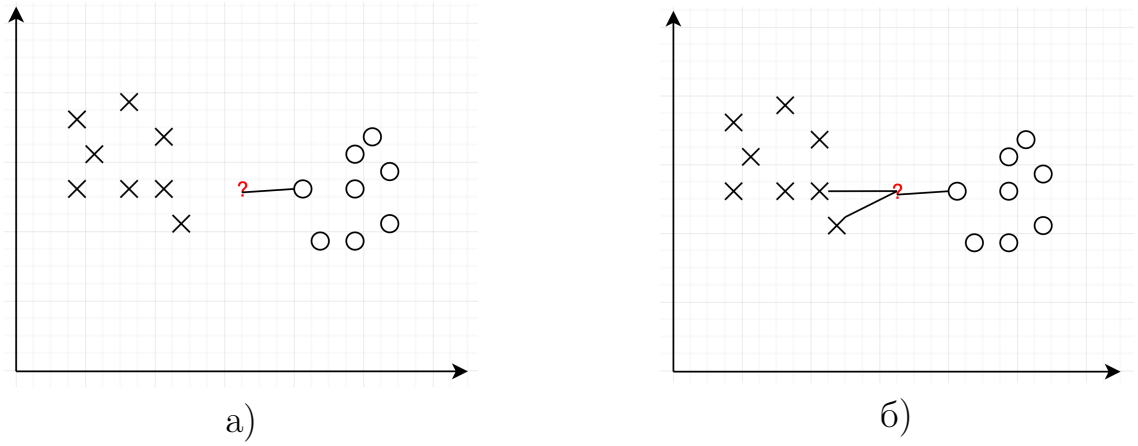

Рис. 2: Выбор количества соседей.

Мы видим, что теперь среди трех ближайших представителей тренировочных данных два крестика и один нолик. Какой класс назначать тестовому наблюдению? Видимо, решение принимается большинством, а значит назначаемый класс (если доверять трем ближайшим соседям) – «крестики».

Грубо говоря, по описанному нами принципу и работает алгоритм k-ближайших соседей (k-nearest neighbors), или, сокращенно, k-NN. Даже на таком простом примере мы видим, что чуть ли не основная задача, возникающая при использовании данного алгоритма, – правильный подбор числа k. О том, как это делать, мы поговорим еще не раз. Сейчас же мы обратим внимание вот на что: оказывается, что проблема выбора k – это вовсе не единственная проблема.

В нашем примере мы то и дело использовали слово «ближайших», и вся наша классификация опиралась на понятие расстояния. В рассмотренном примере расстояние измерялось по прямой, но всегда ли это нужно делать именно так? И вообще, а как еще можно измерять расстояние, и почему иногда это настолько критично? Давайте разбираться.

1.3 Метрики

Задаваясь вопросом о вычислении расстояния между какими-либо двумя объектами, первое, что приходит голову – это взять линейку и произвести замер по прямой (или, что то же самое, по кратчайшему пути), получив заветное число. Будут ли это метры, футы – не важно. В то же время ясно, что такой подход в условиях задачи возможен далеко не всегда. Представьте себе, условно, глобус, и пингвина на южном полюсе. Пусть глобус имеет форму шара, а радиус шара равен, скажем, трем метрам. Пингвин хочет кратчайшим образом попасть на северный полюс, какое при этом расстояние он пройдет? Просто соединить две точки и получить 6 метров не выйдет – пингвин же не пройдет сквозь глобус, верно? А как тогда? Смекалистые, конечно, догадались, что пингвину нужно пройти половину длины окружности центрального сечения шара. Так как это сечение – круг радиуса 3, то половина длины окружности, его ограничивающей – это $3\pi \approx 9.4$ метра. Согласитесь, намного больше, чем 6.

Можно привести и другой пример. Представьте, что вы приехали на экскурсию в Санкт-Петербург и планируете свой маршрут по осмотру достопримечательностей. Расстояние от Зимнего дворца до Петропавловской крепости очень маленькое – рукой подать, да только Нева мешает. А чтобы дойти или доехать, придется сделать неплохой крюк: пройти по Дворцовому мосту, обогнуть здание биржи, пересечь Биржевой мост, пройти еще чуть-чуть по набережной – и мы у цели. Но опять, расстояние, которое пришлось преодолеть, оказалось в несколько раз больше, чем расстояние по прямой!

К чему это мы ведем? К тому, что выбор способа измерения расстояния очень важен. Даже больше, зачастую этот выбор и вовсе оказывается ключевым вопросом при решении той или иной «метрической» задачи.

Определение 1.3.1 *Пусть X – какое-то множество. Метрикой (расстоянием) на множестве X называют функцию $d(x, y) : X \times X \to \mathbb{R}$, которая для любых $x, y, z \in X$ удовлетворяет трем свойствам:*

1. *Она неотрицательна, то есть $d(x, y) \geq 0$, причем*

$$d(x, y) = 0 \Leftrightarrow x = y.$$

2. *Она симметрична, то есть*

$$d(x, y) = d(y, x).$$

3. *Выполняется неравенство треугольника, то есть*

$$d(x, y) \leq d(x, z) + d(y, z).$$

Поясним каждый пункт определения более детально. Первое требование весьма естественно: расстояние должно быть неотрицательным, причем оно равно нулю между объектами тогда и только тогда, когда объекты совпадают. Второе требование говорит, что нет разницы: измерять ли расстояние от точки x до точки y, или от точки y до x, – результат будет один и тот же. Третье же требование есть не что иное, как неравенство треугольника, утверждающее, что если вершины треугольника находятся в точках x, y, z, то длина любой стороны не больше суммы длин двух других.

Рассмотрим теперь основные метрики, которые встретятся нам далее. Предположим, что мы работаем в пространстве $\mathbb{R}^p$, $x, x' \in \mathbb{R}^p$ и

$$x = (x_1, x_2, ..., x_p), \quad x' = (x'_1, x'_2, ..., x'_p).$$

1. Начнем с известной всем еще со школы евклидовой метрики (евклидова расстояния), являющейся, по сути, обобщением теоремы Пифагора. Расстояние $d_E(x, x')$ между точками x и x' полагается равным, по определению, корню из суммы квадратов разностей соответствующих координат:

$$d_E(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + \left(x_p - x'_p\right)^2},$$

$$d_E(x, x') = \sqrt{\sum_{i=1}^{p} (x_i - x'_i)^2}.$$

   Написанное расстояние выдает, наверное, наиболее привычное для нас расстояние – расстояние по прямой. Причем не важно, на прямой (в $\mathbb{R}^1$), на плоскости (в $\mathbb{R}^2$) или в пространстве (в $\mathbb{R}^3$) мы находимся.

2. На самом деле, упомянутое только что евклидово расстояние – это лишь частный случай вот какого расстояния – расстояния Минковского:

$$d_q(x, x') = \sqrt[q]{|x_1 - x'_1|^q + |x_2 - x'_2|^q + \ldots + \left|x_p - x'_p\right|^q}, \quad q \geq 1,$$

$$d_q(x, x') = \sqrt[q]{\sum_{i=1}^{p} |x_i - x'_i|^q}, \quad p \geq 1.$$

   Понятно, что евклидово расстояние получается из расстояния Минковского в случае, когда $q = 2$.

3. Если взять $q = 1$, то получим расстояние, которое называется манхэттенским или расстоянием городских кварталов:

$$d_1(x, x') = \sum_{i=1}^{p} |x_i - x'_i|.$$

Это расстояние, по сути, решает вопрос расчета длины поездки по городу, если все улицы города строго перпендикулярны. В этом случае произвольный «адекватный» маршрут, соединяющий две выбранные точки, имеет одинаковую длину, совпадающую с нашими о ней представлениями (рисунок 3). На рисунке красным, синим и желтым построены «адекватные» маршруты, соединяющие две точки. Как уже говорилось, с точки зрения манхэттенского расстояния все эти маршруты имеют одинаковую длину. Зеленым же построено кратчайшее расстояние – по прямой. Такой маршрут, конечно, невозможен.

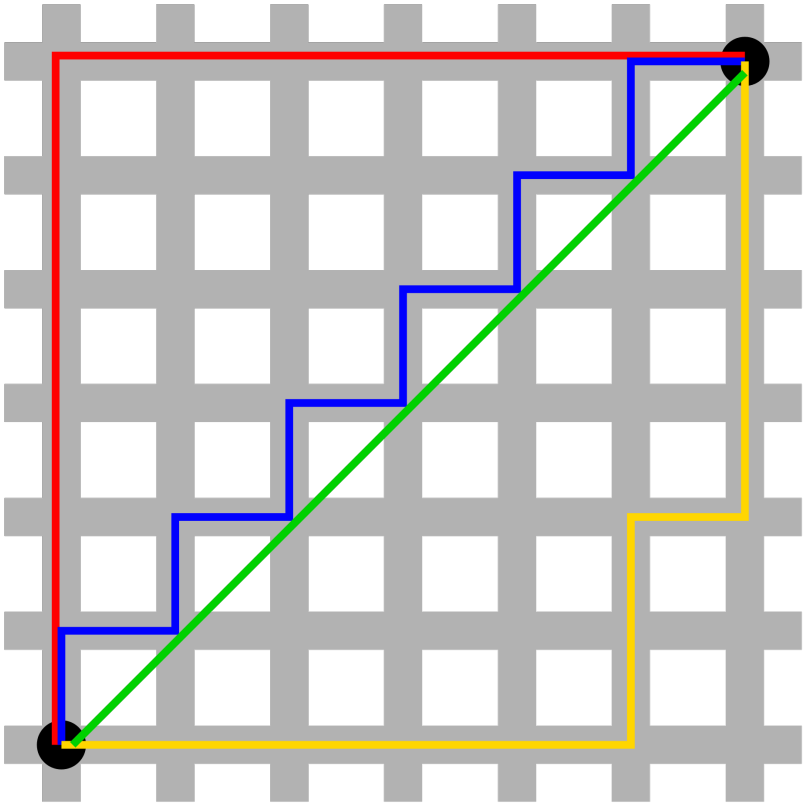

Рис. 3: Сравнение евклидова и манхэттенского расстояний

4. Если же q устремить к $+\infty$, то получим так называемое расстояние Чебышёва:

$$d_\infty(x, x') = \max_{i \in \{1,\ldots,p\}} |x_i - x'_i| .$$

Расстояние Чебышёва показывает максимальное покоординатное отклонение одного объекта от другого и может быть полезно, например, вот в какой ситуации. Представьте, что вы совершаете замеры воды в водохранилищах $1, 2, ..., n$, причем норма уровня воды в каждом из них соответственно равна $x'_1, x'_2, ..., x'_p$. Тогда расстояние Чебышёва показывает максимальное отклонение уровня воды от нормы среди всех

водохранилищ.

Можно проверить, что все функции, которые мы только что обсудили, и правда являются метриками, то есть удовлетворяют определению, сформулированному ранее. Так как мы будем использовать метрики для сравнения «близости» объектов, немаловажно понимать их геометрическую интерпретацию. Заодно увидим, в рамках какой метрики мыслите конкретно Вы :)

Представьте себе множество точек плоскости, удаленных от центра координат $O(0,0)$ на расстояние не большее, чем 1. Что вы себе представили? Вероятнее всего, перед глазами всплывает круг единичного радиуса с центром в начале координат. В чем подвох, спросите вы? Подвох в том, что в изначальном вопросе использовано слово расстояние, и, в зависимости от того, какое расстояние берется, геометрический образ будет сильно отличаться. Чтобы показать это наглядно, переведем заданный ранее вопрос на язык математики: требуется изобразить множество точек $x \in \mathbb{R}^2$ таких, что

$$d_q(O, x) \leq 1, \quad O(0,0),$$

при разных значениях p.

В случае с евклидовым расстоянием, при $p = 2$, заявленное неравенство записывается, как:

$$d_2(O, x) = \sqrt{x_1^2 + x_2^2} \leq 1.$$

Возводя его в квадрат, приходим к эквивалентному неравенству:

$$x_1^2 + x_2^2 \leq 1,$$

которое задает круг радиуса 1 с центром в начале координат. Большинство людей думает именно в терминах евклидовой метрики.

В случае манхэттенского расстояния, при $q = 1$, получаем внутренность ромба (на самом деле – повернутого на $\pi/4$ квадрата):

$$|x_1| + |x_2| \leq 1.$$

В случае расстояния Чебышёва, при $q = +\infty$, получаем внутренность квадрата со стороной два:

$$\max_{i \in \{1,2\}} |x_i| \leq 1 \Leftrightarrow |x_1| \leq 1, |x_2| \leq 1.$$

Все множества можно увидеть на рисунке 4.

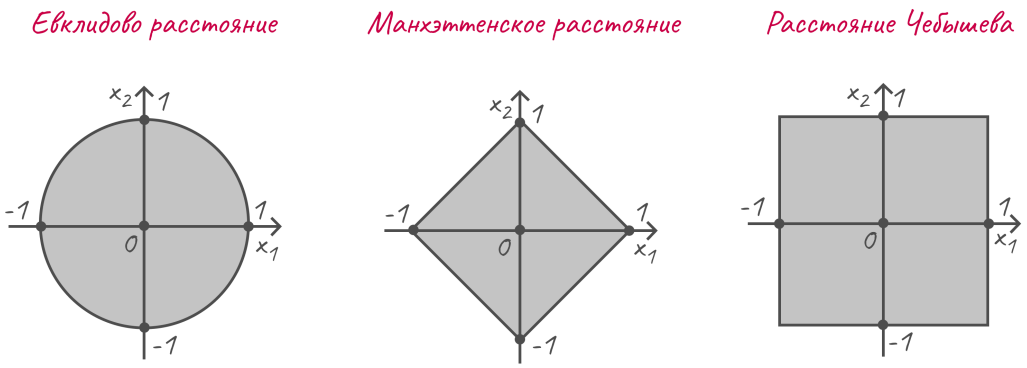

Рис. 4: Расстояния слева направо: евклидово, манхэттенское, Чебышёва.

1.4 Классификация методом k-NN

Разобравшись с понятием метрики, мы теперь смело можем перейти к построению алгоритма классификации методом k-NN. Пусть имеется тренировочный набор данных $X = (x_1, \ldots, x_n)$ объема n,

$$x_i = (x_{i1}, x_{i2}, ..., x_{ip}), \quad i \in \{1, 2, ..., n\},$$

причем каждому объекту x_i соответствует отклик $y_i \in Y$, а на p-мерном множестве объектов задана метрика d. Тогда алгоритм классификации таков:

1. Для нового объекта z вычислить расстояния $d(z, x_i)$ до каждого объекта x_i, $i \in \{1, 2, \ldots, n\}$.

2. Элементы тренировочного набора данных расположить в порядке неубывания расстояний до z:

$$d(z, x_1^{(z)}) \leq d(z, x_2^{(z)}) \leq ... \leq d(z, x_n^{(z)}),$$

где $x_1^{(z)} = x_{t_1}$, а t_1 – любое из решений задачи $\underset{i \in \{1,2,...,n\}}{\operatorname{Arg\,min}} d(z, x_i)$, $x_2^{(z)} = x_{t_2}$, а t_2 – любое из решений задачи $\underset{i \in \{1,2,...,n\} \setminus \{t_1\}}{\operatorname{Arg\,min}} d(z, x_i)$, и так далее.

3. Отклики перенумеровать согласно пункту 2:

$$y_i^{(z)} = y_{t_i}, \quad i \in \{1, 2, ..., n\}.$$

4. Среди ближайших k соседей найти такой класс $y \in Y$, который встречается чаще всего (в случае, если классов несколько – выбрать любой):

$$a\,(z, k) = \underset{y \in Y}{\operatorname{Arg\,max}} \sum_{i=1}^{k} \mathsf{I}\left(y_i^{(z)} = y\right),$$

где функция $\mathsf{I}(A)$ – индикатор события A, равна единице, если событие A произошло (истинно), и нулю иначе. В рассматриваемом случае индикатор дает единицу, если метка класса $y_i^{(z)}$ равна метке класса y.

Итак, в первом пункте мы вычисляем расстояние от тестового объекта до каждого элемента тренировочного набора данных. Во втором пункте мы упорядочиваем эти расстояния по неубыванию и перенумеровываем элементы тренировочного набора данных (первый элемент – самый близкий к тестовому объекту, второй – второй по близости, и так далее). В третьем пункте мы перенумеровываем отклики согласно перенумерованным объектам, а в четвертом ищем тот класс (или те классы), объекты которых встретились среди k ближайших элементов чаще всего. Описанный алгоритм полностью повторяет все те эвристические рассуждения, которые были даны в самом начале.

Конечно, логично считать, что выбранная в алгоритме метрика d достаточно качественно отображает сходство объектов, то есть чем больше значение $d(x, x')$, тем менее схожими являются объекты x и x', и наоборот. Описанный алгоритм классификации как раз-таки и предполагает, что схожие объекты гораздо чаще лежат в одном классе, нежели в разных. Говоря более формально, мы работаем в рамках так называемой **гипотезы компактности**: возможные классы образуют компактно локализованные подмножества в пространстве объектов.

Определение 1.4.1 *Алгоритмы, основанные на анализе сходства объектов при помощи метрики d, часто называют метрическими.*

Из данного определения понятно, что описанный алгоритм k-NN является метрическим. Впрочем, иногда алгоритмы называют метрическими и когда «функция сходства» d метрикой, согласно введенному нами ранее определению, не является (особенно часто в этом случае нарушено неравенство треугольника).

Отметим также, что алгоритм k-NN часто называют **ленивым**, так как само по себе обучение происходит лишь в момент предсказания. В итоге, грубо говоря, обучение состоит просто в хранении тренировочного набора данных.

Замечание 1.4.1 *На практике признаки могут иметь разные единицы измерения, что может искажать реальное расстояние между объектами. Один из способов решения проблемы – нормализация данных. Напомним, что один из самых простых способов – переход к относительным значениям с помощью линейной нормировки:*

$$X'_i = \frac{X_i - X_{\min}}{X_{\max} - X_{\min}},$$

где $X_{\min}$ – минимальное значение рассматриваемого предиктора, $X_{\max}$ – максимальное значение, X_i – нормируемое значение, X'_i – нормированное значение.

1.5 Пример классификации

Давайте рассмотрим работу описанного алгоритма вот на каком примере. Данные вы можете видеть в таблице. Каждый объект характеризуется двумя атрибутами (предикторами): сладость и хруст, а также относится к одному из трех классов: фрукт, овощ или протеин.

Продукт	Сладость	Хруст	Класс
банан	10	1	фрукт
апельсин	7	4	фрукт
виноград	8	3	фрукт
креветка	2	2	протеин
бекон	1	5	протеин
орехи	3	3	протеин
сыр	2	1	протеин
рыба	3	2	протеин
огурец	2	8	овощ
яблоко	9	8	фрукт
морковь	4	10	овощ
сельдерей	2	9	овощ
салат айсберг	3	7	овощ
груша	8	7	фрукт

Итак, начнем с того, что разместим все эти данные, для больше наглядности, на плоскости (рисунок 5). По горизонтали отметим сладость, по вертикали хруст. Таким образом, каждому фрукту, овощу или протеину соответствует точка на плоскости своего цвета: овощам – желтые точки, фруктам – темно-зеленые, а протеинам – светло-зеленые. Мы видим, что данные очень хорошо разделились на группы (рисунок 6).

Теперь предположим, что нам необходимо классифицировать новый объект, зная только значения сладости и хруста. Допустим, это перец со значениями сладость — 6, хруст — 9. Согласно описанному алгоритму, нам необходимо рассчитать расстояние от тестового объекта до всех тренировочных объектов, предварительно выбрав метрику d. Выберем сначала евклидову метрику d_2.

В нашем примере все переменные будем считать нормированными по шкале от 0 до 10. Произведем расчет расстояния между объектами «морковь»

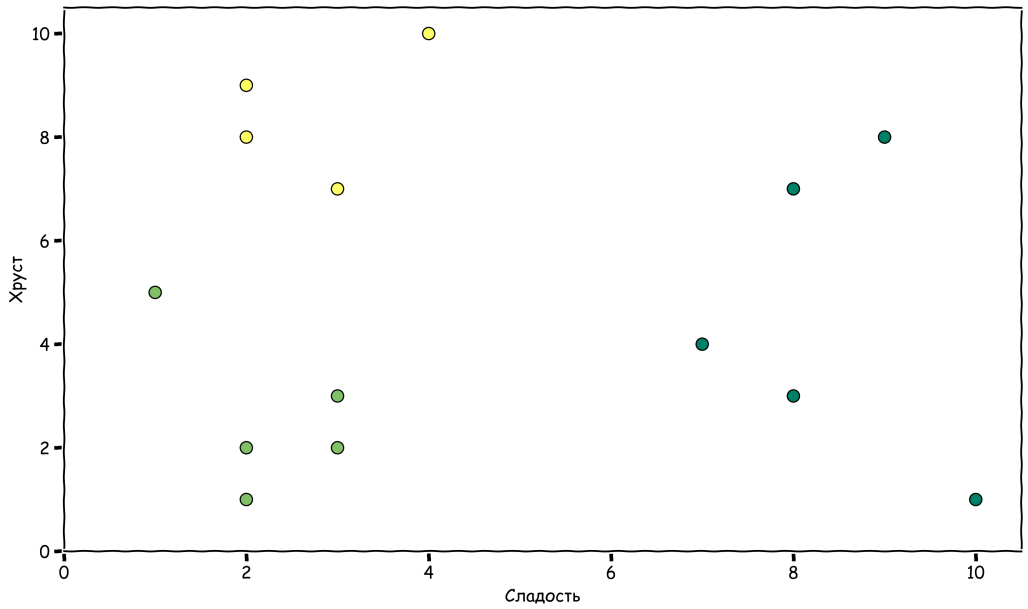

Рис. 5: Визуализация тренировочных данных.

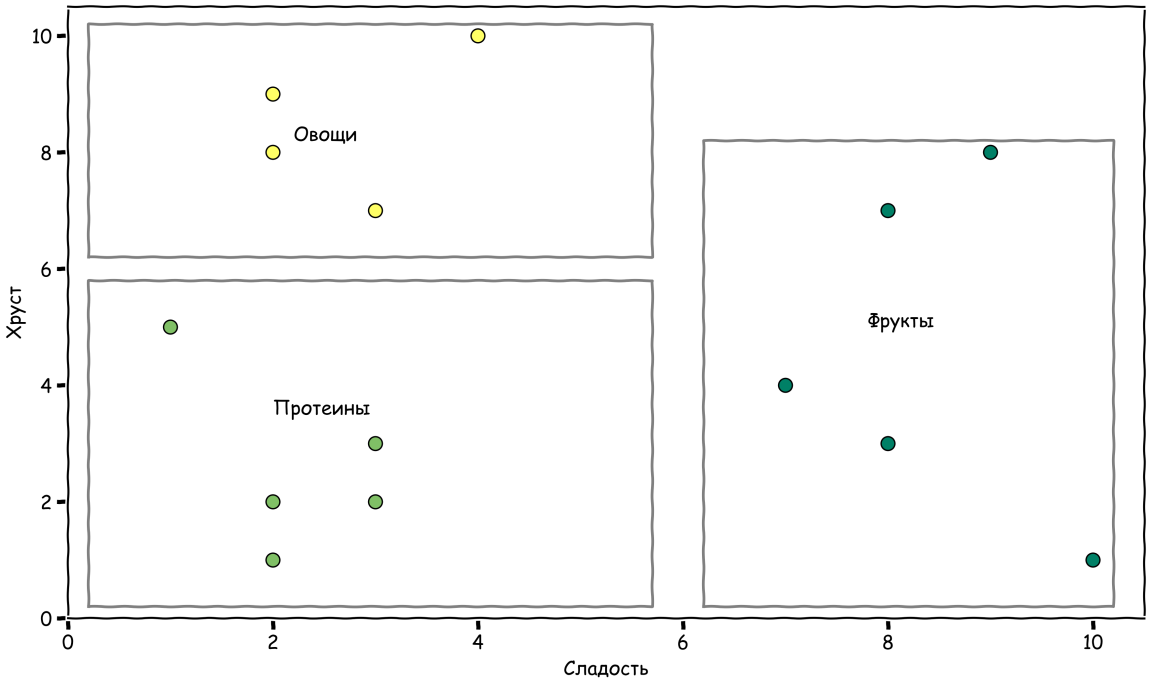

Рис. 6: Объединение в группы.

с атрибутами $(4, 10)$ и «перец» с атрибутами $(6, 9)$:

$$d_2(\text{морковь, перец}) = \sqrt{(6-4)^2 + (9-10)^2} \approx 2.24.$$

Аналогичный расчет произведем для всех остальных тренировочных объектов, результаты можно видеть в таблице:

Продукт	Расстояние
банан	8.94
апельсин	5.10
виноград	6.32
креветка	8.06
бекон	6.4
орехи	6.71
сыр	8.94
рыба	7.62
огурец	4.12
яблоко	3.16
морковь	2.24
сельдерей	4
салат айсберг	3.61
груша	2.83

Отсортируем наши данные по неубыванию расстояния, результат представлен в таблице:

Продукт	Расстояние
морковь	2.24
груша	2.83
яблоко	3.16
салат айсберг	3.61
сельдерей	4
огурец	4.12
апельсин	5.10
виноград	6.32
бекон	6.4
орехи	6.71
рыба	7.62
креветка	8.06
банан	8.94
сыр	8.94

Итак, «ближайший» объект к тестовому – это морковь, затем идет груша, яблоко и так далее. Теперь выберем число k. Пусть $k = 1$, тогда, согласно описанному алгоритму, тестовому объекту присваивается метка класса самого ближайшего объекта тренировочных данных. Так как ближайший объект – это морковь, принадлежащая классу овощей, то и перцу присваивается класс овощ. При $k = 3$ самыми близкими окажутся яблоко, груша и морковь, тогда перец будет отнесен к классу фруктов, так как представителей этого класса среди ближайших соседей большинство. При $k = 4$ к ближайшим соседям добавится салат, и тестовый объект оказывается возможным отнести как к классу фрукт, так и к классу овощ. При $k = 5, 6, 7$, перец будет уверенно классифицирован, как овощ.

Продукт	Класс	Расстояние
морковь	овощ	2.24
груша	фрукт	2.83
яблоко	фрукт	3.16
салат айсберг	овощ	3.61
сельдерей	овощ	4
огурец	овощ	4.12
апельсин	фрукт	5.10
виноград	фрукт	6.32
бекон	протеин	6.4
орехи	протеин	6.71
рыба	протеин	7.62
креветка	протеин	8.06
банан	фрукт	8.94
сыр	протеин	8.94

Теперь давайте посмотрим, изменится ли наша классификация, если взять другую метрику? Возьмем манхеттенское расстояние d_1:

$$d_1(x, x') = \sum_{i=1}^{p} |x_i - x'_i| .$$

Будем классифицировать все тот же перец со значениями предикторов: сладость – 6, хруст – 9. Найдем расстояние между объектами «морковь» и «перец»:

$$d_1(\text{морковь, перец}) = |6 - 4| + |9 - 10| = 3.$$

Различия в способе вычисления расстояния отдельно показаны на рисунке 7. Так, если евклидово расстояние d_2 – это кратчайший путь по прямой (по гипотенузе), то манхэттенское расстояние d_1 – это сумма длин соответствующих

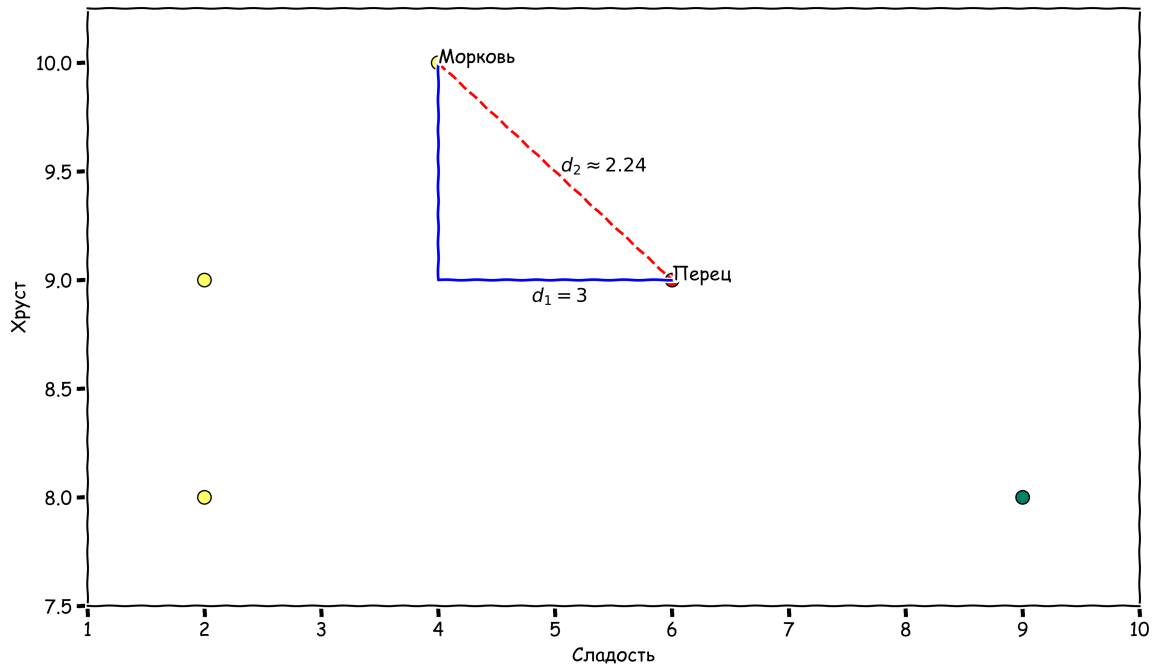

Рис. 7: Отличие в расстояниях.

катетов. Аналогичный расчет произведем для всех точек и сразу отсортируем тренировочные объекты по неубыванию расстояний от тестового объекта, результаты представлены в таблице:

Продукт	Класс	Расстояние
морковь	овощ	3
яблоко	фрукт	4
сельдерей	овощ	4
груша	фрукт	4
огурец	овощ	5
салат айсберг	овощ	5
апельсин	фрукт	6
виноград	фрукт	8
бекон	протеин	9
орехи	протеин	9
рыба	протеин	10
креветка	протеин	11
банан	фрукт	12
сыр	протеин	12

Теперь мы не сможем уверенно классифицировать перец при $k = 2, 3, 4$, но сможем отнести его к его классу овощей при $k = 1, 5, 6, 7$. Также можно обратить внимание, что манхэттенское расстояние дает нам целые значения в рамках рассматриваемого примера. Кроме того, имеется очень много одина-

ковых расстояний, а сортировка равноудаленных объектов сильно влияет на результат.

1.6 Взвешенный k-NN

Итак, на данном этапе мы познакомились с простейшим вариантом алгоритма k-NN. В то же время, при рассмотрении последнего примера мы столкнулись со следующей проблемой: при некоторых значениях k новый, тестовый объект непонятно как классифицировать – ему подходит сразу несколько классов. Конечно, в алгоритме сказано, что в качестве нового класса можно взять любой из подходящих, но иногда это кажется абсурдным. Представьте, скажем, что вы решаете задачу двухклассовой классификации. Тогда невозможность классифицировать новый объект, по своей сути означает, что алгоритм отвечает что-то вроде «решай сам, я не знаю». Такое, конечно, вполне возможно, но нежелательно.

В случае двухклассовой классификации вы, наверное, догадались, что простейшее решение таково: в качестве значений параметра k достаточно брать только нечетные числа. В таком случае количество ближайших соседей не будет делиться (нацело) на 2, а значит и равенства в количестве голосов за тот или иной класс не будет. Таким образом, классификация обязательно будет проведена. А что, если классов больше, чем два? В данной лекции мы изложим лишь один подход к решению этой проблемы – это так называемый взвешенный метод k-NN. Идея метода такова: каждому объекту тренировочных данных x_i приписывается некоторый вес ω_i, зависящий, вообще говоря, от тестового объекта z, т.е.

$$\omega_i = \omega_i(x_i, z), \quad i \in \{1, 2, ..., n\}.$$

По итогу, тестовому наблюдению приписывается тот класс, суммарный вес представителей которого среди k ближайших соседей наибольший (или один из таких классов, если их несколько).

Алгоритм взвешенного k-NN может быть формально записан, например, так. Пусть имеется тренировочный набор данных $X = (x_1, \ldots, x_n)$ объема n,

$$x_i = (x_{i1}, x_{i2}, ..., x_{ip}), \quad i \in \{1, 2, ..., n\},$$

причем каждому объекту x_i соответствует отклик $y_i \in Y$, а на p-мерном множестве объектов задана метрика d.

1. Для нового объекта z вычислить расстояния $d(z, x_i)$ до каждого объекта x_i, а также веса

$$\omega_i = \omega_i(x_i, z), \quad i \in \{1, 2, \ldots, n\}$$

2. Элементы тренировочного набора данных расположить в порядке неубывания расстояний до z:

$$d(z, x_1^{(z)}) \leq d(z, x_2^{(z)}) \leq ... \leq d(z, x_n^{(z)}),$$

где $x_1^{(z)} = x_{t_1}$, а t_1 – любое из решений задачи $\underset{i \in \{1,2,...,n\}}{\text{Arg min}}\, d(z, x_i)$, $x_2^{(z)} = x_{t_2}$, а t_2 – любое из решений задачи $\underset{i \in \{1,2,...,n\} \setminus \{t_1\}}{\text{Arg min}}\, d(z, x_i)$, и так далее.

3. Отклики и веса перенумеровать согласно пункту 2:

$$y_i^{(z)} = y_{t_i}, \quad \omega_i^{(z)} = \omega_{t_i}, \quad i \in \{1, 2, ..., n\}.$$

4. Среди ближайших k соседей найти такой класс $y \in Y$, который имеет наибольший вес (в случае, если классов несколько – выбрать любой):

$$a\,(z, k) = \underset{y \in Y}{\text{Arg max}} \sum_{i=1}^{k} \mathsf{I}\left(y_i^{(z)} = y\right) \omega_i^{(z)}.$$

Один из способов задания весов основан на близости «проверяемого» объекта по отношению к другим. Идея способа проста: чем меньше расстояние от тренировочного объекта до тестового, тем более значимым при классификации является тренировочный объект. Даже точнее, тем более значим голос этого тренировочного объекта, или тем более значима метка его класса. Например, в качестве веса w_i можно взять величину, обратно пропорциональную квадрату расстояния между объектами x_i и z, то есть

$$\omega_i = \omega_i(x_i, z) = \frac{1}{d^2(x_i, z)}.$$

Таким образом, выбранные веса увеличивают значимость ближайших объектов, и уменьшают значимость более дальних объектов. Чуть позже мы подробно разберем числовой пример, но сейчас еще немного поговорим про проблему выбора числа k.

1.7 Немного про число k в алгоритме k-NN

Дадим несколько комментариев по поводу того, как в общем и целом изменение параметра k влияет на классификацию при использовании метода k-NN (конечно, лишь на конкретных данных). Посмотрите на рисунок 8.

Совершенно понятно, что при малых значениях параметра k мы существенно ограничиваем «зону видимости» для нашего классификатора. Например, при $k = 1$ он просто «запоминает» тренировочный набор данных,

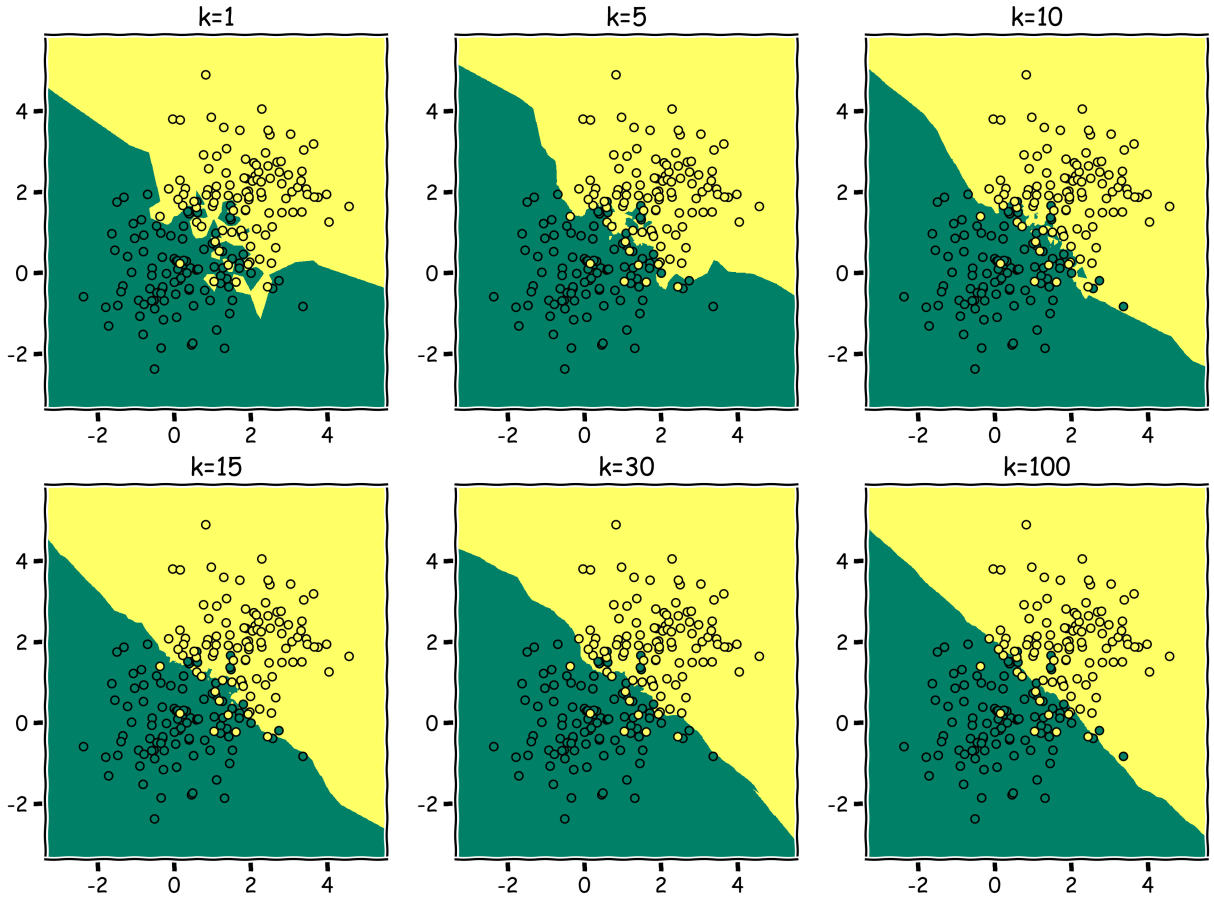

Рис. 8: Влияние числа k.

отсюда и неровности в границах классов, какие-то островки причудливой формы. Поянтно также, что при небольших значениях k классификатор оказывается чувствительным к выбросам, хотя и практически идеально работает с теми данными, на которых учился.

С увеличением k граница разделения становится весьма гладкой и превращается практически в прямую, разделение на классы выглядит более естественным, появляется все больше ошибок на тренировочных данных. Модель оказывается легкой и удобной в интерпретации.

Так какое значение k лучше? Это весьма сложный вопрос. Разумным подходом является оценка частоты ошибок путем разделения исходного набора данных на тестовую и обучающую выборки. Давайте об этом подробно и поговорим.

2 Алгоритмы, их оценка и разделение набора данных

Итак, вопрос, который до сих пор остается нерешенным – это вопрос выбора параметра алгоритма k-NN, то есть параметра k. Оказывается, ответ

на этот вопрос не лежит на поверхности. Чтобы в нем разобраться, придется копнуть несколько глубже и обратиться к идеологии. Точнее, начать с ответа на вопрос: а чего мы пытаемся добиться (глобально), решая задачу классификации, и какие при этом возникают проблемы?

Добиться мы пытаемся следующего: мы хотим произвольному объекту из пространства признаков присвоить какой-то класс (желательно, конечно, правильный) из имеющегося набора. Какие возникают проблемы? Проблемы вполне себе понятны: мы ограничены в знаниях, ведь у нас в руках есть только тренировочный набор данных; мы же должны опыт, полученный на этом конечном наборе данных, распространить, вообще говоря, на больший объем данных (возможно, бесконечный). Итого оказывается, что наша основная цель – научить алгоритм обобщать полученные знания на новые, ранее не встречавшиеся прецеденты. Но сделав это используя, например, k-NN, правильнее скорее задуматься вот о чем: сделали-то сделали, но как теперь проверить, что получилось хорошо, и что вообще значит это «хорошо»? Для того, чтобы осветить эти вопросы, для начала введем базовые определения, которые в дальнейшем будут использоваться чуть ли не всюду.

2.1 Алгоритмы и эмпирический риск

Мы постоянно используем следующие выражения: алгоритм, обучить алгоритм, оценить алгоритм. Но что такое этот «алгоритм»? Оказывается, что все весьма просто.

Определение 2.1.1 *Пусть X – множество всевозможных объектов, а Y – множество откликов. Отображение*

$$a : X \to Y$$

называется алгоритмом.

Итак, алгоритм – это функция, сопоставляющая произвольному объекту некоторый отклик.

Замечание 2.1.1 *Поясним введенные обозначения несколько детальнее. Рассматривая модель простейшей линейной регрессии, в качестве множества X мы рассматривали множество вещественных чисел $\mathbb{R}$ или какое-то его непрерывное подмножество, а алгоритм задавался следующим образом:*

$$a(x) = \theta_0 + \theta_1 x.$$

В только что рассмотренном примере классификации, пространство X – это пространство

$$X = [0, 10] \times [0, 10],$$

так как имеется всего два предиктора (сладость и хруст), которые могут принимать произвольные значения в диапазоне от 0 *до* 10 *(включительно), а* Y *– это трехэлементное множество*

$$Y = \{\textit{овощ}, \textit{фрукт}, \textit{протеин}\}.$$

Алгоритм k-NN (и взвешенный алгоритм k-NN), описанные ранее, как раз-таки являются алгоритмами в смысле введенного определения.

Теперь давайте разбираться, как мы будем оценивать наш алгоритм.

Определение 2.1.2 *Пусть* $a : X \to Y$ *– алгоритм. Функция потерь (loss function)* $L(a, x)$ *– это произвольная неотрицательная функция, характеризующая величину ошибки алгоритма* a *на объекте* x*.*

Понятно, что если ошибки алгоритма на объекте x нет, то, видимо, $L(a, x) = 0$ и оказывается разумным принять следующее определение.

Определение 2.1.3 *Пусть* $a : X \to Y$ *– алгоритм, а* $L(a, x)$ *– функция потерь. Если* $L(a, x) = 0$*, то ответ* $a(x)$ *называют* ***корректным****.*

Конечно, при оценке того или иного алгоритма, разумно рассматривать присущие ему потери не на каком-то одном объекте (вдруг он, например, выброс), а на большем количестве объектов и, что привычно в статистике, как-то эти потери усреднять (то есть смотреть на качество работы алгоритма «в среднем»). Тут и возникает так называемый эмпирический риск.

Определение 2.1.4 *Пусть* $x_1, x_2, ..., x_n \in X$ *– некоторый набор данных,* $a(x) : X \to Y$ *– алгоритм,* $L(a, x)$ *– функция потерь. Функционалом потерь (эмпирическим риском (empirical risk), функционалом средних потерь) называется функционал*

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n} \sum_{i=1}^{n} L(a, x_i).$$

Замечание 2.1.2 *Отметим, что в качестве функции потерь при решении задачи классификации часто используют такую функцию:*

$$L(a, x) = \mathsf{I}(a(x) \neq y(x)) =$$

$$= \begin{cases} 1, & \textit{если отклик } y(x) \textit{ объекта } x \textit{ не совпал с } a(x) \\ 0, & \textit{иначе} \end{cases},$$

где $y(x)$ – это известный отклик объекта x. Смысл этой функции достаточно прост: она выдает 1, если ответ алгоритма не совпадает с истинным откликом, и 0 иначе. В этом случае функционал потерь принимает следующий вид

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} \mathsf{I}(a(x_i) \neq y(x_i))$$

и выдает не что иное, как долю неправильных ответов построенного классификатора на наборе данных $x_1, x_2, ..., x_n$. Ясно, что мы хотим минимизировать значение написанного функционала. Запомним это.

При решении задачи регрессии, мы уже встречались со следующей функцией потерь:

$$L(a, x) = (a(x) - y(x))^2,$$

где снова $y(x)$ – это известный отклик объекта x. Функционал потерь, построенный по данной функции потерь, таков

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y(x_i))^2.$$

Напомним, что решая задачу простейшей линейной регрессии, алгоритм мы искали в виде $a(x) = \theta_0 + \theta_1 x$. Функционал потерь в этом случае принимает уже знакомый вид

$$Q(a, L, x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y(x_i))^2,$$

а для поиска коэффициентов θ_0 и θ_1 мы его, опять-таки, минимизировали, то есть решали задачу

$$\operatorname*{Arg\,min}_{\theta_0,\theta_1} \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y(x_i))^2 = \operatorname*{Arg\,min}_{\theta_0,\theta_1} \sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y(x_i))^2.$$

Как легко заметить, это в точности метод наименьших квадратов! Функции потерь бывают и другого вида, пока что мы не будем касаться этого вопроса более детально.

Написанные наблюдения как, видимо, и здравый смысл подсказывают нам, что алгоритм тем лучше, чем меньше значение соответствующего ему функционала потерь. Тем самым, мы приходим к следующему выводу: имеет

смысл искать такой алгоритм или такие алгоритмы $a(x)$, которые минимизируют функционал потерь, то есть решать следующую задачу:

$$\operatorname*{Arg\,min}_{a \in \mathcal{A}} Q(a, L, x_1, ..., x_n).$$

В написанной задаче $\mathcal{A}$ – это некоторое рассматриваемое нами множество алгоритмов. В случае задачи простейшей линейной регрессии, множество алгоритмов $\mathcal{A}$ описывалось, например, так:

$$\mathcal{A} = \{\theta_0 + \theta_1 x,\ \theta_0, \theta_1 \in \mathbb{R}\}.$$

При рассмотрении метода k-NN под множеством алгоритмов $\mathcal{A}$ можно понимать такое множество:

$$\mathcal{A} = \{\text{алгоритмы k-NN}, \quad k \in \mathbb{N}\}.$$

Замечание 2.1.3 *Все приведенные примеры множеств алгоритмов пока что, по сути, являются ни чем иным, как множеством возможных параметров (или гиперпараметров) некоторых «фиксированных» алгоритмов. На практике часто возникают, конечно, и более «хитрые» множества.*

Обучение алгоритма – это, по сути дела, выбор «наилучшего» алгоритма из набора $\mathcal{A}$.

Итак, нам удалось разобраться с общей постановкой задачи оценивания алгоритма. В то же время, вопрос выбора (или построения) множества $x_1, x_2, ..., x_n$, на котором проводится оценка, все еще остается открытым. Этот вопрос оказывается очень важным, и его никак нельзя недооценивать.

Замечание 2.1.4 *Подчеркнем отдельно, что поднятый вопрос, как и затронутый ранее вопрос выбора множества $\mathcal{A}$ – это не просто какая-то теоретическая трудность; это, в некотором смысле, одна из важнейших отправных точек при построении хорошей модели.*

Для того, чтобы не быть голословными, приведем такой «абсурдный» пример, показывающий, как важно правильно и четко ставить задачу в машинном обучении (да и не только в нем). Скажем, пусть мы решили оценивать алгоритм на ***всем*** *тренировочном наборе данных $x_1, x_2, ..., x_n$ с откликами $y_1, y_2, ..., y_n$, и никак не ограничили множество $\mathcal{A}$. Какой тогда алгоритм будет лучше всего справляться с поставленной задачей? Наверное, вы догадались, вот такой:*

$$a(x_i) = y_i.$$

Конечно, внимательный слушатель скажет, что такой алгоритм не удовлетворяет самому определению алгоритма, ведь он выдает хоть какой-то ответ только на элементах тренировочного набора данных. Не беда, при $x \notin \{x_1, x_2, ..., x_n\}$ *значение* $a(x)$ *можно положить любым, при этом всегда будет выполняться:*

$$Q(a, L, x_1, ..., x_n) = 0,$$

а лучше и быть не может!

Итак, мы составили алгоритм, который идеален с точки зрения функционала потерь, но совершенно бесполезен на практике. Причиной этому служит даже не то, что мы не ограничили как-то множество $\mathcal{A}$. Причиной этому служит то, что алгоритм **обучался** на выборке $x_1, x_2, ..., x_n$, для элементов которой он уже **видел** правильные ответы. Нам же интересно, как он будет себя вести на данных, которых он раньше **не видел**. Но чтобы это оценить, мы сами должны знать «правильные ответы». Давайте посмотрим, как это реализуется на практике.

2.2 Разделение тренировочного набора данных

Как мы поняли в предыдущем пункте, нас на самом-то деле интересует нахождение такого алгоритма $a(x)$, для которого значение функционала потерь было бы маленьким на новом, ранее невиданном алгоритмом наборе данных при условии, что данные «подчиняются закономерностям тренировочного набора данных». Последняя, возможно несколько странная фраза станет понятной чуть позже, когда мы коснемся вероятностной подоплеки кухни задачи классификации. Сейчас же ее понимать можно и нужно именно в смысле озвученного ранее **принципа компактности**.

Замечание 2.2.1 *Подчеркнем отдельно, что данный пункт лекции является, субъективно, очень важным, потому что он пытается пояснить идеологию обучения и тестирования алгоритмов в машинном обучении. Алгоритмы могут быть очень хорошими, но при неправильном обучении ничего «хорошего», скорее всего, не получится.*

Итак, вернемся к вопросу: откуда же нам взять данные, которые алгоритм не видел ранее, если у нас самих есть только тренировочный набор данных

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}?$$

Первая идея – разделить этот набор на две части D_{train} и D_{test} и обучить модель на суженном тренировочном наборе D_{train}, после чего оценить результат на наборе D_{test}. Еще раз подчеркнем, что мы искусственным образом сужаем исходный тренировочный набор данных D до набора D_{train} лишь для

того, чтобы проверить, а как обученный нами алгоритм $a(x)$ работает на тех данных, которых он не видел ранее, вычислив функционал потерь

$$Q(a, L, D_{test})$$

на тестовом наборе D_{test}.

Замечание 2.2.2 *Конечно, возникает вопрос, в каком соотношении и как делить тренировочный набор данных D. Обычно отношение $D_{train} : D_{test}$ составляет примерно* $80 : 20$*, но такое деление очень условно. Куда более деликатен следующий вопрос: а как делить?*

Если нет никакой информации про время, когда собирались те или иные данные из набора D, а набор данных D репрезентативен, то вряд ли можно придумать что-то лучшее, чем случайное разделение.

Если же есть данные о времени сбора, то лучше производить деление по какой-то временной засечке. Например, если данные собираются в течение недели (и есть надежда, что они однородны), то все данные, собранные (условно) до четверга включительно идут в тренировочный набор, а данные, собранные после четверга, в тестовый. Проведя такую засечку, мы полностью выкидываем при обучении алгоритма информацию о данных, полученных в будущем, то есть полностью от него, от будущего, абстрагируемся. Это, кстати, ровно то, чего мы и хотим: чтобы то, что обучилось сегодня, работало завтра. Конечно, все это – эмпирические подходы к оценке алгоритма.

Оказывается, что на практике специалисты часто поступают иначе – более хитро. Они делят набор данных не на две части, а на три, откалывая от D_{test} так называемую validation-part D_{val}. И это совсем нетривиальный, но очень важный момент. Попробуйте догадаться, зачем так делают специалисты. Мы, конечно, это сейчас же обсудим.

Представьте, что у вас есть несколько моделей, скажем, несколько классификаторов k-NN с разными значениями k, причем каждый из этих классификаторов вы обучили на тренировочном наборе данных D_{train}, а также оценили на тестовом D_{test}. Какой вы выберите? Конечно тот, функционал потерь которого на тестовом наборе данных выдаст меньшее значение, не так ли? Если так, то тут и кроется (не идеологическая, а концептуальная) ошибка: финальное решение опять же принято с использованием информации из будущего, а это ну никак не годится! То, чего мы так старались избежать – использования неизвестного при обучении – опять играет злую шутку. Такое переиспользование тестового набора данных часто ведет к так называемому переобучению модели, и ничего хорошего не сулит.

В итоге, важным оказывается следующее наблюдение: использование тестового набора данных D_{test} слишком часто, по своей сути ведет к тому, что

алгоритм начинает обучаться и на наборе данных D_{test}, что противоречит всему тому, что было сказано ранее.

Теперь к практике. Имея набор данных D_{val}, имеет смысл выбрать тот алгоритм, который допускает меньшее число ошибок на этом наборе данных, и тогда именно этот алгоритм и нужно оценить на D_{test}, предварительно обучив на всем наборе D_{train} (включая искуственно отколотый набор D_{val}). Полученное значение функционала потерь на D_{test} как раз-таки разумно считать «средней» ошибкой обученного алгоритма. В следующем пункте мы расскажем про так называемую k-блочную кросс-валидацию (k-fold cross-validation), которая обобщает описанную ранее идею.

2.3 k-блочная кросс-валидация (k-fold cross-validation)

Итак, предположим, что у нас есть некоторый набор алгоритмов. Как же из них выбрать лучший, используя уже озвученную k-блочную кросс-валидацию? На самом деле механизм выбора очень прост, он сразу становится понятным, если посмотреть на следующий рисунок (рисунок 9):

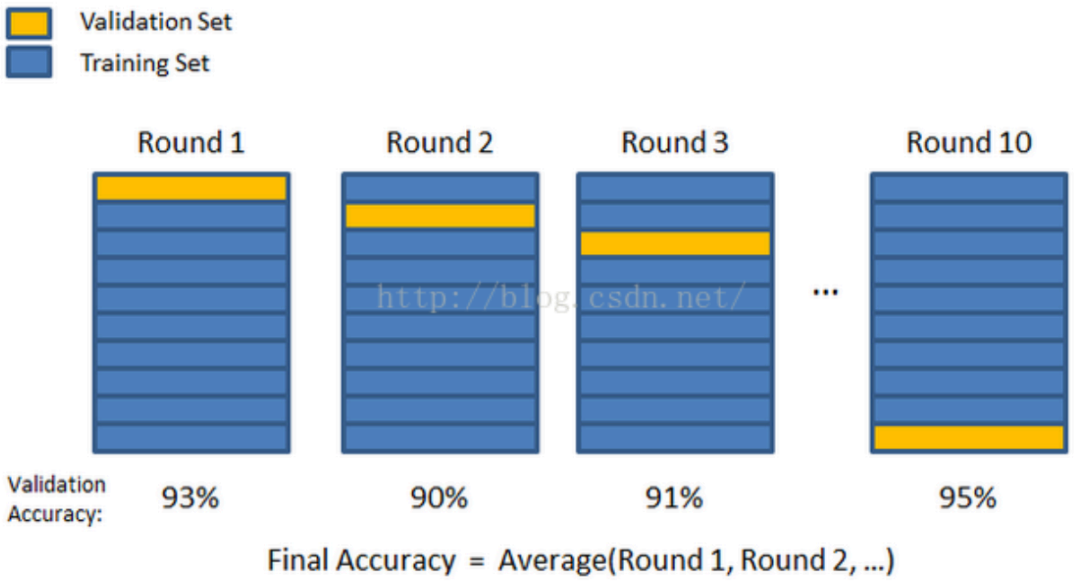

Рис. 9: k-fold cross-validation

Вместе с рисунком проведем и формальное описание алгоритма.

1. Пусть исходный набор данных разделен на D_{train} и D_{test}.

2. Набор данных D_{train} разбивается на k непересекающихся примерно одинаковых по размеру блоков (множеств):

$$D_{train} = D_1 \cup D_2 \cup \ldots \cup D_k.$$

3. Пусть i меняется в диапазоне $\{1, 2, ..., k\}$:

   (a) Производится обучение выбранного алгоритма $a(x)$ на множестве

   $$D^i_{train} = D_{train} \setminus D_i.$$

   (b) Производится тестирование выбранного алгоритма $a(x)$ на множестве $D^i_{test} = D_i$ и вычисляется значение

   $$\mathsf{Loss}_i(a) = Q(a, L, D^i_{test}).$$

4. Финальная оценка ошибки алгоритма $a(x)$ в результате k-блочной кросс-валидации такова:

   $$\mathsf{Loss}(a) = \frac{1}{k} \sum_{i=1}^{k} \mathsf{Loss}_i(a).$$

5. Шаги $3 - 4$ повторяются для каждого исследуемого (сравниваемого) алгоритма.

6. В качестве лучшего алгоритма выбирается тот алгоритм a, значение $\mathsf{Loss}(a)$ которого наименьшее.

7. Выбранный лучший алгоритм обучается на наборе данных D_{train} и тестируется на наборе D_{test}. Полученная ошибка $Q(a, L, D_{test})$ считается «средней» ошибкой лучшего алгоритма.

Итак, по сути дела тренировочный набор данных делится на несколько более маленьких кусочков, каждый из которых на своей итерации выступает тестовым набором данных – набором данных для оценивания алгоритма. Финальная оценка каждого алгоритма – это усреднение оценок на каждой итерации.

2.4 Пример: взвешенный k-NN и k-блочная кросс-валидация

Вернемся к нашему примеру с продуктами и, используя кросс-валидацию при $k = 3$ выясним, какой из алгоритмов (взвешенный k-NN при k = 3, 4 или обычный k-NN при k = 4) справляется лучше. Предположим, что деление на D_{train} и D_{test} уже произведено, а в качестве D_{train} выступает уже знакомый нам набор данных

Продукт	Сладость	Хруст	Класс
банан	10	1	фрукт
апельсин	7	4	фрукт
виноград	8	3	фрукт
креветка	2	2	протеин
бекон	1	5	протеин
орехи	3	3	протеин
сыр	2	1	протеин
рыба	3	2	протеин
огурец	2	8	овощ
яблоко	9	8	фрукт
морковь	4	10	овощ
сельдерей	2	9	овощ
салат айсберг	3	7	овощ
груша	8	7	фрукт

Видно, что объем тренировочного набора данных D_{train} равен 14, а значит блоки могут быть выбраны, например, таких размеров: $5, 5, 4$.

На первом шаге будем использовать первый блок в качестве тестового (таблица 1), а второй и третий – в качестве тренировочных (таблица 2).

Продукт	Сладость	Хруст	Класс
банан	10	1	фрукт
апельсин	7	4	фрукт
виноград	8	3	фрукт
креветка	2	2	протеин
бекон	1	5	протеин

Таблица 1: Итерация 1. Тестовые данные.

Отобразим на плоскости данные из тренировочного набора (рисунок 10) и проведем классификацию продуктов из тестового набора взвешенным методом k-NN, когда параметр k равен 4. В качестве метрики используем евклидово расстояние, а в качестве веса ω_i – величину, обратно пропорциональную квадрату расстояния между объектами x_i и классифицируемым объектом z:

$$\omega_i = \frac{1}{d^2(x_i, z)}.$$

Найдем расстояние от продукта «банан» до всех элементов тренировочного набора данных, а также присущие им веса. Например, «расстояние» от банана до сыра равно:

$$d_2\,(\text{банан, сыр}) = \sqrt{(10-2)^2 + (1-1)^2} = 8.$$

Продукт	Сладость	Хруст	Класс
орехи	3	3	протеин
сыр	2	1	протеин
рыба	3	2	протеин
огурец	2	8	овощ
яблоко	9	8	фрукт
морковь	4	10	овощ
сельдерей	2	9	овощ
салат айсберг	3	7	овощ
груша	8	7	фрукт

Таблица 2: Итерация 1. Тренировочные данные.

Вес тренировочного объекта «сыр» в этом случае равен:

$$w_{\text{банан, сыр}} = \frac{1}{d_2^2\,(\text{банан, сыр})} = \frac{1}{64} \approx 0.016.$$

Аналогичным образом вычислим расстояния и веса до остальных объектов, данные представлены в таблице 3. Назначим класс на основе четырех бли-

Продукт	Класс	Расстояние	Вес
орехи	протеин	7.28	0.019
сыр	протеин	8	0.016
рыба	протеин	7.071	0.020
огурец	овощ	10.63	0.009
яблоко	фрукт	7.071	0.020
морковь	овощ	10.817	0.009
сельдерей	овощ	11.314	0.008
салат айсберг	овощ	9.22	0.012
груша	фрукт	6.325	0.025

Таблица 3: Расстояние и веса для продукта «банан»

жайших соседей: груша, яблоко, рыба и орехи. Если бы мы использовали обычный k-NN, то мы бы столкнулись с проблемой конкуренции двух классов: фрукты и протеины. При использовании взвешенного k-NN такой проблемы удается избежать. Действительно, сумма весов, отвечающих классу фрукты, равна 0.045, а сумма весов, отвечающих классу протеины, равна 0.039. Таким образом, банан будет классифицирован как фрукт.

Аналогичные вычисления выполняются для каждого продукта из тестового набора (таблица 4). Вы можете проверить себя и выполнить расчеты, мы же представим сразу итоговые результаты классификации.

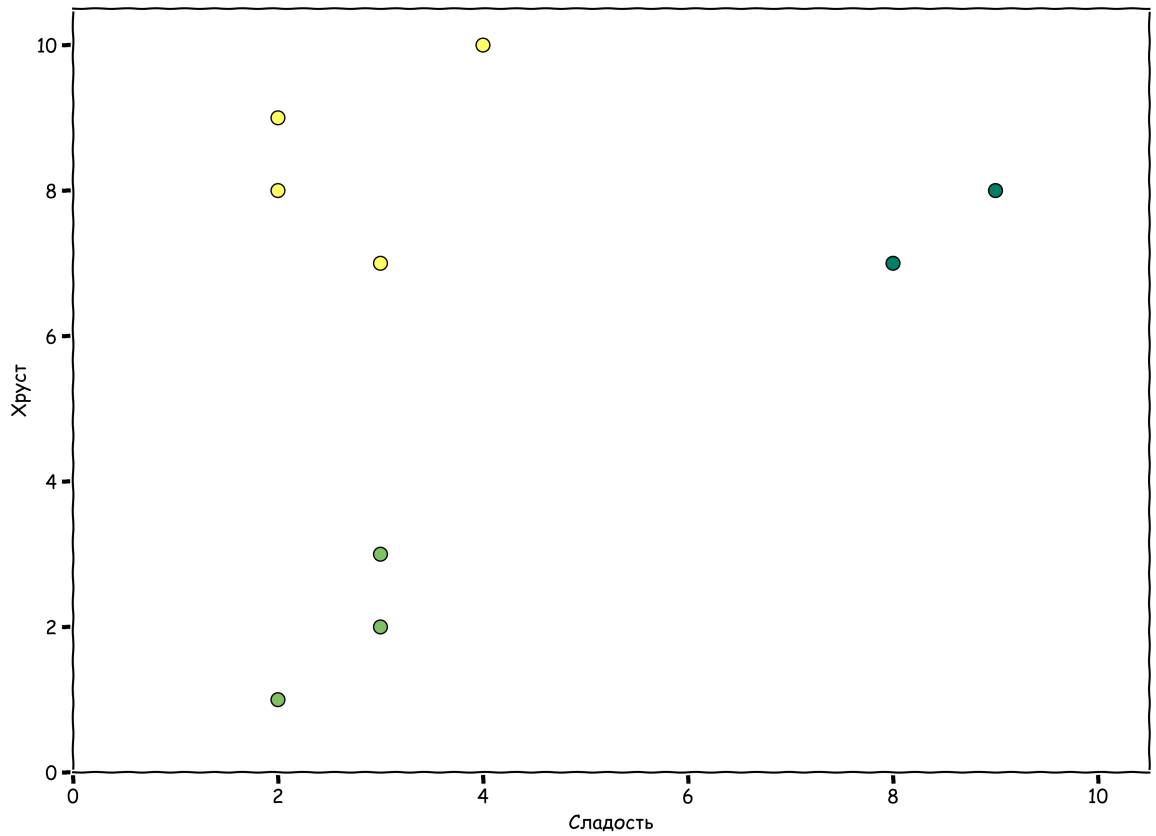

Рис. 10: Итерация 1. Тренировочные данные.

Как видим из таблицы, имеет место 1 ошибка (подвел бекон), а значит при использовании стандартной для задачи классификации функции потерь

$$L(a, x) = \mathsf{I}(a(x) \neq y(x)),$$

значение функционала потерь будет равно 0.2:

$$\mathsf{Loss}_1(\text{взв. 4-NN}) = Q(a, L, D^1_{train}) = \frac{1}{5} = 0.2.$$

Далее вы можете ознакомиться с результатами итераций 2 и 3, для которых повторяются аналогичные шаги (таблицы 5, 6). В этих итерациях назначенные классы полностью совпадают с исходными. Таким образом частота ошибок будет нулевой, а усреднение всех ошибок по трем итерациям составит около 0.07:

$$\mathsf{Loss}(\text{взв. 4-NN}) = \frac{1}{3}\sum_{i=1}^{3} \mathsf{Loss}_i = \frac{1}{15} \approx 0.07.$$

Если в качестве k во взвешенном k-NN использовать число 3, то для нового алгоритма

$$\mathsf{Loss}(\text{взв. 3-NN}) = \frac{1}{3}\sum_{i=1}^{3} \mathsf{Loss}_i = \frac{1}{5} = 0.2.$$

Продукт	Сладость	Хруст	Исходный класс	Назначенный класс
банан	10	1	фрукт	фрукт
апельсин	7	4	фрукт	фрукт
виноград	8	3	фрукт	фрукт
креветка	2	2	протеин	протеин
бекон	1	5	протеин	овощ

Таблица 4: Итерация 1. Тестовые данные.

Если же оставить $k = 4$, но при этом отказаться от взвешивания соседей, то для такого алгоритма

$$\mathsf{Loss}(4\text{-NN}) = \frac{1}{3}\sum_{i=1}^{3}\mathsf{Loss}_i = \frac{1}{4} = 0.25.$$

Итак, согласно описанному алгоритму, в качестве финального классификатора из рассматриваемого набора имеет смысл выбирать тот, оценка средней ошибки Loss для которого является наименьшей. Среди рассмотренных алгоритмов наименьшая ошибка – это ошибка Loss(взв. 4-NN). Значит, среди исследуемых алгоритмов мы выбираем алгоритм взвешенных 4-NN, и для дальнейших прогнозов (для обучения) в качестве тренировочного набора данных используем все множество D_{train}. Для оценки истинной средней ошибки выбранного алгоритма, имеет смысл протестировать его на заранее заготовленном множестве D_{test}.

Продукт	Сладость	Хруст	Исходный класс	Назначенный класс
орехи	3	3	протеин	протеин
сыр	2	1	протеин	протеин
рыба	3	2	протеин	протеин
огурец	2	8	овощ	овощ
яблоко	9	8	фрукт	фрукт

Таблица 5: Итерация 2. Тестовые данные.

2.5 Проклятие размерности

Раз уж мы в предыдущем пункте снова затронули алгоритм k-NN, обратимся к еще одной, теперь уже «метрической» проблеме машинного обучения – проклятию размерности.

Продукт	Сладость	Хруст	Исходный класс	Назначенный класс
морковь	4	10	овощ	овощ
сельдерей	2	9	овощ	овощ
салат айсберг	3	7	овощ	овощ
груша	8	7	фрукт	фрукт

Таблица 6: Итерация 3. Тестовые данные.

Давайте подумаем над такой задачей. Предположим, что наши тренировочные данные (n штук) равномерно распределены в некотором единичном гиперкубе в пространстве $\mathbb{R}^d$ при достаточно большом d. Достаточно большое значение d означает, что каждый объект обладает достаточно большим количеством признаков (d штуками).

Поместим теперь в центр нашего гиперкуба новый тестовый объект и ответим на такой вопрос: гиперкуб с ребром какой длины l должен быть построен вокруг тестового объекта, чтобы внутри этого гиперкуба содержалось k ближайших соседей (рисунок 11), то есть k точек тренировочного набора данных?

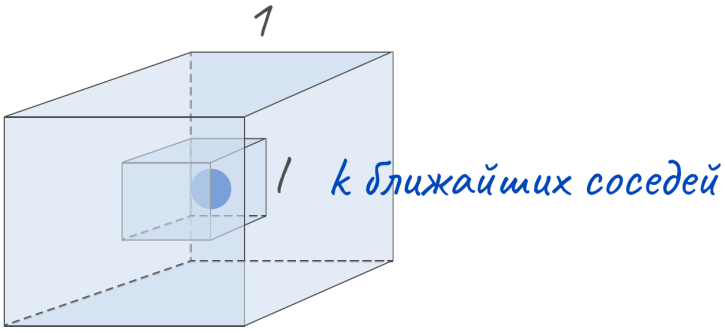

Рис. 11: Многокоробочность

Математика в данном случае очень проста. Объем рассматриваемого гиперкуба равен, конечно, l^d и, в силу того, что тренировочные данные распределены равномерно в гиперкубе с единичным ребром, а внутрь искомого гиперкуба должно попасть k ближайших соседей,

$$l^d \approx \frac{k}{n} \;\Rightarrow\; l \approx \left(\frac{k}{n}\right)^{1/d}.$$

На первый взгляд ничего удивительного, но давайте посчитаем. Для этого зададим какие-то «адекватные» параметры. Пусть, например, $k = 10$, $n =$

1000, тогда исследуем длину ребра при разных d, учитывая, что

$$l \approx \left(\frac{1}{100}\right)^{1/d}.$$

При $d = 2$ получим 0.1 – неплохо. При $d = 10$ получим около 0.631, при $d = 100$ получим около 0.955, а при $d = 1000$ около 0.995. Попытка проиллюстрировать написанные соотношения приведена на рисунке 12.

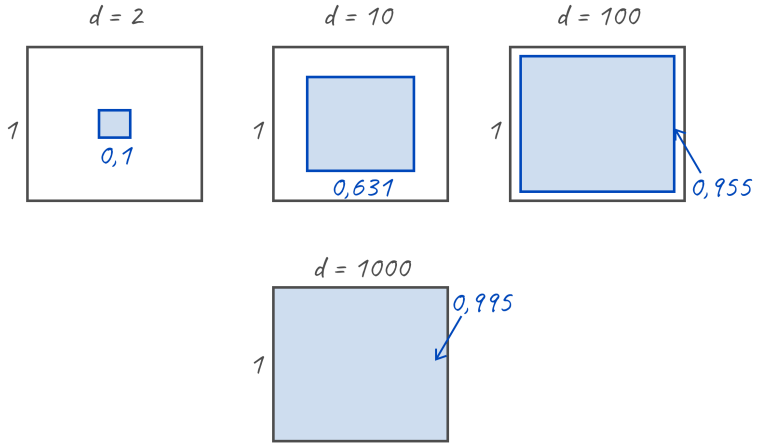

Рис. 12: Заполнение пространства

Что мы видим? А видим мы то, что чем больше значение d, тем дальше все объекты находятся друг от друга и, грубо говоря, тем ближе они расположены к границам гиперкуба. Внимание: они вовсе не внутри этого куба, ведь иначе длину ребра гиперкуба, содержащего k соседей, можно бы было взять меньшей, и не было бы такой заполняемости. И вот это наблюдение – огромная проблема, ведь из него следует, что метод k-NN вовсе несостоятелен! При увеличении d ближайшие соседи, по сути, тоже находятся на границах гиперкуба вместе со всеми остальными тренировочными данными. Выбрав наудачу одного соседа, тут же рядом, на расстоянии пол шага находится куча других тренировочных данных лишь случайно не вошедших в число ближайших соседей: какой тут принцип компактности?!

Все эти рассуждения очень естественны с точки зрения математики, но очень неестественны для нашего понимания реальности, и это абсолютно нормально: мы привыкли к трехмерному пространству. Попробуем дать еще одно объяснение тому, что происходит, уже на вероятностном языке. Давайте найдем вероятность того, что случайно взятая точка внутри гиперкуба находится не на его границе, а внутри. Для этого, конечно, нужно, чтобы каждая одномерная координата точки лежала внутри отрезка, например, $[0, 1]$, а не на его границе. Отступим на $\varepsilon > 0$ от граничных точек, тогда вероятность оказаться во внутренней части отрезка равна $(1 - 2\varepsilon)$ (рисунок 13). Понятно,

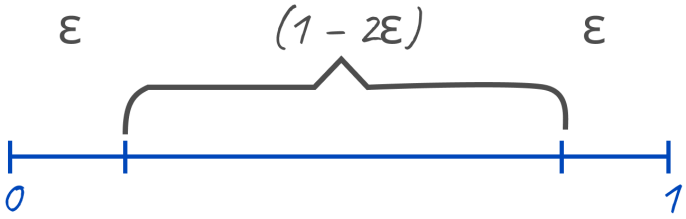

Рис. 13: Вероятностное объяснение

что для d-мерного пространства (в силу независимости признаков) получаем, что вероятность оказаться внутри гиперкуба равна

$$(1-2\varepsilon)^d,$$

и последнее выражение очень быстро стремится к нулю с ростом d. Итак, при больших d вероятность оказаться внутри гиперкуба почти ноль.

На самом деле после всех этих рассуждений может показаться, что метрические подходы и вовсе несостоятельны. Это, конечно, не так, ведь проведенные рассуждения касаются только равномерного распределения данных, что бывает не часто. Куда чаще оказывается, что на самом деле данные заполняют какое-то подпространство пространства $\mathbb{R}^d$ (гиперплокость, более хитрое многообразие), и тогда метод k-NN вполне хорошо справляется со своей задачей. Все сказанное еще раз указывает на важность предварительной обработки данных, снижения размерности и прочим изыскам, ведь на практике достаточно редко складывается ситуация, что все предикторы исходного набора данных важны.

Итак, поговорив достаточно про метрические классификаторы, перейдем теперь к некоторой альтернативе – вероятностным классификаторам.

3 Наивный байесовский классификатор

3.1 Некоторые вводные замечания

В предыдущем пункте мы рассмотрели один из способов решения задачи классификации – с использованием метрического классификатора k-NN. Классифицируя новый объект, мы пользовались лишь имеющимися у нас тренировочными данными, не используя (и, в общем-то, не предполагая) никакой вероятностной зависимости между предикторами и откликами, хотя о, в некотором смысле, наличии таковой говорилось еще во введении. Чем пользовались мы, так это гипотезой компактности. В этой части лекции мы рассмотрим метод классификации, имеющий откровенно вероятностную природу, а основным инструментом будет выступать известная теорема Байеса.

Необходимость (как и адекватность) вероятностных моделей подкрепляется следующими наблюдениями. В задачах обучения с учителем элементы тренировочного набора данных – это, в общем-то, не реальные объекты, а лишь доступные нам данные о реальных объектах, и это наблюдение очень существенно. Например, данные могут быть **неточными**: измерение значений признаков (атрибутов) тренировочных объектов, а также измерение откликов зачастую производится с погрешностью (хотя бы из-за округления) – достаточно вспомнить предположения модели регрессии, обсуждаемой ранее. Кроме того, данные могут быть **неполными**, так как зачастую измеряются не все мыслимые признаки, а лишь те, которые доступны для измерения. Вспомните, иногда и вовсе непонятно, какие признаки присущи объекту, а какие – нет. В результате, одному и тому же описанию объекта x могут в реальности соответствовать как разные объекты, так и разные отклики. Именно поэтому предположение о существовании явной зависимости, дающей по описанию объекта единственно верный ответ (отклик), кажется достаточно наивным. Вероятностные модели позволяют, в некотором смысле, устранить описанную некорректность. Кроме того, явные зависимости являются частным случаем вероятностных, но мы подробно на этом останавливаться не будем.

3.2 Небольшая вероятностная подоплека

В отличие от случая k-NN, где мы начали рассмотрение алгоритма с наглядных геометрических соображений, здесь мы сразу приведем вероятностную подоплеку, и, собственно, сам алгоритм, а затем посмотрим на примеры его применения в реальных задачах.

Для начала напомним некоторые факты из теории вероятностей. Начнем с определения условной вероятности.

Определение 3.2.1 *Пусть событие B таково, что $\mathsf{P}(B) > 0$. Условной вероятностью события A при условии, что произошло событие B, называется число*

$$\mathsf{P}(A|B) = \frac{\mathsf{P}(A \cap B)}{\mathsf{P}(B)}.$$

Иными словами, условная вероятность – это отношение вероятности события $A \cap B$, то есть события «произошло и A, и B», к вероятности события B. Иначе – ищется отношение вероятности доли события A, попавшего в B, к вероятности события B – не что иное, как перенормировка, или, простонапросто, изменение (сужение) вероятностного пространства.

Это определение моментально может быть переписано в виде

$$\mathsf{P}(A \cap B) = \mathsf{P}(A|B)\mathsf{P}(B).$$

Если, к тому же, $\mathsf{P}(A) > 0$, то мы в праве рассмотреть и такую условную вероятность:

$$\mathsf{P}(B|A) = \frac{\mathsf{P}(B \cap A)}{\mathsf{P}(A)} = \frac{\mathsf{P}(A \cap B)}{\mathsf{P}(A)},$$

откуда

$$\mathsf{P}(A \cap B) = \mathsf{P}(B|A)\mathsf{P}(A).$$

Объединяя написанные соотношения, получаем так называемые **правила умножения**:

$$\mathsf{P}\left(A \cap B\right) = \mathsf{P}(A|B)\mathsf{P}(B) = \mathsf{P}(B|A)\mathsf{P}(A).$$

Заменив теперь в определении условной вероятности числитель на второе соотношение, приходим к так называемой **формуле Байеса**, широко используемой не только в теории вероятностей, но и в машинном обучении:

$$\mathsf{P}(A|B) = \frac{\mathsf{P}(B|A)\mathsf{P}(A)}{\mathsf{P}(B)}.$$

Обратите внимание, в этой формуле условные вероятности «поменялись местами». Отметим в определении часто встречающиеся названия объектов, входящих в написанную формулу.

Определение 3.2.2 *Вероятность* $\mathsf{P}(B|A)$ *называют правдоподобием, вероятности* $\mathsf{P}(A)$ *и* $\mathsf{P}(B)$ *– априорными вероятностями, а вероятность* $\mathsf{P}(A|B)$ *– апостериорной вероятностью.*

Естественно пока что остается открытым вопрос: чем нам помогают введенные понятия, и при чем тут вообще машинное обучение? Рассмотрим уже упоминавшуюся ранее жизненную и хорошо известную задачу – задачу классификации входящих писем на «спам» и «не спам» и посмотрим, что мы на данный момент можем получить. Пример тренировочных данных представлен в таблице. Упростим себе задачу, считая, что предиктор всего один и принимает всего два значения: 0 и 1, причем значение, равное 1 означает, что текст письма содержит слово «приз». Столбец «спам» – это отклик, который тоже принимает всего два значения 0 и 1, причем значение, равное 1 означает, что письмо отнесено к спаму.

Текст письма	**«приз»**	**«спам»**
Вы выиграли приз!	1	1
Оплатите первый месяц услуг и получите приз.	1	1
Получите приз за рекомендацию банковских услуг.	1	0
Вам достался дом в наследство.	0	1
.....	...	...

Предположим, что всего анализировалось 80 писем, а слово «приз» встретилось среди пяти писем, помеченных как спам, и среди трех, помеченных как не спам. В оставшихся 72 письмах слова «приз» не было, при этом 27 из них были помечены как «спам», а остальные 45 – как не «спам». Поместим данные в такую таблицу:

	«спам»	**не «спам»**
всего писем	32	48
есть слово «приз»	5	3
нет слова «приз»	27	45

.

Итак, у нас всего 32 письма, помеченных как «спам», среди которых 5 содержат слово «приз», и 48 писем, помеченных как не «спам», среди которых 3 содержат слово «приз». Интересен вопрос: как оценить вероятность, что письмо относится к классу «спам», если оно содержит слово «приз»?

Вспомним, как на основе приведенных данных рассчитываются вероятности различных событий. Конечно, в нашем случае истинные вероятности неизвестны, но мы их можем оценить, используя частоту. Например, вероятность того, что случайное письмо, попавшее в спам, содержит слово «приз», оценивается при помощи частоты числом $5/32$, ведь на основе тренировочных данных всего 32 письма отнесены к категории «спам», и из них ровно 5 содержат слово «приз». Таким образом, согласно собранным данным, резонно считать, что

$$\mathsf{P}(\text{«приз»} \mid \text{«спам»}) = \frac{5}{32}.$$

В свою очередь, вероятность того, что присланное письмо относится к категории «спам», оценивается числом $32/80$, а вероятность того, что письмо содержит слово «приз» – числом $8/80$, то есть

$$\mathsf{P}(\text{«спам»}) = \frac{32}{80},\ \mathsf{P}(\text{«приз»}) = \frac{8}{80}.$$

Теперь, используя формулу Байеса, мы можем найти вероятность того, что письмо является спамом, если оно содержит слово «приз»:

$$\mathsf{P}(\text{«спам»} \mid \text{«приз»}) = \frac{\mathsf{P}(\text{«приз»} \mid \text{«спам»}) \cdot \mathsf{P}(\text{«спам»})}{\mathsf{P}(\text{«приз»})} = \frac{\frac{5}{32} \cdot \frac{32}{80}}{\frac{8}{80}} = \frac{5}{8} = 0.625.$$

Как мы видим, письмо, содержащее слово «приз», скорее стоит отнести к категории «спам», нежели к категории не «спам». Аналогичным образом можно рассчитать вероятности и других событий. Итак, мы вспомнили, как на основе частот рассчитывать вероятности. Теперь можно приступить к построению классификатора.

3.3 Построение модели и вывод формул

Давайте для начала обратимся непосредственно к вероятностной постановке задачи. Пусть X – это множество объектов, каждый из которых описывается p признаками – случайными величинами $X_1, X_2, ..., X_p$, и откликом Y (снова отметим, что отклик принимает конечное множество значений). Будем трактовать $X \times Y$ как вероятностное пространство с некоторым совместным распределением. Кроме того, для простоты изложения будем предполагать, что все случайные величины $X_1, X_2, ..., X_p$ имеют **дискретное распределение с конечным множеством значений**. Наша задача – назначить новому, тестовому объекту с предикторами $X_1, X_2, ..., X_p$, одну из меток классов из множества Y. Как нам это сделать? Наверное понятно, что сначала оказывается логичным вычислить вероятности отнесения нового объекта к каждому из имеющихся классов, то есть набор вероятностей:

$$\mathsf{P}(\text{Класс} = y | X_1, X_2, ..., X_p), \quad y \in Y.$$

Замечание 3.3.1 *Важно понять, что показывает написанное (не совсем строго) выражение при фиксированном $y \in Y$. Оно показывает вероятность отнесения объекта с заранее неизвестным классом к классу y, учитывая то, что объект обладает атрибутами $X_1, X_2, ..., X_p$ (строже – это соотношение задает соответствующее вероятностное распределение).*

Итак, набор вероятностей у нас есть. Как теперь понять к какому классу $y^* \in Y$ логичнее всего отнести тестовый объект? Конечно, к наиболее вероятному (или к одному из наиболее вероятных, если их несколько), то есть выбрать такой $y^* \in Y$, что

$$y^* = \underset{y \in Y}{\text{Arg max}} \ \mathsf{P}(\text{Класс} = y | X_1, X_2, ..., X_p)$$

Пользуясь формулой Байеса, последнее соотношение может быть переписано в виде

$$y^* = \underset{y \in Y}{\text{Arg max}} \ \mathsf{P}(\text{Класс} = y | X_1, X_2, ..., X_p) =$$

$$= \underset{y \in Y}{\text{Arg max}} \frac{\mathsf{P}(X_1, X_2, \ldots, X_p | \text{Класс} = y)\mathsf{P}(\text{Класс} = y)}{\mathsf{P}(X_1, X_2, \ldots, X_p)}.$$

Так как вероятность, стоящая в знаменателе, не зависит от y, то она одинакова для всех y и не влияет на максимизацию, а значит

$$y^* = \underset{y \in Y}{\text{Arg max}} \left(\mathsf{P}(X_1, X_2, \ldots, X_p | \text{Класс} = y)\mathsf{P}(\text{Класс} = y)\right).$$

С точки зрения теории все хорошо, задача вроде бы решена, но что на практике? Проблема заключается в том, что на практике мы не знаем совместного распределения. Понятно, что оценить вероятности $\mathsf{P}(\text{Класс} = y)$ достаточно легко – будем оценивать частоту встречаемости классов в тренировочном наборе данных, а вот оценить условные вероятности $\mathsf{P}(X_1, X_2, \ldots, X_p|\text{Класс} = y)$ не получится – их слишком много. Для того, чтобы оценить их достаточно «хорошо», нужно каждый случай пронаблюдать несколько раз – это, конечно же, невозможно. Здесь и помогает заявленная в названии классификатора «наивность».

Итак, наивный байесовский классификатор работает в предположении **условной независимости предикторов при условии данной метки класса**. Математически это означает следующее:

$$\mathsf{P}(X_1, X_2, \ldots, X_p|\text{Класс} = y) = \mathsf{P}\left(X_1|\text{Класс} = y\right) \cdot \mathsf{P}\left(X_2|\text{Класс} = y\right) \cdot \ldots \cdot$$

$$\mathsf{P}\left(X_p|\text{Класс} = y\right) = \prod_{i=1}^{p} \mathsf{P}\left(X_i|\text{Класс} = y\right).$$

Несмотря на такое, казалось бы, существенное ограничение, оказывается, что байесовский классификатор достаточно хорошо справляется со своей задачей. Даже больше, именно наивный байесовский классификатор до недавнего времени был основным алгоритмом классификации писем на «спам» и не «спам» в большинстве современных почтовых сервисов.

Замечание 3.3.2 *Отметим, что в контексте классификации, скажем, писем или текстов, «наивное» ограничение означает следующее: разные слова в тексте на одну и ту же тему появляются независимо друг от друга. Еще раз вдумайтесь, насколько это, честно говоря, далеко от действительности.*

Где же нам помогает предположение о независимости? Конечно, в оценке условных вероятностей $\mathsf{P}(X_i|\text{Класс} = y)$. Оцениваются эти вероятности как обычно – частотой, подробнее об этом мы скажем чуть позже. В итоге, в предположении наивности, классификатор принимает следующий вид:

$$y^* = \operatorname*{Arg\,max}_{y \in Y} \left(\mathsf{P}(\text{Класс} = y) \prod_{i=1}^{p} \mathsf{P}\left(X_i|\text{Класс} = y\right) \right).$$

Это еще не все. В случае, если объем тренировочных данных велик, а вероятности (или их оценки) малы, то результат перемножения компьютерными средствами, имеющими ограниченную точность, приведет нас к чему-то вроде нуля. Для решения этой проблемы воспользуемся переходом к натуральному логарифму. Так как натуральный логарифм – возрастающая функция, то

под ее действием максимумы переходят в максимумы, а значит класификатор можно представить в следующем виде:

$$y^* = \underset{y \in Y}{\operatorname{Arg\,max}} \left(\ln \left(\mathsf{P}(\text{Класс} = y) \prod_{i=1}^{p} \mathsf{P}\left(X_i | \text{Класс} = y\right) \right) \right),$$

или, пользуясь свойствами логарифма, в виде

$$y^* = \underset{y \in Y}{\operatorname{Arg\,max}} \left(\ln \mathsf{P}(\text{Класс} = y) + \sum_{i=1}^{p} \ln \mathsf{P}\left(X_i | \text{Класс} = y\right) \right).$$

Почти все, остался последний вопрос: а как же вернуться к вероятностям отнесения к классам? Достаточно просто: нужно избавиться обратно от логарифмов и нормировать результаты. Для каждого класса $y \in Y$ вычислим значение

$$F(y) = \ln \mathsf{P}(\text{Класс} = y) + \sum_{i=1}^{p} \ln \mathsf{P}\left(X_i | \text{Класс} = y\right).$$

Тогда, чтобы получить вероятность отнесения объекта X к какому-то классу $y^* \in Y$, остается составить отношение следующего вида:

$$\mathsf{P}(\text{Класс} = y^* | X_1, X_2, ..., X_p) = \frac{e^{F(y^*)}}{\sum\limits_{y \in Y} e^{F(y)}}.$$

Замечание 3.3.3 *Для сокращения числа операций с экспонентой, часто бывает полезным следующее преобразование:*

$$\mathsf{P}(\textit{Класс} = y^* | X_1, X_2, ..., X_p) = \frac{1}{e^{-F(y^*)} \sum\limits_{y \in Y} e^{F(y)}} = \frac{1}{1 + \sum\limits_{y \in Y,\ y \neq y^*} e^{F(y) - F(y^*)}},$$

где в последней сумме индекс пробегает все метки класса, кроме y^*.

Теперь мы готовы сформулировать алгоритм построения классификатора на конкретной выборке.

3.3.1 Алгоритм построения классификатора на конкретной выборке

Пусть имеется тренировочный набор данных $X = (x_1, x_2, ..., x_n)$ объема n,

$$x_i = (x_{i1}, x_{i2}, ..., x_{ip}), \quad i \in \{1, 2, ..., n\},$$

причем каждому объекту x_i соответсвует отклик $y_i \in Y$. Пусть требуется классифицировать новый объект z со значениями предикторов $(z_1, z_2, ..., z_p)$. Тогда объекту z назначается любой класс из набора

$$\underset{y \in Y}{\operatorname{Arg\,max}} \left(\ln \mathsf{P}(\text{Класс} = y) + \sum_{i=1}^{p} \ln \mathsf{P}\left(X_i | \text{Класс} = y\right) \right),$$

где в качестве оценки вероятности $\mathsf{P}(\text{Класс} = y)$ берется частота встречаемости соответствующего класса в тренировочном наборе данных, то есть

$$\mathsf{P}(\text{Класс} = y) = \frac{\sum\limits_{i=1}^{n} \mathsf{I}(y_i = y)}{n} =$$

$$= \frac{\text{количество тренировочных элементов с меткой класса } y}{\text{общее количество тренировочных элементов}},$$

а в качестве оценки вероятности $\mathsf{P}\left(X_i | \text{Класс} = y\right)$ – частота встречи значения предиктора z_i среди тренировочных данных класса y, то есть

$$\mathsf{P}(X_i | \text{Класс} = y) = \frac{\sum\limits_{k=1}^{n} \mathsf{I}(x_{ki} = z_i, y_k = y)}{\sum\limits_{k=1}^{n} \mathsf{I}(y_k = y)}$$

$$= \frac{\text{количество тренировочных элементов класса } y, \text{ у которых } X_i = z_i}{\text{количество тренировочных элементов класса } y}.$$

Замечание 3.3.4 *Отметим отдельно, что введенные выше оценки вероятностей являются, по своей сути, оценками максимального правдоподобия. Доказательство этого достаточно технично, оно может быть найдено в дополнительных материалах.*

3.4 Пример построения классификатора

Чтобы детально разобрать описанный алгоритм, рассмотрим пример построения наивного байесовского классификатора.

Предположим, что вы собрали данные о футбольных матчах, проводимых на спортивной площадке рядом с вашим домом. В день проведения матча вы фиксировали погоду (дождливо, пасмурно или солнечно), среднюю температуру (холодно, прохладно или жарко), влажность (влажно или сухо) и наличие ветра (либо ветер есть, либо безветрие). Данные представлены в следующей таблице:

Погода	**Ср. температура**	**Влажность**	**Наличие ветра**	**Игра**
солнечно	жарко	влажно	нет	нет
солнечно	жарко	влажно	да	нет
пасмурно	жарко	влажно	нет	да
дождливо	прохладно	сухо	нет	да
дождливо	холодно	сухо	нет	да
дождливо	холодно	сухо	да	нет
пасмурно	прохладно	влажно	да	да
пасмурно	прохладно	влажно	нет	нет
пасмурно	прохладно	влажно	да	да

.

Как, используя наивный байесовский классификатор, отнести новое наблюдение z с предикторами $(\text{пасмурно}, \text{холодно}, \text{влажно}, \text{да})$ к одному из двух классов?

Итак, у нас четыре предиктора: X_1 – погода, принимающий 3 различных значения, X_2 – средняя температура, тоже с тремя различными значениями, X_3 – влажность, принимающий 2 различных значения и X_4 – наличие ветра, тоже два различных значения, а также бинарный отклик Y – игра: либо состоялась, либо нет. Новому объекту z, согласно алгоритму, нужно присвоить любой класс из набора

$$\underset{y \in Y}{\operatorname{Arg\,max}} \left(\ln \mathsf{P}(\text{Класс} = y) + \sum_{i=1}^{4} \ln \mathsf{P}\left(X_i | \text{Класс} = y\right) \right).$$

Рассмотрим подробно вычисления для случая, когда игра состоялась, то есть для класса «да». Вычислим для начала

$$\mathsf{P}(\text{Класс} = \text{да}) = \frac{\text{количество тренировочных элементов с меткой класса «да»}}{\text{общее количество тренировочных элементов}}.$$

Легко понять, что у нас всего 9 тренировочных данных, среди которых лишь 5 имеют отклик «да», значит

$$\mathsf{P}(\text{Класс} = \text{да}) = \frac{5}{9}.$$

Теперь вычислим

$$\mathsf{P}(X_1 | \text{Класс} = \text{да}) = \mathsf{P}(\text{Погода} | \text{Класс} = \text{да}).$$

Так как у тестового объекта значение предиктора X_1 – погода – это пасмурно, то нам, согласно алгоритму, нужно разделить количество тренировочных

объектов, имеющих отклик «да» и значение предиктора погода – пасмурно, на общее количество тренировочных объектов класса «да». Тем самым, получаем

$$\mathsf{P}(X_1|\text{Класс} = \text{да}) = \mathsf{P}(\text{Погода} = \text{пасмурно}|\text{Класс} = \text{да}) = \frac{3}{5}.$$

Аналогично, так как у тестового объекта значение предиктора X_2 – средняя температура – это холодно, то нам, согласно алгоритму, нужно разделить количество тренировочных объектов, имеющих отклик «да» и значение предиктора средняя температура – холодно, на общее количество тренировочных объектов класса «да». Тем самым, получаем

$$\mathsf{P}(X_2|\text{Класс} = \text{да}) = \mathsf{P}(\text{Ср. температура} = \text{холодно}|\text{Класс} = \text{да}) = \frac{1}{5}.$$

Продолжая аналогичные рассуждения, получаем

$$\mathsf{P}(X_3|\text{Класс} = \text{да}) = \mathsf{P}(\text{Влажность} = \text{влажно}|\text{Класс} = \text{да}) = \frac{3}{5}$$

и

$$\mathsf{P}(X_4|\text{Класс} = \text{да}) = \mathsf{P}(\text{Наличие ветра} = \text{да}|\text{Класс} = \text{да}) = \frac{2}{5}.$$

Итого, для класса «да» значение исследуемого выражения равно

$$\ln\frac{5}{9} + \ln\frac{3}{5} + \ln\frac{1}{5} + \ln\frac{3}{5} + \ln\frac{2}{5} \approx -4.135.$$

Аналогичные вычисления для класса «нет» приводят к значению

$$\ln\frac{4}{9} + \ln\frac{1}{4} + \ln\frac{1}{4} + \ln\frac{3}{4} + \ln\frac{2}{4} \approx -4.564.$$

Так как первое из полученных выражений больше, чем втрое, то класс, к которому следует отнести тестовое наблюдение – класс «да». Значит, игра в случае пасмурной, влажной, ветреной погоды и холодной средней температуры скорее состоится, нежели нет. Итак, вопрос классификации решен. Но насколько уверен наш классификатор в своем решении? Давайте выясним это. Итак, в обозначениях из лекции,

$$F(\text{да}) \approx -4.135, \quad F(\text{нет}) \approx -4.564,$$

тогда

$$\mathsf{P}(\text{Класс} = \text{да}|X_1, X_2, X_3, X_4) \approx \frac{1}{1 + e^{-4.564+4.135}} \approx 0.606$$

и, тем самым,

$$\mathsf{P}(\text{Класс} = \text{нет}|X_1, X_2, X_3, X_4) = 1 - \mathsf{P}(\text{Класс} = \text{да}|X_1, X_2, X_3, X_4) \approx 0.394.$$

Видно, что классификатор не очень-то уверенно относит наблюдение к классу «да».

3.5 Классификация писем. Сглаживание по Лапласу

Итак, до сих пор мы считали, что количество предикторов у любого объекта заранее известно. Такая ситуация, однако, бывает не всегда. Давайте вернемся к задаче классификации писем на спам и не спам и рассмотрим ее подробнее. Что выступает в качестве предикторов в письме? Понятно, что слова. Но разве мы можем изначально сказать, сколько слов будет в том или ином письме? Вряд ли.

Наверное разумнее всего предположить, что количество предикторов в каждом письме равно количеству входящих в него слов. Значения же все предикторы принимают из набора V – некоторого словаря слов, сформированного на основе тренировочных данных. Итак, словарь представляет из себя большую таблицу с тремя колонками: первая колонка отвечает за само слово, вторая – за количество вхождений этого слова в письма, отнесенные к категории «спам», а третья – за количество вхождений этого слова в письма, отнесенные к категории «не спам».

Сразу рассмотрим пример. Нашими тренировочными данными будут три письма с текстом:

- Win a million rubles – «спам»;
- Ruble drops again – «не спам»;
- A million ways to get rich – «спам».

Сформируем упомянутый словарь V. Отметим, что на практике применяются методы обработки языка, поэтому слова rubles и ruble будем отождествлять, а частицы и артикли будем опускать. Получим следующую таблицу.

Слово (X)	**«спам»**	**«не спам»**
win	1	0
million	2	0
ruble	1	1
again	0	1
drops	0	1
ways	1	0
get	1	0
rich	1	0

Легко оценить вероятности встретить какое-то слово в зависимости от

того, попало письмо в «спам» или нет. Делается это, как и раньше, например:

$$\mathsf{P}(X = \text{win}|\text{Класс} = \text{спам}) = \frac{1}{7},$$

так как слово win встретилось в спам-письмах один раз, а всего в спам-письмах встретилось 7 слов (с учетом повторений). К сожалению, такая оценка может привести и к довольно странным результатам. Например,

$$\mathsf{P}(X = \text{win}|\text{Класс} = \text{не спам}) = 0,$$

а это значит, что письмо, содержащее слово win, никогда не будет отнесено к категории «не спам», даже если все остальные слова в письме – слова again и drops – явные индикаторы того, что письмо спамом не является. Выход – применить так называемое сглаживание по Лапласу – предположить, что мы видели каждое слово на один раз больше. Тогда

$$\mathsf{P}(X|\text{Класс} = y) = \frac{1 + \text{количество слов } X \text{ в классе } y}{|V| + \text{количество слов в классе } y}, \quad y \in \{\text{спам}, \text{не спам}\}$$

где $|V|$ – количество слов в словаре. В этом случае

$$\mathsf{P}(X = \text{win}|\text{Класс} = \text{спам}) = \frac{1+1}{8+7} = \frac{2}{15},$$

$$\mathsf{P}(X = \text{win}|\text{Класс} = \text{не спам}) = \frac{0+1}{8+3} = \frac{1}{11}.$$

Как классифицировать письмо с текстом «Win a ruble to get rich»? Ровно так, как это делалось и раньше. Вычислим для начала $F(\text{спам})$:

$$F(\text{спам}) = \ln \mathsf{P}(\text{спам}) + \ln \mathsf{P}(X = \text{win}|\text{Класс} = \text{спам}) +$$

$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Класс} = \text{спам}) + \ln \mathsf{P}(X = \text{get}|\text{Класс} = \text{спам}) +$$

$$+ \ln \mathsf{P}(X = \text{rich}|\text{Класс} = \text{спам}) =$$

$$= \ln \frac{2}{3} + \ln \frac{2}{15} + \ln \frac{2}{15} + \ln \frac{2}{15} + \ln \frac{2}{15} \approx -8.465.$$

Теперь вычислим $F(\text{не спам})$:

$$F(\text{не спам}) = \ln \mathsf{P}(\text{не спам}) + \ln \mathsf{P}(X = \text{win}|\text{Класс} = \text{не спам}) +$$

$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Класс} = \text{не спам}) + \ln \mathsf{P}(X = \text{get}|\text{Класс} = \text{не спам}) +$$

$$+ \ln \mathsf{P}(X = \text{rich}|\text{Класс} = \text{не спам}) =$$

$$= \ln \frac{1}{3} + \ln \frac{1}{11} + \ln \frac{2}{11} + \ln \frac{1}{11} + \ln \frac{1}{11} \approx -9.997.$$

В итоге, письмо следует отнести к классу «спам».

Остался последний момент. А что, если в письме встретилось слово, которого нет в словаре? Есть два варианта: либо это слово можно выкинуть из рассмотрения, то есть пропустить, либо в момент классификации «переучить» построенный классификатор, используя снова сглаживание по Лапласу:

$$\mathsf{P}(X|\text{Класс} = y) = \frac{1 + \text{количество слов } X \text{ в классе } y}{|V| + r + \text{количество слов в классе } y}, \quad y \in \{\text{спам}, \text{не спам}\},$$

где r – количество слов в письме, которых нет в словаре.

Для примера, классифицируем письмо с текстом «Get rich with ruble». Мы видим, что слова with нет в нашем словаре. Применим сглаживание по Лапласу, учитывая, что $r = 1$, тогда

$$F(\text{спам}) = \ln \mathsf{P}(\text{спам}) + \ln \mathsf{P}(X = \text{get}|\text{Класс} = \text{спам}) +$$

$$+ \ln \mathsf{P}(X = \text{rich}|\text{Класс} = \text{спам}) + \ln \mathsf{P}(X = \text{with}|\text{Класс} = \text{спам}) +$$

$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Класс} = \text{спам}) =$$

$$= \ln \frac{2}{3} + \ln \frac{2}{16} + \ln \frac{2}{16} + \ln \frac{1}{16} + \ln \frac{2}{16} \approx -9.416.$$

Теперь вычислим $F(\text{не спам})$:

$$F(\text{не спам}) = \ln \mathsf{P}(\text{не спам}) + \ln \mathsf{P}(X = \text{get}|\text{Класс} = \text{не спам}) +$$

$$+ \ln \mathsf{P}(X = \text{rich}|\text{Класс} = \text{не спам}) + \ln \mathsf{P}(X = \text{with}|\text{Класс} = \text{не спам}) +$$

$$+ \ln \mathsf{P}(X = \text{ruble}|\text{Класс} = \text{не спам}) =$$

$$= \ln \frac{1}{3} + \ln \frac{1}{12} + \ln \frac{1}{12} + \ln \frac{1}{12} + \ln \frac{2}{12} \approx -10.345.$$

Значит, письмо следует отнести к категории «спам».

4 Заключение

Итак, в этой лекции мы начали изучать подходы к решению задачи классификации, изучив метрические классификаторы k-NN и взвешенный k-NN, а также вероятностный наивный байесовский классификатор. Кроме того, мы поняли, как важно правильно делить набор данных на тестовый и тренировочный, как оценивать модель, а также что знание расстояний – это еще не все. В дальнейшем мы изучим и другие методы и подходы к классификации, ну а пока что все. До новых встреч!