АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 14

# Локально-чувствительное хеширование

Похожие объекты чаще становятся кандидатами, но кандидатов нужно перепроверять.

14

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 3 Динамическое программирование

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

# Цель и содержание лекции

**ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ**

измерить сходство множеств по Жаккару, построить MinHash-подпись и получить кандидатов на совпадение с помощью локально-чувствительного индекса.

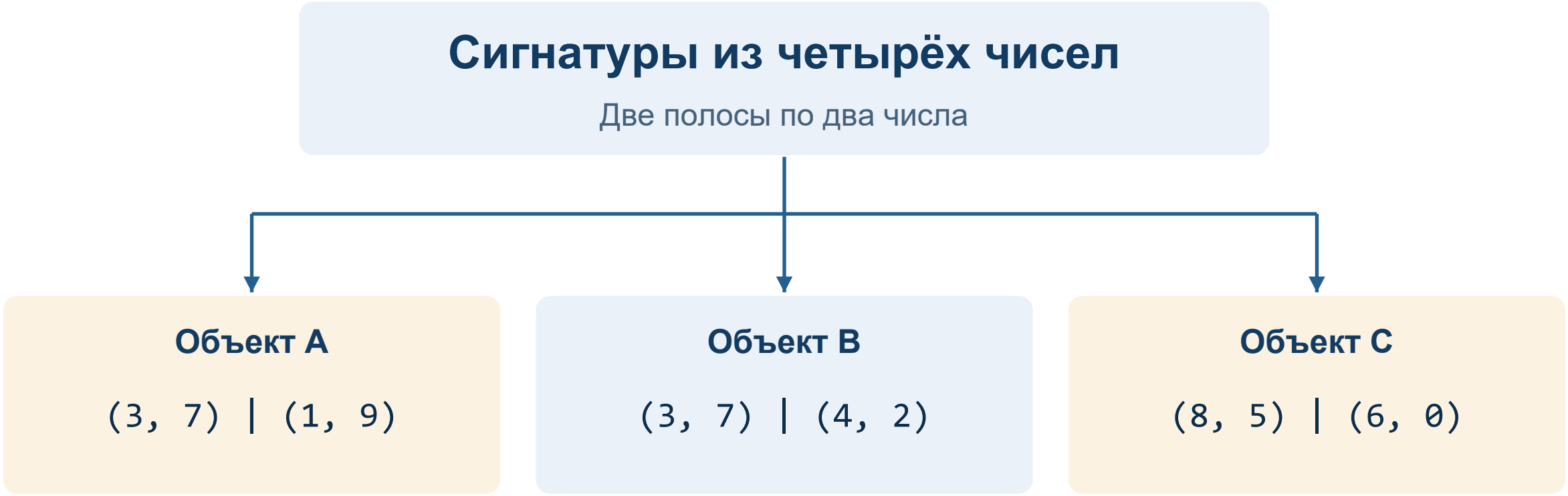
- Задавать меру сходства для выбранного представления.
- Объяснять поиск кандидатов по сигнатурам.
- Различать MinHash и SimHash и учитывать пропуски.

**ПЛАН ЛЕКЦИИ**

1. **Поиск похожих объектов и выбор меры сходства**
2. **MinHash и коэффициент Жаккара**
3. **Разбиение подписи на полосы и поиск кандидатов**
4. **Проверка кандидатов, стоимость поиска и отличие SimHash**

# Совпадение полосы формирует кандидата

СХЕМА




A и B совпали по первой полосе и образуют пару кандидатов. Сходство проверяют на исходных объектах.

# Задача поиска почти дубликатов

01 ТЕМА

Пусть в архиве есть две версии одной инструкции. Во второй исправили опечатку и дописали одно слово. С высокой вероятностью дайджесты SHA-256 различаются, хотя большая часть текста осталась общей. Для поиска таких пар сначала выбирают представление объекта и меру близости. Представим документ множеством нормализованных слов. Можно использовать и множество фрагментов из нескольких последовательных слов, чтобы учитывать локальный порядок. Например, из фразы «дерево хранит ключи» получаются два двухсловных фрагмента «дерево хранит» и «хранит ключи».

Для множеств `A` и `B` коэффициент Жаккара `J(A, B)` равен размеру пересечения, деленному на размер объединения. Если `A = {a, b, c}` и `B = {a, b, d}`, то `J = 2/4 = 0.5`. Для одинаковых непустых множеств значение равно `1`, для непересекающихся — `0`. У двух пустых множеств формула дает деление на ноль, поэтому такие документы обрабатывают по отдельному правилу. В учебном примере все множества непустые.

# Задача поиска почти дубликатов

01 ПРОДОЛЖЕНИЕ

Для одной пары подсчет Жаккара прост. При поиске всех похожих пар среди `n` документов прямой подход выполняет `n(n-1)/2` сравнений. Локально-чувствительное хеширование (LSH) использует случайные функции, которые помещают близкие объекты в одну корзину чаще, чем далекие. Индекс возвращает подмножество документов для подробного сравнения. Понятие «близко» задает конкретная мера сходства.

# Почему работает MinHash



Возьмем общий словарь и случайно переставим все его элементы. Значение MinHash множества — ранг его первого элемента в этой перестановке. Для `A` и `B` хеши совпадут тогда, когда первый элемент объединения окажется в пересечении. Если первый элемент принадлежит только `A`, минимум `A` уже известен, а минимум `B` будет позже. Каждый элемент объединения имеет равную вероятность стать первым. Следовательно, вероятность совпадения двух хешей равна `J(A, B)`.

Одна перестановка дает всего одну проверку на совпадение. Возьмем несколько независимых перестановок и соберем результаты в подпись. Доля совпавших координат служит оценкой Жаккара. Ниже для наглядности вместо рангов указаны сами элементы с минимальным рангом. Дополнительно сравним их с `C = {e, f}`.

# Почему работает MinHash

02 ПРОДОЛЖЕНИЕ

| Перестановка | Минимум A | Минимум B | Минимум C |
|---|---|---|---|
| a, b, c, d, e, f | a | a | e |
| b, d, c, a, f, e | b | b | f |
| c, a, d, b, e, f | c | a | e |
| d, c, b, a, f, e | c | d | f |

Оценка сходства `A` и `B` в таблице равна `2/4 = 0.5`, как и точное значение. Для другой выборки четырех перестановок оценка может отличаться. Увеличение длины подписи снижает случайный разброс, но требует больше времени и памяти. Все документы и запросы нужно обрабатывать в одном порядке перестановок. Иначе одинаковый индекс в двух подписях будет соответствовать разным функциям.

# Полосы подписи и индекс



Даже компактные подписи дорого сравнивать попарно. Разобьем каждую подпись длины `k` на `b` полос по `r` координат, где `k = b·r`. Для каждой полосы строим ключ из ее номера и кортежа значений. Совпадение всех координат хотя бы в одной полосе дает кандидата. В таблице выше при двух полосах по две координаты подписи `A` и `B` совпадают в первой полосе `(a, b)`. Для `C` совпадений с ними нет. Номер полосы в ключе не дает случайно приравнять первую полосу документа к его второй полосе.

Если `s` — точный Жаккар и координаты независимы, шанс совпадения полосы из `r` координат равен `s**r`. Шанс, что пара станет кандидатом хотя бы в одной из `b` полос, равен `1 - (1 - s**r)**b`. Например, при `r = 4` и `b = 8` сходство `0.8` дает вероятность около `0.985`, а сходство `0.2` — около `0.013`. Более длинная полоса требует больше совпадений подряд. Большее число полос увеличивает шанс, что сработает хотя бы одна. Порог точной проверки и настройки индекса задают разные этапы поиска.

# Полосы подписи и индекс

03 ПРИМЕР PYTHON

14_01.py / фрагмент 1 из 3

```
from collections import defaultdict
from random import Random


def signature(tokens, rankings):
    return tuple(min(rank[token] for token in tokens) for rank in rankings)


def band_keys(values, width):
    for start in range(0, len(values), width):
        yield start // width, values[start:start + width]


def jaccard(left, right):
    return len(left & right) / len(left | right)
```

# Полосы подписи и индекс

03 ПРИМЕР PYTHON

14_01.py / фрагмент 2 из 3

```
documents = {
    "B": set("граф поиск вершина ребро очередь".split()),
    "C": set("сигнал частота фаза отсчет спектр".split()),
}
query = set("граф поиск вершина ребро очередь путь".split())
vocabulary = sorted(set.union(query, *documents.values()))
rng = Random(17)
rankings = []
for _ in range(16):
    order = rng.sample(vocabulary, len(vocabulary))
    rankings.append({token: i for i, token in enumerate(order)})

index = defaultdict(set)
for name, tokens in documents.items():
    for key in band_keys(signature(tokens, rankings), width=4):
        index[key].add(name)

candidates = set()
for key in band_keys(signature(query, rankings), width=4):
    candidates.update(index.get(key, ()))
```

# Полосы подписи и индекс

03 ПРИМЕР PYTHON

14_01.py / фрагмент 3 из 3

```
for name in sorted(candidates):
    score = jaccard(query, documents[name])
    if score >= 0.7:
        print(name, f"{score:.3f}")
```

РЕЗУЛЬТАТ ЗАПУСКА

```
B 0.833
```

# Полосы подписи и индекс

03 ПРОДОЛЖЕНИЕ

Документ `B` содержит пять из шести слов запроса и никаких дополнительных. Точный результат равен `5/6`. Для воспроизводимости словарь отсортирован, а генератор инициализирован известным числом. В учебной программе словарь строится по всему маленькому набору. В готовом индексе для произвольных новых слов нужна общая воспроизводимая схема присвоения рангов. Простая замена перестановок произвольными хешами требует проверить качество приближения и влияние коллизий.

# Что гарантирует и чего стоит поиск



Для $k$ явных перестановок словаря размера $U$ нужно $O(kU)$ времени и памяти. Подпись документа из $s$ уникальных токенов вычисляется за $O(ks)$. Для $n$ документов индекс содержит $nb$ ссылок на документы и до $O(nk)$ значений в ключах полос. При запросе время также зависит от числа совпавших записей в корзинах и числа кандидатов, которым нужно посчитать точное сходство. Если большая часть коллекции попадает в одни и те же корзины, проверка может приблизиться к полному перебору.

Точная проверка удаляет кандидатов со сходством ниже порога. Пара с высоким сходством может отсутствовать среди кандидатов, если полосы не совпали. Чтобы оценить параметры, на небольшой выборке можно посчитать истинные пары полным перебором и измерить долю таких пар, найденную индексом. Если индекс возвращает почти все документы, последующая проверка потребует почти столько же времени, сколько полный перебор.

# Что гарантирует и чего стоит поиск

04 ПРОДОЛЖЕНИЕ

В учебнике для близких текстов приведен SimHash. Он строит битовую подпись вектора признаков. В геометрической формулировке один бит показывает, по какую сторону случайной гиперплоскости находится вектор. Доля различающихся битов приближает угол между векторами, деленный на π. Различия между битовыми подписями считают по расстоянию Хэмминга. MinHash использует минимум по перестановке и связан с Жаккаром множеств. Оба подхода локально-чувствительны, а выбор зависит от представления объектов и нужной меры близости.

# Что нужно уметь после лекции

ИТОГИ

- Для равномерной случайной перестановки вероятность совпадения MinHash равна сходству Жаккара.
- Полосы позволяют искать кандидатов по ключу без попарного сравнения всех подписей.
- Точный Жаккар уточняет результат для найденных кандидатов. SimHash работает с векторами и битовыми подписями.

> Похожие объекты чаще становятся кандидатами, но кандидатов нужно перепроверять.

# Проверьте понимание

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Почему для MinHash важно, лежит ли первый элемент объединения в пересечении?
2. Для чего номер полосы входит в ключ индекса?
3. Может ли точная проверка найти пару, отсутствующую среди кандидатов?
4. Как выбор признаков влияет на смысл близости в MinHash и SimHash?

# Самостоятельная работа

**ПРАКТИКА**

1. Для четырех перестановок из таблицы найдите MinHash-подпись множества `{a, e, f}`. Сравните ее с подписями `A` и `C`.
2. Для подписи из `16` координат сравните вероятность попадания в кандидаты при `s = 0.5` для схем `8 × 2` и `4 × 4`, где первое число — количество полос.
3. Добавьте три документа и сравните кандидатов при ширине полосы `2`, `4` и `8`. Объясните, какие пары отсеялись при точной проверке.

# Источники и документация

## ЧТЕНИЕ

### ОСНОВНОЕ ЧТЕНИЕ

**А. Бхаргава. Грокаем алгоритмы**

СПб.: Питер, 2017. Глава 11. «Локально-чувствительное хеширование». С. 269–270.

### ДОПОЛНИТЕЛЬНО


Broder A. Z. On the Resemblance and Containment of Documents. https://doi.org/10.1109/SEQUEN.1997.666900

Charikar M. S. Similarity Estimation Techniques from Rounding Algorithms. https://doi.org/10.1145/509907.509965