清华大学数据库技术与应用

数据准备 II

授课教师：计算机系王健楠

授课学期：2026年（春季）

清華大學 Tsinghua University

回顾

清華大學 Tsinghua University

01 数据准备概述

- 为什么数据准备难?
- Workflow GUI vs Spreadsheet GUI vs Notebook GUI

02 数据准备任务

- 数据收集
- 清洗与EDA
- 数据集成
- 正则表达式

03 面向大语言模型 (LLM) 的数据准备

今天课程内容

回顾

清華大學 Tsinghua University

01 数据准备概述

- 为什么数据准备难?
- Workflow GUI vs Spreadsheet GUI vs Notebook GUI

02 数据准备任务

- 数据收集
- 清洗与EDA
- 数据集成
- **正则表达式**

03 面向大语言模型 (LLM) 的数据准备

提纲

1 为什么需要正则表达式

2 正则语法：从字符到模式

3 Python 实战

4 正则引擎与性能

5 正则 × 大模型

提纲

任务：从杂乱文本中提取结构化数据

你拿到一批用户反馈，需要提取其中的手机号、日期和金额：

```
"请在2024-03-15之前把¥599退回到我的账户，我的电话是138-0013-2456"
"2024/4/1打过客服电话 13912345678，说好退¥1200，到现在没处理"
"投诉！手机号13800001111，订单金额¥89.9，日期2024.05.20"
```

难点：格式不统一

- 手机号： 13912345678 / 138-0013-2456（有无横线）
- 日期：2024-03-15 / 2024/4/1 / 2024.05.20（分隔符不同，位数不同）
- 金额：¥599 / ¥1200 / ¥89.9（整数或小数）

用字符串方法，你会怎么做？

字符串方法 vs 正则表达式

字符串方法：逐格式处理，代码冗长

```
# 提取手机号（仅两种格式）
for word in text.split():
  cleaned = word.replace("-","")
  if len(cleaned)==11 \
     and cleaned.startswith("1") \
     and cleaned[1] in "3456789" \
     and cleaned.isdigit():
    phones.append(cleaned)
# 日期呢？金额呢？每种都要写
```

正则表达式：描述模式，一行搞定

```
import re

# 手机号（自动处理横线）
re.findall(r"1[3-9][\d-]{9,11}\d",
           text)

# 日期（三种分隔符）
re.findall(r"\d{4}[-/.]\d{1,2}[-/.]\d{1,2}",
           text)

# 金额
re.findall(r"[¥]\d+\.?\d*", text)
```

核心区别：字符串方法描述**怎么找**（过程），正则描述**长什么样**（模式）

- 格式稍有变化，字符串方法就要改代码；正则只需调整模式
- 数据量越大、格式越多样，正则的优势越明显

提纲

正则语法全貌

正则表达式 = 四类基本构件的组合

1. 字符匹配	2. 数量控制	3. 位置锚定	4. 分组与逻辑
匹配哪些字符	重复多少次	出现在哪里	结构与选择
a . [0-9] \d	* + ? {n,m}	^ $ \b	() | (?:)

例子：验证社会安全号 (SSN)

匹配 "231-41-5121"　　**整体: 3个数字 + 横线 + 2个数字 + 横线 + 4个数字**

拆解：

```
r"[0-9]{3}-[0-9]{2}-[0-9]{4}"

[0-9]    字符匹配 — 匹配一个数字
{3}      数量控制 — 重复3次
-        字面量  — 匹配短横线本身
```

注意：模式前加 r（raw string），避免 Python 转义和正则转义冲突

```
r"\b+"    ✓ 反斜杠直接传给正则引擎
"\b+"     ✗ Python先处理 \b（不认识，可能出错或静默变化）

经验：正则模式永远用 r"..." 前缀
```

字符匹配（1）—— 字面量与通配符

问题：如何在文本中找到 cat 这个词？

答案：直接写 **cat** —— 字面量字符精确匹配自己

```
r"cat"
```

文本 **"a cat sat"** ✓ **匹配 cat**

文本 **"a Cat sat"** ✗ **大小写不同**

进一步：如果某个字符不确定呢？—— 用 . 通配符

- 匹配任意一个字符（除换行符）

```
r"c.t"
```

文本 **"cat"** ✓ **. 匹配 a**

文本 **"c2t"** ✓ **. 匹配 2**

文本 **"c t"** ✓ **. 匹配 空格**

文本 **"ct"** ✗ **缺少中间字符**

字符匹配（2）—— 字符集合 [...]

问题：只想匹配 cat 或 bat，不要 sat?

[cb] —— 从集合中任选一个字符

```
r"[cb]at"
```

"cat" ✓ [cb] 匹配 c
"bat" ✓ [cb] 匹配 b
"sat" ✗ s 不在 [cb] 中

范围简写：[a-z] 表示 a 到 z 任意小写字母

```
r"[a-z]at"
```

"cat" ✓ c 在 a-z 范围内
"hat" ✓ h 在 a-z 范围内
"Cat" ✗ C 是大写，不在 a-z 中

取反：[^cb] 表示“不是 c 也不是 b”的任意字符

```
r"[^cb]at"
```

"sat" ✓ s 不是 c/b
"cat" ✗ c 在排除集中

规律：[abc] 任选一个 | [a-z] 范围 | [^abc] 取反。方括号内永远只匹配一个字符！

字符匹配（3）—— 常用字符类缩写

问题：每次写 [A-Za-z0-9_] 太麻烦？—— 用缩写！

\w = [A-Za-z0-9_]

单词字符

\W = [^A-Za-z0-9_] 非单词

```
r"\w+"
```

"hello_1" ✓

"!@#" ✗

\d = [0-9]

任意数字

\D = [^0-9] 非数字

```
r"\d\d\d"
```

"123" ✓

"abc" ✗

\s = [\t\n\r]

空白字符

\S = [^ \t\n\r] 非空白

```
r"\S+"
```

"hello" ✓

" " ✗

实战示例：匹配电话号码 138-0013-2456

```
r"\d{3}-\d{4}-\d{4}"   ← \d 匹配数字，{3} 是数量控制（下一节讲）
```

小结：字面量 (a) → 通配符 (.) → 字符集合 ([]) → 缩写 (\d \w \s)。

字符匹配（4）—— 转义特殊字符

问题：如何匹配 3.14 中的点号？直接写 . 会匹配任意字符！

答案：在特殊字符前加 **** 就变成字面量

```
r"3\.14"
```

文本 **"3.14"** ✓ **\. 匹配真正的点**

文本 **"3a14"** ✗ **a 不是点**

对比：不转义的 . 是通配符

```
r"3.14"
```

文本 **"3.14"** ✓ **. 匹配 .**

文本 **"3a14"** ✓ **. 也匹配 a（不是我们想要的）**

需要转义的特殊字符：

```
. * + ? ^ $ { } [ ] ( ) | \    ← 在前面加 \ 就变成字面量
```

数量控制（1）—— 基本量词？ + *

问题：如何同时匹配 color 和 colour?

? = 前面的字符出现 0 或 1 次（可有可无）

```
r"colo?r"
```

"color" ✓ **u 出现 0 次**
"colour" ✓ **u 出现 1 次**

进一步：如何匹配一个或多个连续数字?

+ = 前面的字符出现 1 次或多次（至少一次）

```
r"\d+"
```

"9" ✓ **1个数字**
"123" ✓ **3个数字**
"abc" ✗ **没有数字**

再进一步：0 次也行呢?

***** = 前面的字符出现 0 次或多次（可以没有）

```
r"ab*c"
```

"ac" ✓ **b 出现 0 次**
"abc" ✓ **b 出现 1 次**
"abbc" ✓ **b 出现 2 次**

数量控制（2）—— 精确量词与贪婪模式

精确次数：如何匹配恰好 4 位数字（如年份）？

{n} 恰好 n 次　　**{n,m}** n 到 m 次　　**{n,}** 至少 n 次

r"\d{4}"　　**"2024" ✓ 恰好 4 位**

"123" ✗ 只有 3 位

r"\d{1,3}"　　**"5" "42" "100" ✓ 1到3位都行**

贪婪 vs 懒惰：量词默认贪婪（尽可能多匹配）

文本："name: Alice and Bob"

贪婪　r".*"　→ "**Alice and Bob**" 匹配到最后一个

懒惰　r".*?"　→ "**Alice**" 和 "**Bob**" 最近的

规则：量词后加 **?** 切换为懒惰模式：***?　+?　??　{n,m}?**

小结：？（0/1次）　+（1+次）　*（0+次）　{n,m}（精确）。默认贪婪，加？变懒惰。

位置锚定 —— 限定匹配出现的位置

问题：如何只匹配行首的 Hello，不要中间出现的？

^ = 行首（字符串开头）

```
r"^Hello"
```

"Hello world" ✓ **Hello 在行首**
"Say Hello" ✗ **Hello 不在行首**

$ = 行尾（字符串结尾）

```
r"end$"
```

"the end" ✓ **end 在行尾**
"endless" ✗ **end 后面还有字符**

\b = 单词边界（单词字符与非单词字符之间）

```
r"\bcat\b"
```

"a cat sat" ✓ **cat 是独立单词**
"catch" ✗ **cat 是 catch 的一部分**

组合示例：匹配以数字结尾的行

```
r"\d+$"    ← \d+ 一个或多个数字 + $ 必须在行尾
```

分组与逻辑（1）—— 分组与选择

问题：如何匹配 "cat" 或 "dog"？

| = 或（选择其一）

```
r"cat|dog"
```

"I have a cat" ✓ **匹配 cat**
"I have a dog" ✓ **匹配 dog**
"I have a bird" ✗ **既不是 cat 也不是 dog**

进一步：如何匹配 highway 或 subway？

() = 分组，把多个字符当作一个整体

```
r"(high|sub)way"
```

"highway" ✓ **high + way**
"subway" ✓ **sub + way**
"myway" ✗ **my 不在选项中**

注意优先级：分组还能改变运算顺序

AB* → A + 0或多个B → A, AB, ABB...　　**(AB)*** → 0或多个AB → ε, AB, ABAB...

规律：| 做选择，() 做分组。优先级：() > 量词 > 连接 > |

分组与逻辑（2）—— 捕获组

问题：不仅要匹配日期，还要分别取出年、月、日？

() 除了分组，还会自动**捕获**匹配到的内容，供后续提取

```
r"(\d{4})-(\d{2})-(\d{2})"
```

第1组　　第2组　　第3组

匹配 "2024-03-15"：

group(1) → **"2024"**　　group(2) → **"03"**　　group(3) → **"15"**

实战示例：从日志中提取时间和级别

文本："2024-03-15 10:30:22 ERROR: Connection timeout"

```
r"(\d{4}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2})\s+(\w+): (.+)"
```

group(1)="2024-03-15" group(2)="10:30:22" group(3)="ERROR" group(4)="Connection timeout"

补充：如果只想分组不想捕获？用 **(?:...)** 非捕获组

小结：() 同时实现分组和捕获。用 group(N) 提取第 N 对括号的内容。(?:) 只分组不捕获。

课堂练习

综合运用字符匹配、数量控制、位置锚定、分组与逻辑：

Q1：匹配形如 “2024-03-15 09:30” 的日期时间

测试: "会议从 2024-03-15 09:30 开始，到 2024-03-15 11:00 结束"

提示：想想需要哪几种构件？数字用什么匹配？横线和冒号怎么处理？日期和时间之间是什么？

Q2：匹配文本中的所有价格（人民币 ¥ 或美元 $，可能有小数）

测试: "苹果 ¥8.5/斤，进口车厘子 $12.99/磅，香蕉 ¥3/斤"

提示：货币符号怎么匹配？整数部分？小数部分是可选的，怎么表达“有或没有”？

练习答案

```
# Q1: 日期时间 — 数量控制 + 字面量
r"\d{4}-\d{2}-\d{2} \d{2}:\d{2}"
# \d{4} 年 + - + \d{2} 月 + - + \d{2} 日 + 空格 + \d{2} 时 + : + \d{2} 分
```

```
# Q2: 价格 — 字符集合 + 转义 + 数量 + 分组
r"[¥$]\d+(\.\d+)?"
# [¥$] 货币符号 + \d+ 整数 + (\.\d+)? 可选小数部分
```

提纲

第一个任务：从文本中提取信息

💡 你已经会写正则了，但怎么在 Python 中执行呢？——用 re 模块

任务：从杂乱文本中提取所有手机号

```
import re

text = "张三 138-0013-2456, 李四 13912345678"

# re.findall(模式, 文本) → 返回所有匹配的列表
phones = re.findall(r"1[3-9][\d-]{9,11}", text)

print(phones)
```

输出: **['138-0013-2456', '13912345678']**

re.findall 返回所有匹配结果的列表。只想找第一个？用 **re.search**（返回 Match 对象）

Python re 模块

方法	功能	返回值
`re.search(p, s)`	找第一个匹配	Match 对象 / None
`re.match(p, s)`	从开头匹配	Match 对象 / None
`re.findall(p, s)`	找所有匹配	列表
`re.sub(p, repl, s)`	替换所有匹配	新字符串
`re.split(p, s)`	按模式分割	列表
`re.compile(p)`	预编译（提高性能）	Pattern 对象

更多例子：替换与分割

1：**身份证号脱敏**——只显示前6位和后4位，剩下的用*表示

```
id_text = "身份证: 110101199001011234"

# re.sub(模式, 替换, 文本) → 返回替换后的新字符串
result = re.sub(r"(\d{6})\d{8}(\d{4})", r"\1********\2", id_text)
```

输出: **'身份证: 110101********1234'**

2：按多种标点分割句子

```
s_text = "你好，世界；Hello,World！"
# re.split(模式, 文本) → 返回分割后的列表
re.split(r"[,，；。！]+", s_text)
```

输出: **['你好', '世界', 'Hello', 'World', '']**

数据量大了怎么办？—— Pandas .str

💡 100 万行文本要提取日期，逐行 re 太慢？

逐行循环（慢、冗长）

```
dates = []
for text in df["text"]:
    m = re.search(
        r"(\d{4}-\d{2}-\d{2})",
        text)
    dates.append(m...)
```

→

Pandas .str（快、简洁）

```
# 一行搞定，自动处理 NaN
df["date"] =
    df["text"]
    .str.extract(
        r"(\d{4}-\d{2}-\d{2})"
    )
```

原理：Pandas .str 做了什么

- 预编译正则：内部自动调用 re.compile()，避免重复编译
- 批量调度：在 C/Cython 层循环调用已编译的正则，减少 Python 解释器开销
- 统一缺失值处理：遇到 NaN 自动跳过，不需要手动 if 判断
- 结果自动对齐：extract() 直接返回 DataFrame，列名 = 捕获组编号

总结：Python 正则工具箱

re 模块 （单条文本）

re.findall(p, s)
所有匹配 → 列表
re.search(p, s)
第一个匹配 → Match
re.sub(p, r, s)
替换 → 新字符串
re.split(p, s)
分割 → 列表
re.compile(p)
预编译（重复使用时）

Pandas .str （批量 DataFrame）

.str.findall(p)
所有匹配 → 列表列
.str.extract(p)
捕获组 → DataFrame
.str.replace(p, r)
替换 → 新 Series
.str.split(p)
分割 → 列表列
.str.contains(p)
是否匹配 → 布尔列

常用正则模式速查

手机号 r"1[3-9]\d{9}"

日期 r"\d{4}[-/.]\d{1,2}[-/.]\d{1,2}"

邮箱 r"[\w.+-]+@[\w-]+\.[\w.]+"

URL r"https?://\S+"

中文 r"[\u4e00-\u9fff]+"

HTML标签 r"<[^>]+>"

在线调试：regex101.com — 选 Python 模式，实时测试正则表达式。

Python re模块 vs Pandas.str

	re 模块（单条文本）	Pandas .str（向量化）
查找所有匹配	re.findall(p, s)	df.col.str.findall(p)
查找第一个匹配	re.search(p, s)	df.col.str.extract(p)
替换	re.sub(p, repl, s)	df.col.str.replace(p, repl)
分割	re.split(p, s)	df.col.str.split(p)
判断是否匹配	bool(re.search(p, s))	df.col.str.contains(p)
适用场景	单条文本、脚本处理	大规模表格数据（DataFrame）
性能特点	灵活，逐条处理	向量化执行，批量快数倍

课堂练习

给定以下订单 DataFrame，用 Pandas .str 方法完成数据清洗：

```
import pandas as pd
df = pd.DataFrame({
    "info": ["ORD-2024-001 张三 138-0013-8888 ¥2,580.00",
             "ORD-2024-002 李四 13912345678 ¥399.90",
             "ORD-2024-003 王五 155-2233-4455 ¥1,024.50"]
})
```

任务：从 info 列中提取订单号、手机号、金额，分别放入三个新列

期望输出：

order_id	phone	amount
ORD-2024-001	138-0013-8888	￥2,580.00
ORD-2024-002	13912345678	$399.90
ORD-2024-003	155-2233-4455	$1,024.50

练习答案

用 .str.extract 配合捕获组提取三个字段

```
# 提取订单号：ORD- + 数字和连字符
df["order_id"] = df.info.str.extract(r"(ORD-[\d-]+)")

# 提取手机号：1开头的9-11位数字（允许连字符）
df["phone"]  = df.info.str.extract(r"(1[3-9][\d-]{9,11})")

# 提取金额：¥后的数字（含逗号和小数点）
df["amount"]  = df.info.str.extract(r"¥([\d,.]+)")
```

提纲

正则匹配的性能问题

- **哪个更慢?**

模式 A

```
pattern = r"a{25}b"
text    = "a" * 25 + "c"
re.match(pattern, text)
```

模式 B

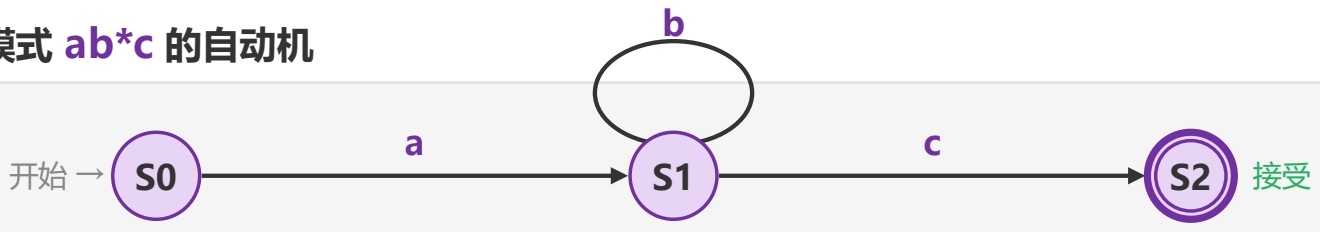
```
pattern = r"(a+)+b"
text    = "a" * 25 + "c"
re.match(pattern, text)
```

- **区别**：仅将 a{25} 改为 (a+)+，即嵌套量词
- **原因**：涉及正则引擎的工作原理 —— 有限自动机与回溯机制

正则引擎的工作原理：有限自动机

- 每个正则表达式会被编译为一个**有限状态自动机（Finite Automaton）**
- 自动机 = 状态集合 + 转移规则：每读入一个字符，根据规则转移到下一个状态
- 读完所有字符后，若处于接受状态 → 匹配成功；否则失败

示例：模式 ab*c 的自动机



匹配过程追踪：

"abbc" S0 –a→ S1 –b→ S1 –b→ S1 –c→ **S2** ✓ 到达接受状态，匹配成功

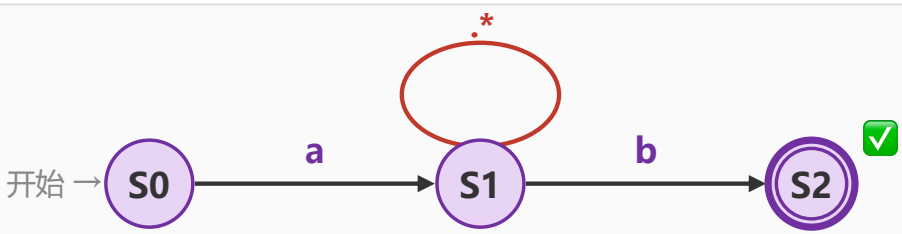
"ad" S0 –a→ S1 –d→ **?** ✗ 无对应转移规则，匹配失败

关键特征：当每个状态对每个输入只有唯一转移时，匹配过程是确定的，效率很高

非确定性转移与回溯机制

- 当模式包含 **.*** | **+** **?** 等量词时，同一状态可能有多条转移路径
- 引擎需要选择一条路径尝试，若失败则退回尝试另一条 —— 这就是回溯（Backtracking）

示例：模式 a.*b 的自动机



S1 可以走 .* 继续吃字符，也可以走 b 尝试结束

匹配文本 **"axxc"**

回溯过程：

1. a 匹配 a，.* 贪婪吃完 xxc
2. 模式剩 b，文本已尽 → 回溯，吐出 c
3. c ≠ b → 再吐 x，x ≠ b → 再吐 x
4. 所有路径穷尽 → 匹配失败 ✗

- **回溯的代价**：文本较短、路径较少时，回溯开销可以忽略

模式	路径特征	回溯规模
abc	确定，无分支	0
a|b	单个分支点	O(1)
a.*b	每个字符产生分支	O(n)
(a+)+b	嵌套量词，分支指数增长	$O(2^n)$

灾难性回溯：指数级回溯的成因

- 嵌套量词 **(a+)+** 使得同一段文本存在指数级别的分组方式。为什么？

核心观察： n 个 a 之间有 n−1 个“缝隙”，每个缝隙可以选择“切开”或“不切”—— 两种选择

以 5 个 a 为例：4 个缝隙，每个缝隙 2 种选择 = 2^4 = 16 种分组

4 个缝隙，每个 2 种选择
= 2 × 2 × 2 × 2 = 2^4 = **16**

举例：

(aaaaa)	全不切 → 1 组
(aaa)(aa)	切第3个
(a)(aaa)(a)	切第1、4个
(a)(a)(a)(a)(a)	全切 → 5 组

一般规律： n 个字符 → n−1 个缝隙 → **2^{n-1} 种分组方式**，匹配失败时每种都要试一遍

- **安全威胁：** 攻击者可构造特定输入触发指数回溯，称为 **ReDoS**（Regular Expression Denial of Service）

如何避免正则性能陷阱

- **规则 1：避免嵌套量词**

  ❌ (a+)+ (\d+)* → ✅ a+ \d+

- **规则 2：避免重叠的选择支**

  ❌ (a|a+) → ✅ a+

- **规则 3：使用工具检测性能风险**
  - regex101.com — 在线调试，可检测性能警告
  - google/re2 — DFA 引擎，保证线性时间复杂度

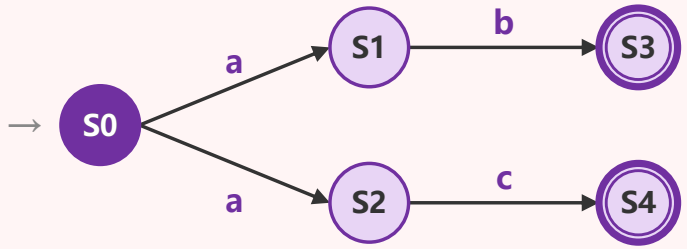
NFA 与 DFA：两种引擎的本质区别

- 前面介绍的“多路径尝试 + 回溯”是 **NFA（非确定性有限自动机）** 的工作方式
- 另一种引擎是 **DFA（确定性有限自动机）**：构建时已消除所有非确定性，每步转移唯一，因此不需回溯

	NFA（非确定型）	DFA（确定型）
核心特征	同一输入可转移到多个状态	同一输入转移唯一确定
匹配方式	逐条尝试路径，失败则回溯	每步唯一转移，无回溯
最坏时间	可能指数级 $O(2^n)$	始终线性 O(n)
功能支持	反向引用、环视等高级功能	仅标准正则语言（无反向引用）
典型实现	`Python re, Java, JavaScript`	`Google RE2, Rust regex, Go`

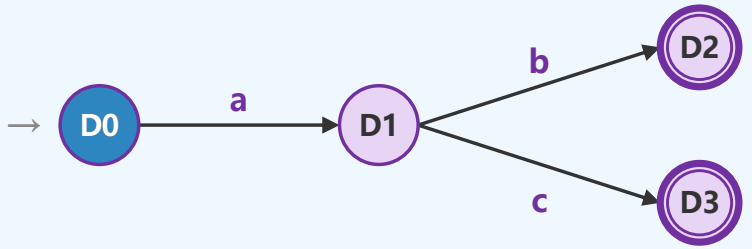
示例模式：ab|ac

NFA



输入 a 时，两条路径 → 需逐条尝试，失败则回溯

DFA



输入 a 时，唯一路径 → 直接前进，无需回溯

提纲

大模型如何辅助正则

正则表达式的主要困难：语法难写、难读、难调试。大模型可以显著降低这些门槛。

① 自然语言生成正则

- **描述需求：** “匹配中国手机号，允许中间有连字符” → LLM 直接输出正则
- **处理复杂模式：** 多条件组合、特殊格式等，比手写效率高得多

② 解释已有正则

- 接手别人的代码，遇到复杂正则看不懂 → 让 LLM 逐段解读

③ 调试与优化

- “这条正则没匹配到“xx”，帮我检查哪里写错了”
- LLM 还可以识别性能风险（如嵌套量词），建议更安全的写法

但是： LLM 生成的正则不一定正确。你必须看懂正则语法，才能验证和修正 LLM 的输出。
“用 LLM 生成，用正则知识审查”是最佳实践。

正则如何辅助大模型

LLM 调用成本高、速度慢、输出不稳定。将确定性任务交给正则，可以大幅提升整体效率。

① 预处理：减少输入成本

- 用正则预先提取关键片段，只将相关内容发给 LLM，减少 token 消耗
- 示例：10 万条日志中，先用 re.findall 筛出含报错的行，再用 LLM 分析原因

② 后处理：结构化 LLM 输出

- LLM 输出是自由文本，用正则提取其中的结构化信息（如 JSON、数字、日期）
- 示例：LLM 返回“该订单金额为 128.50 元”，用 re.search(r"\d+\.\d+") 提取数值

③ 替代：确定性任务无需 LLM

- 格式验证、数据清洗、模式提取等任务，正则比 LLM 更快、更便宜、结果可复现
- 原则：能用规则解决的不用模型，节省成本并提高稳定性

课堂练习 3

场景：你有 10 万条用户评论，需要分析哪些评论涉及产品质量问题并提取关键信息。

```
comments = [
    "ORD-20240315: 等了三周才到，包装破损，差评!!",
    "ORD-20240401: 颜色和图片一样，非常满意",
    "ORD-20240412: 发货快，但尺码偏小，建议买大一号"，# ... 10万条
]
```

任务：请设计一个“正则 + LLM”的处理流程，完成以下目标：

- **步骤 1（正则）：**从每条评论中提取订单号，并筛选出含负面关键词（破损/差评/退货等）的评论
- **步骤 2（LLM）：**将筛选出的评论发给 LLM，让它分类问题类型（物流/质量/尺码）并给出处理建议

练习 3 参考答案

步骤 1（正则）：提取订单号 + 筛选负面评论

```
import re

# 提取订单号：ORD-开头 + 数字
orders = {re.search(r"ORD-\d+", c).group(): c for c in comments}

# 筛选含负面关键词的评论（大幅减少发给 LLM 的数据量）
neg = [c for c in comments if re.search(r"破损|差评|退货|投诉|发霍|偏小|偏大", c)]
```

步骤 2（LLM）：只处理筛选后的少量评论

```
prompt = f"""对以下用户评论进行分类：
{neg}  # 只发送筛选后的评论，而非全部 10 万条
请按问题类型分类（物流/质量/尺码），并给出处理建议。"""

result = call_llm(prompt)  # 只处理几百条，而非 10 万条
```

总结

1. **为什么学正则**
   文本数据无处不在，正则是描述文本模式的标准工具，比字符串方法更简洁、更通用

2. **正则语法四类构件**
   字符匹配， 数量控制， 位置锚定， 分组与逻辑

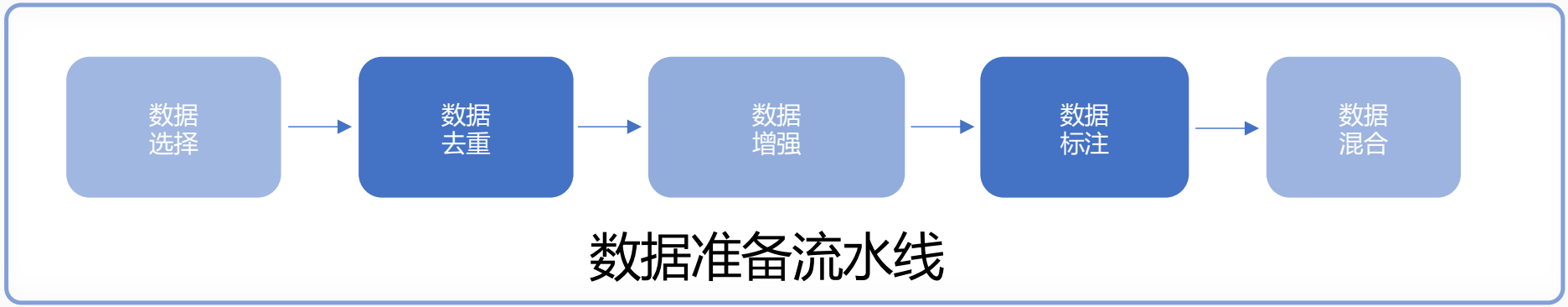
3. **Python 实战**
   re 模块：findall / search / sub / split；Pandas .str 向量化操作处理大规模数据

4. **正则引擎与性能**
   正则 → 有限自动机（NFA/DFA）；避免嵌套量词引发的灾难性回溯

5. **正则 × 大模型**
   LLM 辅助写正则（生成、解释、调试）；正则辅助 LLM（预处理筛选、后处理提取、替代确定性任务以节省成本）

目录

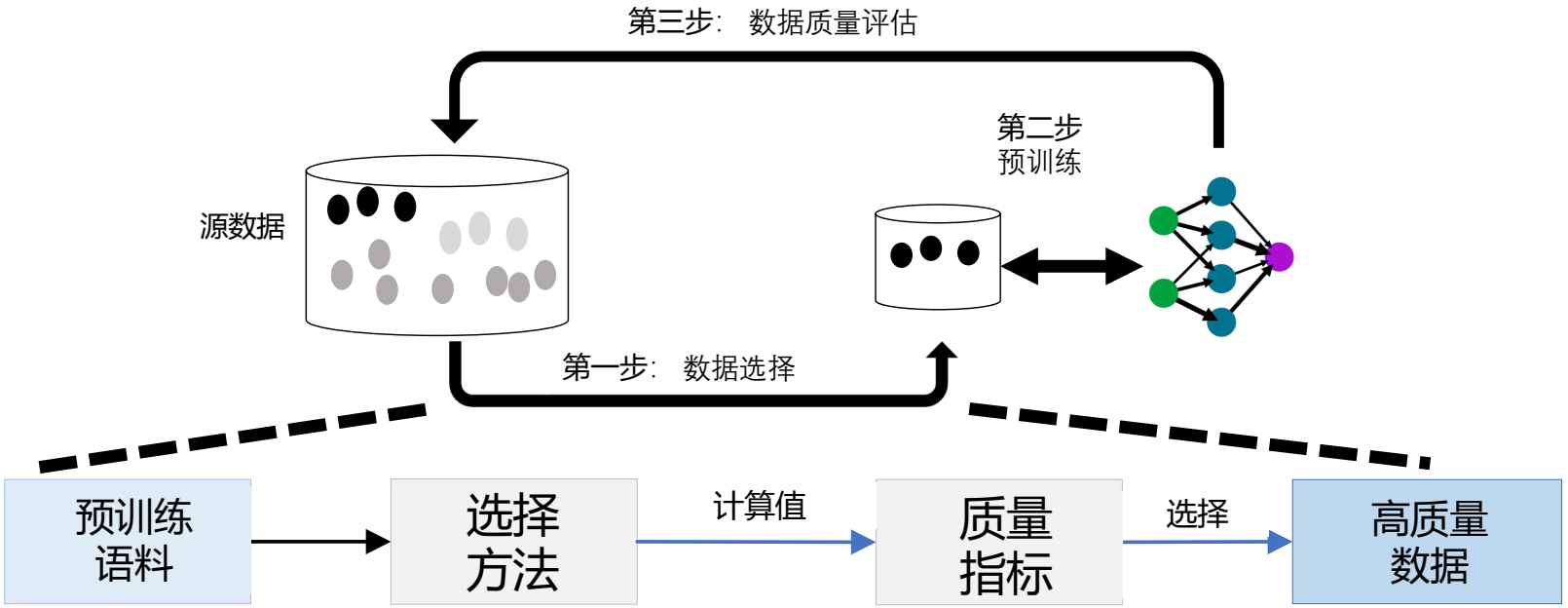
大模型数据准备流水线



- 挑战
  - 依赖专家
  - 耗时
  - 难以找到最优解
    - 例如： 大量候选流水线

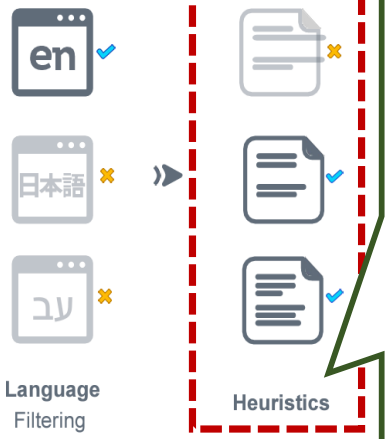
数据选择

- **数据选择：** 在减少数据量的同时，获得相似甚至更好的训练效果



数据选择：基于规则的选择

基于规则的选择：使用启发式规则选择理想数据启发式规则



确保数据连贯、内容丰富、无偏见

目标	启发式规则
确保文本质量	**词数：** 50 - 100,000 词
合适的词长	**平均长度：** 3 - 10 个字符
管理符号使用	**符号比例：** # 和 ... 的比例 <0.1
限制列表格式	**列表控制：** 以项目符号开头 <90%，以省略号结尾 <30%
要求字母词	**字母存在率：** 80% 的词
过滤不连贯的英语	**停用词：** 必须包含至少两个常用词

Rae J W, Borgeaud S, Cai T, et al. Scaling language models: Methods, analysis & insights from training gopher[J]. arXiv preprint arXiv:2112.11446, 2021.

数据选择：基于内容的选择

基于内容的选择： 选择高质量数据（例如：人工编辑的数据；来自可信来源如同行评审论文的数据）

- **基于分类**：识别可能与已知"高质量"数据语料库具有相同（或相似）分布的数据点
- **基于困惑度**：训练大语言模型并在数据上评估以实现更高的选择性能
- **基于标准**：使用模型针对感知质量的多个维度对文档进行评分 → 捕捉人类对数据质量的直觉

数据选择：基于内容的选择

基于分类：识别可能与已知"高质量"数据语料库具有相同（或相似）分布的数据点

第一步：特征哈希

- 考虑文本词语"the"、"quick"、"brown"、"fox"。使用哈希函数，它们可能被映射到大小为20的特征向量的索引 [5,17,3,12]。

第二步：使用精选/其他页面训练分类器

- 类别 1（精选内容）：高质量来源，如维基百科、书籍和精选网站。
- 类别 2（其他网页）：互联网上的典型网页。

第三步：使用训练好的分类器评分

- 根据网页内容与精选类别的相似程度，为网页分配质量评分。

第四步：使用帕累托分布采样

- 平衡低质量页面的纳入以防止偏差：

数据选择：基于内容的选择

基于困惑度：训练大语言模型并在数据上评估以实现更高的选择性能

- 句子示例：
  - "*I love machine learning*"
- 计算条件概率
  - P(i)=0.2
  - P(love | i)=0.1
  - P(machine | i,love)=0.05
  - P(learning | i,love,machine)=0.01
  - N=4

概率分布为 ***P*** 的模型预测 N 个词的序列 $w_1, w_2, \ldots, w_N$

$$PP(W) = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_1,\ldots,w_{i-1})}$$

$$\frac{1}{4}(log0.2 + log0.1 + log0.05 + log0.01) \approx -2.8782$$

$$Perplexity(P) = \exp(-(-2.8782)) \approx 17.77$$

较低的困惑度 意味着模型的概率分布更接近真实数据分布

Brown T, et al. Language models are few-shot learners[J]. NeurIPS, 2020, 33: 1877-1901.

数据选择：基于内容的选择

基于标准：使用模型沿感知质量的多个维度对文档进行评分 → *捕捉人类对数据质量的直觉*

质量标准：

C1. 写作风格：*用词优美或精炼*

C2. 专业性：*语料库的难度级别*

C3. 事实与知识：*具有高密度的长尾事实知识*

C4. 教育价值：*包含清晰的解释、逐步推理或问答内容*

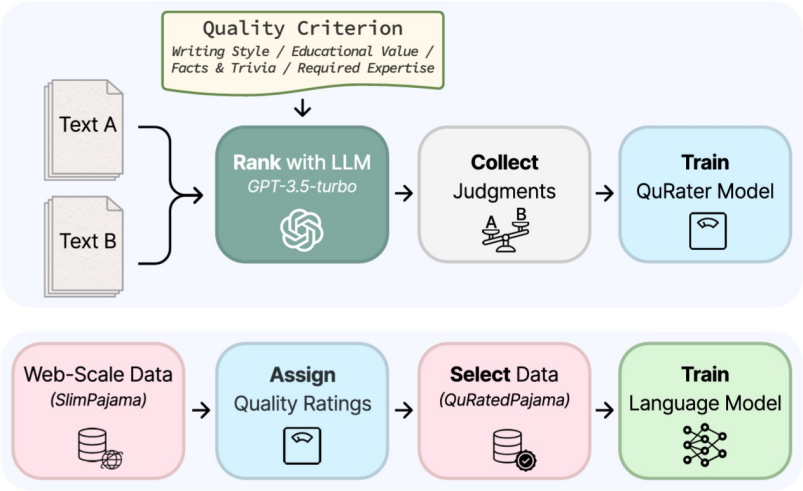
数据选择：基于内容的选择

基于标准：使用模型沿感知质量的多个维度对文档进行评分

- **1.** 从大量文档集合中采样文本对 (A, B)
- **2.** 根据标准和文本对 (A, B)，大语言模型（如 GPT3.5）给出*B 优于 A 的置信度，即* $p_{B \succ A} \in [0, 1]$
- **3.** 生成判断数据集

$$\mathcal{J} = \{(t_i, t_j, p_{i \succ j})\}$$

- **4.** 微调 1.3B Sheared-Llama 模型
  - 在四个标准下预测质量评分



数据去重

数据去重： 在相同文档上训练会减慢训练速度并可能损害性能 → 识别相同/相似文档并保留一份

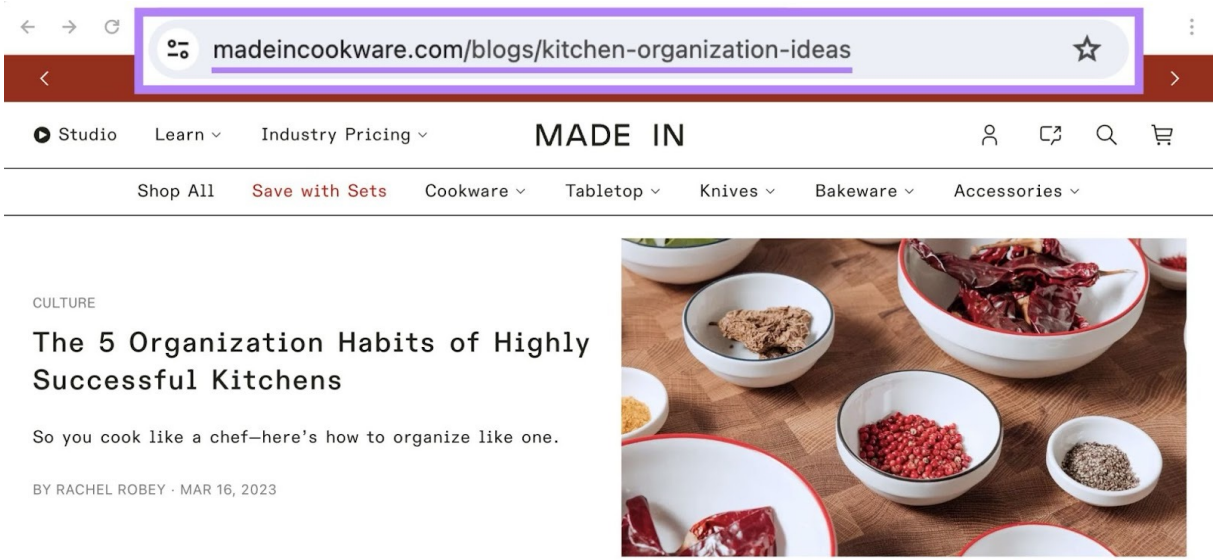
- **精确匹配**：利用MD5哈希确保文档完全相同。
- **近似匹配**：使用min-hash/sim-hash定位重叠文本，以Jaccard 相似度衡量
- **语义匹配**：使用预训练嵌入对文档进行聚类

数据去重：精确匹配技术

精确匹配技术：

1. URL去重：移除共享相同URL的数据

- 单个网页可能出现多次



数据去重：精确匹配技术

精确匹配技术：

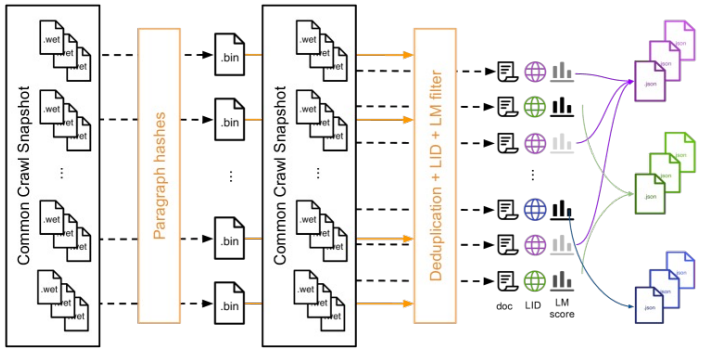
2. 哈希函数：保证找到所有精确匹配

(1) 初始化哈希集合

一个集合——已遇到的文本条目的哈希值。

(2) 对每个文本条目进行哈希

对每个文本条目计算简单的哈希值（例如，其字符的 ASCII 值之和）。



(3) 检查重复

如果当前条目的哈希值已在集合中，则为重复项，将被忽略。

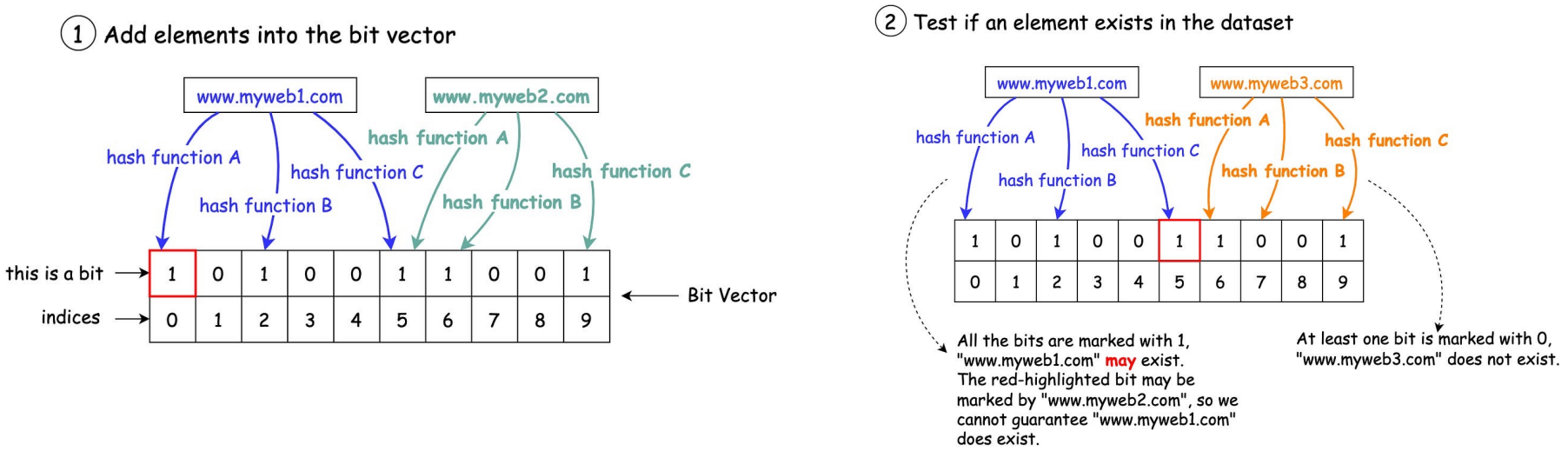
如果哈希值不在集合中，则将其添加到集合中并保留该条目。

高效快速，但可能因哈希冲突产生误报并移除不匹配的文档

数据去重：精确匹配技术

精确匹配技术：

3. 布隆过滤器：使用位数组进行文档比较的空间高效方法。



高度空间高效

但可能将非重复文档错误识别为重复文档

数据去重：近似匹配技术

① 问题：给定 N 条记录，如何快速找到内容相似的记录对?

- 两两比较 $O(N^2)$，当 N 很大时无法承受。需要一种方法快速筛选候选相似对。

② 相似度度量：Jaccard 指数

将每条记录表示为 token 集合（单词集合），计算重叠程度：

$$J(A, B) = |A \cap B| / |A \cup B|$$

示例（餐馆数据）：

```
A = {"szechwan", "garden", "chinese"}
B = {"szechwan", "house", "chinese"}

A ∩ B = {"szechwan", "chinese"}  → 2
A ∪ B = {"szechwan","garden","chinese","house"} → 4
J(A,B) = 2/4 = 0.5
```

③ MinHash 的思路：用签名近似 Jaccard

不直接比较集合，而是：

① 将每个 token 映射为整数 ID（**token_to_id**）

② 对每条记录的 token-id 集合应用多个哈希函数

③ 每个哈希函数取最小值，得到“签名”（**signature**）

④ 签名重合率 ≈ Jaccard 相似度

效果：将 O(|A|+|B|) 的集合比较压缩为 O(k) 的签名比较

数学保证：对于任意哈希函数 h，**P(min(h(A)) = min(h(B))) = J(A, B)**

使用 k 个独立哈希函数，签名重合率可无偏估计 Jaccard 相似度。k 越大，估计越精确。

数据去重：近似匹配技术（MinHash 签名构建）

步骤 1：建立 token 编码表 token_to_id

```
# 所有唯一 token 按字典序排序，从 1 开始编号
{"chinese": 1, "garden": 2, "house": 3,
 "szechwan": 4, ...}
```

步骤 2：构造 token-id 集合 token_id_set

```
A = {1, 2, 4}   # chinese, garden, szechwan
B = {1, 3, 4}   # chinese, house, szechwan
```

示例：使用 3 个哈希函数，prime = 7

	$h_1=(2x+1)\%7$	$h_2=(3x+2)\%7$	$h_3=(5x+3)\%7$
① x=1	3	5	1
② x=2	5	1	6
③ x=3	0	4	4
④ x=4	2	0	2

步骤 3：对每个哈希函数，计算 min((a*x + b) % p for x in set)

A = {1, 2, 4}：取行 ①②④ 的每列最小值

h_1: min(3,5,2) = **2**　h_2: min(5,1,0) = **0**　h_3: min(1,6,2) = **1**

Sig(A) = [2, 0, 1]

B = {1, 3, 4}：取行 ①③④ 的每列最小值

h_1: min(3,0,2) = **0**　h_2: min(5,4,0) = **0**　h_3: min(1,4,2) = **1**

Sig(B) = [0, 0, 1]

步骤 4：比较签名

[2,0,1] vs [0,0,1]

h_1: 2≠0 ✗　h_2: 0=0 ✓　h_3: 1=1 ✓

重合率 = 2/3 ≈ 0.67（真实 Jaccard = 0.5）

k 越大，估计越接近真实值

数据去重：近似匹配技术（LSH分段）

① **问题**：即使有了签名，N 条记录仍需 $O(N^2)$ 次签名比较。如何只比较“可能相似”的对?

② **解法：将签名分段（Banding），同一 band 内哈希值完全相同的记录对才成为候选对**

③ **接续上一页的例子**：将哈希函数扩展到 6 个，并增加第三条记录 C

A: "szechwan garden chinese" → {1,2,4}　**B:** "szechwan house chinese" → {1,3,4}　**C:** "house szechwan" → {3,4}

签名分为 3 个 band（每个 band 2 行）

		A	B	C
band 0	h_1	2	0	0
	h_2	0	0	0
band 1	h_3	1	1	2
	h_4	1	0	0
band 2	h_5	0	0	0
	h_6	0	0	4

A: (2,0) (1,1) (0,0)
B: (0,0) (1,0) (0,0)
C: (0,0) (2,0) (0,4)

判断规则：任意一个 band 完全相同 → 候选对

A vs B:
band0: (2,0) vs (0,0) ✗
band1: (1,1) vs (1,0) ✗
band2: (0,0) vs (0,0) ✓
→ 候选对！（确实相似，J=0.5）

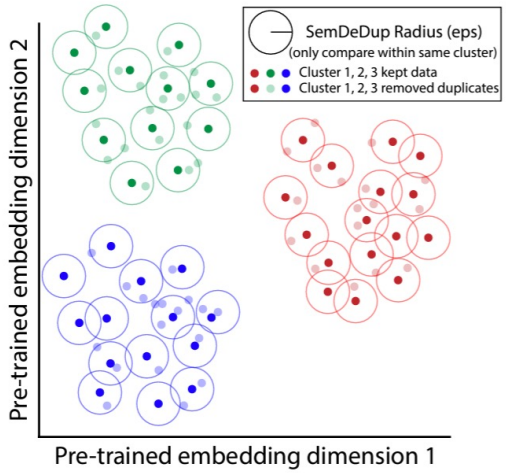
B vs C:
band0: (0,0) vs (0,0) ✓
band1: (1,0) vs (2,0) ✗
band2: (0,0) vs (0,4) ✗
→ 候选对！（共享 house, szechwan）

A vs C:
band0/1/2 均不同 → 非候选对

数据去重：近似匹配技术

2. 基于模型的方法：使用预训练模型进行语义去重

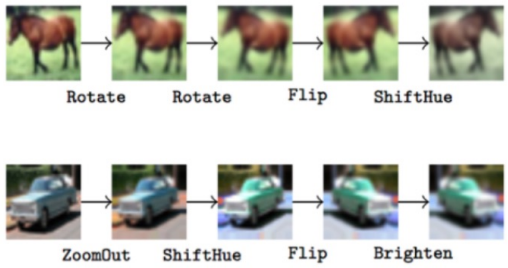
- **步骤 1：**利用预训练大语言模型创建的嵌入空间，提供语义上有意义的距离度量来识别重复项
- **步骤 2：**使用大语言模型对每个数据点进行嵌入
- **步骤 3：**使用 k-means 对嵌入的数据点进行聚类
- **步骤 4：**在每个聚类内，计算数据点之间的成对余弦相似度。
- **步骤 5：**对于聚类内识别出的重复项，仅保留与聚类中心余弦相似度最低的数据点，移除其他重复项。



数据增强

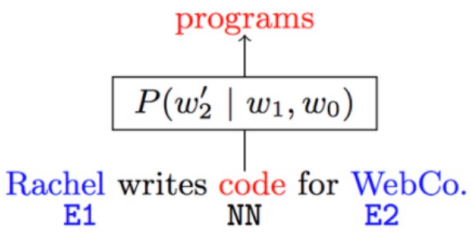
数据增强： 寻找最接近目标数据分布的辅助数据（例如，医学或法律领域）。

Images



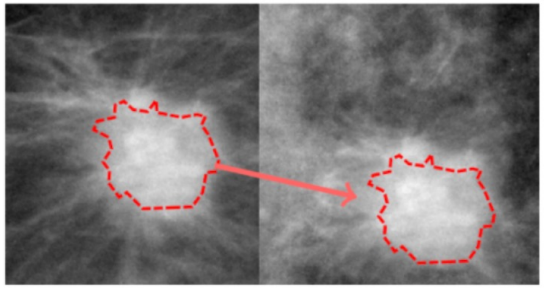
- Rotations
- Scaling / Zooms
- Brightness
- Color Shifts
- Etc…

Text



- Synonymy
- Positional Swaps
- Etc…

Medical



Domain-specific transformations.
Ex:

1. *Segment tumor mass*
2. *Move*
3. *Resample background tissue*
4. *Blend*

数据增强

挑战：如何选择高质量的预训练数据集？

- 数据增强： 目标是找到最接近**辅助数据**最接近领域内数据分布的
- 领域特定选择：设 ***I*** 为领域内数据集，***N*** 为通用数据集，N_I 为 *N*中我们希望发现的领域内子集。"*x(i)* 从 *N* 中随机抽取的属于 N_I"的概率为：

Moore-Lewis 选择方法

$$P(N_I|x^{(i)},N)=\frac{P(x^{(i)}|I)P(N_I|N)}{P(x^{(i)}|N)},\qquad \frac{P(x^{(i)}|I)}{P(x^{(i)}|N)}\propto\frac{P(x^{(i)}|I)P(N_I|N)}{P(x^{(i)}|N)}$$

- 训练模型来估计 $P(x^{(i)}|I)$ and $P(x^{(i)}|N)$，在 *I* 和 *N* 的采样
- $\frac{P(x^{(i)}|I)}{P(x^{(i)}|N)}$ 近似为 $\log(P(x^{(i)}|I))-\log(P(x^{(i)}|N))$，即在 *I* 和 *N训练的模型的cross-entropy loss。*

数据混合

数据准备：将大量脏数据转化为优质数据子集

- 数据混合：数据混合优化训练语料库中不同数据域的**权重分配**，以提升模型训练效率和性能。

Component	Raw Size	Weight	Epochs	Effective Size	Mean Document Size
Pile-CC	227.12 GiB	18.11%	1.0	227.12 GiB	4.33 KiB
PubMed Central	90.27 GiB	14.40%	2.0	180.55 GiB	30.55 KiB
Books3†	100.96 GiB	12.07%	1.5	151.44 GiB	538.36 KiB
OpenWebText2	62.77 GiB	10.01%	2.0	125.54 GiB	3.85 KiB
ArXiv	56.21 GiB	8.96%	2.0	112.42 GiB	46.61 KiB
Github	95.16 GiB	7.59%	1.0	95.16 GiB	5.25 KiB
YoutubeSubtitles	3.73 GiB	0.60%	2.0	7.47 GiB	22.55 KiB
PhilPapers	2.38 GiB	0.38%	2.0	4.76 GiB	73.37 KiB
NIH ExPorter	1.89 GiB	0.30%	2.0	3.79 GiB	2.11 KiB
Enron Emails†	0.88 GiB	0.14%	2.0	1.76 GiB	1.78 KiB
The Pile	**825.18 GiB**			**1254.20 GiB**	**5.91 KiB**

Table 1: Overview of datasets in the Pile before creating the held out sets. Raw Size is the size before any up- or down-sampling. Weight is the percentage of bytes in the final dataset occupied by each dataset. Epochs is the number of passes over each constituent dataset during a full epoch over the Pile. Effective Size is the approximate number of bytes in the Pile occupied by each dataset. Datasets marked with a † are used with minimal preprocessing from prior work.

数据混合：经验确定法

挑战：如何选择高质量的预训练数据集?

- 数据混合： 确定最优的领域比例以提高训练效率和模型性能
- 经验确定方法
  - **规则 1**：防止小规模数据源（如 MultiUN）被过度采样；
  - **规则 2**：大比例的代码数据（如 50%）不会损害自然语言性能，且有利于推理任务；
  - **规则 3**：在小规模大语言模型（如 1B 参数）上测试不同组合。

Github	Microsoft	2008-4	-	All
mC4	Google Research	2021-6	251 GB	All
MNBVC	Liwu Community	2023-1	20811 GB	All
MTP	BAAI	2023-9	1.3 TB	All
MultiUN	German Research Center for Artificial Intelligence (DFKI) GmbH	2010-5	4353 MB	All
News-crawl	UKRI et al.	2019-1	110 GB	All

Nan Du, et al. GLaM: Efficient scaling of language models with mixture-of-experts . ICML, 2022.

数据混合：模型确定法

挑战：如何选择高质量的预训练数据集？

- 数据混合： 确定最优的领域比例以提高训练效率和模型性能
- 模型确定方法：在不依赖下游任务的情况下，优化训练模型时分配给不同领域的比例
  - 使用小型代理模型优化领域比例

$$\min_{\theta} \max_{g \in \mathcal{G}} \mathbb{E}_{(x,y)\sim\mathcal{D}_g}[\ell(f_\theta(x), y)]$$

最小化所有领域的最大损失

- θ: Model parameters
- g: Group/domain
- $\mathcal{D}_g$: Data distribution for group g
- ℓ: Loss function

  - 使用优化后的领域比例训练更大的模型

Sang Michael Xie, Hieu Pham , et al. Doremi: Optimizing data mixtures speeds up language model pretraining . NeurIPS, 2023.

总结

1. **数据选择**
   基于规则（启发式过滤）与基于内容（分类、困惑度、多维度质量评分）两类方法，在减少数据量的同时获得相似或更优的训练效果

2. **数据去重**
   精确匹配（URL / 哈希 / 布隆过滤器）、近似匹配（MinHash）与语义匹配（嵌入聚类）三层去重策略

3. **数据增强**
   寻找最接近目标领域分布的辅助数据，通过 Moore-Lewis 方法评估领域内与通用数据的分布差异

4. **数据混合**
   优化训练语料中不同数据域的比例分配：经验规则确定与模型优化确定（如 DoReMi）