iTAPE工具报告

191250132 唐萁

1.简介

iTAPE是研发者为了能够为bug report自动生成标题而制作的工具。在bug报告的质量中，标题的良好与否是一个重要的指标。一个好的标题应该能够让人们从中快速而唯一的识别出错误报告的类型，但在现实生活尤其是开源社区中，存在大量标题质量较差的bug报告。

iTAPE在OpenNMT（一个Seq2Seq总结模型）的基础上进行优化，解决了训练数据缺失和低频令牌无法识别的问题，实现了为bug report生成优质的标题摘要。

2.核心算法

2.1核心理念

iTAPE的目的本质上是为代码世界问题报告生成一句话总结的任务，其核心理念是对开源社区的bug report进行筛选和低频术语令牌处理后，训练一个在计算机问题世界可以对文章总结中心内容的Seq2Seq模型。

2.2创新点

过去的一些相似工作主要是以有监督的方法、用众包引发的属性来总结bug报告，或是构建无监督方法为报告制定包含几句话的总结或介绍。iTAPE是第一个考虑为bug report生成一句话标题，并且基于Seq2seq总结的工具。

一般的摘要任务主要的应用场景是新闻摘要，该应用领域有大量数据集和性对同一的术语频率分布，但为bug report生成标题面临着两个计算机特定领域的难点：

1. **不存在适合任务的现成的示例集：**

   现实的样本存在大量的问题，如标题过长或过短、标题不是bug report内容的良好总结、标题是从内容中直接摘取的错误信息。

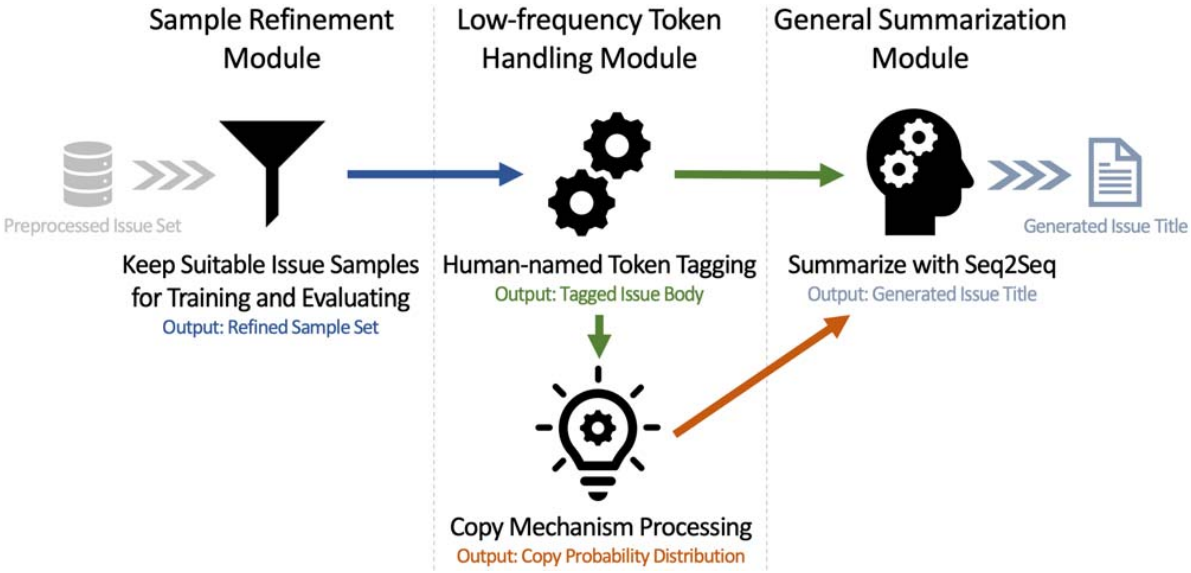
2. **普通情况下神经模型无法对于人为命名的令牌（如标识符和版本号）有良好的的理解：**

   在bug report中存在着大量的人为命名令牌（human_named tokens)，即标识符和版本号，它们在报告中是低频率但非常重要的，但在普通的生活语言场景下（如新闻）它们是可以被忽略的。而基本的seq2seq模型通过反复调整**固定词汇表**中标记的参数权重，如word Embedding vectors，来学习理解令牌序列的语义.因此这些低频令牌通常不能被模型很好的理解，甚至常常出现“Out Of Vocabulary"(OOV)问题，结果是被简单地替换为统一的令牌。在这种OOV到UNK令牌的映射下，单词的所有信息都会丢失，单词的每个OOV单词都具有相同的表示形式，这对于内容理解是不利的。

工具运用了 3个启发式的规则来改进获得的bug report样本，为任务构建了第一个基准数据集；用一种轻量级的“标记”方法，结合复制机制，帮助模型能够有效地处理低频的人为命名令牌；在完成以上两项的基础上，工具能够基于Seq2Seq模型进行良好的总结。

2.3具体算法

为了实现上述功能，工具共有3个模块：样本细化模块、低频令牌处理模块和一般总结模块



2.3.1样本细化

样本细化基于3个启发式规则：

1. **标题的单词数少于5或大于15，或标题含有url的bug report样本会被过滤**：

   过短的标题不能提供足够的摘要信息；过长的和包含url的标题增加了阅读理解的负担，但是对于总结文章中心并没有太大的帮助。（例子如下图）

Case A	*Case C*
Title: Cuda 3.0 not working **Body:** When installing from pip tensorflow GPU version (Linux), minimum cuda version is 3.5. Install instructions have been changed to: TensorFlow GPU support requires having a GPU with NVidia Compute Capability>=3.0 However, gpu_device.cc still requires cuda 3.5: std::vector supported_cuda_compute_capabilit ies={ CudaVersion("3.5"), CudaVersion("5.2")}; *(omit trivial code)* *Collected from https://github.com/tensorflow/tensorflow/issues/1440*	**Title:** To support nuxt-optimized-images module - https://www.bazzite.com/docs/nuxt-opti mized-images/ **Body:** I can't use nuxt-optimized-images module with Storybook. The 'include' [query](https://www.bazzite.com/docs/nuxt-optimized-images/usage/#include) doest not works in Storybook component preview. Solution 1: To detect nuxt-optimized-images module and setup necessary webpack config. Solution 2: To write a setup guide in the docs. *Collected from https://github.com/storybookjs/storybook/issues/7060*

2. **如果标题中超过70%的单词无法在报告具体内容中找到，该样本被过滤**：

   当title中的很大一部分缺失在bug report的body中，title不太可能是对于body的良好总结。（例子如Case D，titile描述了作者想要的东西，而body描述了作者为什么想要这个特性，body是title的补充而非总结）

Case D

Title: Give unsaved Markdown files a title, using the first header in the file

Body:

This feature exists in Sublime, and I think it would be useful in VSC as well.

I'll often open several Markdown files without saving them, just to keep some temporary notes, and if they are all untitled, then it's hard to tell them apart. But this feature has been very useful in Sublime, since it's easy to identify them when they all have titles.*(omit an image)*

Collected from https://github.com/microsoft/vscode/issues/46204

3. **如果标题中存在一个超过标题长度70%的子序列，可以完全匹配报告具体内容的一部分，该样本被过滤**：

这种情况是由于bug report的作者直接抽取body中的一个句子作为标题，而不是重写一个单独的简短句子，仅仅赋值了错误信息，而不提供其他信息，这对于一句话总结任务来说是不合适的。（例子如Case E)

Case E

Title: RequestTaskMap consistency error: no request corresponding to task found
Body:
Whenever I kill app when API is going on then when I launch application once again then app crashes for as loon as I try to open it. I have to delete app and install it again. Below is the line from RequestTaskMap swift file where app crashes: Error: Fatal error: RequestTaskMap consistency error: no request corresponding to task found.: file > *(omit lengthy path)*/Project/Pods/Alamofire/Source/RequestTaskMap.s wift, line 61
Tried to Invalidate Alamofire sessions and tried to cancel requests too on application-WillTerminate method but still issue persists.

Collected from https://github.com/Alamofire/Alamofire/issues/3013

上述三个规则可以帮助构建一个更好的生成性能的训练集和一个更可靠的评估结果的测试集，最后为整个任务构建一个基准测试集。

2.3.2低频令牌处理

1. **复制的机制**

   复制机制使用网络指针为令牌生成提供除了“从固定词汇表学习选择”的另一种访问：从输入中复制任何必要的令牌。（但是，由于缺少相应的单词嵌入向量，它无法存储语义信息来理解这些令牌）。因此，具有复制机制的工具既可以根据固定词汇表的学习从其中生成目标令牌，也可以根据整个输入序列的注意力直接从输入中复制任何必要的令牌。

   复制开关值的计算方法：

$$p_{copy}^{i} = \sigma(\omega_c^T c_i + \omega_h^T h_i' + \omega_x^T x_i'^{emb} + b_{copy})$$

   其中**h**i和**x**i分别是当前时间步长的隐藏状态和解码器的输入嵌入量,**c**i是编码器所有隐藏状态的加权和，σ是sigmoid函数，**ω**c、**ω**h、**ω**x、**b**copy分别是可学习参数。

   复制令牌的概率分布为：$$P_{copy}^{i}(y_i) = \sum_{i:x_j=y_i} \alpha_j^i$$

   其中**y**i是当前的目标令牌，x是输入的body序列，**a**i为输入序列的单签注意力分布

2. **给human-named token打标签**

   检测文本中的标识符和版本号，在标识符的前后分别插入“”和“<I$>”标记，在版本号的前后分别插入“”和“<V$>”标记。在插入这协标记后，神经模型能够通过学习这些标签的单词嵌入来存储它们的信息，这些信息可以指示所有相应类型的人类命名的标记的语义意义，这有助于模型理解和复制人类命名的标记。

示例如图：

Issue body BEFORE tagging:
When I add the ad_mob (firebase_admob : ^ 0.6.1 + 1) Android emulator crashing before any code executes. As soon as I take it out, it works. I tried to start LogCat, but no logs are generated (I am using VS Code)
I have the following plugins working successfully with firebase
flutter_crashlytics : ^ 0.1.1
firebase_analytics : ^ 1.0.4
But ad_mob just won't work.

VersionNo. (<V>, <V$>): 0.6.1, 0.1.1, 1.0.4
Identifiers (<I>, <I$>): ad_mob, firebase_admob, firebase_analytics, flutter_crashlytics, LogCat

---

Issue body AFTER tagging:
When I add the <I> ad_mob <I$> (<I> firebase_admob <I$> : ^ <V> 0.6.1 <V$> +1) Android emulator crashing before any code executes. As soon as I take it out, it works. I tried to start <I> LogCat <I$>, but no logs are generated (I am using VS Code)
I have the following plugins working successfully with firebase
<I> flutter_crashlytics <I$> : ^ <V> 0.1.1 <V$>
<I> firebase_analytics <I$> : ^ <V> 1.0.4 <V$>
But <I> ad_mob <I$> just won't work.

2.3.3总结模型

使用结合了复制机制的基于注意力RNN Encoder-Decoder模型作为seq2seq总结模型，具体使用了该结构体体系的OpenNMT-py模型实现，OpenNMT-p提供了一个SOTA级别的模型设置和便捷的API。

模型主要由编码器网络、解码器网络、注意层、最终生成器四个部分组成。

1. 源序列（即报告body的令牌序列）x=(x1,x2,...,x_Nx)，会被送入编码器产生一组编码隐藏向量h={h1,h2,...,h_Nx}和一个向量c，h和c分别表示每个编码时间步长的处理状态和整个输入的上下文信息。
2. 接下来，解码器接收c，并处理生成解码隐藏向量 h'={h1',h2',...h'_Nx}。同时，注意层接收 h 和 hi'来计算源序列上的注意得分向量 ai。
3. 然后生成器将其与 hi' 结合，计算第 i 个目标；令牌在固定词汇表上的生成概率分布。具体来说，我们有 $P^i_{gen} = softmax(\omega'(\omega[h'_i, c_i] + b) + b')$

   其中 $c_i = \sum_j^{N_x} \alpha^i_j h_j$，是编码器隐藏状态的加权和，hi'是当前解码器隐藏状态，ω, ω', b, b'是参数，[,]代表连接运算。
4. 之后生成器集合复制概率分布，得到最终的概率分布如下
   $$P^i = (1 - p^i_{copy})P^i_{gen} + p^i_{copy}P^i_{copy}$$
5. 最后，执行 ArgMax 操作生成第 i 个代牌 yi，所有的输出令牌都包含在目标序列 y=(y1,y2,...y_Ny)中，并作为一个句子的总结。这里 Ny 表示目标序列的长度。在推断时，将之前的输出单词 yi-1输入到解码器以生成当前单词 yi，直到生成了句子结束令牌(End-OfSentence token,)。

6. 另外，训练时，将前一个时间步长的y'i-1输入到解码器，进行教师强迫训练，因 此训练损失计算为：

$$Loss = -\frac{1}{N_{y'}} \sum_{i=1}^{N_{y'}} \log p(y'_i | y'_1, \ldots, y'_{i-1}, c)$$

此外，还采用了一些辅助方法来提高一般性能：使用长期短期记忆(LSTM)细胞作为 RNN 细胞，因为它记忆过去的时间步信息的出色能力。编码器使用双向 LSTM，解码器使用单向 LSTM。为了加速模型的收敛性，我们使用了预先训练过的GloVe向量作为初始单词嵌入权值。GloVe 向量在一般新闻语料库上进行训练， 从而表示标记的一些上下文语义意义。应用具有自动学习速率预热和衰减的 Adam 优化器来稳定梯度降序过程。编码器和解码器共享词汇表和单词嵌入，以减轻学习任务。在推断过程中，使用光束搜索来选择最佳输出，并应用 ngram 重复块来避免重复短语。

3.具体实现

3.1输入输出

3.1.1输入

该工具的原始输入为从github获取的issue样本，每个issue样本包括四个属性，依次为： `number`，`title`， `body`，`repo`，以json文件的形式储存样本序列（文件：“github.json”）。github社区的isuue被视为bug report，title为bug报告的标题，body为bug报告的具体内容，这是在后续操作中必须用到的属性。

代码的目标是建立一个模型，从这些title-body中的合适样例学会，从body预测出合适的title。

3.1.2输出

在模型建立完成后，模型会根据测试集的每一个body预测一个title，将所有的title有序地储存到一个TXT文件中，作为输出.

3.2功能模块

3.2.1预处理

0-0-process.py 用来进行数据的预处理，以去除难以标记的样本并裁剪数据中的杂项内容。它从“**github.json**”读取issue的列表all_issues，扫描所有issue进行预处理，得到了一个初步处理的issue列表 preprocessed_issues，存储到数集**”preprocessed_issues.json“**中：

```
#1.数据预处理：去除难以标记的样本并裁剪数据中的杂项内容
print("preprocessing...")
preprocessed_issues = []#储存预处理之后的样本list
for idx, issue in enumerate(tqdm(all_issues)):
    #扫描所有issue样本
    #1.1用占位符替换body中的杂项信息：  url、超链接、图像、 代码片段和以未选中的Markdown开头的行复选框
    issue['body'], _ = improve_body(issue['body'])
    # 1.2过滤掉body不合适的issue样本：body长度不合适 or 在markdown的代码区域的外含有HTML标签
    #fiter_body()返回结果为true，issue应被过滤，跳过当前循环，扫描下一个issue(idx+1)
    issue_body_tokenize = nltk.word_tokenize(issue['body'])# nltk.word_tokenize（string）：以空格形式实现分词
    if filter_body(issue['body'], issue_body_tokenize):
        continue
    #1.3去掉title中的“tag：  ”，“[tag]”，和markdown重点标记符“**”
    issue['title'] = improve_title(issue['title'])
    #1.4将预处理之后的issue存入preprocessed_issues
    preprocessed_issues.append(issue)
with open("preprocessed_issues.json", "w") as f:
    json.dump(preprocessed_issues, f)
print("preprocess finish. Begin to obtain", len(preprocessed_issues), "handlable issues.")
```

1. **body中的非文本内容会被替换为占位符**（improve_body()函数实现）

一些问题正文包含 url、超链接、图像、 代码片段、以未选中的markdown复选框开头的行、换行符。尽管这种非文本信息通常有助于更深入地研究这些问题，但它们给标记化和模型处理带来了许多不必要的复杂性。实际上，对于大多数高质量的body来说，仅凭文本信息就足以进行总结。

具体的，运用正则表达式找到上述内容，然后替换为对应的占位符，正则表达式在作者的论文中给出：

Type	Regular Expression or Condition
Code Snippet	```(?:.|\n)+?```
Image	\!\[(.*?)\](.+?)
HyperLink	(?<!\!)\[(.*?)\](.+?)
Line with Unchecked Box	- +[ ].*
URL	(https?|ftp)://[^\s/$.?#].[^\s]*
“[Tag]” at Beginning	^(\s*\[.*?\])+
“Tag: ” at Beginning	^.*?: *and length less than half of title length*

函数实现及其解释如下图：

```
'''
1.1用占位符替换body中的杂项信息， url、超链接、图像、 代码片段和以未选中的Markdown复选框开头的行
输入：某个issue的body
输出：将body中杂项信息替换为占位符后的body + 占位符数目
'''
def improve_body(origin_str, return_max_cnt = False):
    maxcnt = 0 #占位符的数目

    #(1)将代码片段转换为占位符 “phofcode"
    search = re.compile("```(?:.|\n)+?```") #re.compile()将正则表达式转化为对象，多次调用一个正则表达式就重复利用这个正则对象->re.se
    if return_max_cnt:
        maxcnt = max(maxcnt, len(search.findall(origin_str)))
    origin_str = search.sub(" phofcode ", origin_str)#re.sub(pattern, repl, string),pattern：正则表达式，被正则表达式搜索到的目

    #(2)将图像转换为占位符 text+"phofimage"
    search = re.compile("\!\[(.*?)\]\(.+?\)")
    if return_max_cnt:
        maxcnt = max(maxcnt, len(search.findall(origin_str)))
    origin_str = search.sub(lambda x: " " + x.group(1) + " phofimage ", origin_str)#group(1) 列出第一个括号匹配部分

    #(3)将hyper link转化为占位符 text+"phofhyperlink"
    search = re.compile("(?<!\!)\[(.*?)\]\(.+?\)")
    if return_max_cnt:
        maxcnt = max(maxcnt, len(search.findall(origin_str)))
    origin_str = search.sub(lambda x: " " + x.group(1) + " phofhyperlink ", origin_str)

    #(4)将URL转换为占位符 “phofurl"
    search = re.compile("(https?|ftp)://[^\s/$.?#].[^\s]*")
    if return_max_cnt:
        maxcnt = max(maxcnt, len(search.findall(origin_str)))
    origin_str = search.sub(" phofurl ", origin_str)

    #(5)将markdown中以未选中的选框开头的行，删除（替换为“”）
    search = re.compile("- +\[ \].*") #line with unchecked tickbox in markdown (template provided useless desc) -> (rem
    origin_str = search.sub("", origin_str)

    #(6)避免***abc***被NLTK识别为令牌
    search = re.compile("(\*{1,})([^\s]+?)(\*{1,})")
    origin_str = search.sub(lambda x: " " + x.group(1) + " " + x.group(2) + " " + x.group(3) + " ", origin_str)

    #(7)换行符替换为占位符 “phofnewline"
    search = re.compile("(\n\r)|(\r\n)|(\n)")
    if return_max_cnt:
        maxcnt = max(maxcnt, len(search.findall(origin_str)))
    origin_str = search.sub(" phofnewline ", origin_str)

    if return_max_cnt:
        return origin_str, maxcnt
    else:
        return origin_str, 0
```

2. **过滤body不合适的样本**（filter_body()函数实现）

   - 丢弃body长度小于30或大于300的issue样本：

     过短的body无法为总结标题提供足够的问题信息，过长的body往往携带了处理难度较大的标记代码字段外的日志

   - 丢弃body中含有标记代码字段之外的HTML标签的issue样本

     美化内容但不携带关键信息的HTML标签（如"）和一些描述性文字（如"error throw when bc"）的正则表达式高度相似，无法精准取出HTML标签，因此选择直接丢弃这类issue而不是替换HTML标签。

     正则表达式为：

     “<[^<]+?>”.

     函数实现及其解释如下图：

```
'''
1.2过滤掉body不合适的issue样本：body长度不合适 or 在markdown的代码区域的外含有HTML标签
输入：某个issue的body + body的词语list
输出：当body不合适（该issue样本应该被丢弃），返回TRUE，否则FALSE
'''
def filter_body(issue_body, issue_body_tokenize):
    length = len(issue_body_tokenize)
    if length < 30:
        return True
    if length > 300:
        return True
    if len(re.findall("<[^<]+?>", issue_body)) > 0:
        return True
    return False

'''
```

3. **删除title中的开始标签字符串和markdown强调标志**（improve_title()函数实现）

   一些问题标题以“tag：”或“[tag]”开始，然后是描述性文本内容，其中“tag”可以是问题类型(如。“Feature：”、“[OSX]”等)或特定的异常名称;此外，一些作者会在题目中写有Markdown重点标志符，如“*”“**”，但标题很少被markdown翻译，因此两类信息都被删除。

   正则表达式在第1点的图中给出

   函数实现及其解释如下图：

```
'''
1.3去掉title中的tag标签和markdown重点标记符“**”
输入：某个issue的title
输出：修正后的title
'''
def improve_title(issue_title):
    original_len = len(issue_title)
    #移除在title开头的 “[tag]”
    issue_title = re.sub("^(\s*\[.*?\])+", "", issue_title)
    #去掉“tag: ”：寻找冒号所在的位置，并且要小于1/2的titile长度才能判定为符合情况
    pos = issue_title.find(": ")
    if -1 < pos < len(issue_title) - original_len / 2:
        issue_title = issue_title[pos + 1:].strip() #strip(): 移除字符串收尾的空格
    # 再次移除“[tag]”
    issue_title = re.sub("^(\s*\[.*?\])+", "", issue_title)
    #移除markdown的重点标记符“**”
    issue_title = re.sub("(\*{1,})(.+?)(\*{1,})", lambda x: x.group(2), issue_title)
    return issue_title.strip()
```

3.2.2样本细化模块

0-1 refine_and_deal_human_named_token.py实现了用3个启发式规则（在2.3.1中已经对3个启发式规则进行了说明）过滤样本。读取"**preprocessed_issues.json**"文件得到preprocessed_issues列表，检查每个issue是否符合任意一个规则：

```
for idx, issue in enumerate(tqdm(preprocessed_issues)):
    issue_body_tokenize = nltk.word_tokenize(issue['body'])
    issue_title_tokenize = nltk.word_tokenize(issue['title'])
    # 将所有单词转化为小写
    issue_title_words = [x.lower() for x in issue_title_tokenize if re.match("\S*[A-Za-z0-9]+\S*", x)]
    issue_body_words = [x.lower() for x in issue_body_tokenize if re.match("\S*[A-Za-z0-9]+\S*", x)]
    # # 3.样本细化，使用三个启发式的规则
    # 当某个issue符合任何一个rule，该issue被过滤,跳过当前循环，扫描下一个issue
    if (rule1checker(issue['title'], issue_title_words)) \
            or (rule2checker(issue_title_words, issue_body_words)) \
            or (rule3checker(issue_title_tokenize, issue_body_tokenize)):
        continue
```

1. 启发式规则一：**标题的单词数少于5或大于15，或标题含有url的bug report样本会被过滤**：检查title的单词数，用正则表达式搜索title中是否有url

```
'''
3.1启发式规则1：过滤title单词数<5 or >15，或title含有url的issue
输入：某个issue的title和title的单词list
输出：当issue需要被过滤，返回TRUE，否则FALSE
'''
def rule1checker(issue_title, issue_title_word):
    length = len(issue_title_word)
    if length < 5:
        return True
    if length > 15:
        return True
    if len(re.findall("(https?|ftp)://[^\s/$.?#].[^\s]*", issue_title)) > 0:
        return True
    return False
```

2. 启发式规则二：**如果标题中超过70%的单词无法在报告具体内容中找到，该样本被过滤**：检查title中的每一个word是否在body中存在，计算不存在的比例

```
'''
3.2启发式规则2：过滤title中超过70%的单词无法在body中找到的issue
输入：某个issue的title和body各自的的单词list
输出：当issue需要被过滤，返回TRUE，否则FALSE
'''
def rule2checker(issue_title_words, issue_body_words):
    body_words_set = set(issue_body_words)
    cnt_each = 0
    for word in issue_title_words:
        if word in body_words_set:
            cnt_each += 1
    if cnt_each <= len(issue_title_words) * 0.3:
        return True
    return False
```

3. 启发式规则三：**如果标题中存在一个超过标题长度70%的子序列，可以完全匹配报告具体内容的一部分，该样本被过滤**

```
'''
3.3启发式规则3：过滤掉title中存在一个超过其70%长度的子序列，能够完全匹配body中某个句子的issue
输入：某个issue的title和body各自的的单词list
输出：当issue需要被过滤，返回TRUE，否则FALSE
'''
def rule3checker(issue_title_tokenize, issue_body_tokenize):
    title_words = [x.lower() for x in issue_title_tokenize]
    body_words = [x.lower() for x in issue_body_tokenize]
    exp = ""
    # build substring location RE: (\s+AA)(\s+BB)(\s+CC)(\s+DD) to match AA BB CC (more p
    for _ in title_words:
        _ = re.escape(_)
        exp += "(" + "\s+" + _ + ")?"
    re_iter = re.compile(exp)
    each_cnt = 0
    for s in re_iter.finditer(" " + " ".join(body_words)):
        each_each_cnt = 0
        for _ in s.groups():
            if _ is not None:
                each_each_cnt += 1
        each_cnt = max(each_cnt, each_each_cnt)
    if each_cnt >= len(title_words) * 0.7:
        return True
    return False

'''
```

3.2.3低频令牌处理模块

0-1 refine_and_deal_human_named_token.py实现了对于低频令牌打标签的工作，将preprocessed_issues的列表项按照8:1:1的比例分为了train, valid, test（训练、验证、测试）三类，对于每个issue进行操作，最后得到三个列表valid_issues_train, valid_issues_val, valid_issues_test，将三个列表存储到一个列表save中并存入**"good_issues.json"**，得到了可靠的、适合机器学习的数据集：）

1. 在issue原本的key（如"title" "body"）添加新的key "_spcyok"，key对应的value为一个字典result，结构为：

   issue{"title":"xxx", "body":"xxx", ... ,

   "__spcyok":{"ver":{版本号:[order,出现次数]},{...}, {...}.. ,

   "idt":{版本号:[order,出现次数]},{...}, {...}.. } }

   获得"_spcuok"的value通过函数get_versuiob_list()和get_identifier_list()：

```
# 4.1在issue里添加新的key"_spctok"，value为一个下属字典。下属字典有两个key:"ver""idt"，value分别为该issue的body中版本号和标识符的result字典
issue["_spctok"] = {}
issue["_spctok"]["ver"] = get_version_list(" " + issue['body'] + " ")
issue["_spctok"]["idt"] = get_identifier_list(" " + issue['body'] + " ")
```

```
'''
输入：字符串string
输出：一个result字典：字典的每个key是版本号，其内储存一个[序号，版本号出现次数]的list
'''
def get_version_list(string):
    # e.g.  v1  V1.1  2.3  py3.6  1.2.3-alpha1  3.1rc  v2-beta3
    result = {}
    for item in re.findall("(?<=\W)((([vV][0-9]+)|([a-zA-Z_]*[0-9]+\w*(\.[a-zA-Z_]*[0-9]\w*)))([\.-]\w+)*)(?=\W)",string):
        key = item[0].strip()
        if key not in result:
            result[key] = [len(result), 0]  # order, term-freq
        result[key][1] += 1
    return result
```

```
'''
输入：字符串string
输出：一个result字典：字典的每个key是标识符，其内储存一个[序号，标识符出现次数]的list
'''
def get_identifier_list(string):
    # e.g.  smallCamelCase  BigCamelCase  _underline_name_  test_123  func123Test
    result = {}
    for item in re.findall("(?<=\W)(([A-Z]*[a-z_][a-z0-9_]*)([A-Z_][a-z0-9_]*)+)(?=\W)", string):
        key = item[0].strip()
        if key not in result:
            result[key] = [len(result), 0]  # order, term-freq
        result[key][1] += 1
    return result
```

2. 在每个版本号和标识符前后分别加上tag

   如图，利用正则表达式在body中搜索版本号和标识符，并在前后加上tag

```
# 4.2在每个版本号和标识符前后分别加上tag
# 4.2.1在每个版本号前后非别加上"verid40" 和 "verid0"
version_list = issue["_spctok"]["ver"]
for version, stat in sorted(version_list.items(), key=lambda x: (len(x[0]))):
    issue['body'] = re.sub(re.escape(version), " verid40 " + version + " verid0 ", issue['body'],flags=re.IGNORECASE)
    issue['title'] = re.sub(re.escape(version), " " + version + " ", issue['title'], flags=re.IGNORECASE)
# 4.2.2在每个标识符前后非别加上"idtid40" 和 "idtid0"
identifier_list = issue["_spctok"]["idt"]
for idt, stat in sorted(identifier_list.items(), key=lambda x: (len(x[0]))):
    issue['body'] = re.sub(re.escape(idt), " idtid40 " + idt + " idtid0 ", issue['body'], flags=re.IGNORECASE)
    issue['title'] = re.sub(re.escape(idt), " " + idt + " ", issue['title'], flags=re.IGNORECASE)
```

3. 分类储存

```
    if idx < sep1:
        valid_issues_train.append(issue)
    elif idx < sep2:
        valid_issues_val.append(issue)
    else:
        valid_issues_test.append(issue)
    valid_issues.append(issue)
print("refinement and tagging finish. obtain", len(valid_issues),"good issues for txt.")

#储存
save=[valid_issues_train, valid_issues_val, valid_issues_test]
with open("data/good_issues.json", "w") as f:
    json.dump(save, f)
print("refining and tagging success. good sample set is saved to data/" + str(
    len(valid_issues_train) + len(valid_issues_val) + len(
        valid_issues_test)) + "good_issues.json'")
```

3.2.4生成TXT文件

0-2-export_txt_data.py将JSON数据文件转换为TXT文件（"**body.train.txt**" "**title.train.txt**" "**body.valid.txt**" "**title.valid.txt**" "**body.test.txt**" "**title.test.txt**"），供之后在OpenNMT的汇总模型中进行处理。

```
def main():
    #导出数据到txt文件中
    with open("332159good_issues.json") as f:
        valid_issues_train, valid_issues_val, valid_issues_test = json.load(f)
    with open("body.train.txt", "w",encoding='utf-8') as fbody, open("title.train.txt", "w",encoding='utf-8') as ftitle:
        bodies = [x['body'] + "\n" for x in valid_issues_train]
        titles = [x['title'] + "\n" for x in valid_issues_train]
        fbody.writelines(bodies)
        ftitle.writelines(titles)
    with open("body.valid.txt", "w",encoding='utf-8') as fbody, open("title.valid.txt", "w",encoding='utf-8') as ftitle:
        bodies = [x['body'] + "\n" for x in valid_issues_val]
        titles = [x['title'] + "\n" for x in valid_issues_val]
        fbody.writelines(bodies)
        ftitle.writelines(titles)
    with open("body.test.txt", "w",encoding='utf-8') as fbody, open("title.test.txt", "w",encoding='utf-8') as ftitle:
        bodies = [x['body'] + "\n" for x in valid_issues_test]
        titles = [x['title'] + "\n" for x in valid_issues_test]
        fbody.writelines(bodies)
        ftitle.writelines(titles)
```

3.2.5一般总结模块

1. 首先，为了训练模型，**1-1-build.sh**预处理了数据，构建 source-target 对齐的数据集，并提取训练特征为模型生成词表文件，结合词表文件和一个训练好的embedding文件，得到本程序语料库下的embedding文件

   输入：之前得到的body.train.txt, title.train,txt, body.valid.txt, title.valid.txt

   输出：data.train.0.pt, data.train.1.pt, data.train.2.pt, data.valid.0.pt以及词表文件data.vocab.pt

   参数说明：

   - --`dynamic_dict`：使用了copy-attention时，需要预处理数据集，以使source和target对齐。
   - --`share_vocab`：使source和target使用相同的字典。

   ```
   # build aligned dataset and vocab for training
   onmt_preprocess -train_src data/body.train.txt \
                   -train_tgt data/title.train.txt \
                   -valid_src data/body.valid.txt \
                   -valid_tgt data/title.valid.txt \
                   -save_data data/data \
                   -src_seq_length 10000 \
                   -tgt_seq_length 10000 \
                   -src_seq_length_trunc 400 \
                   -tgt_seq_length_trunc 100 \
                   -dynamic_dict \
                   -share_vocab \
                   -shard_size 100000 \
                   -num_threads 1
   ```

2. 然后，**1-2-embedding.sh**从Glove得到了用于分配初始嵌入权重的文件

   输入：glove.6B.100d.txt（一个在大规模语料库上预先训练过的GloVe向量）、data.vocab.pt

输出：本程序语料库的嵌入文件embeddings.dec.pt, embeddings.enc.pt

```
#following step requires the source codes of OpenNMT. this step will generate initial embedding weights based on glove vecto
OpenNMT-py/tools/embeddings_to_torch.py -emb_file_both "data/glove.6B.100d.txt" -dict_file "data/data.vocab.pt" -output_file
exec /bin/bash
```

3. `2-train.sh`中提供了训练命令和参数配置， 运行此脚本将训练一个基于 LSTM 的单层 Seq2Seq 摘要模型，该模型具有通过使用 `-copy_attn` 选项激活的复制机制。

```
onmt_train -save_model models/iTAPE \
           -data data/data \
           -copy_attn \
           -reuse_copy_attn \
           -copy_loss_by_seqlength \
           -global_attention mlp \
           -word_vec_size 100 \
           -rnn_size 512 \
           -layers 1 \
           -encoder_type brnn \
           -train_steps 25000 \
           -max_grad_norm 2 \
           -dropout 0.0 \
           -batch_size 32 \
           -valid_batch_size 32 \
           -valid_steps 5000 \
           -report_every 100 \
           -optim adam \
           -learning_rate 2 \
           -warmup_steps 8000 \
           -decay_method noam \
           -bridge \
           -seed 789 \
           -world_size 1 \
           -gpu_ranks 0 \
           -pre_word_vecs_enc "data/embeddings.enc.pt" \
           -pre_word_vecs_dec "data/embeddings.dec.pt" \
           -save_checkpoint_steps 5000 \
           -share_embeddings
```

输入：预处理好的scr-tgt数据集、初始嵌入

输出：模型iTAPE

参数说明：

- `-copy_attn`、`-reuse_copy_attn`和`-copy_loss_by_seqlength`用于激活复制机制。
  - `-copy_attn`允许模型从源中复制单词（2.3.2节提到的复制机制）
  - `-reuse_copy_attn`: 将standard attention重用为copy attention。没有这个，模型会学习一个额外的注意力，仅用于复制。

        - `-copy_loss_by_seqlength`：将损失除以序列长度。实践中我们发现这可以使inference时生成长序列。然而，这种效果也可以通过在解码期间使用惩罚来实现。
    - `-global_attention mlp`：使用 Bahdanau 等的注意力机制替代了 Luong 等人的机制（global_attention dot）。
    - `-bridge`：这是一个附加层，它使用编码器的最终隐藏状态作为输入并计算解码器的初始隐藏状态。没有这个，解码器直接用编码器的最终隐藏状态初始化。
    - `-pre_word_vecs_enc` 和 `-pre_word_vecs_dec` 用于分配初始嵌入权重（例如从预训练的 GloVe 中获得）。
    - `-optim`、`-learning_rate`、`-warmup_steps` 和`-decay_method` 用于指定有效且稳定的优化器。
    - `-share_embeddings`：使编码器和解码器共享词嵌入。减少了模型必须学习的参数数量。
4. 为了在测试集中为问题主体生成标题，提供了 `3-test.sh` 来应用这一步的训练模型。 运行此脚本后，生成的标题将被有序保存到 testout 目录下的iTAPE_step_25000_minlen8.txt文件中。

```
nohup onmt_translate -gpu 0  \
                -batch_size 32 \
                -beam_size 10 \
                -model models/iTAPE_step_25000.pt \
                -src data/body.test.txt \
                -output testout/iTAPE_step_25000_minlen8.txt \
                -min_length 8 \
                -stepwise_penalty \
                -length_penalty wu \
                -alpha 0.9 \
                -replace_unk \
                -block_ngram_repeat 2
```

输入：训练完成的iTAPE模型、用于测试部分的body的txt文件

输出：储存模型预测出的标题的txt文件

参数说明：

`-beam_size`：用于激活波束搜索

`-min_length`：用于设置最小输出阈值（toks）

`-stepwise_penalty`: 在每一step运用惩罚

`-length_penalty wu`：使用Wu的长度惩罚项

`-alpha 0.9`: 长度惩罚项的参数.

`-replace_unk`：防止生成无效的

`-block_ngram_repeat 2`：为了避免在我们的输出目标中进行无用的重复

4.工具验证

使用Rouge(Recall-Oriented Understudy for Gisting Evaluation)来评估模型的预测效果。Rouge是评估自动文摘的一组指标。它通过将自动生成的摘要与一组参考摘要进行比较计算，得出相应的分值，以衡量自动生成的摘要与参考摘要之间的“相似度”。

从TXT文件中获取title列表，a1为模型预测出的title列表，b1为原本测试集真实的title列表，获得rouge_score存储每组对应title的rouge-1, rouge-2, rouge-l分数，计算得出整个模型的平均rouge-1 F1, rouge-2 F1, rouge-l F1分数：

```
from rouge import Rouge
#获得预测的title列表a1
f=open(r'testout/iTAPE_step_25000_minlen8.txt','r',encoding='utf-8')
a=list(f)
a1=[]
for per in a:
    per=per.strip('\n')#去掉读取txt文件获得的换行符
    a1.append(per)
f.close()

#获得真实的title列表b1
f=open(r'title.train.txt','r',encoding='utf-8')
b=list(f)
b1=[]
for per in b:
    per=per.strip('\n')#去掉读取txt文件获得的换行符
    b1.append(per)
f.close()

#获得测试集的平均Rouge-1, Rouge-2, Rouge-L的F1评分
rouge = Rouge()
rouge_score = rouge.get_scores(a1, b1)
average_1=0
average_2=0
average_l=0
for item in rouge_score:#rouge_score的结构为：[{xxx},{xxx},...{'rouge-1'
    average_1+=item["rouge-1"]['f']
    average_2 += item["rouge-2"]['f']
    average_l += item["rouge-l"]['f']
average_1=average_1/len(rouge_score)
average_2=average_2/len(rouge_score)
average_l=average_l/len(rouge_score)
print(average_1)
print(average_2)
print(average_l)
```

5.Reference

Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, Baowen Xu. Stay Professional and Efficient: Automatically Generate Titles for Your Bug Reports. IEEE/ACM International Conference on Automated Software Engineering(ASE)., 2020, 385-397.