

А. Прямая снова крутится

Рассмотрим линейный классификатор на плоскости $\mathbb{R}^2$, задаваемый прямой $x + Ay = 0$. Точка (x, y) относится к первому классу, если $x + Ay > 0$. В противном случае, если $x + Ay \leq 0$, точка относится ко второму классу.

Приведите пример таких **целых** чисел A , b и c , что для трёхчлена $P(x) = x^2 + bx + c$ точка $(2, P(2))$ относится к первому классу, а точки $(1, P(1))$ и $(3, P(3))$ — ко второму классу.

Решение.

Один из примеров: $A = -4$, $b = -4$, $c = 4$, то есть $P(x) = (x - 2)^2 = x^2 - 4x + 4$. Положим $L(x, y) = x - 4y$. Тогда

$$L(1, P(1)) = L(1, 1) = -3 \leq 0 \implies \text{точка } (1, P(1)) \text{ относится ко второму классу};$$

$$L(2, P(2)) = L(2, 0) = 2 > 0 \implies \text{точка } (2, P(2)) \text{ относится к первому классу};$$

$$L(3, P(3)) = L(3, 1) = -1 \leq 0 \implies \text{точка } (3, P(3)) \text{ относится ко второму классу}.$$

В. Генератор случайности

Генератор случайных чисел работает следующим образом. Если задать ему натуральное число $m \geq 6$, он выдаст последовательно 10 натуральных чисел. Каждое число выбирается им независимо и равновероятно из множества $\{1, 2, \dots, m\}$.

В силу системной ошибки, результат генератора выводится на экран следующим образом: числа 1, 2, 3, 4, 5 выводятся без изменений, а вместо любого другого числа выводится число 6.

При каком натуральном $m \geq 6$ вероятность получить на экране последовательность чисел

$$6, 2, 6, 6, 1, 6, 4, 6, 3, 5$$

максимальна?

Решение.

Ответ: $m = 10$.

Вероятность того, что на данном месте в последовательности будет число $k \in \{1, 2, 3, 4, 5\}$ составляет

$$\mathbb{P}(k) = \frac{1}{m}.$$

В связи с тем, что все остальные числа заменены на 6, та же вероятность для числа 6 составляет

$$\mathbb{P}(6) = \frac{m-5}{m}.$$

В изучаемой последовательности значение 6 встретилось 5 раз, и ещё 5 раз встретились числа 1, ..., 5. Поэтому вероятность получить эту последовательность равна

$$\left(\frac{1}{m}\right)^5 \left(\frac{m-5}{m}\right)^5 = \left(\frac{m-5}{m^2}\right)^5.$$

Максимальное значение этой вероятности достигается, когда наибольшее возможное значение принимает величина

$$\frac{m-5}{m^2} = \frac{1}{m} - \frac{5}{m^2}.$$

Положим $x = \frac{1}{m}$. Поскольку максимум квадратичной функции $f(x) = x - 5x^2$ достигается при $x = \frac{1}{10}$, искомое значение $m = 10$.

С. Градиентный спуск на листочке

Рассмотрим функцию $f(x, y) = x^{20} + y^{26}$. Ниже приведён алгоритм, который по начальной точке (a, b) строит последовательность точек (X_i, Y_i) , $i = 0, 1, 2, \dots$.

Ниже в коде `decrease_lr = False` для пункта (а), и `decrease_lr = True` для пункта (б).

```
X0, Y0 ← a, b
dx, dy ← 0, 1
lr ← 1
for i = 1, 2, 3, ... do
    dx, dy ← dy, -dx
    if decrease_lr and i · lr ≥ 2 then:
        lr ← lr / 2
    end if
    Xi, Yi ← Xi-1, Yi-1
    if f(Xi + dx · lr, Yi + dy · lr) < f(Xi, Yi) then
        Xi ← Xi + dx · lr
        Yi ← Yi + dy · lr
    end if
end for
```

Докажите, что, какая бы точка ни была начальной, справедливы следующие утверждения.

- (а) Найдётся такой номер N , что расстояние от точки (X_N, Y_N) до начала координат не превосходит 1.
- (б) Найдётся такой номер N_0 , что для любого номера $N \geq N_0$ расстояние от точки (X_N, Y_N) до начала координат не превосходит $\frac{1}{1000}$.

Решение.

Изучим очередной шаг алгоритма. Делается *попытка* изменить значение одной из координат на некоторую величину lr . Это изменение происходит в том и только в том случае, когда уменьшается модуль соответствующей координаты, поскольку

$$f(x + lr, y) < f(x, y) \iff |x + lr| < |x|,$$

$$f(x, y + lr) < f(x, y) \iff |y + lr| < |y|.$$

Пункт (а).

По координате x алгоритм делает попытки изменения $+1, -1, +1, -1, \dots$, по координате y , наоборот, $-1, +1, -1, +1, \dots$.

Рассмотрим две последовательные попытки изменения x -координаты.

- Если $|x| > \frac{1}{2}$, то ровно одна из двух попыток приводит к изменению координаты, при этом $|x|$ уменьшается ровно на 1.
- Если $|x| \leq \frac{1}{2}$, то ни одна из попыток к изменению координаты не приводит.

Таким образом, после конечного числа таких пар шагов алгоритма будет выполнено неравенство $|x| \leq \frac{1}{2}$, так как $|x|$ не может бесконечное количество раз уменьшаться на 1.

То же верно и для y -координаты, значит, для некоторого номера N будут выполнены неравенства

$$|X_N| \leq \frac{1}{2}, \quad |Y_N| \leq \frac{1}{2}.$$

Тогда и расстояние до начала координат в этого момент будет меньше, чем 1:

$$X_N^2 + Y_N^2 \leq \frac{1}{4} + \frac{1}{4} = \frac{1}{2} < 1.$$

В силу сказанного выше, это неравенство будет справедливо и для любого номера $M \geq N$, это соображение нам потребуется для решения другого пункта.

Пункт (b).

Разделим шаги алгоритма на блоки последовательных с одним и тем же значением величины lr . Рассмотрим m -й блок, состоящий из шагов с номерами $i = 2^m + 1, \dots, 2^{m+1}$, где $lr = \frac{1}{2^m}$. Далее изучаем блоки для $m \geq 3$.

Тогда аналогично пункту а) внутри каждого блока каждая вторая попытка изменения x -координаты уменьшает значение $|x|$, пока выполняется неравенство $|x| > \frac{lr}{2} = \frac{1}{2^{m+1}}$.

Значит, для m -го блока шагов справедливо хотя бы одно из следующих утверждений.

1. После выполнения всех шагов m -го блока $|x| \leq \frac{lr}{2}$.
2. В результате каждого четвёртого шага алгоритма (то есть каждой второй попытки изменения x -координаты) величина $|x|$ уменьшалась на lr . Таким образом, за этот блок шагов она суммарно уменьшилась на $2^{m-2} \cdot lr = \frac{1}{4}$.

Заметим, что случай 2 мог возникнуть лишь для конечного числа блоков, так как $|x|$ не может уменьшаться бесконечное количество раз на $\frac{1}{4}$.

Следовательно, существует такое число M_x , что для блоков с номерам, большими M_x , справедливо первое утверждение.

Итого при $N > N_x = \max(2^{M_x+1}, 10)$ будут выполнены неравенства

$$|X_N| \leq \frac{1}{2^{N_x+1}} \leq \frac{1}{2^{11}} < \frac{1}{2000}.$$

Аналогично и для координаты y найдётся такой номер N_y , что при $N > N_y$ выполнено неравенство $|Y_N| < \frac{1}{2000}$.

Тогда при $N > \max(N_x, N_y)$ имеют место неравенства $|X_N| < \frac{1}{2000}$ и $|Y_N| < \frac{1}{2000}$. Значит,

$$\sqrt{|X_N|^2 + |Y_N|^2} \leq \sqrt{|X_N|^2 + 2 \cdot |X_N||Y_N| + |Y|^2} = |X_N| + |Y_N| < \frac{1}{2000} + \frac{1}{2000} = \frac{1}{1000}.$$

Таким образом, номер $N_0 = \max(N_x, N_y)$ удовлетворяет условиям задачи.

Замечание. Из приведённого решения следует, что последовательность точек (X_i, Y_i) сходится к точке минимума функции $f(x, y)$ — началу координат.

Д. Лазер

Пол прямоугольной камеры с зеркальными стенами имеет форму клетчатого прямоугольника 1000×3000 . В **узлах сетки** расположены датчики температуры (всего $1001 \cdot 3001$ датчиков). Изначально все датчики показывают температуру 0°C .

В **центрах некоторых клеток** расположено по одному устройству, испускающему лазерные лучи. Каждое устройство испускает лазерный луч по диагонали клетки, в центре которой оно находится (то есть в одном из четырех возможных направлений, в сторону одного из узлов соответствующей клетки). При этом то же устройство поглощает лучи, приходящие из противоположного узла. На лучи трёх других направлений устройство не влияет.

Всего установлено 300 устройств, которые излучают по одному лучу. Каждый луч отражается зеркально от стенок (попадая в угол, луч отражается в противоположном направлении), пока не поглотится каким-то из устройств.

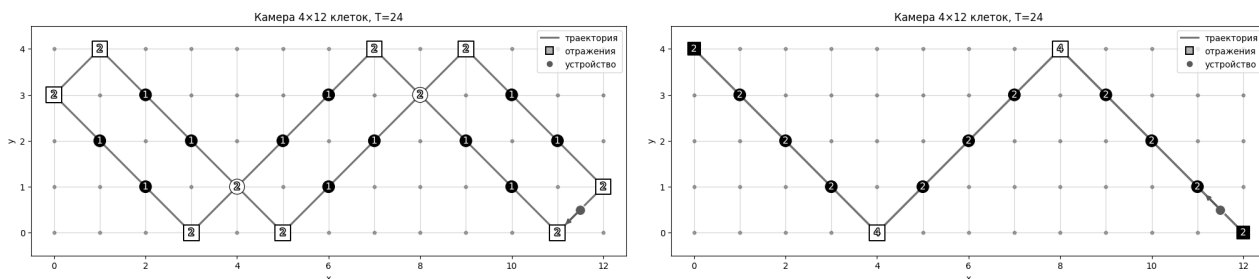
Лазерный луч, «посещая» очередной датчик температуры, увеличивает его показание на 1°C . Если же в точке, где располагается датчик, луч отражается, он увеличивает показание соответствующего датчика сразу на 2°C . Назовём *тепловой картой* матрицу A размера 1001×3001 , элементы которой равны итоговым показаниям соответствующих датчиков температуры.

Для проверки корректности показаний датчиков, полученные значения анализируют следующим образом. Выбирается квадратная матрица K размера $s \times s$, называемая *ядром* размера s , после чего вычисляется *свёртка* матрицы A с ядром K , то есть матрица $B = A \star K$ размера $(1002 - s) \times (3002 - s)$, где

$$B[i][j] = \sum_{u=0}^{s-1} \sum_{v=0}^{s-1} A[i+u][j+v] K[u][v], \quad 0 \leq i \leq 1001 - s, \quad 0 \leq j \leq 3001 - s.$$

Назовём ядро K *определяющим*, если для любой тепловой карты A все элементы матрицы B равны нулю, при этом не все элементы ядра K равны 0. При каком наименьшем значении s существует определяющее ядро размера s ?

Ниже на рисунке вы можете видеть как запущенный луч изменит температуру в датчиках до момента поглощения.



Решение.

Ответ: $s = 3$.

Рассмотрим ядро

$$K = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}.$$

Покажем, что оно подходит. Заметим, что траектории всех лучей разбиваются на циклы потому что каждый датчик принимает и испускает ровно один луч (отметим, что один цикл может быть объединением траекторий сразу нескольких лучей). Заметим, что достаточно проверить условие для одной циклической траектории (в силу линейности свёртки). Будем мыслить такую траекторию как набор диагональных отрезков между последовательными отражениями, где каждый отрезок увеличивает значения каждого датчика, который на нём расположен, на 1°C . Легко видеть, что тепловая карта траектории равна сумме тепловых карт этих отрезков.

Значит, достаточно проверить, что в результате свёртки с ядром K любой матрицы, в которой единицы расположены на одном диагональном ряду (остальные элементы — нули), получается нулевая матрица. Это следует из того, что сумма элементов матрицы K на любом диагональном ряду равна нулю.

Остаётся доказать, что меньшие значения s не подходят. Достаточно доказать это для $s = 2$. Предположим обратное, что есть определяющее ядро

$$K = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

Рассмотрим циклическую траекторию, аналогичную приведённой в условии на втором рисунке (проходящую через два противоположных угла). Легко видеть, что если расставить в центры клеток, которые она пересекает, 300 устройств, можно добиться, чтобы получилась **только** эта траектория (нужно направить все лучи из устройств «вдоль» неё).

Пусть она проходит через левый нижний угол $(0, 0)$. Тогда после свёртки с ядром K мы получим $B[1][0] = 2a$ и $B[0][1] = 2d$. Итого $a = d = 0$. Рассуждая аналогично для траектории, проходящей через два других угла, мы получим, что $b = c = 0$, противоречие.

Е. Новая выборка

Модель по вещественному числу x предсказывает вещественное число методом *линейной регрессии*, то есть по формуле $f(x) = ax + b$, где $a, b \in \mathbb{R}$ — параметры модели f .

Дана *обучающая выборка* $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, где $x_1, \dots, x_n, y_1, \dots, y_n \in \mathbb{R}$. Модель f обучается на этих данных: параметры a и b выбираются *методом наименьших квадратов*, то есть так, чтобы значение выражения

$$\sum_{i=1}^n (y_i - f(x_i))^2$$

было минимальным.

Определим *коэффициент детерминации* R_0^2 , посчитанный на этой выборке, следующим образом:

$$R_0^2 = 1 - \frac{\sum_{i=1}^n (y_i - f(x_i))^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \quad \text{где} \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i.$$

Построим новую выборку, добавив к исходной n объектов $(x_i, f(x_i)), i = 1, 2, \dots, n$. Пусть на получившейся выборке из $2n$ объектов обучается модель $g(x) = cx + d$ также методом наименьших квадратов. Обозначим через R_1^2 коэффициент детерминации для модели g , посчитанный на выборке из всех $2n$ объектов.

Предполагается, что параметры обеих моделей удалось подобрать методом наименьших квадратов единственным образом.

Выразите R_1^2 через R_0^2 .

Решение.

Ответ: $R_1^2 = \frac{2R_0^2}{1 + R_0^2}$.

Положим $\hat{y}_i = f(x_i)$, (предсказание); $SSE_0 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ (сумма квадратов ошибок), $SST_0 = \sum_{i=1}^n (y_i - \bar{y})^2$ (общая сумма квадратов). Тогда $R_0^2 = 1 - \frac{SSE_0}{SST_0}$.

Аналогично определим величины SSE_1 и SST_1 для второй модели.

Лемма 1. Модель, обученная на объединённой выборке, совпадает с исходной, и потому $SSE_1 = SSE_0$.

Доказательство. Для модели f на добавленных точках $(x_i, \hat{y}_i)$ ошибки $(\hat{y}_i - f(x_i))^2$ равны нулю, а для исходных точек сумма квадратов ошибок — наименьшая возможная. Значит, минимум на объединённой выборке достигается для тех же значений параметров (тем самым, модели совпадают). $\square$

Лемма 2. Имеют место равенства $\sum_{i=1}^n (y_i - \hat{y}_i) = 0$ и $\sum_{i=1}^n x_i (y_i - \hat{y}_i) = 0$.

Доказательство. Точка (a, b) выбирается методом наименьших квадратов, то есть совпадает с наименьшим значением функции

$$F(u, v) = \sum_{i=1}^n (y_i - ux_i - v)^2.$$

Значит, если зафиксировать $u = a$ у получившейся квадратичной функции от переменной v наименьшее значение достигается в точке b . Это означает, что v -координата вершины соответствующей параболы равна b , то есть

$$-b = \frac{1}{n} \left(\sum_{i=1}^n (ax_i - y_i) \right) \implies 0 = \sum_{i=1}^n (ax_i + b) - \sum_{i=1}^n y_i = \sum_{i=1}^n (\hat{y}_i - y_i).$$

Рассуждая аналогично для $v = b$, получаем второе заявленное соотношение:

$$-a \cdot \sum_{i=1}^n x_i^2 = \sum_{i=1}^n (b - y_i)x_i \implies 0 = \sum_{i=1}^n x_i(ax_i + b - y_i) = \sum_{i=1}^n x_i(\hat{y}_i - y_i).$$

□

Лемма 3. *Имеет место равенство $SST_1 = 2SST_0 - SSE_0$.*

Доказательство. Согласно лемме 2, справедливо равенство $\frac{1}{n} \sum_{i=1}^n \hat{y}_i = \bar{y}$. Значит,

$$SST_1 = \sum_{i=1}^n (y_i - \bar{y})^2 + \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 = SST_0 + \sum_{i=1}^n (\hat{y}_i - \bar{y})^2.$$

Кроме того, $y_i - \bar{y} = (y_i - \hat{y}_i) + (\hat{y}_i - \bar{y})$. Возведём все такие равенства в квадрат, после чего просуммируем. Мы получим, что

$$SST_0 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 + 2 \sum_{i=1}^n (y_i - \hat{y}_i)(\hat{y}_i - \bar{y}).$$

Последний член равен нулю. Действительно, воспользуемся леммой 2 следующим образом:

$$\sum_{i=1}^n (y_i - \hat{y}_i)(\hat{y}_i - \bar{y}) = \sum_{i=1}^n (y_i - \hat{y}_i)\hat{y}_i = a \sum_{i=1}^n x_i(y_i - \hat{y}_i) + b \sum_{i=1}^n (y_i - \hat{y}_i) = 0.$$

Тогда $SST_0 = SSE_0 + \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$, что и даёт требуемое равенство. □

Собирая вместе результаты лемм 1–3, имеем:

$$R_1^2 = 1 - \frac{SSE_1}{SST_1} = 1 - \frac{SSE_0}{2SST_0 - SSE_0} = \frac{2(SST_0 - SSE_0)}{2SST_0 - SSE_0}.$$

Разделив числитель и знаменатель на SST_0 , мы получаем заявленный ответ:

$$R_1^2 = \frac{2(1 - \frac{SSE_0}{SST_0})}{2 - \frac{SSE_0}{SST_0}} = \frac{2R_0^2}{1 + R_0^2}.$$

Комментарий. Лемму 2 можно доказать по-другому, воспользовавшись тем, что частные производных функции $F(u, v)$ в точке (a, b) равны нулю.

Г. Вымышленная ситуация

Артур и Таня готовят вычислительные мощности к практическому туру для 1000 участников финала всероссийской олимпиады школьников по ИИ. Изначально они хотели вернуть всю инфраструктуру в облаке и выдавать видеокарты (GPU) из общего набора, но, чтобы исключить сетевые задержки, было решено собрать каждому участнику персональный кластер из 8 устройств.

На складе партнёров олимпиады есть по 8000 устройств каждого из трех типов.

- Тип 1 (зеленые) — CUDA, быстрые, но горячие.
- Тип 2 (красные) — поддерживают открытые стандарты, подходят для обучения больших языковых моделей (LLM), но требуют сложной настройки драйверов.
- Тип 3 (желтые) — экспериментальные ускорители (TPU).

По условиям поставки устройства типа 1 можно получать только партиями по 7 штук; устройств каждого типа должно быть не меньше, чем 2026; всего со склада следует взять ровно 8000 устройств. Если в соответствии с этими условиями можно привезти со склада y_1 устройств первого типа, y_2 — второго и y_3 — третьего, будем называть тройку целых чисел (y_1, y_2, y_3) *допустимой*, то есть:

$$y_1 + y_2 + y_3 = 8000, \quad y_t \geq 2026 \ (t = 1, 2, 3), \quad y_1 \div 7.$$

Каждый участник имеет персональный шифр $i = 1, 2, \dots, 1000$. Устройства, поставленные со склада, распределяются между участниками. Пусть участник i получает персональный кластер $S_i = (x_{i,1}, x_{i,2}, x_{i,3})$, в котором $x_{i,1}, x_{i,2}, x_{i,3}$ — количество устройств первого, второго и третьего типа соответственно. Числа $x_{i,1}, x_{i,2}, x_{i,3}$ — целые неотрицательные, причём $x_{i,1} + x_{i,2} + x_{i,3} = 8$.

Оказалось, что кто-то из участников предпочитает «зеленых», кто-то фанат открытых стандартов и выбирает «красных», а кто-то любит необычных «жёлтых». Для каждого участника заданы *предпочтения* устройств $u_{i,1}, u_{i,2}, u_{i,3}$, указанные во входном файле `gpu.csv`.

Файл содержит заголовок и 1000 строк с 3 столбцами с данными: в i -й строке записаны три целых числа $u_{i,1}, u_{i,2}, u_{i,3}$ (именно в таком порядке), соответствующие предпочтениям участника i .

Полезность кластера $S = (x_1, x_2, x_3)$ для участника i задаётся формулой

$$V_i(S) = u_{i,1}x_1 + u_{i,2}x_2 + u_{i,3}x_3.$$

Распределение устройств называется *честным*, если неравенство $V_i(S_i) \geq V_i(S_j)$ справедливо для любых двух участников i, j .

- Приведите пример допустимой тройки значений (y_1, y_2, y_3) , при которой честного распределения не существует, и докажите это.
- Существует ли допустимая тройка (y_1, y_2, y_3) , при которой существует честное распределение?

Решение.

Будем называть величины $u_{i,j}$ *полезностями* устройств типа j для участника i .

Лемма 1. Если умножить значения полезностей произвольного участника i на одно и то же положительное число, любое честное распределение останется честным, а любое нечестное — нечестным.

Доказательство. Обе части неравенств, в которых фигурирует полезность V_i , мы умножим на одну и ту же положительную константу, тем самым выполнение системы новых и изначальных неравенств равносильно. $\square$

Наблюдение. По данным файла `gru.csv`, школьники делятся на две группы.

- группа A : 511 школьников, набор полезностей которых пропорционален

$$u^A = (2053, 2063, 2070);$$

- группа B : 489 школьников, набор полезностей которых пропорционален

$$u^B = (2081, 2027, 2003).$$

В силу леммы 1, можно считать, что всем участникам группы A соответствует набор полезностей u^A , а всем участникам группы B — набор полезностей u^B .

Лемма 2. В любом честном распределении все школьники группы B получают один и тот же набор устройств.

Доказательство. Возьмём двух участников группы B . Пусть первый получил b_i устройств типа i , второй — b'_i устройств типа i . Поскольку предпочтения у этих участников совпадают, полезности кластеров (b_1, b_2, b_3) и (b'_1, b'_2, b'_3) для них также совпадают и равны между собой. То есть

$$2081b_1 + 2027b_2 + 2003b_3 = 2087b'_1 + 2027b'_2 + 2003b'_3.$$

Поскольку в обоих кластерах по 8 устройств, вычитая из обеих частей $2003 \cdot 8$, получаем

$$\begin{aligned} (2081 - 2003)b_1 + (2027 - 2003)b_2 &= (2081 - 2003)b'_1 + (2027 - 2003)b'_2 \Rightarrow \\ 78(b_1 - b'_1) + 24(b_2 - b'_2) &= 0 \Rightarrow 13(b_1 - b'_1) + 4(b_2 - b'_2) = 0 \end{aligned}$$

Значит, число $b_2 - b'_2$ кратно 13. Однако, поскольку $0 \leq b_2, b'_2 \leq 8$, это возможно лишь в случае $b_2 = b'_2$, а тогда и $b_1 = b'_1$, $b_3 = b'_3$, что завершает доказательство леммы. $\square$

Лемма 3. В любом честном распределении все школьники группы A получают один и тот же набор устройств.

Доказательство. Рассуждение, в целом, аналогично предыдущей лемме. Вновь, предполагая противная, имеем два кластера (a_1, a_2, a_3) и (a'_1, a'_2, a'_3) одинаковой полезности для участника группы A . Теперь, поскольку $2053 - 2070 = -17$ и $2063 - 2070 = -7$, имеем: $-17(a_1 - a'_1) - 7(a_2 - a'_2) = 0$. Следовательно, $a_2 - a'_2$ делится на 17, поэтому $a_2 = a'_2$, а тогда и $a_1 = a'_1$, $a_3 = a'_3$, лемма доказана. $\square$

Итого все 511 школьников группы A получают один и тот же кластер $a = (a_1, a_2, a_3)$, а все 489 школьников группы B — один и тот же кластер $b = (b_1, b_2, b_3)$.

Посчитаем количество устройств каждого типа:

$$y_t = 511a_t + 489b_t \quad (t = 1, 2, 3).$$

Пункт (а).

Рассмотрим набор $(y_1, y_2, y_3) = (2100, 2100, 3800)$. Легко видеть, что он допустим: 2100 делится на 7, все y_i не меньше, чем 2026 и $2100 + 2100 + 3800 = 8000$. Докажем, что для этого набора честного распределения не существует.

Предположим противное. В силу сказанного выше, для некоторых кластеров $a = (a_1, a_2, a_3)$ и $b = (b_1, b_2, b_3)$ имеет место равенство $2100 = y_1 = 511a_1 + 489b_1$. Но числа вида $2100 - 511n$ при $n = 0, 1, 2, 3$ не делятся на 489, а при $n \geq 4$ — отрицательны, противоречие.

Пункт (b).

Ответ: такой набор существует.

Рассмотрим следующее распределение: каждому школьнику группы A соберём кластер $(0, 4, 4)$, а каждому школьнику группы B — кластер $(7, 1, 0)$.

Вычислим количество устройств каждого типа:

$$y_1 = 511 \cdot 0 + 489 \cdot 7 = 3423, \quad y_2 = 511 \cdot 4 + 489 \cdot 1 = 2044 + 489 = 2533, \quad y_3 = 511 \cdot 4 + 489 \cdot 0 = 2044.$$

Отметим, что y_1 делится на 7, и количества устройств каждого типа не меньше, чем 2026. Тем самым, такое распределение соответствует допустимой тройке

$$(y_1, y_2, y_3) = (3423, 2533, 2044).$$

Проверим, что получилось честное распределение. Для любого школьника i из группы A :

$$V_i(a) = 2053 \cdot 0 + 2063 \cdot 4 + 2070 \cdot 4 = 8252 + 8280 = 16532,$$

$$V_i(b) = 2053 \cdot 7 + 2063 \cdot 1 + 2070 \cdot 0 = 14371 + 2063 = 16434.$$

Значит, $V_i(a) > V_i(b)$. Прделаем то же вычисление для школьника участника j из группы B :

$$V_j(a) = 2081 \cdot 0 + 2027 \cdot 4 + 2003 \cdot 4 = 8108 + 8012 = 16120,$$

$$V_j(b) = 2081 \cdot 7 + 2027 \cdot 1 + 2003 \cdot 0 = 14567 + 2027 = 16594.$$

Следовательно, $V_j(b) > V_j(a)$. Итого каждый участник группы B предпочитает свой кластер кластеру группы A и наоборот. Поскольку внутри каждой из групп участники получают одинаковые кластеры, распределение честное. Таким образом, подходит набор

$$(y_1, y_2, y_3) = (3423, 2533, 2044).$$

Замечание. Можно доказать, что указанный набор — единственный, подходящий под условия пункта (b). Тем самым, любой другой допустимый набор подойдёт в качестве примера для пункта (a).

А. Прямая снова крутится

А. Верный ответ без проверки — 45 баллов.

В. Генератор случайности

А. Найдено $\mathbb{P}(6) = (m - 5)/m - 5$ баллов

В. Найдена вероятность последовательности из условия при фиксированном m — 15 баллов.

С. Заявлено, что указанная в предыдущем критерии вероятность возрастает при целых m от 6 до 10 и убывает далее — 15 баллов

Д. Вероятность посчитана с ошибкой в константу раз (и другие не влияющие на ход решения неточности) — снимается не менее 5 баллов.

М. Доказано, что число 10 является экстремумом. Используется, но не доказано, что это именно максимум — снимается не менее 5 баллов.

- Все приведенные продвижения и штрафы суммируются.

С. Градиентный спуск на листочке

Общая часть.

С1 (5 баллов) Сформулировано чередование направлений спуска с периодом 4.

С2 (5 баллов) Установлено убывание $|X|$ и $|Y|$.

Пункт (а).

А. Верное доказательство пункта (а) — 20 баллов.

МА. Неверно указана область остановки или используется без доказательства существование такой области — снимается 10 баллов.

Пункт (b).

В. Верное доказательство пункта (b) — 20 баллов.

В1. Введена структура блоков, с указанием связи их размера и общим значением lr — 5 баллов.

В2. Доказано, что вклад каждого блока в изменение параметров $|X|$ и $|Y|$ составляет не меньше некоторой константы, если все шаги этого блока приводили к изменению соответствующих координат.

МВ. Ошибки в изложении оставшейся части доказательства — снимается не менее 5 баллов.

- Продвижения и штрафы по трем приведенным частям задачи суммируются.

Д. Лазер

Часть А (пример). Часть В (оценка).

А1. Приведена правильная матрица K — 10 баллов.

А2. Показано, что матрица обнуляет диагональные ряд — 5 баллов.

А3. Завершение обоснования, почему матрица K является определяющим ядром — 10 баллов.

В1. Получено линейное уравнение на элементы ядра — 5 баллов.

М. Ошибки в построении «подходящего луча» (он рассматривается только локально, ставится меньшее число устройств, чем требует условие и т.д.) — снимается не менее 5 баллов.

- Все приведенные продвижения и штрафы суммируются.
-

Е. Новая выборка

Общеизвестные свойства линейной регрессии принимаются без доказательства **при наличии верной формулировки**.

А. Верный ответ — 10 баллов.

В. Доказано, что $f = g$ — 10 баллов.

- Если только сформулировано, что $f = g$, выставляется 5 баллов.
- Если доказательство опирается на свойства линейной регрессии, должно быть явно сформулировано свойство добавляемых данных, которое позволяет сделать вывод о равенстве коэффициентов моделей.
- Если это используется и даже не формулируется, баллы по критерию B не начисляются.
- За вывод о равенстве числителей в выражениях коэффициента детерминации дополнительные баллы не начисляются.

С1. Установлены зависимости между $y_i, x_i, f(x_i)$ (лемма 2) — 10 баллов.

С2. Получены выражения для a и b (формула коэффициентов линейной регрессии) с доказательством равенства средних — 10 баллов.

- Должны быть установлены **обе** зависимости (в том виде, как это указано С1 или С2) и явно сформулировано равенство средних значений старой и новой выборки. За получения любого одного из соотношений начисляется 5 баллов за часть C .

М. Лемма 3 (или аналогичное равенство) используется в решении без доказательства — снимается 15 баллов.

- За алгебраические преобразования, не приводящие к зависимости R_1^2 **только** от R_0^2 баллы не начисляются.

Ф. Вымышленная ситуация

А. Верный ответ в пункте (а) — 0 баллов.

В. Лемма 1 (о домножении полезностей на константу) вместе с наблюдением о разделении участников на две группы — 10 баллов.

В1. Указано разделение агентов делится на 2 группы по предпочтениям — 5 баллов.

В2. Лемма 1 только сформулирована — 5 баллов.

С. Формулировка и доказательство леммы 2 и 3 (о равенстве кластеров внутри каждой из групп участников) — 15 баллов.

С1. Формулировка леммы 2 и 3 — 5 баллов.

Д. Вывод пункта (а) из лемм 2 и 3 — 5 баллов.

Е. Ответ в пункте (б) — 10 баллов.

Ф. Проверка ответа в пункте (б) — 10 баллов.

А. Что посмотреть?

1.1 Резюме

Этот ноутбук читает данные из файлов в той же папке, решает 3 подзадачи и сохраняет ответы в CSV:

- **A1** favorite_genre → answer_a1.csv
- **A2** actor_match → answer_a2.csv
- **A4** next_in_session → answer_a3.csv

1.2 Файлы, которые должен видеть ноутбук

Вход (в одной папке с ноутбуком): - queries_A.csv — запросы (по 5 кандидатов c1..c5) - items_A.csv — метаданные фильмов (genre, duration) - events_A.csv — события пользователей (open/finish/like) - item_meta_A.json — метаданные фильмов (списки актёров actors) - sessions_A.json — сессии (внутри sessions[*].path)

Выход (создаётся рядом с ноутбуком):

Файл	Относительный путь	Назначение
answer_a1.csv	./answer_a1.csv	ответы для favorite_genre
answer_a2.csv	./answer_a2.csv	ответы для actor_match
answer_a4.csv	./answer_a3.csv	ответы для next_in_session

1.3 Универсальный паттерн решения

Одинаковый каркас используется во всех трёх подзадачах:

- 1) melt переводит кандидатов c1..c5 из wide в long (по строке на кандидата).
- 2) merge(..., how='left') присоединяет признаки/скор, **не теряя кандидатов**.
- 3) Если есть списки (актёры, пары переходов), используем explode. Важно: **пустые списки дают NaN строку** (это удобно, чтобы оставить кандидата и получить вклад 0).
- 4) Выбор победителя делается сортировкой sort_values + drop_duplicates('query_id') (первый в отсортированном порядке).
- 5) Сохранение — to_csv(index=False).

Ссылки на документацию pandas (официально):

- pandas.melt:
<https://pandas.pydata.org/docs/reference/api/pandas.melt.html>
- DataFrame.explode:
<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.explode.html>
- pandas.merge:
<https://pandas.pydata.org/docs/reference/api/pandas.merge.html>

- `DataFrame.sort_values`:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html
- `DataFrame.drop_duplicates`:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html
- `DataFrame.to_csv`:
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html

```
[1]: import pandas as pd
import matplotlib.pyplot as plt
```

1.3.1 Быстрая проверка загрузки данных

Ниже читаем все файлы один раз, смотрим размеры таблиц и распределение `query_type`.

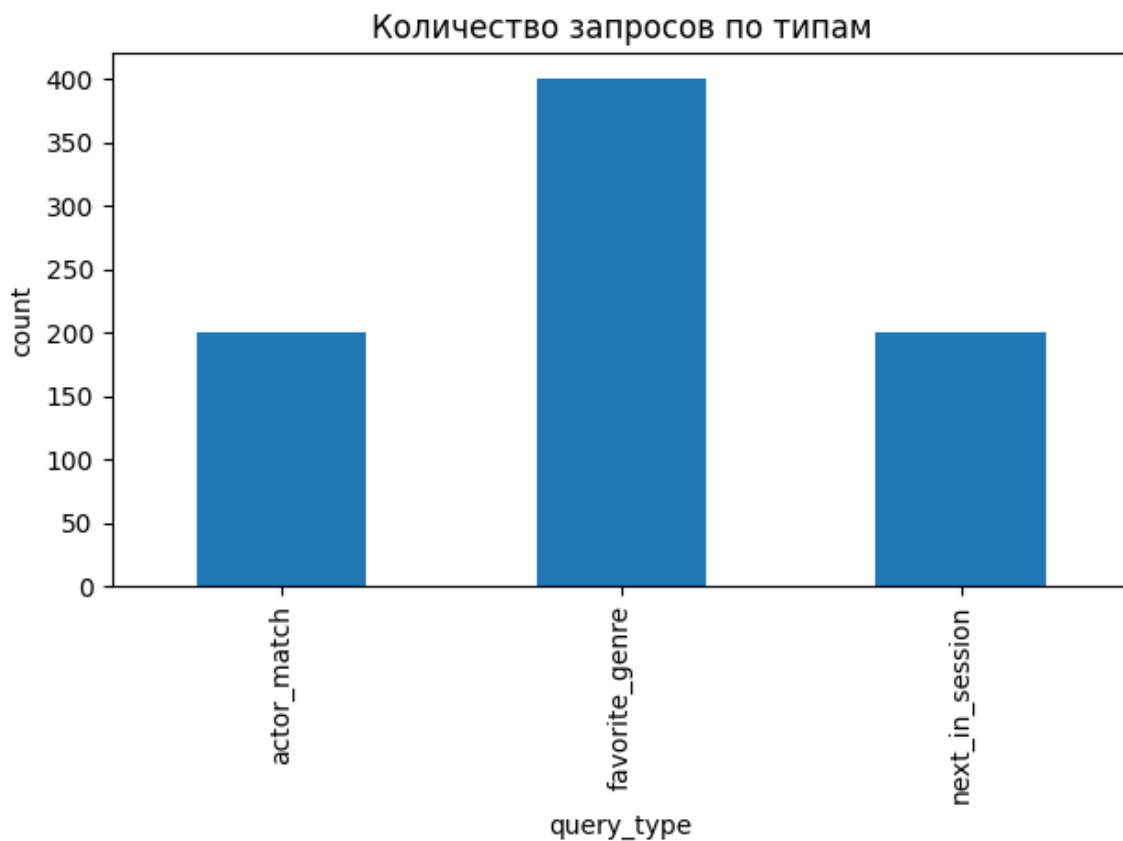
```
[2]: Q = pd.read_csv('queries_A.csv')
I = pd.read_csv('items_A.csv')
E = pd.read_csv('events_A.csv')
M = pd.read_json('item_meta_A.json')[['item_id', 'actors']]
S = pd.read_json('sessions_A.json')

print('Q', Q.shape, 'I', I.shape, 'E', E.shape, 'M', M.shape, 'S', S.shape)
Q.query_type.value_counts()
```

Q (800, 8) I (260, 3) E (5334, 3) M (260, 2) S (180, 2)

```
[2]: query_type
favorite_genre      400
actor_match         200
next_in_session     200
Name: count, dtype: int64
```

```
[3]: vc = Q.query_type.value_counts().sort_index()
ax = vc.plot(kind='bar', title='Количество запросов по типам')
ax.set_xlabel('query_type')
ax.set_ylabel('count')
plt.tight_layout()
plt.show()
```



1.3.2 A1. favorite_genre

```
[4]: q = Q[Q.query_type ==
      'favorite_genre'][['query_id', 'user_id', 'c1', 'c2', 'c3', 'c4', 'c5']]

e = E[['user_id', 'item_id', 'event_type']].copy()
e['w'] = e.event_type.map({'open': 1, 'finish': 2, 'like': 3})
e = e.merge(I[['item_id', 'genre']], on='item_id', how='left')
g = e.groupby(['user_id', 'genre'], as_index=False)['w'].sum()

c = q.melt(['query_id', 'user_id'], ['c1', 'c2', 'c3', 'c4', 'c5'],
           value_name='item_id')[['query_id', 'user_id', 'item_id']]
c = c.merge(I[['item_id', 'genre', 'duration']], on='item_id', how='left')
c = c.merge(g, on=['user_id', 'genre'], how='left')
c['w'] = c.w.fillna(0)

c = c.sort_values(['query_id', 'w', 'duration', 'item_id'], ascending=[1,0,1,1])
a1 = c.drop_duplicates('query_id')[['query_id', 'item_id']]

a1.to_csv('answer_a1.csv', index=False)
a1.head()
```

```
[4]:      query_id  item_id
400           1    1199
```

1201	2	1065
1602	3	1211
403	4	1086
4	5	1163

1.3.3 A2. actor_match

```
[5]: q = Q[Q.query_type ==
      'actor_match'][['query_id', 'user_id', 'c1', 'c2', 'c3', 'c4', 'c5']]
e = E[E.event_type.isin(['finish', 'like'])][['user_id', 'item_id']]

m = M[['item_id', 'actors']].copy()
m['actors'] = m.actors.apply(lambda x: x if isinstance(x, list) else [])

ua = e.merge(m, on='item_id', how='left')
ua = ua.explode('actors').dropna(subset=['actors'])
ua = ua.drop_duplicates(['user_id', 'actors'])
ua['hit'] = 1
ua = ua[['user_id', 'actors', 'hit']]

c = q.melt(['query_id', 'user_id'], ['c1', 'c2', 'c3', 'c4', 'c5'],
           value_name='item_id')[['query_id', 'user_id', 'item_id']]
c = c.merge(m, on='item_id', how='left')
c = c.explode('actors')
c = c.drop_duplicates(['query_id', 'item_id', 'actors'])

c = c.merge(ua, on=['user_id', 'actors'], how='left')
c['hit'] = c.hit.fillna(0)

s = c.groupby(['query_id', 'item_id'], as_index=False)['hit'].sum()
s = s.sort_values(['query_id', 'hit', 'item_id'], ascending=[1,0,1])
a2 = s.drop_duplicates('query_id')[['query_id', 'item_id']]

a2.to_csv('answer_a2.csv', index=False)
a2.head()
```

```
[5]:   query_id  item_id
4         401     1252
9         402     1255
14        403     1227
19        404     1216
24        405     1232
```

1.3.4 A3. next_in_session

```
[6]: q = Q[Q.query_type ==
      'next_in_session'][['query_id', 'user_id', 'c1', 'c2', 'c3', 'c4', 'c5']]

fin = E[E.event_type == 'finish'][['user_id', 'item_id']].drop_duplicates()
fin = fin.rename(columns={'item_id': 'from_item'})
```

```
t = S.explode('sessions').dropna(subset=['sessions'])
t['path'] = t.sessions.apply(lambda x: x.get('path', []))
t = t[['user_id', 'path']]
t['pair'] = t.path.apply(lambda p: list(zip(p[:-1], p[1:])))
t = t.explode('pair').dropna(subset=['pair'])
t['from_item'] = t.pair.str[0]
t['item_id'] = t.pair.str[1]
t['hit'] = 1
tr = t.groupby(['user_id', 'from_item', 'item_id'], as_index=False)['hit'].
    .sum()

c = q.melt(['query_id', 'user_id'], ['c1', 'c2', 'c3', 'c4', 'c5'],
value_name='item_id')[['query_id', 'user_id', 'item_id']]
c = c.merge(fin, on='user_id', how='left')
c = c.merge(tr, on=['user_id', 'from_item', 'item_id'], how='left')
c['hit'] = c.hit.fillna(0)

s = c.groupby(['query_id', 'item_id'], as_index=False)['hit'].sum()
s = s.sort_values(['query_id', 'hit', 'item_id'], ascending=[1,0,1])
a3 = s.drop_duplicates('query_id')[['query_id', 'item_id']]

a3.to_csv('answer_a3.csv', index=False)
a3.head()
```

```
[6]:
```

	query_id	item_id
2	601	1028
5	602	1071
11	603	1049
17	604	1101
20	605	1004

В. Сейсмоактивный остров

```
[ ]: import numpy as np
import pandas as pd

data = np.load('data_B.npy')
N = data.shape[0]
m = np.nanmean(data, axis=1)

# Попарная корреляция
sim = np.zeros((N, N))
for i in range(N):
    for j in range(i, N):
        mask = ~np.isnan(m[i]) & ~np.isnan(m[j])
        a, b = m[i, mask] - m[i, mask].mean(), m[j, mask] - m[j, mask].mean()
        d = np.sqrt((a**2).sum() * (b**2).sum())
        if d > 0:
            sim[i, j] = sim[j, i] = (a * b).sum() / d

# Каждому сэмплу - кластер с максимальной средней корреляцией
# Инициализация: жадно
labels = np.full(N, -1)
reps = []
for i in range(N):
    best_sim, best_c = -1, -1
    for c, r in enumerate(reps):
        if sim[i, r] > best_sim:
            best_sim = sim[i, r]
            best_c = c
    if best_sim > 0.35:
        labels[i] = best_c
    else:
        labels[i] = len(reps)
        reps.append(i)

# Уточнение: переназначаем по средней корреляции с кластером
for _ in range(10):
    clusters = [np.where(labels == c)[0] for c in range(max(labels) + 1)]
    changed = False
    for i in range(N):
        best_score, best_c = -2, labels[i]
        for c, members in enumerate(clusters):
            others = members[members != i]
            score = sim[i, others].mean() if len(others) > 0 else sim[i,
members[0]]
            if score > best_score:
                best_score = score
                best_c = c
        if best_c != labels[i]:
            labels[i] = best_c
```

```
        changed = True
    if not changed:
        break
    # Обновляем кластеры
    clusters = [np.where(labels == c)[0] for c in range(max(labels) + 1)]

y = pd.read_csv('y.csv')['cluster'].values
from sklearn.metrics import adjusted_rand_score
print(f"Clusters: {len(set(labels))}, ARI = {adjusted_rand_score(y, labels):.6f}")

pd.DataFrame({'ID': range(N), 'target': labels}).to_csv('submission_B.csv',
index=False)
```

С. Ошибка новичка

```
[1]: import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from catboost import CatBoostClassifier
import warnings
from IPython.display import display
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Фиксируем сид для воспроизводимости всех шагов
SEED = 42
np.random.seed(SEED)

# Загрузка доступных файлов
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
solution = pd.read_csv('ytest.csv')
with open('map.json', 'r') as f:
    map_data = json.load(f)

display(train.head())
display(test.head())
```

	longitude	latitude	target	habitat_quality_score	biodiversity_index	\
0	-68.624132	-32.868160	0	38.746018	0.869221	
1	-51.964445	1.978971	0	17.456343	3.414865	
2	-50.585866	-5.300425	0	43.883046	1.930266	
3	-64.666741	-32.100087	0	29.736058	1.724559	
4	-72.512342	-5.718456	1	66.547123	9.257867	

	canopy_density	soil_moisture_level	forest_age_years	\
0	0.584679	6.031727	97.911830	
1	0.327984	4.694215	169.739698	
2	0.750885	7.864357	214.602112	
3	0.322355	4.448480	275.153342	
4	0.428812	3.822563	179.540444	

	tree_density_per_hectare	annual_rainfall_mm	habitat_fragmentation_index	\
0	396.493164	1293.066100	0.507221	
1	350.579890	2353.814818	0.247614	
2	237.872776	1147.090078	0.574802	
3	458.725205	1252.207036	0.648601	
4	346.391230	604.782917	0.732562	

	predator_pressure_score	vegetation_complexity	forest_type	\
0	1.040142	3.0	tropical_rainforest	

1	7.522185	9.0	dry_forest
2	3.255146	4.0	dry_forest
3	6.404152	9.0	tropical_rainforest
4	4.880406	2.0	dry_forest

	climate_zone	soil_type	disturbance_level	conservation_status	\
0	montane	peat	low	unprotected	
1	temperate	loam	minimal	unprotected	
2	subtropical	loam	moderate	unprotected	
3	temperate	loam	low	buffer_zone	
4	tropical	clay	low	unprotected	

	dominant_tree_species	water_source_type	temperature_variability	\
0	palm	stream	4.037666	
1	pine	none	5.882945	
2	palm	seasonal	3.366330	
3	mahogany	stream	8.984897	
4	bamboo	seasonal	3.966467	

	elevation_range	human_activity_index	canopy_height_avg	\
0	730.654928	0.075256	6.710190	
1	316.780429	3.614124	30.770752	
2	474.802210	2.251489	5.000000	
3	187.292371	3.063365	16.284970	
4	191.639490	3.888626	36.796504	

	seasonal_variation_score
0	6.874974
1	6.384739
2	5.690438
3	6.328448
4	5.780874

	longitude	latitude	habitat_quality_score	biodiversity_index	\
0	-72.102869	19.626859	45.511580	3.743753	
1	-50.441167	-8.396785	34.751298	4.712511	
2	-58.372430	0.312577	73.162533	2.579567	
3	-76.362920	-14.279388	86.832262	4.097908	
4	-67.162159	6.450007	63.631579	4.895632	

	canopy_density	soil_moisture_level	forest_age_years	\
0	0.648016	3.260149	172.921350	
1	0.226638	5.191137	357.568190	
2	0.747227	7.533181	300.220431	
3	0.602197	6.429846	325.612904	
4	1.000000	4.502920	308.714905	

	tree_density_per_hectare	annual_rainfall_mm	habitat_fragmentation_index	\
0	226.157888	1177.791799	0.558007	
1	429.731562	1922.536202	0.332591	

2	144.840477	1844.822680	0.654825
3	323.532762	1571.667498	0.383862
4	693.209698	1688.758358	0.344897

	predator_pressure_score	vegetation_complexity	forest_type \
0	7.795448	3.0	secondary_forest
1	5.712367	7.0	dry_forest
2	2.073700	1.0	cloud_forest
3	0.798348	5.0	dry_forest
4	1.378641	1.0	tropical_rainforest

	climate_zone	soil_type	disturbance_level	conservation_status \
0	montane	laterite	high	unprotected
1	temperate	sandy	moderate	protected
2	subtropical	loam	high	protected
3	tropical	alluvial	minimal	buffer_zone
4	subtropical	sandy	minimal	protected

	dominant_tree_species	water_source_type	temperature_variability \
0	kapok	stream	5.205539
1	palm	seasonal	8.634001
2	pine	none	5.382953
3	pine	stream	8.282785
4	pine	stream	10.690082

	elevation_range	human_activity_index	canopy_height_avg \
0	318.967595	3.626041	23.686628
1	184.900374	1.830241	28.697284
2	536.571256	0.074063	26.366329
3	649.036995	0.000000	34.803212
4	388.522453	4.500977	45.697464

	seasonal_variation_score	ID
0	6.492307	0
1	4.313709	1
2	3.480140	2
3	3.417066	3
4	6.823274	4

1.4 Функция для визуализации

Создадим вспомогательную функцию для визуализации точек на карте с контуром береговой линии (map.json).

```
[2]: def plot_map(points, coastline, color_by=None, figsize=(5, 5)):
    """
    Визуализирует точки на карте с контуром береговой линии.

    Параметры:
    -----
    points : pd.DataFrame
```

```
DataFrame с точками, должен содержать колонки 'longitude' и 'latitude'
coastline : list
    Список координат береговой линии [[lon1, lat1], [lon2, lat2], ...]
color_by : str, optional
    Название колонки для раскраски точек. Если None, все точки одного цвета
figsize : tuple, optional
    Размер фигуры (ширина, высота)

Пример использования:
-----
# Простая визуализация без раскраски
plot_map(train, coastline)

# С раскраской по таргету
plot_map(train, coastline, color_by='target')

# Изменение размера
plot_map(train, coastline, color_by='target', figsize=(16, 10))
"""
fig, ax = plt.subplots(figsize=figsize)

# Рисуем береговую линию
if coastline:
    coastline_array = np.array(coastline)

    # Рисуем все точки побережья как плотный scatter с квадратными
маркерами
    ax.scatter(coastline_array[:, 0], coastline_array[:, 1],
               s=1.5, marker='s', color='#CCCCCC', alpha=0.7,
label='Coastline')

# Рисуем точки
if color_by is None:
    # Без раскраски - все точки одного цвета
    ax.scatter(points['longitude'], points['latitude'],
               s=50, alpha=0.6, edgecolors='black', linewidths=0.5,
               label='Points')
else:
    # С раскраской по указанной колонке
    if points[color_by].dtype in ['object', 'category']:
        # Категориальная переменная
        unique_vals = points[color_by].unique()
        colors = plt.cm.tab10(np.linspace(0, 1, len(unique_vals)))

        for i, val in enumerate(unique_vals):
            mask = points[color_by] == val
            ax.scatter(points[mask]['longitude'],
points[mask]['latitude'],
```

```
        s=50, alpha=0.6, edgecolors='black', linewidths=0.5,
        label=f'{color_by}={val}', color=colors[i])
    else:
        # Числовая переменная
        scatter = ax.scatter(points['longitude'], points['latitude'],
                              c=points[color_by], cmap='RdYlGn',
                              s=50, alpha=0.6, edgecolors='black',
                              linewidths=0.5)
        plt.colorbar(scatter, ax=ax, label=color_by)

    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')
    ax.set_title(f'Карта точек (n={len(points)})')
    ax.legend()
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

print("Функция plot_map() готова к использованию")
```

Функция plot_map() готова к использованию

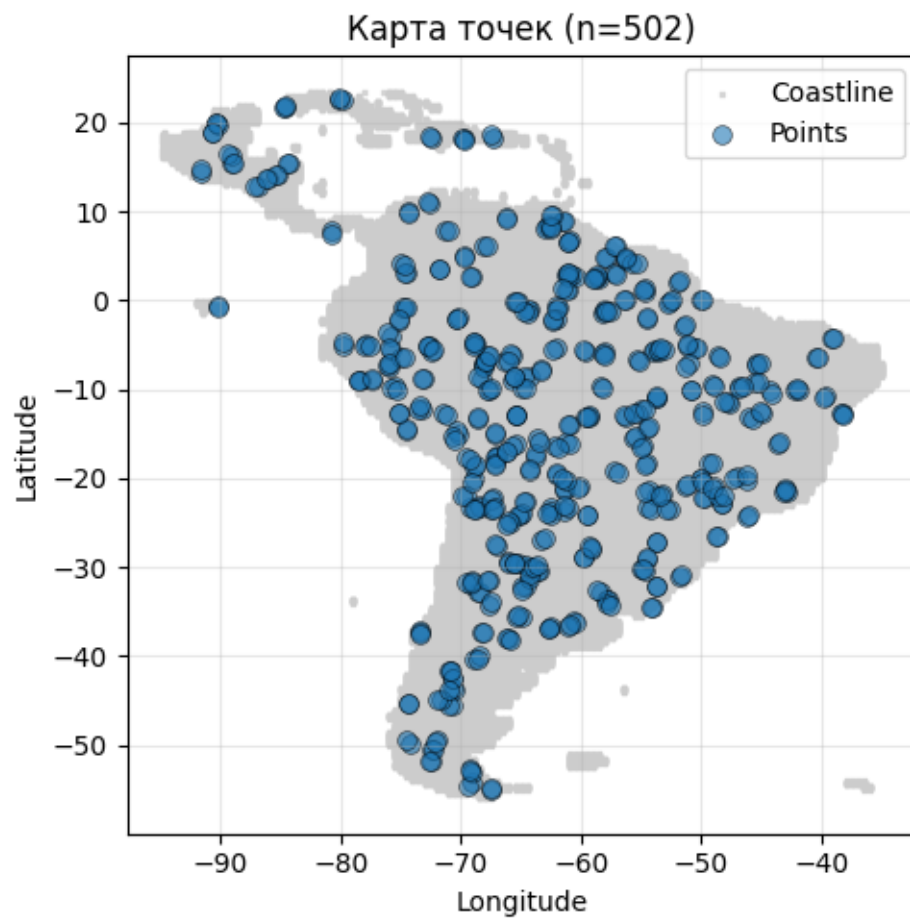
1.5 Демонстрация функции визуализации

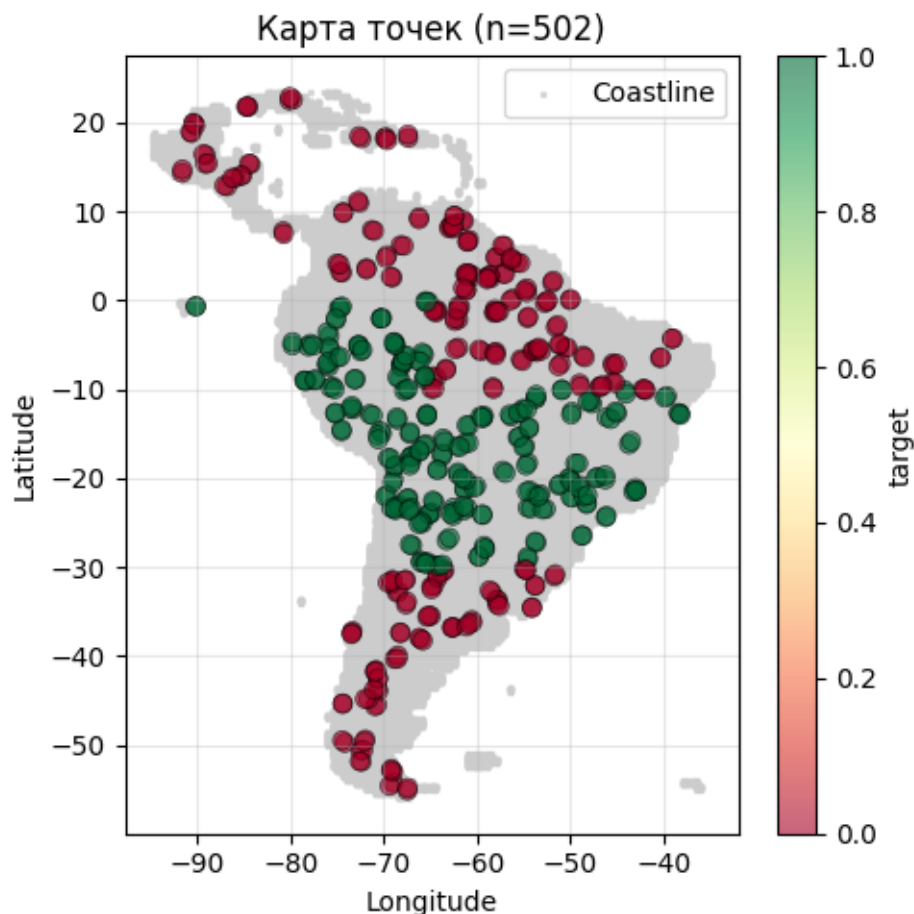
Попробуем функцию на загруженных данных.

```
[3]: # Получаем coastline из map.json
coastline = map_data.get('coastline', [])

# Пример 1: Train без раскраски
plot_map(train, coastline)

# Пример 2: Train с раскраской по маркету
plot_map(train, coastline, color_by='target')
```





1.6 Шаг 1: Первый базовый подход

По условию задачи у нас есть `train.csv` (прямые наблюдения) и `test.csv`, где часть точек добавлена по спутниковым данным.

Сначала строим честный baseline на всех признаках и смотрим **F1 score** на разметке из `ytest.csv`. Это нужно как контрольная точка, чтобы понять, насколько модель склонна переобучаться в исходной постановке.

```
[4]: # Функция для оценки качества
def evaluate(y_pred, solution):
    """Оценка F1-score: overall, public, private"""
    from sklearn.metrics import f1_score

    merged = solution.copy()
    merged['pred'] = np.asarray(y_pred).reshape(-1)

    # Считаем метрику на уже округленных предсказаниях (0/1)
    merged['pred_label'] = np.clip(np rint(merged['pred']), 0, 1).astype(int)

    public_mask = merged['Usage'] == 'public'
    private_mask = merged['Usage'] == 'private'
```

```
overall_f1 = f1_score(merged['target'], merged['pred_label'])
public_f1 = f1_score(merged.loc[public_mask, 'target'], merged.
↪loc[public_mask, 'pred_label'])
private_f1 = f1_score(merged.loc[private_mask, 'target'], merged.
↪loc[private_mask, 'pred_label'])

print(f"F1 overall: {overall_f1:.6f}")
print(f"F1 public: {public_f1:.6f}")
print(f"F1 private: {private_f1:.6f}")

return public_f1, private_f1, overall_f1

# Выделяем признаки (все кроме служебных)
feature_cols_numeric = [col for col in train.columns
                        if col not in ['ID', 'target', 'cluster', 'is_train']
                        and train[col].dtype in ['int64', 'float64']]

categorical_cols = train.select_dtypes(include=['object']).columns.tolist()
categorical_cols = [col for col in categorical_cols if col not in ['ID']]

all_features = feature_cols_numeric + categorical_cols

X_train = train[all_features]
y_train = train['target']
X_test = test[all_features]

from sklearn.model_selection import train_test_split
X_train_tr, X_train_val, y_train_tr, y_train_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=SEED, stratify=y_train
)

model = CatBoostClassifier(
    iterations=500,
    learning_rate=0.1,
    depth=12,
    random_seed=SEED,
    verbose=0,
)

model.fit(X_train_tr, y_train_tr, cat_features=categorical_cols)
y_pred_test_baseline = np.asarray(model.predict(X_test)).reshape(-1).
↪astype(int)

baseline_public, baseline_private, baseline_overall =
evaluate(y_pred_test_baseline, solution)
```

F1 overall: 0.646440

F1 public: 0.645707

F1 private: 0.647208

```
[5]: from sklearn.metrics import f1_score
f1_score(solution.target.values, (solution.target.values * 0 + 0.5 >= 0.5).
      ↪astype(int))
```

[5]: 0.5270917803169922

1.7 Шаг 2: Анализ важности признаков

Связываем результат baseline с условием задачи. Если в топе важности оказываются координаты (longitude, latitude), модель может использовать географическую «подсказку» вместо устойчивых экологических закономерностей. Это и есть первая типичная ошибка новичка: неявная утечка/спуриация через признаки, которые слишком прямо кодируют разделение.

```
[6]: # Feature importance
importance_df = pd.DataFrame({
    'feature': all_features,
    'importance': model.get_feature_importance()
}).sort_values('importance', ascending=False)

print("\nTop 10 признаков:")
print(importance_df.head(10).to_string(index=False))

print("Это может означать, что модель переобучается на координаты.")
print("\nВНИМАНИЕ: longitude и latitude в топе важности!")
```

Top 10 признаков:

	feature	importance
	latitude	25.842854
	habitat_quality_score	18.788004
	biodiversity_index	12.708726
	longitude	9.644228
	forest_type	6.298411
	disturbance_level	5.888199
	climate_zone	2.864203
	conservation_status	1.980591
	seasonal_variation_score	1.594778
	human_activity_index	1.516413

Это может означать, что модель переобучается на координаты.

ВНИМАНИЕ: longitude и latitude в топе важности!

1.8 Шаг 3: Визуализация координат

Посмотрим, как распределены классы в пространстве координат.

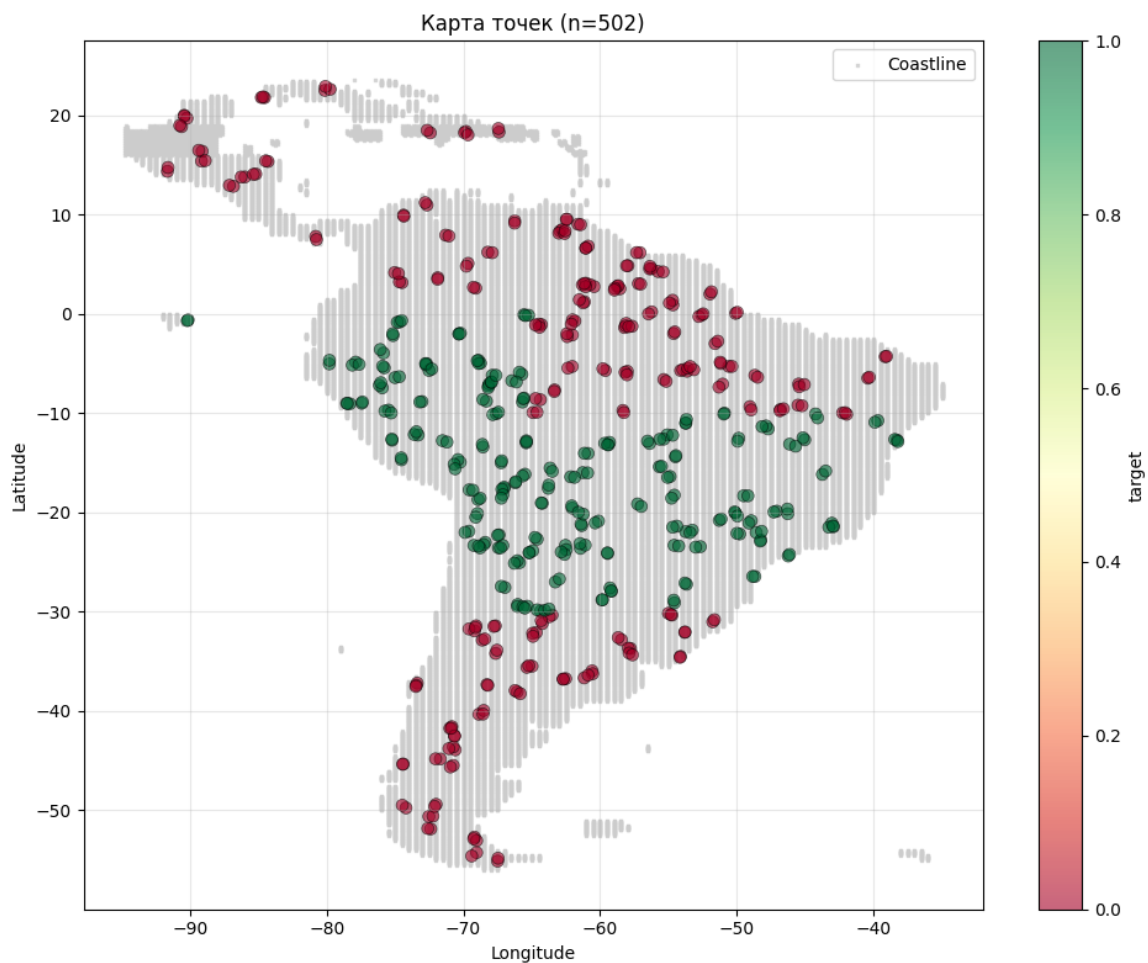
```
[7]: # Используем нашу функцию для визуализации
print("Train данные с раскраской по таргету:")
plot_map(train, coastline, color_by='target', figsize=(10, 8))
```

```
# Анализ распределения классов
print("\n" + "="*70)
print("АНАЛИЗ КООРДИНАТ")
print("="*70)

# Статистика по долготе
print("\nРаспределение классов по долготе:")
for target_val in [0, 1]:
    subset = train[train['target'] == target_val]
    print(f"\nКласс {target_val}:")
    print(f"  Longitude: min={subset['longitude'].min():.2f},
max={subset['longitude'].max():.2f}, mean={subset['longitude'].mean():.2f}")
    print(f"  Latitude:  min={subset['latitude'].min():.2f},
max={subset['latitude'].max():.2f}, mean={subset['latitude'].mean():.2f}")

print("\n" + "="*70)
print("ВЫВОД: Классы четко разделены по координатам!")
print("  Модель запомнила географическую границу между классами.")
print("  В реальном тесте эта закономерность не работает.")
print("  РЕШЕНИЕ: Нужно убрать longitude и latitude из признаков.")
print("="*70)
```

Train данные с раскраской по таргету:



```
=====
АНАЛИЗ КООРДИНАТ
=====
```

Распределение классов по долготе:

Класс 0:

```
Longitude: min=-91.65, max=-39.05, mean=-64.11
Latitude:  min=-55.12, max=22.89, mean=-10.67
```

Класс 1:

```
Longitude: min=-90.20, max=-38.22, mean=-62.39
Latitude:  min=-29.95, max=-0.10, mean=-15.78
```

```
=====
ВЫВОД: Классы четко разделены по координатам!
```

Модель запомнила географическую границу между классами.

В реальном тесте эта закономерность не работает.

РЕШЕНИЕ: Нужно убрать longitude и latitude из признаков.

```
=====
```

1.9 Шаг 4: Убираем координаты и переобучаем модель

Исправляем первую ошибку: убираем longitude и latitude, чтобы модель опиралась на содержательные признаки среды. Снова считаем F1 score и сравниваем с baseline. Если качество растёт, значит координаты действительно мешали обобщению.

```
[8]: # Убираем координаты
features_no_coords = [f for f in all_features if f not in ['longitude',
'latitude']]
categorical_cols_filtered = [c for c in categorical_cols if c in
features_no_coords]

X_train_no_coords = train[features_no_coords]
X_test_no_coords = test[features_no_coords]

X_train_tr_no_coords, X_train_val_no_coords, y_train_tr_no_coords,
y_train_val_no_coords = train_test_split(
    X_train_no_coords, y_train, test_size=0.2, random_state=SEED,
    stratify=y_train
)

model.fit(X_train_tr_no_coords, y_train_tr_no_coords,
cat_features=categorical_cols_filtered)
y_pred_test_no_coords = np.asarray(model.predict(X_test_no_coords)).
    ↪ reshape(-1).astype(int)
```

```
nocoords_public, nocoords_private, nocoords_overall =  
evaluate(y_pred_test_no_coords, solution)
```

F1 overall: 0.783972

F1 public: 0.798155

F1 private: 0.769591

1.10 Шаг 5: Детектор train vs test (поиск сдвига распределения)

Проверяем вторую ошибку новичка из условия: игнорирование того, что **train** и часть **test** собраны разными способами. Если модель легко отличает **train** от **test**, это признак domain shift (распределения различаются). Тогда у части test-точек стандартная модель может быть слишком уверенной и ошибаться системно.

```
[9]: from catboost import cv, Pool  
  
# Создаем датасет для классификации train vs test  
X_combined = pd.concat([X_train_no_coords, X_test_no_coords], axis=0)  
y_combined = np.concatenate([np.ones(len(X_train_no_coords)), np.  
    ↪ zeros(len(X_test_no_coords))])  
  
model_ood = CatBoostClassifier(  
    iterations=50,  
    learning_rate=0.1,  
    depth=6,  
    random_seed=SEED,  
    verbose=0  
)  
  
cv_data = Pool(data=X_combined, label=y_combined,  
cat_features=categorical_cols_filtered)  
_ = cv(  
    pool=cv_data,  
    params={  
        'iterations': 50,  
        'learning_rate': 0.1,  
        'depth': 6,  
        'loss_function': 'Logloss',  
        'eval_metric': 'AUC',  
        'random_seed': SEED,  
        'verbose': False  
    },  
    fold_count=5,  
    shuffle=True,  
    stratified=True,  
    partition_random_seed=SEED  
)  
  
model_ood.fit(X_combined, y_combined, cat_features=categorical_cols_filtered)  
test_similarity_scores = model_ood.predict(X_test_no_coords,  
prediction_type='Probability')[:, 1]
```

```
plt.figure(figsize=(10, 5))
plt.hist(test_similarity_scores, bins=100, alpha=0.7, edgecolor='black')
plt.xlabel('Вероятность принадлежности к train')
plt.ylabel('Количество объектов')
plt.title('Train/Test Classifier: Распределение вероятностей на test')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Training on fold [0/5]

bestTest = 0.6328465347
bestIteration = 9

Training on fold [1/5]

bestTest = 0.6250990099
bestIteration = 4

Training on fold [2/5]

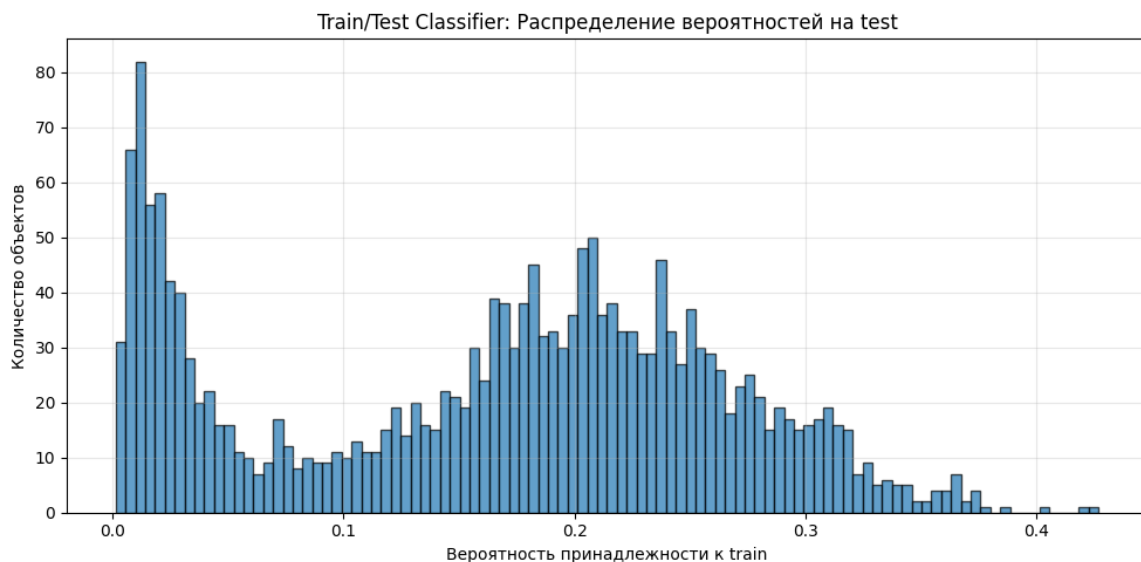
bestTest = 0.6888
bestIteration = 6

Training on fold [3/5]

bestTest = 0.6909022556
bestIteration = 11

Training on fold [4/5]

bestTest = 0.6967669173
bestIteration = 16



1.11 Шаг 6: OOD-корректировка предсказаний

Используем train/test-детектор как индикатор OOD-точек (похожести тестовой точки на train). Для точек с очень низкой похожестью понижаем вероятность класса 1 и снова считаем только F1 score.

Так мы адресуем вторую ошибку: учитываем сдвиг распределения и делаем предсказания устойчивее на «спутниковой» части теста.

```
[10]: print("\nКорректируем предсказания для OOD объектов...")
      print("="*70)

      # Порог OOD: точки с низкой похожестью на train считаем сдвинутыми по
      # распределению
      ood_threshold = 0.07

      y_pred_corrected = y_pred_test_no_coords.copy()
      ood_mask = test_similarity_scores < ood_threshold
      y_pred_corrected[ood_mask] = 0

      # Выводим F1-score для каждого подхода: overall/public/private
      _, _, _ = evaluate(y_pred_test_baseline, solution)
      _, _, _ = evaluate(y_pred_test_no_coords, solution)
      _, _, corrected_overall = evaluate(y_pred_corrected, solution)
```

Корректируем предсказания для OOD объектов...

```
=====
F1 overall: 0.646440
F1 public: 0.645707
F1 private: 0.647208
F1 overall: 0.783972
F1 public: 0.798155
F1 private: 0.769591
F1 overall: 0.918549
F1 public: 0.918340
F1 private: 0.918768
```

```
[11]: # Сохраняем submission с лучшими доступными предсказаниями
      final_pred = y_pred_corrected

      submission = pd.DataFrame({
          'ID': test['ID'].values,
          'target': (final_pred > 0.5) * 1
      })
      submission.to_csv('author_submission.csv', index=False)
```

Д. Наследие Человечества

```
[1]: # Шаг 0. Импорты
import csv
import json
from pathlib import Path

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

# Фиксируем генератор случайных чисел для воспроизводимости
rng = np.random.default_rng(42)
```

1.12 Разбор решения по шагам

Решения идут по порядку и усложняются шаг за шагом: 1. Решение 0: IOU/Jaccard по словам. 2. Решение 1: unigram TF-IDF. 3. Решение 2: word+char TF-IDF. 4. Решение 3: решение 2 + greedy one-to-one. 5. Решение 4: решение 3 + logreg-reranker. 6. Решение 5: решение 4 + greedy one-to-one.

1.12.1 Шаг 1. Загрузка train/test/ytest

Читаем train.json и test.json, восстанавливаем train-пары, а true_test и разбиение Public/Private строим по ytest.csv через соответствия left_id ↔ right_id и колонку Usage.

```
[2]: # Шаг 1. Загрузка train/test/ytest (ещё компактнее)
base_dir = Path("/Users/aguschin/Git/uni/vsosh/zakl/nlp")
train_json_path, test_json_path, ytest_csv_path = (
    base_dir / "train.json",
    base_dir / "test.json",
    base_dir / "ytest.csv",
)

def split_poem(text):
    lines = [line.strip() for line in text.split("\n") if line.strip()]
    cut = min(8, len(lines) - 1)
    if cut < 1:
        return None, None
    left, right = "\n".join(lines[:cut]).strip(), "\n".join(lines[cut:]).strip()
    return (left, right) if left and right else (None, None)

# train: из полных текстов -> пары половинок -> shuffle правых
train_pairs = [split_poem(text) for text in json.loads(train_json_path.read_text(encoding="utf-8"))]
train_pairs = [(left, right) for left, right in train_pairs if left and right]
```

```
left_parts = [left for left, _ in train_pairs]
right_parts = [right for _, right in train_pairs]
train_idx = np.arange(len(left_parts), dtype=int)
perm_train = np.random.default_rng(123).permutation(len(left_parts))
left_train = left_parts
right_train_shuffled = [right_parts[i] for i in perm_train]
true_train = np.empty(len(left_parts), dtype=int)
true_train[perm_train] = np.arange(len(left_parts))
n_train = len(left_train)

# test: зомовые left/right
test_data = json.loads(test_json_path.read_text(encoding="utf-8"))
left_items, right_items = test_data["left"], test_data["right"]
left_test = [item["left_text"] for item in left_items]
left_ids = [item["left_id"] for item in left_items]
right_test_shuffled = [item["right_text"] for item in right_items]
right_ids = [item["right_id"] for item in right_items]
n_test = len(left_test)

# ytest: left_id -> right_id u Usage
with open(ytest_csv_path, "r", encoding="utf-8") as f:
    ytest_rows = [row for row in csv.DictReader(f)]

left_to_right = {}
left_to_usage = {}
for row in ytest_rows:
    left_id = (row.get("left_id") or "").strip()
    right_id = (row.get("right_id") or "").strip()
    usage = (row.get("Usage") or "Private").strip().title()
    if not left_id or not right_id:
        continue
    if usage not in {"Public", "Private"}:
        usage = "Private"
    left_to_right[left_id] = right_id
    left_to_usage[left_id] = usage

right_id_to_pos = {rid: j for j, rid in enumerate(right_ids)}
true_test = np.array([right_id_to_pos[left_to_right[lid]] for lid in
left_ids], dtype=int)
usage_test = np.array([left_to_usage.get(lid, "Private") for lid in
left_ids], dtype=object)
public_mask = usage_test == "Public"
private_mask = usage_test == "Private"

# проверки и служебные переменные
assert all(lid in left_to_right for lid in left_ids), "Не все left_id из test
есть в ytest.csv"
assert all(left_to_right[lid] in right_id_to_pos for lid in left_ids),
"Некоторые right_id из ytest.csv отсутствуют в test.json"
vowels = set("aeёиоуыяюя")
```

```
test_idx = np.arange(n_test, dtype=int) + 10_000_000
overlap = len(set(train_idx.tolist()) & set(test_idx.tolist()))
assert overlap == 0, "train/test overlap detected"

print(f"Источник train: {train_json_path}")
print(f"Источник test : {test_json_path}")
print(f"Источник gt    : {ytest_csv_path}")
print(f"Всего пар: Train: {n_train} | Test: {n_test}")
print(f"Usage split: Public={int(public_mask.sum())} | Private={int(private_mask.sum())}")
print(f"Честность split: overlap(train,test)={overlap}\n")
```

```
Источник train: /Users/aguschin/Git/uni/vsosh/zakl/nlp/train.json
Источник test  : /Users/aguschin/Git/uni/vsosh/zakl/nlp/test.json
Источник gt    : /Users/aguschin/Git/uni/vsosh/zakl/nlp/ytest.csv
Всего пар: Train: 2000 | Test: 640
Usage split: Public=320 | Private=320
Честность split: overlap(train,test)=0
```

1.12.2 Решение 0 — IOU/Jaccard по словам

Базовый скор: для каждой пары половинок считаем пересечение слов и делим на объединение. Оценка/выбор пары: top1 через argmax (greedy matching не используется).

```
[3]: # Шаг 2. Решение 0: IOU/Jaccard через split()

# Решение 0: базовый скор по пересечению слов

def _masked_acc(pred_idx: np.ndarray, true_idx: np.ndarray, mask: np.
    ndarray):
    return float((pred_idx[mask] == true_idx[mask]).mean()) if mask.any()
    else float("nan")

# Функция: считает единый score через argmax (Overall/Public/Private)
def evaluate_similarity_argmax(sim: np.ndarray, true_idx: np.ndarray):
    pred = sim.argmax(axis=1)
    score = float((pred == true_idx).mean())
    score_public = _masked_acc(pred, true_idx, public_mask)
    score_private = _masked_acc(pred, true_idx, private_mask)
    return {
        "score": score,
        "score_public": score_public,
        "score_private": score_private,
    }

# Функция: превращает каждый текст в множество токенов
def _token_sets(texts):
    return [set(text.split()) for text in texts]

# Функция: строит матрицу Jaccard/IOU для всех пар left-right
```

```
def _jaccard_matrix(left_sets, right_sets):
    n_left = len(left_sets)
    n_right = len(right_sets)
    sim = np.zeros((n_left, n_right), dtype=np.float32)
    for i, lset in enumerate(left_sets):
        for j, rset in enumerate(right_sets):
            union = lset | rset
            if union:
                sim[i, j] = len(lset & rset) / len(union)
    return sim

word_train_l = _token_sets(left_train)
word_train_r = _token_sets(right_train_shuffled)
word_test_l = _token_sets(left_test)
word_test_r = _token_sets(right_test_shuffled)

sim0_test = _jaccard_matrix(word_test_l, word_test_r)
metrics0 = evaluate_similarity_argmax(sim0_test, true_test)
print(
    f"0_iou_argmax: "
    f"score={metrics0['score']:.4f} (pub={metrics0['score_public']:.4f},
    priv={metrics0['score_private']:.4f})"
)
```

0_iou_argmax: score=0.1734 (pub=0.1594, priv=0.1875)

1.12.3 Решение 1 — unigram TF-IDF

Улучшаем решение 0: заменяем простое пересечение слов на TF-IDF-представление слов и косинусную близость.

```
[4]: # Шаг 2. Решение 1: unigram TF-IDF

# Решение 1: улучшаем решение 0 с помощью TF-IDF признаков
vec1 = TfidfVectorizer(ngram_range=(1, 1), min_df=5, max_df=0.95,
    sublinear_tf=False)
vec1.fit(left_train + right_train_shuffled)
sim1_test = cosine_similarity(vec1.transform(left_test), vec1.
    ↪transform(right_test_shuffled))
metrics1 = evaluate_similarity_argmax(sim1_test, true_test)
print(
    f"1_unigram_tfidf: "
    f"score={metrics1['score']:.4f} (pub={metrics1['score_public']:.4f},
    priv={metrics1['score_private']:.4f})"
)
```

1_unigram_tfidf: score=0.2734 (pub=0.2562, priv=0.2906)

1.12.4 Решение 2 — word+char TF-IDF

Улучшаем решение 1: смешиваем similarity из word TF-IDF и char TF-IDF, затем считаем argmax-метрики.

[5]: *# Шаг 3. Решение 2: word+char TF-IDF*

```
# Функция: строит blended similarity из word и char TF-IDF
def _build_tfidf_blend(left_train, right_train, left_eval, right_eval, cfg,
alpha):
    vec_word = TfidfVectorizer(analyzer="word", **cfg["word"])
    vec_char = TfidfVectorizer(analyzer="char_wb", **cfg["char"])
    vec_word.fit(left_train + right_train)
    vec_char.fit(left_train + right_train)
    sim_word = cosine_similarity(vec_word.transform(left_eval), vec_word.
↪transform(right_eval))
    sim_char = cosine_similarity(vec_char.transform(left_eval), vec_char.
↪transform(right_eval))
    return alpha * sim_word + (1.0 - alpha) * sim_char

cfg2b = {
    "word": dict(ngram_range=(1, 2), min_df=2, max_df=0.95,
sublinear_tf=True),
    "char": dict(ngram_range=(2, 5), min_df=1, max_df=0.99,
sublinear_tf=True),
}
alpha2b = 0.25

# Решение 2: улучшаем решение 1 смешением word/char признаков
sim2b_train = _build_tfidf_blend(left_train, right_train_shuffled,
left_train, right_train_shuffled, cfg2b, alpha2b)
sim2b_test = _build_tfidf_blend(left_train, right_train_shuffled, left_test,
right_test_shuffled, cfg2b, alpha2b)
metrics2 = evaluate_similarity_argmax(sim2b_test, true_test)
print(
    f"2_word_char_tfidf: "
    f"score={metrics2['score']:.4f} (pub={metrics2['score_public']:.4f},
priv={metrics2['score_private']:.4f})"
)
```

2_word_char_tfidf: score=0.4938 (pub=0.4969, priv=0.4906)

1.12.5 Решение 3 — TF-IDF + greedy one-to-one

Берём матрицу скоров решения 2 и добавляем жадное взаимно-однозначное сопоставление для роста one2one.

[6]: *# Функция: строит жадное взаимно-однозначное соответствие*

```
def greedy_one2one_matching(sim: np.ndarray):
    n = sim.shape[0]
    flat_order = np.argsort(sim, axis=None)[::-1]
    pred = np.full(n, -1, dtype=int)
    used_left = np.zeros(n, dtype=bool)
    used_right = np.zeros(n, dtype=bool)
    matched = 0
    for idx in flat_order:
```

```
i, j = divmod(idx, n)
if (not used_left[i]) and (not used_right[j]):
    pred[i] = j
    used_left[i] = True
    used_right[j] = True
    matched += 1
    if matched == n:
        break
return pred

# Функция: считает единый score через greedy
def evaluate_similarity_greedy(sim: np.ndarray, true_idx: np.ndarray):
    pred = greedy_one2one_matching(sim)
    score = float((pred == true_idx).mean())
    score_public = _masked_acc(pred, true_idx, public_mask)
    score_private = _masked_acc(pred, true_idx, private_mask)
    return {
        "score": score,
        "score_public": score_public,
        "score_private": score_private,
    }

# Решение 3: добавляем greedy к решению 2
metrics3 = evaluate_similarity_greedy(sim2b_test, true_test)
print(
    f"3_tfidf_greedy: "
    f"score={metrics3['score']:.4f} (pub={metrics3['score_public']:.4f},
    priv={metrics3['score_private']:.4f})"
)
```

3_tfidf_greedy: score=0.5062 (pub=0.5094, priv=0.5031)

1.12.6 Решение 4 — logreg reranker

Улучшаем решение 3: добавляем pairwise-фичи и логистическую регрессию, которая переоценивает кандидатов перед `argmax`.

```
[7]: # Коротко: считаем статистики половинок и собираем pairwise-feature cube
def _half_stats(texts):
    stats = []
    for text in texts:
        lines = [line.strip() for line in text.split("\n") if line.strip()]
    or [text.strip()]
        words = [" ".join(ch for ch in w if ch.isalpha()) for w in text.
        ↪replace("\n", " ").split()]
        words = [w for w in words if w]
        letters = [ch for ch in text if ch.isalpha()]
        v_all = [sum(1 for ch in line.lower() if ch in vowels) for line in
        lines]
        v_edge = [sum(1 for ch in line.lower() if ch in vowels) for line in
        lines[-4:]]
```

```
cap_ratio = (sum(1 for w in words if w[0].isupper()) / len(words)) if
words else 0.0
latin_ratio = (sum(1 for ch in letters if 'a' <= ch.lower() <= 'z') /
len(letters)) if letters else 0.0
stats.append([len(lines), np.mean([len(line) for line in lines]), np.
↪mean(v_all), np.mean(v_edge), cap_ratio, latin_ratio])
return np.asarray(stats, dtype=np.float32)

# Коротко: переводим разницу скалярного признака в similarity
def _pair_sim(left_vals, right_vals):
    diff = np.abs(left_vals[:, None] - right_vals[None, :])
    norm = np.maximum(left_vals[:, None], right_vals[None, :])
    return 1.0 - diff / (norm + 1e-12)

# Коротко: формируем обучающие пары (true + hard negatives + random
negatives)
def _sample_pairs(cube, true_idx, hard_source_sim, hard_k=8, rand_k=8,
random_state=42):
    rng_local = np.random.default_rng(random_state)
    n = cube.shape[0]
    rows, labels = [], []
    for i in range(n):
        j_true = true_idx[i]
        rows.append(cube[i, j_true]); labels.append(1)
        hard_neg = [j for j in np.argsort(-hard_source_sim[i]) if j !=
j_true][:hard_k]
        forbidden = set(hard_neg + [j_true])
        pool = np.array([j for j in range(n) if j not in forbidden],
dtype=int)
        rand_neg = rng_local.choice(pool, size=min(rand_k, len(pool)),
replace=False).tolist() if len(pool) else []
        for j in hard_neg + rand_neg:
            rows.append(cube[i, j]); labels.append(0)
    return np.asarray(rows, dtype=np.float32), np.asarray(labels, dtype=np.
↪int32)

left_train_feat = _half_stats(left_train)
right_train_feat = _half_stats(right_train_shuffled)
left_test_feat = _half_stats(left_test)
right_test_feat = _half_stats(right_test_shuffled)

train_maps = [sim2b_train] + [_pair_sim(left_train_feat[:, k],
right_train_feat[:, k]) for k in range(left_train_feat.shape[1])] +
[_jaccard_matrix(word_train_l, word_train_r).astype(np.float32)]
test_maps = [sim2b_test] + [_pair_sim(left_test_feat[:, k], right_test_feat[:
↪, k]) for k in range(left_test_feat.shape[1])] +
[_jaccard_matrix(word_test_l, word_test_r).astype(np.float32)]
cube_train = np.stack(train_maps, axis=2).astype(np.float32)
cube_test = np.stack(test_maps, axis=2).astype(np.float32)
```

```
x_train, y_train = _sample_pairs(cube_train, true_train, sim2b_train,
hard_k=8, rand_k=8, random_state=42)
scaler = StandardScaler()
model = LogisticRegression(C=0.03, max_iter=2000, class_weight="balanced",
solver="lbfgs", random_state=42)
model.fit(scaler.fit_transform(x_train), y_train)
flat_scores = model.predict_proba(scaler.transform(cube_test.reshape(-1,
cube_test.shape[-1]))[:, 1]
test_scores_4 = flat_scores.reshape(cube_test.shape[0], cube_test.shape[1])

alpha4 = 0.82
sim4_test = alpha4 * test_scores_4 + (1.0 - alpha4) * sim2b_test
metrics4 = evaluate_similarity_argmax(sim4_test, true_test)
print(
    f"4_logreg_rerank: "
    f"score={metrics4['score']:.4f} (pub={metrics4['score_public']:.4f},
priv={metrics4['score_private']:.4f})"
)
```

4_logreg_rerank: score=0.5203 (pub=0.5031, priv=0.5375)

1.12.7 Решение 5 — logreg reranker + greedy

Финальное улучшение: к score-матрице решения 4 снова применяем greedy one-to-one для максимального one2one.

```
[8]: # Решение 5: добавляем greedy к скалам решения 4
metrics5 = evaluate_similarity_greedy(sim4_test, true_test)
print(
    f"5_logreg_rerank_greedy: "
    f"score={metrics5['score']:.4f} (pub={metrics5['score_public']:.4f},
priv={metrics5['score_private']:.4f})"
)
```

5_logreg_rerank_greedy: score=0.5656 (pub=0.5687, priv=0.5625)

```
[10]: # Сохраняем сабмит для автора из лучшего решения
sim_test = sim4_test
pred_idx = sim_test.argmax(axis=1)

author_submission_path = base_dir / "author_submission.csv"
with open(author_submission_path, "w", newline="", encoding="utf-8") as f:
    writer = csv.writer(f)
    writer.writerow(["left_id", "right_id"])
    for i, lid in enumerate(left_ids):
        writer.writerow([lid, right_ids[int(pred_idx[i])]])

print(f"Saved {author_submission_path}")
```

Saved /Users/aguschin/Git/uni/vsosh/zakl/nlp/author_submission.csv

1.12.8 Шаг 5. Результаты матчинга

Показываем только примеры матчинга для лучшего текущего решения: лучшие и худшие пары.

```
[12]: # Шаг 5. Результаты матчинга по квантилям для лучшего решения

def _preview_text(text, max_chars=220, max_lines=8):
    lines = [line.strip() for line in text.split("\n") if line.strip()][:
    ↪max_lines]
    preview = "\n".join(lines)
    return preview[:max_chars] + ("..." if len(preview) > max_chars else "")

sim_test = sim4_test
pred_idx = sim_test.argmax(axis=1)

quantiles_to_show = [0.99, 0.75, 0.5, 0.25, 0.0] # можно менять вручную
if not quantiles_to_show or any(q < 0 or q > 1 for q in quantiles_to_show):
    raise ValueError("Квантили должны быть в [0, 1], список не должен быть пустым")

match_scores = sim_test[np.arange(len(left_test)), pred_idx]
used = set()

def pick_idx_for_quantile(q):
    target = float(np.quantile(match_scores, q))
    order = np.argsort(np.abs(match_scores - target))
    idx = next((int(i) for i in order if int(i) not in used), int(order[0]))
    used.add(idx)
    return idx, target

print(f"Матчинг для лучшего решения")
print(f"Квантили: {quantiles_to_show}")
print()

for q in quantiles_to_show:
    i, target = pick_idx_for_quantile(q)
    j = int(pred_idx[i])
    print(f"=== q={q:.2f} | target={target:.4f} | sim={match_scores[i]:.4f} ===")
    print(f"left={i} -> pred_right={j} | true={true_test[i]} | correct={j == true_test[i]}")
    print("LEFT :", _preview_text(left_test[i]))
    print("RIGHT:", _preview_text(right_test_shuffled[j]))
    print()
```

Матчинг для лучшего решения

Квантили: [0.99, 0.75, 0.5, 0.25, 0.0]

=== q=0.99 | target=0.9055 | sim=0.9076 ===

left=383 -> pred_right=628 | true=628 | correct=True

LEFT : Человек живет на белом свете.

Где - не знаю. Суть совсем не в том.
Я - лежу в пристрелянном кювете,
Он - с мороза входит в теплый дом.
Человек живет на белом свете,
Он - в квартиру поднялся уже.
Я - лежу в пристрелянном ...
RIGHT: Человек живет на белом свете
Он - в квартире зажигает свет
Я - лежу в пристрелянном кювете,
Я - вмерзаю в ледяной кювет.
Снег не тает. Губы, щеки, веки
Он засыпал. И велит дрожать...
С думой о далеком человеке
Легче до а...

=== q=0.75 | target=0.7871 | sim=0.7870 ===
left=208 -> pred_right=508 | true=508 | correct=True
LEFT : Бронепоезда взывают вдруг,
Стылый ветер грудью разрывая.
Бронепоезда идут на юг
Вдоль твоих перронов,
Лозовая!
Звезды первую звезду зовут.
Дым заката холоден и розов.
Над бронеплощадками плывут
RIGHT: Бескозырки черные матросов.
Говорит, гремит, вздыхает бронь
Отдаленно
и громоподобно.
И горит на станции огонь,
Керосиновый огонь бездомный.
Лист осенний, запоздавший лист,
Братьев в путь-дорогу созывает.

=== q=0.50 | target=0.7115 | sim=0.7113 ===
left=342 -> pred_right=350 | true=350 | correct=True
LEFT : Славянка тихая, сколь ток приятен твой,
Когда, в осенний день, в твои глядятся воды
Холмы, одетые последнею красой
Полуотцветшие природы.
Спешу к твоим брегам... свод неба тих и чист;
При свете солнечном прохлада повезает...
RIGHT: Иду под рощю излучистой тропой;
Что шаг, то новая в глазах моих картина;
То вдруг сквозь чашу древ мелькает предо мной,
Как в дыме, светлая долина;
То вдруг исчезло все... окрест сгустился лес;
Все дико вокруг меня, и су...

=== q=0.25 | target=0.6497 | sim=0.6498 ===
left=532 -> pred_right=415 | true=415 | correct=True

LEFT : Посвящается Феллини
Мертвец играл на дудочке,
По городу гулял,
И незнакомой дурочке
Он руку предлагал.
А дурочка, как Золушка,
Ему в глаза глядит, -
Он говорит о золоте,
RIGHT: О славе говорит.
Мертвец, певец и умница,
Его слова просты -
Пусты ночные улицы,
И площади пусты.
«Мне больно, мне невесело,
Мне холодно зимой,
Возьми меня невестою,

```
=== q=0.00 | target=0.1137 | sim=0.1137 ===  
left=533 -> pred_right=373 | true=329 | correct=False  
LEFT : Нет.  
Это неправда.  
Нет!  
И ты?  
Любимая,  
за что,  
за что же?!  
Хорошо -  
RIGHT: Что нашу грусть -  
В листы,  
И груз - в цветы  
Всего за только всхруст  
Руки  
В руке:  
Игру.  
Индуc, а может Златоуст
```

Е. Подозрительные пирожные

Решение состоит из следующих этапов: 1. Загружаем предобученную CNN и получаем эмбединги для картинок. 2. Вычисляем для каждой картинки outlier-метрики `robust_mahalanobis`, `global_knn`, `class_knn`. 3. Агрегируем метрики с весами (2.0, 1.5, 0.5). 4. Сортируем картинки по значению матрицы, выбираем top-K и сохраняем в `submission_author.csv`.

```
[1]: import json
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset

ARTIFACTS_DIR = "."
TEST_PACKAGE = f"{ARTIFACTS_DIR}/public_test_package.npz"
TEST_META = f"{ARTIFACTS_DIR}/public_test_meta.json"
WEIGHTS_PATH = f"{ARTIFACTS_DIR}/model_weights.pt"
SUBMISSION_PATH = "submission.csv"

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", DEVICE)
```

Device: cpu

Код сети и получения эмбедингов картинок:

```
[2]: class SmallCNN(nn.Module):
    def __init__(self, n_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
        )
        self.fc1 = nn.Linear(64 * 8 * 8, 64)
        self.act = nn.ReLU(inplace=True)
        self.fc2 = nn.Linear(64, n_classes)

    def forward(self, x):
        z = self.features(x)
        z = z.flatten(1)
        h = self.act(self.fc1(z))
        logits = self.fc2(h)
        return logits, h
```

```
def infer_embeddings_and_preds(model, x_np, batch_size=256):
    ds = TensorDataset(torch.from_numpy(x_np).float())
    loader = DataLoader(ds, batch_size=batch_size, shuffle=False)
    all_emb, all_pred = [], []
    model.eval()
    with torch.no_grad():
        for (xb,) in loader:
            xb = xb.to(DEVICE)
            logits, emb = model(xb)
            all_emb.append(emb.cpu().numpy())
            all_pred.append(logits.argmax(dim=1).cpu().numpy())
    emb = np.concatenate(all_emb, axis=0).astype(np.float32)
    pred = np.concatenate(all_pred, axis=0).astype(np.int64)
    return emb, pred

state = torch.load(WEIGHTS_PATH, map_location=DEVICE)
n_classes = int(state["fc2.weight"].shape[0]) if "fc2.weight" in state else 10
model = SmallCNN(n_classes=n_classes)
model.load_state_dict(state)
model = model.to(DEVICE).eval()
print("Model loaded")
```

Model loaded

```
[3]: test = np.load(TEST_PACKAGE)
x_test = test["images"].astype(np.float32)

K = 1000

emb_test, pred_test = infer_embeddings_and_preds(model, x_test)

n_test = len(x_test)
print(f"n_test={n_test}, K={K}, emb_dim={emb_test.shape[1]}")
```

n_test=10000, K=1000, emb_dim=64

1.12.9 Идея решения

Будем искать выбросы в пространстве эмбедингов нейросети.

Предполагаем, что после прохождения через обученную модель объекты одного и того же класса образуют в пространстве признаков компактные кластеры. Тогда выбросы можно искать как точки, которые расположены “нетипично” относительно остальных точек.

Для этого используются несколько типов метрик:

1. Global kNN score.

Для каждого объекта считается среднее расстояние до его ближайших соседей среди всех объектов датасета.

Если объект находится изолированно, его score будет большим.

2. Class kNN score.

Аналогично предыдущему пункту, но соседи ищутся только среди объектов того же

предсказанного класса.

Это позволяет находить точки, которые выглядят необычно именно внутри своего класса, даже если глобально они не слишком далеки от других.

3. Robust Mahalanobis score внутри класса.

Для каждого предсказанного класса оценивается среднее μ и матрица ковариаций Σ эмбедингов, после чего для каждого объекта считается расстояние Махаланобиса до центра своего класса. Расстояние Махаланобиса считается так:

$$(x_i - \mu)^T \Sigma^{-1} (x_i - \mu)$$

Идея использования ковариации в том, что Махаланобис смотрит на точку относительно формы распределения класса. Иногда точку-выброс можно поймать тем, что она отклоняется не по длине, а по “неправильному направлению” в пространстве. Чтобы выбросы не искажали изначальные оценки μ и Σ по выборке, используем робастную схему: на каждой итерации временно отбрасываем самые далёкие точки, и оцениваем параметры только по оставшемуся “ядру” класса.

Таким образом, решение опирается на следующую гипотезу:

выбросы — это объекты, которые в пространстве признаков либо далеки от типичного распределения своего класса, либо плохо вписываются в локальную структуру соседей.

Примечание: без использования расстояния Махаланобиса можно получить до 80% баллов за задачу.

Ниже реализованы функции метрик:

```
[4]: import numpy as np

def score_robust_mahalanobis(emb, pred, lam=0.15, trim_frac=0.10, iters=2):
    """
    Считает робастный Mahalanobis score внутри каждого предсказанного класса.

    Идея:
    - для каждого класса оцениваем "нормальное" распределение эмбедингов;
    - затем для каждого объекта считаем расстояние Махаланобиса до центра
    класса;
    - чтобы выбросы не портили оценку среднего и ковариации,
      несколько раз отбрасываем самые далёкие точки и пересчитываем
    статистики.

    Параметры:
    - emb: матрица эмбедингов shape [N, d]
    - pred: предсказанные классы shape [N]
    - lam: коэффициент регуляризации ковариации
    - trim_frac: доля самых далёких точек, временно исключаемых при trimming
    - iters: число итераций робастного пересчёта
    """
    d = emb.shape[1] # размерность пространства признаков
    out = np.zeros(len(emb), dtype=np.float64) # итоговый score для всех
    объектов
    eye = np.eye(d, dtype=np.float64) # единичная матрица для регуляризации
```

```
# Обрабатываем каждый класс отдельно
for c in np.unique(pred):
    m = pred == c          # маска объектов класса c
    h = emb[m].astype(np.float64) # эмбединги только этого класса
    n = len(h)

    # Если объектов слишком мало, робастную оценку делать ненадёжно.
    # Тогда просто считаем обычное расстояние Махаланобиса.
    if n < 8:
        mu = h.mean(axis=0, keepdims=True) # центр класса
        cen = h - mu                        # центрированные эмбединги

        # Ковариация + диагональная регуляризация для устойчивости
        cov = (cen.T @ cen) / max(1, n - 1) + lam * eye

        # Псевдообратная матрица вместо обычной обратной -
        # устойчивее, если ковариация плохо обусловлена
        inv = np.linalg.pinv(cov)

        # Квадрат расстояния Махаланобиса для каждой точки:
        #  $(x - \mu)^T \text{inv} (x - \mu)$ 
        out[m] = np.einsum("bi,ij,bj->b", cen, inv, cen)
        continue

    # Изначально считаем, что сохраняем все точки класса
    keep = np.ones(n, dtype=bool)

    # Начальная грубая оценка среднего и обратной ковариации
    mu = h.mean(axis=0, keepdims=True)
    inv = np.linalg.pinv(np.cov(h.T) + lam * eye)

    # Несколько итераций робастного trimming
    for _ in range(max(1, iters)):
        # Берём только текущие "надёжные" точки
        hh = h[keep]

        # Пересчитываем центр по оставшимся точкам
        mu = hh.mean(axis=0, keepdims=True)

        # Центрируем оставшиеся точки
        cen_hh = hh - mu

        # Пересчитываем ковариацию по очищенному подмножеству
        cov = (cen_hh.T @ cen_hh) / max(1, len(hh) - 1) + lam * eye
        inv = np.linalg.pinv(cov)

    # Считаем расстояния Махаланобиса уже для всех точек класса
    cen_all = h - mu
    dist_all = np.einsum("bi,ij,bj->b", cen_all, inv, cen_all)
```

```
# Порог: оставляем (1 - trim_frac) долю ближайших точек
thr = float(np.quantile(dist_all, 1.0 - trim_frac))
keep = dist_all <= thr

# Защита от ситуации, когда осталось слишком мало точек:
# тогда ослабляем trimming
if keep.sum() < max(5, int(0.5 * n)):
    keep = dist_all <= float(np.quantile(dist_all, 0.7))

# После финальной оценки считаем итоговый Mahalanobis score
# для всех точек этого класса
cen = h - mu
out[m] = np.einsum("bi,ij,bj->b", cen, inv, cen)

return out

def score_global_knn(emb, k=10):
    """
    Считает глобальный kNN score:
    среднее расстояние до k ближайших соседей по всему датасету.

    Если точка находится изолированно от остальных,
    её score будет большим.
    """
    n = emb.shape[0]

    # Если точек слишком мало, meaningful score нет
    if n <= 2:
        return np.zeros(n, dtype=np.float64)

    # Нельзя брать соседей больше, чем число остальных точек
    kk = min(k, n - 1)

    x = emb.astype(np.float64)

    # Квадраты норм для всех точек
    x2 = np.sum(x * x, axis=1, keepdims=True)

    # Матрица квадратов попарных евклидовых расстояний:
    # ||xi - xj||^2 = ||xi||^2 + ||xj||^2 - 2 <xi, xj>
    d2 = x2 + x2.T - 2.0 * (x @ x.T)

    # Убираем расстояние точки до самой себя
    np.fill_diagonal(d2, np.inf)

    # Берём kk наименьших расстояний в каждой строке
    knn = np.partition(d2, kk - 1, axis=1)[: , :kk]
```

```
# Возвращаем среднее обычное расстояние до ближайших соседей
return np.mean(np.sqrt(np.maximum(knn, 0.0)), axis=1)

def score_class_knn(emb, pred, k=8):
    """
    Считает class-wise kNN score:
    среднее расстояние до k ближайших соседей внутри предсказанного класса.

    Это помогает находить объекты, которые нетипичны
    именно для своего класса.
    """
    out = np.zeros(len(emb), dtype=np.float64)

    # Считаем score отдельно внутри каждого класса
    for c in np.unique(pred):
        m = pred == c
        h = emb[m].astype(np.float64)
        n = len(h)

        # Если в классе слишком мало точек, score считаем нулевым
        if n <= 2:
            out[m] = 0.0
            continue

        kk = min(k, n - 1)

        # Квадраты норм точек внутри класса
        h2 = np.sum(h * h, axis=1, keepdims=True)

        # Матрица квадратов попарных расстояний внутри класса
        d2 = h2 + h2.T - 2.0 * (h @ h.T)

        # Исключаем расстояние до самой себя
        np.fill_diagonal(d2, np.inf)

        # Находим kk ближайших соседей
        knn = np.partition(d2, kk - 1, axis=1)[: , :kk]

        # Среднее расстояние до ближайших соседей внутри класса
        out[m] = np.mean(np.sqrt(np.maximum(knn, 0.0)), axis=1)

    return out

def rank_score(x):
    """
    Преобразует массив значений в ранги.

    Наименьший элемент получает ранг 0,
    следующий - 1, и так далее.
    """
```

```
Это удобно, если потом нужно объединять  
несколько score разного масштаба.  
"""  
# Индексы элементов в порядке возрастания значений x  
order = np.argsort(x)  
  
# Массив для рангов  
r = np.empty_like(order, dtype=np.float64)  
  
# Записываем для каждого элемента его позицию в отсортированном порядке  
r[order] = np.arange(len(x), dtype=np.float64)  
return r
```

```
[5]: # Строим outlier-score на тестовых эмбедингах по ансамблю рангов  
# из трёх метрик: robust Mahalanobis + global kNN + class kNN.  
  
# Робастное расстояние Махаланобиса внутри предсказанного класса:  
# показывает, насколько объект нетипичен относительно распределения своего  
класса.  
s_robust = score_robust_mahalanobis(emb_test, pred_test, lam=0.15,  
trim_frac=0.10, iters=2)  
  
# Глобальный kNN-score:  
# среднее расстояние до ближайших соседей во всём наборе.  
# Большие значения соответствуют более изолированным точкам.  
s_gknn = score_global_knn(emb_test, k=10)  
  
# Class-wise kNN-score:  
# среднее расстояние до ближайших соседей только внутри своего предсказанного  
класса.  
# Помогает находить точки, которые странно выглядят именно внутри класса.  
s_cknn = score_class_knn(emb_test, pred_test, k=8)  
  
# Объединяем три score в один итоговый показатель подозрительности.  
# Перед объединением переводим каждый score в ранги,  
# чтобы разные шкалы значений не мешали друг другу.  
# Здесь robust Mahalanobis имеет наибольший вес,  
# global kNN - средний, class kNN - меньший дополнительный вклад.  
final_score = (  
    2.0 * rank_score(s_robust)  
    + 1.5 * rank_score(s_gknn)  
    + 0.5 * rank_score(s_cknn)  
)  
  
# Выбираем K объектов с наибольшим итоговым score  
# как наиболее вероятные выбросы.  
topk = np.argsort(-final_score)[:K]  
  
# Формируем бинарный вектор ответов:  
# 1 - объект считаем выбросом, 0 - обычным объектом.
```

```
is_outlier = np.zeros(n_test, dtype=np.int64)
is_outlier[topk] = 1

# Собираем файл для отправки:
# для каждого id указываем предсказание is_outlier.
submission = pd.DataFrame({
    "id": np.arange(n_test, dtype=np.int64),
    "is_outlier": is_outlier,
})

# Сохраняем submission в CSV без индекса DataFrame.
submission.to_csv(SUBMISSION_PATH, index=False)

# Выводим служебную информацию:
# куда сохранён файл и сколько объектов помечено как выбросы.
print("Saved:", SUBMISSION_PATH)
print("Predicted outliers:", int(is_outlier.sum()))

# Показываем первые строки submission-таблицы.
submission.head()
```

Saved: submission.csv
Predicted outliers: 1000

```
[5]:   id  is_outlier
0    0           0
1    1           0
2    2           0
3    3           0
4    4           0
```

Проверим решение: ячейка ниже вычисляет скор на public и private частях датасета.

```
[6]: #!/usr/bin/env python3
"""
Validate submission.csv against y_test.csv and compute hits@k metrics.

Expected columns:
y_test.csv: id, is_outlier_true, Usage
submission.csv: id, is_outlier
"""

from __future__ import annotations

import argparse
import json
from pathlib import Path
from typing import Dict, List

import pandas as pd

BASELINE_SOLUTION_SCORE_PUBLIC=113
```

```
BASELINE_SOLUTION_SCORE_PRIVATE=95
AUTHOR_SOLUTION_SCORE_PUBLIC=338
AUTHOR_SOLUTION_SCORE_PRIVATE=308

def _strip_jupyter_kernel_args(unknown: List[str]) -> List[str]:
    cleaned: List[str] = []
    i = 0
    while i < len(unknown):
        tok = unknown[i]
        if tok == "-f" and i + 1 < len(unknown) and unknown[i + 1].
↪endswith(".json"):
            i += 2
            continue
        cleaned.append(tok)
        i += 1
    return cleaned

def _validate_binary_column(col: pd.Series, name: str) -> pd.Series:
    numeric = pd.to_numeric(col, errors="coerce")
    bad = numeric.isna() | (~numeric.isin([0, 1]))
    if bad.any():
        first_bad_idx = int(col.index[bad][0])
        raise ValueError(
            f"Колонка '{name}' должна содержать только 0/1. "
            f"Первая некорректная строка: {first_bad_idx}"
        )
    return numeric.astype(int)

def _clip_score_0_100(value: float) -> float:
    return max(0.0, min(100.0, float(value)))

def _compute_hits_metrics(
    df: pd.DataFrame, include_score_0_100: bool = True
) -> Dict[str, float | int | bool]:
    k_true = int(df["is_outlier_true"].sum())
    k_pred = int(df["is_outlier"].sum())
    hits_at_k = int(((df["is_outlier_true"] == 1) & (df["is_outlier"] == 1)).
↪sum())
    if AUTHOR_SOLUTION_SCORE <= BASELINE_SOLUTION_SCORE:
        normalized_score_0_100 = 0.0
    else:
        normalized_score_0_100 = 100.0 * (
            (hits_at_k - BASELINE_SOLUTION_SCORE)
            / (AUTHOR_SOLUTION_SCORE - BASELINE_SOLUTION_SCORE)
        )
    normalized_score_0_100 = _clip_score_0_100(normalized_score_0_100)
```

```
metrics: Dict[str, float | int | bool] = {
    "n_samples": int(len(df)),
    "k_true": k_true,
    "k_pred": k_pred,
    "k_match": bool(k_true == k_pred),
    "hits_at_k": hits_at_k,
}
if include_score_0_100:
    metrics["score_0_100"] = normalized_score_0_100
return metrics

def validate_and_score_submission(ytest_path: Path, submission_path: Path) ->
Dict[str, object]:
    try:
        ytest = pd.read_csv(ytest_path)
    except Exception as exc: # pragma: no cover
        raise ValueError(f"Ошибка чтения ytest.csv: {exc}") from exc

    try:
        submission = pd.read_csv(submission_path)
    except Exception as exc: # pragma: no cover
        raise ValueError(f"Ошибка чтения submission.csv: {exc}") from exc

    if ytest.empty:
        raise ValueError("ytest.csv пустой")
    if submission.empty:
        raise ValueError("submission.csv пустой")

    required_ytest_cols = {"id", "is_outlier_true", "Usage"}
    required_submission_cols = {"id", "is_outlier"}

    missing_ytest_cols = required_ytest_cols - set(ytest.columns)
    if missing_ytest_cols:
        raise ValueError(f"В ytest.csv отсутствуют колонки:
{sorted(missing_ytest_cols)}")

    missing_submission_cols = required_submission_cols - set(submission.
↪columns)
    if missing_submission_cols:
        raise ValueError(
            f"В submission.csv отсутствуют колонки:
{sorted(missing_submission_cols)}"
        )

    ytest = ytest[["id", "is_outlier_true", "Usage"]].copy()
    submission = submission[["id", "is_outlier"]].copy()

    ytest["id"] = ytest["id"].astype(str).str.strip()
```

```
ytest["Usage"] = ytest["Usage"].astype(str).str.strip().str.title()

submission["id"] = submission["id"].astype(str).str.strip()

if (ytest["id"] == "").any():
    raise ValueError("B ytest.csv есть пустые id")
if (submission["id"] == "").any():
    raise ValueError("B submission.csv есть пустые id")

invalid_usage = sorted(set(ytest["Usage"]) - {"Public", "Private"})
if invalid_usage:
    raise ValueError(f"B ytest.csv недопустимые значения Usage: {invalid_usage}")

if ytest["id"].duplicated().any():
    duplicates = (
        ytest.loc[ytest["id"].duplicated(keep=False), "id"]
        .unique()
        .tolist()
    )
    suffix = " ..." if len(duplicates) > 10 else ""
    raise ValueError(
        f"B ytest.csv есть дубликаты id: {duplicates[:10]}{suffix} "
        f"(всего {len(duplicates)})"
    )

if submission["id"].duplicated().any():
    duplicates = (
        submission.loc[submission["id"].duplicated(keep=False), "id"]
        .unique()
        .tolist()
    )
    suffix = " ..." if len(duplicates) > 10 else ""
    raise ValueError(
        f"B submission.csv есть дубликаты id: {duplicates[:10]}{suffix} "
        f"(всего {len(duplicates)})"
    )

y_ids = set(ytest["id"])
s_ids = set(submission["id"])

missing_ids = y_ids - s_ids
if missing_ids:
    miss = sorted(list(missing_ids))
    suffix = " ..." if len(miss) > 10 else ""
    raise ValueError(
        f"B submission.csv отсутствуют id: {miss[:10]}{suffix} "
        f"(всего {len(miss)})"
    )
```

```
extra_ids = s_ids - y_ids
if extra_ids:
    extra = sorted(list(extra_ids))
    suffix = " ..." if len(extra) > 10 else ""
    raise ValueError(
        f"B submission.csv есть лишние id: {extra[:10]}{suffix} "
        f"(всего {len(extra)})"
    )

ytest["is_outlier_true"] =
_validate_binary_column(ytest["is_outlier_true"], "is_outlier_true")
submission["is_outlier"] =
_validate_binary_column(submission["is_outlier"], "is_outlier")

merged = ytest.merge(submission, on="id", how="left",
validate="one_to_one")

if merged["is_outlier"].isna().any():
    raise ValueError("После merge есть NaN в предсказаниях")

k_true_total = int(merged["is_outlier_true"].sum())
k_pred_total = int(merged["is_outlier"].sum())
if k_pred_total != k_true_total:
    raise ValueError(
        "B submission.csv должно быть ровно K единиц в колонке
is_outlier, "
        f"где K={k_true_total}. Сейчас: {k_pred_total}"
    )

overall_metrics = _compute_hits_metrics(merged, include_score_0_100=True)
public_metrics = _compute_hits_metrics(
    merged.loc[merged["Usage"] == "Public"], include_score_0_100=False
)
private_metrics = _compute_hits_metrics(
    merged.loc[merged["Usage"] == "Private"], include_score_0_100=False
)
overall_metrics["score_0_100"] =
_clip_score_0_100(overall_metrics["score_0_100"])

return {
    **overall_metrics,
    "n_total": int(len(merged)),
    "n_public": int((merged["Usage"] == "Public").sum()),
    "n_private": int((merged["Usage"] == "Private").sum()),
    "public": public_metrics,
    "private": private_metrics,
}

def main(argv: List[str] | None = None) -> None:
```

```
parser = argparse.ArgumentParser(description="Проверка сабмита и подсчет  
hits@k")  
parser.add_argument("--ytest", type=str, default="y_test.csv", help="Путь  
к y_test.csv")  
parser.add_argument(  
    "--submission",  
    type=str,  
    default="submission.csv",  
    help="Путь к submission.csv",  
)  
parser.add_argument(  
    "--save-json",  
    type=str,  
    default=None,  
    help="Опциональный путь для сохранения метрик в JSON",  
)  
args, unknown = parser.parse_known_args(argv)  
unknown = _strip_jupyter_kernel_args(unknown)  
if unknown:  
    parser.error(f"unrecognized arguments: {' '.join(unknown)}")  
  
ytest_path = Path(args.ytest)  
submission_path = Path(args.submission)  
  
if not ytest_path.exists():  
    raise FileNotFoundError(f"файл ytest не найден: {ytest_path}")  
if not submission_path.exists():  
    raise FileNotFoundError(f"файл submission не найден:  
{submission_path}")  
  
metrics = validate_and_score_submission(ytest_path, submission_path)  
print(json.dumps(metrics, indent=2))  
  
if args.save_json is not None:  
    out_path = Path(args.save_json)  
    with out_path.open("w", encoding="utf-8") as file:  
        json.dump(metrics, file, indent=2)  
    print(f"Метрики сохранены в JSON: {out_path}")  
  
if __name__ == "__main__":  
    main()
```