Введение в машинное обучение

Национальный исследовательский университет "Высшая школа экономики"
Yandex School of Data Analysis

Неофициальный конспект по курсу.

12 апреля 2021 г.

Содержание

1 Метрические методы

1.1 Определение

Следующие методы, которые мы рассмотрим, будут **метрические методы**, которые используют функции *расстояния* или *метрики* в пространстве объектов.

Исходная идея заключается в предположении, что в практических задачах часто встречаются зависимости, которые непрерывны (хотя это справедливо скорее для задач регрессии, уточнения описаны ниже):

- **Гипотеза непрерывности** (для регрессии):

  близким объектам соответствуют близкие объекты.

- **Гипотеза компактности** (для классификации):

  близкие объекты, как правило, лежат в одном классе.

Впрочем, далее в этом документе мы будем рассматривать *метрические методы* именно для *задач классификации*, регрессия будет потом.

Теперь зададимся вопросом, как формализовать эту «близость»?

- **Формализация понятия «близости»:**

  Задана функция расстояния $\rho : X \times X \to [0, \infty)$.

По сути эта функция от пары объектов, которая паре ставит соответствие — неотрицательное число.

Часто также накладывают требование, чтобы это была метрика в пространстве объектов, то есть чтобы она была и симметричной, и выполнялось неравенство треугольника. Однако формально в методах нигде не используется предположение о том, что это метрика, поэтому, в общем-то, и функции расстояния, не являющиеся метриками, нам тоже подходят.

1.2 Примеры расстояний в задачах

1.2.1 Евклидово пространство и его обобщение

Самым известным примером расстояния, наверное, является **евклидово расстояние** в признаковом пространстве. Также ее эту конструкцию можно обобщить (добавить иную степень, или ввести веса признаков), и мы получим два следующих варианта:

$$\rho(x, x_i) = \left(\sum_{j=1}^{n} |x^j - x_i^j|^2 \right)^{\frac{1}{2}} \qquad \rho(x, x_i) = \left(w_j \sum_{j=1}^{n} |x^j - x_i^j|^p \right)^{\frac{1}{p}}$$

$x = (x^1, \ldots, x^n)$ — вектор признаков объекта x.
$x_i = (x_i^1, \ldots, x_i^n)$ — вектор признаков объекта x_i.

Впрочем, обобщенный вариант требует дополнительных усилий, неких процедур, которые будут обучать эти дополнительные параметры.

1.2.2 Редакторское расстояние Левенштейна

Далеко не всегда, не во всех задачах, именно использование евклидовой метрики осмысленно, и есть масса задач, где у нас исходно *нет признаковых описаний объектов*, и гораздо проще пользоваться не признаками, а *непосредственно* мерять расстояние между объектами.

Примеры таких объектов — это *символьные строки*. Это могут быть тексты естественного языка, это могут быть нуклеотидные или аминокислотные последовательности, которые изучаются в биоинформатике.

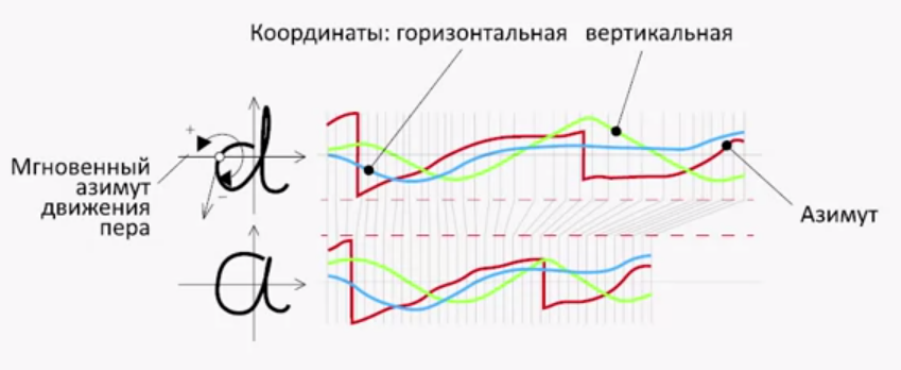
В таких задачах, чтобы сравнить текстовые строки, вообще говоря, произвольной длины, часто используется так называемое **редакторское расстояние Левенштейна** — это количество вставок и замен символов, которое необходимо произвести, чтобы привести одну строку в другую:

CTGGGGTAAAAGGTCCCTTAGCC....TTTAGAAAAA..GGGCCATTAGGAAATTGC
CTGGGACTAAA.....CCTTAGCCTATTTACAAAAATGGGCCATTAGG.....TTGC

Описать две сравниваемых строки признаками было бы намного труднее, чем посчитать подобное расстояние, которое к тому же достаточно эффективно считается.

1.2.3 Энергия сжатий и растяжений

Следующий пример уже между объектами еще *более сложной природы*: допустим, мы хотим сравнить 2 подписи, которые сделали люди специальным пером электронным, которое запоминает координаты кончика пера ну и еще азимут движения пера. Значит, мы как подпись имеем 3 кривые, 3 временных ряда, и нам нужно сравнивать такие временные ряды:



На картинке представлен пример такого сравнения, когда 2 похожие буквы сравниваются, и мы видим, что эти 3 кривые они очень *похожи* с точностью до каких-то *сжатий, растяжений, выравниваний*.

Можно придумать модель расстояния между такими траекториями, как *модель пружинок*, которые слегка сжимаются или растягиваются между маленькими локальными кусочками этих кривых, и посчитать энергию сжатий и растяжений пружинок — это и будет *расстояние* между этими объектами.

Опять же, расстояние посчитать легко, а придумать здесь признаки было бы значительно сложнее.

1.3 Обобщенный метрический классификатор

Допустим, что кто-то нам *уже дал* функцию расстояния и теперь перед нами стоит задача — *научиться классифицировать объекты.*

Самый простой алгоритм классификации вообще из всех алгоритмов, известных в машинном обучении, — это **алгоритм ближайшего соседа**. Чтобы классифицировать объект x, возьмем все объекты обучающей выборки и найдем среди них ближайший к x, и отнесем x к тому же классу, к которому принадлежит этот объект.

Конечно, использовать для классификации только одного ближайшего соседа — это не очень удачная идея, лучше взять окрестность, учесть несколько ближайших соседей, как-то по ним усреднить. И поэтому возникает такая **обобщенная идея**: *упорядочим все объекты обучающей выборки по возрастанию расстояний до нашего классифицированного объекта* x:

- Для произвольного $x \in X$ отранжируем объекты $x_1, \ldots, x_l$:

$$\rho(x, x^{(1)} \leqslant \rho(x, x^{(2)} \leqslant \ldots \leqslant \rho(x, x^{(l)},$$

$x^{(i)}$ — i-й сосед объекта x среди $x_1, \ldots, x_l$;

$y^{(i)}$ — ответ на i-м соседе объекта x.

Дополнительно введем *вес* i-того соседа или его *ценность, полезность* при классификации объекта x. Будем обозначать этот вес $w(i, x)$, то есть это функция от объекта и от порядкового номера соседа.

Соответственно, если мы *усредним* в каждом классе значение этих весов, то мы получим в итоге оценку близости объекта к классу: мы ее обозначаем Γ с индексом y от объекта x.Индекс y показывает, что такая оценка она относится к определенному классу. *Таким образом получаем*:

- **Метрический алгоритм классификации:**

$$a(x, X^l) = arg \max_{y \in Y} \underbrace{\sum_{i=1}^{l} [y^{(i)} = y] w(i, x)}_{\Gamma_y(x)},$$

$w(i, x)$ — вес, *оценка сходства объекта* x с его i-м *соседом.*
Неотрицательная, не возрастающая по i.

$\Gamma_y(x)$ — *оценка близости объекта* x к *классу* y.

Самый естественный принцип классификации — это отнести объект x к тому классу, для которого эта оценка близости максимальна, что мы и делаем выше.

Соответственно, есть масса частных случаев этой общей формы метрического алгоритма классификации.

1.3.1 Метод k ближайших соседей (k nearest neighbors, kNN)

Метод ближайших соседей или его частный самый простой случай — **метод первого ближайшего соседа**:

$w(i,x) = [i \leqslant k]$

$w(i,x) = [i \leqslant 1]$

Преимущество этого метода в том, что он очень просто реализуется. Еще говорят, что это *«ленивое»* обучение, потому что, собственно, никакого обучения здесь не происходит — мы просто запоминаем выборку, а после классификация заключается в поиске ближайших соседей:

- простота реализации (lazy learning);
- параметр k можно оптимизировать по критерию скользящего контроля (leave-one-out):

$$\text{LOO}(k, x^l) = \sum_{i=1}^{l} \Big[a(x_i;\ X^l \backslash \{x_i\},\ k) \neq y_i \Big] \ \to \min_q$$

  Дело в том, что если мы берем *слишком мало* соседей, то среди них могут оказаться какие-то случайные шумовые выбросы. Если мы берем *слишком много* соседей, то мы построим слишком простой классификатор, который в любой точке пространства по сути будет выдавать одно и то же значение. И где-то *посередине* находится тот классификатор, который действительно будет прилично работать.

Проблемы:

- Неоднозначность классификации $\Gamma_y(x) = \Gamma_s(x),\ y \neq s$.

  То есть мы можем столкнуться с ситуацией, когда *для двух классов оценки* степени близости объекта к классу *совпадают*, и мы поэтому не понимаем, к какому из этих двух классов отнести наш объект. (Например, k четное и половина соседей относятся к одному классу, вторая половина к другому.)

- Мы нигде не учитываем сами значения расстояний.

  После того как мы отранжировали объекты обучения по возрастанию расстояний, у нас остались только порядковые номера этих объектов. Может возникнуть ситуация, когда мы, например, классифицируем по пяти ближайшим соседям: четыре соседа находятся действительно близко от объекта, а пятый где-то совсем далеко. Конечно, в таком случае его надо было бы учесть с меньшим весом.

Пример зависимости LOO от числа соседей

Пару слов о том, как происходит *оптимизация* параметра числа ближайших соседей и почему *обязательно* надо использовать функционалы, которые сам классифицируемый объект изымает из обучающей выборки.

Нам этом графике приведены две кривые: это уже известная нам выборка цветков ириса, и использованы два алгоритма.

Первый алгоритм — это классификация методом k ближайших соседей. И когда классифицируется некий объект обучающей выборки, сам же он является своим собственным соседом, и это дает синюю кривую.

Если же сам объект исключается из числа своих соседей, получается красная кривая.

Видно, что синяя кривая нам дает *оптимистически смещенную оценку числа ошибок* на контрольных выборках. Нам кажется, что оптимально использовать одного соседа, но если объект сам же является своим собственным соседом, то понятно, что это некий самообман.

И здесь же на красной кривой мы видим, что в данной задаче оптимально брать где-то от 30 до 60 соседей, а синяя кривая нам дает *ложный* ответ, что надо пользоваться методом первого ближайшего соседа. Этот метод практически никогда не работает надежно.

1.3.2 Метод окна Парзена

Пойдем дальше и посмотрим, как можно еще обобщить *метод ближайших соседей*, чтобы справиться вот с теми двумя недостатками, которые были отмечены ранее.

Мы вольны выбирать функцию весов соседей, как мы хотим. Давайте сделаем так, чтобы вес соседа убывал по мере возрастания расстояния до него.

Введем два новых понятия: это **ядро** и **ширина окна**:

$w(i,x) = K\Big(\frac{\rho(x,x^{(i)}}{h}\Big)$, где h — ширина окна,

$K(r)$ — ядро, не возрастает и положительно на $[0,1]$.

То есть мы функцию веса зададим как такую конструкцию — ядро от расстояния поделить на ширину окна — и получаем взвешенную функцию, которая *придает меньшие веса тем соседям, которые находятся дальше*. Этот метод называется **метод окна Парзера**.

У этого метода есть два варианта:

- Метод парзеновского окна *фиксированной ширины*:

$$A(x;\ X^l, h, K) = arg\ \max_{y\in Y} \sum_{i=1}^{l} [y_i = y] K\Big(\frac{\rho(x, x^{(i)})}{h}\Big)$$

  Но это не очень хорошо работает в тех задачах, где плотность распределения объектов неравномерна и есть области пространства, где объекты лежат густо, а в других областях объекты лежат более разреженно, и в таком случае хочется в первой области использовать более узкое окно, а во втором случае — более широкое.

- Метод парзеновского окна *переменной ширины*:

$$A(x;\ X^l, k, K) = arg\ \max_{y\in Y} \sum_{i=1}^{l} [y_i = y] K\Big(\frac{\rho(x, x^{(i)})}{\rho(x, x^{k+1})}\Big)$$

  Решает предыдущую проблему, и в таком случае в качестве ширины окна берется расстояние до (k + 1) соседа. Это означает, что *для k-ближайших соседей будут ненулевые веса*. И таким образом решается эта проблема неравномерности распределения объектов в пространстве.

Оптимизация параметров может происходить по тому же критерию LOO:

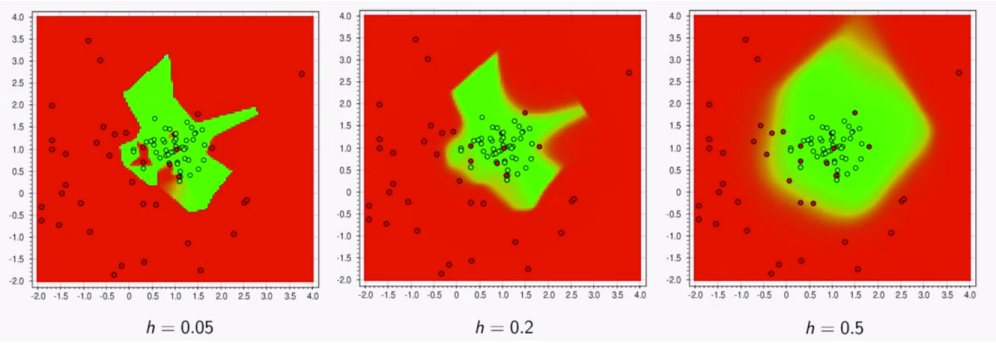
- выбор ширины окна h или числа соседей k
- выбор ядра K при этом *слабо влияет* на качество классификации

Пример задачи с парзеновским окном фиксированной ширины h

Пусть у нас есть синтетическая двумерная выборка, объекты в которой принадлежат к двум классам $Y = \{-1, +1\}$.

$$a(x) = arg \max_{y \in Y} \Gamma_y(x) = sign(\Gamma_{+1}(x) - \Gamma_{-1}(x))$$

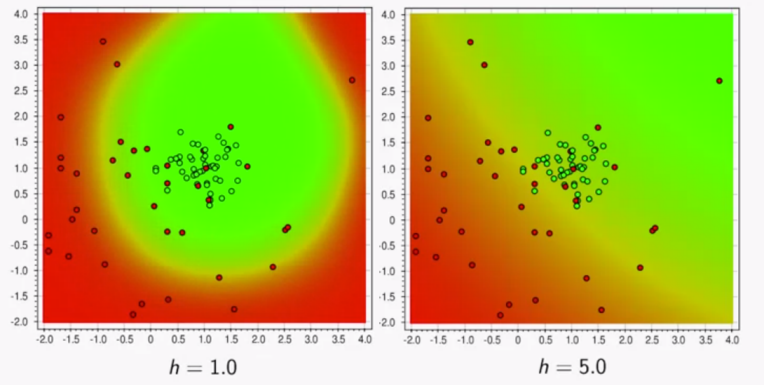
То есть мы нашу функцию метрического классификатора записали как разность оценка близости объекта x к $(+1)$ классу и оценки близости к (-1) классу. Соответственно, на графике мы это изобразим разными цветами:



Серия графиков сейчас показывает, как меняется фактически граница между классами по мере увеличения ширины окна. Когда ширина окна была 0.1, то фактически работал метод ближайшего соседа. Это слишком узкое окно.

Далее, когда мы его потихоньку увеличиваем, граница между классами размазывается, становится более похожей на окружность. И на самом деле окружность — это оптимальная граница между этими двумя классами. Мы их на самом деле так синтезировали, мы их так задумали. В данном случае облако точек зеленого цвета — класс $(+1)$ — распределен в сферической области.

Соответственно, алгоритм, который работал только по первому ближайшему соседу, очевидно, плохо восстанавливает эту сферическую область. С другой стороны, если сделать окно слишком широким, то в какой-то момент наш алгоритм классификации уже начинает вырождаться:



1.3.3 Метод потенциальных функций

$w(i,x) = \gamma^{(i)} K\Big(\frac{rho(x,x^{(i)})}{h^{(i)}}\Big)$

Более простая запись (без ранжирования объектов):

$$A(x;\ X^l) = arg\ \max_{y \in Y} \sum_{i=1}^{l} [y_i = y] \gamma_i K\Big(\frac{\rho(x, x^{(i)})}{h_i}\Big),$$

где γ_i — веса объектов, $\gamma_i \geqslant 0$, $h_i > 0$.

Он очень *похож* на метод парзеновского окна, но он меняет наш взгляд на то, что относительно чего расположено. Вот в методе парзеновского окна мы можем представить себя, стоящими в точке x (это точка классифицируемого объекта), и мы смотрим на *окрестность* этой точки (с учетом ширины окна) и подсчитываем число объектов, попавших в эту окрестность, а в итоге принимаем решение о классе.

Можно считать, что функция расстояния *симметрична*, поэтому мы можем представить себя, стоящими по очереди в каждом обучающем объекте x_i, и думать, что каждый объект обучающей выборки распространяет вокруг себя *потенциал* $gamma_i$ своего класса. Это прямая аналогия с **электростатикой**:

γ_i — величина «заряда» в точке x_i;

h_i — «радиус действия» потенциала с центром в точке x_i;

y_i — знак «заряда» (в случае двух классов $Y = \{-1, +1\}$;

В электростатике $K(r) = \frac{1}{r}$ или $\frac{1}{r+a}$,
хотя в задачах классификации таких ограничений для K нет, мы все еще можем подобрать любую функцию.

То есть можно представить представить себе, что точки — это *заряженные частицы* (положительные и отрицательные), то каждая точка *распространяет* вокруг себя электрическое поле. И фактически знак этого поля в произвольной точке пространства x как раз и покажет, *к чему мы ближе*: к положительно заряженным объектам или к отрицательно заряженным объектам.

Но этот метод хорош тем, что в нем теперь появляются параметры, и с каждым объектом обучающей выборки мы можем связать даже *два числа*: γ_i и h_i. Таким образом у нас получается классификатор, *существенно обогащенный параметрами*. Теперь можно настраивать эти параметры по обучающей выборке. Собственно, *алгоритм перестает быть ленивым*. Нам не только надо хранить обучающую выборку, но и настроить эти параметры.

Настройка параметров метода потенциальных функций
Пусть имеем два класса: $Y = \{-1, +1\}$.

$$a(x) = arg\ \max_{y \in Y} \Gamma_y(x) = sign(\Gamma_{+1}(x) - \Gamma_{-1}(x)) = sign \sum_{i=1}^{l} \gamma_i y_i K\Big(\frac{\rho(x, x^{(i)})}{h_i}\Big).$$

Сравним с линейной моделью классификации:

$$a(x) = sign \sum_{i=1}^{n} \gamma_i f_j(x).$$

- функции $f_j(x) = y_j K\left(\frac{1}{h_j}\rho(x, x_j)\right)$ — признаки объекта x
- γ_j — веса линейного классификатора
- $n = l$ — число признаков равно числу объектов обучения.

То есть на самом деле классификатор метода потенциальных функций — *это разновидность линейного классификатора.*

О линейных классификаторах мы будем говорить позднее, и изучим подробно методы, которыми можно настраивать веса в линейных классификаторах. Сейчас для нас принципиально важен факт, что здесь *в качестве признаков* мы фактически *используем функции близости* между классифицируемым объектом и объектом обучающей выборки.

То есть каждый обучающий объект *индуцирует* признак, и признаков здесь ровно столько, *сколько объектов обучающей выборки.* Это глубинное сходство между метрическим классификатором (методом потенциальных функций) и линейным классификатором дает очень большую свободу при использовании линейных моделей классификации.

Мы понимаем, что признаки — это могут быть *исходно имеющиеся* признаки, а можно в качестве признаков использовать *близость до объектов* обучающей выборки. Это очень интересная возможность, которая в разных задачах может принести успех и повышение точности классификации.

2 Резюме

- **Метрические классификаторы** — одни из самых простых. Качество классификации определяется качеством метрики.
- **Что можно обучать:**
  - число ближайших соседей k или ширину окна h;
  - веса объектов;
  - набор эталонов (prototype selection);
  - метрику (distance learning, similarity learning);
  - веса признаков;
  - функцию ядра $K(r)$.