

Обучение классических моделей на больших данных



План



Типы больших данных



План



Типы больших данных



Классические модели



План



Типы больших данных



Классические модели



Популярные реализации



Какие они — большие данные?

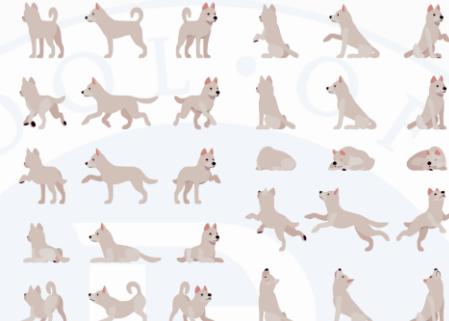
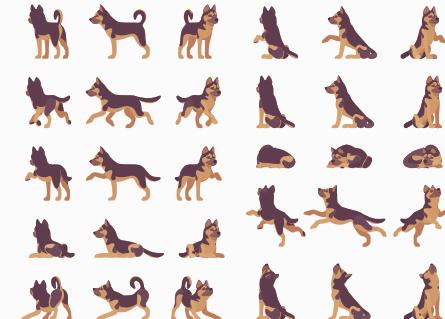
→ Большие данные могут принимать самые разнообразные формы



Какие они — большие данные?



В машинном обучении
данные — это
набор наблюдений,
состоящих из признаков
и правильного ответа



Какие они — большие данные?

→ Поэтому данные для обучения представляются в виде таблицы, где столбцы — признаки, строчки — наблюдения

Признаки

Наблюдения

Ответ	Признак 1	Признак 2	...
1	1.25	2	...
1	0.53	1	...
0	0.34	1	...
1	1.34	3	...
...

Какие они — большие данные?



Табличные данные могут быть большими по двум параметрам:

- Широкие — много столбцов в таблице
- Высокие — много строк в таблице

«Ширина» данных

«Высота» данных

Ответ	Признак 1	Признак 2	...
1	1.25	2	...
1	0.53	1	...
0	0.34	1	...
1	1.34	3	...
...

Высокие данные

→ Высокие данные в обучении — это огромное количество самих наблюдений



Высокие данные



Примеры таких данных:

→ Логи сервера



Высокие данные



Примеры таких данных:

→ Логи сервера



→ Рейтинги фильмов



Высокие данные



Примеры таких данных:

→ Логи сервера



→ Рейтинги фильмов

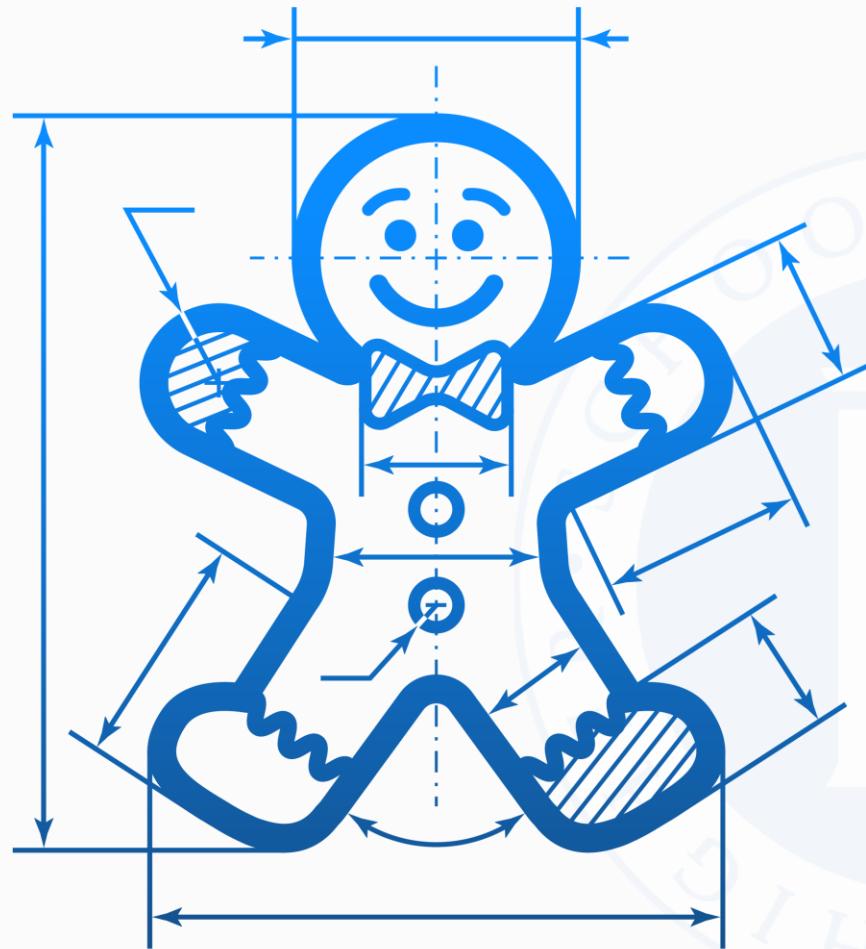


→ Клики по рекламе



Широкие данные

→ Широкие данные — это огромное количество признаков в одном наблюдении



Широкие данные



Примеры таких данных:



Граф друзей в социальной сети



Широкие данные



Примеры таких данных:

→ Граф друзей в социальной сети

→ Пользовательские предпочтения



Широкие данные



Примеры таких данных:



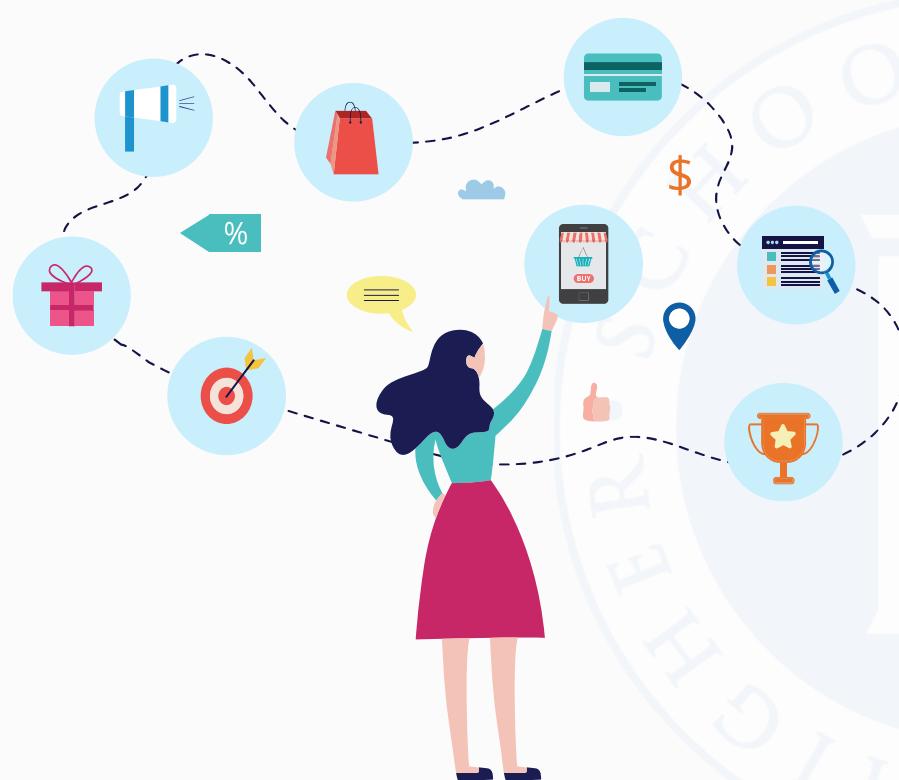
Граф друзей в социальной сети



Тексты



Пользовательские предпочтения



Широкие данные



Примеры таких данных:

→ Граф друзей в социальной сети

→ Тексты

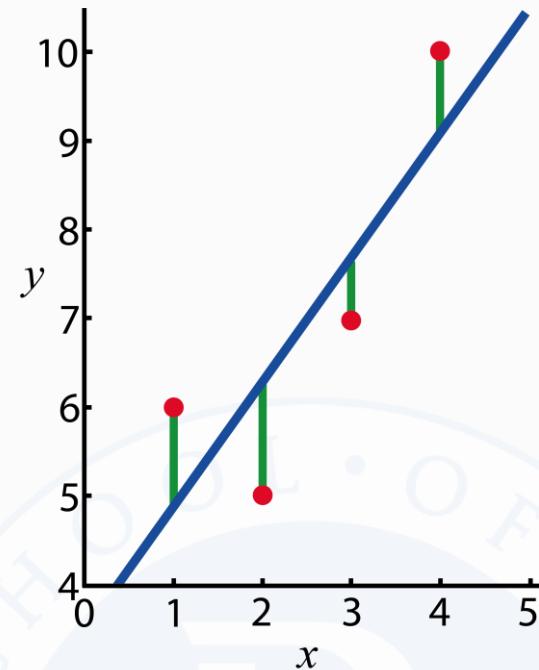
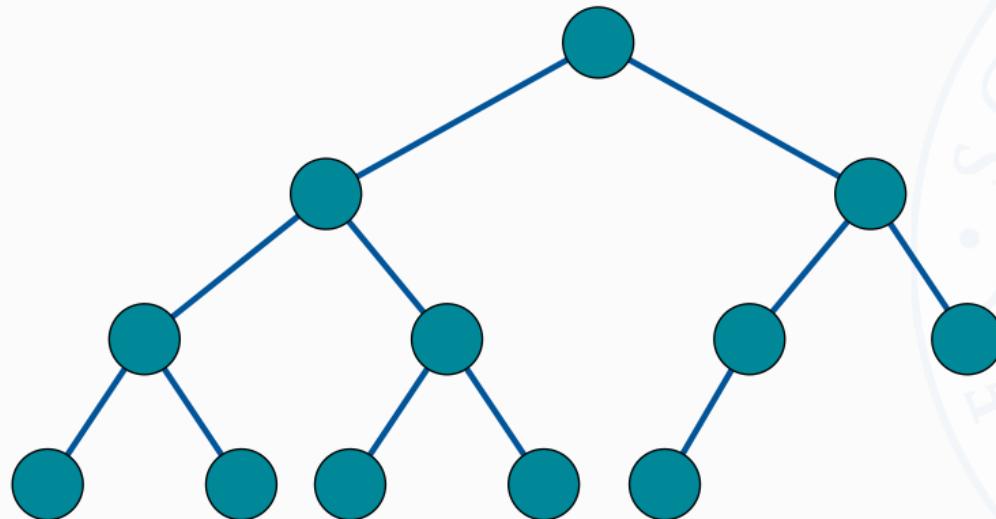
→ Пользовательские предпочтения

→ Геном человека



Что обучаем?

- На очень больших данных простые модели часто работают хорошо
- К таким относятся **линейные модели** и модели, основанные на **деревьях решений**



Что обучаем?

→ На этой неделе мы изучим, какие техники позволяют эффективно проводить обучение и какие популярные инструменты для этого используются



VOWPAL WABBIT



Yandex
CatBoost



Проблема широких данных



Где можно встретить широкие данные?

→ Проблема широких данных — признаковое пространство нашей модели слишком **большое**

«Ширина» данных

«Высота» данных

Ответ	Признак 1	Признак 2	...
1	1.25	2	...
1	0.53	1	...
0	0.34	1	...
1	1.34	3	...
...

Bag of words

→ Bag-of-words кодирование текстов может выдавать до нескольких миллионов признаков



Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед Додон в дуду дудел
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед Додон в дуду дудел
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед **Додон** в дуду дудел
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед Додон **в** дуду дудел
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед Додон в **дуду** дудел
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед Додон в дуду **дудел**
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words



Объект выборки — какой-то текст

Наличие слова в тексте — это **отдельный признак**

- 1) Дед Додон в дуду дудел
- 2) Димку дед дудой задел

№	дед	Додон	Димка	дуда	дудеть	задеть	в
1)	1	1	0	1	1	0	1
2)	1	0	1	1	0	1	0

Bag of words

→ Таким образом, количество уникальных слов — это количество признаков

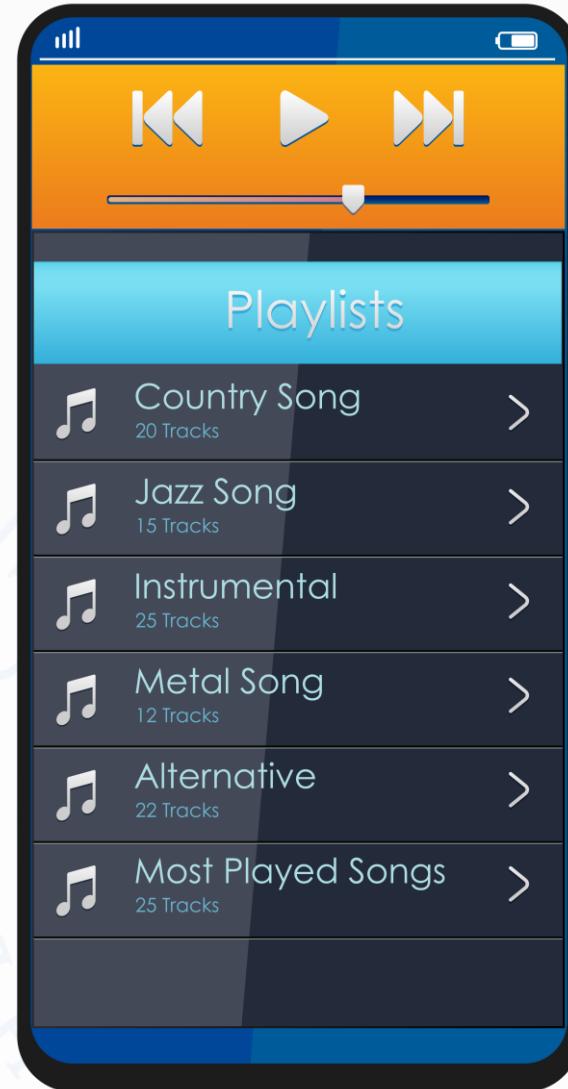
→ В английском языке уникальных слов около 1 миллиона
В русском языке — около 500 тысяч

Признаки

A	Абажур	Абзац	...	Ящер	Ящик	Ящур
1	0	0	...	1	1	0

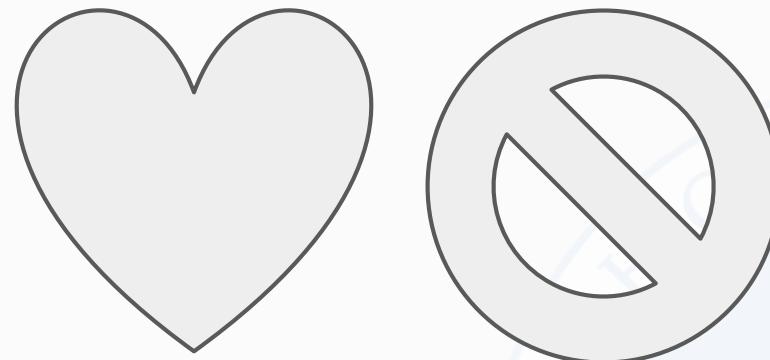
Предпочтения

- Пользователь может оценивать какие-то объекты — например, музыку
- Тогда пользовательские предпочтения — это **СПИСОК ОЦЕНОК** пользователя для **всех элементов** в коллекции



Предпочтения

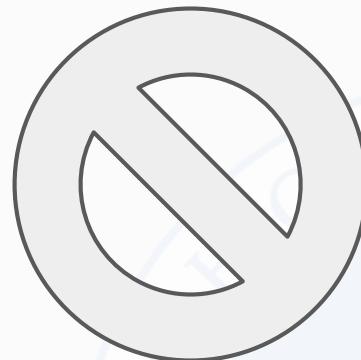
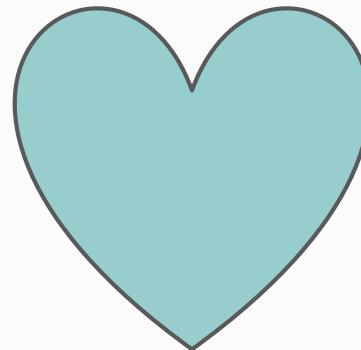
→ Так как признаковое пространство должно быть общим, для каждого пользователя нужно хранить список всех его оценок



Bob Dylan	John Lennon	The Beatles	...	Nirvana
1	0	N\A	...	1

Предпочтения

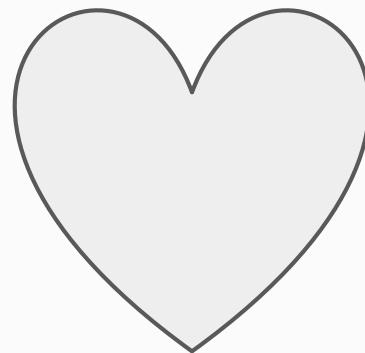
→ Так как признаковое пространство должно быть общим, для каждого пользователя нужно хранить список всех его оценок



Bob Dylan	John Lennon	The Beatles	...	Nirvana
1	0	N\A	...	1

Предпочтения

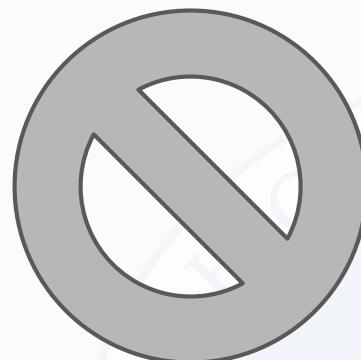
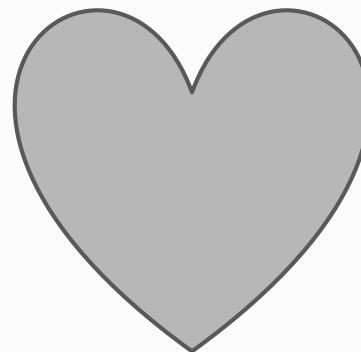
→ Так как признаковое пространство должно быть общим, для каждого пользователя нужно хранить список всех его оценок



Bob Dylan	John Lennon	The Beatles	...	Nirvana
1	0	N\A	...	1

Предпочтения

→ Так как признаковое пространство должно быть общим, для каждого пользователя нужно хранить список всех его оценок



Bob Dylan	John Lennon	The Beatles	...	Nirvana
1	0	N\A	...	1

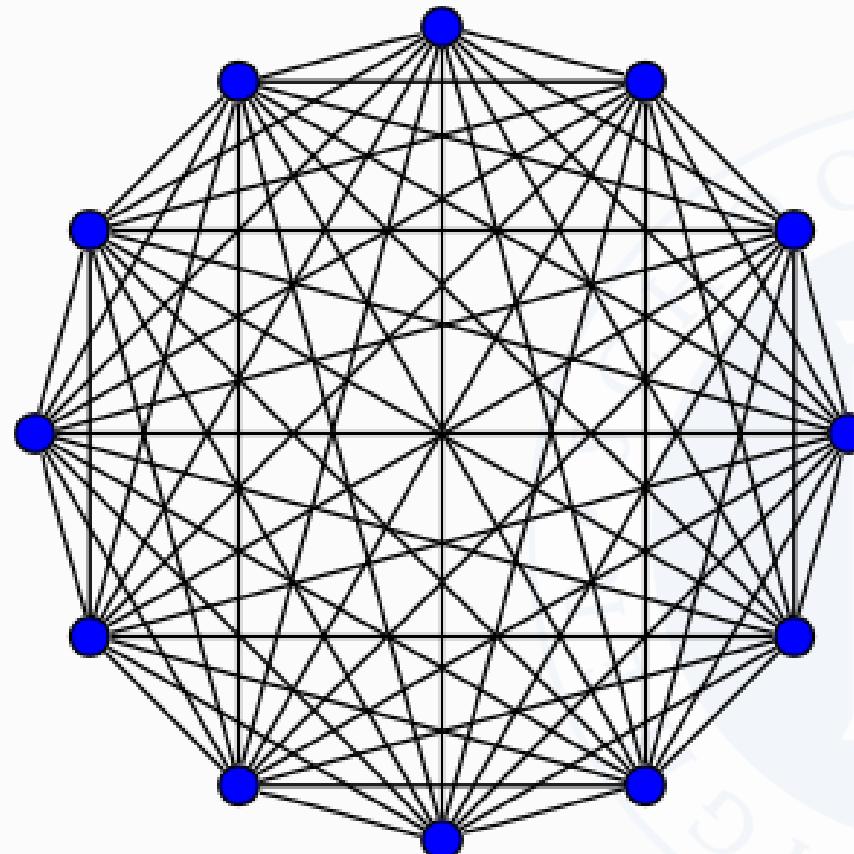
Предпочтения

- Размер признакового пространства будет не меньше размера ассортимента в сервисе
- Для примера, в Spotify около 50 миллионов песен



Комбинация признаков

→ Для повышения качества иногда приходится комбинировать признаки. Новых признаков может быть в несколько раз больше



Комбинация признаков



Например, мы хотим предсказать оценку пользователя в приложении для такси. Изначально у нас могут быть такие признаки:



День недели — 7 значений



Комбинация признаков



Например, мы хотим предсказать оценку пользователя в приложении для такси. Изначально у нас могут быть такие признаки:

- День недели — 7 значений
- Время суток — 4 значения



Комбинация признаков



Например, мы хотим предсказать оценку пользователя в приложении для такси. Изначально у нас могут быть такие признаки:

- День недели — 7 значений
- Время суток — 4 значения
- Месяц — 12 значений



Комбинация признаков



Например, мы хотим предсказать оценку пользователя в приложении для такси. Изначально у нас могут быть такие признаки:

- День недели — 7 значений
- Время суток — 4 значения
- Месяц — 12 значений
- Улица — 500 значений



Комбинация признаков



Например, мы хотим предсказать оценку пользователя в приложении для такси. Изначально у нас могут быть такие признаки:

- День недели — 7 значений
- Время суток — 4 значения
- Месяц — 12 значений
- Улица — 500 значений
- Тип погоды — 10 значений



Комбинация признаков



Например, мы хотим предсказать оценку пользователя в приложении для такси. Изначально у нас могут быть такие признаки:

- День недели — 7 значений
- Время суток — 4 значения
- Месяц — 12 значений
- Улица — 500 значений
- Тип погоды — 10 значений
- Модель машины — 50 значений



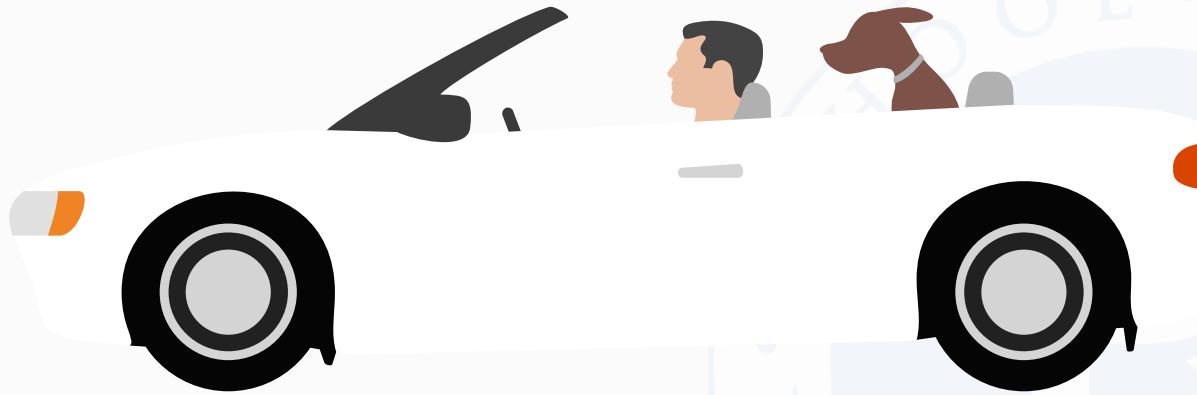
Комбинация признаков



Простым моделям может не хватать информации без контекста. С простыми признаками модель задается вопросом:



Машина без крыши хорошо или плохо влияет на оценку пользователя?



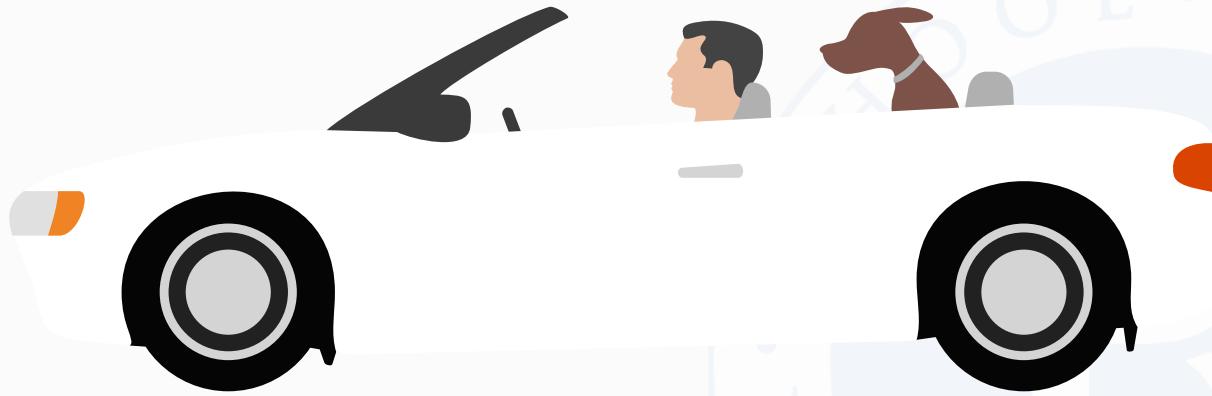
Комбинация признаков



Простым моделям может не хватать информации без контекста. С простыми признаками модель задается вопросом:



Машина без крыши хорошо или плохо влияет на оценку пользователя?



Качество с таким общим вопросом может быть не самым высоким

Комбинация признаков

→ Можем скомбинировать с признаком погоды. Тогда модель будет задаваться более сложными вопросами:

Машина без крыши в дождь **хорошо или плохо** влияет на оценку пользователя?

Машина без крыши в солнечную погоду **хорошо или плохо** влияет на оценку пользователя?



Комбинация признаков

→ Можем скомбинировать с признаком погоды. Тогда модель будет задаваться более сложными вопросами:

Машина без крыши в дождь хорошо или плохо влияет на оценку пользователя?

Машина без крыши в солнечную погоду хорошо или плохо влияет на оценку пользователя?



→ Такие признаки позволяют модели использовать больше информации

Комбинация признаков

→ Скombинируем все признаки. Итоговый признак будет означать следующее:

В день A в четверти дня B в месяц C
на улице D, когда на улице была погода E,
пользователь поехал на машине F

→ Где A, B, C, D, E, F — это все возможные варианты изначальных признаков

Комбинация признаков

→ Количество таких новых признаков несложно подсчитать:

$$7 \times 4 \times 12 \times 500 \times 10 \times 50 = 84\,000\,000$$

Комбинация признаков

→ Количество таких новых признаков несложно подсчитать:

$$7 \times 4 \times 12 \times 500 \times 10 \times 50 = 84\ 000\ 000$$

→ Таким образом, изначально **небольшое** количество признаков превратилось в **несколько миллионов**

Комбинация признаков

→ Количество таких новых признаков несложно подсчитать:

$$7 \times 4 \times 12 \times 500 \times 10 \times 50 = 84\,000\,000$$

→ Таким образом, изначально **небольшое** количество признаков превратилось в **несколько миллионов**

→ Дальше больше — если добавить пол (2 значения), то количество признаков станет равным уже **168 000 000**

Комбинация признаков

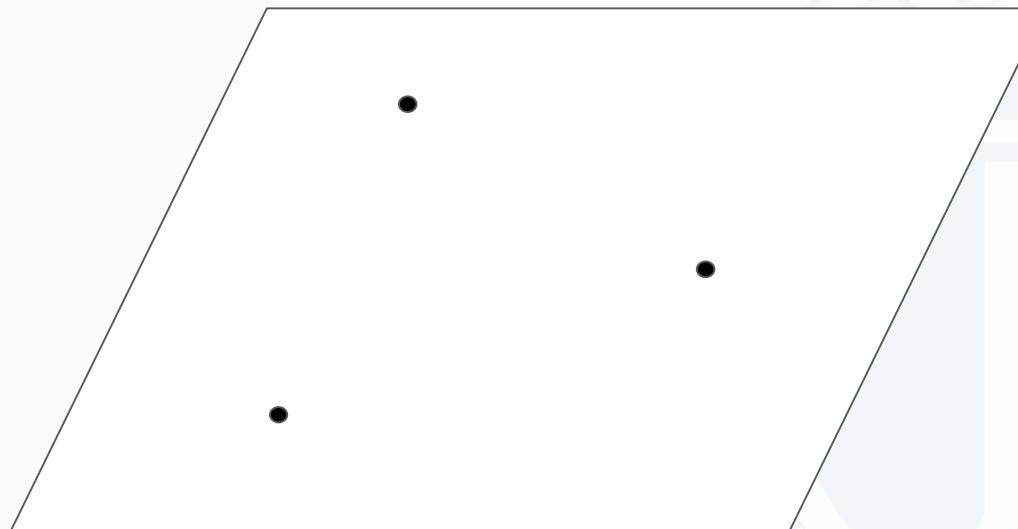


Даже на изначально небольших данных могут помочь техники работы с большими данными

В дождливый понедельник сентября днем на Тверской улице пользователь поехал на Kia Rio	В дождливый вторник сентября днем на Тверской улице пользователь поехал на Kia Rio	...
1	0	...

Основные проблемы при обучении

- В 3-мерном пространстве требуется 3 точки, чтобы провести плоскость
- Аналогично в N-мерном пространстве требуется N точек, чтобы провести гиперплоскость



Основные проблемы при обучении

→ Таким образом, чтобы выявить хотя бы линейную зависимость в N-мерном пространстве нужно не меньше N объектов



Основные проблемы при обучении

→ Таким образом, чтобы выявить хотя бы линейную зависимость в N-мерном пространстве нужно не меньше N объектов

→ Если объектов будет меньше, то слишком высокая размерность «запутает» модель — она может обучаться **нестабильно** и с **плохим качеством**



Основные проблемы при обучении

- Если же в наборе данных больше N объектов, возникает другая проблема — данные перестают помещаться в машине



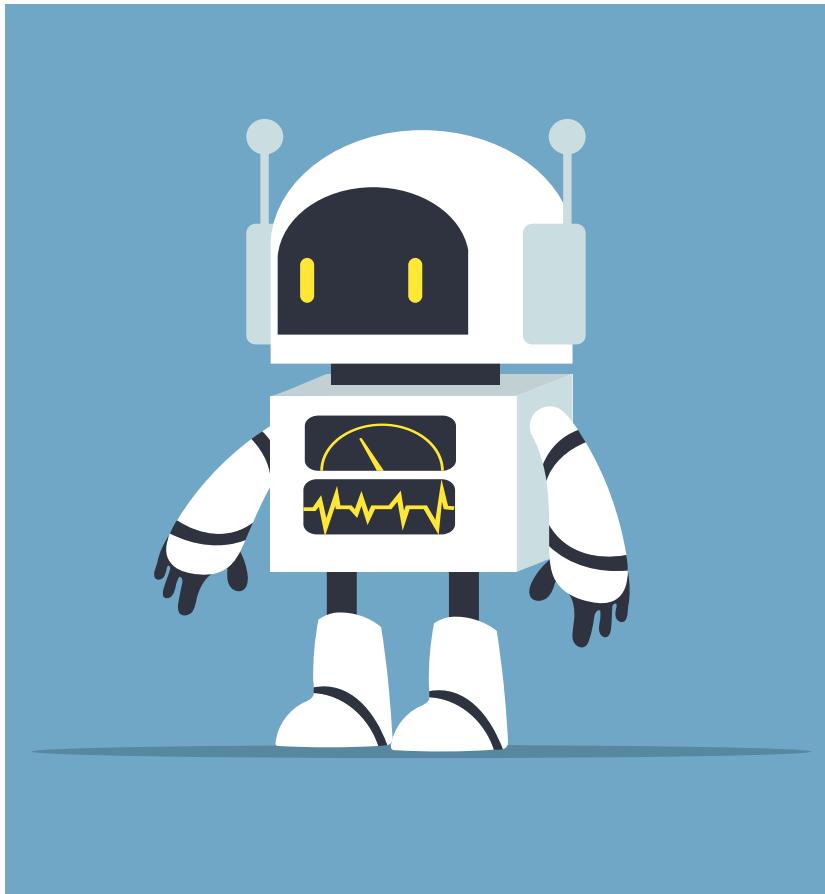
Основные проблемы при обучении

- Если же в наборе данных больше N объектов, возникает другая проблема — данные перестают помещаться в машине
- Предположим, у нас **50 миллионов** признаков и **50 миллионов** объектов. Если один признак будет весить **1 бит**, то на все потребуется минимум **310 Терабайт**

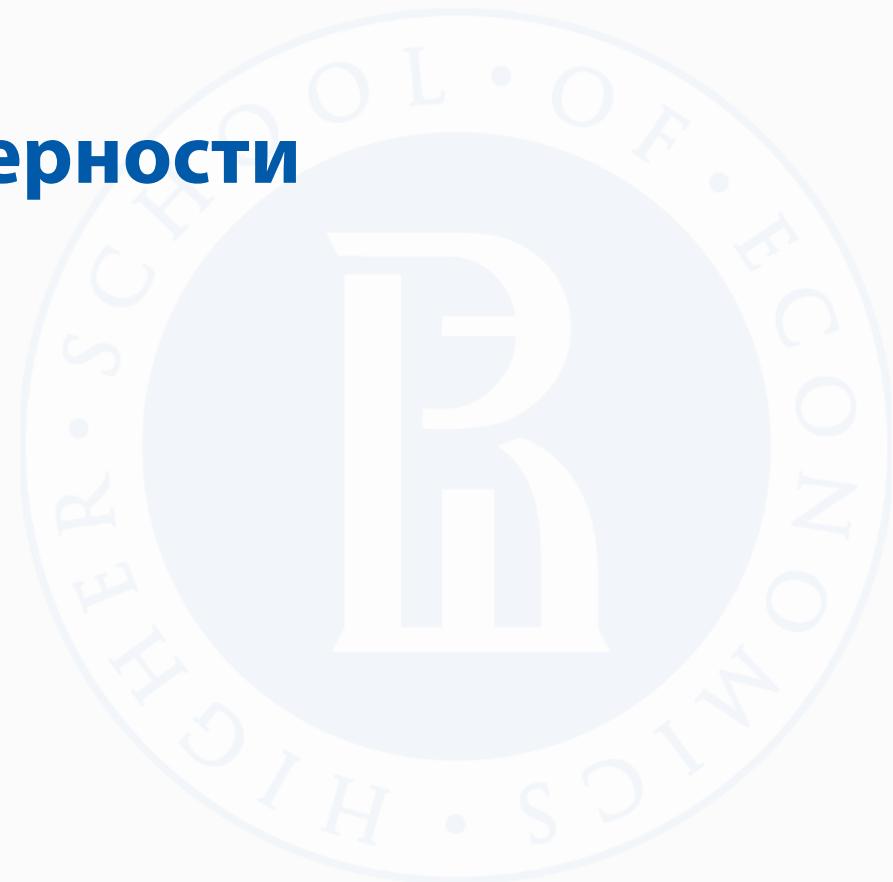


Основные проблемы при обучении

→ Такие размеры **сложно поместить** в одну машину,
а сама модель будет обучаться **невероятно долго**



Снижение размерности



Снижаем размерность

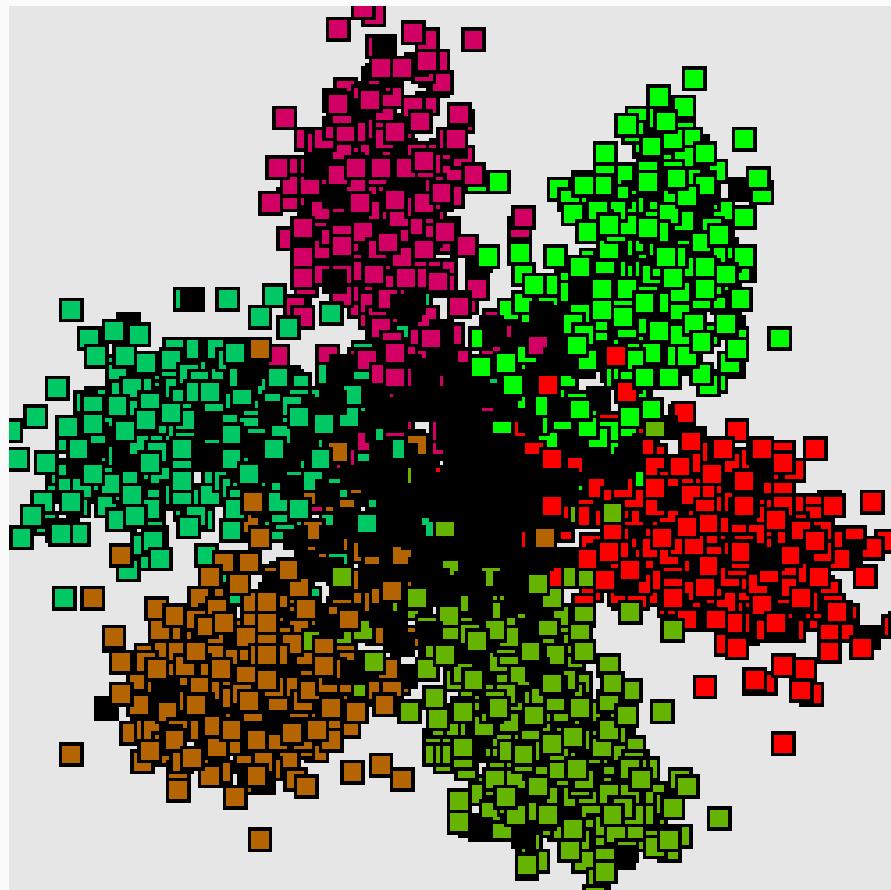
→ Если размерность данных слишком большая, то первая разумная мысль — уменьшить эту размерность!



Метод главных компонент

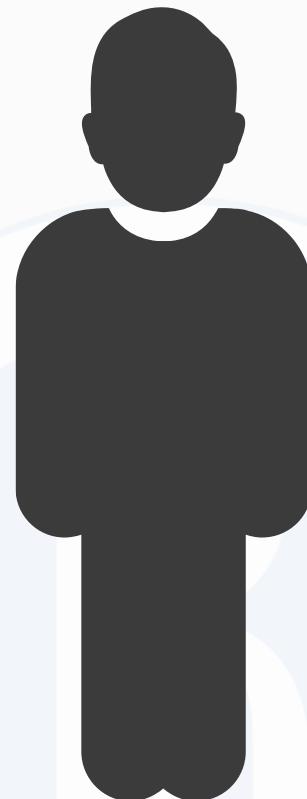


Классическим метод снижения размерности — метод
главных компонент



Метод главных компонент

→ По сути, этот метод старается превратить **изначальные признаки в новые признаки** так, чтобы потерять как можно меньше информации

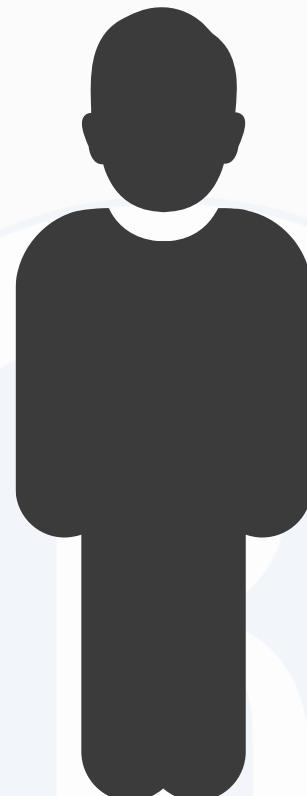


Метод главных компонент

→ По сути, этот метод старается превратить **изначальные признаки в новые признаки** так, чтобы потерять как можно меньше информации

→ Предположим, что мы предсказываем рост человека, и у нас есть признаки:

- Длина левой руки
- Длина правой руки
- Длина левой ноги
- Длина правой ноги



Метод главных компонент

→ Посмотрев на данные, метод может «понять», что длины рук практически не отличаются (как и длины ног)

Метод главных компонент

- Посмотрев на данные, метод может «понять», что длины рук практически не отличаются (как и длины ног)
- Если использовать средние значения, то мы почти ничего не потеряем

Метод главных компонент

- Посмотрев на данные, метод может «понять», что длины рук практически не отличаются (как и длины ног)
- Если использовать средние значения, то мы почти ничего не потеряем
 - Длина правой руки
 - Длина левой руки
 - Длина правой ноги
 - Длина левой ноги

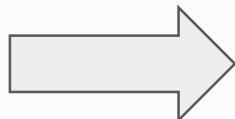


Метод главных компонент

→ Посмотрев на данные, метод может «понять», что длины рук практически не отличаются (как и длины ног)

→ Если использовать средние значения, то мы почти ничего не потеряем

- Длина правой руки
- Длина левой руки
- Длина правой ноги
- Длина левой ноги



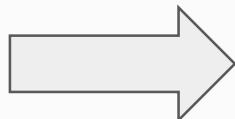
- Полусумма длин правой и левой рук
- Полусумма длин правой и левой ног

Метод главных компонент

→ Посмотрев на данные, метод может «понять», что длины рук практически **не отличаются** (как и длины ног)

→ Если использовать **средние значения**, то мы почти ничего не потеряем

- Длина правой руки
- Длина левой руки
- Длина правой ноги
- Длина левой ноги



- Полусумма длин правой и левой рук
- Полусумма длин правой и левой ног

→ Таким образом, мы уменьшили количество признаков в два раза, не потеряв в качестве!

Метод главных компонент

- В общем случае, метод из [линейных комбинаций](#) оригинальных признаков строит новые признаки

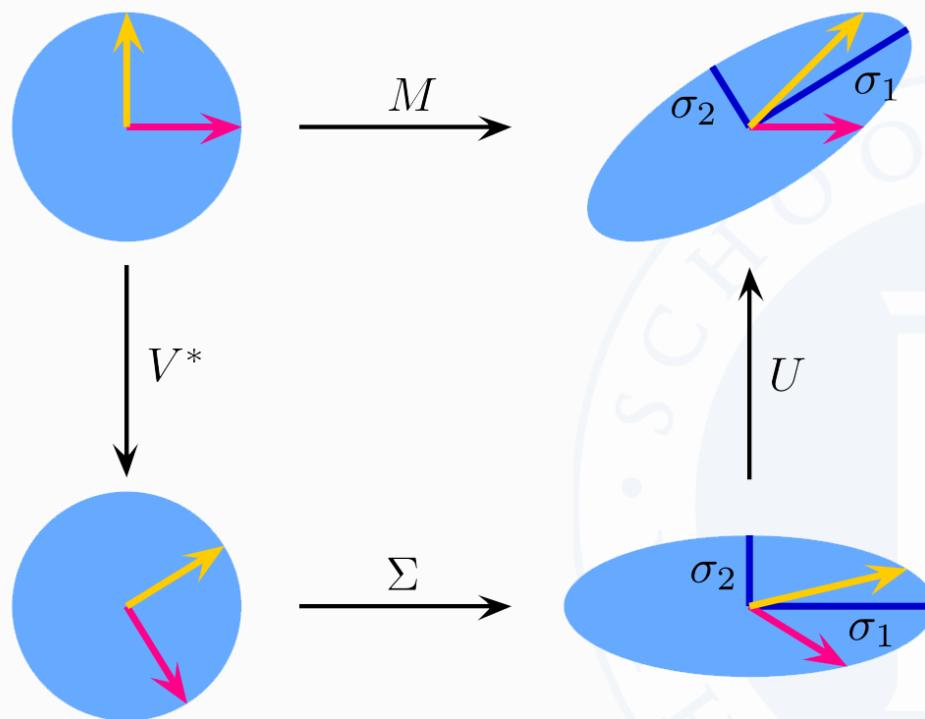
Метод главных компонент

- В общем случае, метод из **линейных комбинаций** оригинальных признаков строит новые признаки
- На реальных данных метод может выдавать **менее очевидные и интерпретируемые** признаки. Например, может оказаться, что самый информативный признак:

$$Z = 0.23 \times \text{Длина левой руки} + 0.56 \times \text{Длина правой руки} - 0.4 \times \text{Длина левой ноги} + 0.68 \times \text{Длина правой ноги}$$

Метод главных компонент

→ Так как все преобразования линейные, то задачу нахождения главных компонент можно задать в матричном виде



$$M = U \cdot \Sigma \cdot V^*$$

Метод главных компонент

→ Это крайне полезное свойство, так как существует много эффективных реализаций матричных методов общего назначения

Метод главных компонент

→ Это крайне полезное свойство, так как существует много эффективных реализаций матричных методов общего назначения



L	A	P	A	C	K
L	-A	P	-A	C	-K
L	A	P	A	-C	-K
L	-A	P	-A	-C	K
L	A	-P	-A	C	K
L	-A	-P	A	C	-K

Метод главных компонент

→ Это крайне полезное свойство, так как существует много эффективных реализаций матричных методов общего назначения



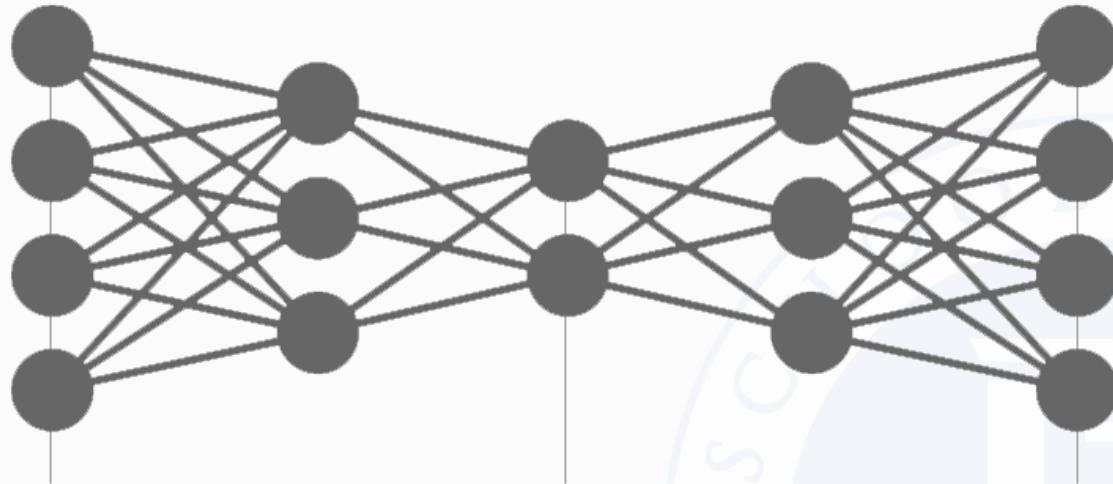
MLlib

L	A	P	A	C	K
L	-A	P	-A	C	-K
L	A	P	A	-C	-K
L	-A	P	-A	-C	K
L	A	-P	-A	C	K
L	-A	-P	A	C	-K

→ Как и специальных алгоритмов для метода главных компонент, таких как: **NIPALS**, **Batch PCA**, **Online PCA**

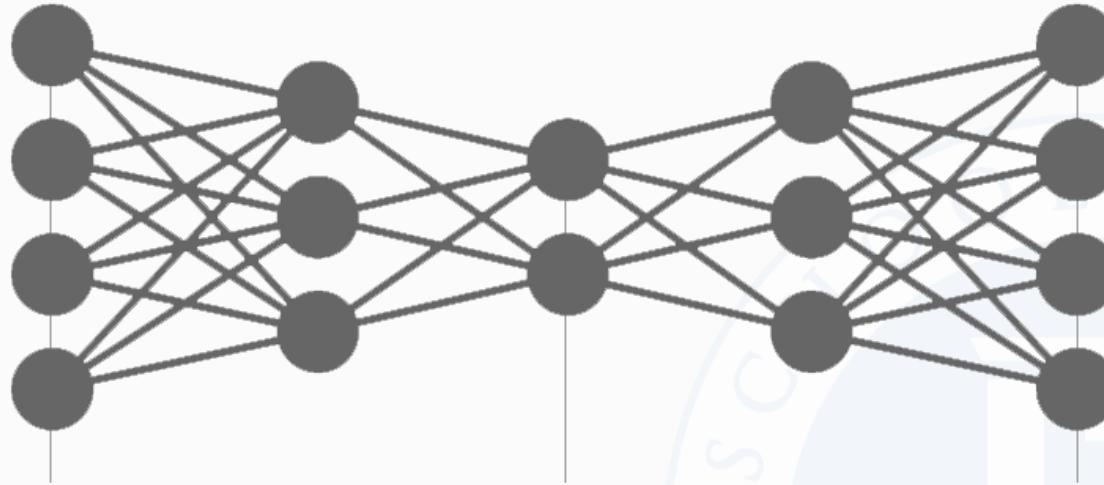
Альтернативные методы сжатия

→ Существуют и более продвинутые методы сжатия:
например, UMAP, t-SNE, Autoencoder



Альтернативные методы сжатия

→ Существуют и более продвинутые методы сжатия:
например, UMAP, t-SNE, Autoencoder



→ Используя нелинейные способы сжатия, они могут давать **лучшее качество**, однако на больших данных их использование может быть **слишком затратным**

Минусы метода главных компонент

Метод кажется рациональным выбором для борьбы с высокой размерностью, но у него есть недостатки:

- Необходимость предподсчета для всего набора данных

Минусы метода главных компонент

Метод кажется рациональным выбором для борьбы с высокой размерностью, но у него есть недостатки:

- Необходимость предподсчета для всего набора данных
- Фиксированное количество изначальных признаков

Хеширование признаков



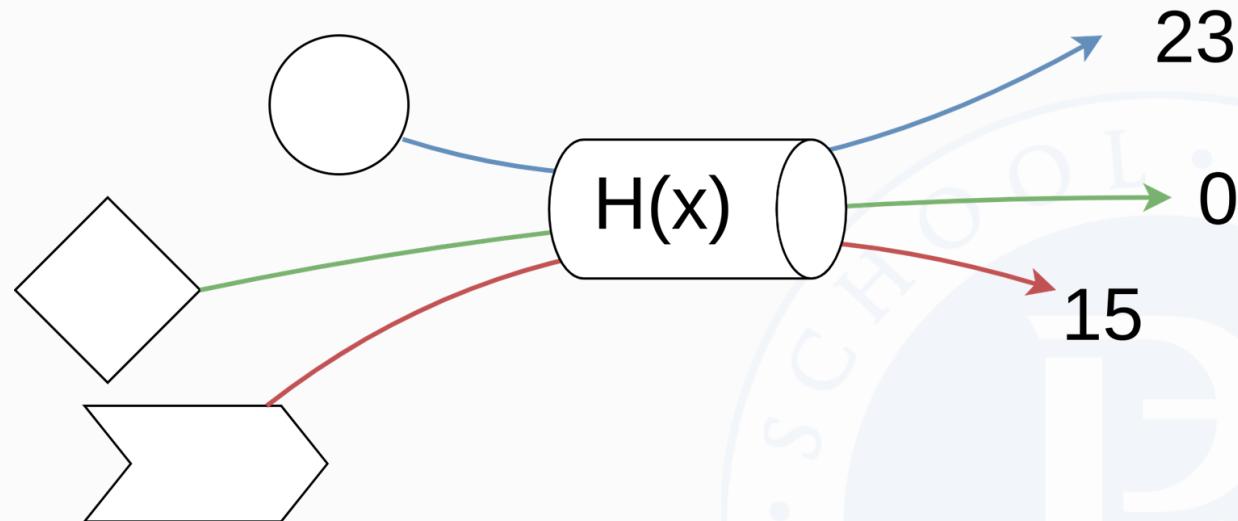
Хеширование признаков

→ Альтернативный подход к борьбе с высокой размерностью — [хеширование признаков](#)



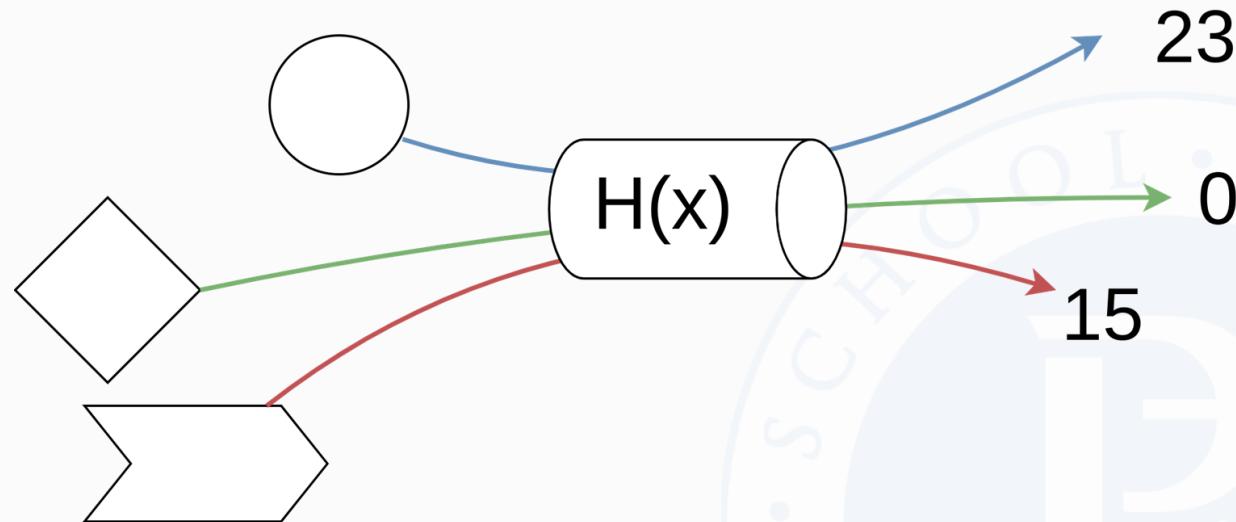
Хеш-функция

→ Хеш-функция — детерминированная функция, которая превращает изначальные объекты в числа от 0 до K



Хеш-функция

→ Хеш-функция — детерминированная функция, которая превращает изначальные объекты в числа от 0 до K

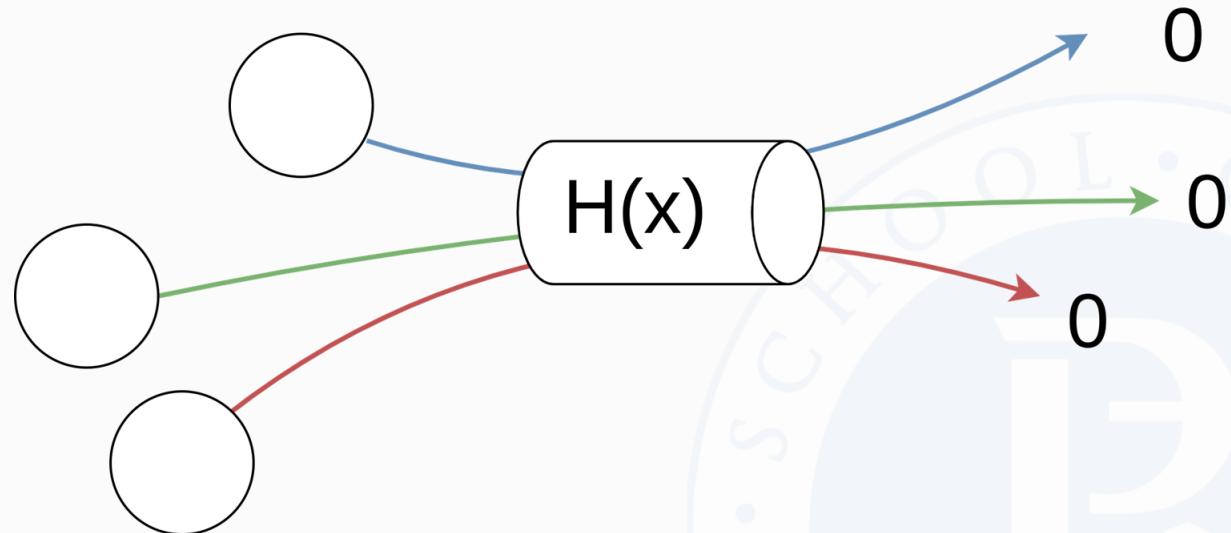


→ Если хеш-функция **хорошая**, то восстановить изначальный объект по значению функции **сложно**

Хеш-функция

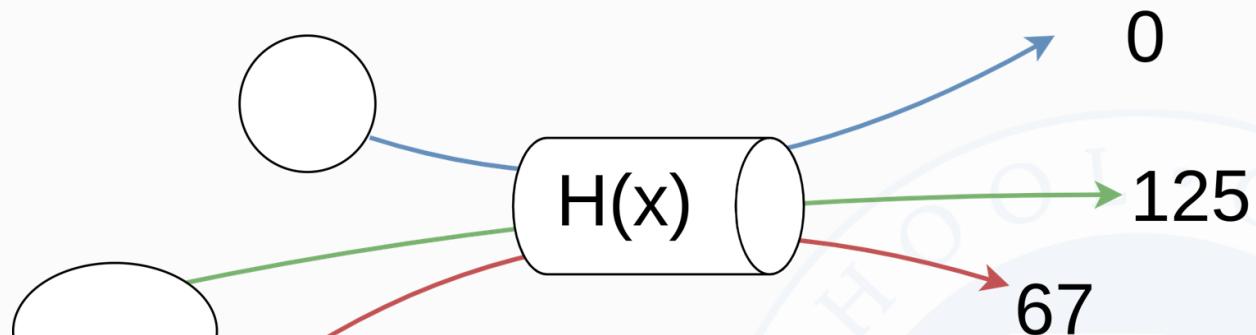


Важно понимать, что функция **не является случайной**
Значение на **одинаковых** объектах всегда будет
одинаковым



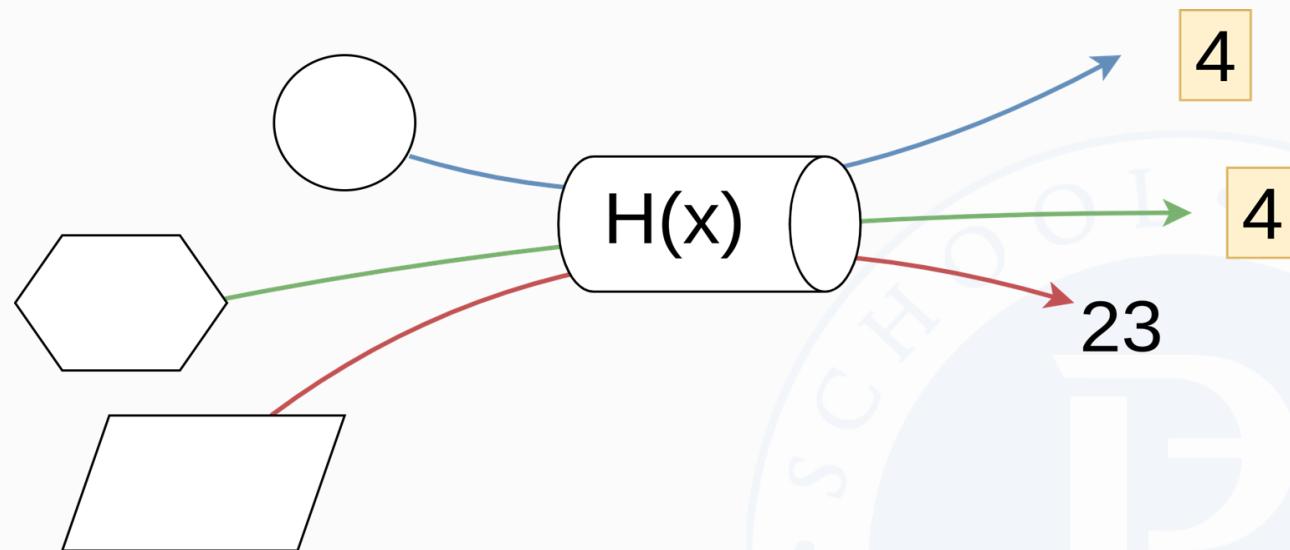
Хеш-функция

→ Однако на очень **близких** изначальных объектах она может давать совершенно **разные ответы**



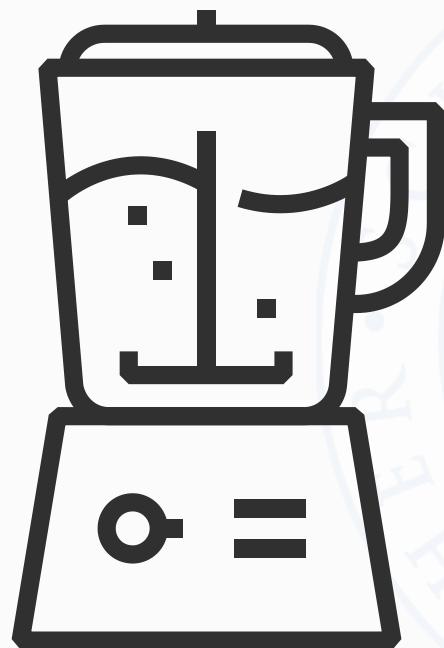
Хеш-функция

→ Так как значения функции ограничены (всего K возможных значений), то могут происходить **коллизии** — ситуации, когда значения на **разных** объектах **совпадают**



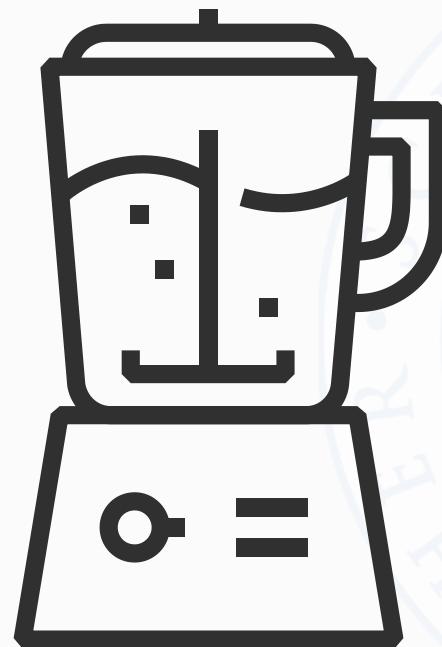
Хеш-функция

→ Чтобы реализовать хеш-функцию, нужно придумать любую функцию из объектов в числа и брать значения по модулю K



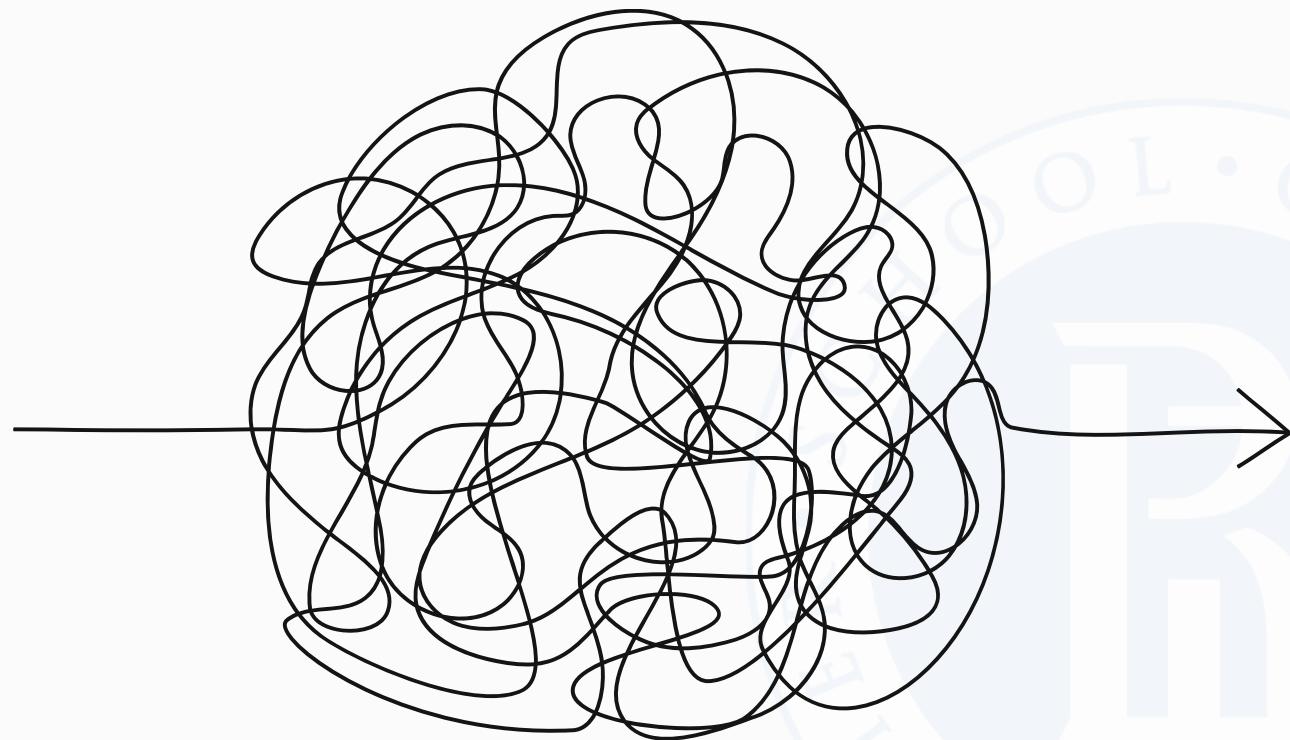
Хеш-функция

- Чтобы реализовать хеш-функцию, нужно придумать любую функцию из объектов в числа и брать значения по модулю K
- Хорошая функция — та, которая хорошо «перемешивает» входные данные



Хеш-функция

→ Чтобы хеш-функция стала **хорошой**, нужно постараться добавить много **сложных нелинейных операторов**



Хеш-функция

→ Пример для чисел

$$H(x) = [2^{(5x + 2x^2 + x)} * \sin(x)] \bmod K$$

Хеш-функция



Пример для чисел

$$H(x) = [2^{(5x + 2x^2 + x)} * \sin(x)] \bmod K$$



Пример для строк

$$H(x) = [\text{len}(x) * (\#x_1 + \#x_2^2 + \#x_3^3 + \dots)] \bmod K$$

$\#x_i$ = номер буквы на позиции i в алфавите

Хеш-функция

→ Долго думать над функцией не обязательно. Математика показывает, что **полиномиальная** хеш-функция уже очень хорошо работает

$$H(x) = [h_1 * x + h_2 * x^2 + \dots + h_n * x^n] \bmod K$$

Хеш-функция

→ Долго думать над функцией не обязательно. Математика показывает, что **полиномиальная** хеш-функция уже очень хорошо работает

$$H(x) = [h_1 * x + h_2 * x^2 + \dots + h_n * x^n] \bmod K$$

→ Более того, уже разработаны хорошие хеш-функции, которые используют в индустрии. Например:

- MD5
- SHA-3
- MurMurHash
- HMAC

Хеширование признаков

→ Хеш-функцию можно использовать для того, чтобы **снизить размерность** признакового пространства

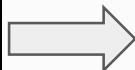
Хеширование признаков

- Хеш-функцию можно использовать для того, чтобы **снизить размерность** признакового пространства
- Для этого название **изначального** признака пропустим через хеш-функцию. Полученное число — **новое название** признака. **Значение** этого признака просто **копируем**

Хеширование признаков

- Хеш-функцию можно использовать для того, чтобы **снизить размерность** признакового пространства
- Для этого название **изначального** признака пропустим через хеш-функцию. Полученное число — **новое название** признака. **Значение** этого признака просто **копируем**

Возраст	M	Ж	...
19	1	0	...
24	0	1	...



72	23	15	...
19	1	0	...
24	0	1	...

$$H(\text{Возраст}) = 72, H(M) = 23, H(Ж) = 15$$

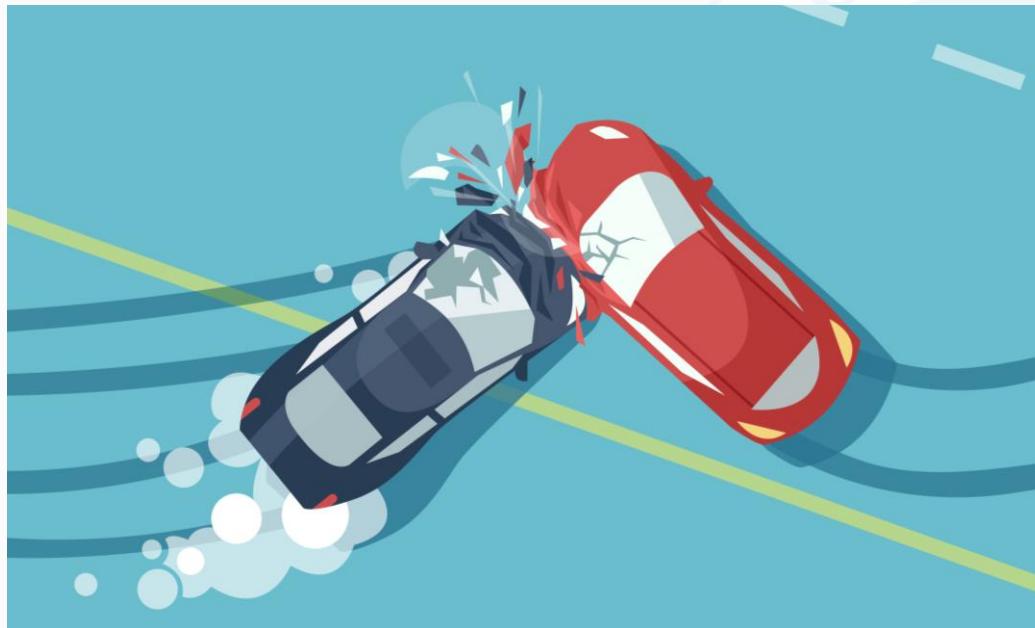
Хеширование признаков

→ Так как хеш-функция выдает числа только от 0 до K, то после преобразования количество признаков станет равным K

Хеширование признаков

→ Так как хеш-функция выдает числа только от 0 до K, то после преобразования количество признаков станет равным K

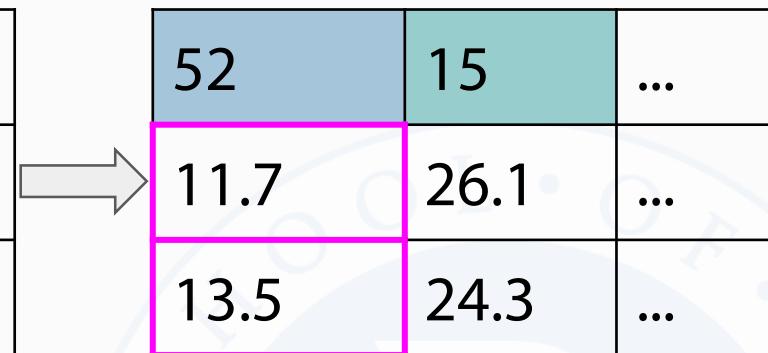
→ Однако из-за коллизий какие-то изначальные признаки могут начать конфликтовать, попав в один новый



Хеширование признаков

→ В случае коллизии мы просто выставляем для признака последнее подсчитанное значение

Нога	Рука	Голова	...
52.6	26.1	11.7	...
50.1	24.3	13.5	...



The diagram illustrates a transformation of a 3x4 matrix into a 3x3 matrix. An arrow points from the original matrix on the left to the transformed matrix on the right. In the original matrix, the third column contains values 11.7, 13.5, and 11.7 respectively. These values are highlighted with a magenta border. In the transformed matrix, the first two columns remain the same, but the third column contains the values 15, 26.1, and 24.3 respectively, indicating that the last calculated value (11.7) has been stored over the previous values.

52	15	...
11.7	26.1	...
13.5	24.3	...

$$H(\text{Нога}) = 52, H(\text{Голова}) = 52, H(\text{Рука}) = 15$$

Хеширование признаков



Помимо высокой скорости расчета, метод позволяет экономить на размере данных

Хеширование признаков

- Помимо высокой скорости расчета, метод позволяет экономить на размере данных
- Для объекта необходимо знать **только ненулевые** признаки, чтобы подсчитать новые признаки. Хранить нули для остальных не требуется

Хеширование признаков

- Помимо высокой скорости расчета, метод позволяет экономить на размере данных
- Для объекта необходимо знать **только ненулевые** признаки, чтобы подсчитать новые признаки.
Хранить нули для остальных не требуется

Пол: М
Возраст: 19
Глаза: Синие

$H(\text{Пол_M}) = 64$
 $H(\text{Возраст}) = 1$
 $H(\text{Глаза_Синие}) = 9$

0	1	...	64
0	19	...	1

Хеширование признаков

На первый взгляд кажется, что такое перемешивание признаков должно сделать модель **непригодной** для работы, но практика показывает, что:

- Коллизии **нулевых** признаков **не важны**, поэтому на **разреженных** данных работает хорошо

Хеширование признаков

На первый взгляд кажется, что такое перемешивание признаков должно сделать модель **непригодной** для работы, но практика показывает, что:

- Коллизии **нулевых** признаков **не важны**, поэтому на **разреженных** данных работает хорошо

- Скорость вычисления позволяет делать больше эпох обучения, что частично **компенсирует** перемешивание

Хеширование признаков

На первый взгляд кажется, что такое перемешивание признаков должно сделать модель **непригодной** для работы, но практика показывает, что:

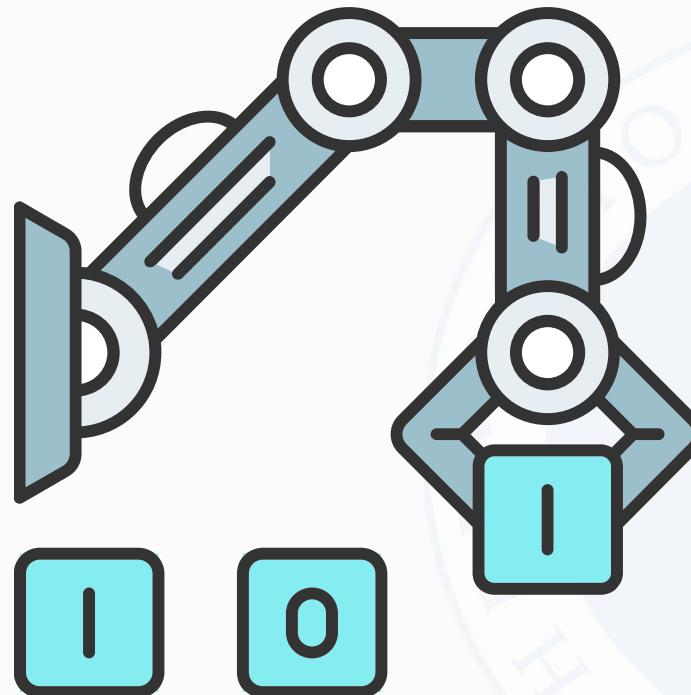
- Коллизии **нулевых** признаков **не важны**, поэтому на **разреженных** данных работает хорошо
- Скорость вычисления позволяет делать больше эпох обучения, что частично **компенсирует** перемешивание
- “Done is better than perfect!”
Обученная таким образом модель лучше, чем модель, которую **не удалось обучить** вовсе

Нативная работа с категориальными признаками



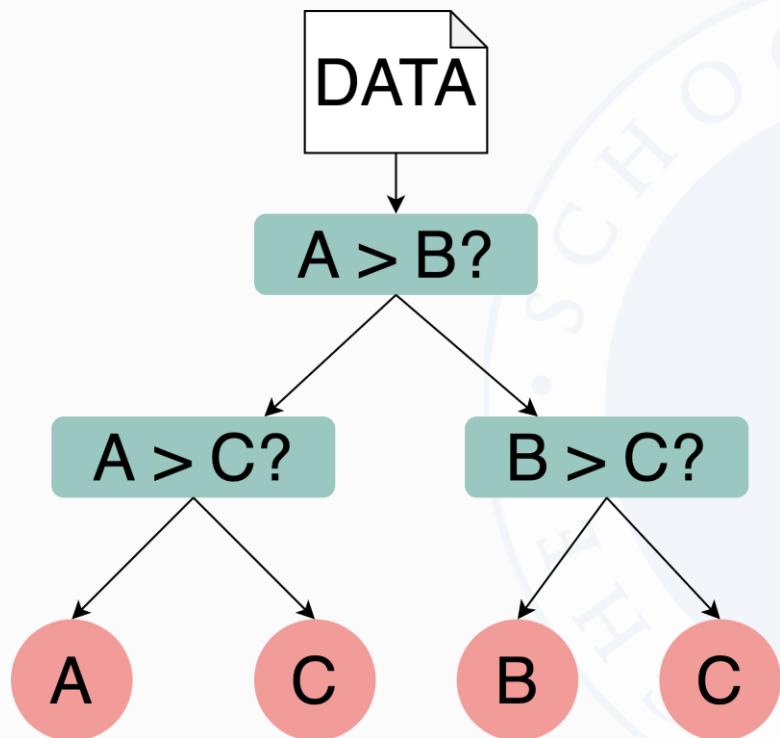
Работа с категориями

→ Если причина высокой размерности — категориальные признаки, то можно использовать модели, которые эффективно умеют обращаться с такими признаками



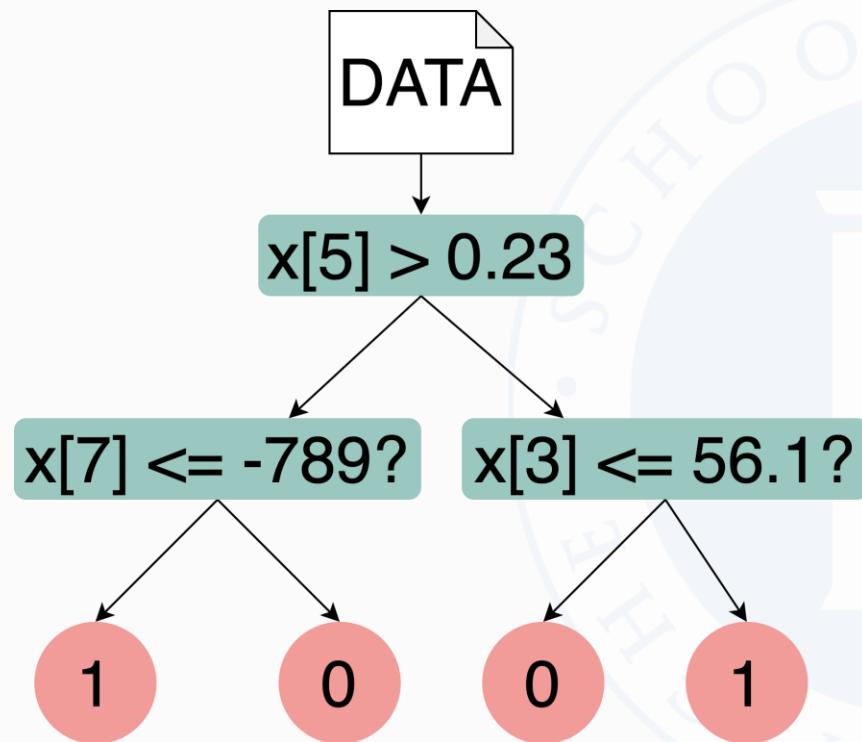
Деревья решений

→ Пример такой модели — деревья решений



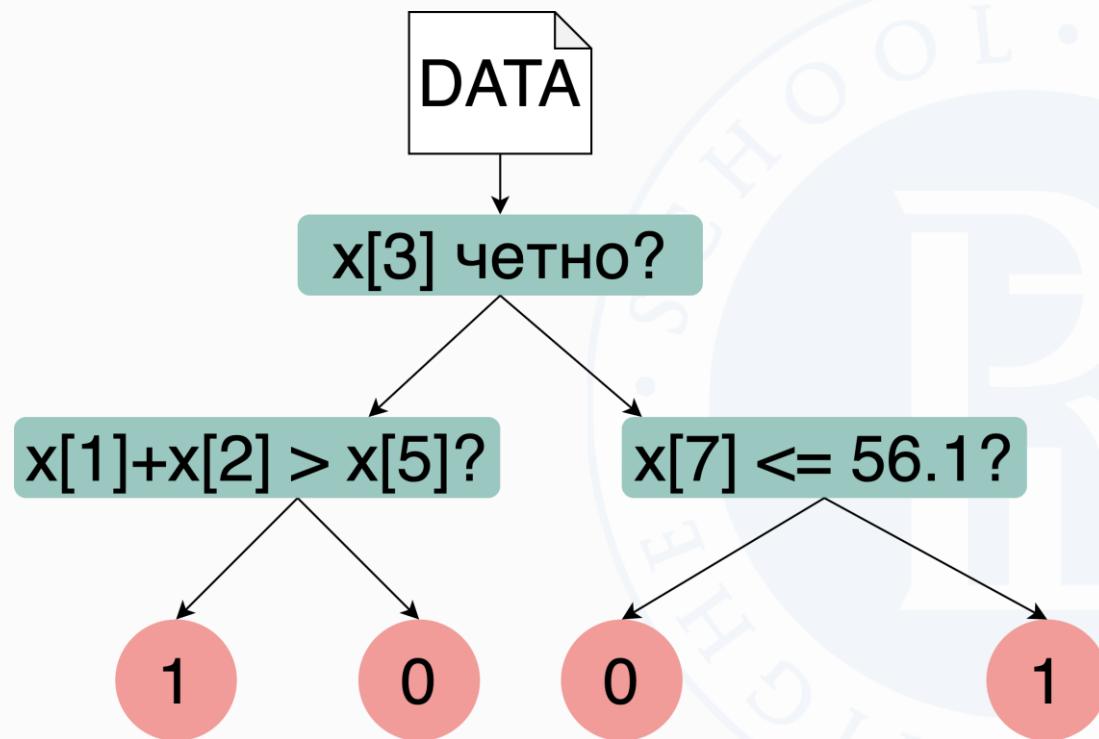
Деревья решений

→ Для задания дерева необходимо в узлы поместить предикаты — вопросы относительно каких-то признаков



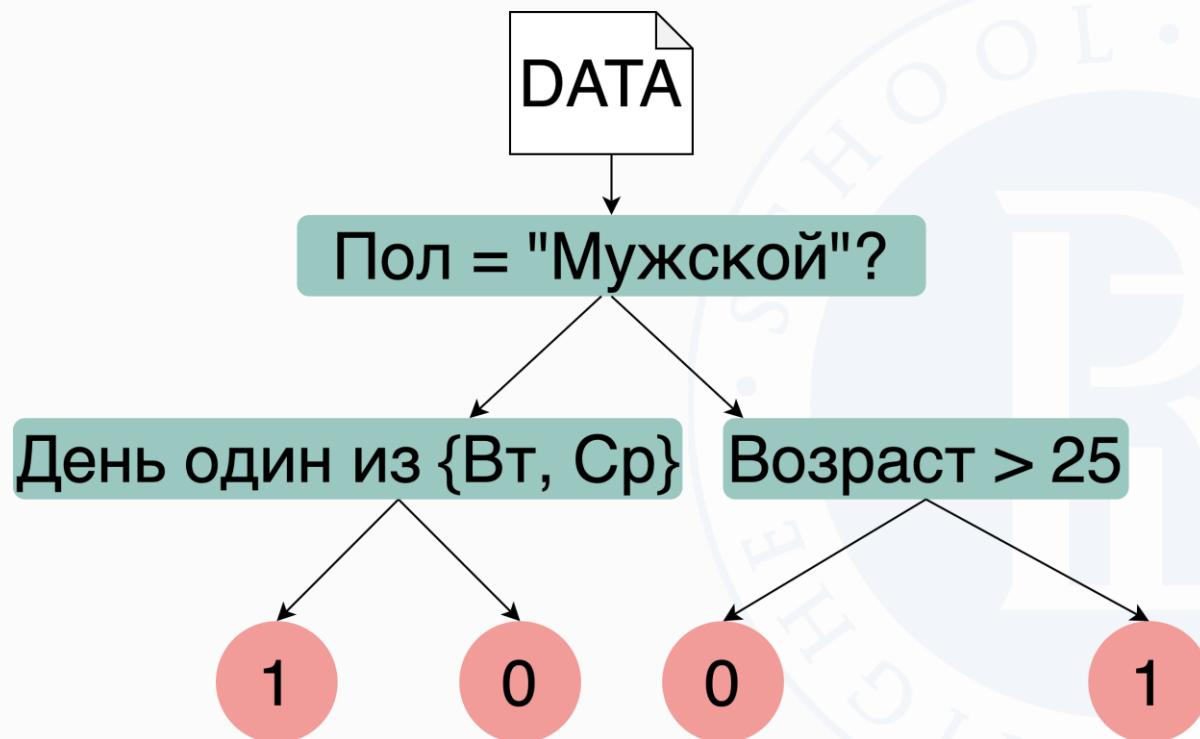
Деревья решений

- Обычно предикат — это вопрос «больше-меньше» про какой-то **численный** признак
- Однако нас **никто не ограничивает** и мы можем придумать свои предикаты



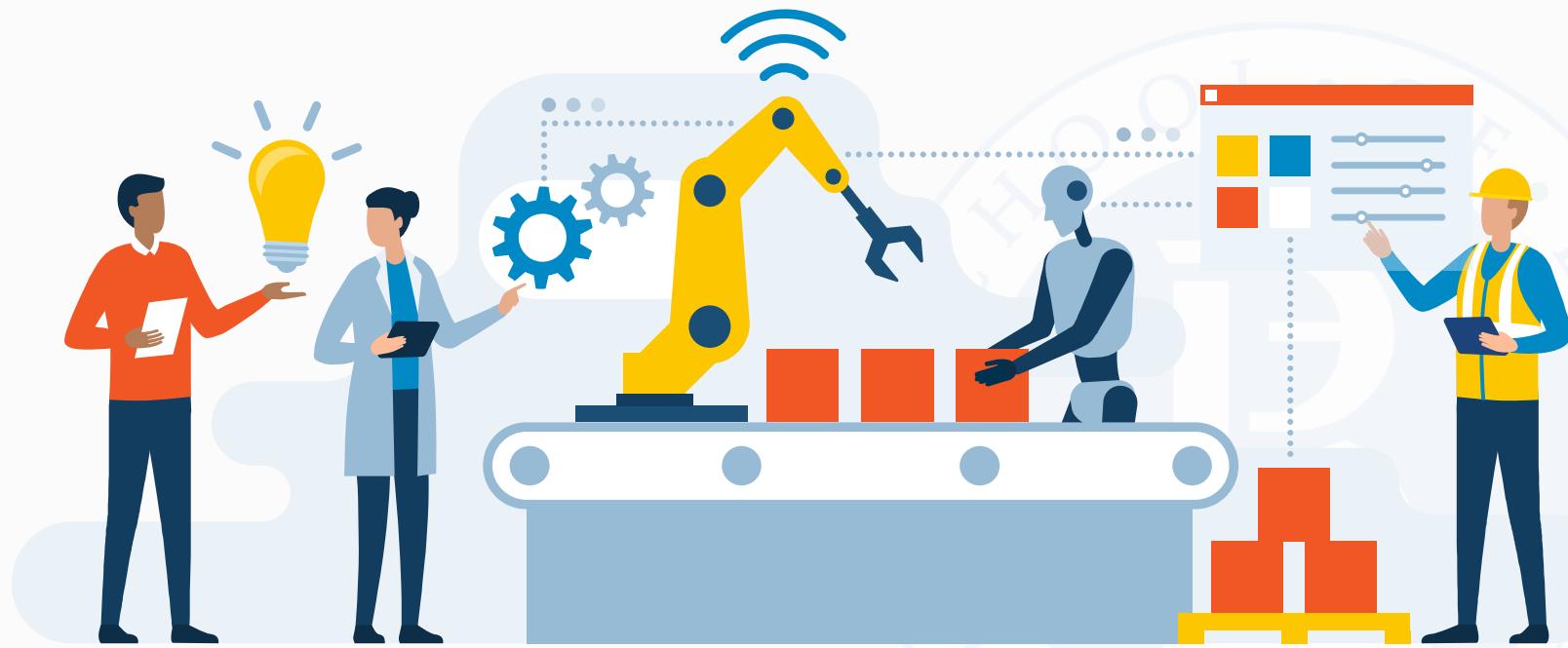
Деревья решений

→ Таким образом мы сразу можем задавать предикаты про **конкретные значения** категориальных признаков, не превращая их искусственно в численные признаки



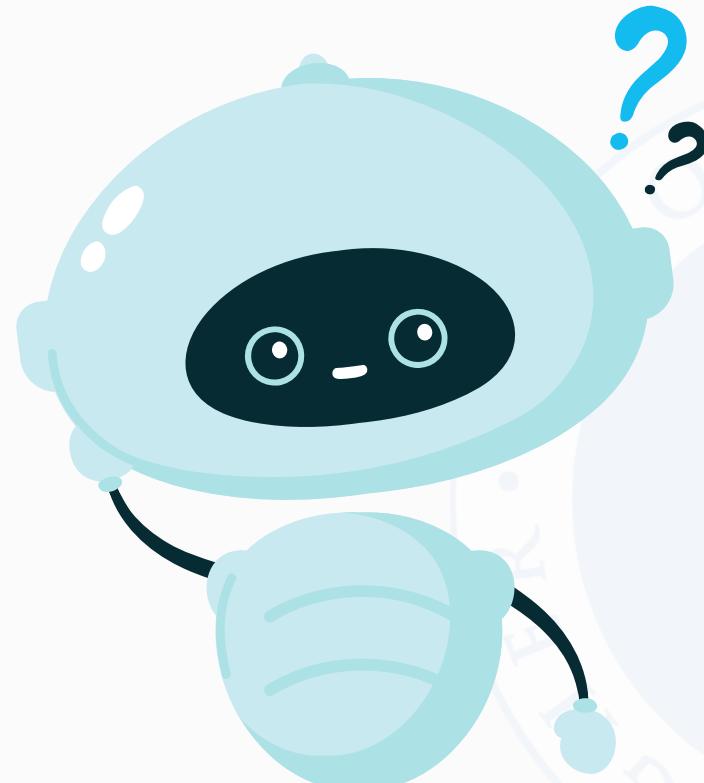
Деревья решений

→ Модель **нативно** работает с разными типами признаков,
не нужно тратить время и память на перекодирование
изначальных объектов



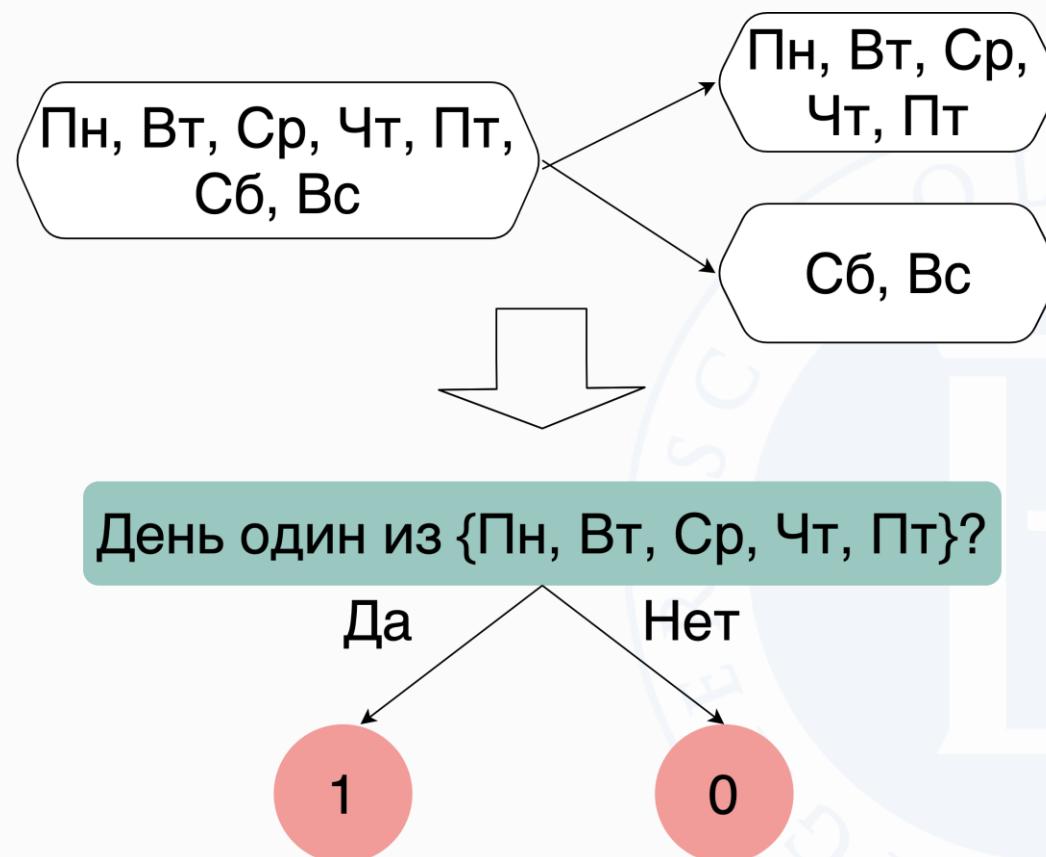
Деревья решений

→ Однако более сложные предикаты могут замедлить процесс обучения. Можно ли эффективно обучать такие деревья?



Подсчет статистики

→ Для задания предиката необходимо разбить возможные значения признака на две части — для правого и левого поддеревьев



Подсчет статистики

- Если подсчитать **статистику** о распределении ответов для признака, то вместо **всех 2^N** разбиений можно перебрать всего **N**
- Пример: хотим предсказать, будет ли пользователь работать в определенный день. У нас есть признак «день недели», в котором 7 значений



Подсчет статистики

→ По данным посчитаем, какой процент дней пользователь работал, для каждого дня

Пн	60,4%
Вт	90,5%
Ср	96,7%
Чт	89,9%
Пт	95,4%
Сб	1,3%
Вс	10,2%

Подсчет статистики

→ Видно, что со вторника по пятницу пользователь почти всегда работает

Пн	60,4%
Вт	90,5%
Ср	96,7%
Чт	89,9%
Пт	95,4%
Сб	1,3%
Вс	10,2%

Подсчет статистики

→ При этом в субботу и воскресение он преимущественно отдыхает

Пн	60,4%
Вт	90,5%
Ср	96,7%
Чт	89,9%
Пт	95,4%
Сб	1,3%
Вс	10,2%

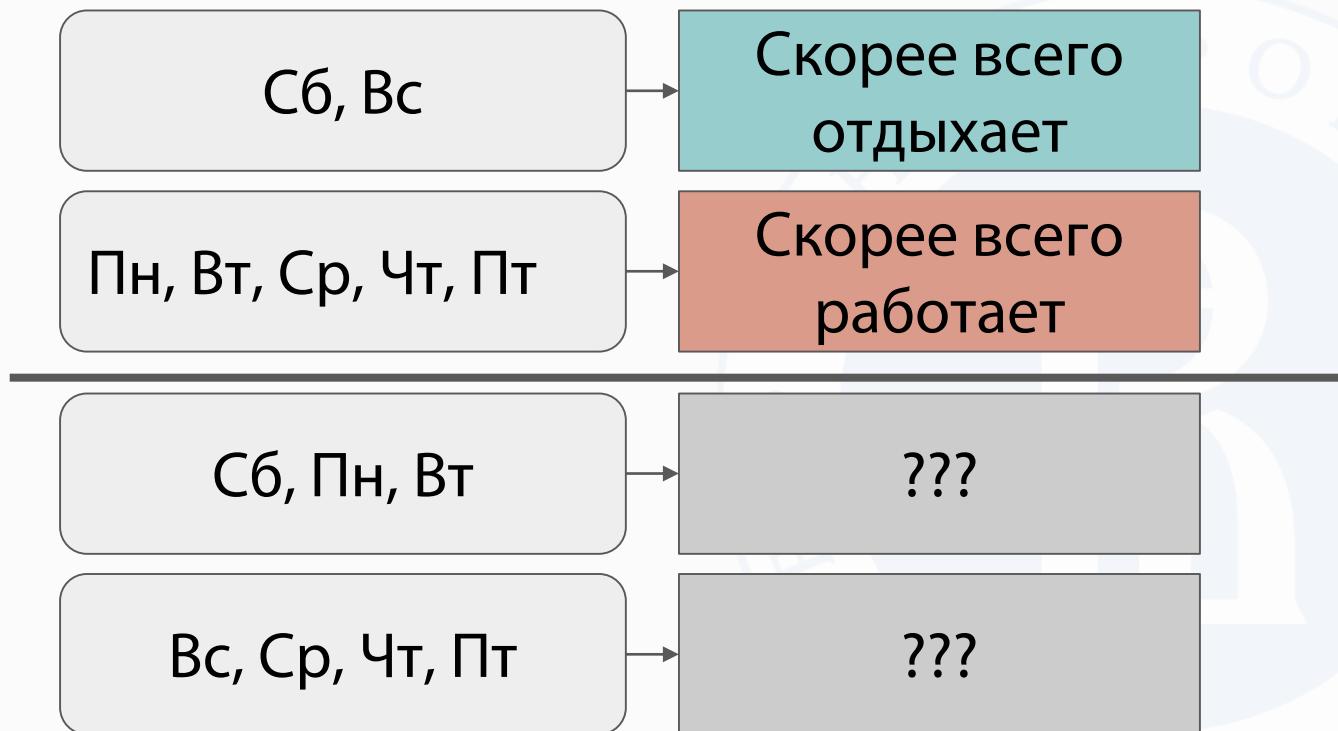
Подсчет статистики

→ Про понедельник не совсем понятно — иногда работает, иногда отдыхает

Пн	60,4%
Вт	90,5%
Ср	96,7%
Чт	89,9%
Пт	95,4%
Сб	1,3%
Вс	10,2%

Подсчет статистики

- Интуитивно понятно, что лучше всего сгруппировать дни таким образом — {Сб, Вс}, {Пн, Вт, Ср, Чт, Пт}
- Принадлежность к какой-то из этих групп **даст больше информации**, чем, например, {Сб, Пн, Вт}, {Вс, Ср, Чт, Пт}



Подсчет статистики



Формализуем нашу интуицию — отсортируем наши признаки по значению статистики

Сб	Вс	Пн	Чт	Вт	Пт	Ср
1,3%	10,2%	60,4%	89,9%	90,5%	95,4%	96,7%

Подсчет статистики



Формализуем нашу интуицию — отсортируем наши признаки по значению статистики

Сб	Вс	Пн	Чт	Вт	Пт	Ср
1,3%	10,2%	60,4%	89,9%	90,5%	95,4%	96,7%



Осталось разделить его на правую и левую части для оптимального разбиения

Подсчет статистики



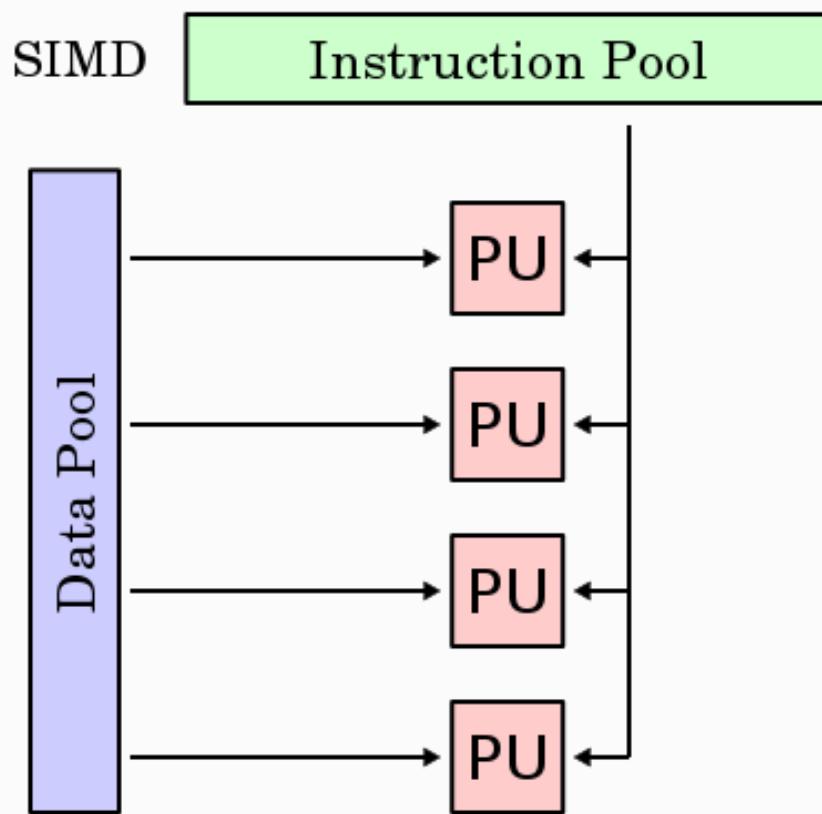
Формализуем нашу интуицию — [отсортируем](#) наши признаки по значению статистики

Сб	Вс	Пн	Чт	Вт	Пт	Ср
1,3%	10,2%	60,4%	89,9%	90,5%	95,4%	96,7%

→ Осталось разделить его на правую и левую части для оптимального разбиения. Для этого нужно проверить всего [6](#) вариантов вместо [128](#)

Подсчет статистики

→ Подсчет статистик — несложная задача, которая может быть эффективно решена как на одиночной машине, так и на целом кластере



Подсчет статистики

→ Таким образом, грамотное использование деревьев позволяет справиться с проблемами размерности для категориальных признаков



Проблема высоких данных



Где можно встретить высокие данные?



Проблема высоких данных — **количество** самих наблюдений слишком велико

«Ширина» данных

«Высота» данных

Ответ	Признак 1	Признак 2	...
1	1.25	2	...
1	0.53	1	...
0	0.34	1	...
1	1.34	3	...
...

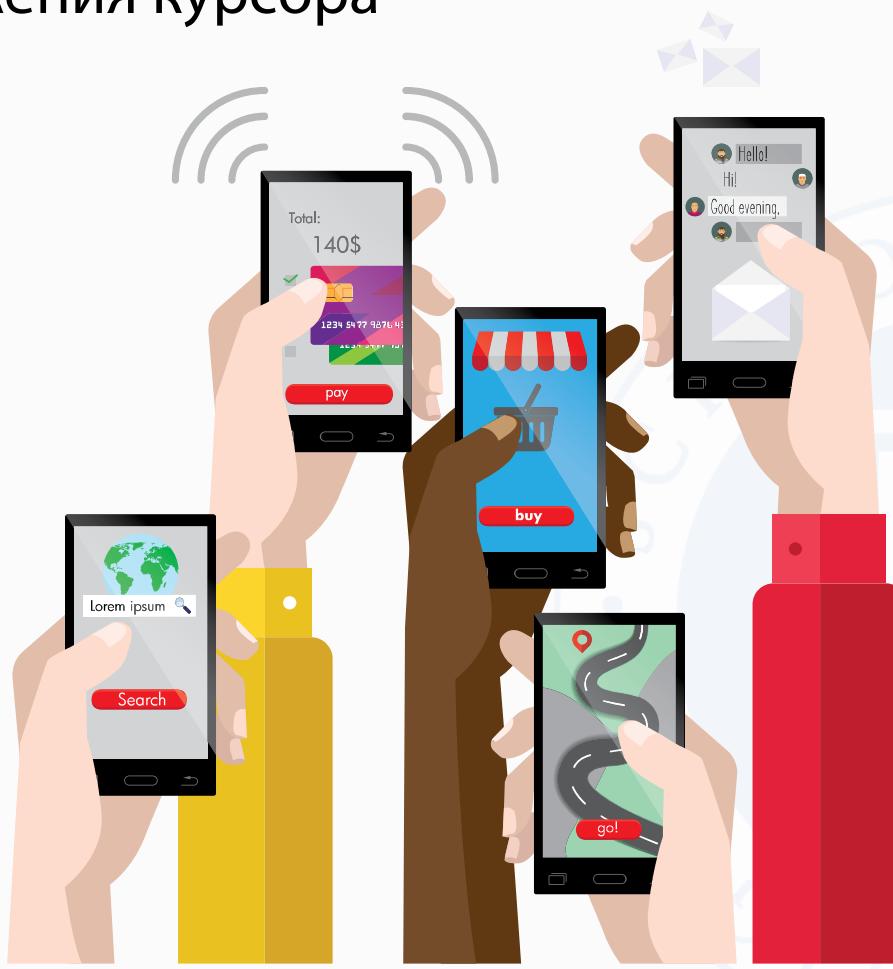
Логирование событий

→ Различные системы способны генерировать огромное количество событий, которые **сохраняются в общий журнал событий**



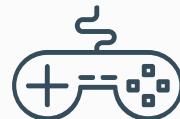
Логирование событий

→ Все современные интернет-сервисы собирают информацию по действиям пользователя. Переходы, клики, даже движения курсора



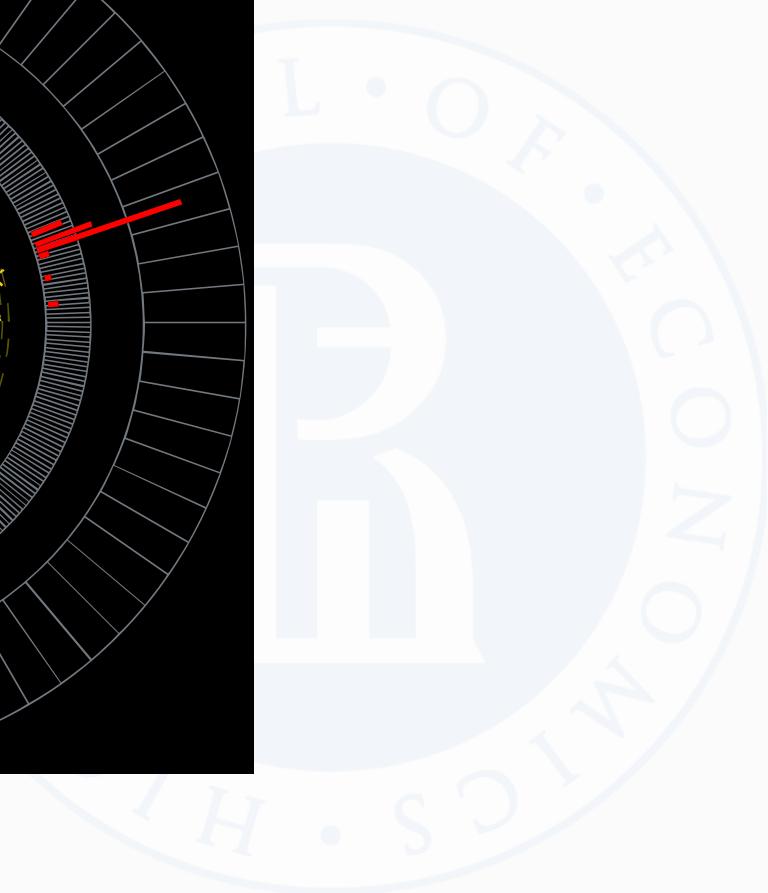
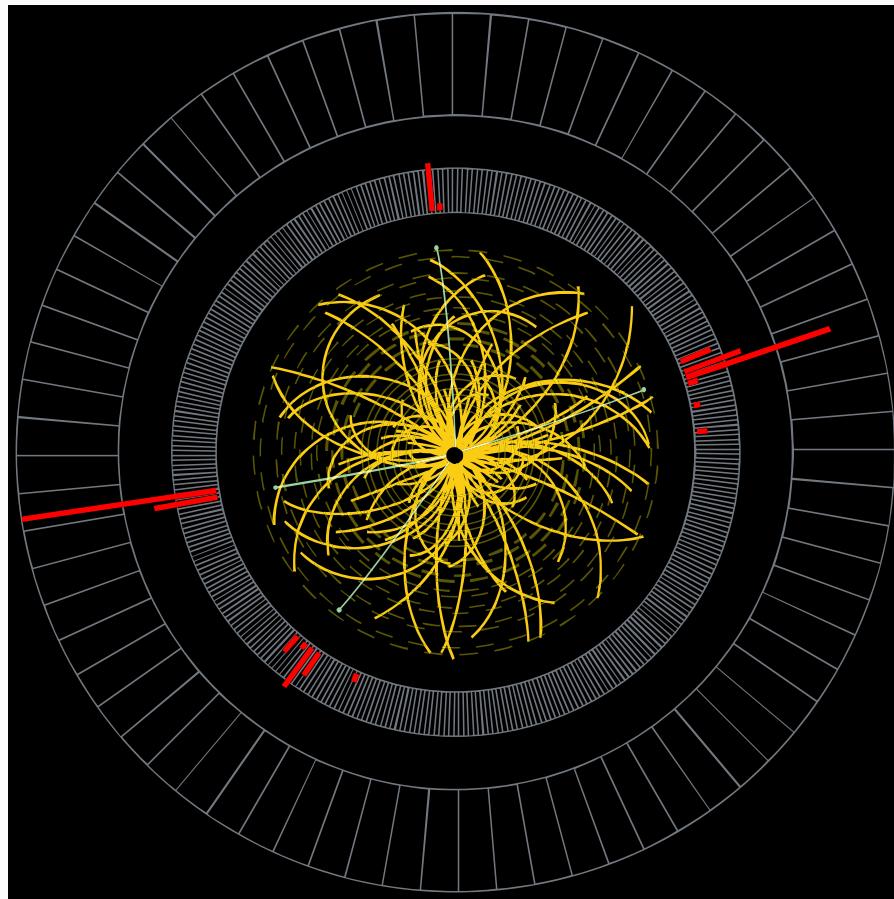
Логирование событий

→ Различные **датчики** могут генерировать свои показания с большой частотой. Метеорологические приборы, приборы на производстве, «умные» устройства, пропускные системы и так далее



Логирование событий

→ По оценкам, большой адронный коллайдер на пике может генерировать [до 1 Петабайта данных в секунду](#)



Аугментация

→ Не только естественные причины могут стать причиной огромного числа данных. Мы можем **искусственно генерировать данные** для улучшения качества модели



Аугментация

→ Например, у нас есть фотографии с дорогами в городе. Фотографии можно изменять так, что **результаты** будут **также пригодны для обучения**. Таким образом мы можем улучшить качество модели



Аугментация



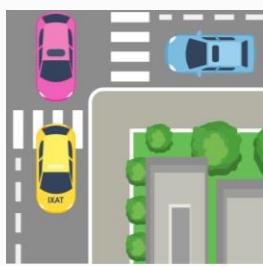
Примеры таких изменений — **повороты и отражения**

0

90

180

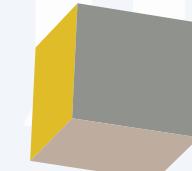
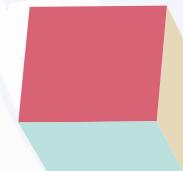
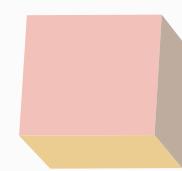
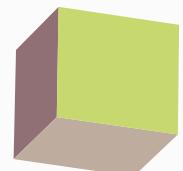
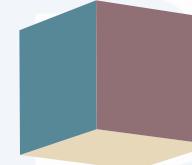
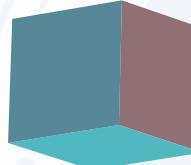
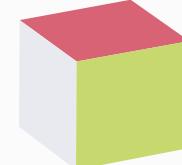
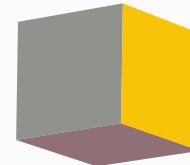
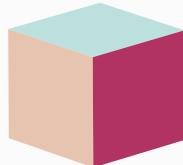
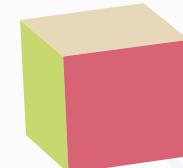
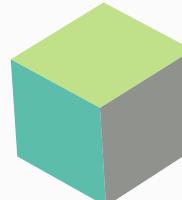
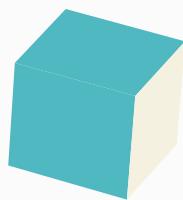
270



Аугментация



Имея в арсенале только 4 поворота и 3 отражения,
мы из одной картинки можем сделать 12. Причем все
результаты будут **новыми** и не будут повторяться



Аугментация

→ Помимо картинок, можно аугментировать [текст](#), [звук](#) и другие подобные «однородные» данные



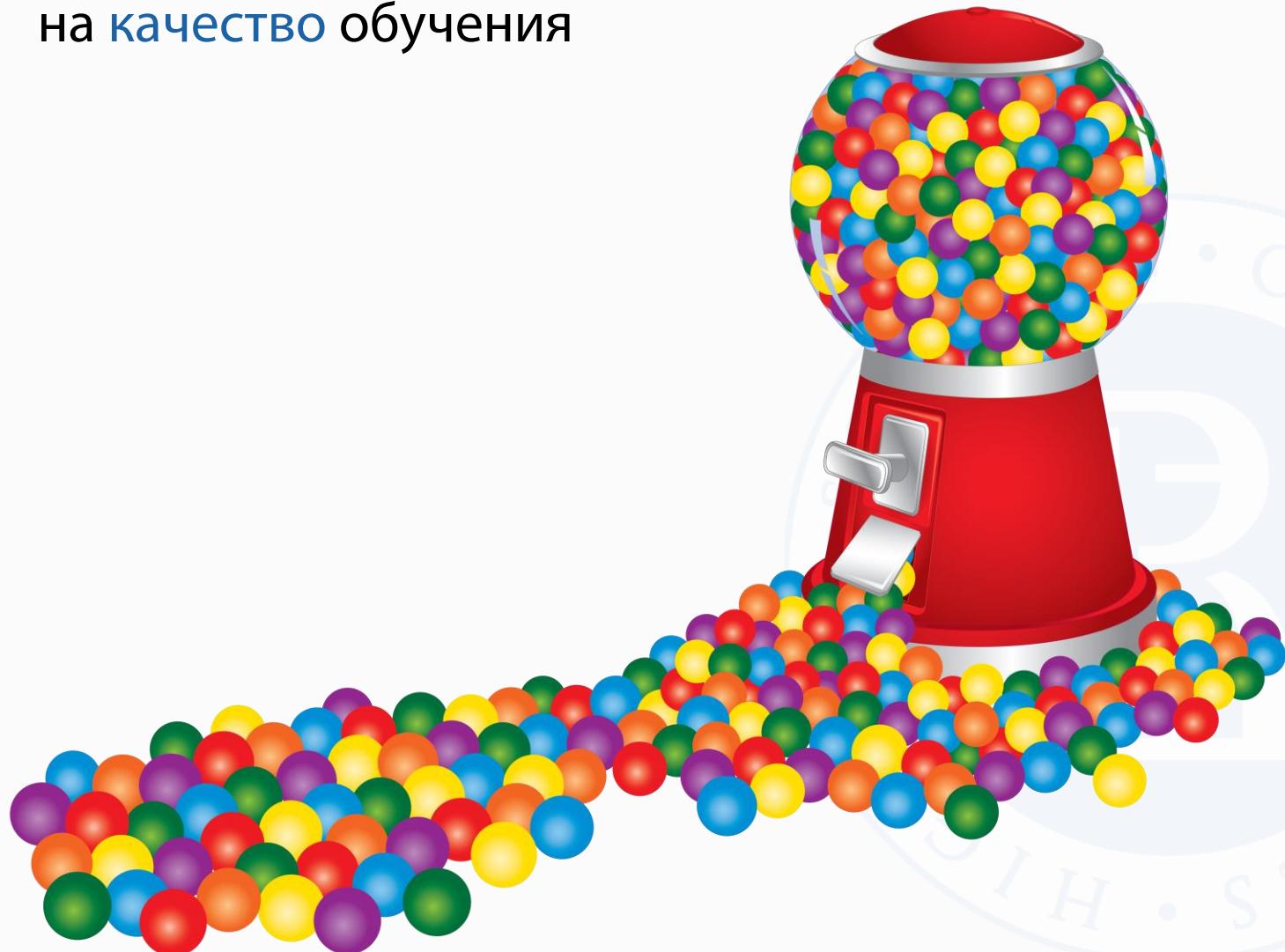
Аугментация

- Помимо картинок, можно аугментировать текст, звук и другие подобные «однородные» данные
- В современных подходах комбинации операций могут увеличивать исходные данные в тысячи раз



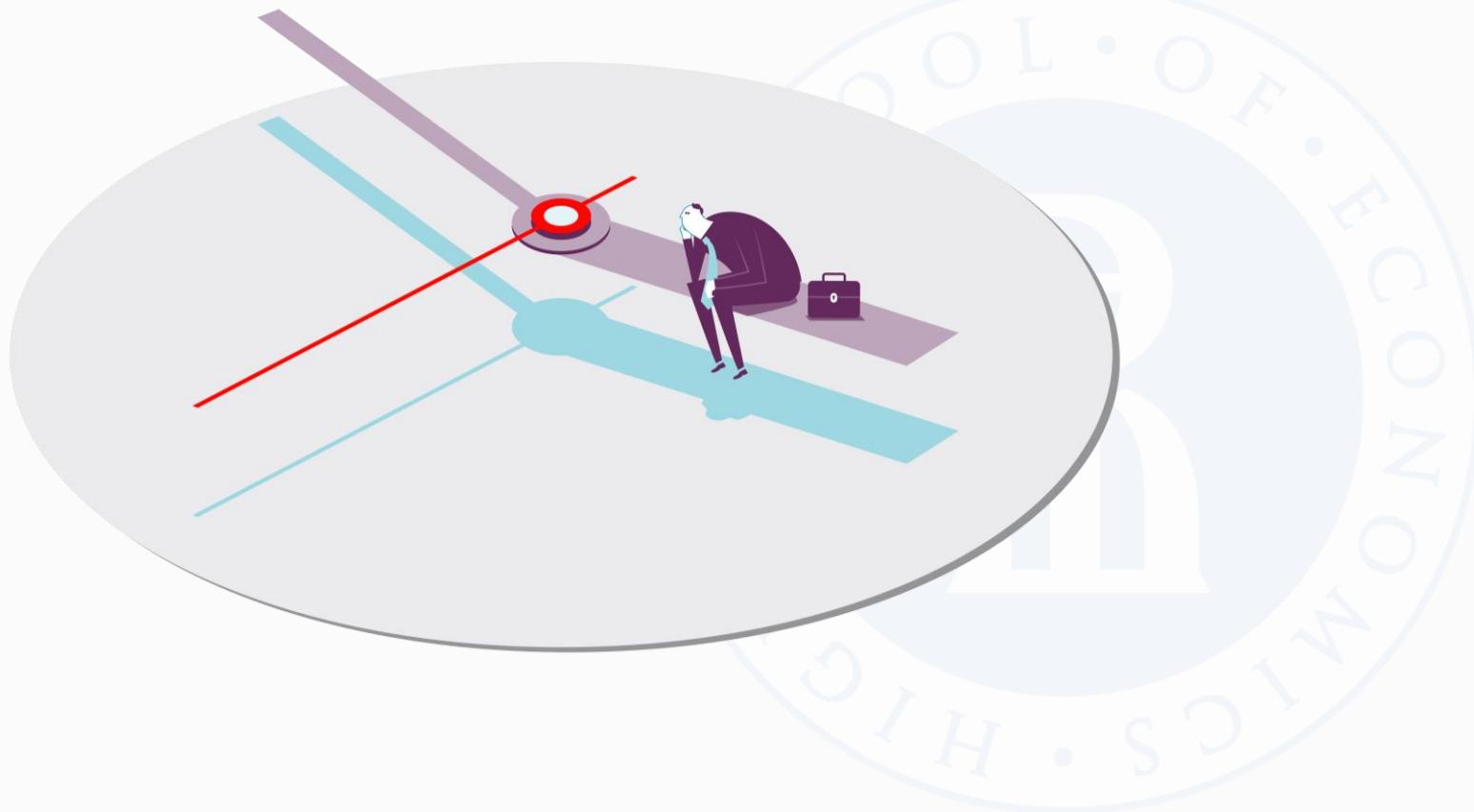
Основные проблемы при обучении

- Данные могут физически **не поместиться** на одной машине, а использование только небольшой части будет влиять на **качество** обучения



Основные проблемы при обучении

→ Если же данные смогли поместиться, обработка их только одним компьютером может оказаться слишком долгой

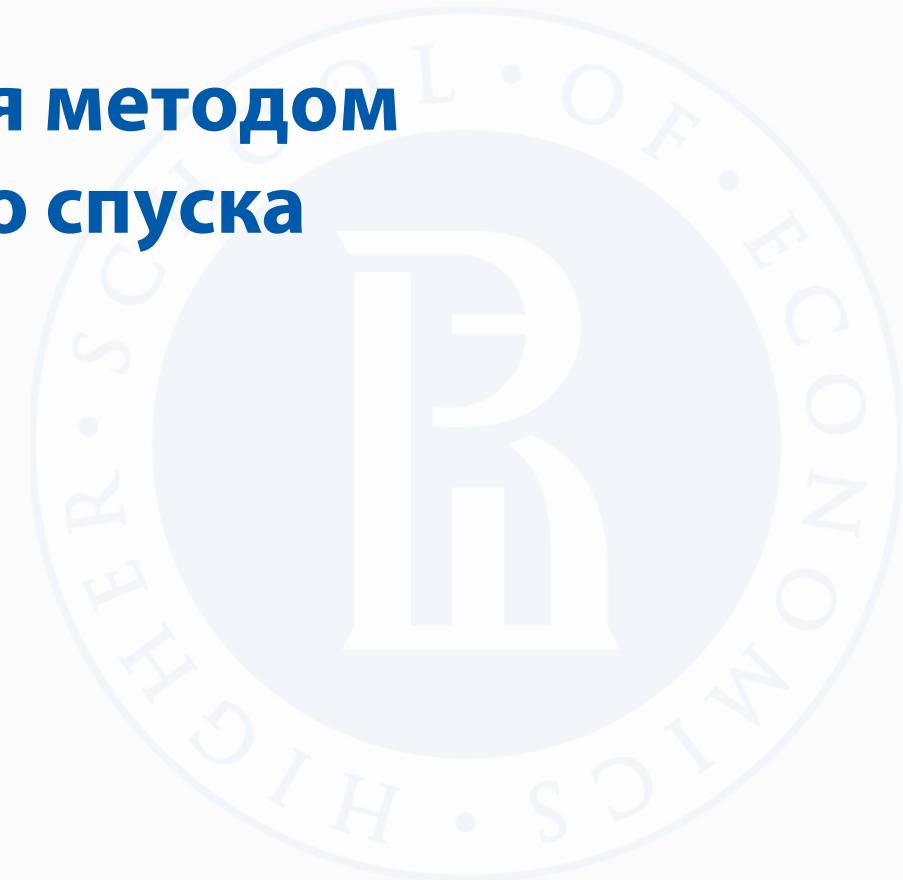


Основные проблемы при обучении

→ Просто добавить больше железа не всегда вариант —
нужно уметь эффективно использовать вычислительные
мощности для обработки большого объема данных

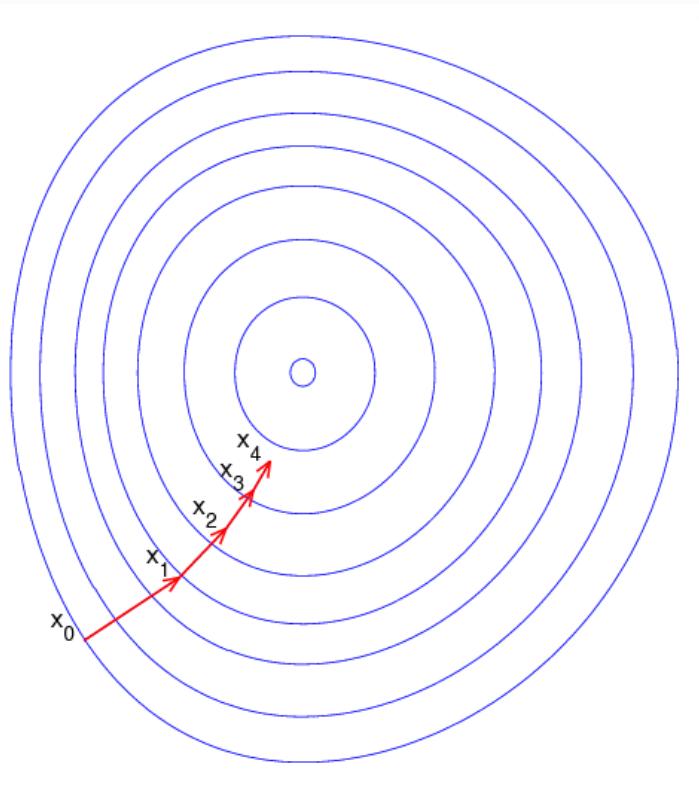


Оптимизация методом градиентного спуска



Сервер параметров

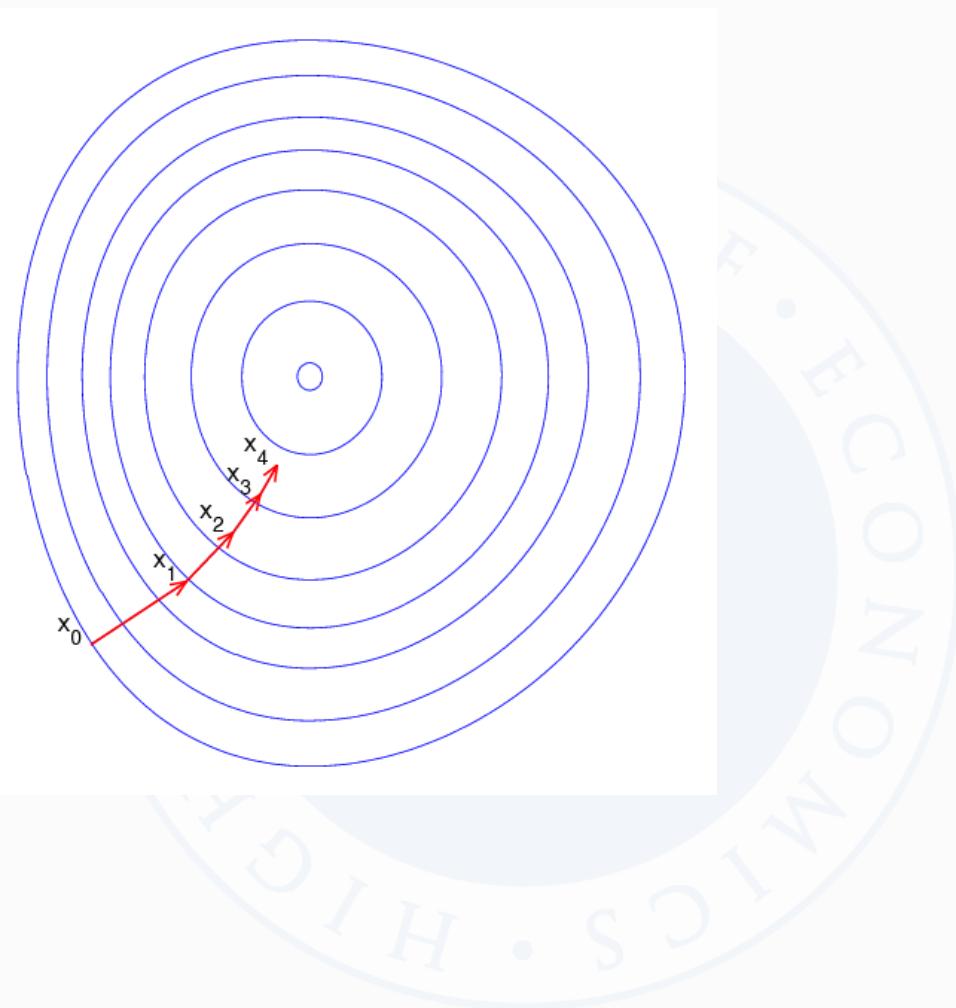
→ Классическая модель обучения модели с гладкой функцией потерь — это итеративный алгоритм, на каждом шаге которого 2 этапа:



Сервер параметров

→ Классическая модель обучения модели с гладкой функцией потерь — это итеративный алгоритм, на каждом шаге которого 2 этапа:

→ Рассчитать градиент функции потерь на данных

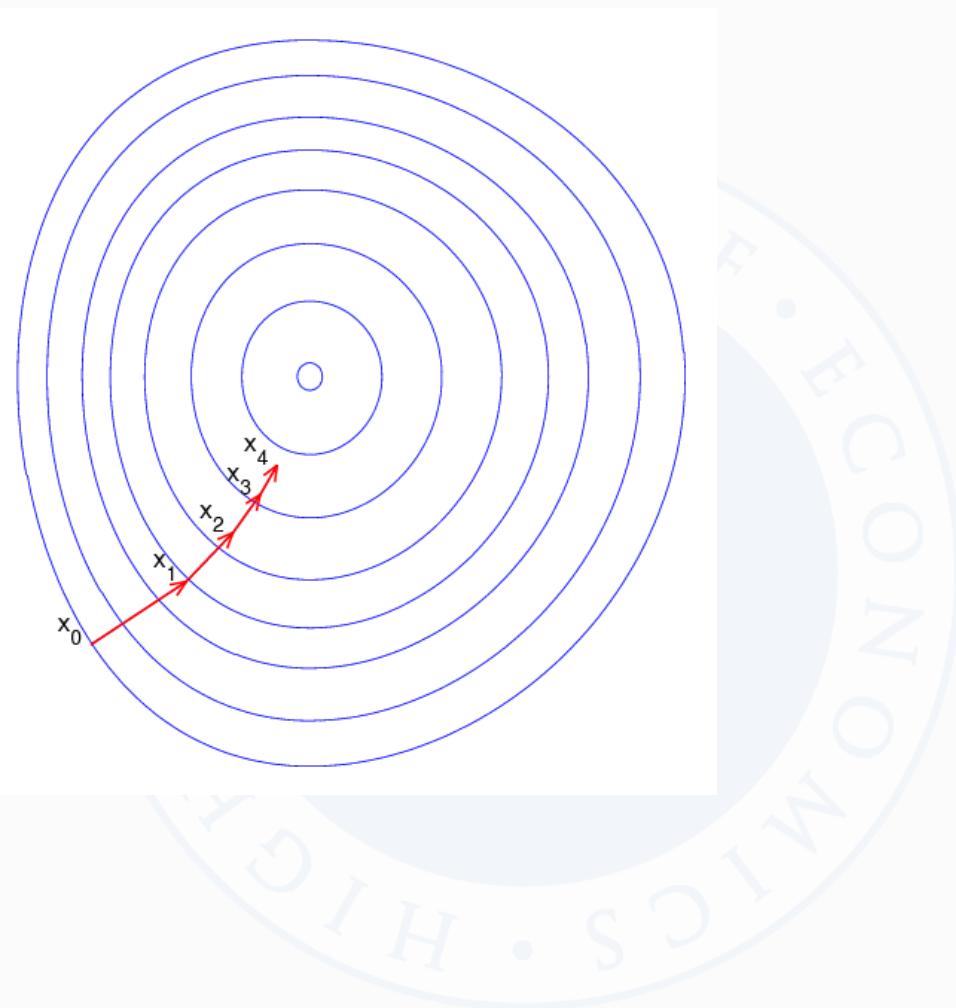


Сервер параметров

→ Классическая модель обучения модели с гладкой функцией потерь — это итеративный алгоритм, на каждом шаге которого 2 этапа:

→ Рассчитать градиент функции потерь на данных

→ Обновить веса, шагнув вдоль антиградиента



Сервер параметров

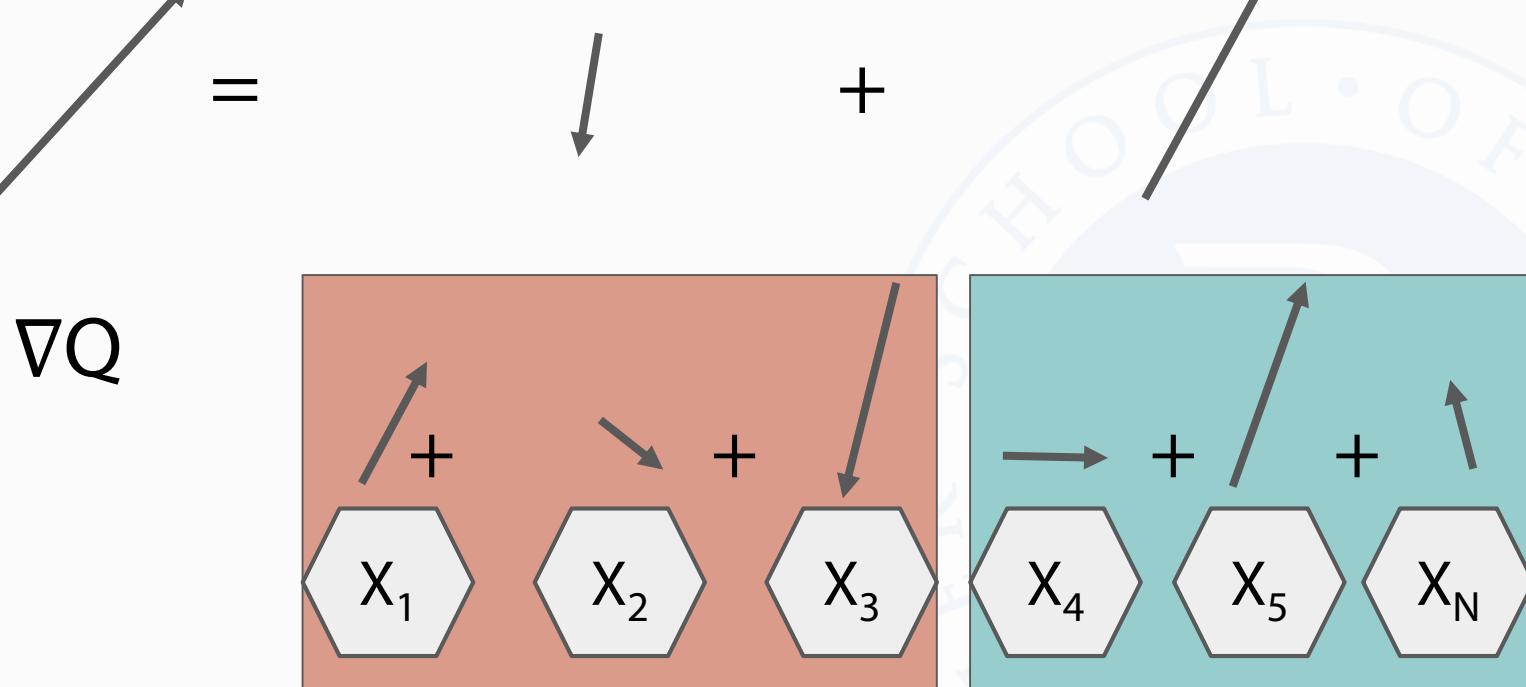
→ Итоговый градиент — это сумма градиентов подсчитанных на каждом объекте выборки

$$\nabla Q = \nabla X_1 + \nabla X_2 + \nabla X_3 + \nabla X_4 + \nabla X_5 + \dots + \nabla X_N$$

Сервер параметров



Сложение **не зависит от порядка**, а значит мы можем считать градиенты на объектах **параллельно** и потом сложить частичные суммы



Сервер параметров

→ Это свойство позволяет ускорять расчет как на **одной машине**, используя многоядерные CPU или GPU, так и на **нескольких машинах**. Базовый подход называется «Сервер параметров» (Parameter server)



Сервер параметров



Рабочий



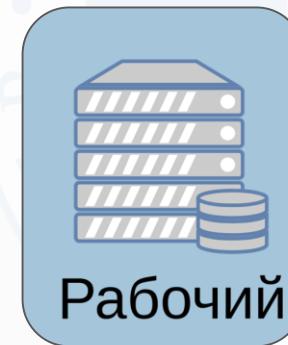
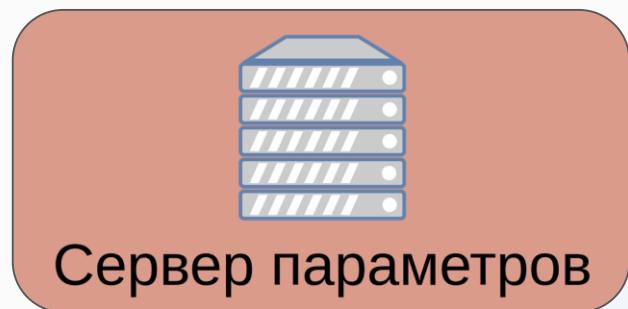
Рабочий



Рабочий

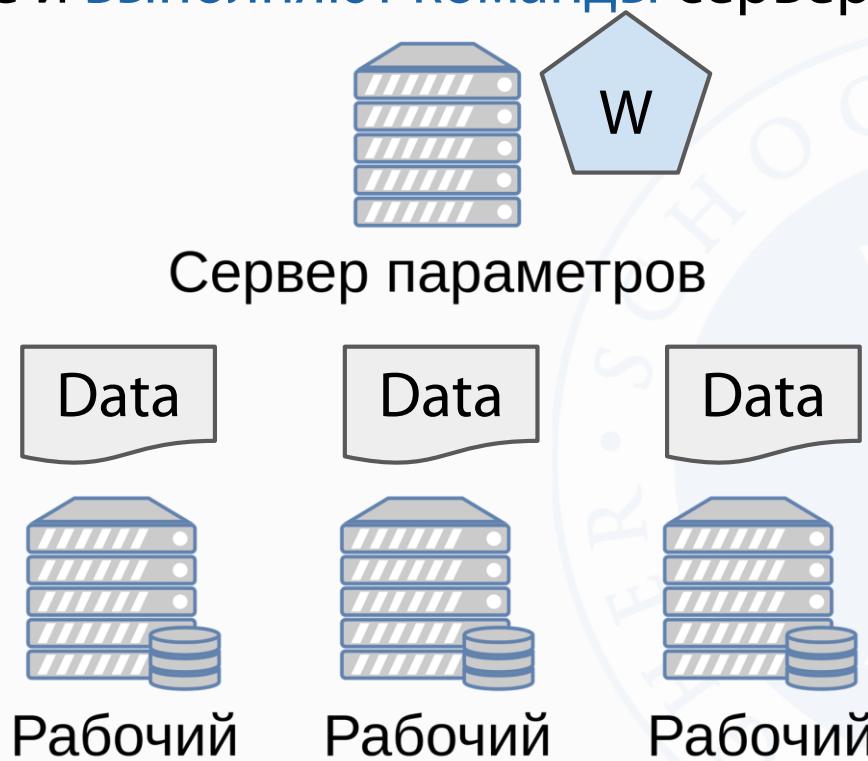
Сервер параметров

→ В этой архитектуре есть **главная машина** — сервер параметров — и рабочие



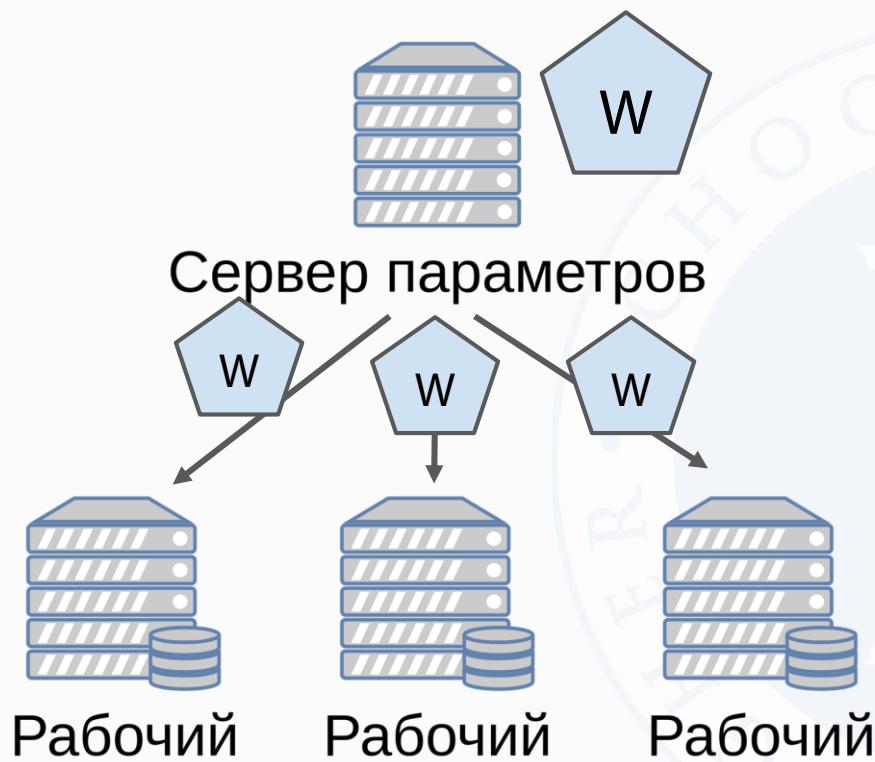
Сервер параметров

- Главная машина хранит веса модели и управляет всем процессом
- Рабочие между собой равномерно распределяют все данные и выполняют команды сервера параметров



Сервер параметров

→ 1 — сервер параметров раздает всем текущие [параметры](#) модели

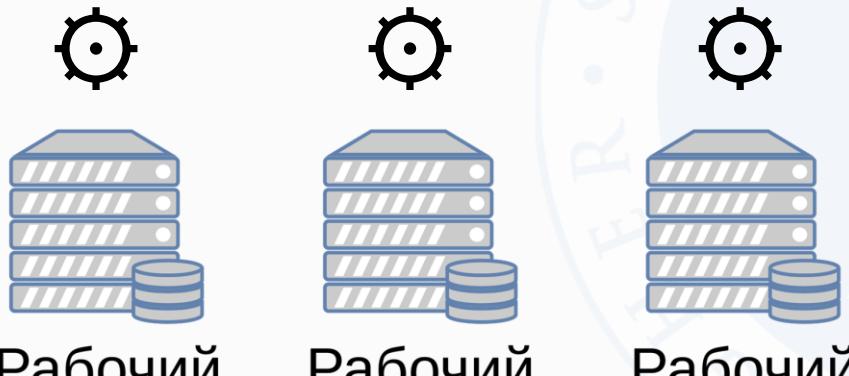


Сервер параметров

→ 2 — каждый рабочий рассчитывает частичный градиент на своей части данных

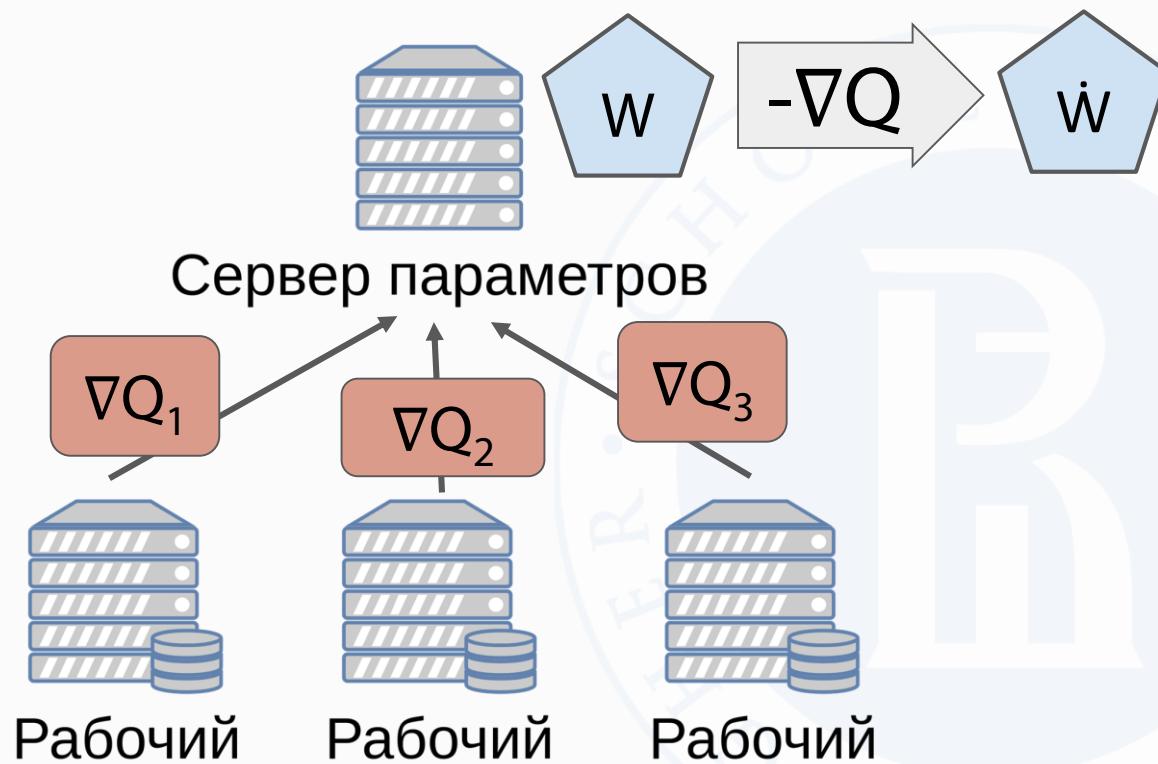


Сервер параметров



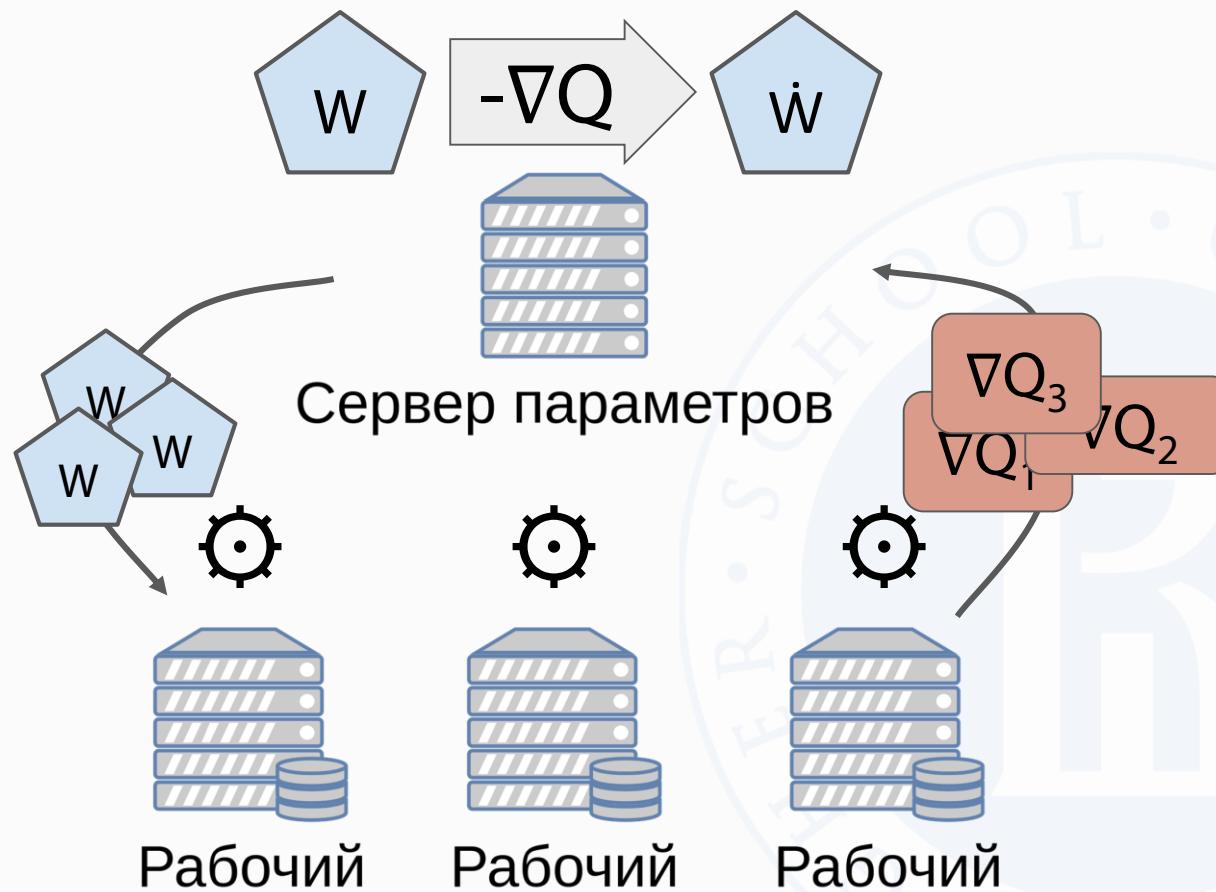
Сервер параметров

→ 3 — сервер параметров получает частичные градиенты от рабочих, складывает их и обновляет параметры модели



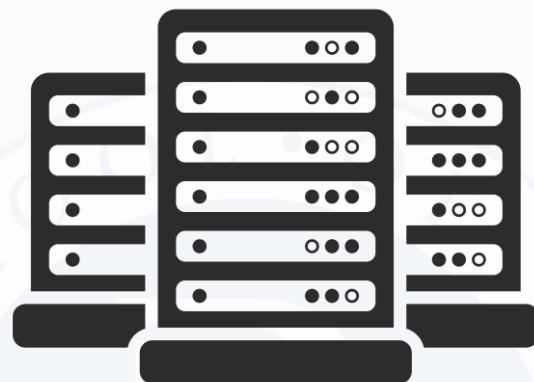
Сервер параметров

→ Этот процесс повторяется, пока градиентный спуск не сойдется. По итогу на главной машине окажутся параметры модели



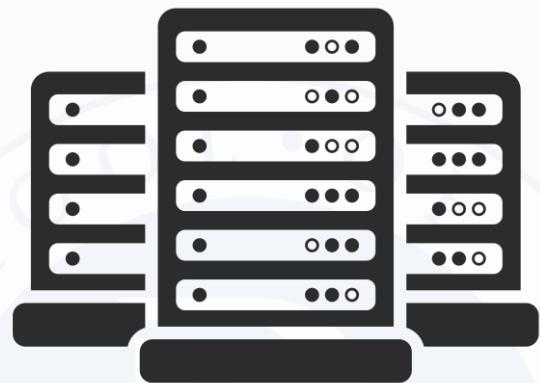
Сервер параметров

→ Метод концептуально простой, но уже позволяет обработать **потенциально произвольное количество данных**



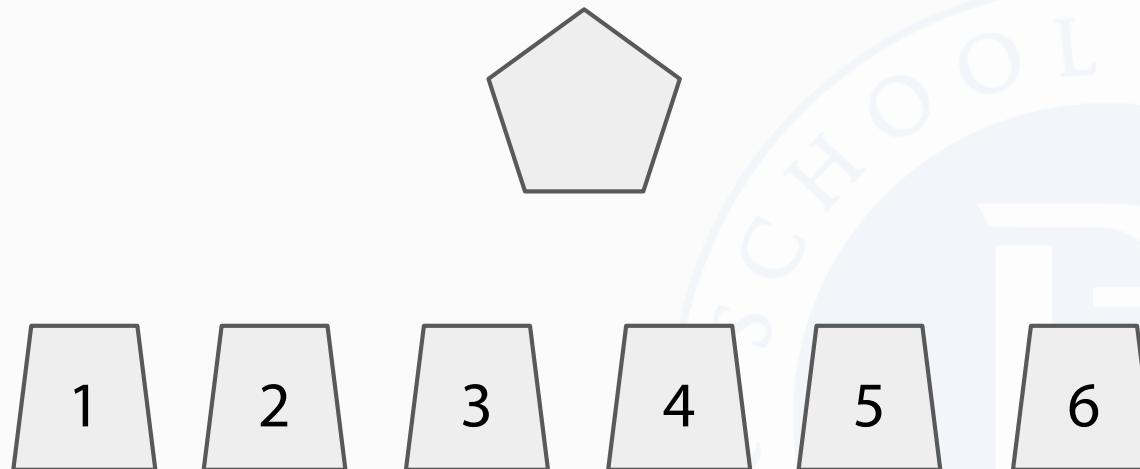
Сервер параметров

- Метод концептуально простой, но уже позволяет обработать **потенциально произвольное количество данных**
- Вычислительные мощности используются достаточно **эффективно**, однако теперь проблемой может стать передача данных по сети



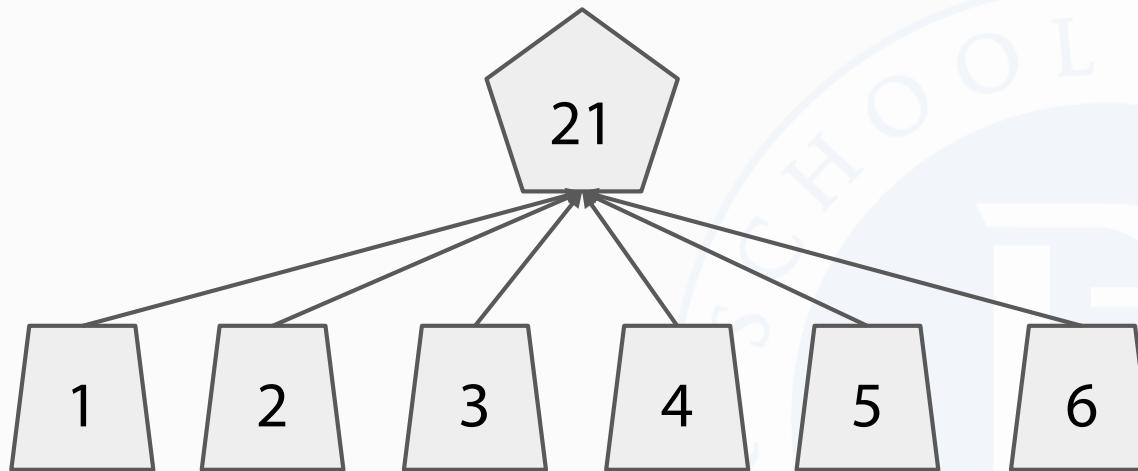
Tree Allreduce

- Для наглядности рассмотрим простую задачу [сложения чисел](#), но на нескольких машинах
- Подход «Сервер параметров» тогда выглядел бы вот так:



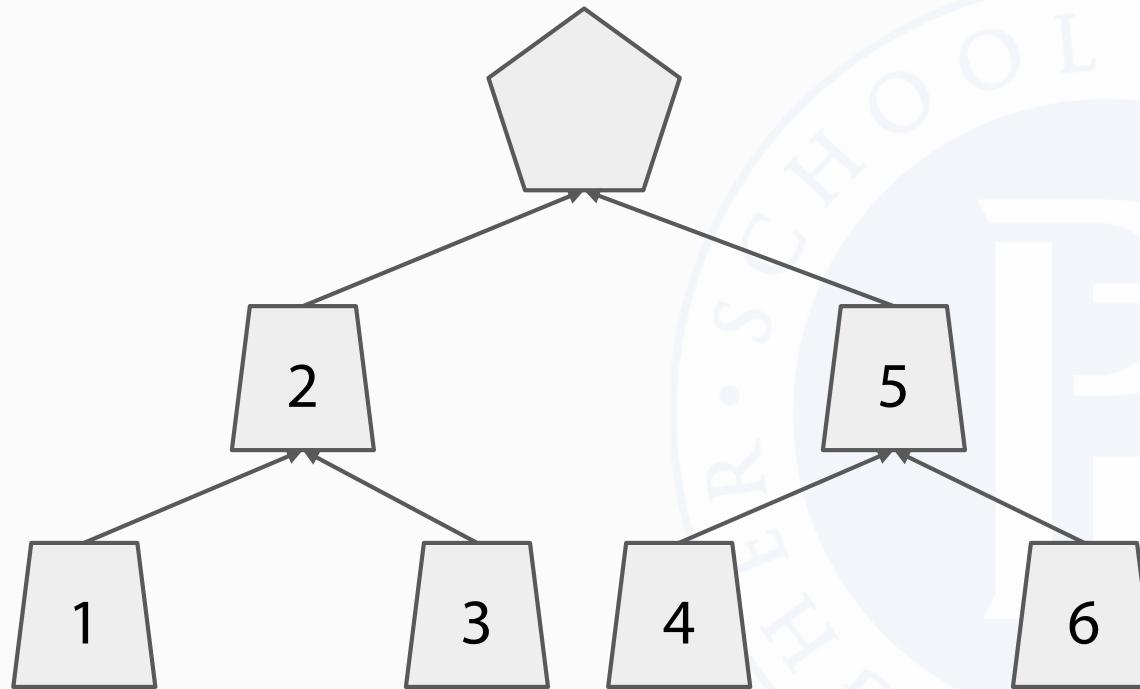
Tree Allreduce

→ Каждая машина передала бы свое число на главную машину, которая просто сложила бы все числа



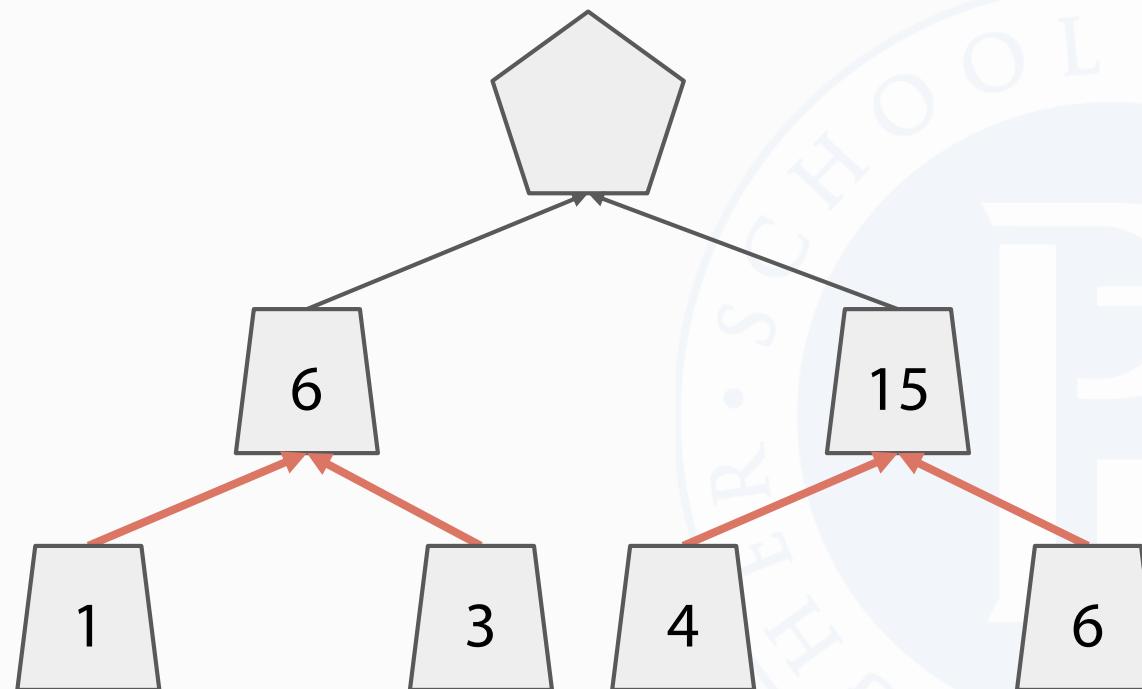
Tree Allreduce

→ Теперь попробуем немного переорганизовать машины, чтобы они выстроились в «дерево»



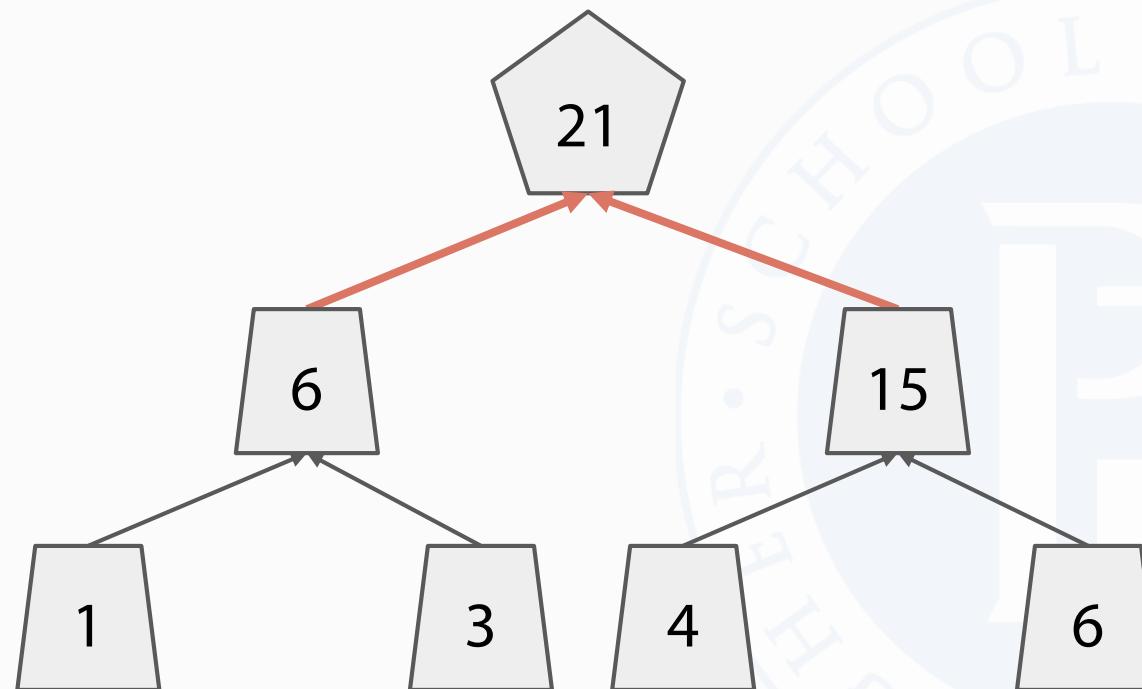
Tree Allreduce

→ Теперь расчет пойдет в несколько этапов. Сначала данные передадут машины **внизу** дерева. Машины выше получат данные и **сложат со своими**



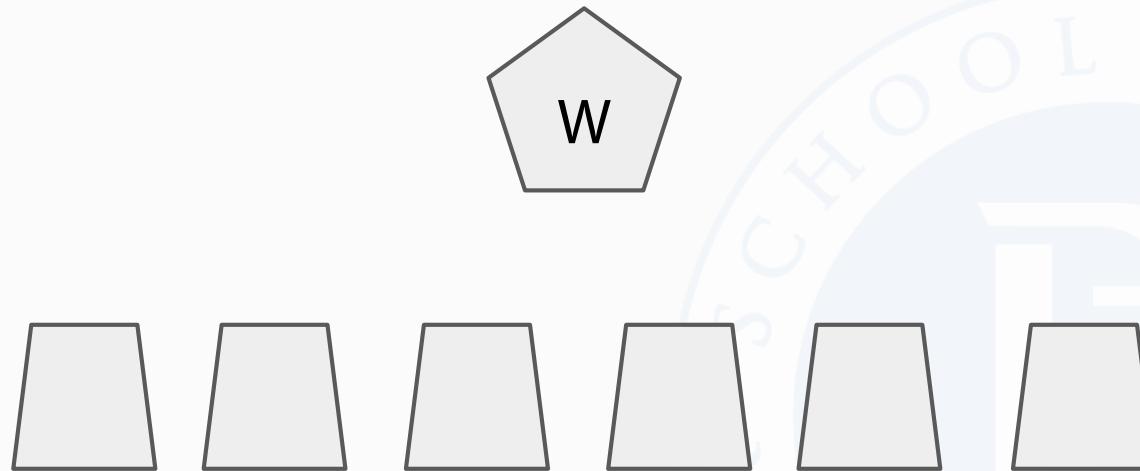
Tree Allreduce

→ Данные будут передаваться по дереву, пока они не дойдут до корня. В данном примере оставшиеся две машины сразу передадут числа в главную машину



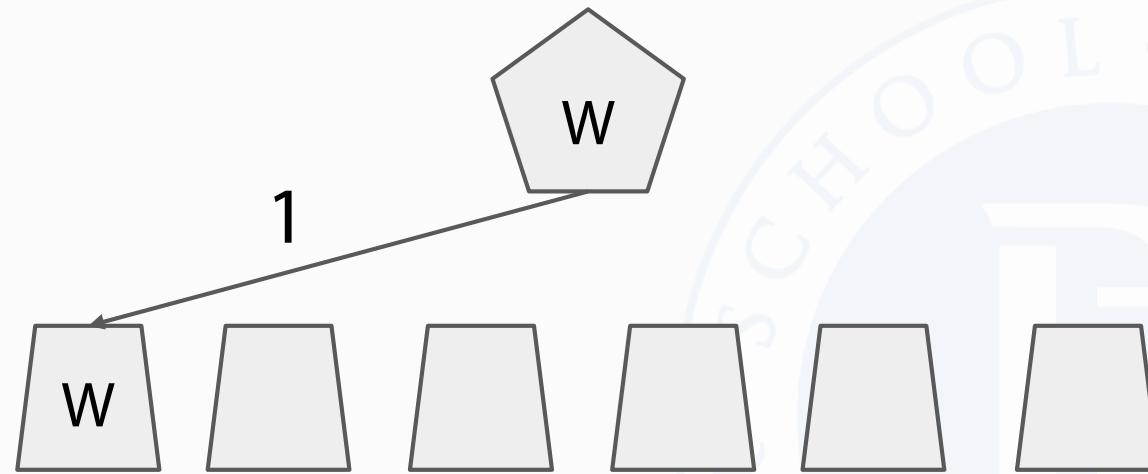
Tree Allreduce

- Может показаться, что такая схема работает еще дольше, чем простой сервер параметров
- Однако давайте посчитаем реальное время на передачу данных в обоих случаях



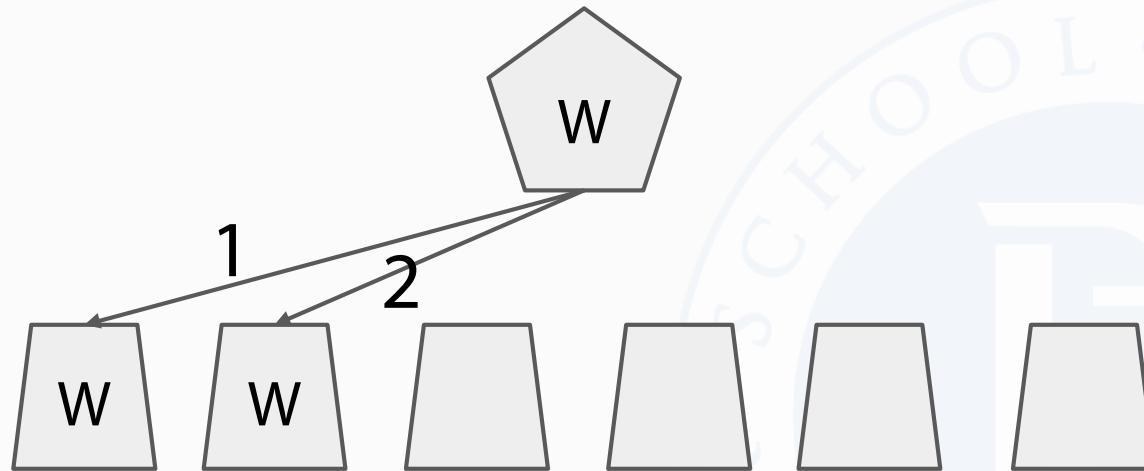
Tree Allreduce

→ Вначале серверу нужно передать коэффициенты всем рабочим



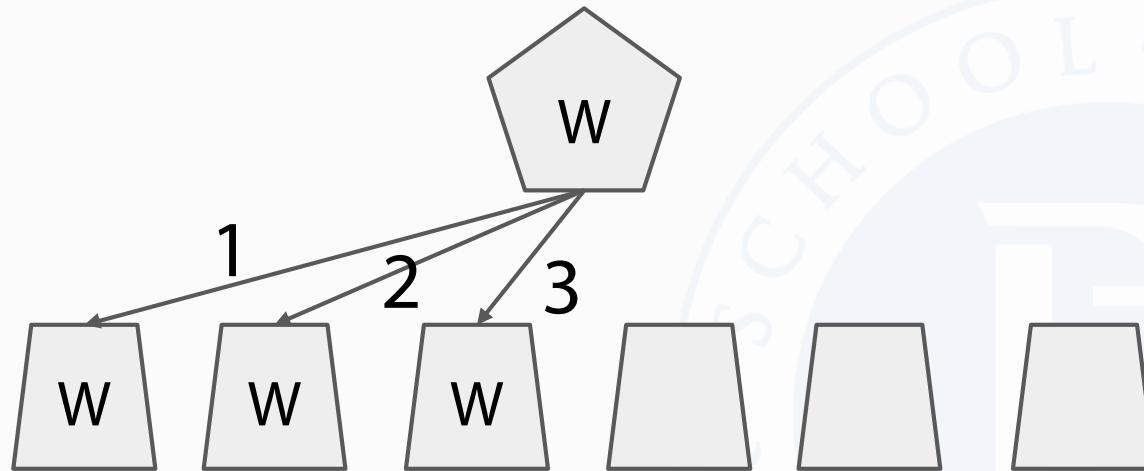
Tree Allreduce

→ Вначале серверу нужно передать коэффициенты всем рабочим



Tree Allreduce

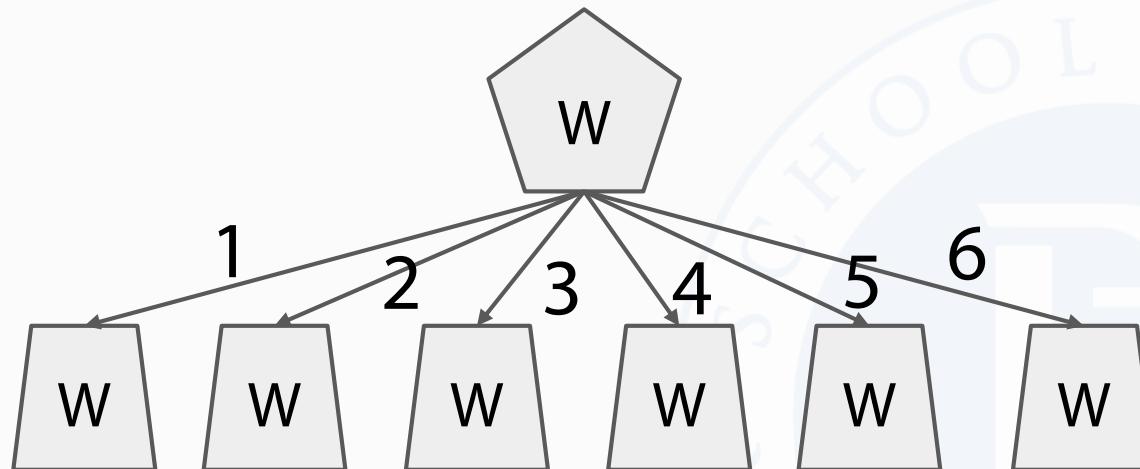
→ Вначале серверу нужно передать коэффициенты всем рабочим



Tree Allreduce

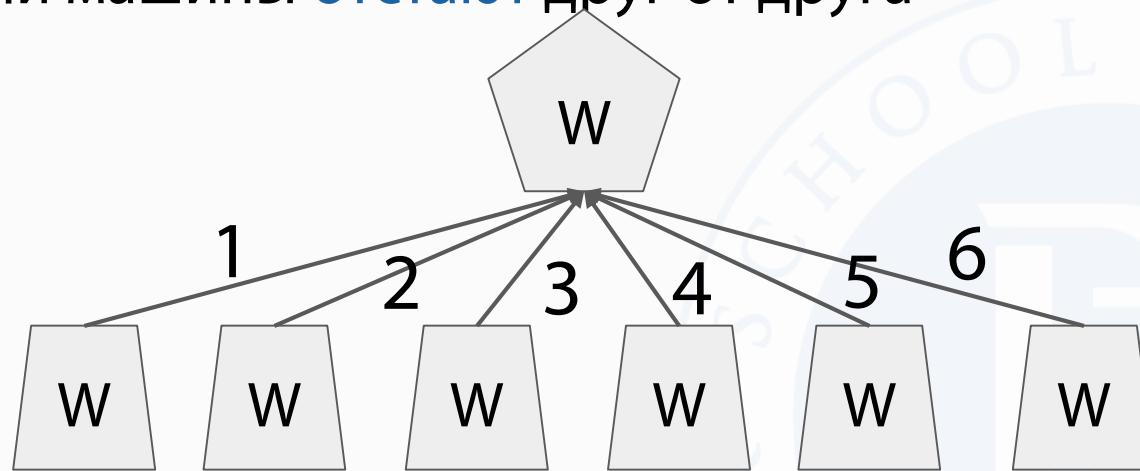


Всего 6 «тактов» потребовалось на то, чтобы раздать параметры



Tree Allreduce

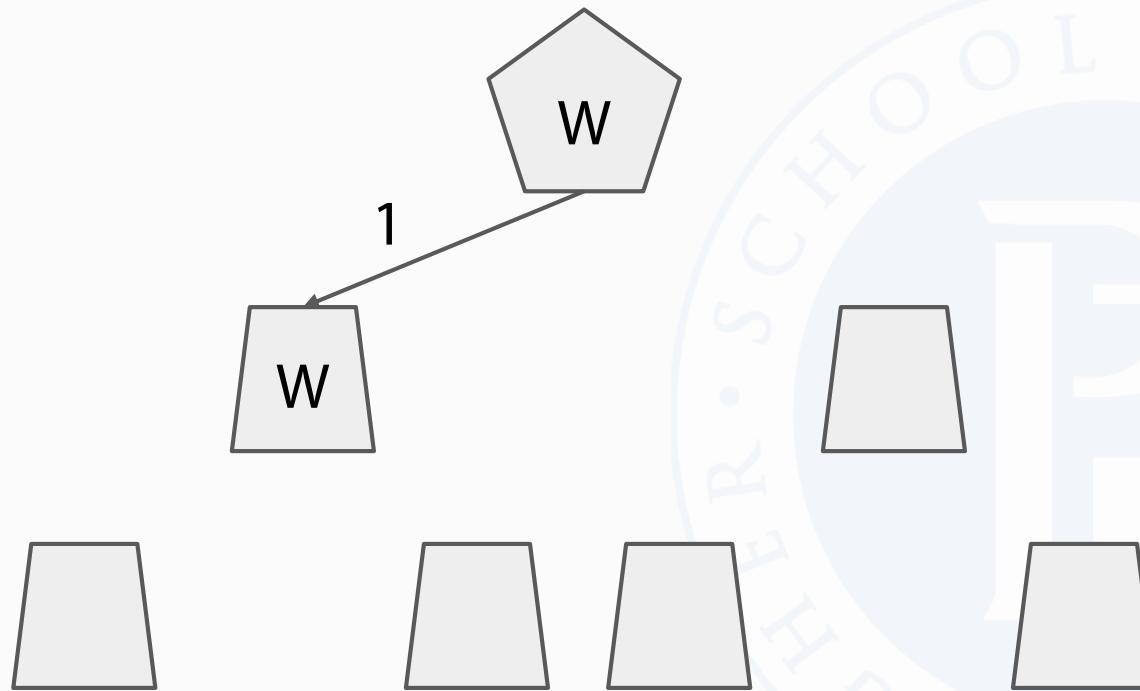
- И еще 6 «тактов» для передачи градиентов от рабочих по такой же схеме
- Сетевая способность ограничена, и эффективно принимать данные сразу от всех — нереалистично, а из-за долгой раздачи машины отстают друг от друга



Tree Allreduce

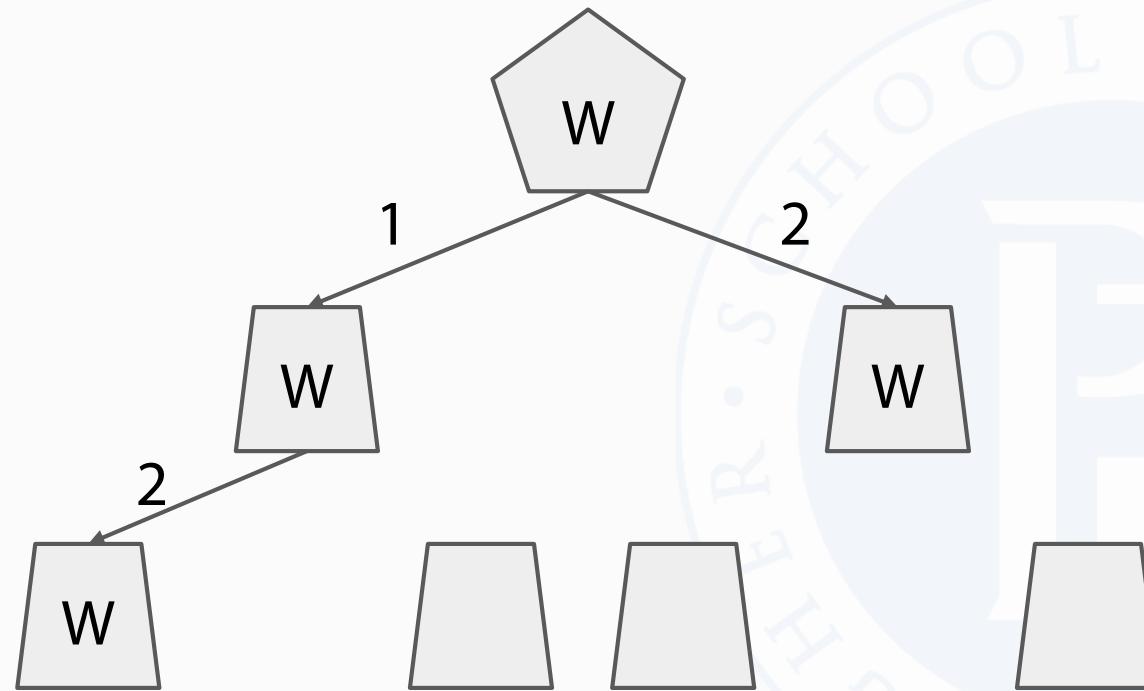


Посмотрим теперь, как работает схема с деревом



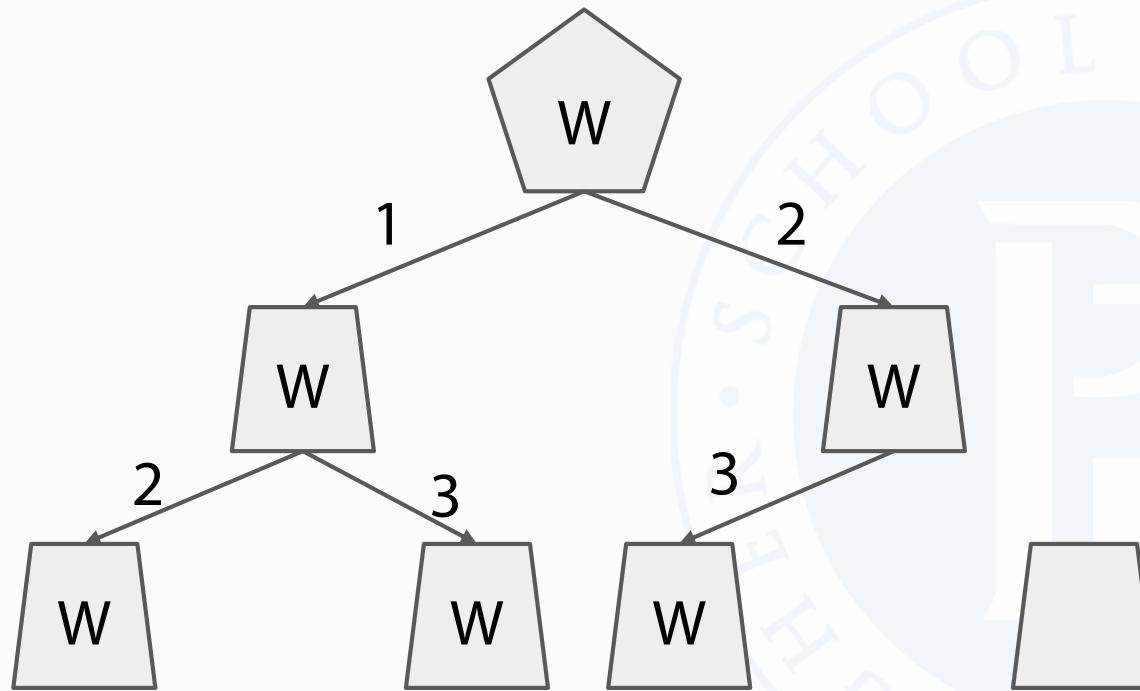
Tree Allreduce

→ Так как вторая машина уже получила данные, она сама может начать передавать их другим



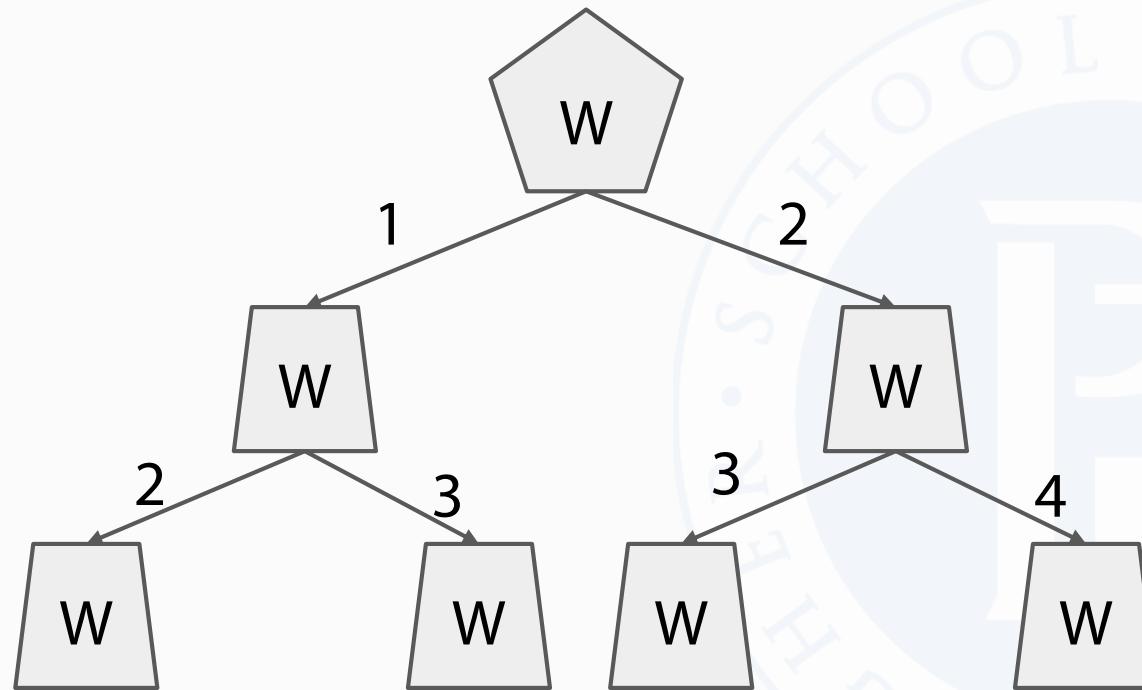
Tree Allreduce

→ Так как вторая машина уже получила данные, она сама может начать передавать их другим



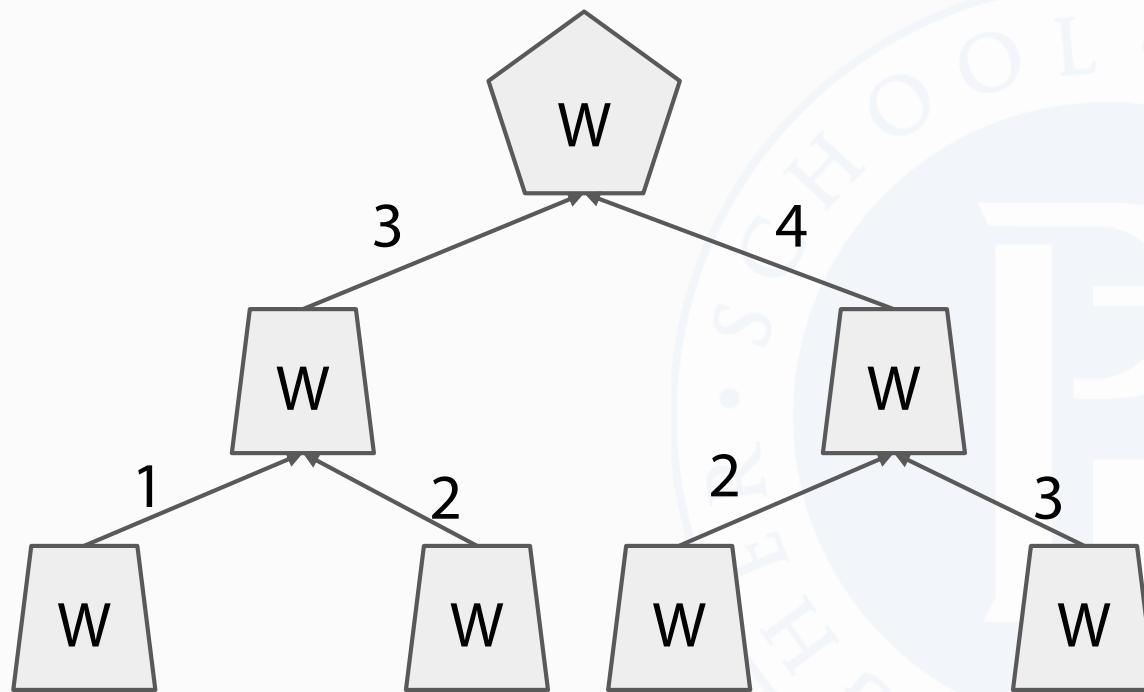
Tree Allreduce

→ Таким образом, при построении деревом **последняя** машина получит данные уже через **4 «такта»**



Tree Allreduce

→ Аналогичным образом за 4 «такта» рассчитанные частичные градиенты передадутся на главную машину

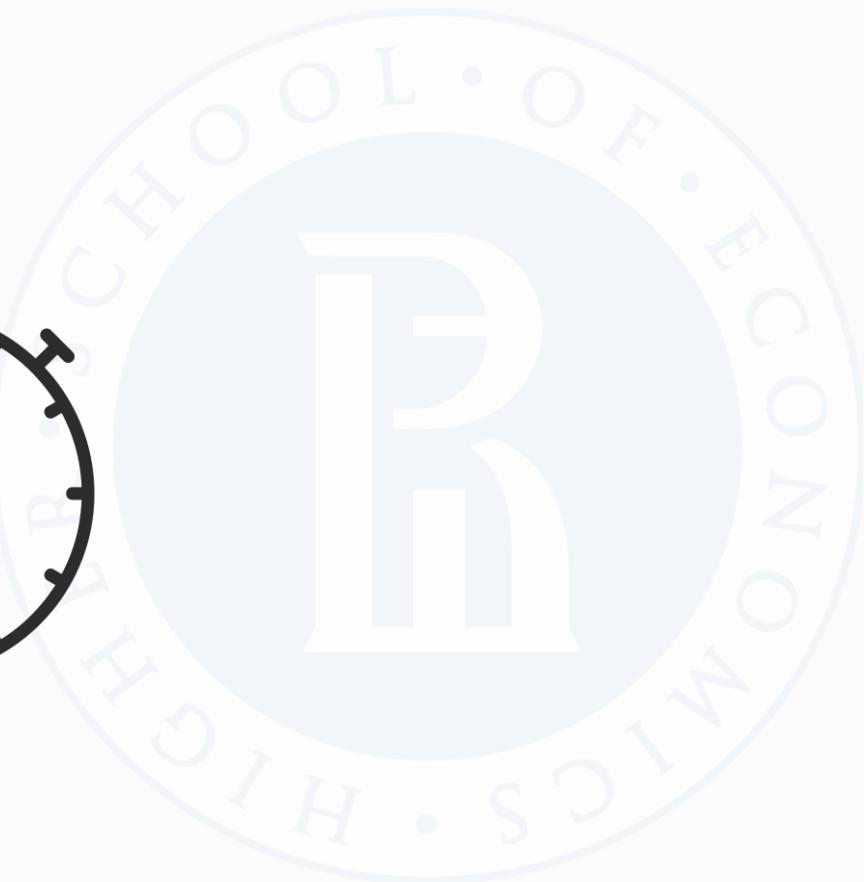


Tree Allreduce

- Обычный сервер параметров: 12 «тактов»
В общем случае для N рабочих — $2N$ «тактов»
- Схема из дерева: 8 «тактов»
В общем случае для N рабочих — $4\log_2(N+2)$ «тактов»

Tree Allreduce

- Логарифм растет гораздо медленнее, чем линейная функция



Tree Allreduce

- Логарифм растет гораздо медленнее, чем линейная функция
- Так, для кластера из 1022 рабочих обычная схема будет занимать 2044 «такта» на всего один шаг спуска



Tree Allreduce

- Логарифм растет гораздо медленнее, чем линейная функция
- Так, для кластера из 1022 рабочих обычная схема будет занимать 2044 «такта» на всего один шаг спуска
- При этом схема с деревом будет работать за 40 «тактов»



Tree Allreduce

→ Подобный подход называется Tree Allreduce, и он позволяет эффективно масштабировать вычислительные мощности для обучения



Tree Allreduce

- Подобный подход называется Tree Allreduce, и он позволяет эффективно масштабировать вычислительные мощности для обучения
- Схема может использоваться не только для градиентного спуска — очень много алгоритмов может быть улучшено через эту схему

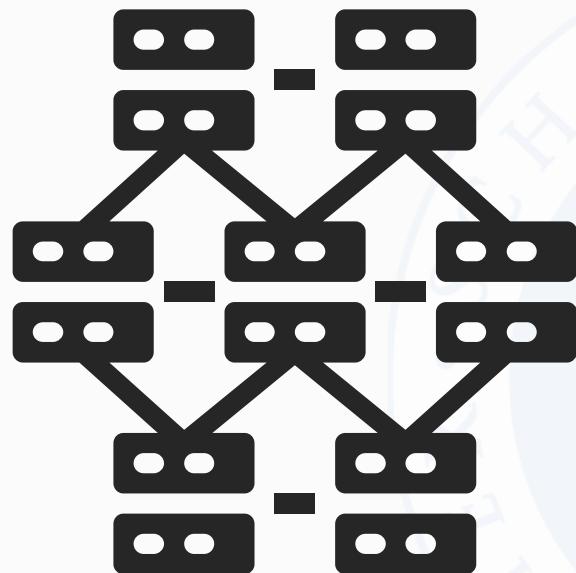


Распределенное обучение деревьев



Деревья и ансамбли

→ Если данные физически помещаются на одну машину, то есть достаточно много способов распараллелить обучение

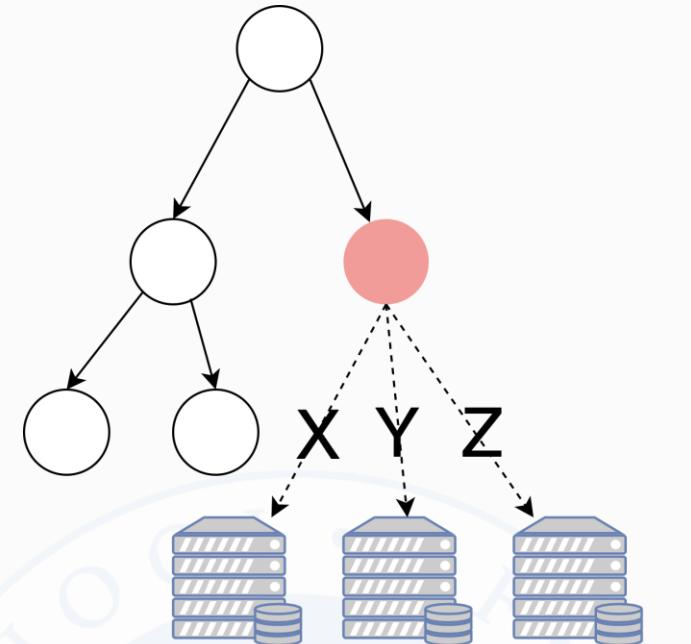
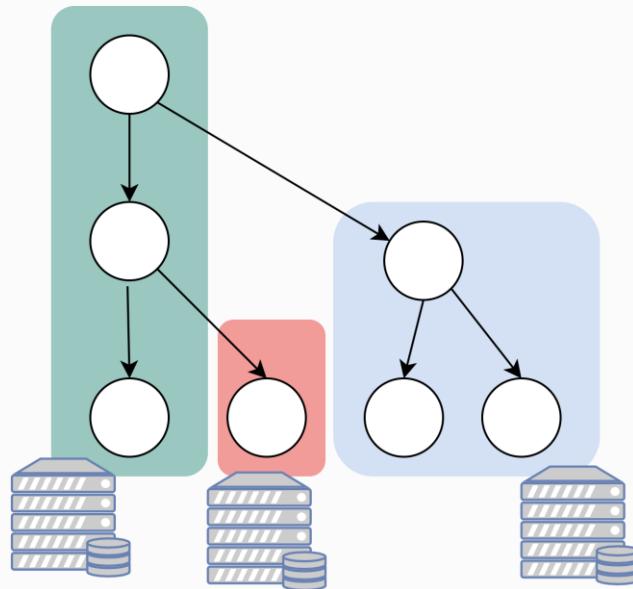


Деревья и ансамбли



Можно разделить работу по:

- Обучению отдельных поддеревьев
- Поиску лучшего граничного значения
- Поиску лучшего признака
- Деревьям в ансамбле



Деревья и ансамбли

→ Однако когда данные не помещаются в машину, возникает проблема: чтобы выбрать [лучшее разбиение](#) в вершине, нужна информация [про всю выборку](#), а передавать все данные между машинами — очень [затратно](#)



В поисках эффективного алгоритма

- Хочется найти алгоритм, который бы работал примерно как «сервер параметров»:
- Рабочие локально считали бы что-то компактное на своих данных



В поисках эффективного алгоритма

→ Хочется найти алгоритм, который бы работал примерно как «сервер параметров»:

- Рабочие локально считали бы что-то компактное на своих данных
- Передавали на главный сервер



В поисках эффективного алгоритма

→ Хочется найти алгоритм, который бы работал примерно как «сервер параметров»:

- Рабочие локально считали бы что-то компактное на своих данных
- Передавали на главный сервер
- Главный сервер агрегировал бы результаты и принимал итоговое решение

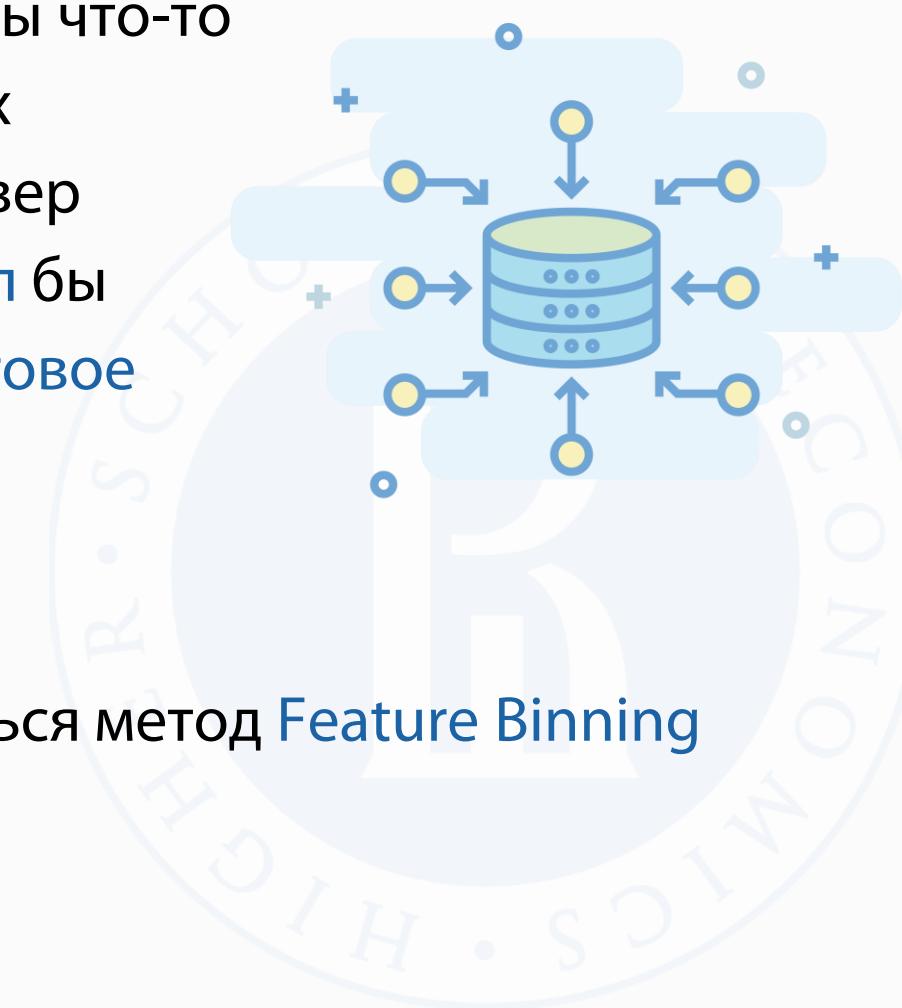


В поисках эффективного алгоритма

→ Хочется найти алгоритм, который бы работал примерно как «сервер параметров»:

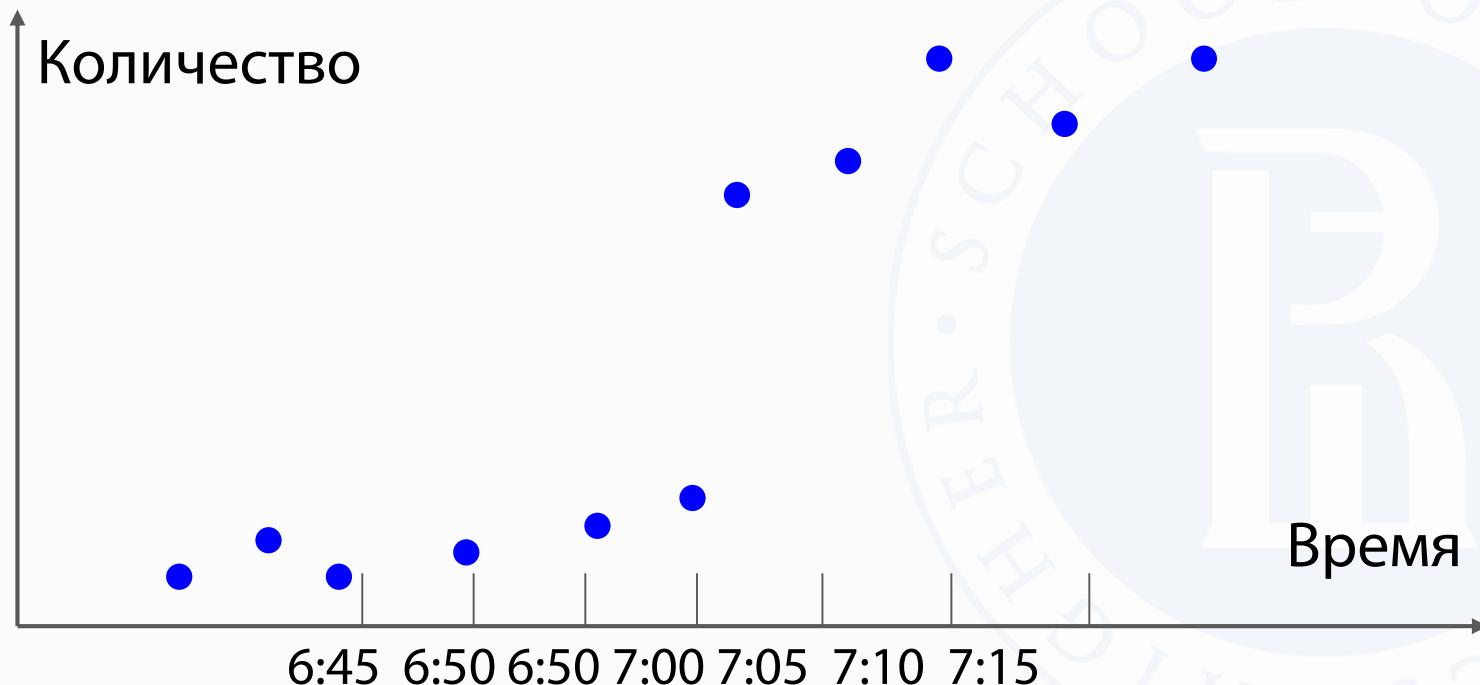
- Рабочие локально считали бы что-то компактное на своих данных
- Передавали на главный сервер
- Главный сервер агрегировал бы результаты и принимал итоговое решение

→ Подобного позволяет добиться метод Feature Binning



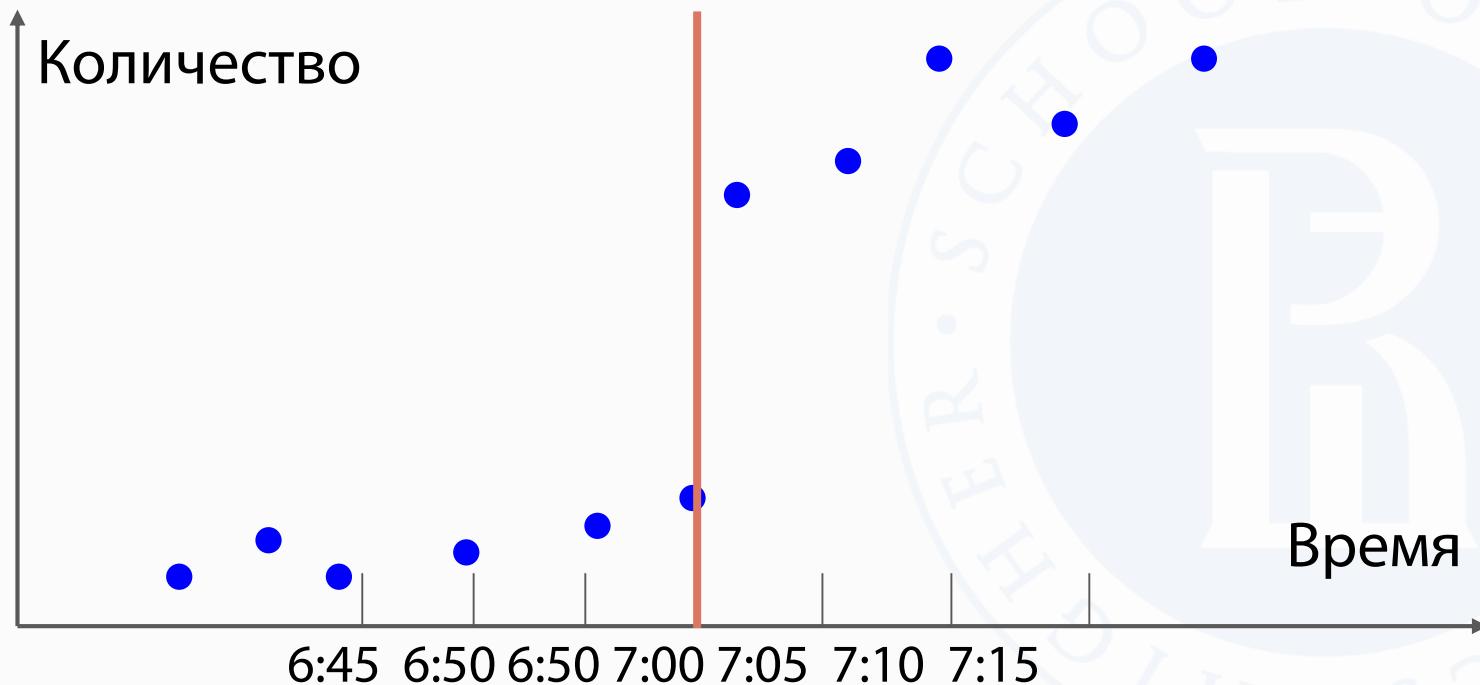
В поисках эффективного алгоритма

- Пример для наглядности — хотим предсказывать количество посетителей на сайте с прогнозом погоды
- Один из признаков — время. Нам нужно понять, как разбить для него вершину. Данные, которые у нас есть:



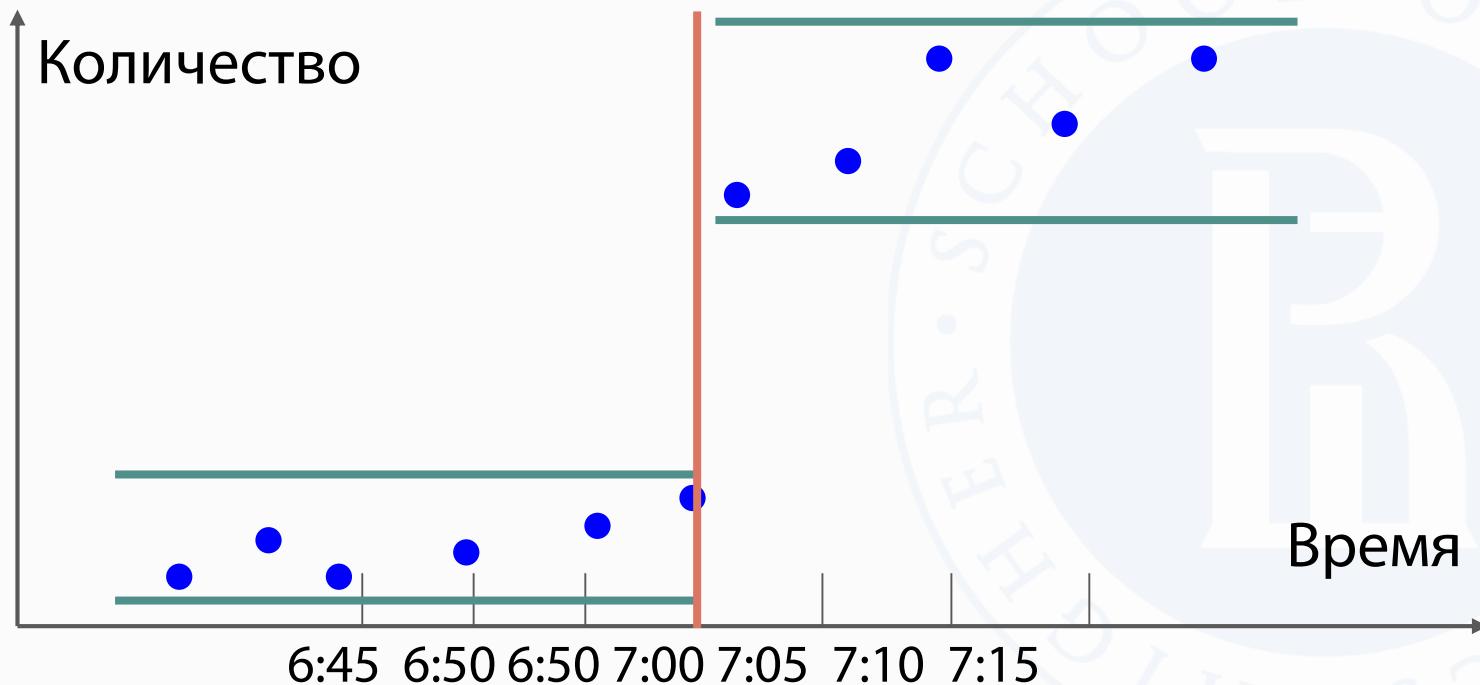
В поисках эффективного алгоритма

→ Интуитивно понятно, что разделить признак нужно в 7:00. Ответ на «Сейчас больше 7?» даст **больше всего информации** про то, сколько сейчас людей: много или мало



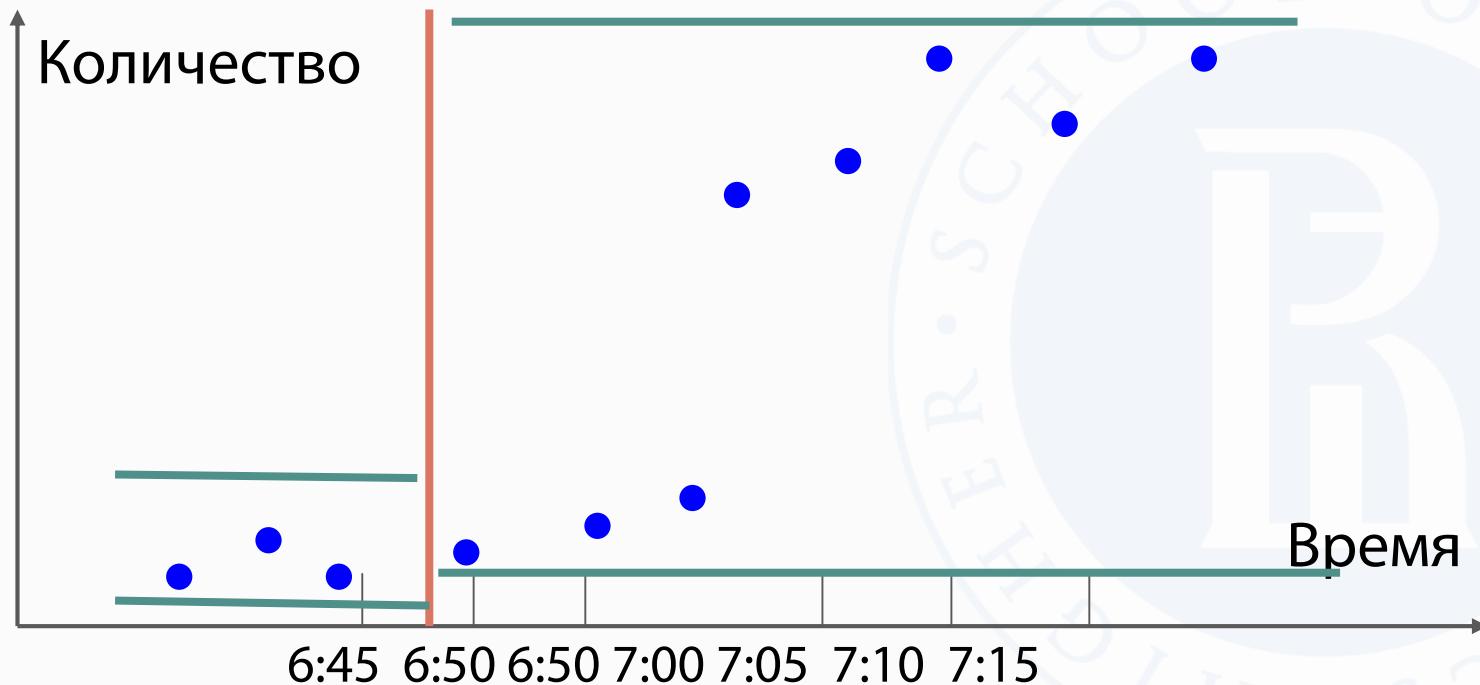
В поисках эффективного алгоритма

→ Для машины это можно объяснить так: [дисперсия](#) (разброс) в двух кучках по отдельности должна быть очень маленькой



В поисках эффективного алгоритма

→ Нужно лишь перебрать все возможные разбиения на кучки и найти с **наименьшей дисперсией** в кучках



В поисках эффективного алгоритма

→ Формула дисперсии:

$$\sum(Y - \bar{Y}/N)^2/N = \sum Y^2 / N - (\sum Y / N)^2$$

В поисках эффективного алгоритма

→ Формула дисперсии:

$$\sum(Y - \bar{Y}/N)^2/N = \sum Y^2 / N - (\sum Y / N)^2$$

→ Таким образом, чтобы расчитать дисперсию, нужно знать три значения:

$$\sum Y^2, \sum Y \text{ и } N$$

В поисках эффективного алгоритма

→ Формула дисперсии:

$$\sum(Y - \bar{Y}/N)^2/N = \sum Y^2 / N - (\sum Y / N)^2$$

→ Таким образом, чтобы расчитать дисперсию, нужно знать три значения:

$$\sum Y^2, \sum Y \text{ и } N$$

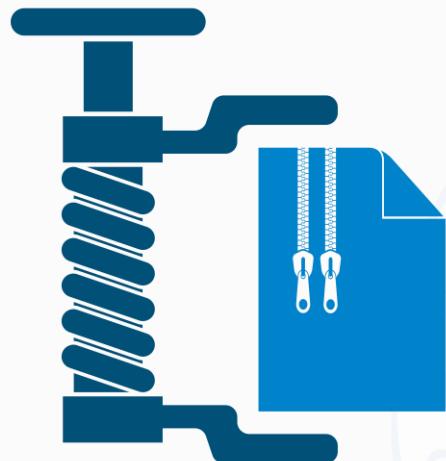
→ Так как все три компонента — это сумма, значит их все можно считать **параллельно**, а потом **агрегировать**

В поисках эффективного алгоритма

→ Остается проблема — если в выборке N элементов, то это потенциально N различных значений, которые нужно перебрать для нахождения разбиения

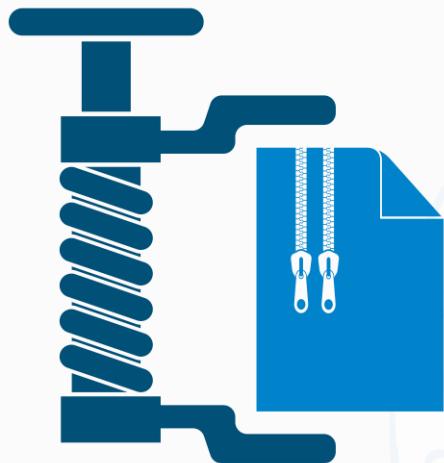
В поисках эффективного алгоритма

- Остается проблема — если в выборке N элементов, то это потенциально N различных значений, которые нужно перебрать для нахождения разбиения
- При больших N считать такое очень долго



В поисках эффективного алгоритма

- Остается проблема — если в выборке N элементов, то это потенциально N различных значений, которые нужно перебрать для нахождения разбиения
- При больших N считать такое очень долго



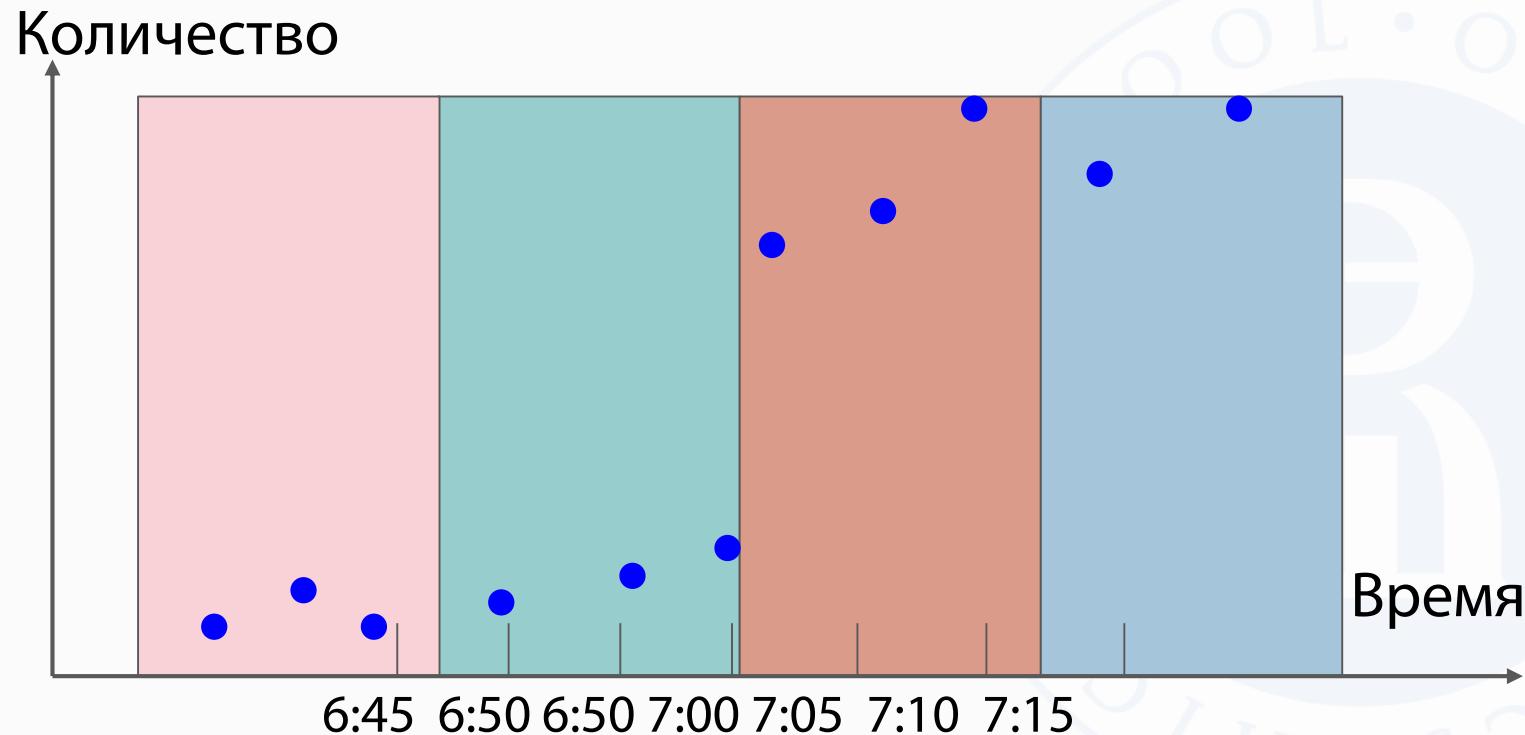
- Хочется как-то сжать данные, не потеряв при этом много информации

Feature binning



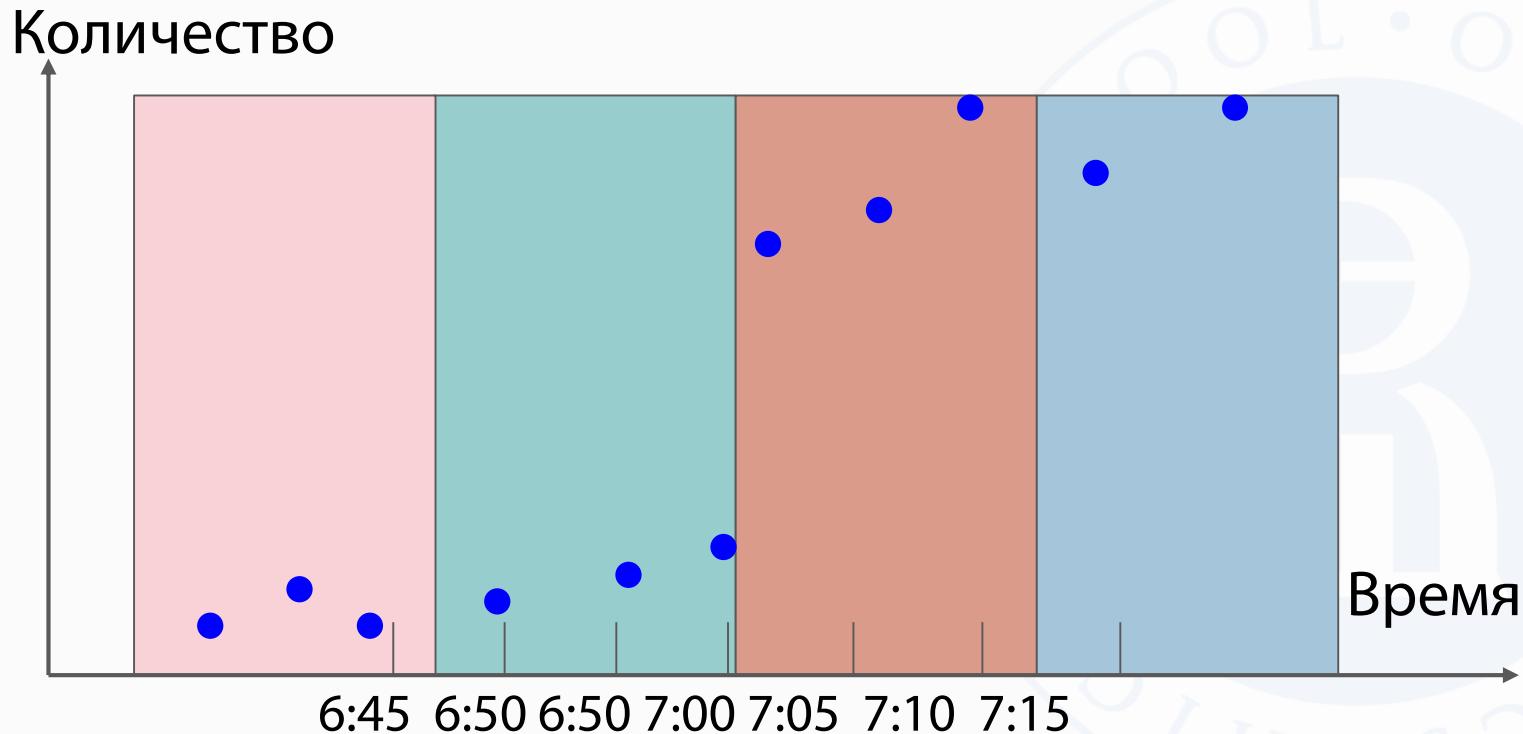
Идея: сгруппируем данные «по корзинкам».

Каждая корзинка будет хранить элементы от какого-то значения i до j



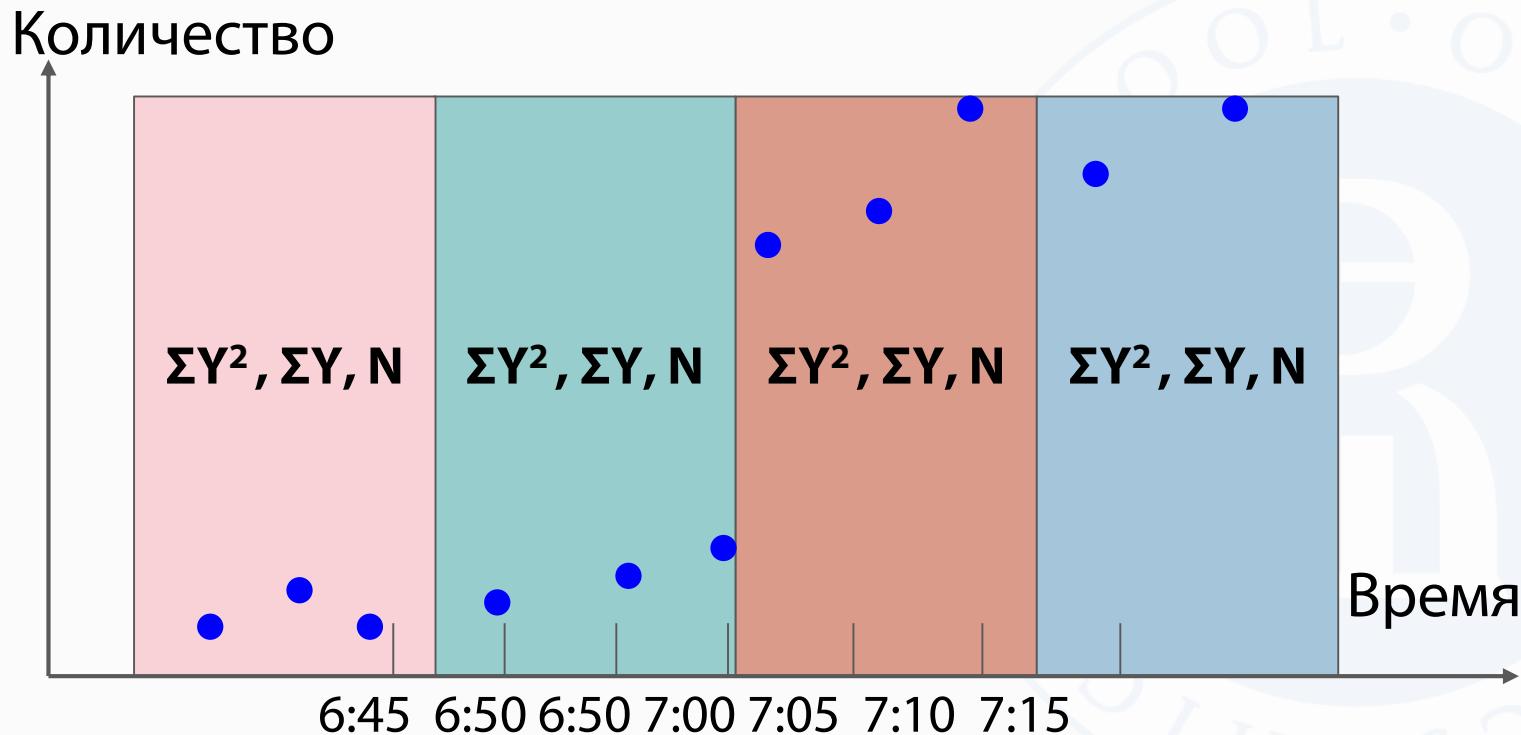
Feature binning

- Идея: сгруппируем данные «по корзинкам». Каждая корзинка будет хранить элементы от какого-то значения i до j
- В нашем примере — 4 корзинки вместо 11 изначальных объектов



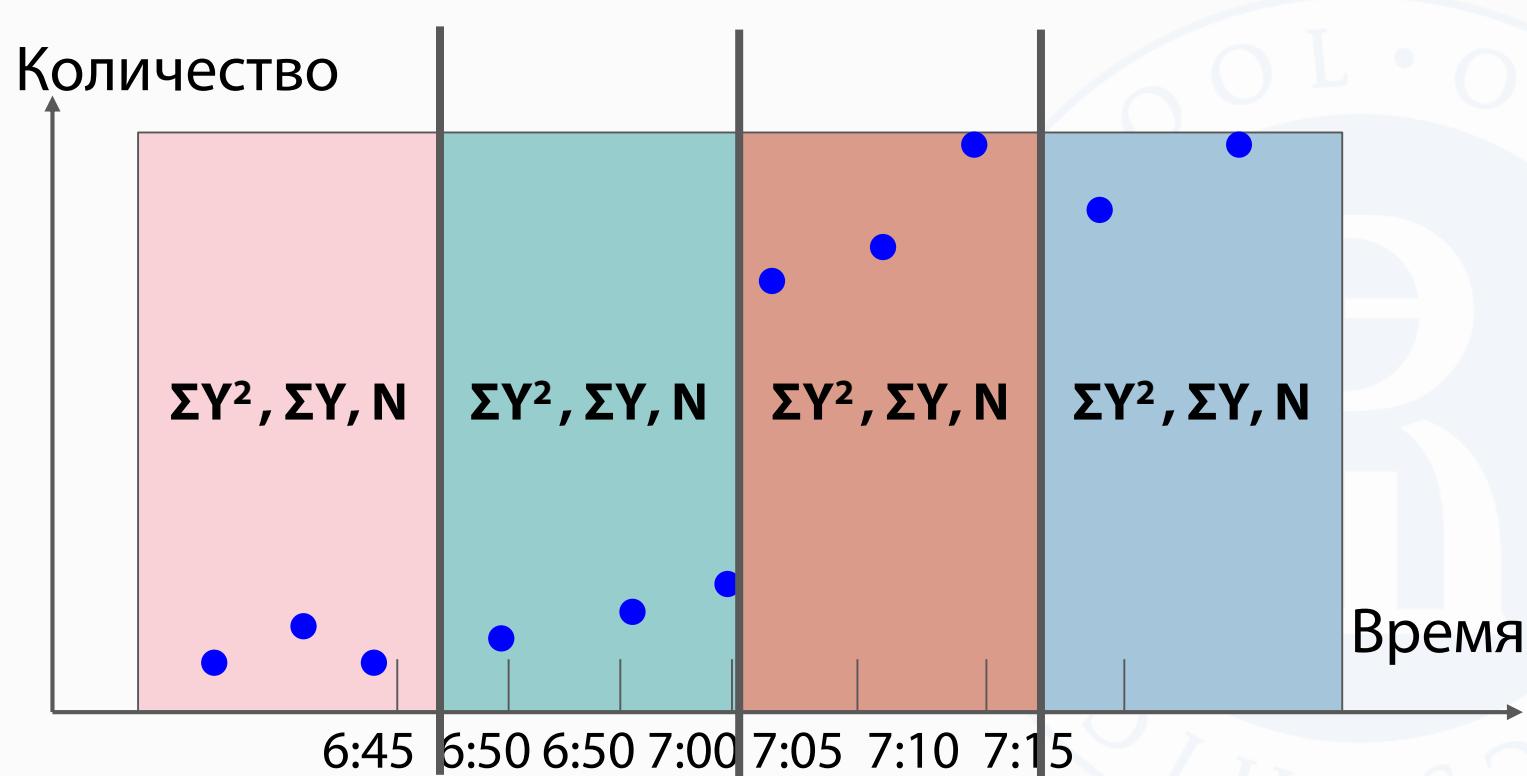
Feature binning

→ Так как мы хотим считать дисперсию, в каждую корзинку положим три значения для элементов корзины:
 ΣY^2 , ΣY и N



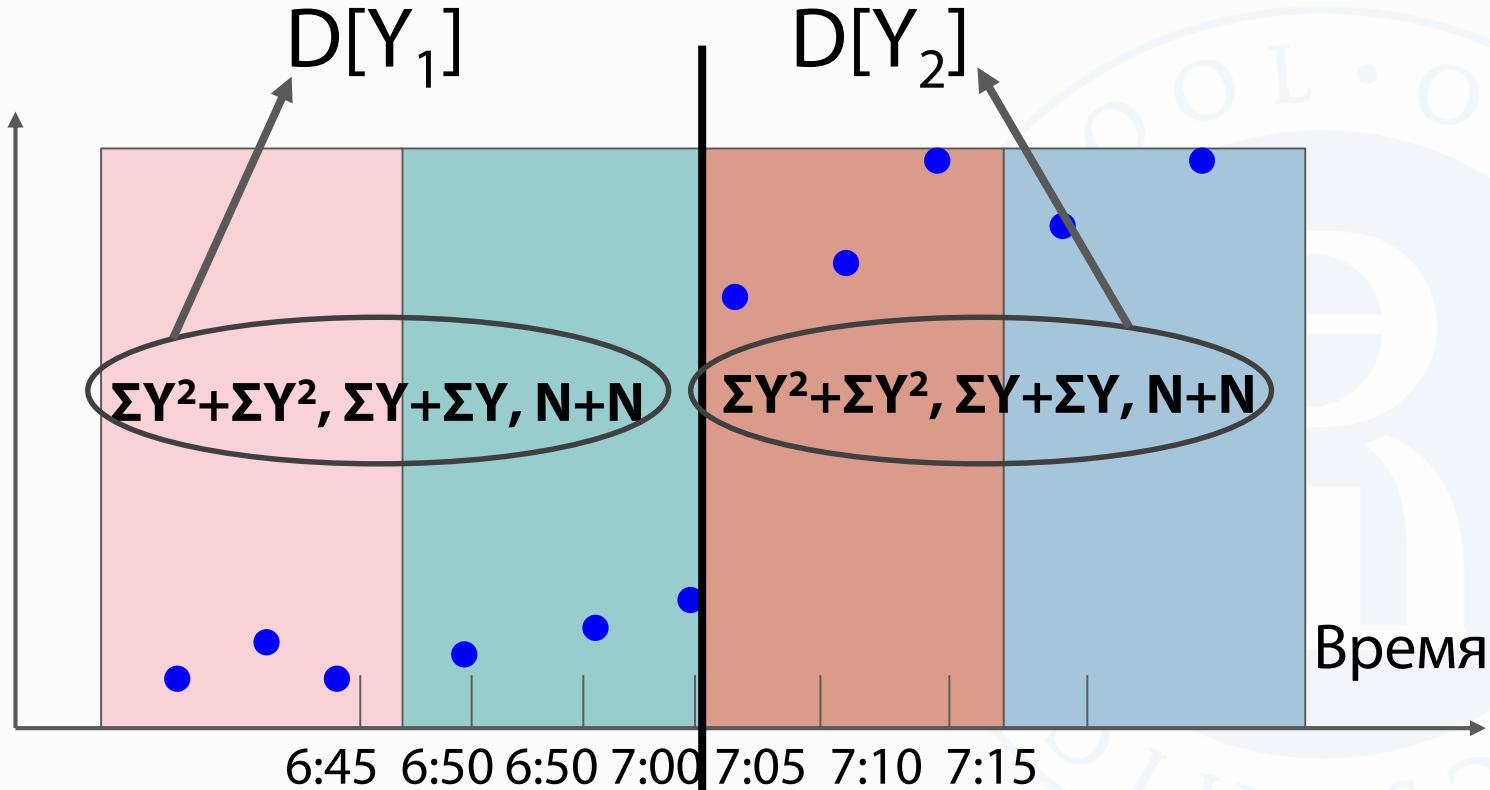
Feature binning

→ Теперь можно перебирать не все значения, а **только «по границам» корзинок**



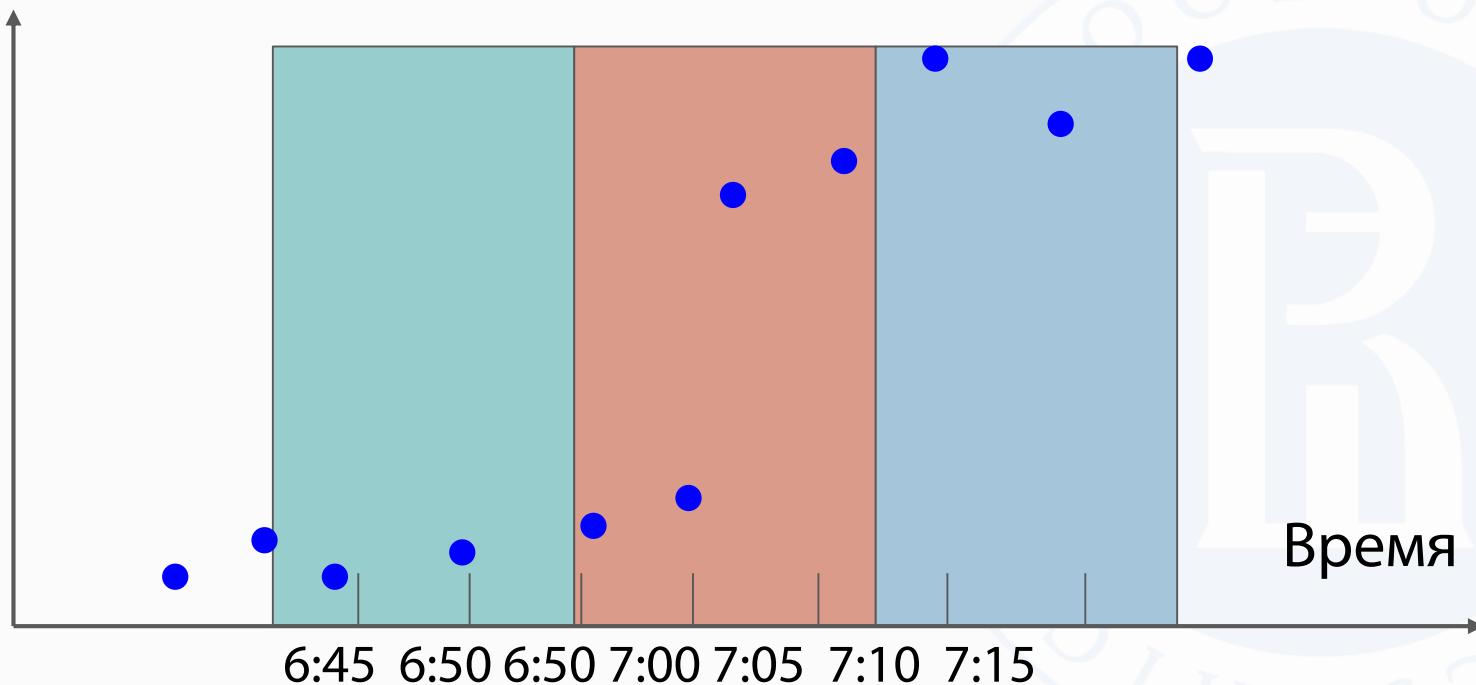
Feature binning

→ Чтобы узнать дисперсию элементов в нескольких корзинках, достаточно сложить три статистики между корзинками и подсчитать по ним дисперсию



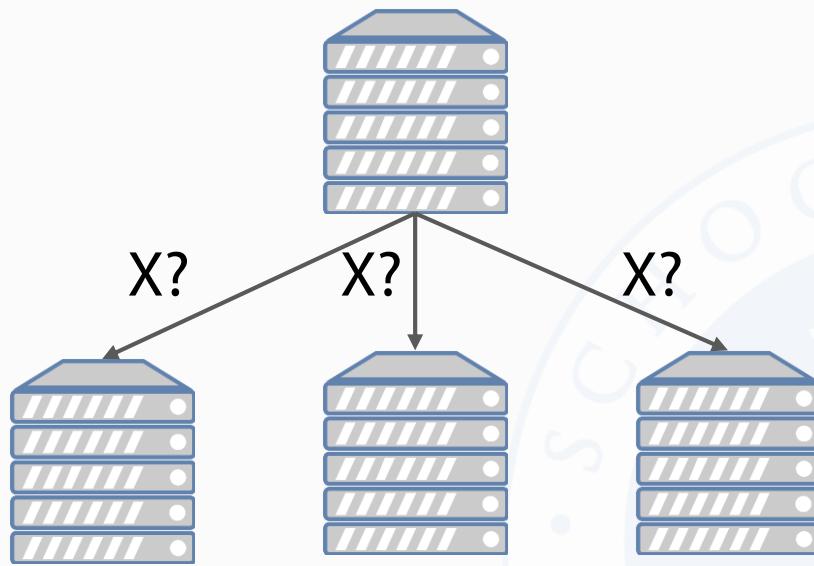
Feature binning

- Такое «сгрубление» данных может немного **ухудшать** качество, если лучшая граница попадет внутрь корзинки, но корзинки можно делать **не слишком широкими**
- Потеря качества компенсируется скоростью обучения



Feature binning

→ 1 — главный сервер говорит, что нужно посчитать разбиение для признака X



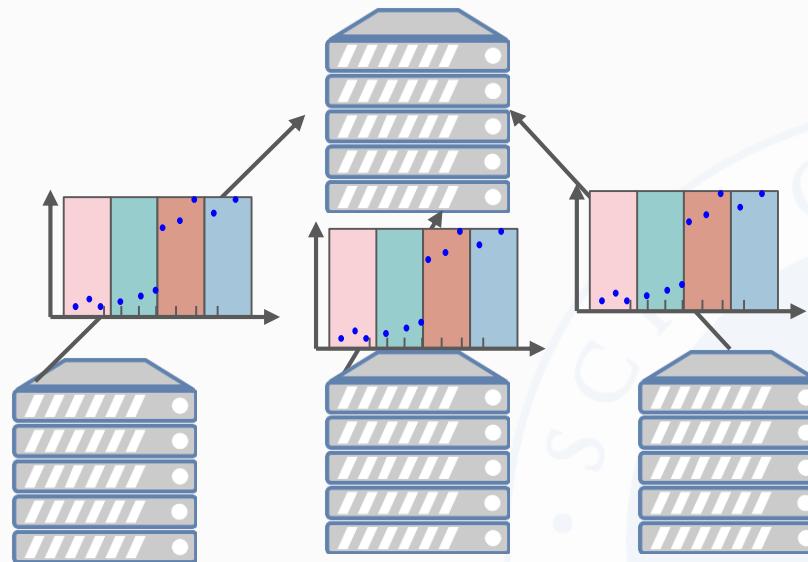
Feature binning

- 2 — каждый рабочий считает корзинки по своим данным.
Важно, что **границы** для корзинок у всех **одинаковые**,
чтобы их можно было объединить



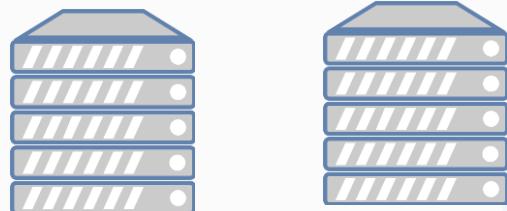
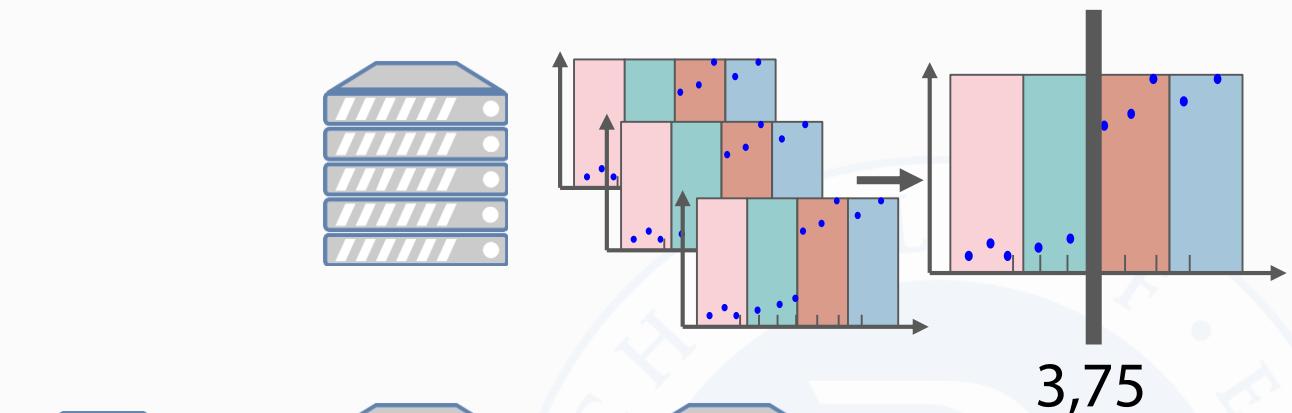
Feature binning

→ 3 — после подсчета рабочие передают в главный сервер
только свои корзинки



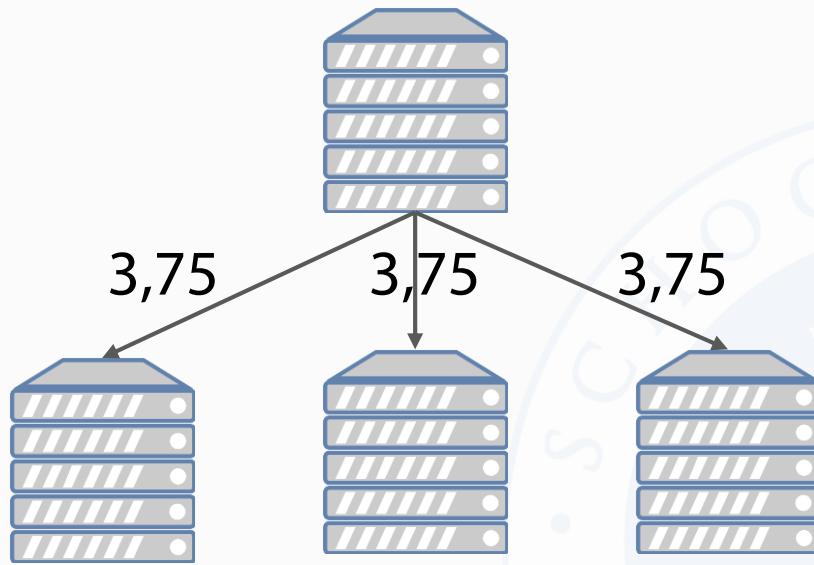
Feature binning

→ 4 — главный сервер складывает все наборы корзинок в один и по нему выбирает лучшее разбиение



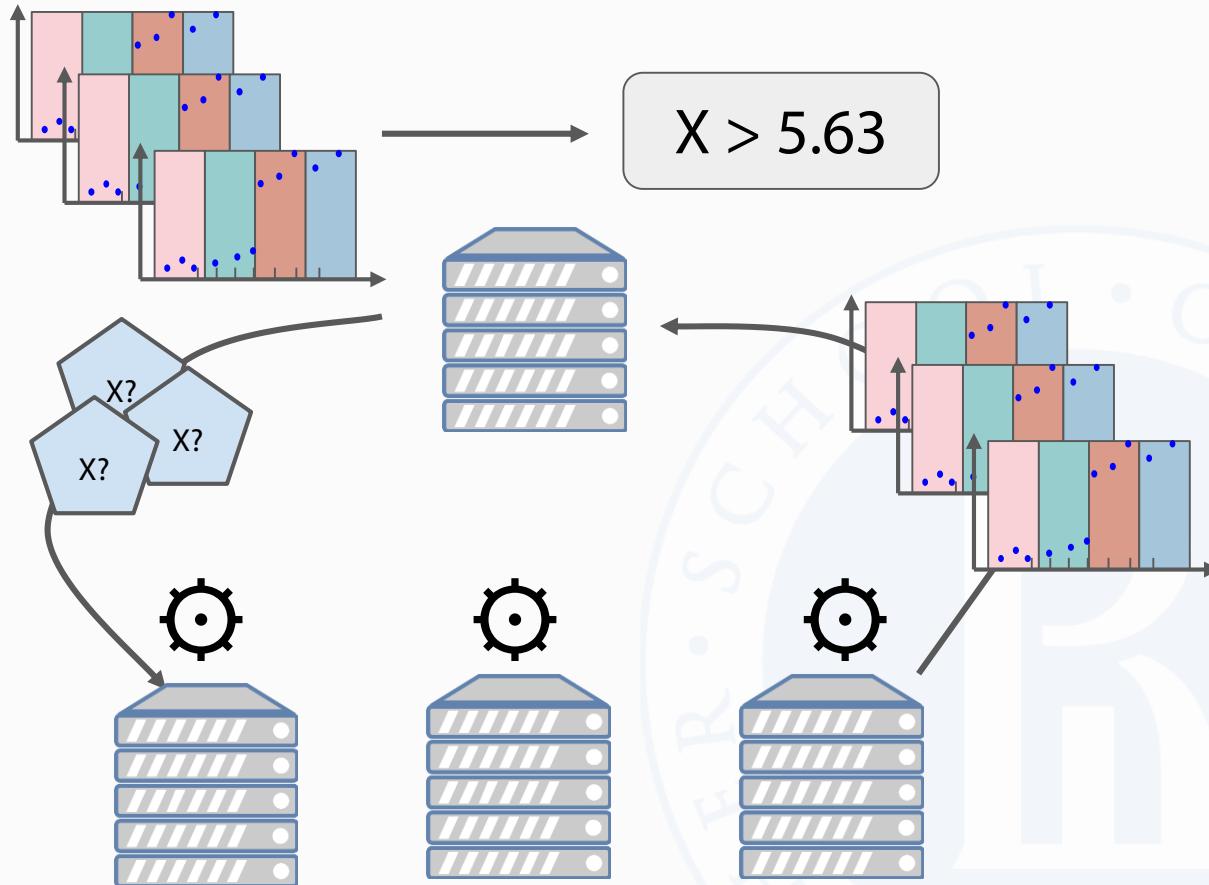
Feature binning

→ 5 — раздает информацию про выбранное разбиение всем машинам



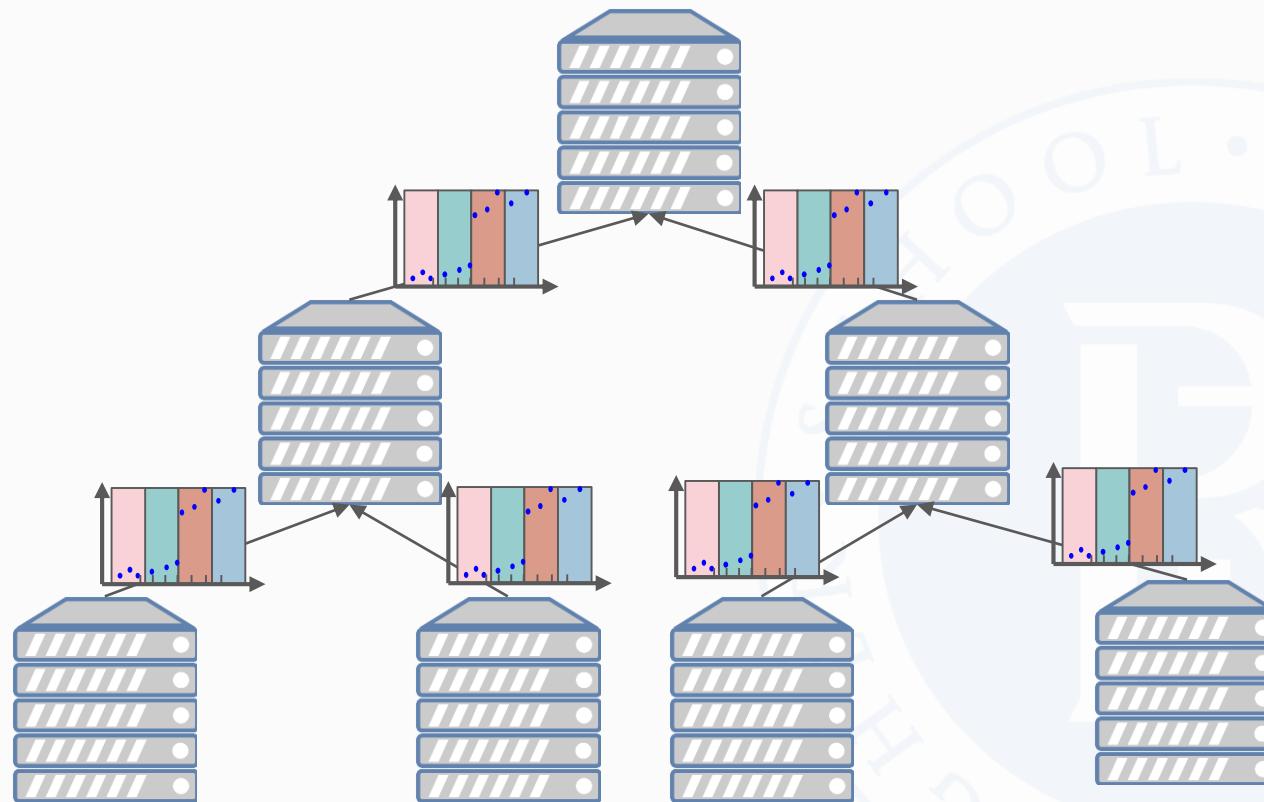
Feature binning

→ Процесс повторяется, пока не будет построено итоговое дерево



Feature binning

→ Отдельно можно заметить, что так как итоговая корзинка строится просто **сложением** значений всех собранных наборов, то машины также можно организовать в дерево и использовать схему **Tree Allreduce**



Распределенное обучение моделей

→ С помощью несложных приемов мы научились эффективно использовать вычислительные ресурсы для распределенного обучения классических моделей машинного обучения

