一、正则表达式

1. 应用场景

实际项目中，可能会需要对文本内容中的某些模式进行处理的情况：

1. 查找并处理文本中特定格式的日期、邮箱地址或手机号码等。
2. 从混乱的文本中提取具有特定格式的关键信息。
3. …

上述任务可以考虑使用正则表达式来完成。
正则表达式是一种用于匹配和处理文本模式的工具。它是由**一系列字符和控制符组成的字符串**，用于描述、查找和操作符合特定模式的文本，当需要处理、搜索或匹配符合特定模式的文本时，正则表达式是一个非常有用的工具。

2. 正则表达式基本语法

简单来说，撰写正则表达式就是根据需要来设计字符和控制符的过程。

I. 元字符

语法	说明
.	匹配除换行符外的任意字符
\w	匹配字母、数字、下划线、汉字
\s	匹配任意空白符
\d	匹配数字
^	从行开头开始匹配
$	从行结尾开始匹配

In [86]:
```python
import re

# 给定文本
text = "请问如何写一个普通的正则表达式？请问怎么使用python3实现正则表达式？"

# 提前编译正则表达式

# 匹配数字
# regex = '\d'

# 匹配英文、数字、下划线、汉字
# regex = '\w'

# 匹配任意字
# regex = '请问.'

# 从字符串的最开始进行匹配
# 脱字符
regex = '^请问.'

regex = re.compile(regex)
result = re.findall(regex, text)
result
```

Out[86]:
```
['请问如']
```

II. 重复限定符

语法	说明
*	重复零次或以上
+	重复一次或以上
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次
{n,m}	重复n次到m次

In [89]:
```
import re

# 给定文本
text = "如有疑问请于12月20日前致电13312345612咨询。"

# 提前编译正则表达式

# 匹配数字组合
# regex = '\d+'


# 匹配1开头的11位数字-手机号
regex = '1\d{10}'


regex = re.compile(regex)
result = re.findall(regex, text)
result
```

Out[89]:
```
['13312345612']
```

III. 操作符

语法	说明	举例
()	分组，将括号括起来的内容视作1个字符单位	(ab)：将a和b视为一个字符
\	转义，将各种正则字符转换为普通字符，避免曲解	*：将星号视作普通字符而非重复匹配字符
|	分支条件、或，满足分支条件之一即可算作匹配	(a|b)：匹配a或b
[]	区间	[A-Z]：匹配从A-Z的26个字母

In [90]:
```
import re

# 给定文本
text = "我的邮箱账号是myemail123@outlook.com，劳烦Richard把相关材料发送至我的邮箱。"

# 提前编译正则表达式

# 匹配邮箱
regex = '[A-Za-z0-9]+@[A-Za-z0-9]+\.com'
```

```
regex = re.compile(regex)
result = re.findall(regex, text)
result
```

Out[90]:
```
['myemail123@outlook.com']
```

IV. 其他

掌握基本的正则表达式语法已足够应对大部分场景，复杂场景大部分情况下可以通过基本语法的组合来解决。

几个学习正则表达式的网站：

- 正则表达式测试网站
  https://regex101.com/
- 常用正则表达式大全
  https://www.cnblogs.com/zxin/archive/2013/01/26/2877765.html

3. 正则表达式代码Demo

In [91]:
```
# 论文的评审内容有时会存在大量列举reference（参考文献）的情况
# 这些reference会占据大量的评审内容篇幅
# 且多数情况下仅为参考文献的标题、年份等无意义信息

text = 'REVIEW title:\nVery interesting work and the proposed approach is well explained. The experimental section could be improve
print(text)
```

REVIEW title:
Very interesting work and the proposed approach is well explained. The experimental section could be improved.
REVIEW rating:
8: Top 50% of accepted papers, clear accept
REVIEW review:
Summary:
The manuscript extends the Neural Expectation Maximization framework by integrating an interaction function that allows asymmetric pairwise effects between objects. The network is demonstrated to learn compositional object representations which group together pixels, optimizing a predictive coding objective. The effectiveness of the approach is demonstrated on bouncing balls sequences and gameplay videos from Space Invaders. The proposed R-NEM model generalizes

Review:
Very interesting work and the proposed approach is well explained. The experimental section could be improved.
I have a few questions/comments:
1) Some limitations could have been discussed, e.g. how would the model perform on sequences involving more complicated deformations of objects than in the Space Invaders experiment? As you always take the first frame of the 4-frame stacks in the data set, do the objects deform at all?
2) It would have been interesting to vary K, e.g. study the behaviour for K in {1,5,10,25,50}. In Space Invaders the model would probably really group together separate objects. What happens if you train with K=8 on sequences of 4 balls and then run on 8-ball sequences instead of providing (approximately) the right number of components both at training and test time (in the extrapolation experiment).
3) One work that should be mentioned in the related work section is Michalski et al. (2014), which also uses noise and predictive coding to model sequences of bouncing balls and NORBvideos. Their model uses a factorization that also discovers relations between components of the frames, but in contrast to R-NEM the components overlap.
4) A quantitative evaluation of the bouncing balls with curtain and Space Invaders experiments would be useful for comparison.
5) I think the hyperparameters of the RNN and LSTM are missing from the manuscript. Did you perform any hyperparameter optimization on these models?
6) Stronger baselines would improve the experimental section, maybe Seo et al (2016). Alternatively, you could train the model on Moving MNIST (Srivastava et al., 2015) and compare with other published results.

I would consider increasing the score, if at least some of the above points are sufficiently addressed.

References:
Michalski, Vincent, Roland Memisevic, and Kishore Konda. "Modeling deep temporal dependencies with recurrent grammar cells"." In Advances in neural information processing systems, pp. 1925-1933. 2014.
Seo, Youngjoo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. "Structured sequence modeling with graph convolutional recurrent networks." arXiv preprint arXiv:1612.07659 (2016).
Srivastava, Nitish, Elman Mansimov, and Ruslan Salakhudinov. "Unsupervised learning of video representations using lstms." In International Conference on Machine Learning, pp. 843-852. 2015.
REVIEW confidence:
5: The reviewer is absolutely certain that the evaluation is correct and very familiar with the relevant literature

In [92]:
```
# 以\nreference作为开头，以19XX.或20XX.作为结尾，这是列举一大段参考文献的范式，
# flags定义2种匹配模式，(re.S|re.I)分别指支持“匹配换行符（即匹配所有行，无需换行符隔开）”和“忽略大小写”
```

```
regex = re.compile("\n(reference)(.*)(19|20\d+).*?\.", flags=(re.S|re.I))

# re.sub()将会根据正则表达式从给定文本中匹配出相应的内容，并替换为特定的字符串
text_processed = re.sub(regex, "", text)
print(text_processed)
```

```
REVIEW title:
Very interesting work and the proposed approach is well explained. The experimental section could be improved.
REVIEW rating:
8: Top 50% of accepted papers, clear accept
REVIEW review:
Summary:
The manuscript extends the Neural Expectation Maximization framework by integrating an interaction function that allows asymmetric pairwise effects between objects. The network is demonstrated to learn compositional object representations which group together pixels, optimizing a predictive coding objective. The effectiveness of the approach is demonstrated on bouncing balls sequences and gameplay videos from Space Invaders. The proposed R-NEM model generalizes

Review:
Very interesting work and the proposed approach is well explained. The experimental section could be improved.
I have a few questions/comments:
1) Some limitations could have been discussed, e.g. how would the model perform on sequences involving more complicated deformations of objects than in the Space Invaders experiment? As you always take the first frame of the 4-frame stacks in the data set, do the objects deform at all?
2) It would have been interesting to vary K, e.g. study the behaviour for K in {1,5,10,25,50}. In Space Invaders the model would probably really group together separate objects. What happens if you train with K=8 on sequences of 4 balls and then run on 8-ball sequences instead of providing (approximately) the right number of components both at training and test time (in the extrapolation experiment).
3) One work that should be mentioned in the related work section is Michalski et al. (2014), which also uses noise and predictive coding to model sequences of bouncing balls and NORBvideos. Their model uses a factorization that also discovers relations between components of the frames, but in contrast to R-NEM the components overlap.
4) A quantitative evaluation of the bouncing balls with curtain and Space Invaders experiments would be useful for comparison.
5) I think the hyperparameters of the RNN and LSTM are missing from the manuscript. Did you perform any hyperparameter optimization on these models?
6) Stronger baselines would improve the experimental section, maybe Seo et al (2016). Alternatively, you could train the model on Moving MNIST (Srivastava et al., 2015) and compare with other published results.

I would consider increasing the score, if at least some of the above points are sufficiently addressed.

REVIEW confidence:
5: The reviewer is absolutely certain that the evaluation is correct and very familiar with the relevant literature
```

二、AC自动机词库匹配

1. 应用场景

实际项目中可能会遇到一些需要进行高效模式匹配的场景：

1. 从大量文本中匹配出预设词库中的敏感词。
2. 从大量文本中判断是否存在预设词库中的某些词。
3. …

2. “模式”和“高效”的定义

I. 模式与模式匹配

在这里提及的“模式”这个概念可能会让人感觉有些混淆，实际上“一个模式”在我们的词匹配场景中可以大体理解为“一个字符串”，那么这样一来，“模式匹配”指的其实就是“字符串匹配”、“多模式匹配”指的就是“多个字符串匹配”。AC（Aho-Corasick）自动机就是多模式匹配算法的一种。

模式匹配任务的抽象定义：有m个模式构成的模式集合$\{p_1, p_2, \cdots, p_m\}$，给定连续文本$t$，找到连续文本$t$内出现过的模式。

模式匹配任务的简单例子：有模式集合$\{bd, ac, ab, abc\}$，给定连续文本$oabcabda$，其中出现过bd, ab, abc共3个模式。

II. 高效

在谈论高效之前，我们先谈论什么是“不高效”，因为它是相对于不高效来定义的。
还是上述例子的场景：有模式集合$\{bd, ac, ab, abc\}$，给定连续文本$oabcabda$，考虑使用python中最容易想到的方法来实现词库匹配（模式匹配）：将模式集合转换成模式集合（列表）`keywords=["bd", "ac", "ab", "abc"]`，定义连续文本（字符串变量）`string="oabcabda"`，只要逐一遍历词库列表中的元素，并判断其是否存在字符串`string`中即可。

```
keywords=["bd", "ac", "ab", "abc"]
string="oabcabda"
for keyword in keywords:
    if keyword in string:
        print(keyword)
```

这是相当简单的匹配实现，但实际上随着取4次列表元素，字符串也被遍历检索了4次，这其实是十分低效的，尤其是在词库量（模式集合）极大以及文本量、文本长度都较大的情况下，基于简单实现的匹配会十分缓慢，因此有必要采用AC自动机进行匹配，在AC自动机的有关实现中，“字符串”自始至终都将只会被遍历检索1次，这将大大缩短匹配时间，因此说AC自动机在模式匹配中是高效的。
在后续讨论的有关内容中，将进一步阐释AC自动机是如何实现这点的。

3. AC自动机匹配过程

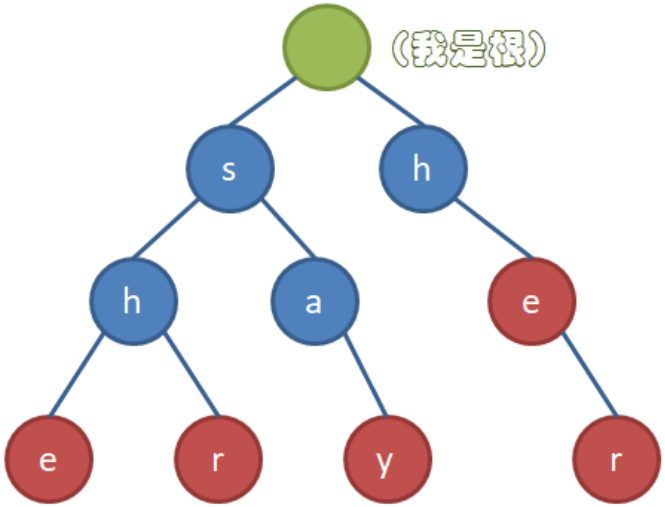
给定词库（模式集合）{” she ”,” shr ”,” say ”,” he ”,” her ”}，以及一段连续文本” $aasherhsy$ ”，需要从连续文本中找到出现过的词库中的词。

> 部分案例及图来自：https://www.cnblogs.com/sclbgw7/p/9260756.html

通过AC自动机的方式完成上述任务，以此了解AC自动机的过程。

1. 建立存储有词库字符串的Trie树。

> Trie树：字典树、单词查找树，一般用作字符串查找，借助字符串的公共前缀来表示字符串集合。具体可参照下图。

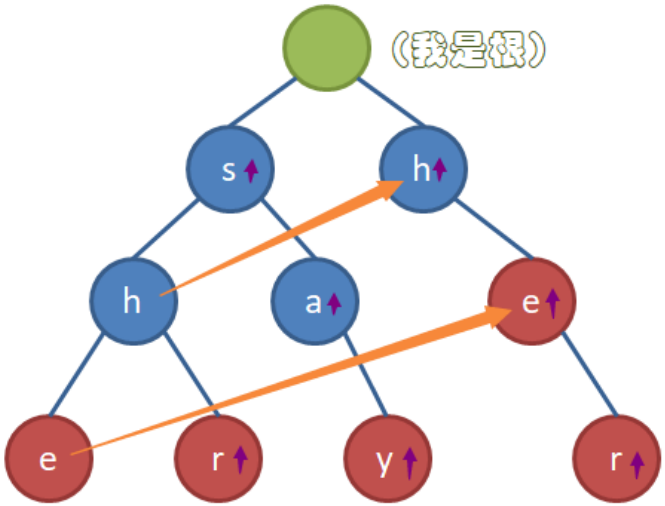


2. 在相应节点定义接收态。

> 接收态：返回匹配到的字符串的节点，即根节点到一个接收态的完整路径，对应一个词（模式）。如上图的红色节点即为接收态。

3. 为Trie树的每个节点定义失败路径机制。

> 失败路径机制：位于当前节点无法再进一步往下匹配时，将指向其“节点路径的最大后缀同值节点”（如下图的橙色箭头），如果不存在“节点路径的最大后缀同值节点”，则指向根节点（如下图的紫色箭头代表指向根节点）。
>
> 节点路径的最大后缀同值节点：参照下图，例如左侧的节点“e”位于路径“she”上，路径“she”的最大后缀同值指的就是“he”，则最大后缀同值节点指的就是右侧的“e”。如果没有节点路径的最大后缀同值节点，那么就指向根节点。



4. 迭代输入的连续文本字符，沿着带有接收态、失败路径机制的Trie树进行匹配。走到接收态的时候，就返回接收态所在的路径，也即匹配到的词（模式）。当无法往下匹配的时候，就根据节点的失败路径指向来跳转走向。因此连续文本”$aasherhsy$”可以匹配出”she”,”he”,”her”，实际只需要遍历1次连续文本即可得到匹配结果，效率较高。

AC自动机代码Demo

In [93]:
```
# !python -m pip install esmre
# 如果提示编译失败，则
```

```
# !git clone https://github.com/wharris/esmre
# !cd esmre
# !vim pyproject.toml
# 将其中的requires = ["setuptools", "wheel", "Cython"]改为requires = ["setuptools", "wheel", "Cython<3"], 并保存
# !python -m pip install -e .
```

In [94]:
```
import esm

# 词库
keywords = ["保罗", "小卡", "贝弗利"]

# 连续文本
text = """NBA季后赛西部决赛，快船与太阳移师洛杉矶展开了他们系列赛第三场较量，上一场太阳凭借艾顿的空接绝杀惊险胜出，此役保罗火线复出
```

In [95]:
```
# 实例化AC自动机
index = esm.Index()

# 将词库中的词送入index实例
for keyword in keywords:
    index.enter(keyword)
index.fix()

# 针对连续文本进行匹配
result = index.query(text)
print(result)
```

```
[((162, 168), '保罗'), ((186, 192), '小卡'), ((246, 252), '保罗'), ((478, 484), '保罗'), ((846, 855), '贝弗利')]
```

三、困惑度过滤低质文本

1. 应用场景

实际项目中可能存在文本内容杂乱无序、语义不连贯的情况：

1. 爬取自Web的文本数据，由于格式适应问题无法获取到连贯的文本内容。
2. 论坛文本中胡乱生成的刷屏低质内容。
3. …

为识别出上述低质文本，最基本的想法是对文本的流畅程度、通顺程度进行度量。
为此可引入语言模型、尤其是预训练语言模型来进行相关度量。

2. 困惑度（perplexity）

一般情况下，语言模型即是评估文本序列符合人类语言使用习惯的模型，因为语言模型是学习自自然语言语料、且以最大化"语料现象"为目标进行训练的模型。以最经典的预训练模型GPT系列为例，其训练目标是"最小化交叉熵损失"（如下式），即追求预测分布与实际分布差距最小，也就是说在给定当前上下文的情况下，预测出的概率分布应尽可能接近实际下一个词的概率分布——这其实就是在学习自然语料的"习惯"，所以训练好的语言模型推理出的结果是具有一定的"通顺度"参考意义的。
对于一个长度为T的文本序列来说，参数为θ的GPT训练时的损失函数为：

$$loss = -\frac{1}{T}\sum_{t=1}^{T} logP(w_t|w_{<t};\theta) = -\frac{1}{T}[logP(w_1;\theta) + logP(w_2|w_1;\theta) + \cdots + logP(w_T|w_{T-1},\cdots,w_2,w_1;\theta)]$$

对于一个长度为T的文本序列$[w_1, w_2, \cdots, w_T]$来说，其由参数为θ的语言模型给出的困惑度的计算方式如下所述。

$$ppl = (\prod_{t=1}^{T} P(w_t|w_{<t};\theta))^{-\frac{1}{T}}$$

因此对于同个文本、同个语言模型来说，损失函数值与困惑度之间存在如下关系。这也侧面说明了困惑度确实可以度量文本的通顺程度，exp是单调递增函数：损失越小，模型的困惑度越小；损失越大，模型的困惑度越大。

$$ppl = exp(loss)$$

所以在实际的代码实现中，可以直接通过loss进一步计算得到困惑度。

3. 困惑度过滤代码Demo

In [96]:
```
import torch
from transformers import BertTokenizer, GPT2LMHeadModel

# model_dir = "../models/gpt2-chinese-cluecorpussmall"
```

```
model_path = "uer/gpt2-chinese-cluecorpussmall"
tokenizer = BertTokenizer.from_pretrained(model_path)
model = GPT2LMHeadModel.from_pretrained(model_path)
```

In [98]:
```
sentences = ["今天是个好日子。", "天今子日。个是好", "这个婴儿有900000克呢。", "我不会忘记和你一起奋斗的时光。",
             "我不会记忘和你一起奋斗的时光。", "会我记忘和你斗起一奋的时光。"]

inputs = tokenizer(sentences, padding='max_length', max_length=50, truncation=True, return_tensors="pt")
inputs.keys()
```

Out[98]:
```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
```

In [99]:
```
# batch_size, sequence_length
bs, sl = inputs['input_ids'].size()
print(bs, sl)
```

```
6 50
```

In [100…
```
outputs = model(**inputs, labels=inputs['input_ids'])
logits = outputs["logits"]
logits.shape
```

Out[100]:
```
torch.Size([6, 50, 21128])
```

In [101…
```
# 错位构造logits和label
# 后续可用于计算交叉熵损失
shift_logits = logits[:, :-1, :].contiguous()
shift_labels = inputs['input_ids'][:, 1:].contiguous()
shift_attentions = inputs['attention_mask'][:, 1:].contiguous()

print(logits.shape)
print(shift_logits.shape)
```

```
torch.Size([6, 50, 21128])
torch.Size([6, 49, 21128])
```

In [102…
```
# reshape成(bs*sl, vocab_size)
loss_fct = CrossEntropyLoss(ignore_index=0, reduction="none")
loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.view(-1)).detach().reshape(bs, -1)
loss.shape
```

Out[102]:
```
torch.Size([6, 49])
```

In [103…

```
# 计算平均损失，求平均时不计入padding
meanloss = loss.sum(1) / shift_attentions.sum(1)
meanloss.shape
```

Out[103]:

```
torch.Size([6])
```

In [104…

```
# 计算ppl
ppls = torch.exp(meanloss).numpy().tolist()
for sentence, ppl in zip(sentences, ppls):
    print('"{}"这句话的困惑度为: {}'.format(sentence, ppl))
```

```
"今天是个好日子。"这句话的困惑度为: 10.874716758728027
"天今子日。个是好"这句话的困惑度为: 905.6909790039062
"这个婴儿有900000克呢。"这句话的困惑度为: 156.19229125976562
"我不会忘记和你一起奋斗的时光。"这句话的困惑度为: 14.507042884826666
"我不会记忘和你一起奋斗的时光。"这句话的困惑度为: 38.39504623413086
"会我记忘和你斗起一奋的时光。"这句话的困惑度为: 519.60400390625
```

四、最小哈希算法实现文本去重

1. 应用场景

实际项目中可能会遇到一些存在重复文本数据的场景：

1. 重复存在的文档：同样的文档内容因为命名不同，被存放了好几份。
2. Web数据中的镜像网页：爬取的互联网数据中，存在同个网页架设在不同的服务器、不同的URL中，通过URL没法完全辨别两个网页是否相同，但实际上两个网页的内容又是相同的。
3. Web数据中被复用在多个网站的新闻稿：爬取的互联网数据中，一些不同站点的新闻稿很有可能是转载自某个同源平台的同篇新闻稿。
4. …

在进行处理的时候，对这些数据进行去重的最基本思路就是，度量各个文本集合中共同存在的元素，然后把共同出现元素数较多的文本视为相似，并将其一给剔除，即为完成去重。

2. Jaccard相似度

I. 基本定义

根据共同出现元素来度量两个文本是否相等，最常用的指标就是**Jaccard相似度。**

有两个文本S_1, S_2，其Jaccard相似度为

$$Jac(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

II. 简单举例

举个简单例子，有文本$S_1 = \{a, b, c\}$、文本$S_2 = \{b, c, d\}$、文本$S_3 = \{b, e\}$，a, b, c, d, e均为当前各个文本里出现过的、互不相同的词元（字或词），度量S_1和S_2的Jaccard相似度：

1. S_1和S_2的交集为$\{b, c\}$，共2个元素。
2. S_1和S_2的并集为$\{a, b, c, d\}$，共4个元素。
3. 则S_1和S_2的Jaccard相似度为$\text{Jac}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{2}{4} = 0.5$。

III. 基于原始语料计算Jaccard相似度的局限性

Jaccard相似度的计算思路并不复杂，但是对原始文本集合统计交并集的过程在计算机中的运行效率较低，**原始语料往往是海量存在的、具有大段文字的文本，即文本数量庞大（文本量大）、文本内元素比较多（文本长度长）的时候，计算速度会慢得夸张。**所以就需要引入**MinHash**来以快速的低维计算代替相对计算缓慢的jaccard相似度。

3. MinHash

I. MinHashing过程

先通过简单例子直观地、感性地理解MinHash降维的处理过程，然后再去理解为什么这样做能够近似等价对两个文本计算Jaccard相似度。

有文本$S_1=\{a,b,c,d\}$、文本$S_2=\{b,c,d\}$、文本$S_3=\{a,d\}$，其中a,b,c,d是4个互不相同的词，自然地，实际使用中需要先对文本进行分词处理。则这3个文本进行MinHash的过程为如下所示。

1.用矩阵表示文本情况，矩阵的行index为词，矩阵的列index为文本名称，对于每个文本（列）来说，如果该文本包含有相应的词（行），那么则在对应的元素位置赋1，否则赋0，可以理解为最简单的词袋模型，每个词即为文本的1个特征，则每个文本都由4维特征所表示，从而得到下表。

行号	-	S_1	S_2	S_3
0	**a**	1	0	1
1	**b**	1	1	0
2	**c**	1	1	0
3	**d**	1	1	1

2.行打乱：第1次将行**随机**打乱，该次打乱操作记录为h_1，得到下表。

> **需要注意的是，“行号”是为了便于理解而引入的抽象概念，并不实际存在于特征矩阵中，因此行打乱的操作不影响行号。**

行号	h_1	S_1	S_2	S_3
0	**b**	**1**	**1**	0
1	**c**	1	1	0
2	**a**	1	0	**1**
3	**d**	1	1	1

3.取MinHash特征：对于各个文本（列）来说，其自上而下第一个1所在行的行号（自0起算）即为该文本由h_1操作得到的MinHash特征值，如对于S_1来说，经过h_1打乱后，其自上而下第一个1所在行的行号为0，S_2也为0，S_3为2，得到的MinHash特征值如下表所示。

MinHash	S_1	S_2	S_3
h_1	0	0	2

4.行打乱：第2次将行**随机**打乱，该次打乱操作记录为h_2，得到下表。

MinHash	S_1	S_2	S_3

行号	h_2	S_1	S_2	S_3
0	**a**	**1**	0	**1**
1	**c**	1	**1**	0
2	**d**	1	1	1
3	**b**	1	1	0

5.取MinHash特征：经过h_2打乱后，记录各文本自上而下第一个1所在行号，结合h_1打乱的结果，得到的MinHash特征值如下表所示。

MinHash	S_1	S_2	S_3
h_1	0	0	2
h_2	0	1	0

6.行打乱：第3次将行**随机**打乱，该次打乱操作记录为h_3，得到下表。

行号	h_3	S_1	S_2	S_3
0	**c**	**1**	**1**	0
1	**a**	1	0	**1**
2	**b**	1	1	0
3	**d**	1	1	1

7.取MinHash特征：经过h_3打乱后，记录各文本自上而下第一个1所在行号，结合h_1, h_2打乱的结果，得到的MinHash特征值如下表所示。

MinHash	S_1	S_2	S_3
h_1	0	0	2
h_2	0	1	0
h_3	0	0	1

8.· · ·

9.假设我们只进行了3次打乱，也即我们只取3个MinHash特征，那么文本原先的4维特征就变成了3维特征表示，实现了降维。

10.计算MinHash相似度，公式如下所示，其中n为MinHash特征数，i为MinHash特征序号，$I[h_i(S_1)=h_i(S_2)]$指"当文本A和文本B的第i个MinHash特征相等时取1，否则取0"。

$$Sim(A,B)=P(MinHash(A)=MinHash(B))=\frac{\sum_{i=1}^{n}I[h_i(A)=h_i(B)]}{n}$$

结合第7点，由上式可得，S_1,S_2,S_3两两间的相似度如下所示。

$$Sim(S_1,S_2)=P(MinHash(S_1)=MinHash(S_2))=\frac{I[h_1(S_1)=h_1(S_2)]+I[h_2(S_1)=h_2(S_2)]+I[h_3(S_1)=h_3(S_2)]}{n}=\frac{1+0+1}{3}=\frac{2}{3}$$

$$Sim(S_1,S_3)=P(MinHash(S_1)=MinHash(S_3))=\frac{I[h_1(S_1)=h_1(S_3)]+I[h_2(S_1)=h_2(S_3)]+I[h_3(S_1)=h_3(S_3)]}{n}=\frac{0+1+0}{3}=\frac{1}{3}$$

$$Sim(S_2,S_3)=P(MinHash(S_2)=MinHash(S_3))=\frac{I[h_1(S_2)=h_1(S_3)]+I[h_2(S_2)=h_2(S_3)]+I[h_3(S_2)=h_3(S_3)]}{n}=\frac{0+0+0}{3}=0$$

II. MinHash相似度取代Jaccard相似度的合理性

这边先提供一个比较好理解的角度：

MinHash使用随机排列来生成特征矩阵，当两个文本集合的Jaccard相似度很高时，往往意味着它们的交集会很大，相应地，即使经过随机地行打乱，也依然很轻易就能达成"两文本（列）自上而下首个1位于同行"的情况——以行号为准的MinHash特征也将是相等的，这将致使最后计算的MinHash相似度也较高。以下述简单例子来说明。

1.有高度重合的两个文本$S_1 = \{a, b, c, d\}$和$S_2 = \{a, b, c, d, e\}$，可构建特征矩阵如下。

行号	元素	S_1	S_2
0	**a**	1	1
1	**b**	1	1
2	**c**	1	1
3	**d**	1	1
4	**e**	0	1

2.其Jaccard相似度为0.8，具有较高的相似度。

$$Jac(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{4}{5} = 0.8$$

3.按照MinHashing操作对该特征矩阵进行任意地行打乱（下表以3次为例），因为两文本存在诸多重合项，因此在多数时候随机打乱后"自上而下取首1的行号"往往是行号相同的。

行号	h_1	S_1	S_2	h_2	S_1	S_2	h_3	S_1	S_2
0	**b**	**1**	**1**	**d**	**1**	**1**	**a**	**1**	**1**
1	**c**	1	1	**a**	1	1	**e**	0	1
2	**a**	1	1	**e**	0	1	**b**	1	1
3	**d**	1	1	**c**	1	1	**c**	1	1
4	**e**	0	1	**b**	1	1	**d**	1	1

4.所以取出来的MinHash特征也是一致的。

MinHash	S_1	S_2
h_1	0	0
h_2	0	0
h_3	0	0

5.由此计算MinHash相似度为1，结论为两文本高度相似——MinHash相似度在度量趋势上与Jaccard相似度是一致的，所以可以近似替代Jaccard相似度来度量文本的重合程度。

$$Sim(S_1,S_2)=\frac{I[h_1(S_1)=h_1(S_2)]+I[h_2(S_1)=h_2(S_2)]+I[h_3(S_1)=h_3(S_2)]}{n}=\frac{1+1+1}{3}=1$$

相对地，当两个文本集合的Jaccard相似度很低时，往往也意味着它们的交集很小，经过随机地行打乱，“两文本（列）自上而下首1位于同行”的情况会更加罕见，下方举简单例子进行说明。

1.有几乎不重合的两个文本$S_1=\{a,b,c\}$和$S_2=\{c,d,e\}$，可构建特征矩阵如下。

行号	元素	S_1	S_2
0	**a**	1	0
1	**b**	1	0
2	**c**	1	1
3	**d**	0	1
4	**e**	0	1

2.其Jaccard相似度为0.2，具有不太高的相似度。

$$Jac(S_1,S_2)=\frac{|S_1\cap S_2|}{|S_1\cup S_2|}=\frac{1}{5}=0.2$$

3.按照MinHashing操作对该特征矩阵进行任意地行打乱（下表以3次为例），因为两文本存在诸多非重合项，因此在多数时候随机打乱后“自上而下取首1的行号”往往是行号不同的。

行号	h_1	S_1	S_2	h_2	S_1	S_2	h_3	S_1	S_2
0	**b**	**1**	0	**d**	0	**1**	**a**	**1**	0
1	**c**	1	**1**	**a**	**1**	0	**e**	0	**1**
2	**a**	1	0	**e**	0	1	**b**	1	0
3	**d**	0	1	**c**	1	1	**c**	1	1
4	**e**	0	1	**b**	1	0	**d**	0	1

4.所以取出来的MinHash特征也是高度不一致的。

MinHash	S_1	S_2
h_1	0	1
h_2	1	0
h_3	0	1

5.由此计算MinHash相似度为0，结论为两文本不相似——MinHash相似度在度量趋势上与Jaccard相似度是一致的，所以可以近似替代Jaccard相似度来度量文本的重合程度。

$$Sim(S_1,S_2)=\frac{I[h_1(S_1)=h_1(S_2)]+I[h_2(S_1)=h_2(S_2)]+I[h_3(S_1)=h_3(S_2)]}{n}=\frac{0+0+0}{3}=0$$

有的资料会基于MinHash选取策略的角度来论述。

1.有文本$S_1=\{a,d,e\}$和文本$S_2=\{c,e\}$，所有语料的总词集合为$U=\{a,b,c,d,e\}$，则有下表。

行号	元素	S_1	S_2
0	**a**	1	0
1	**b**	0	0
2	**c**	0	1
3	**d**	1	0
4	**e**	1	1

2.定义一种用以描述“矩阵每行特征对齐”的类别：如果该行特征均为1，则该行可划分为**X**类；如果该行特征一个为1、另一个为0，则该行将划分为**Y**类；如果该行特征均为0，那么该行会划分为**Z**类。具体可如下图所示。

行号	元素	S_1	S_2	类别
0	**a**	1	0	Y
1	**b**	0	0	Z
2	**c**	0	1	Y
3	**d**	1	0	Y
4	**e**	1	1	X

3.结合第2点的定义，对于Jaccard相似度的计算来说，实际上就是在计算下式。

$$Jac(S_1,S_2)=\frac{|S_1\cap S_2|}{|S_1\cup S_2|}=\frac{|X|}{|X|+|Y|}$$

4.因为MinHash特征的选取策略是“自上而下第一个1（非0）的行号”，Z类对这样的策略取法毫无影响（因为自上而下取非0），因此总量不考虑$|Z|$，且$P(MinHash(A)=MinHash(B))$关注的是$MinHash(A)=MinHash(B)$的概率，也即X的概率，由大数定律可得，采样到一定程度时，X的频率接近其概率，在这个选取策略中，X的频率计算公式为$|X|/(|X|+|Y|)$，也就是第3点提及的Jaccard相似度。

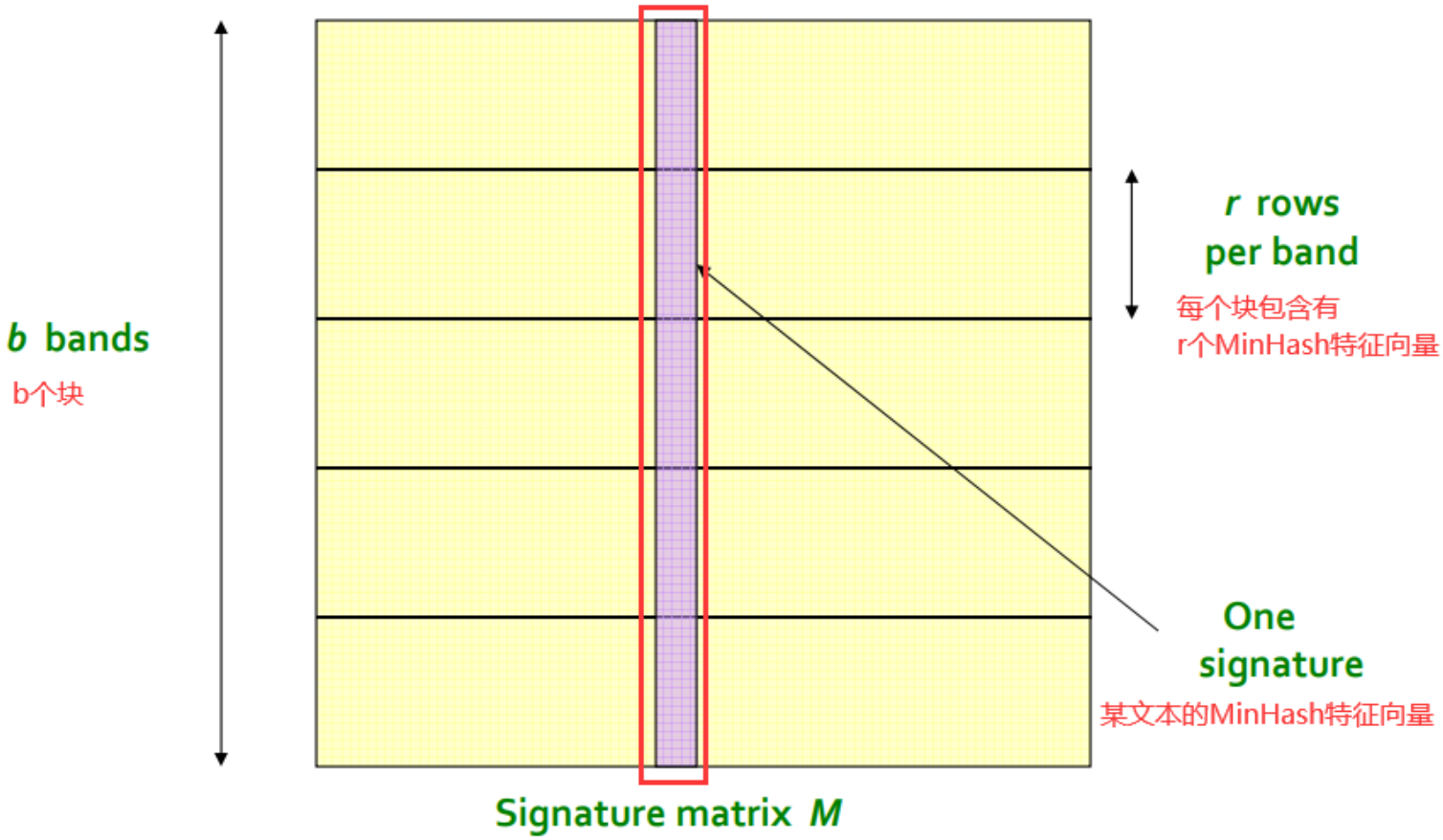
4. MinHash-LSH相似度

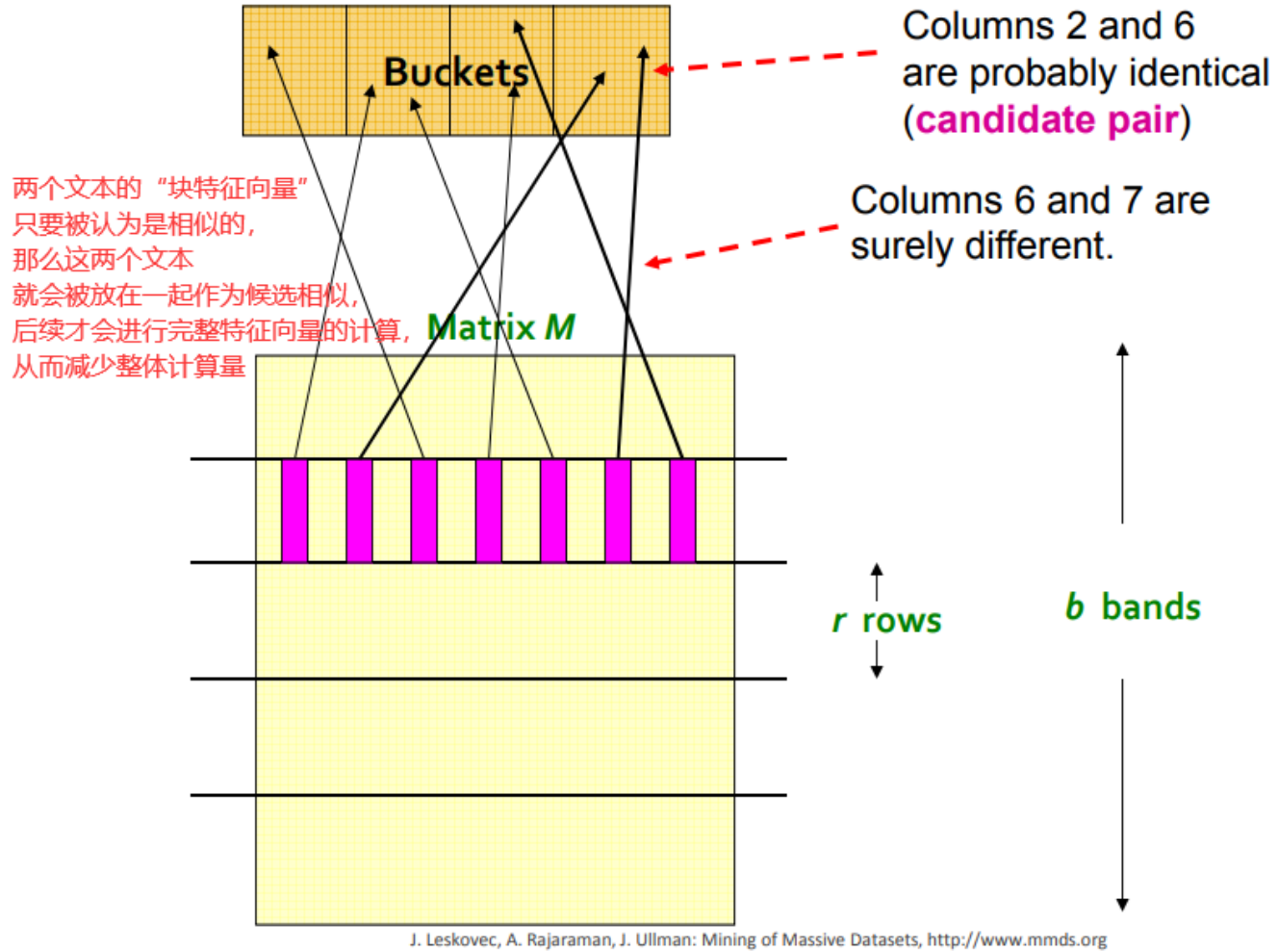
LSH指Locality Sensitive Hashing（局部敏感哈希）。上述的MinHash虽然相较于直接对原文计算Jaccard已经有不小的特征维数方面的优化，但仍需对确切的两两文本进行相似度计算，这依然是个十分耗时的工程。因此还可在Minhash的基础上引入LSH来进行局部相似度计算，从而缩小两两文本的计算范围。

I. LSH的“粗筛”

LSH的基本思想是将MinHash特征进行分块得到“块特征”，通过度量分块特征之间的相似度来决定相应的两个文本是否还有必要进行相似度计算，如果两个文本的块特征已然不相似，那么很大概率这两个文本的完整MinHash特征就是不相似的，后续就没有必要进行精确的MinHash相似度计算——这个结论是基于概率推广的推导得到的。
大体上来看在MinHash中引入LSH更像是对文本的相似度计算候选集进行了粗筛。在未引入LSH前，即使是使用了MinHash进行低维计算，也还是需要对两两文本进行计算，各个文本的相似度计算候选集是“除自己之外的其他所有文本”，在海量文本的场景下也仍旧是相当夸张的计算量；通过引入LSH，先使用更轻量更低维（块特征）的计算来对各个文本的相似度计算候选集进行粗筛，使得各个文本的相似度候选集不再是“除自己之外的其他所有文本”那么大的量级，由此从相似度计算频次的角度减少了计算量。

下图取自 http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf


b bands
b个块
r rows
per band
每个块包含有
r个MinHash特征向量
One
signature
某文本的MinHash特征向量
Signature matrix M



II. LSH的合理性

主要是针对块相似概率与全相似概率进行的讨论，可作为扩展阅读来学习。

https://blog.csdn.net/u013179327/article/details/38794609

https://blog.csdn.net/Vihagle/article/details/119319039

5. MinHash LSH代码Demo

In [73]:
```
# !python -m pip install datasketch
# !python -m pip install jieba
```

In [105…
```
from datasketch import MinHash, MinHashLSH
import jieba
import re

query = "有些鸟儿是永远关不住的,因为它们的每一片羽翼上都沾满了自由的光辉。"

sentences = [
    "有些鸟儿是永远不会被关在牢笼里的，因为它们的每一片羽毛都闪耀着自由的光辉。",
    "这世上到处都是害怕主动迈出第一步的孤独之人。"
]

regex = re.compile(', |。')

def split_word(sentence):
    global regex
    return [word for word in jieba.lcut(re.sub(regex, ' ', sentence)) if word.strip()]
```

In [106…
```
# 分词
query_lcut = split_word(query)
sentences_lcut = [split_word(sentence) for sentence in sentences]
print(query_lcut)
print(sentences_lcut)
```

```
['有些', '鸟儿', '是', '永远', '关不住', '的', ',', '因为', '它们', '的', '每', '一片', '羽翼', '上', '都', '沾满', '了', '自由', '的', '光辉']
[['有些', '鸟儿', '是', '永远', '不会', '被关', '在', '牢笼', '里', '的', '因为', '它们', '的', '每', '一片', '羽毛', '都', '闪耀着', '自由', '的', '光辉'], ['这', '世上', '到处', '都', '是', '害怕', '主动', '迈出', '第一步', '的', '孤独', '之', '人']]
```

In [107…
```
# 实例化LSH管理器
# threshold=0.5，指Jaccard相似度阈值设置为0.5，即返回大于0.5的待匹配句子
# num_perm=128，指Hash置换函数的个数，如需提高精度可适当提高，如256
threshold = 0.5
num_perm=128
lsh = MinHashLSH(threshold=threshold, num_perm=num_perm)
for idx, sentence_lcut in enumerate(sentences_lcut):
    # 对每个待匹配句子实例化1个MinHash()对象
```

```
    minhash = MinHash()
    # 将文本的词序列传入MinHash对象
    minhash.update_batch([word.encode('utf-8') for word in sentence_lcut])
    # 将MinHash对象传入LSH对象进行管理
    lsh.insert("minhash_sentence_{}".format(idx+1), minhash)
```

In [108…

```
# 调用LSH实例的keys属性，查看目前存有的文本的key
list(lsh.keys)
```

Out[108]:

```
['minhash_sentence_1', 'minhash_sentence_2']
```

In [109…

```
# query语句也需要进行MinHash实例化
minhash_query = MinHash()
minhash_query.update_batch([word.encode('utf-8') for word in query_lcut])
```

In [110…

```
# 调用LSH实例的query方法对之前传入的待匹配句子进行相似文本匹配
simi_result = lsh.query(minhash_query)
print("近似Jaccard相似度>{}的句子有：\n{}".format(threshold, simi_result))
```

```
近似Jaccard相似度>0.5的句子有：
['minhash_sentence_1']
```

In [111…

```
# 调用LSH实例的remove方法删除其中存有的文本，传入参数为相应文本的key
lsh.remove('minhash_sentence_1')
```

In [112…

```
# 调用LSH实例的keys属性，查看目前存有的文本的key
list(lsh.keys)
```

Out[112]:

```
['minhash_sentence_2']
```