&1 Логические методы классификации: Решающие Деревья

> Логические методы делают классификацию объектов на основе простых правил, благодаря чему являются интерпретируемыми и легкими в реализации. При объединении в композицию логические модели позволяют решать многие задачи с высоким качеством. Основной класс логических алгоритмов — решающие деревья. **Решающие деревья** - важный класс методов машинного обучения. Эти методы предназначены исходно для решения задач классификации, а также есть их специальные варианты, приспособленные для решения задач регрессии.

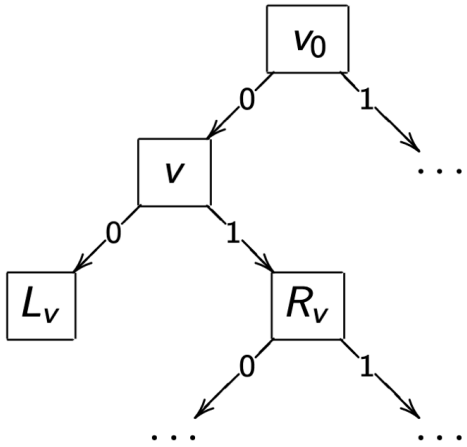
Принцип работы

- Решающие деревья возникли как попытка формализовать тот способ мышления, который используют люди при принятии решений. (Это хорошо иллюстрируется логикой работы врача, когда он говорит с пациентом и задает один за другим уточняющие вопросы для того, чтобы поставить диагноз. Аналогия всей информации, которой пользуется врач риводит к конструкции решающих деревьев)

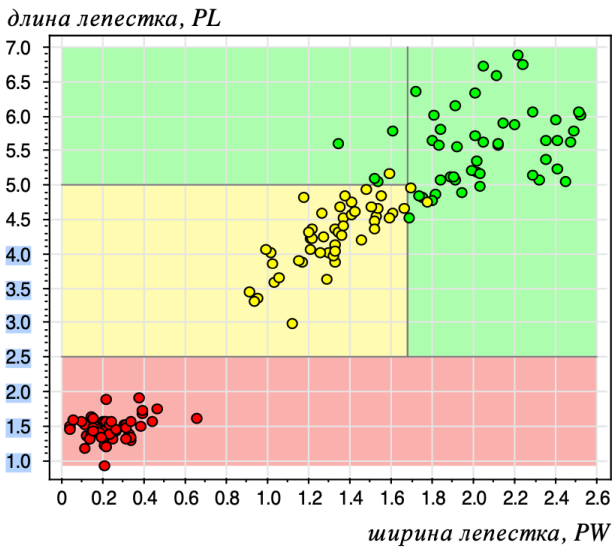
- У нас есть обучающая выборка: объекты, заданные n признаками, и каждому объекту соответствует какой-то правильный ответ. Наша задача — построить алгоритм классификации, который был бы способен классифицировать новые объекты. Но при алгоритм классификации будем строить в виде дерева -> в виде последовательности принимаемых решений

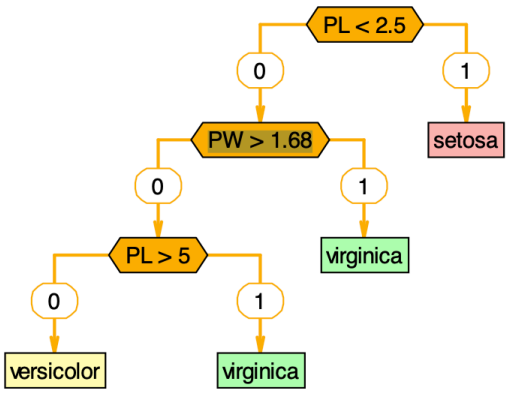
Бинарное решающее дерево — алгоритм классификации $a(x)$, задающийся бинарным деревом:
1) $\forall v \in V_{\text{внутр}} \;\rightarrow\;$ предикат $\beta_v : X \to \{0,1\},\ \beta_v \in \mathscr{B}$,
2) $\forall v \in V_{\text{лист}} \;\rightarrow\;$ имя класса $c_v \in Y$,
где $\mathscr{B}$ — множество бинарных признаков или предикатов (например, вида $\beta(x) = [x^j \geqslant \theta_j],\ x^j \in \mathbb{R}$)

1: $v := v_0$;
2: **пока** $v \in V_{\text{внутр}}$
3: **если** $\beta_v(x) = 1$ **то**
4: переход вправо: $v := R_v$;
5: **иначе**
6: переход влево: $v := L_v$;
7: **вернуть** c_v.

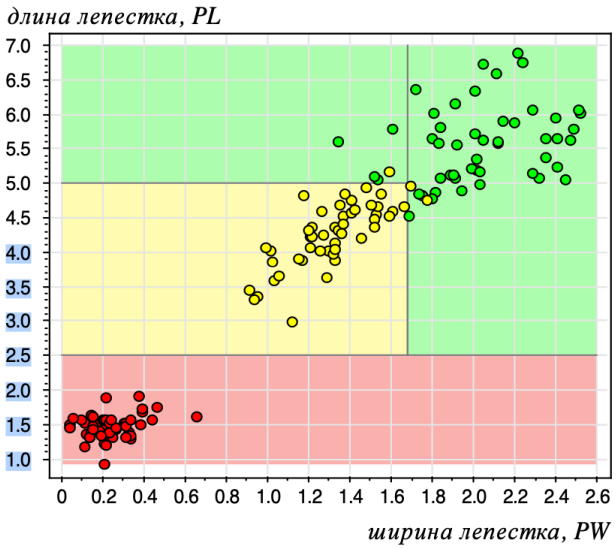


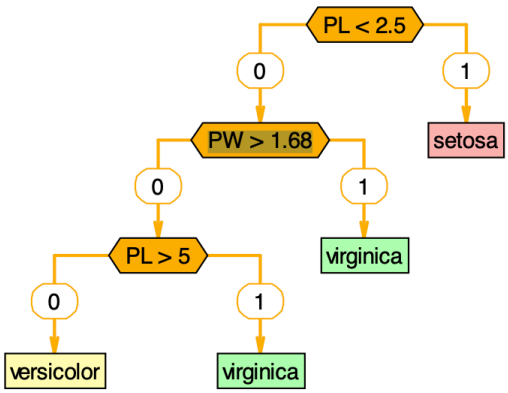
Пример





setosa	$r_1(x) = [PL \leqslant 2.5]$
virginica	$r_2(x) = [PL > 2.5] \wedge [PW > 1.68]$
virginica	$r_3(x) = [PL > 5] \wedge [PW \leqslant 1.68]$
versicolor	$r_4(x) = [PL > 2.5] \wedge [PL \leqslant 5] \wedge [PW < 1.68]$





setosa	$r_1(x) = [PL \leqslant 2.5]$
virginica	$r_2(x) = [PL > 2.5] \wedge [PW > 1.68]$
virginica	$r_3(x) = [PL > 5] \wedge [PW \leqslant 1.68]$
versicolor	$r_4(x) = [PL > 2.5] \wedge [PL \leqslant 5] \wedge [PW < 1.68]$

ID3

> Для построения решающих деревьев существует процедура, которая называется LearnID3. ID3 — сокращенно от Induction of Decision Tree, рекурсивная процедура.

- Допустим, рекурсивной процедуре на вход пришла некоторая подвыборка U. Если мы обнаружим, что все объекты из U лежат в одном классе, то мы образуем новую листовую вершину, запишем в нее этот самый класс и эту листовую вершину вернем.

1: **ПРОЦЕДУРА** LearnID3 $(U \subseteq X^\ell)$;
2: **если** все объекты из U лежат в одном классе $c \in Y$ **то**
3: **вернуть** новый лист v, $c_v := c$;
4: найти предикат с максимальной информативностью:
$\beta := \arg\max_{\beta \in \mathscr{B}} I(\beta, U)$;
5: разбить выборку на две части $U = U_0 \sqcup U_1$ по предикату β:
$U_0 := \{x \in U : \beta(x) = 0\}$;
$U_1 := \{x \in U : \beta(x) = 1\}$;
6: **если** $U_0 = \varnothing$ или $U_1 = \varnothing$ **то**
7: **вернуть** новый лист v, $c_v :=$ Мажоритарный класс(U);
8: создать новую внутреннюю вершину v: $\beta_v := \beta$;
построить левое поддерево: $L_v :=$ LearnID3 (U_0);
построить правое поддерево: $R_v :=$ LearnID3 (U_1);
9: **вернуть** v;

- Таким образом, процедура работает рекурсивно. В каждой внутренней вершине, когда происходит разделение выборки на U0 и U1, мы вызываем процедуру еще раз. Она вызывает сама себя. Это рекурсия, и это самый удобный и лаконичный способ записать процедуру обучения решающего дерева. На этой процедуре основано огромное количество алгоритмов.

Критерии ветвления ID3

- критерий Джини

  > насколько часто объекты одних классов объединяются (сколько существует пар объектов, лежащих в одном и том же классе, которые вместе идут либо в левую дочернюю вершину, либо в правую дочернюю вершину. У этих объектов должны совпадать метки классов и совпадать значение предиката)

- критерий Донского

  > насколько данный предикат обладает способностью разделять объекты разных классов

- энтропийный критерий

Достоинства решающих деревьев ID3

- интерпретируемость и простота классификации
- допустимы разнотипные данные и данные с пропусками
- трудоемкость линейна по длине выборки
- не бывает отказов от классификации

Недостатки решающих деревьев ID3

- жадный ID3 переусложняет структуру дерева, а как следствие, сильно переобучается
- фрагментация выборки: чем дальше от корня, тем меньше статистическая надёжность выбора
- высокая чувствительность к шуму, составу выборки, критерию информативности (вводятся штрафы)

Programming Assignment

> Попробуй решить (https://github.com/anafisa/Introduction-to-ML-hse-yandex/tree/master/Week1)

&2 Метрические методы классификации: kNN

> Метрические методы проводят классификацию на основе сходства, благодаря чему могут работать на данных со сложной структурой — главное, чтобы между объектами можно было измерить расстояние. **Метод k ближайших соседей** - важный класс методов машинного обучения, использующий функцию расстояния (метрику в пространстве объектов).

Принцип работы

- Исходная идея данного метода основано на гипотизе компактности — предположении о том, что близкие объекты, как правило, лежат в одном классе.

- Что такое близость между объектами, как ее можно формализовать? Как правило, задается некая функция расстояния — это функция от пары объектов, которая паре ставит соответствие — неотрицательное число. Часто еще накладывают требование, чтобы это была метрика в пространстве объектов, то есть чтобы она была и симметричной, и выполнялось неравенство треугольника.

- Чтобы классифицировать объект x, давайте возьмем все объекты обучающей выборки и найдем среди них ближайший к x, и отнесем x к тому же классу, к которому принадлежит этот объект. Использовать для классификации только одного ближайшего соседа — это не очень удачная идея, лучше взять окрестность, учесть несколько ближайших соседей, как-то по ним усреднить.

Для произвольного $x \in X$ отранжируем объекты $x_1, \ldots, x_\ell$:

$$\rho(x, x^{(1)}) \leqslant \rho(x, x^{(2)}) \leqslant \cdots \leqslant \rho(x, x^{(\ell)}),$$

$x^{(i)}$ — i-й сосед объекта x среди $x_1, \ldots, x_\ell$;
$y^{(i)}$ — ответ на i-м соседе объекта x.

Метрический алгоритм классификации:

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

$w(i, x)$ — 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.

Достоинства kNN

- простота реализации
- параметр кол-ва соседей можно оптимизировать по скользящему контролю (кросс-валидация)

Недостатки kNN

- неоднозначность классификации (оценки близости для двух классов могут совпадать)
- учитываются не значения расстаяний, а их ранги

Метод окна Парзена

> Метод окна Парзена - некоторое обобщение kNN. Данный метод способен справиться со всеми недостатками стандартного алгоритма k ближайших соседей.

- Мы можем выбирать функцию весов соседей любым образом. Сделаем так, чтобы вес соседа убывал по мере возрастания расстояния до него. Введем два новых понятия: это ядро, это функция положительная, не возрастающая на отрезке [0, 1], и ширина окна. И если функцию веса задать как конструкцию — ядро от расстояния поделить на ширину окна, то получим взвешенную функцию, которая придает меньшие веса тем соседям, которые находятся дальше. Этот метод называется методом окна Парзена (окна бывают фиксированной и переменной длинны)

$w(i,x) = K\left(\frac{\rho(x,x^{(i)})}{h}\right)$, где h — ширина окна,
$K(r)$ — ядро, не возрастает и положительно на $[0,1]$.

Метод парзеновского окна *фиксированной ширины:*

$$a(x; X^\ell, h, K) = \arg\max_{y \in Y} \sum_{i=1}^{\ell} [y_i = y]\, K\left(\frac{\rho(x, x_i)}{h}\right)$$

Метод парзеновского окна *переменной ширины:*

$$a(x; X^\ell, k, K) = \arg\max_{y \in Y} \sum_{i=1}^{\ell} [y_i = y]\, K\left(\frac{\rho(x, x_i)}{\rho(x, x^{(k+1)})}\right)$$

Оптимизация параметров — по критерию LOO:

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

- 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 $K(r)$.

Programming Assignment

> Попробуй решить (https://github.com/anafisa/Introduction-to-ML-hse-yandex/tree/master/Week2/K%20Neighbors)

&3 Линейные методы классификации: Метод стахостического градиента SG, SAG

> Линейные алгоритмы — распространенный класс моделей, которые отличается своей простотой и скоростью работы.Линейные модели — один из наиболее изученных классов алгоритмов в машинном обучении. Они легко масштабируются и широко применяются для работы с большими данными. Их можно обучать за разумное время на очень больших объемах данных, и при этом они могут работать с любыми типами признаков — вещественными, категориальными, разреженными.

Метод стохастического градиента

> Один из самых простых методов обучения линейных моделей — метод стохастического градиента. На каждой итерации алгоритма из обучающей выборки случайным образом выбирается только один объект. Таким образом вектор w настраивается на каждый вновь выбираемый объект

- В задачах обучения с учителем дана выборка объектов и ответов. Ответы в случае регрессии — вещественные числа, будем считать, что все признаки вещественные. Линейная модель регрессии — это взвешенная сумма всех признаков. В таких случаях мы пользуемся методом наименьших квадратов, это самый распространенный способ решения задачи. Выписав в сумму функции потерь по всем объектам обучающей выборки, мы получаем функционал, который можно минимизировать по вектору весов линейной модели w.

- Геометрический смысл состоит в том, что w — это направляющий вектор разделяющей гиперплоскости в n-мерном пространстве. И если у нас знак скалярного произведения положителен, то есть точка лежит по одну сторону от разделяющей гиперплоскости, то мы классифицируем объект за класс +1. Если знак скалярного произведения отрицателен, то это считаем объектом класса –1. Если выписать естественную функцию потерь, эта функция потерь оказывается бинарной: либо классификатор ошибается, либо не ошибается.

- Данная функция потерь неудобна тем, что минимизировать получающийся функционал по вектору весов w неудобно, так как этот функционал оказывается кусочно-постоянным. Нельзя продифференцировать и приравнять нулю производную. Чтобы упростить решение задачи, пользуются очень распространенным приемом подмены функционала. Вместо бинарной функции потерь будем использовать некоторую ее непрерывную аппроксимацию. Чтобы ввести такую аппроксимацию, понадобится понятие отступа. Это очень важное понятие в теории классификации, очень многие методы основаны на использовании этой оценки.

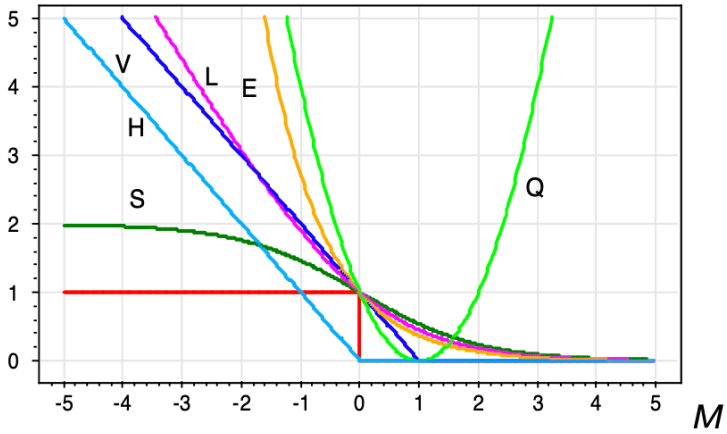
> Margin оценивает, насколько далеко объект отстоит от разделяющей поверхности (от разделяющей гиперплоскости), причем знак этой величины показывает, насколько правильна классификация. Если знак положительный, значит ошибки нет, если знак отрицательный — ошибка есть, а абсолютная величина отступа показывает, насколько далеко объект находится от разделяющей поверхности. Поэтому мы можем взять величину отступа и штрафовать за то, что объект попал в чужой класс, то есть отступ отрицательный. Чем больше по абсолютной величине отрицательный отступ, тем больше должен быть штраф.

- Отсюда вывод: можно ввести непрерывную аппроксимацию бинарной функции потерь, если воспользоваться некоторой невозрастающей непрерывной функцией от отступа. Чем отступ больше, тем меньше значение функции потерь l. И таким образом, мы снова можем свести задачу обучения к задаче оптимизации. Мы можем выписать функционал числа ошибок на обучающей выборке, а потом, воспользовавшись оценкой сверху для каждого слагаемого, получить новый функционал, который теперь уже дифференцируем по параметру, и в нем появилась некая степень свободы.

Обучающая выборка: $X^\ell = (x_i, y_i)_{i=1}^\ell$, $x_i \in \mathbb{R}^n$, $y_i \in \{-1, +1\}$

1. Модель классификации — *линейная*:
$$a(x, w) = \text{sign}\langle x, w\rangle$$
2. Непрерывная аппроксимация бинарной функции потерь:
$$\mathscr{L}(a, y) = \big[\langle x_i, w\rangle y_i < 0\big] \leqslant \mathscr{L}\big(\langle x_i, w\rangle y_i\big),$$
где $M_i(w) = \langle x_i, w\rangle y_i$ — *отступ* (margin) объекта x_i
3. Метод обучения — *минимизация эмпирического риска*:
$$Q(w) = \sum_{i=1}^{\ell} \big[a(x_i, w) y_i < 0\big] \leqslant \sum_{i=1}^{\ell} \mathscr{L}\big(\langle x_i, w\rangle y_i\big) \to \min_w$$
4. Проверка по тестовой выборке $X^k = (\tilde{x}_i, \tilde{y}_i)_{i=1}^k$:
$$\bar{Q}(w) = \frac{1}{k} \sum_{i=1}^{k} \big[\langle \tilde{x}_i, w\rangle \tilde{y}_i < 0\big]$$

Часто используемые непрерывные функции потерь $\mathscr{L}(M)$:



$V(M) = (1 - M)_+$ — кусочно-линейная (SVM);
$H(M) = (-M)_+$ — кусочно-линейная (Hebb's rule);
$L(M) = \log_2(1 + e^{-M})$ — логарифмическая (LR);
$Q(M) = (1 - M)^2$ — квадратичная (FLD);
$S(M) = 2(1 + e^{M})^{-1}$ — сигмоидная (ANN);
$E(M) = e^{-M}$ — экспоненциальная (AdaBoost);
$[M < 0]$ — пороговая функция потерь.

Достоинства метода стахостического градиента

- легко реализуется
- применим к любым моделям и функциям потерь
- допускает обучение в режиме онлайн (потоковое обучение)
- на сверхбольших выборках позволяет получить неплохие решения, даже не обработав все (xi, yi)
- всё чаще и чаще применяется в BigData

Недостатки метода стахостического градиента

- возможно застревание в локальных экстремумах

Градиентные методы и алгоритм SAG (стохастический средний градиент)

- Мы можем применять различные методы численной оптимизации для того, чтобы минимизировать наш аппроксимированный функционал, поскольку он теперь является непрерывной функцией (или даже гладкой), в зависимости от того, какую функцию потерь мы использовали.

> Самый простой метод численной оптимизации — это метод **градиентного спуска**. Он заключается в том, что сначала фиксируется некоторое начальное приближение для искомого вектора весов. Например, случайное. И затем делается последовательность градиентных шагов. То есть каждая итерация — это небольшое смещение вектора весов по антиградиенту. Почему антиградиент? Градиент — это вектор, который показывает направление наискорейшего возрастания функции. Соответственно, минус этот вектор, или антиградиент, он показывает направление наискорейшего убывания.

- **Здесь возникает несколько проблем**:

  1. проблема, будет ли этот метод сходиться;
  2. проблема выбора градиентного шага;
  3. проблема выбора начального приближения.

- Как ускорить процесс спуска в том случае, когда функционал, который мы оптимизируем, представляет собой сумму большого числа слагаемых? (а это как раз распространённый случай). Идея ускорения сходимости здесь заключается в том, чтобы не вычислять сумму сразу по всем объектам, а брать каждый объект (обычно их берут в случайном порядке по одному) и после каждого объекта обновлять вектор весов. Оказывается, это приводит к существенному ускорению сходимости, это называется процедурой Роббинса–Монро, которая была уже более полувека назад предложена для решения задач оптимизации вот такого сорта функционалов, и называется **методом стохастической аппроксимации**.

- Процедура заключается в следующем:
  - сначала инициализируется вектор весов. При текущем положении вектора весов вычисляется оценка функционала качества на обучающей выборке, с нашей аппроксимированной функцией потерь, и затем начинается основной процесс.
  - На каждом шаге этого итерационного процесса мы выбираем объект и обучающие выборки случайным образом. Вычисляем значение функции потерь, обозначенного εi, и делаем градиентный шаг. Далее необходимо с учетом сделанной поправки к вектору весов переоценить значение функционала. Это нужно для того, чтобы понять, в какой момент стоит останавливаться, когда значение функционала к чему-то сойдется или перестанет существенно меняться.

Programming Assignment

> Попробуй решить (https://github.com/anafisa/Introduction-to-ML-hse-yandex/tree/master/Week2/Linear%20Classification%20Methods)

&4 Подклассы линейных методов: Метод опорных векторов, Логистическая регрессия

> Линейные методы имеют несколько очень важных подвидов. Метод опорных векторов максимизирует отступы объектов, что тесно связано с минимизацией вероятности переобучения. При этом он позволяет очень легко перейти к построению нелинейной разделяющей поверхности благодаря ядровому переходу. Логистическая регрессия позволяет оценивать вероятности принадлежености классам, что оказывается полезным во многих прикладных задачах.

Метод опорных векторов

> **Метод опорных векторов** максимизирует отступы объектов, что тесно связано с минимизацией вероятности переобучения. При этом он позволяет очень легко перейти к построению нелинейной разделяющей поверхности благодаря ядровому переходу.

- Ставится задача классификация объектов, заданных признаками в Rn, n вещественными числами, будем рассматривать простой двухклассовый случай, когда метки классов –1 и +1. Цель — построить линейный классификатор, то есть такую функцию, которая возвращает –1 или + 1, и при этом зависит от скалярного произведения вектора признаков x на вектор весов признаков той же размерности w. Есть еще свободный член w0. Вот это вот скалярное произведение минус свободный член принято называть дискриминантной функцией. Знак дискриминантной функции показывает, к какому классу будет отнесен объект x. Если дискриминантная функция больше нуля, то объект относится к классу +1, если меньше нуля, то к классу –1.

- Возникает задача: как по обучающей выборке определить значение параметров w и w0? Мы будем сводить задачу к оптимизационной. Самый простой критерий, который может быть использован, — это минимизация эмпирического риска. То есть мы хотим найти такие параметры w (вектор размерности n) и w0 (скаляр), чтобы число ошибок классификации на обучающей выборке было минимально. Запишем это условие в несколько ином виде. Мы введем величину отступа (по-английски margin) объекта, которая является произведением дискриминантной функции на метку правильного ответа на объекте xi. То есть получается, что если дискриминантная функция и правильный ответ одного знака, то отступ положительный, ошибки на объекте нет. Если они разного знака, то происходит ошибка на объекте, отступ отрицательный. Отступ — величина непрерывная, и интуитивно кажется, что чем меньше значение отступа, тем хуже. Чем больше значение отступа, тем дальше объект находится от разделяющей гиперплоскости, он лежит глубоко внутри своего класса, и на нем классификация надежна. И отсюда возникает идея, мерить ошибку не как бинарную величину — отступ отрицательный или положительный, а использовать саму величину отступа.

Дано:
Обучающая выборка $X^\ell = (x_i, y_i)_{i=1}^\ell$,
x_i — объекты, векторы из множества $X = \mathbb{R}^n$,
y_i — метки классов, элементы множества $Y = \{-1, +1\}$.

Найти:
Параметры $w \in \mathbb{R}^n$, $w_0 \in \mathbb{R}$ линейной модели классификации

$$a(x; w, w_0) = \text{sign}\big(\langle x, w\rangle - w_0\big).$$

Критерий — минимизация эмпирического риска:

$$\sum_{i=1}^{\ell}\big[a(x_i; w, w_0) \neq y_i\big] = \sum_{i=1}^{\ell}\big[M_i(w, w_0) < 0\big] \;\to\; \min_{w, w_0}.$$

где $M_i(w, w_0) = \big(\langle x_i, w\rangle - w_0\big)y_i$ — *отступ* (margin) объекта x_i,
$b(x) = \langle x, w\rangle - w_0$ — *дискриминантная функция*.

- Возникает идея, мерить ошибку не как бинарную величину — отступ отрицательный или положительный, а использовать саму величину отступа. Для этого вводится аппроксимация. Аппроксимации могут быть разными, рассмотрим кусочно-линейную аппроксимацию, которая будет штрафовать объекты за приближение к границе между классами. И если объект переходит через границу класса, оказывается в чужом классе и продолжает двигаться дальше, то штраф будет линейно возрастать. Функционал, который мы вводим, мажорирует сверху (функционал эмпирического риска, который просто является числом ошибок на обучающей выборке). Поэтому если мы будем минимизировать наш новый функционал, то мы тем самым будем минимизировать и исходный функционал числа ошибок.
- Кроме аппроксимации пороговой функции потерь кусочно-линейной непрерывной функции, можно ввести еще одну оценку сверху на функционал в виде штрафного слагаемого регуляризатора, который наказывает решение за слишком большую норму вектора коэффициентов. Такое штрафное слагаемое позволяет избежать проблемы переобучения, которая может возникнуть из-за мультиколлинеарности, когда среди признаков есть линейно-зависимые.

Логистическая регрессия

> **Логистическая регрессия** — это метод построения линейных классификаторов. Логистическая регрессия позволяет оценивать вероятности принадлежености классам, что оказывается полезным во многих прикладных задачах.

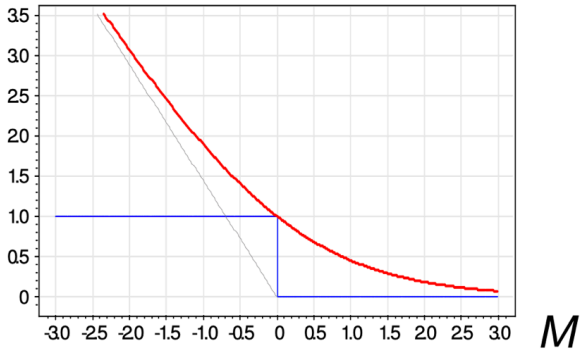
- Несмотря на название — регрессия, это метод для решения задач классификации, а почему «регрессия», мы сейчас поймем. Давайте рассмотрим постановку задачи. У нас имеется обучающая выборка. Это пара объект-ответ. Объекты описываются n-вещественными признаками, то есть это векторы из Rn, а ответы — это числа (–1) и (+1), то есть рассматривается двухклассовая задача. Итак, линейная модель классификации — это скалярное произведение вектора объектов на вектор весов. Вектор весов — это направляющий вектор разделяющей гиперплоскости. Если скалярное произведение положительное, то объект относится к классу (+1), если отрицательное, то к классу (–1). Рассмотрим непрерывную аппроксимацию бинарной функции потерь. Если мы в качестве потери рассматриваем бинарную величину (есть ошибка или нет ошибки), то мы просто имеем число ошибок классификатора на обучающей выборке. Если же мы используем непрерывную функцию потерь, которая мажорирует сверху бинарную функцию потерь, то мы получаем функционал, который гораздо удобнее минимизировать, потому что он является непрерывным или даже гладким. Но поскольку он мажорирует сверху, то, минимизируя этот функционал, мы также будем минимизировать и исходное число ошибок.

- При этом появляется важное понятие — отступ объекта. Это скалярное произведение, умноженное на правильный ответ, правильный ответ ± 1. Поэтому если у нас на объекте есть ошибка — отступ отрицательный, если нет ошибки — отступ положительный. При этом имеется функция потерь l, если она монотонно убывает, то она штрафует нас за ошибки и даже штрафует за приближение к границе между классами.

- Будем рассматривать конкретную функцию потерь, которая называется логарифмической. Оказывается, что она приобретает очень интересный вероятностный смысл, если мы предположим, что у нас имеется вероятностная модель порождения данных. А именно: будем предполагать, что наша выборка является выборкой независимых наблюдений из одного и того же параметрического семейства распределений. Это совместная плотность распределений иксов и игреков с каким-то параметром w. Поскольку выборка независимая, то, исходя из принципа максимума правдоподобия, мы можем определить параметр w, а независимость дает нам возможность представить правдоподобие в очень удобном виде как сумму логарифмов совместных плотностей объектов и ответов. Вот совместную плотность p(x, y) можно представить по формуле условной вероятности в виде произведения условной вероятности ответа y при условии x и безусловной плотности p(x). Так вот, наше предположение будет заключаться в том, что плотность p(x) не зависит от параметра модели, а параметр модели используется только для описания условной или, говорят, апостериорной вероятности класса для данного объекта x. Так вот оказывается, что если мы вполне определенный вид этой апостериорной вероятности предположим, то принцип максимума правдоподобия даст ровно тот же функционал, который мы ввели выше из чисто эвристических соображений. А именно: оказывается, что апостериорная вероятность класса yi-тое для объекта xi-тое дается вот такой вот функцией, которая называется сигмоидной функцией — функция от отступа. И получается, что функционал, который мы ввели выше, — это минимум аппроксимированного эмпирического риска, и функционал правдоподобия — максимум логарифма правдоподобия.

- Логарифмическая функция потерь, как функция отступа M:

$$\mathscr{L}(M) = \log\left(1 + e^{-M}\right)$$



- Оказывается, что если мы будем оптимизировать такой функционал, то мы заодно сможем оценить и апостериорную вероятность класса для каждого объекта, который мы будем классифицировать. И в этом есть основное отличие логистической регрессии от других методов линейной классификации. Она позволяет оценивать вероятности классов. Как же производится оптимизация этого функционала? Есть 2 подхода. С одним мы уже познакомились — это методы первого порядка. В частности, можно использовать метод стохастического градиента. И если расписать формулу стохастического градиента, то окажется, что в нее тоже войдет вот эта самая вероятность правильной классификации — апостериорная вероятность класса yi-тое (правильного класса) на объекте xi-тое.

> Попробуй решить (https://github.com/anafisa/Introduction-to-ML-hse-yandex/tree/master/Week3)

&5 Линейные модели регрессии: линейная регрессия и метод главных компонент

> Как строить многомерную линейную регрессию, как проходит процесс обучения?

Линейная регрессия

> Имеется n числовых признаков, имеется обучающая выборка, мы хотим построить по обучающей выборке модель многомерной линейной регрессии, то есть это просто взвешенная сумма значений признаков, весовые коэффициенты αj.

- Перейдем к матричным обозначениям. **Матрица объекты-признаки** — это матрица, в которой строки соответствуют объектам, столбцы — признакам.

- Еще понадобится вектор ответов или **целевой вектор** — число строк равно числу объектов обучающей выборки. И **вектор коэффициентов** — число строк равно числу признаков, столбец один.

- В этих трех векторно-матричных обозначениях очень удобно записать постановку задачи наименьших квадратов. Если мы запишем сумму по всем объектам обучающей выборки квадрата разности правильного ответа yi, и ответа нашей модели f (xi, α), то в матричной записи это не что иное, как квадрат нормы разности двух векторов. Вектор y — это вектор правильных ответов, а произведение Fα, матрица F * вектор α, — это есть вектор, который аппроксимирует вектор правильных ответов согласно нашей линейной модели. Задача — это найти вектор α при известной матрице F и известному вектор-столбцу y. Запишем необходимые условия минимума.

$f_1(x), \ldots, f_n(x)$ — числовые признаки;
Модель многомерной линейной регрессии:

$$f(x, \alpha) = \sum_{j=1}^{n} \alpha_j f_j(x), \qquad \alpha \in \mathbb{R}^n.$$

Матричные обозначения:

$$\underset{\ell \times n}{F} = \begin{pmatrix} f_1(x_1) & \ldots & f_n(x_1) \\ \ldots & \ldots & \ldots \\ f_1(x_\ell) & \ldots & f_n(x_\ell) \end{pmatrix}, \quad \underset{\ell \times 1}{y} = \begin{pmatrix} y_1 \\ \ldots \\ y_\ell \end{pmatrix}, \quad \underset{n \times 1}{\alpha} = \begin{pmatrix} \alpha_1 \\ \ldots \\ \alpha_n \end{pmatrix}.$$

Функционал квадрата ошибки:

$$Q(\alpha, X^\ell) = \sum_{i=1}^{\ell} \big(f(x_i, \alpha) - y_i\big)^2 = \|F\alpha - y\|^2 \to \min_{\alpha}.$$

- Мы получаем систему уравнений, опять-таки в матричном виде, из которой мы можем выразить искомый вектор α, и если опять-таки в матричном виде записать этот вектор α, то получается такая очень простая формула — нам нужно псевдо-обратную матрицу F с крестом умножить на вектор правильных ответов y. Это и есть решение.

Необходимое условие минимума в матричном виде:

$$\frac{\partial Q}{\partial \alpha}(\alpha) = 2F^{\intercal}(F\alpha - y) = 0,$$

откуда следует *нормальная система* задачи МНК:

$$F^{\intercal}F\alpha = F^{\intercal}y,$$

где $F^{\intercal}F$ — $n \times n$-матрица.

Решение системы: $\alpha^* = (F^{\intercal}F)^{-1}F^{\intercal}y = F^{+}y.$

Значение функционала: $Q(\alpha^*) = \|P_F y - y\|^2,$
где $P_F = FF^{+} = F(F^{\intercal}F)^{-1}F^{\intercal}$ — *проекционная матрица.*

- Вопрос лишь в том, как искать эту матрицу F с крестом и какие могут быть проблемы в процессе ее вычисления. Проблем может быть много, и, в частности, проблема может быть в той же самой **мультиколлинеарности**, что если столбцы матрицы F линейно-зависимы, то нам вообще не удастся найти обратную матрицу к F транспонированное, она будет вырождена. Если же столбцы этой матрицы F почти линейно-зависимы, то обращаемая матрица будет плохо обусловлена и у нас возникнет масса вычислительных проблем с обращением этой матрицы.

Произвольная $\ell \times n$-матрица представима в виде *сингулярного разложения* (singular value decomposition, SVD):

$$F = VDU^{\mathsf{T}}.$$

Основные свойства сингулярного разложения:

1. $\ell \times n$-матрица $V = (v_1, \ldots, v_n)$ ортогональна, $V^{\mathsf{T}}V = I_n$, столбцы v_j — собственные векторы матрицы FF^{T};
2. $n \times n$-матрица $U = (u_1, \ldots, u_n)$ ортогональна, $U^{\mathsf{T}}U = I_n$, столбцы u_j — собственные векторы матрицы $F^{\mathsf{T}}F$;
3. $n \times n$-матрица D диагональна, $D = \operatorname{diag}\left(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n}\right)$, $\lambda_j \geqslant 0$ — собственные значения матриц $F^{\mathsf{T}}F$ и FF^{T}, $\sqrt{\lambda_j}$ — сингулярные числа матрицы F.

- Как же считать псевдо-обратную и произведение псевдо-обратной на y, чтобы найти решение нашей системы? Воспользуемся **сингулярным разложением**. Это очень полезное разложение, которое позволяет произвольную прямоугольную матрицу представить в виде произведения трех матриц. Две из них — левая V и правая — U транспонированная, являются ортогональными матрицами, а матрица, которая стоит посерединке — диагональна. Причем ортогональные матрицы не простые, а их столбцы являются собственными векторами, соответственно матриц F на F транспонированное и F транспонированное F. Из линейной алгебры известно, что эти две матрицы имеют хотя и разные собственные векторы, но собственные значения у них совпадают, именно эти собственные значения и записаны на диагонали матрицы D, точнее корни квадратные из этих собственных значений, которые называются сингулярными числами матрицы F. Это представление очень полезно, так как мы сейчас с его помощью буквально за несколько матричных операций получим другое, гораздо более удобное и понятно интерпретируемое представление для вектора решения задачи наименьших квадратов.

Метод главных компонент

> В прикладных задачах часто возникает потребность в уменьшении количества признаков — например, для ускорения работы моделей. Существует несколько подходов к отбору признаков, среди них метод главных компонент - один из самых популярных методов понижения размерности.

- Имеется множество исходных n числовых признаков, наша задача - перейти в пространство новых числовых признаков, желательно сделать так, чтобы их оказалось меньше и новые признаки содержали всю основную информацию о старых. Как это требование можно формализовать? 1) Старые признаки должны восстанавливаться по новым. Можно наложить то или иное ограничение на класс преобразований, с помощью которых старые признаки восстанавливаются по новым. Метод главных компонент — это частный случай, когда мы предполагаем, что это **преобразование линейно**.

- Старые признаки должны хорошо восстанавливаться по новым, а новые признаки j требуется найти, более того, требуется, совместно с ними, найти и матрицу преобразований, которая восстанавливает старые признаки по новым. То есть получается так, если мы применяем метод наименьших квадратов, мы минимизируем сумму квадратов разностей между значениями старыми признаков fj(xi), то есть берем каждый признак и в каждом объекте, и восстановленное значение. Если записывать эту постановку задачи в матричном виде, то ее решать гораздо удобнее. Введем матрицы «объекты-признаки» — старая матрица. Строки — это объекты выборки, столбцы — это признаки. Новая матрица G, строки соответствуют тем же самым объектам, столбцы — это новые признаки. И нам нужна матрица линейного преобразования, перехода от новых признаков к восстановленным старым.

- Получается так, что матрица F имеет размер l на n, и в общем случае она имеет полный ранг, то есть если объектов больше чем признаков, то ранг ее равен n, и мы хотим ее приблизить матрицей, которая есть произведение двух матриц размера l на m и m на n. Вот эта промежуточная размерность m, она может быть существенно меньше, чем n, то есть мы делаем низкоранговое матричное приближение исходной матрицы F. Итак, в матричном виде задача заключается в том, чтобы минимизировать квадрат нормы разности двух матриц — заданная исходная матрица F и произведение двух матриц G и U транспонированное, которые имеют, вот это произведение оно имеет ранг, может быть много меньше, чем ранг матрицы F. Есть теорема, которая решает эту задачу исчерпывающим образом, она гласит следующее, что при техническом предположении, что ранг матрицы F не меньше, чем n, минимум нашего функционала достигается в том случае, когда мы возьмем в качестве столбцов матрицы U собственные векторы матрицы F транспонированное F, соответствующие максимальным собственным значениям λ1, ..., λm. Ну вот поскольку мы берем максимальные собственные значения, они называются главные компоненты, отсюда и название метода — метод главных компонент.

- Метод главных компонент позволяет приближать матрицу ее низкоранговым разложением. Для этого достаточно взять сингулярное разложение матрицы, взять ее первые m сингулярных чисел, которые имеют максимальные значения, а все остальные отбросить. Этот прием часто применяется в анализе данных не только для решения задачи регрессии, но и для классификации, для сжатия данных, используется он также в обработке изображений.