Анализ данных (Data Mining)

Лекция 4. Контроль качества

Московский авиационный институт
«МАИ»

19 июля 2021 г.

Проблема контроля качества

Как оценить качество / ошибку алгоритма?

Разделим все имеющиеся данные на

– обучающую выборку $\left(x_i^{\text{train}}, y_i^{\text{train}}\right)$, $i = 1, \ldots, N^{\text{train}}$

– тестовую выборку (отложенный контроль) $\left(x_i^{\text{test}}, y_i^{\text{test}}\right)$, $i = 1, \ldots, N^{\text{test}}$

В задаче обучения по прецедентам есть два этапа

- Этап обучения (training):
  метод по обучающей выборке строит алгоритм $f\left(X_{\text{train}}\right)$
- Этап применения (testing):
  алгоритм для объектов тестовой x_i выборки выдаёт ответы $a_i = f(x_i) \in \mathcal{Y}$

Отложенный контроль (validation data)

Оценка ошибки зависит от конкретной выбранной отложенной выборки, часто сильно меняется при другом выборе

Если переобучить алгоритм для всех данных, то мы не знаем оценку его ошибки (в каком-то смысле, неустранимый недостаток)

в какой пропорции делить...

(обычно $\sim 20\%$ от выборки)

Большое обучение

алгоритм больше похож на обученный по всем данным

обучающая выборка более репрезентативная

Большой контроль

оценка качества более надежна

Ошибка на обучении (train error) и ошибка на контроле (test error) как правило очень различаются!

Что можно посчитать

$\overline{R}_{\text{train}}(f) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} L\left(y_i^{\text{train}}, a_i^{\text{train}}\right)$ — средний эмпирический риск на обучающей выборке

$\overline{R}_{\text{test}}(f) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} L\left(y_i^{\text{test}}, a_i^{\text{test}}\right)$ — средний эмпирический риск на тестовой выборке

$L(\cdot,\cdot)$ — функция потерь, величина ошибки алгоритма на объекте

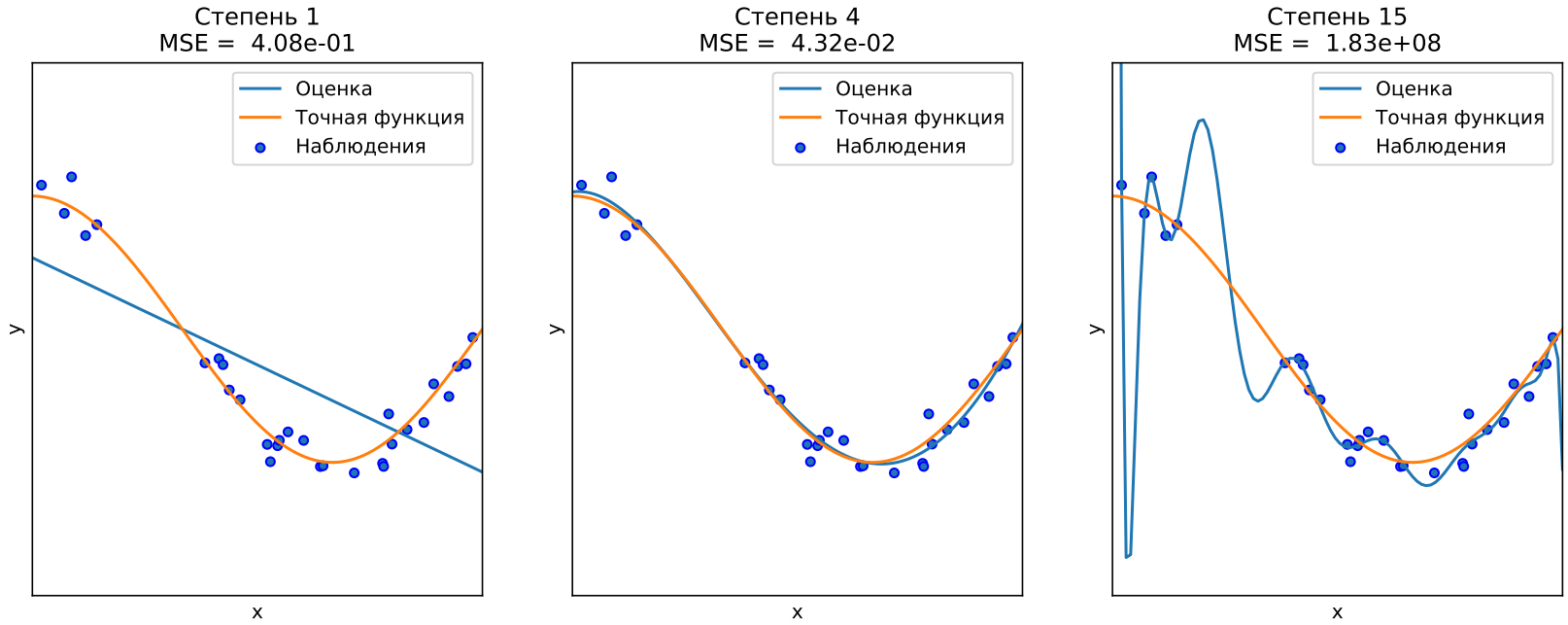
для задачи классификации $L(y,a) = \mathrm{I}(y = a)$

для регрессии, например, $L(y,a) = |y - a|$

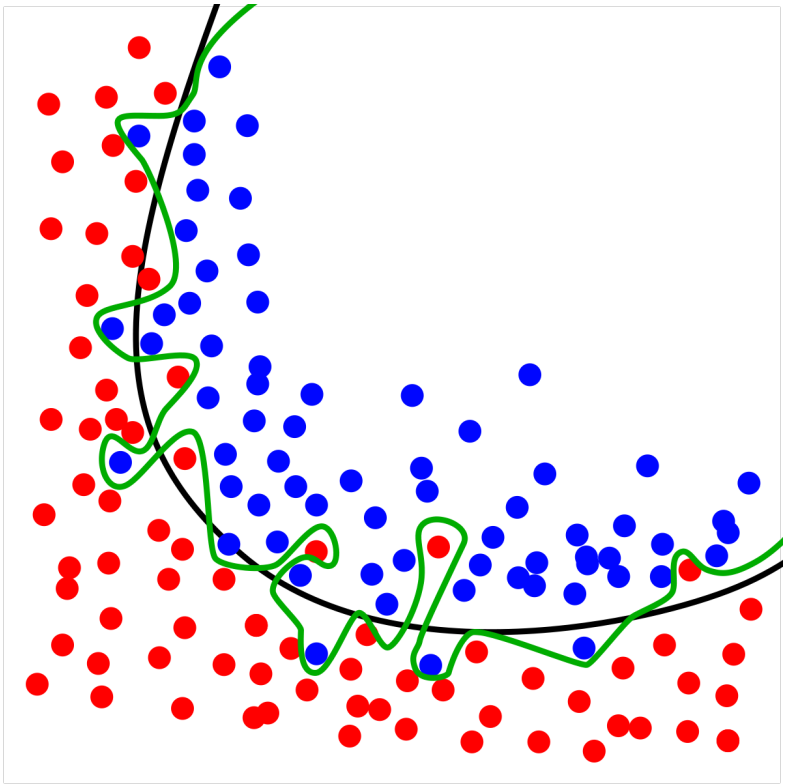
Проблемы недообучения и переобучения

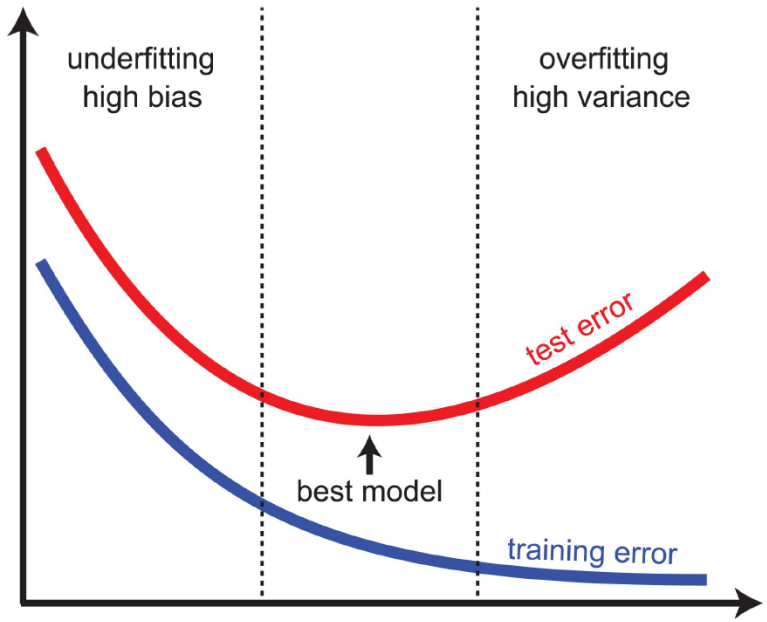
Недообучение (underfitting) — модель слишком проста

Переобучение (overfitting) — модель слишком сложна



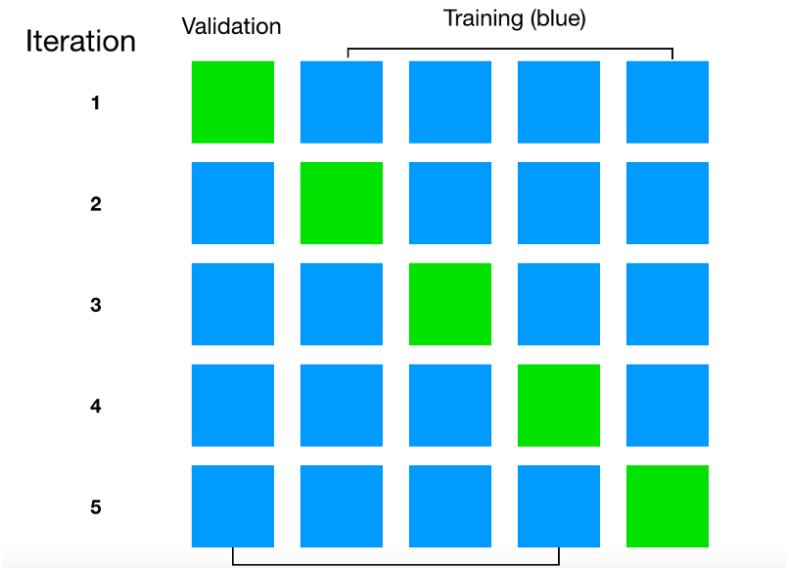
Контроль качества





Перекрестная проверка (cross validation)

Перекрестная проверка по k блокам (k-fold CV)
Разбиение на обучающую и тестовую выборку можно проделать k раз (например, случайно)

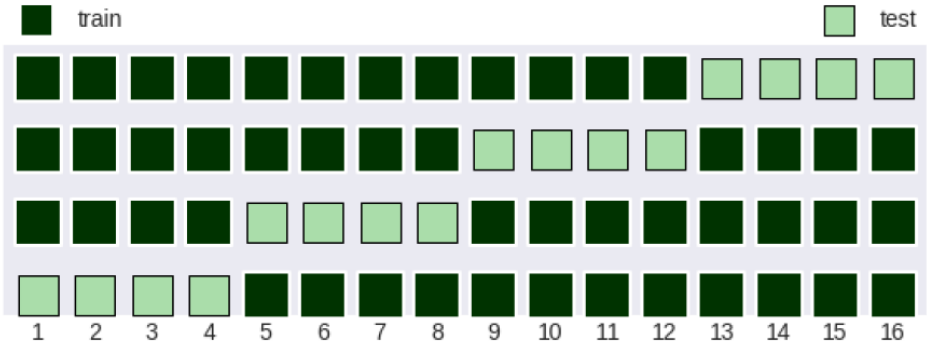


В результате получим M решающих функций: $f_1, f_2, \ldots, f_k$ и значений эмпирического риска $\overline{R}^1_{\text{test}}, \overline{R}^2_{\text{test}}, \ldots, \overline{R}^k_{\text{test}}$

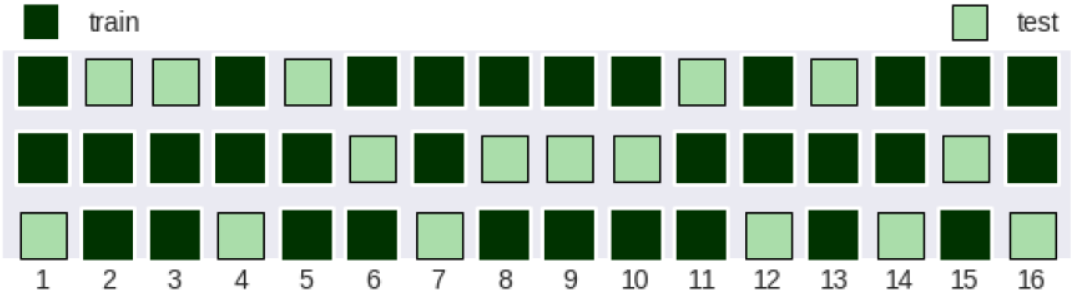
Вычисляем средний cv-риск на $\overline{R} = \sum_{j=1}^{k} \overline{R}^j_{\text{test}}$

Перекрестная проверка

sklearn.model_selection.KFold(n_splits='warn', shuffle=False, random_state=None)



sklearn.model_selection.KFold(n_splits=3, shuffle=True, random_state=None)



https://scikit-learn.org/stable/modules/cross_validation.html

Перекрёстная проверка по фолдам

«k примерно равных частей»
в последней части может быть меньше объектов, чем в остальных, если k не делит нацело объём выборки

Обычно выбирают $k = 10$

Большие значения k:

- надежнее оценка качества
- обучающая выборка больше походит на все данные
- время контроля возрастает (линейно)!
- не любое качество адекватно оценивается на маленьких подвыборках

Перекрестная проверка

Процедуру перекрестного контроля можно повторить L раз, каждый раз разбивая случайно выборку на k частей

Тогда оценка риска будет вычисляться на основе $k \cdot L$ оценок эмпирического риска

$k = m$ — метод перекрестного контроля с одним отделяемым элементом (leave-one-outcross-validation, LOOCV)

Обучающие выборки похожи друг на друга,следовательно, решающие функции f_j, по-видимому, будут похожина функцию f, построенную по всей выборке, поэтому LOOCV обычно дает лучшие оценки среднего риска.

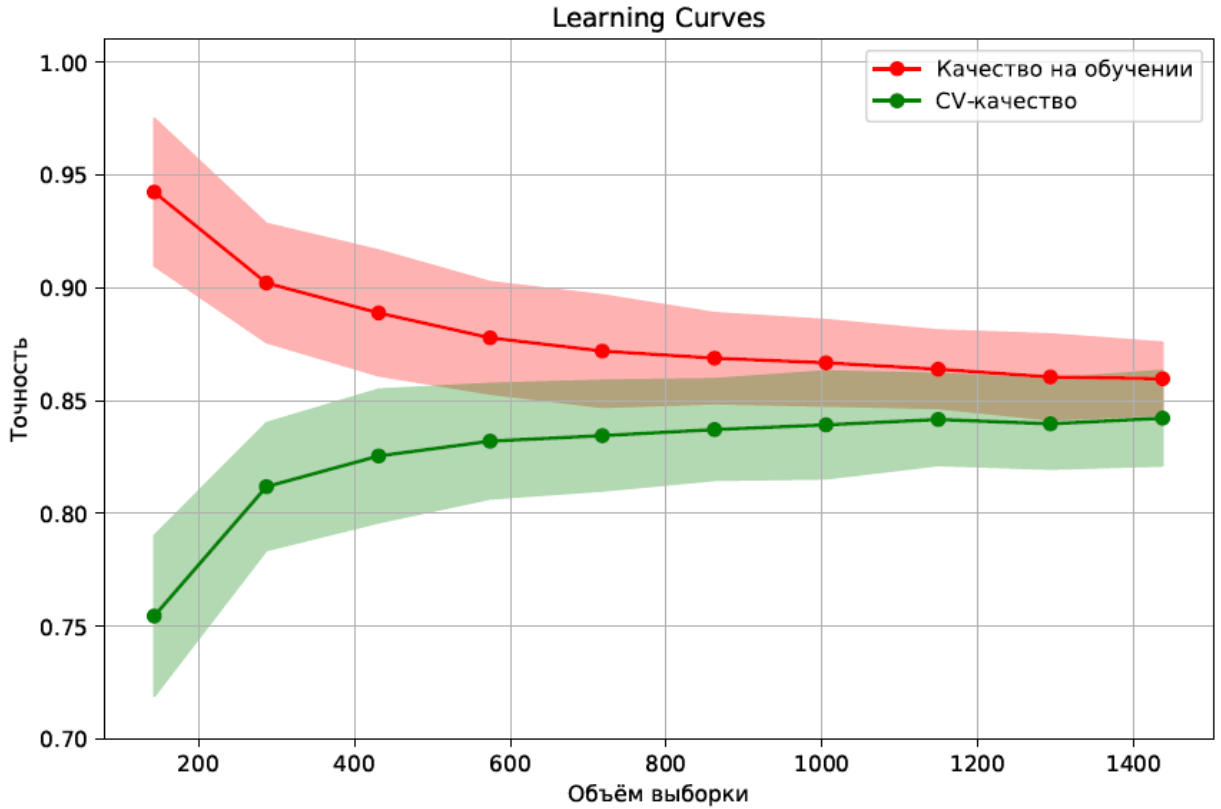
LOOCV — самый точный, но требует много времени

Где используется выбор CV-контроля

- оценка качества модели
- получение «честных» ответов на обучении:
  как бы алгоритм отвечал, если бы не обучался на этих объектах
- контроль алгоритмов
- ансамблирование (метапризнаки)
- настройка гиперпараметров
- построение кривых зависимостей качества от параметров
  - validation curves (от значений)
  - learning curves (от объёма выборки)

Кривые обучения (Learning Curves)

Делим данные на обучение и контроль (м.б. очень много раз)
Обучаемся на k% от обучающей выборки для разных k
Строим графики ошибок/качества на train/CV от k



Выбор параметров алгоритма

Алгоритм обучения по обучающей выборке строит решающую функцию $f(\cdot,\alpha)$

α — гиперпараметры алгоритма (метода) обучения

Для одной обучающей выборки, но разных значений α мы получим разные решающие функции

Какую из них выбрать?

Идеально:

$$\alpha^* = \arg\min_{\alpha} R(f(\cdot,\alpha))$$

Но приходится искать

$$\alpha^o = \arg\min_{\alpha} \overline{R}_{\text{train}}(f(\cdot,\alpha)) \text{ или } \quad \alpha^o = \arg\min_{\alpha} \overline{R}_{\text{CV}}(f(\cdot,\alpha))$$

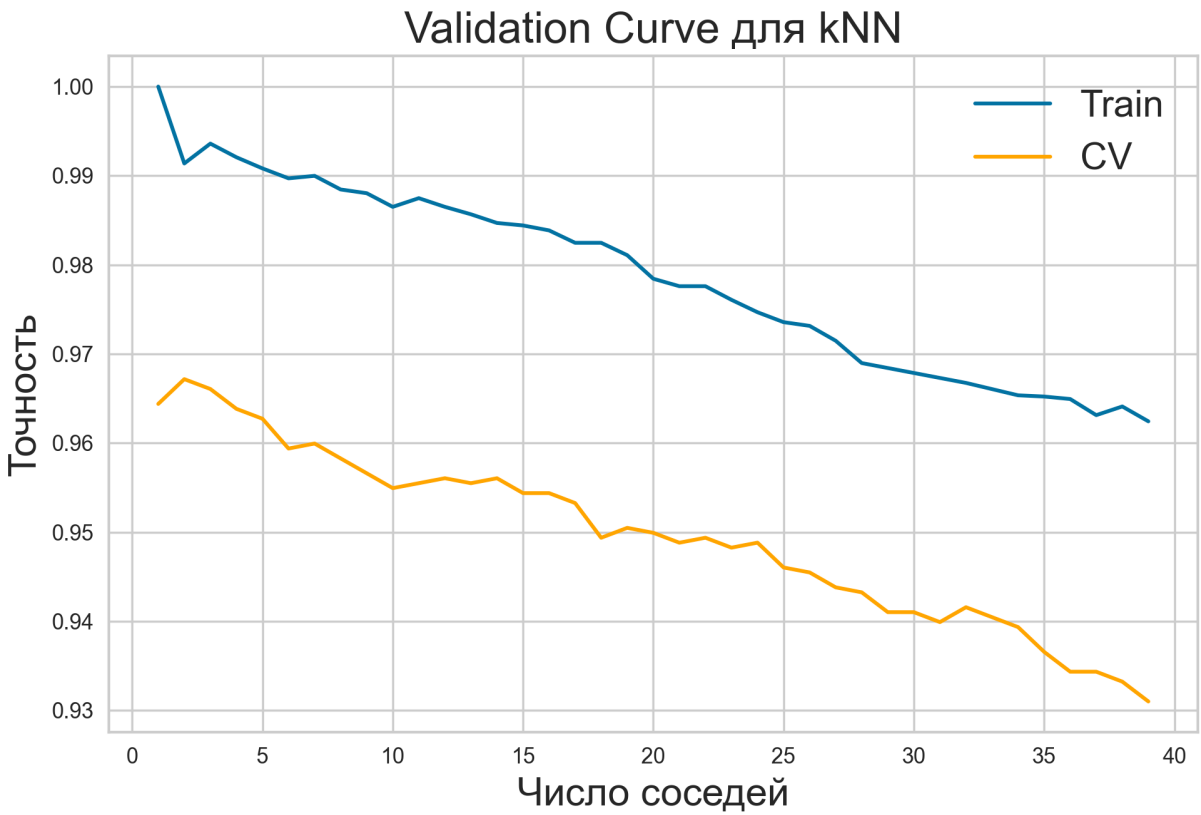
Качество от параметров (validation curves)

Делим данные на обучение и контроль (м.б. очень много раз)

При разных значениях параметров обучаемся и проверяем качество

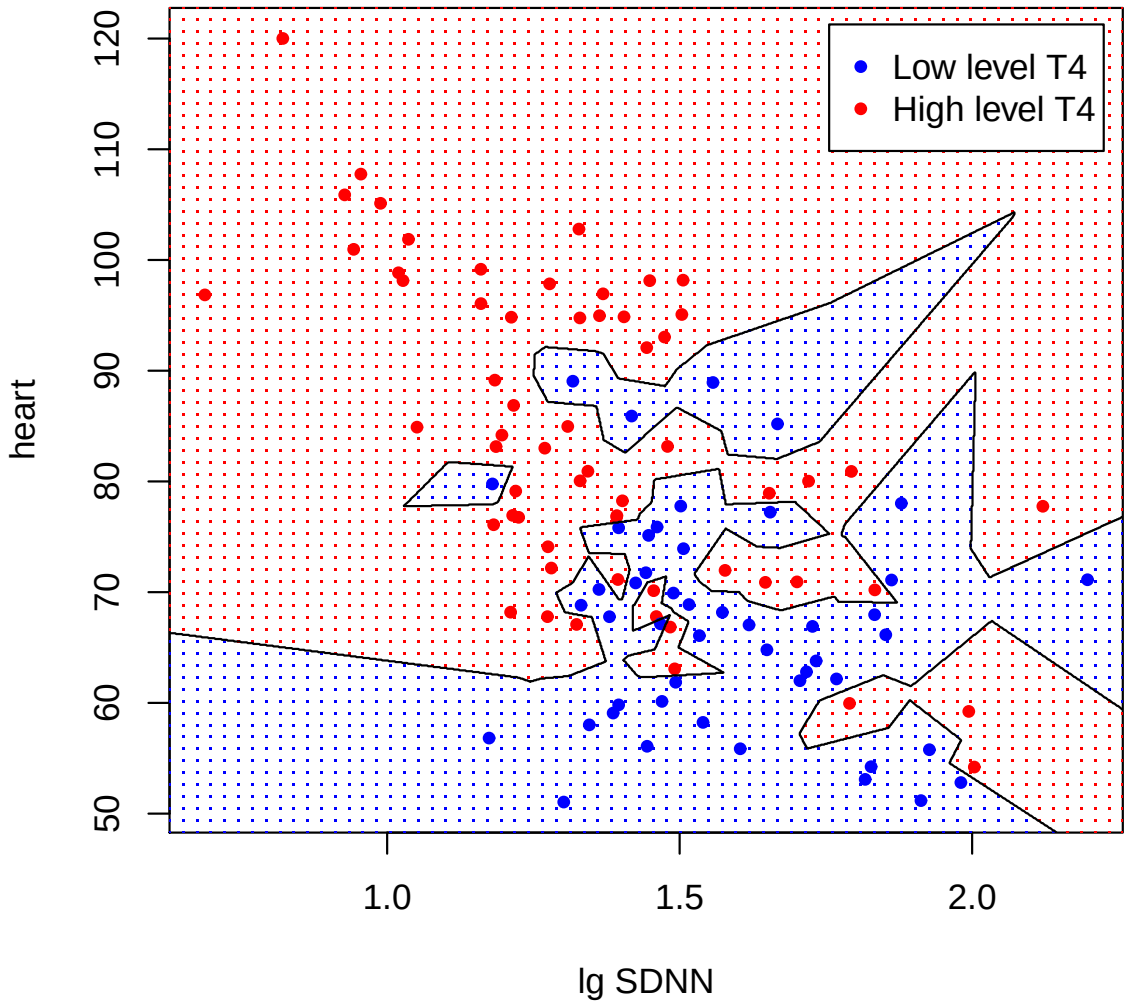
Строим графики ошибок/качества на train/CV от значения параметра

sklearn.model_selection.validation_curve



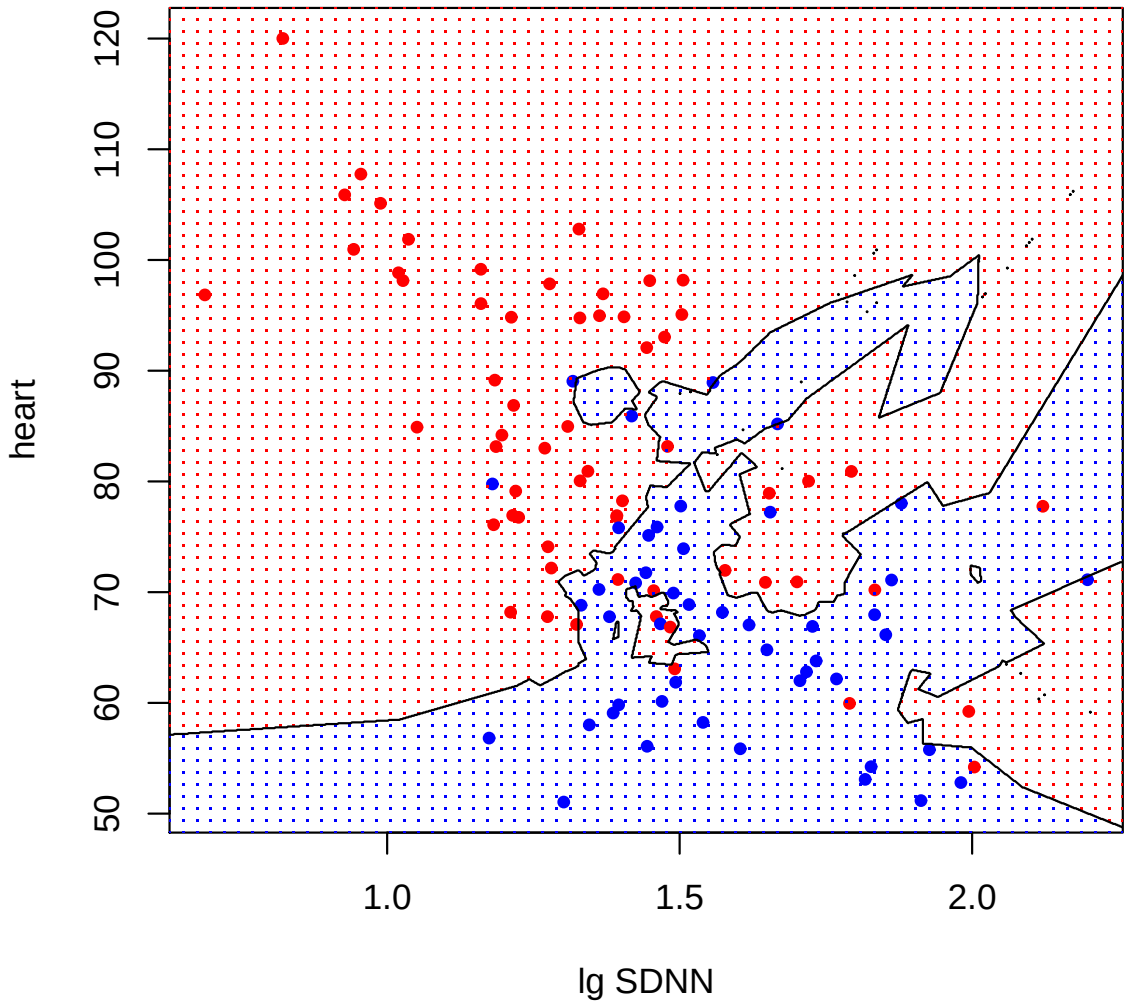
Перекрестная проверка

Задача медицинской диагностики. Метод 1 ближайшего соседа. 10-CV ошибка 0.30



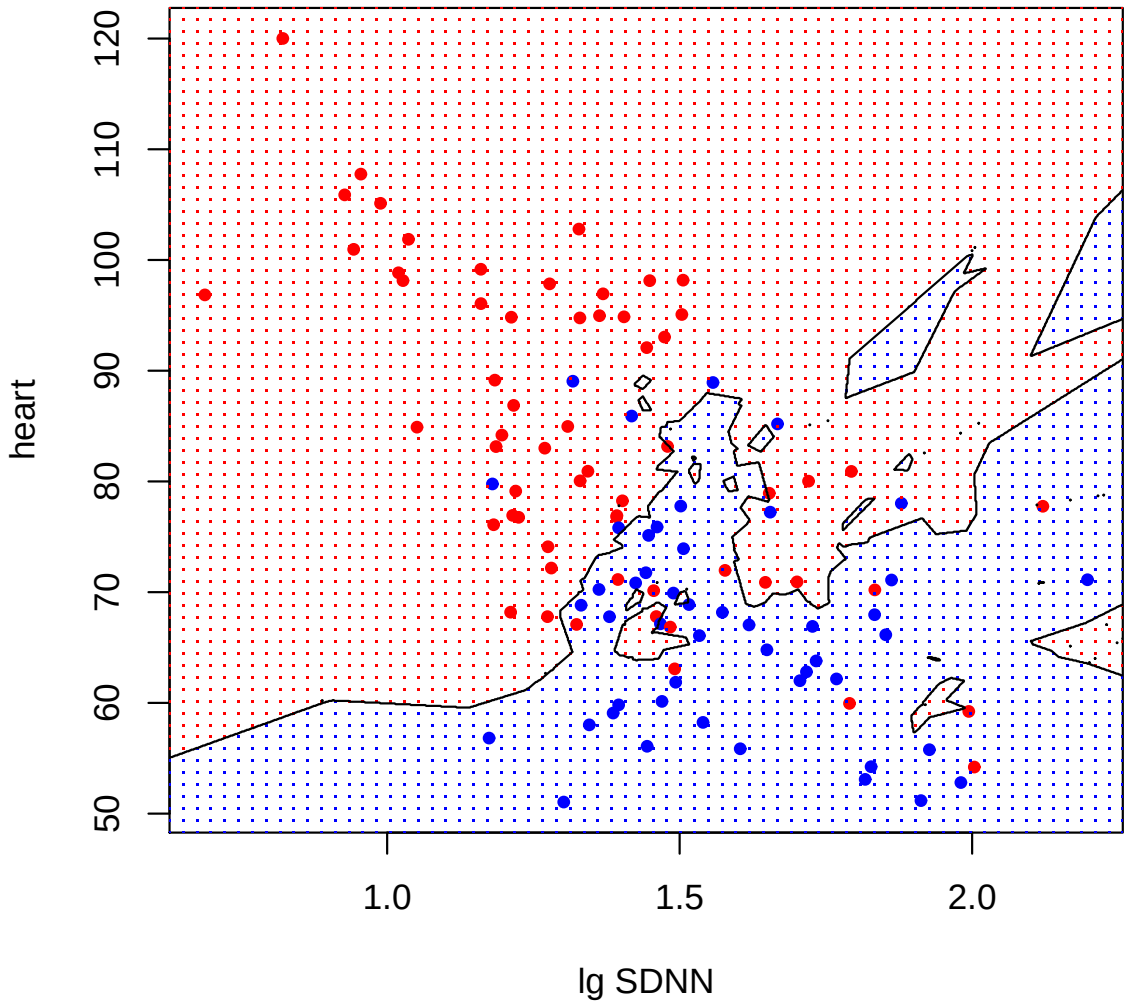
Перекрестная проверка

Метод 3 ближайших соседей 10-CV ошибка 0.26



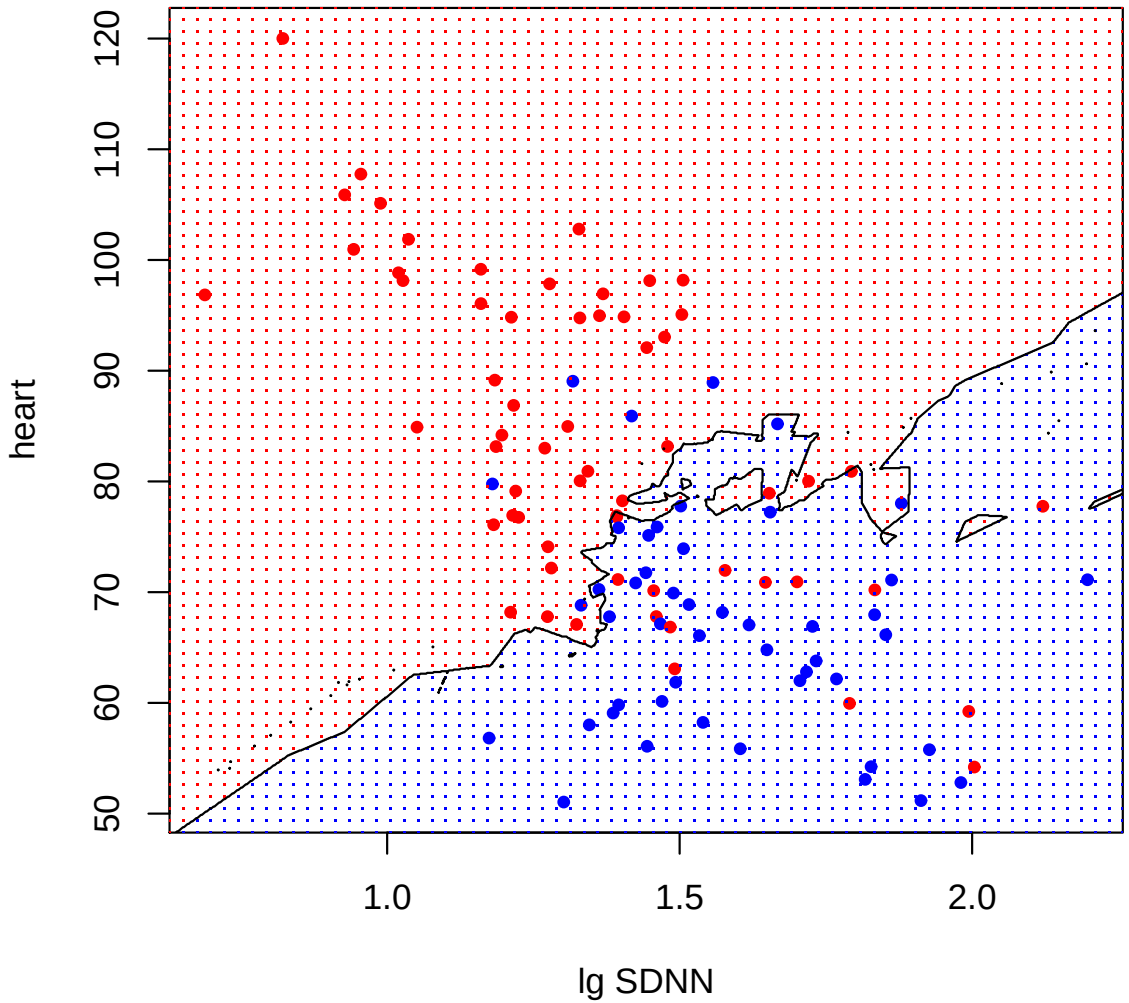
Перекрестная проверка

Метод 5 ближайших соседей 10-CV ошибка 0.27



Перекрестная проверка

Метод 15 ближайших соседей 10-CV ошибка 0.25



Перебор параметров

Делим данные на обучение и контроль (м.б. очень много раз)

При разных значениях параметров обучаемся и проверяем качество

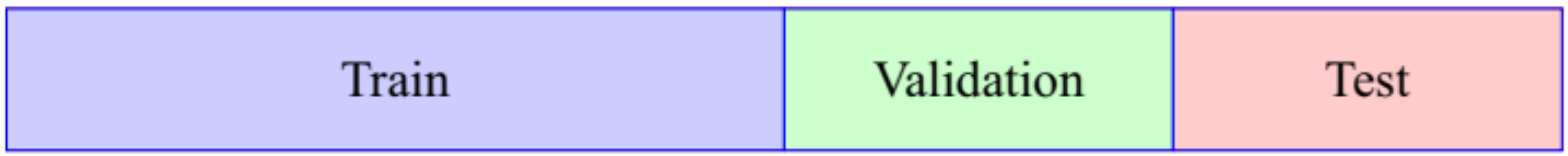
sklearn.model_selection.GridSearchCV

	$k=1$	$k=3$	$k=5$	$k=7$	$k=9$	$k=11$
euclidean	76.0	77.0	79.0	78.5	80.5	82.5
manhattan	74.0	74.0	79.0	79.5	80.5	81.0
chebyshev	76.5	78.5	80.0	80.0	81.0	81.5

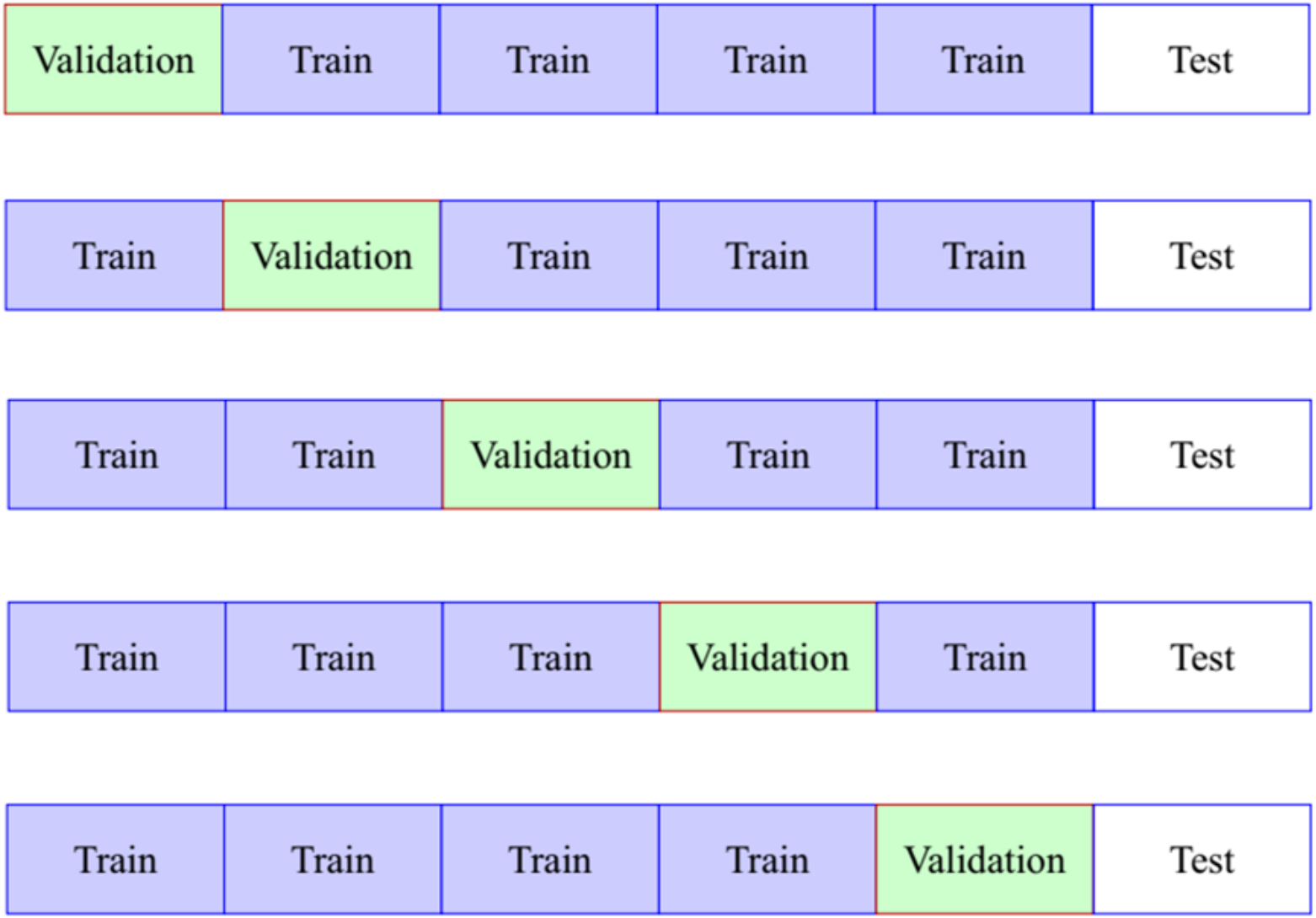
Перекрестная проверка

Если данных достаточно, то их делят

- на обучающую (train) выборку, для построения моделей $f(\cdot,\alpha)$ для разных параметров алгоритма
- на проверочную (validation) выборку, для оценки среднего риска каждой из построенной модели и выбора наилучшей модели
- на тестовую (test) выборку, для оценки ошибки предсказания выбранной модели



Перекрестная проверка



Функционалы качества

Задача — ДНК

Дано
Найти
Критерий

Построить алгоритм легко!
Чтобы улучшить... надо уметь оценивать

~~Метрики~~
- функции ошибки
- функционалы качества

Функции ошибки / функционалы качества
Пожалуй, самое главное при решении задачи...
иногда важнее данных!

Функционалы качества бинарной классификации

Рассматриваем задачу классификации на два класса
Обозначим метки 0 и 1, т.е. $\mathcal{Y} = \{0,1\}$
y_i — метка i-го объекта, $a_i = f(x_i)$ — ответ на этом объекте нашего алгоритма, m — число объектов в выборке

Простым и распространённым функционалом качества является точность (Accuracy или Mean Consequential Error):

$$MCE = \frac{1}{m}\sum_{i=1}^{m} \mathrm{I}(a_i = y_i)$$

Это доля (процент) объектов, на которых алгоритм выдал правильные ответы
Недостаток такого функционала очевиден: он плох в случае дисбаланса классов, когда представителей одного из класса существенно больше, чем другого

Матрица несоответствий / ошибок (confusion matrix)

	$y = 1$	$y = 0$
$a = 1$	TP	FP
$a = 0$	FN	TN

```
♯ построение матрицы в питоне
from sklearn.metrics import confusion_matrix
import pandas import pd
 n = confusion_matrix (y,a) ♯ 1й способ
 n = pd.crosstab (y,a) ♯ 2й способ
```

positive class — объекты, которым алгоритм присвоил ‘1‘
negative class — объекты, которым алгоритм присвоил ‘0‘

TN — true negative, FP — false positive, FN — false negative, TP — true positive

TP — количество правильно классифицированных объектов с меткой ‘1‘

FN — количество объектов с истинной меткой ‘1‘, но прогнозом ‘0‘

FP — количество объектов с истинной меткой ‘0‘, но прогнозом ‘1‘

TN — количество правильно классифицированных объектов с меткой ‘0‘

positive class = TP + FP Python: tp, fn, fp, tn = confusion_matrix (y,a).ravel()

Модельный пример

y	a
0	0
1	0
1	1
0	0
0	1
1	0
1	1
0	0
1	1
0	0

	y=1	y=0
$a = 1$	3	1
$a = 0$	2	4

Замечание. Иногда матрицу несоответствий изображают в другом порядке

Замечание. Стандартная терминология немного нелогична: естественно называть положительными объектами объекты положительного класса, но здесь — объекты, отнесённые алгоритмом к положительному классу (т.е. это даже не свойство объектов, а алгоритма). Но в контексте употребления терминов «истинно положительный» и «ложно положительный» это уже кажется логичным.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$ — общая точность

Ошибки классификатора делятся на две группы: первого и второго рода. В идеале (когда точность равна 100%) матрица несоответствий диагональная, ошибки вызывают отличие от нуля двух недиагональных элементов:

$\alpha = FPR = \dfrac{FP}{FP + TN}$ — вероятность ошибки 1-го рода или false positive rate.

$\beta = FNR = \dfrac{FN}{FN + TP}$ — вероятность ошибки 2-го рода или false negative rate.

Обычно ошибки 1-го и 2-го рода имеют семантику «ложная тревога» и «пропуск цели», соответственно.

Сдача зачета. Преподаватель выступит в роли «злобного классификатора», цель которого — выявить нерадивых студентов (0 — зачёт, 1 — пересдача). В таком случае, ошибка 1-го рода (ложная тревога) — это «учил, но не сдал», а ошибка 2-го рода (пропуск цели) — «не учил, но сдал».

Функционалы качества

$TPR = \frac{TP}{TP + FN}$ — доля правильно классифицированных объектов с меткой ‘1‘, специфичность или полнота (sensitivity, recall)

$TNR = \frac{TN}{TN + FP}$ — доля правильно классифицированных объектов с меткой ‘0‘, чувствительность (Specificity)

$PPV = \frac{TP}{TP + FP}$ — точность (precision) или positive predictive value

$NPV = \frac{TN}{TN + FN}$ — negative predictive value

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} —$$

коэффициент Мэттьюса (Matthews correlation coefficient), используется для несбалансированных данных

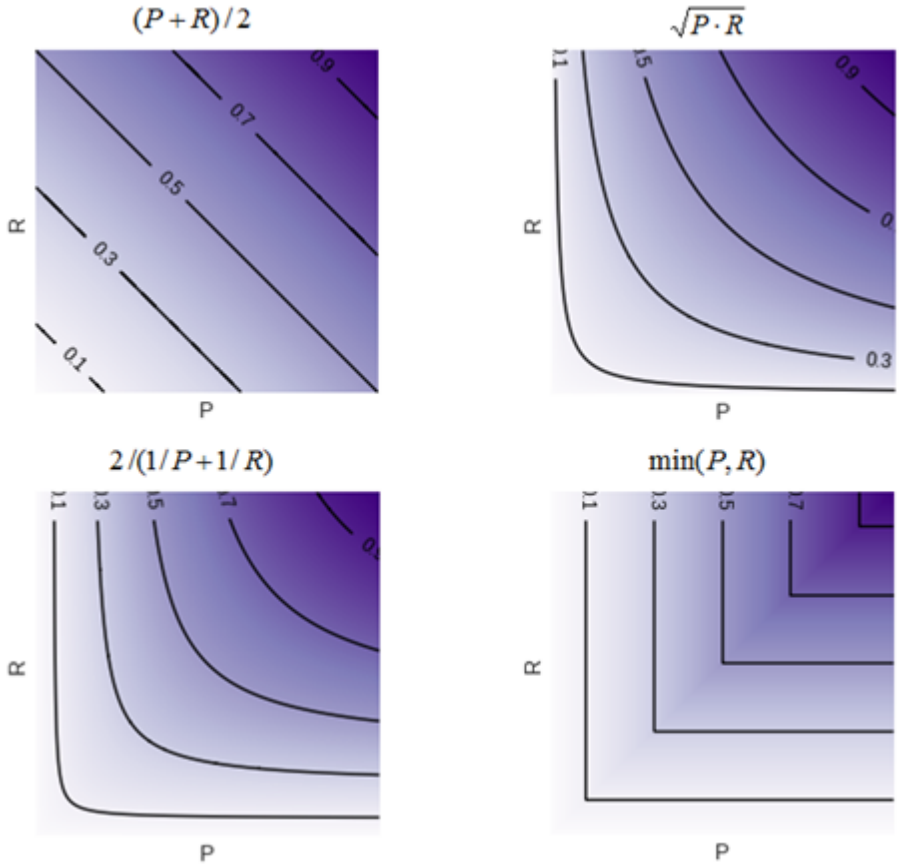
Функционалы качества

F_1-мера (F_1 score) является средним гармоническим точности (precision) и полноты, максимизация этого функционала приводит к одновременной максимизации этих двух «ортогональных критериев»:

$$F_1 = \frac{1}{\frac{1}{TP/(TP+FP)} + \frac{1}{TP/(TP+FN)}} = \frac{2TP}{2TP+FP+FN}$$

Линии уровня среднего гармонического сильно похожи на «уголки», т.е. на линии функции min, что вынуждает при максимизации функционала сильнее «тянуть вверх» меньшее значение.

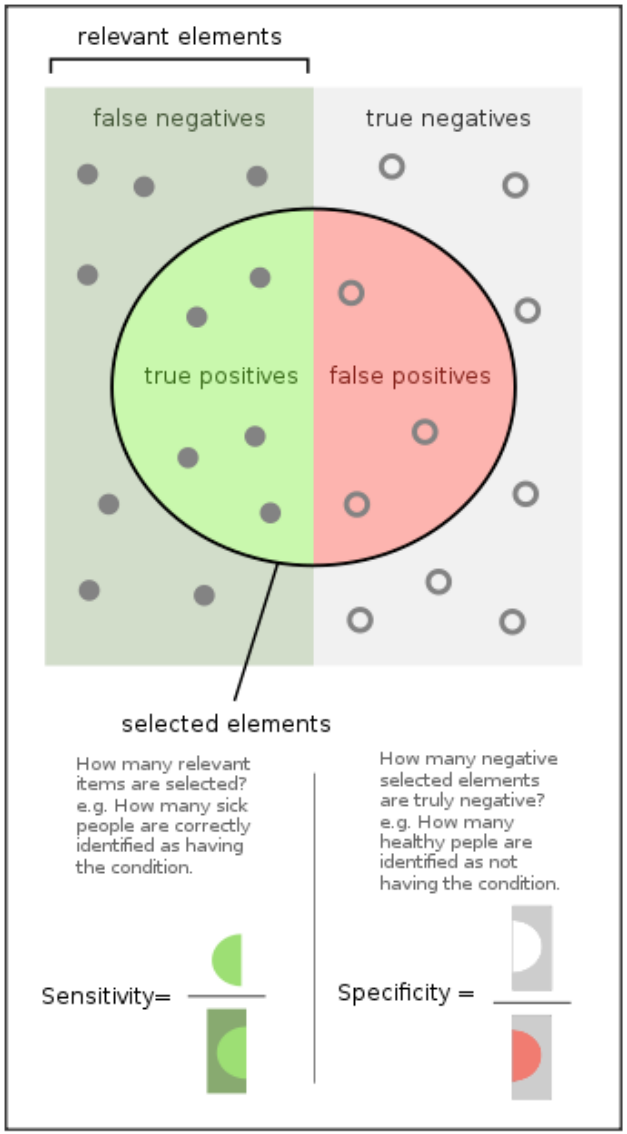
Обозначим P — precision, R — recall



Для несбалансированных данных также применяется
Каппа Коэна (Cohen's Kappa),
сбалансированная точность (Balanced Accuracy)

https://dyakonov.org/2019/05/31/
https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Функционалы качества



Модельный пример

	y=1	y=0
$a = 1$	3	1
$a = 0$	2	4

$$Acc = \frac{3+4}{3+4+1+2} = 0.70; \quad FPR = \frac{1}{1+4} = 0.20; \quad FNR = \frac{2}{2+3} = 0.40;$$

$$TPR = \frac{3}{3+2} = 0.60; \quad TNR = \frac{4}{4+1} = 0.80; \quad PPV = \frac{3}{3+1} = 0.75;$$

$$NPV = \frac{4}{4+2} = 0.66; \quad F_1 = \frac{2 \cdot 3}{2 \cdot 3 + 1 + 2} = 0.66;$$

$$MCC = \frac{3 \cdot 4 - 1 \cdot 2}{\sqrt{(3+1)(3+2)(4+1)(4+2)}} = 0.41$$

ROC-кривая (receiver operating characteristic curve)

Пусть алгоритм выдаёт некоторую оценку (не обязательно вероятность) принадлежности объекта к классу '1'.
Можно считать, что оценка принадлежит отрезку $[0, 1]$.
Часто результат работы алгоритма на фиксированной тестовой выборке визуализируют с помощью ROC-кривой, а качество оценивают как площадь под этой кривой — AUC (area under the curve).
Пример. Пусть алгоритм выдал оценки, как показано в табл. 1. Упорядочим строки табл. 1 по убыванию ответов алгоритма — получим табл. 2. Ясно, что в идеале её столбец «класс» тоже станет упорядочен (сначала идут 1, потом 0); в самом худшем случае – порядок будет обратный (сначала 0, потом 1); в случае «слепого угадывания» будет случайное распределение 0 и 1.

id	оценка	класс
1	0.5	0
2	0.1	0
3	0.2	0
4	0.6	1
5	0.2	1
6	0.3	1
7	0.0	0

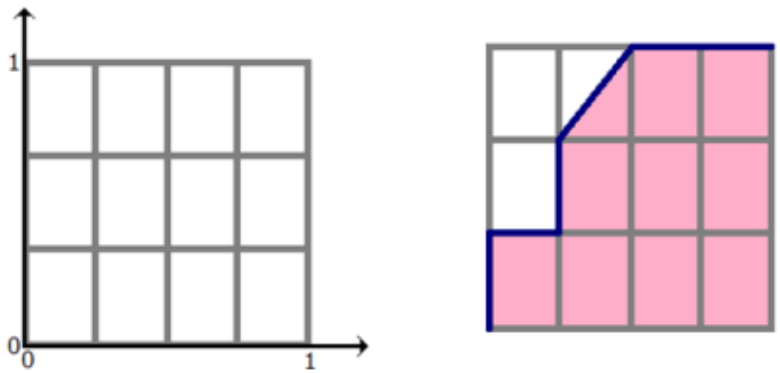
id	оценка	класс
4	0.6	1
1	0.5	0
6	0.3	1
3	0.2	0
5	0.2	1
2	0.1	0
7	0.0	0

Чтобы нарисовать ROC-кривую, надо взять единичный квадрат на координатной плоскости, разбить его на m равных частей горизонтальными линиями и на n — вертикальными, где m — число 1 среди правильных меток теста (в нашем примере $m = 3$), n — число нулей ($n = 4$).

В результате квадрат разбивается сеткой на $m \times n$ блоков.

ROC-кривая

Теперь будем просматривать строки табл. 2 сверху вниз и прорисовывать на сетке линии, переходя их одного узла в другой. Стартуем из точки $(0, 0)$. Если значение метки класса в просматриваемой строке 1, то делаем шаг вверх; если 0, то делаем шаг вправо. Ясно, что в итоге мы попадём в точку $(1, 1)$, т.к. сделаем в сумме m шагов вверх и n шагов вправо.



На рисунке показан путь для нашего примера — это и является ROC-кривой.

ROC-кривая

Важный момент: если у нескольких объектов значения оценок равны, то мы делаем шаг в точку, которая на a блоков выше и b блоков правее, где a — число единиц в группе, b — число нулей. В частности, если все объекты имеют одинаковую метку, то мы сразу шагаем из точки $(0, 0)$ в точку $(1, 1)$.

AUC ROC — площадь под ROC-кривой — часто используют для оценивания качества упорядочивания алгоритмом объектов двух классов. Ясно, что это значение лежит на отрезке $[0, 1]$. В нашем примере AUC ROC $= 9.5/12 \approx 0.79$.

Выше мы описали случаи идеального, наихудшего и случайного следования меток в упорядоченной таблице. Идеальному соответствует ROC-кривая, проходящая через точку $(0, 1)$, площадь под ней равна 1.

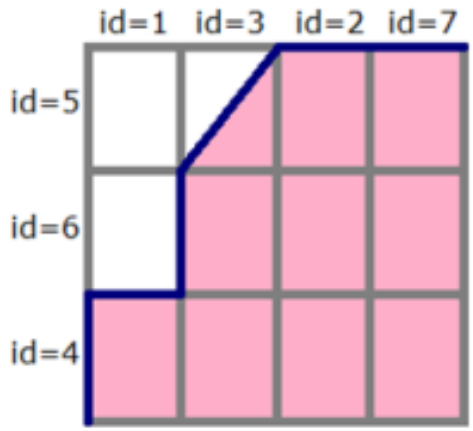
Наихудшему – ROC-кривая, проходящая через точку $(1, 0)$, площадь под ней — 0.

Случайному — что-то похожее на диагональ квадрата, площадь примерно равна 0.5.

Замечание. ROC-кривая считается неопределённой для тестовой выборки целиком состоящей из объектов только одного класса. Большинство современных реализаций выдают ошибку при попытки построить её в этом случае.

Смысл AUC ROC

Сетка на рисунке разбила квадрат на mn блоков. Каждый закрашенный блок на рисунке соответствует паре (объект класса 1, объект класса 0), для которой наш алгоритм правильно предсказал порядок (объект класса 1 получил оценку выше, чем объект класса 0), незакрашенный блок — паре, на которой ошибся.



Таким образом, **AUC ROC равен доле пар объектов вида (объект класса 1, объект класса 0), которые алгоритм верно упорядочил**, т.е. первый объект идёт в упорядоченном списке раньше.

ROC-кривая

$$\text{AUC ROC} = \frac{\sum_{i=1}^{q}\sum_{j=1}^{q} \mathrm{I}(y_i < y_j)\,\mathrm{I}^*(a_i < a_j)}{\sum_{i=1}^{q}\sum_{j=1}^{q} \mathrm{I}(y_i < y_j)}$$

$$\mathrm{I}^*(a_i < a_j) = \begin{cases} 0, & a_i > a_j \\ 0.5, & a_i = a_j \\ 1, & a_i < a_j \end{cases} \qquad \mathrm{I}(y_i < y_j) = \begin{cases} 0, & y_i \geq y_j \\ 1, & y_i < y_j \end{cases}$$

q — число объектов в тесте.

Подробнее о ROC-кривой:

https://en.wikipedia.org/wiki/Receiver_operating _characteristic

https://www.researchgate.net/publication/222511520_Introduction_to _ROC_analysis

Принятие решений на основе ROC-кривой

На практике нам часто надо будет решить:

какие объекты отнести к классу 1, а какие к классу 0.

Для этого нужно будет выбрать некоторый порог (threshold) (объекты с оценками выше порога считаем принадлежащими классу 1, остальные — 0).

ROC-кривая — график зависимости TPR (доля точек класса 1, которые **верно** классифицированы алгоритмом) от FPR (это доля точек класса 0, которые **неверно** классифицированы нашим алгоритмом) при изменении порога в дискриминантной функции (в Scikit-learn — decision function) бинарного классификатора.

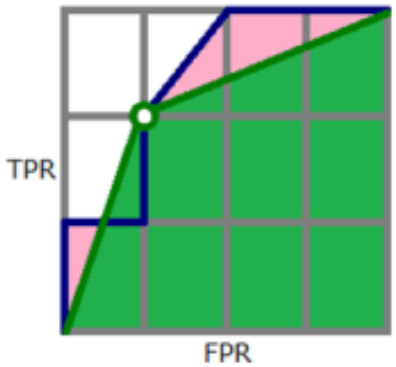
Выбору порога соответствует выбор точки на ROC-кривой.

ROC-кривая

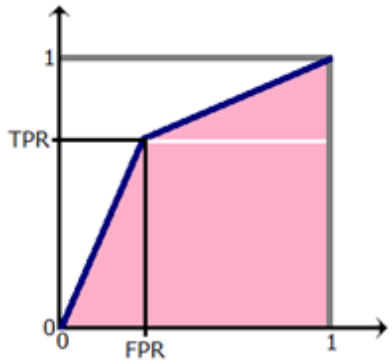
Зададим порог равным 0.25, тогда для нашего примера

id	> 0.25	класс
4	1	1
1	1	0
6	1	1
3	0	0
5	0	1
2	0	0
7	0	0

на ROC-кривой это точка $(1/4, 2/3)$



Заметим, что 1/4 — это процент точек класса 0, которые неверно классифицированы нашим алгоритмом, т.е FPR, 2/3 — процент точек класса 1, которые верно классифицированы нашим алгоритмом, т.е TPR.
На рисунке ниже (зелёным) показана ROC-кривая бинаризованного решения, заметим, что AUC после бинаризации уменьшился и стал равным $8.5/12 \approx 0.71$.



В общем случае AUC ROC для бинарного решения равна

$$\frac{TPR \cdot FPR}{2} + TPR \cdot (1 - FPR) + \frac{(1 - TPR)(1 - FPR)}{2} = \frac{1 + TPR - FPR}{2}$$

как сумма площадей двух треугольников и квадрата.

Алгоритм построения ROC-кривой

Дано: выборка из объектов $\{(x_1,y_1),\ldots,(x_q,y_q)\}$, множество ответов алгоритма $a_1,\ldots,a_q$
Результат: $\{(TPR_j,FRP_j)\}_{j=0}^{q}$ — точки ROC-кривой, AUC
Для простоты предположим, что все ответы a_j не совпадают.

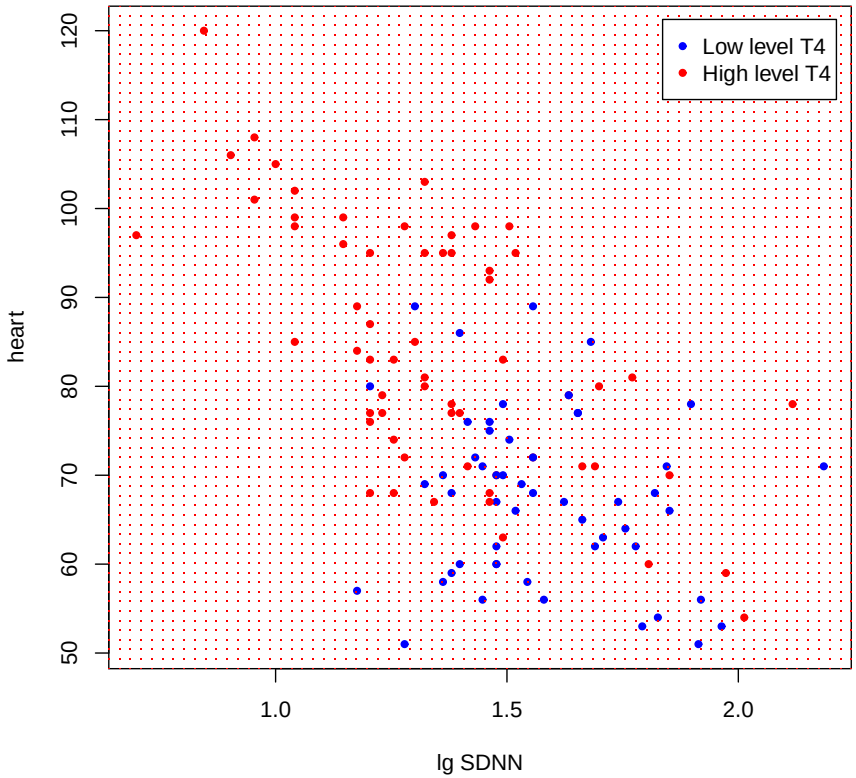
Алгоритм.

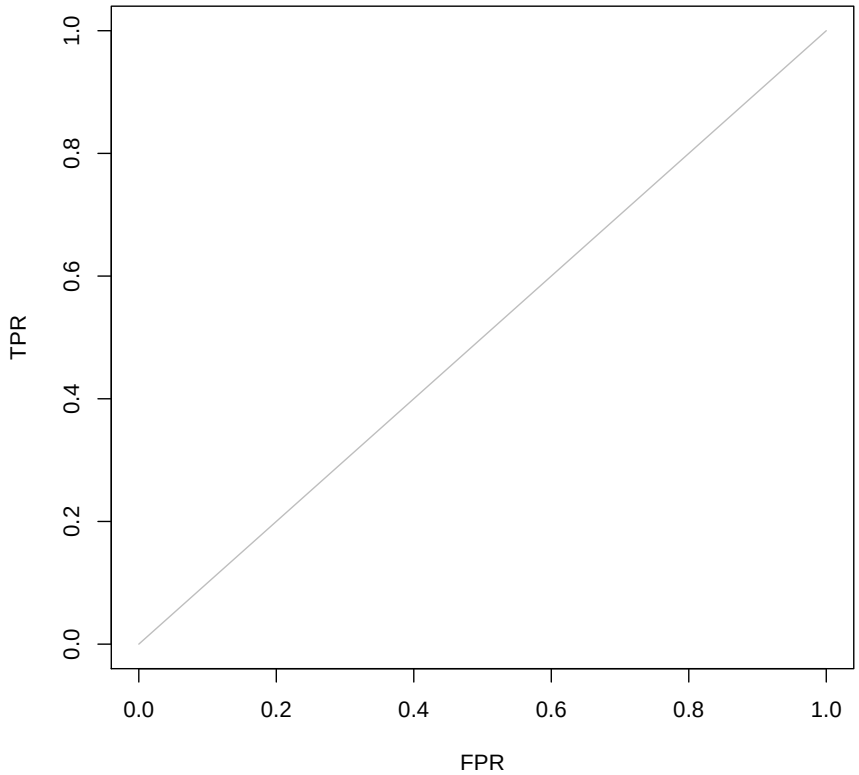
1. **вычислить** количество представителей классов 0 и 1 в выборке:

$$q_0 = \sum_{j=1}^{q} \mathrm{I}(y_j = 0), \quad q_1 = \sum_{j=1}^{q} \mathrm{I}(y_j = 1)$$

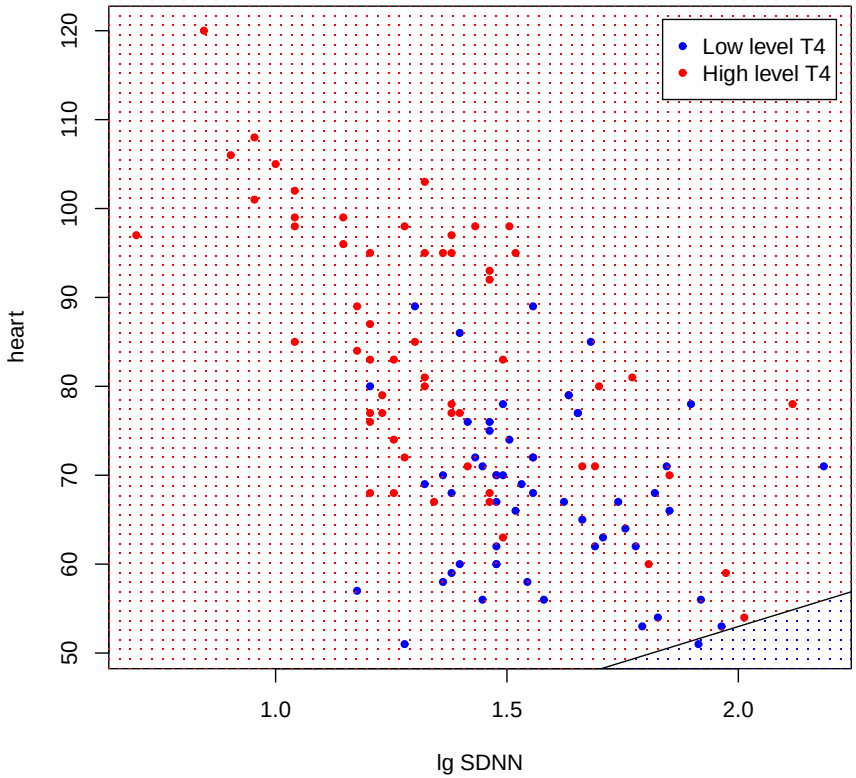
2. **упорядочить** объекты $x_1,\ldots,x_q$ по убыванию ответов a_j
3. **задать** начальную точку ROC-кривой: $(TPR_0,FRP_0) = (0,0)$, $AUC = 0$
4. для всех $j = 1,\ldots,q$
   если $y_j = 0$, то сместиться на один шаг вправо:
   $FPR_j = FPR_{j-1} + \frac{1}{q_0}$, $TPR_j = TPR_{j-1}$,
   $AUC = AUC + \frac{1}{q_0} TPR_j$
   иначе сместиться на один шаг вверх:
   $FPR_j = FPR_{j-1}$, $TPR_j = TPR_{j-1} + \frac{1}{q_1}$,
   $AUC = AUC + \frac{1}{q_0} TPR_j$

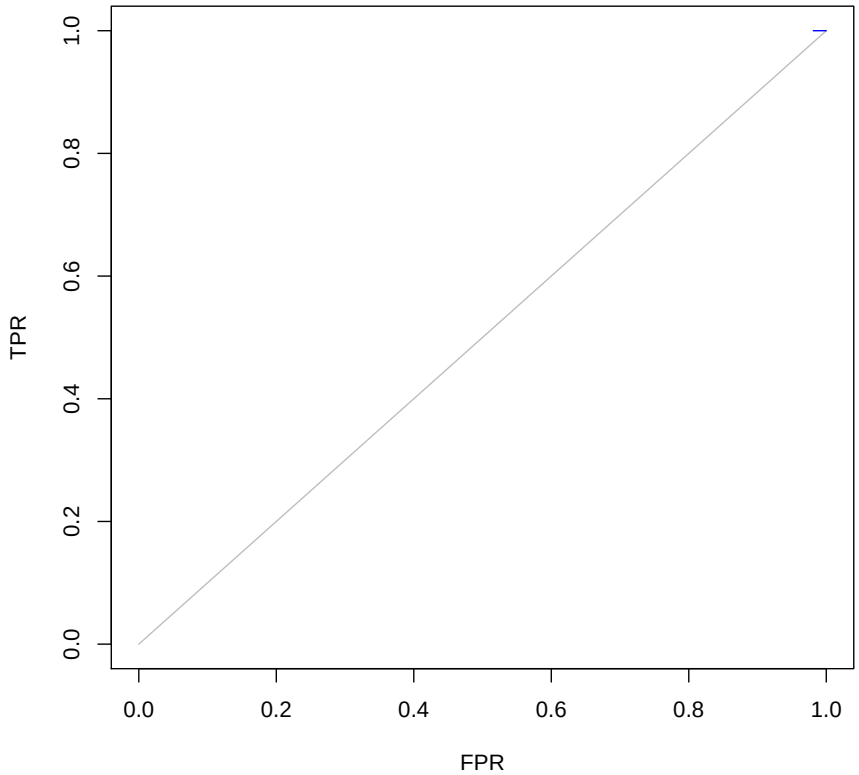
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

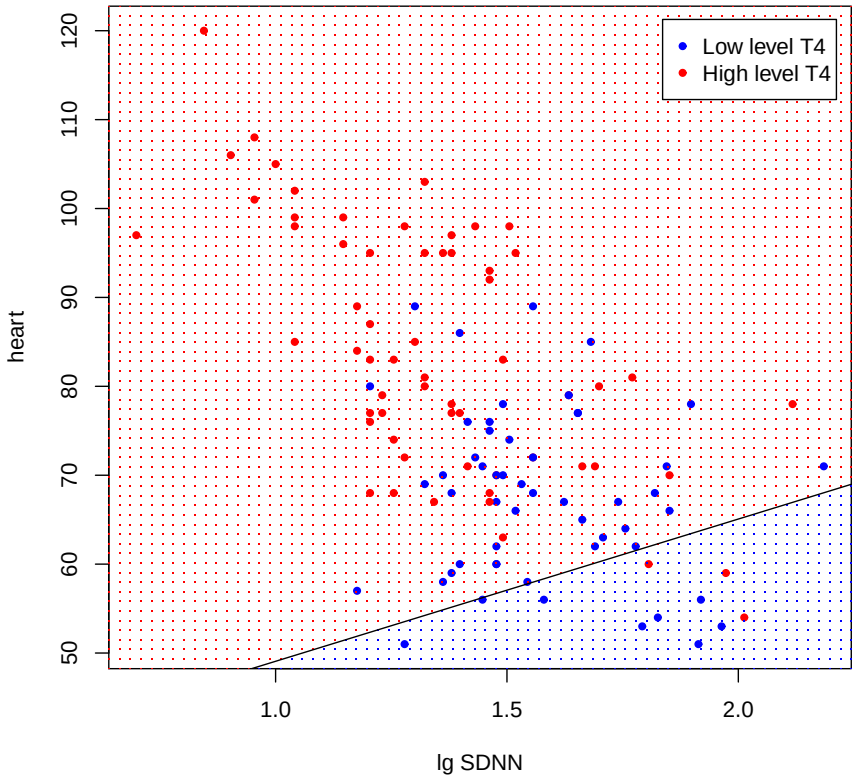

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

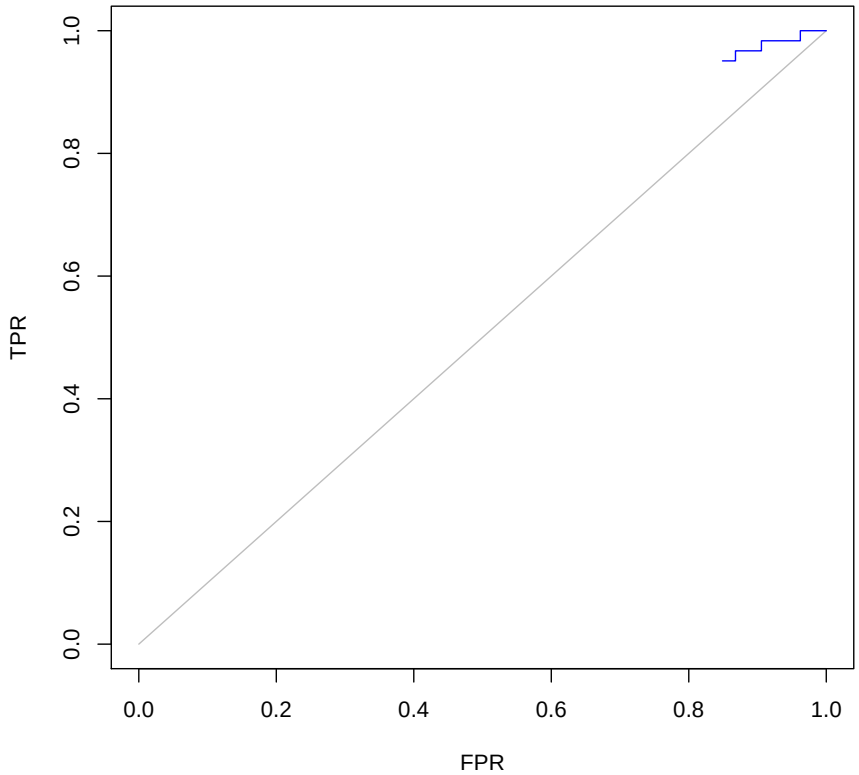
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

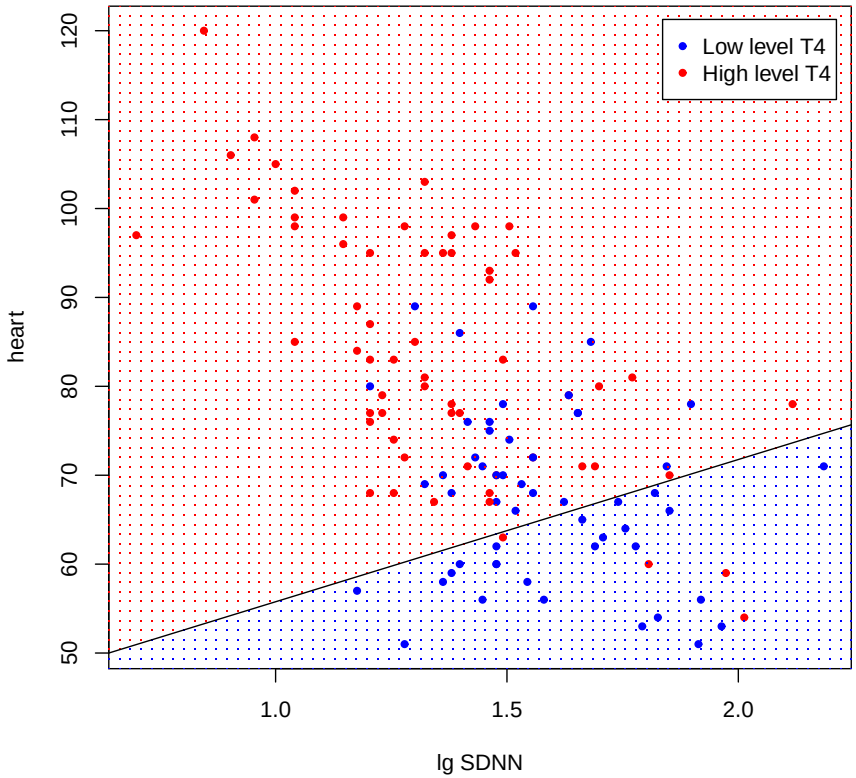

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

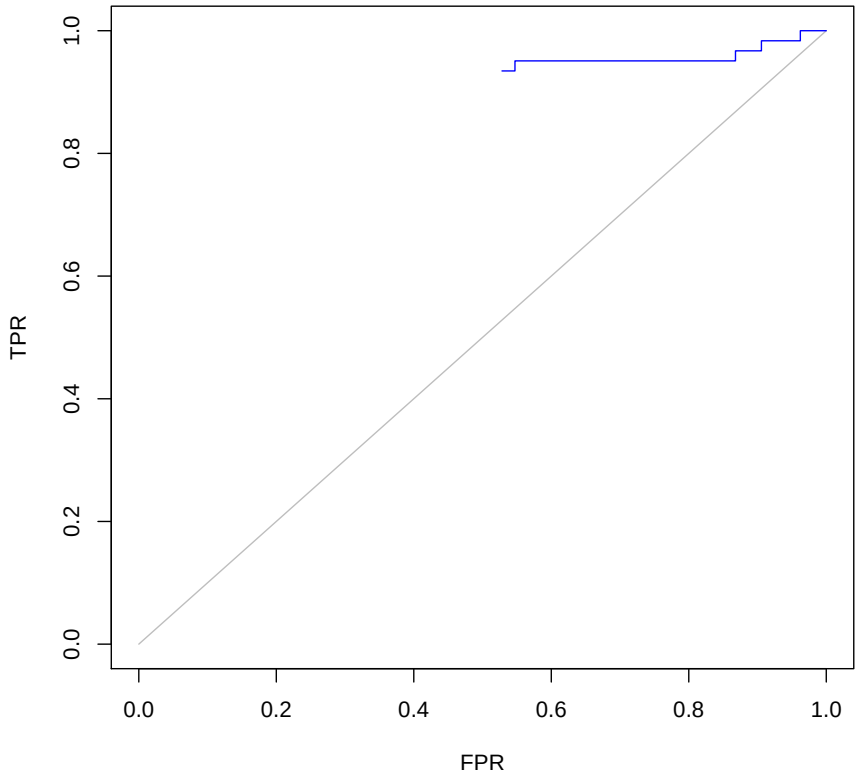
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

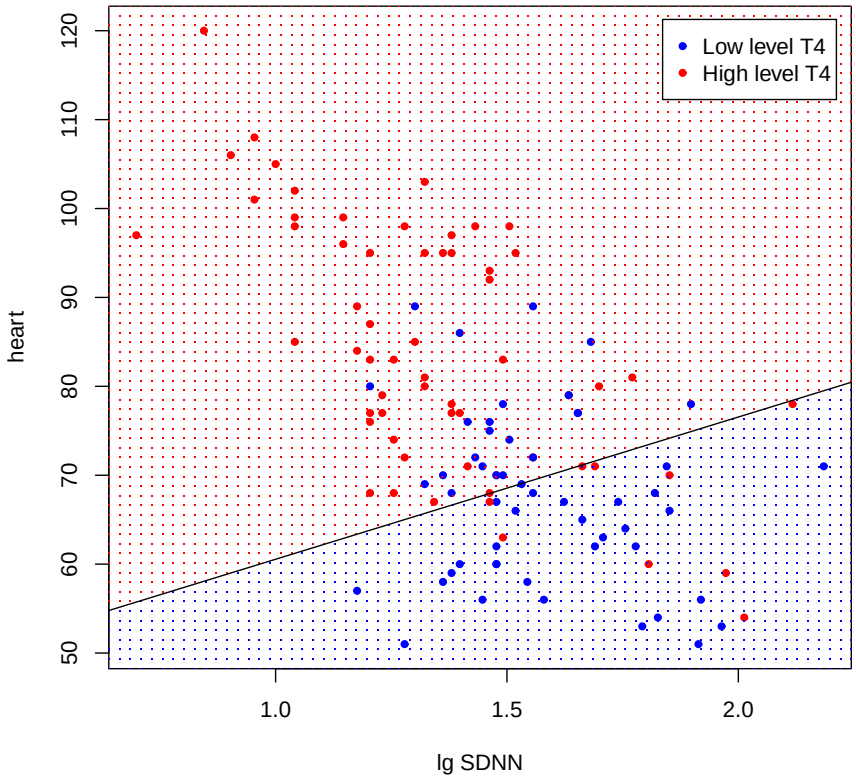

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

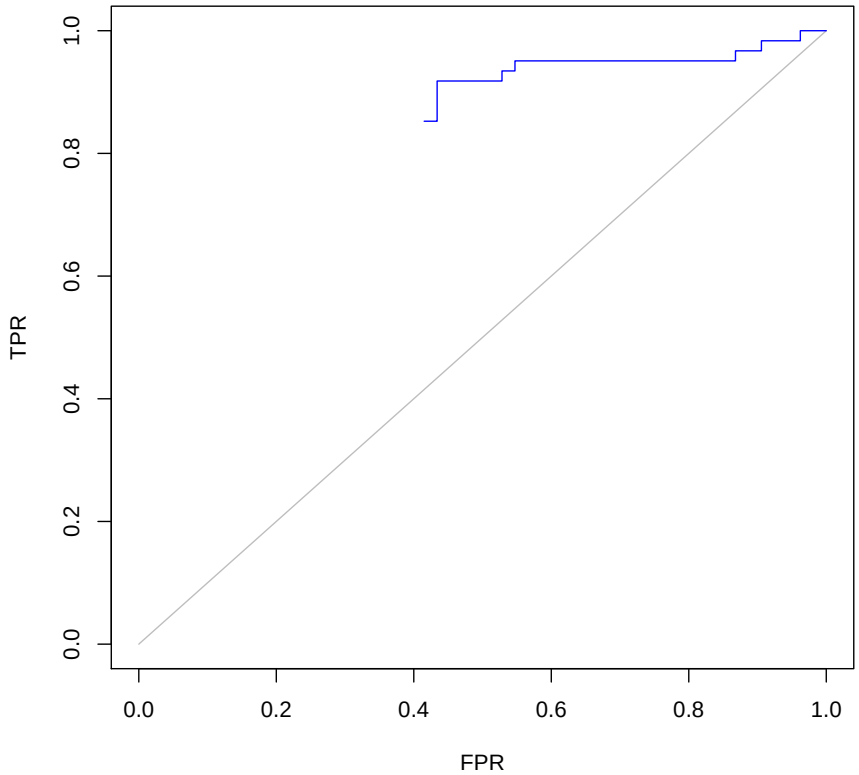
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

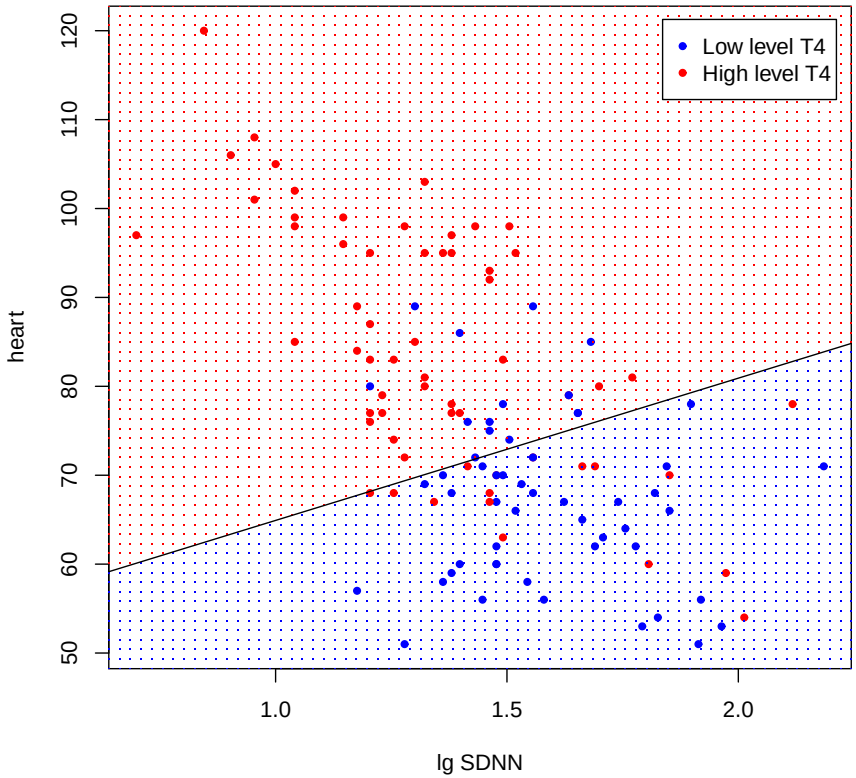

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

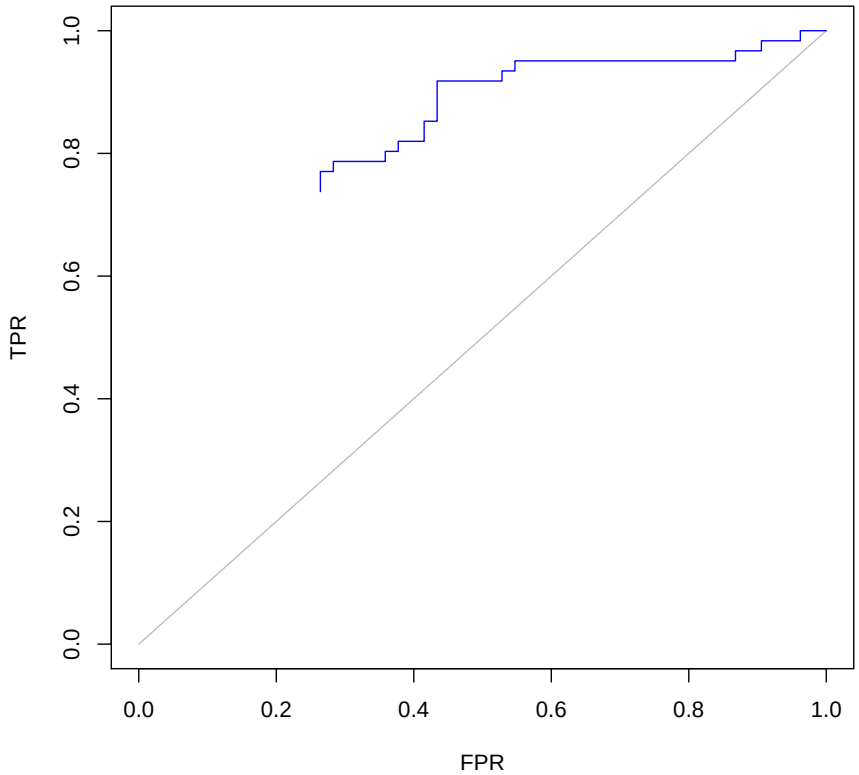
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

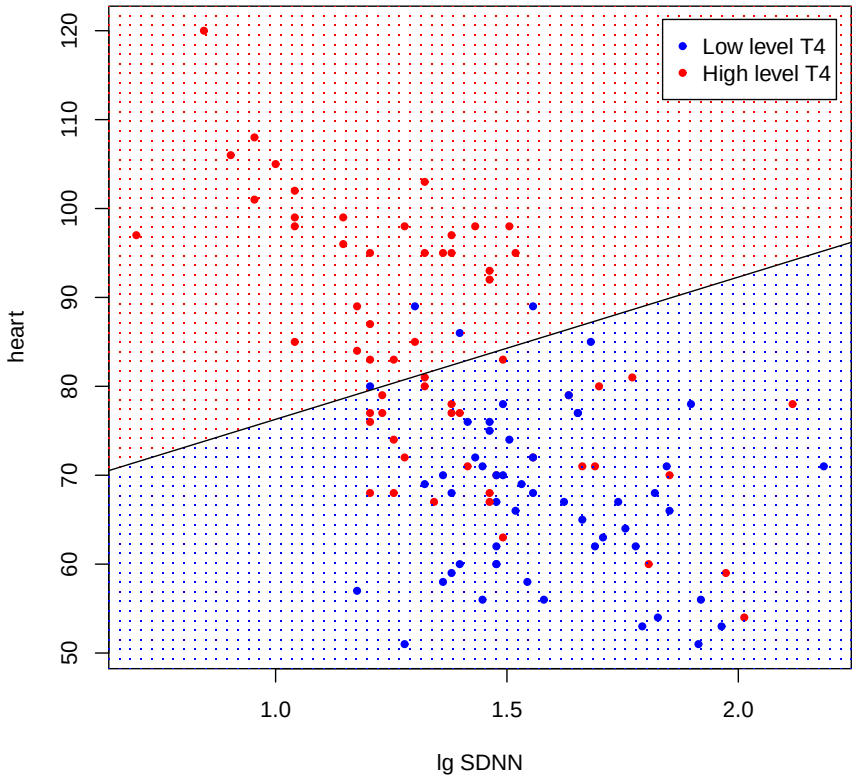

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

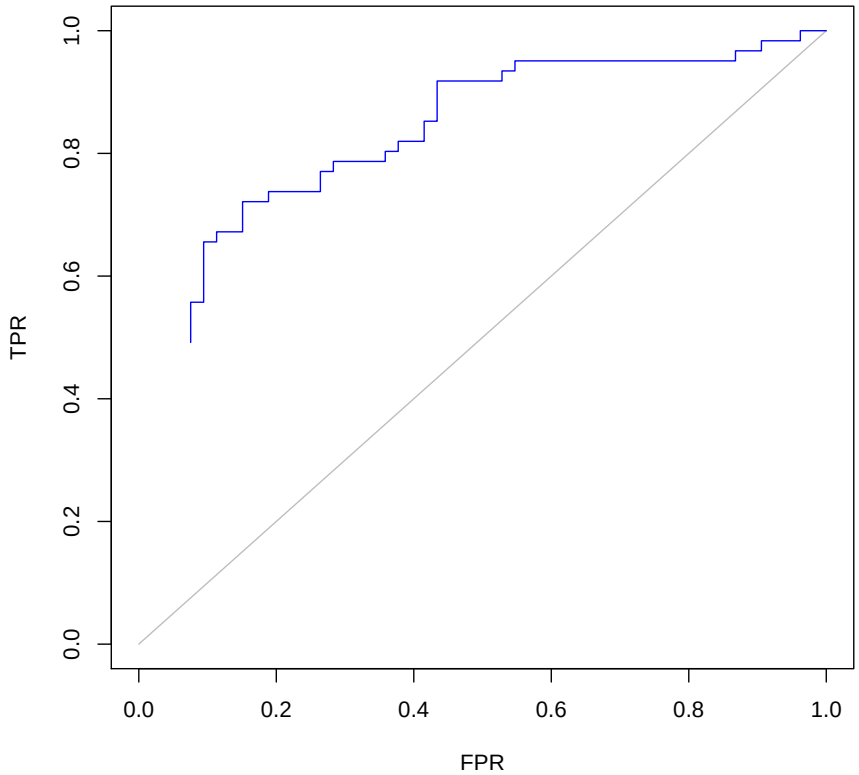
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

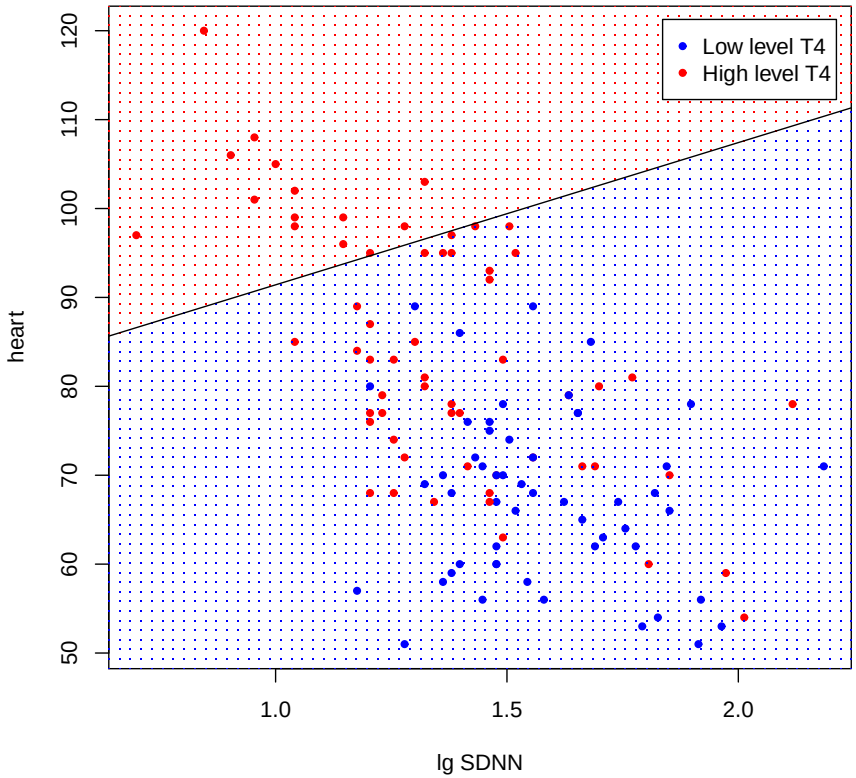

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

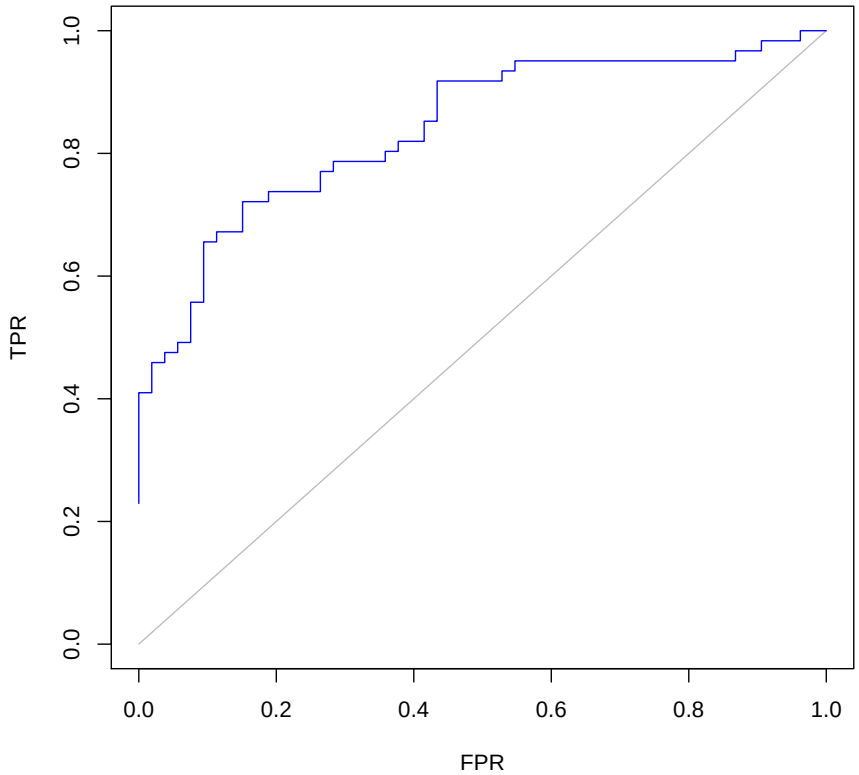
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0

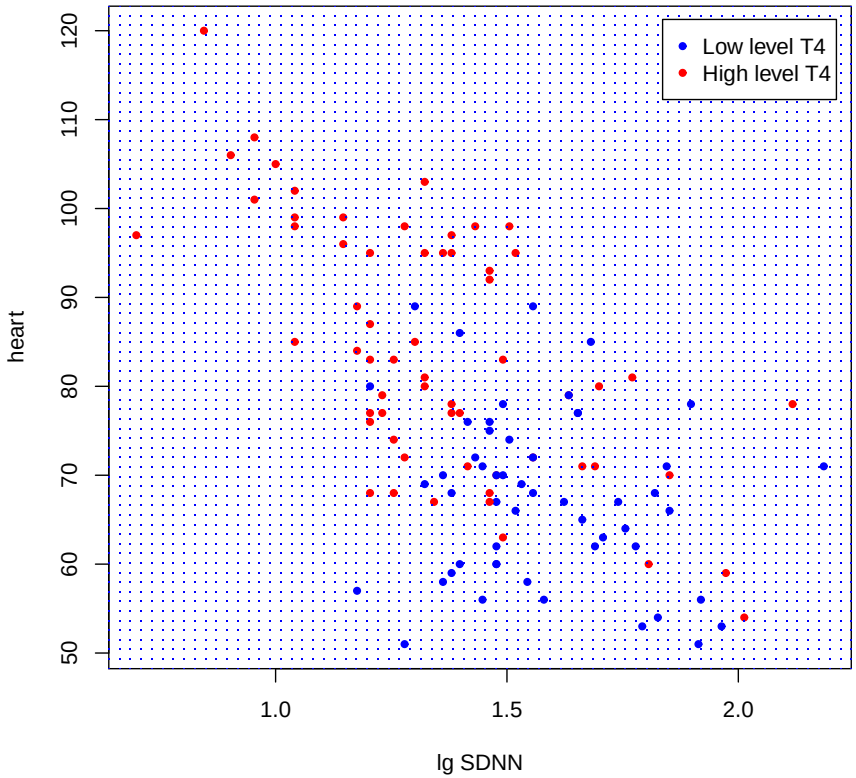

TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

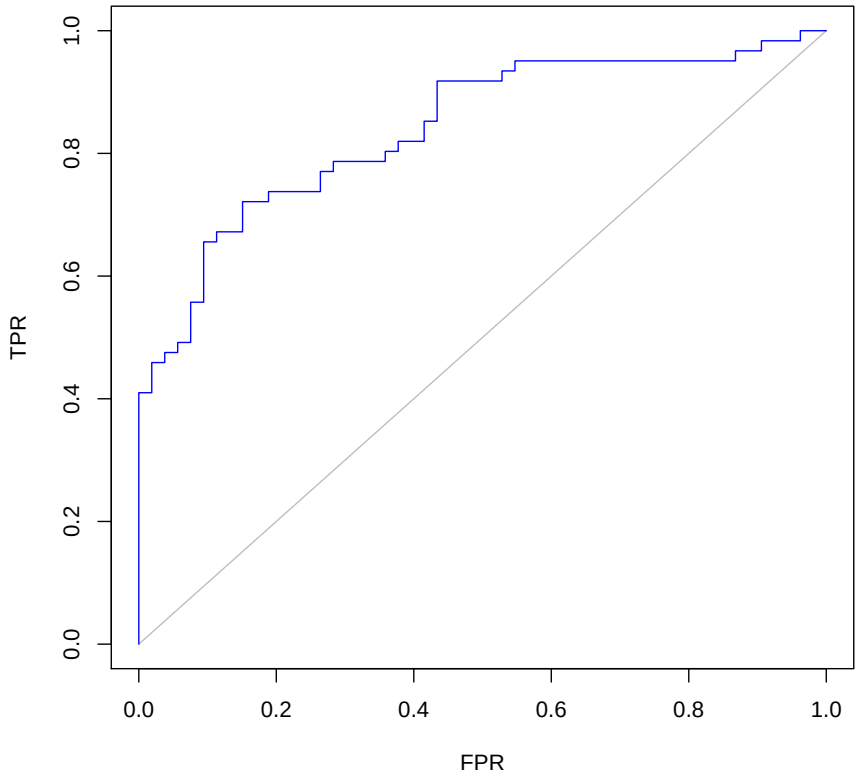
ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0


TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

ROC-кривая


Low level T4
High level T4
heart
lg SDNN
50
60
70
80
90
100
110
120
1.0
1.5
2.0


TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

Максимизация AUC ROC на практике

Оптимизировать AUC ROC напрямую затруднительно по нескольким причинам:

- эта функция недифференцируема по параметрам алгоритма
- она в явном виде не разбивается на отдельные слагаемые, которые зависят от ответа только на одном объекте

Есть несколько подходов к оптимизации

- замена индикаторной функции на похожую дифференцируемую функцию
- использование смысла функционала (если это вероятность верного упорядочивания пары объектов, то можно перейти к новой выборке, состоящей из пар)
- ансамблирование нескольких алгоритмов с преобразованием их оценок в ранги

Замечания

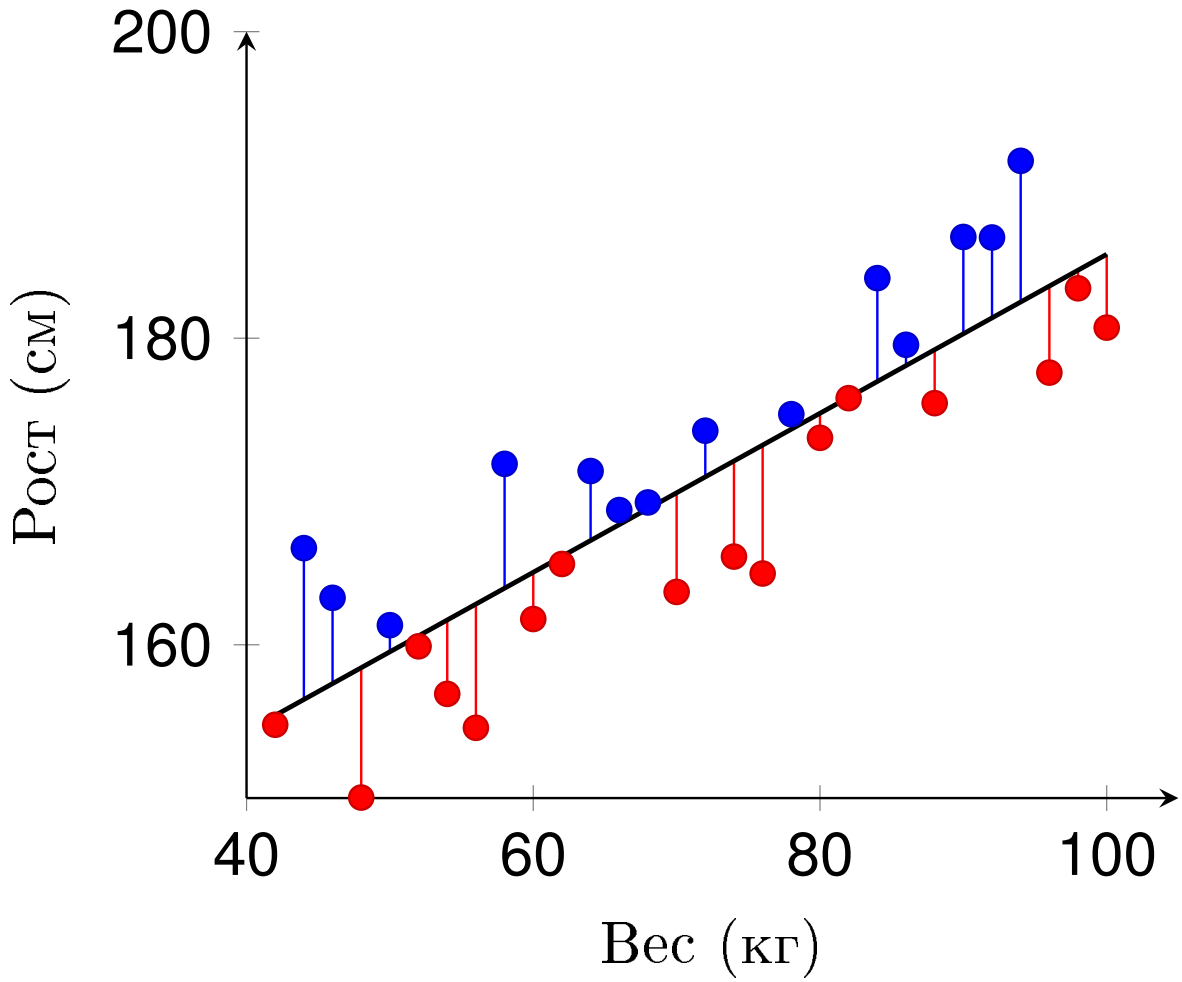
- AUC ROC не зависит от строго возрастающего преобразования ответов алгоритма (например, возведения в квадрат), поскольку зависит не от самих ответов, а от меток классов объектов при упорядочении по этим ответам.
- Часто используют критерий качества Gini, он принимает значение на отрезке $[-1, +1]$:
$$Gini = 2 \cdot \text{AUC ROC} - 1.$$
- AUC ROC можно использовать для оценки качества признаков. Считаем, что значения признака — это ответы нашего алгоритма, тогда выражение $2 \cdot |\text{AUC ROC} - 0.5|$ подойдет для оценки качества признака: оно максимально, если по этому признаку два класса строго разделяются и минимально, если они «перемешаны».
- Приведённый алгоритм является частным случаем и не учитывает одинаковые ответы алгоритма для разных меток. Обобщение простое.
- AUC ROC для задач с сильным дисбалансом классов (не просто разная мощность классов, а разное отношение) следует применять осторожно или использовать обобщения AUC.pdf.

Замечания

- «завышает» качество
- оценивает не конкретный классификатор, а регрессию
- сложно объяснить заказчику
- учитывает разную мощность классов
- В банковском скоринге AUC ROC очень популярный функционал, хотя очевидно, что он также здесь не очень подходит. Банк может выдать ограниченное число кредитов, поэтому главное требование к алгоритму — чтобы среди объектов, которые получили наименьшие оценки были только представители класса 0 («вернёт кредит», если мы считаем, что класс 1 — «не вернёт» и алгоритм оценивает вероятность невозврата).
- Об ROC и AUC ROC
  https://dyakonov.org/2017/07/28/
  https://dyakonov.org/2015/10/09/

Меры точности в задаче регрессии

Функции ошибки в задачах регрессии



Меры точности в задаче регрессии

Средний модуль отклонения (Mean Absolute Error):

$$MAE = \frac{1}{m}\sum_{j=1}^{m}|y_j - a_j|$$

Замечание. Если мы строим постоянную регрессию (константу), то решением задачи $MAE \to \min$ будет медиана

$$a = \mathrm{median}\{y_1 \ \ldots, y_m\}$$

Взвешенный вариант MAE (weighted MAE):

$$wMAE = \frac{1}{m}\sum_{j=1}^{m} w_j|y_j - a_j|$$

Меры точности в задаче регрессии

Средний квадрат отклонения (Mean Squared Error):

$$MSE = \frac{1}{m}\sum_{j=1}^{m}|y_j - a_j|^2$$

RMSE (Root Mean Squared Error):

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{m}\sum_{j=1}^{m}|y_j - a_j|^2}$$

Замечание. Если мы строим постоянную регрессию, то решением задачи $MSE \to \min$ будет выборочное среднее:

$$\overline{a} = \frac{1}{m}\sum_{j=1}^{m} y_j$$

Меры точности в задаче регрессии

Коэффициент детерминации (Coefficient of determination):

$$R^2 = \frac{\frac{1}{m}\sum_{j=1}^{m}|y_j - a_j|^2}{\frac{1}{m}\sum_{j=1}^{m}|\overline{y} - y_j|^2}, \quad \overline{y} = \frac{1}{m}\sum_{j=1}^{m} y_j$$

Замечание. Коэффициент детерминации можно использовать только совместно с MSE.
В R^2 происходит нормировка: MSE-ошибка алгоритма делится на MSE-ошибку постоянной регрессии. Принимает максимальное значение 1 в случае абсолютно точной регрессии, значение 0 для постоянной регрессии.

Откуда берётся MSE: ещё одно «оправдание»

Пусть функция ошибки: $L(y,a) = g(y-a)$

Логично предположить:

1. $g(0) = 0$
2. $|z_1| \leq |z_2| \;\Rightarrow\; g(z_1) \leq g(z_2)$
3. достаточно гладкая, т.е.

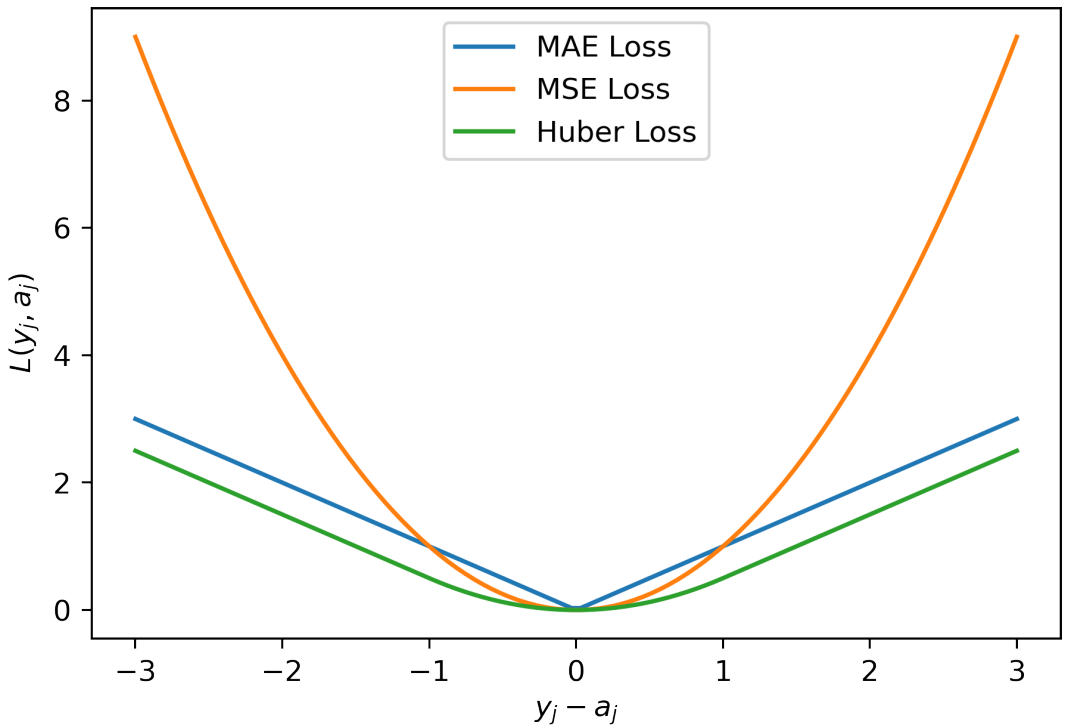
$$g(z) = g(0) + g'(0)z + \frac{g''(0)}{2}z^2 + o\left(z^2\right)$$

Тогда в окрестности $y = a$

$$L(y,a) \approx \underbrace{g(0)}_{=0 \text{ из } 1.} + \underbrace{g'(0)(y-a)}_{=0 \text{ из } 2.} + \frac{g''(0)}{2}(y-1)^2 = \underbrace{C}_{>0}(y-a)^2$$

Функция ошибки Хьюбера (Huber loss):

$$H_\delta = \sum_{j=1}^{m} H_\delta(y_j, a_j), \quad H_\delta(y_j, a_j) = \begin{cases} \frac{1}{2}|y_j - a_j|^2, & |y_j - a_j| \le \delta, \\ \delta\left(|y_j - a_j| - \frac{1}{2}\delta^2\right), & |y_j - a_j| > \delta \end{cases}$$



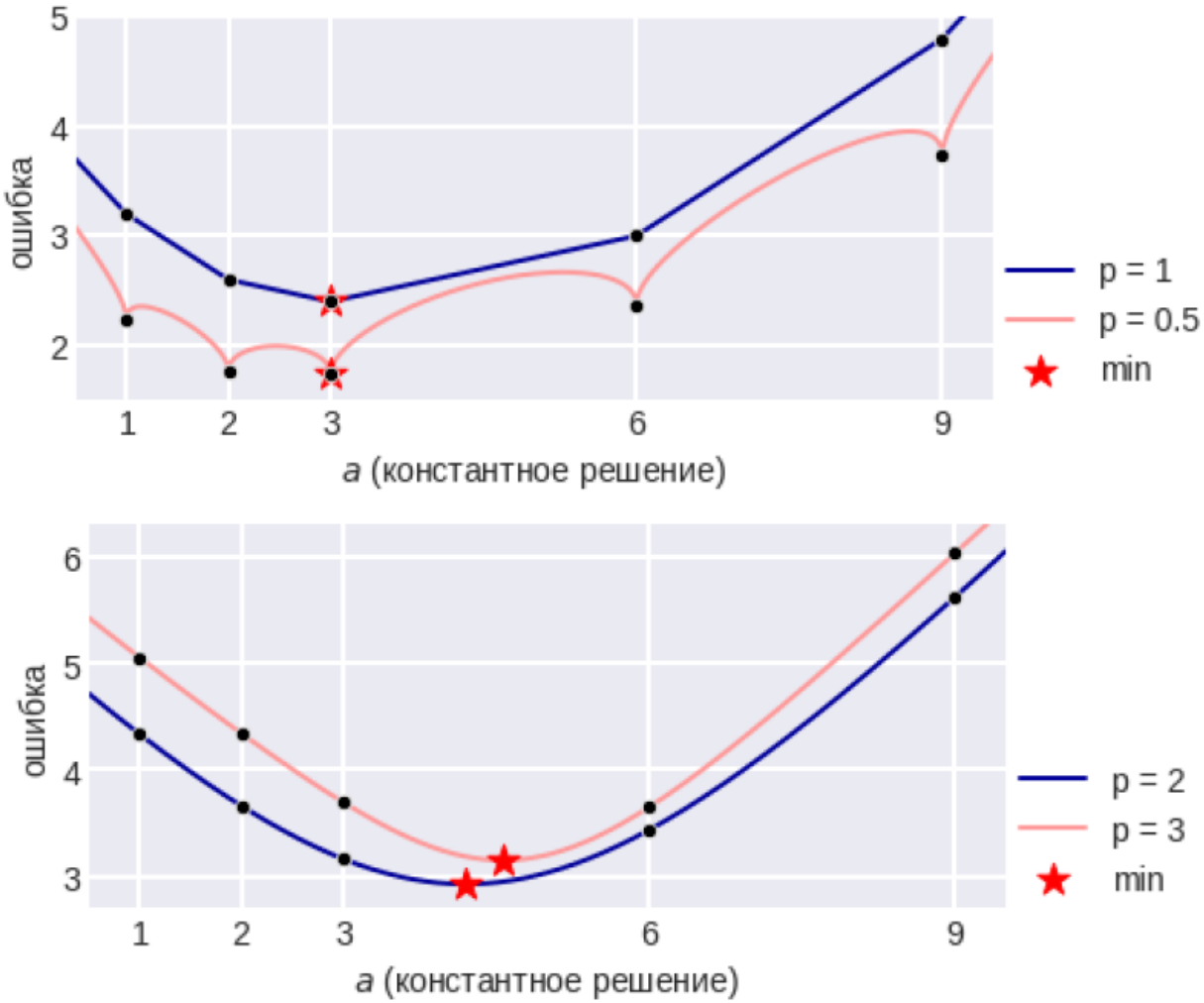
Обобщения

$$\sqrt[p]{\sum_{i=1}^{m} w_i |\varphi(a_i) - \varphi(y_i)|^p}, \quad \sum_{i=1}^{m} w_i = 1, \quad w_i \geq 0$$

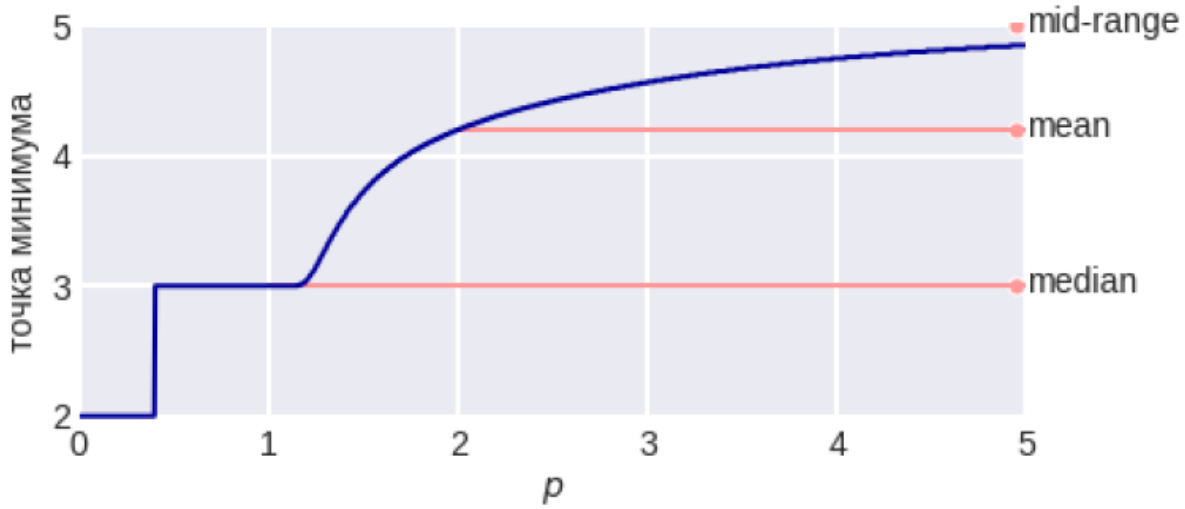
Что делать с таким функционалом качества:

1. Преобразование целевого вектора $\varphi(y)$ (затем $\varphi^{-1}(y)$)
2. Веса можно рассматривать как вероятности появления объектов и использовать модели, которые поддерживают веса объектов
3. В случае нетривиальных p — прямая настройка

Про нетривиальные p



Как точка минимума зависит от степени



Меры точности в задаче регрессии

Следующие функции пытаются измерить ошибку в процентах.

Средний процент отклонения (Mean Absolute Percent Error):

$$MAPE = \frac{1}{m}\sum_{j=1}^{m}\frac{|y_j - a_j|}{|y_j|}\cdot 100\%$$

Симметричный средний процент отклонения (Symmetric Mean Absolute Percentage Error):

$$SMAPE = \frac{1}{m}\sum_{j=1}^{m}\frac{|y_j - a_j|}{(y_j + a_j)/2}\cdot 100\%$$

Проблема SMAPE: пусть $y_j = 0$ и $a_j > y_j$, тогда ошибка будет равна 200%. Мы пытаемся в % сравнить целое число с нулём. Поэтому при минимизации SMAPE алгоритм прежде всего сосредотачивается на «угадывании нулей».

Меры точности в задаче регрессии

Сравнение двух алгоритмов.

Есть целый класс мер, которые основаны на сравнении ошибкок двух алгоритмов, например с некоторым бенчмарком. Пусть $\tilde{a}_j$ — метки, полученные бенчмарком.

Средняя относительная абсолютная ошибка (Mean Relative Absolute Error):

$$MRAE = \frac{1}{m}\sum_{j=1}^{m}\frac{|y_j - a_j|}{|y_j - \tilde{a}_j|}$$

или отношение суммарных ошибок

$$REL\_MAE = \frac{1}{m}\frac{\sum_{j=1}^{m}|y_j - a_j|}{\sum_{j=1}^{m}|y_j - \tilde{a}_j|}$$

Меры точности в задаче регрессии

Процент случаев, в которых алгоритм лучше бенчмарка (Percent Better):

$$PB(MAE) = \frac{1}{m}\sum_{j=1}^{m} \mathrm{I}(|y_j - a_j| < |y_j - \tilde{a}_j|$$

В качестве бенчмарка можно брать

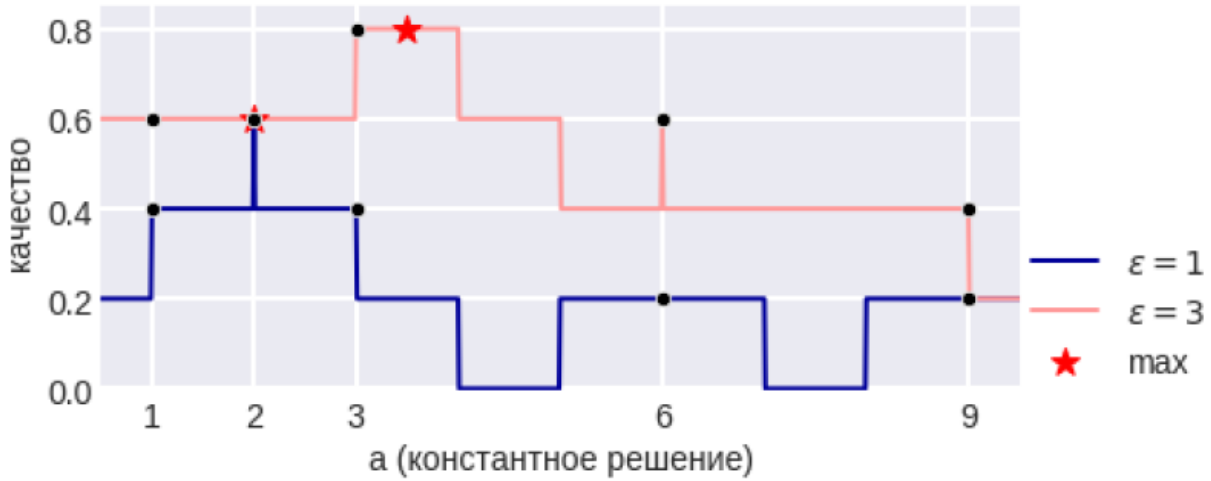
- простой константный алгоритм (часто используют нулевой, сравните MAPE и MRAE)
- предыдущую версию Вашего алгоритма (чтобы оценить, насколько улучшилось решения)
- естественный алгоритм (например,в задачах прогнозировании последовательности самый естественный и простой прогноз: значение последовательности в следующий момент времени такое же, как в предыдущий)

Меры точности в задаче регрессии

С точностью до порога

$$eB = \frac{1}{m}\sum_{i=1}^{m} \mathrm{I}(|y_i - a_i| < \varepsilon)$$

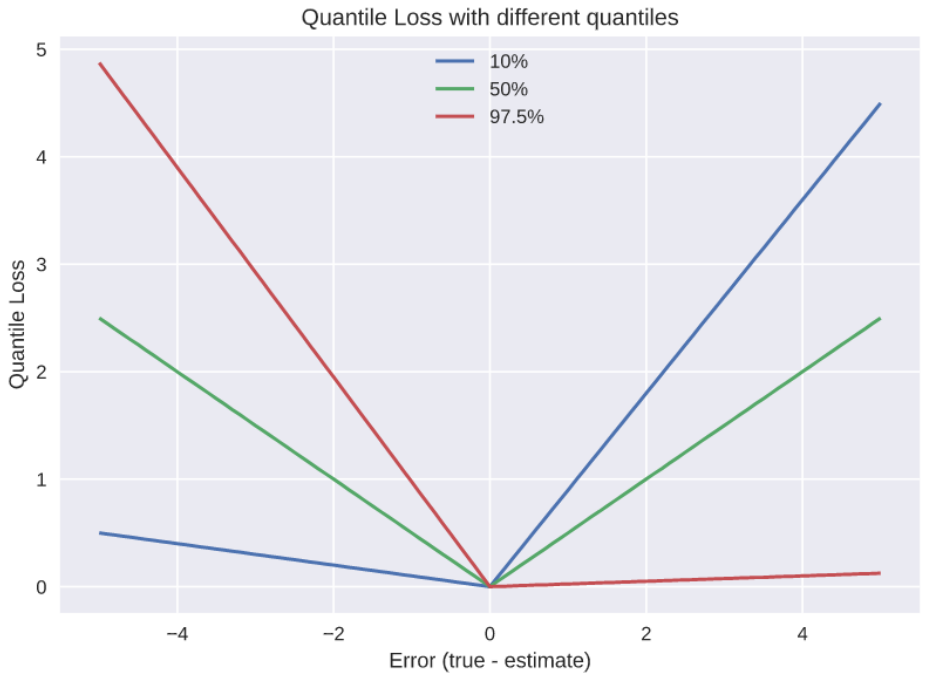
был в задаче Dunnhumby https://www.kaggle.com/c/dunnhumbychallenge



Несимметричные меры

На практике часто ошибки в разные стороны (завышение и занижение) неравнозначны. Например, занизить или завысить цену на товар. С точки зрения денежных потерь, эти ошибки имеют разную стоимость. **Квантильная мера (Quantile loss):**

$$Q_\alpha = \sum_{j=1}^{m} Q_\alpha(y_j, a_j), \quad Q_\alpha(y_j, a_j) = (y_j - a_j)(\alpha - \mathrm{I}(y_j - a_j)), \quad \alpha \in (0; 1)$$



Меры точности в задаче регрессии

Или даже так

$$\frac{1}{m}\sum_{j=1}^{m}\begin{cases} g(|y_i - a_i|), & y_i < a_i, \\ h(|y_i - a_i|), & y_i \geq a_i \end{cases}$$