详细设计算法部分

# 文章数据预处理模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: “文章内容”} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <文章内容> | str |
| 解释 | 每次处理当前新获取到的所有文章，文章以{id: 内容}的键值对组织表示为dict数据类型。文章内容的每个自然段之间用’\n’分隔 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: [“原文词语1”, “原文词语2”, …], “pos”:[“词类1”, “词类2”, …], “dp”: [“依存关系1”, “依存关系2, …”], “head”:[“依存中心1”, “依存中心2”]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文词语> | str |
| <词类> | str enumeration |
| <依存关系> | str enumeration |
| <依存中心> | int |
| 解释 | 每个文章包含若干个句子，句子按照其出现的顺序排列。每个句子有若干个词语组成，按照出现顺序排列。每个词语对应一个词类，一个依存关系，以及一个依存中心。依存关系即为词语与其依存中心所指向的词语之间的语义依存关系 | |

## 实现逻辑

将文章原文以 ‘\n’ 分隔为自然段，对每个自然段使用LTP工具(pip install LTP)进行分句，分词，词类分析，依存分析。将LTP返回的依存分析arc分别去其依存关系，依存中心。最后将所有数据合并返回。

# 实体识别+链接模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: [“原文词语1”, “原文词语2”, …], “pos”:[“词类1”, “词类2”, …], “dp”: [“依存关系1”, “依存关系2, …”], “head”:[“依存中心1”, “依存中心2”]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文词语> | str |
| <词类> | str enumeration |
| <依存关系> | str enumeration |
| <依存中心> | int |
| 解释 | 实体识别模块每次处理当前需要进行实体识别的所有文章，文章以{id: 内容}的键值对组织表示为dict数据类型，该输入格式与文章预处理模块的输出相同。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: “原文句子”, “entities”: [{“form”: “实体在文中的表述”, “name”: “实体在知识库中的名称”, “type”: “实体类型”, “wikiID”: “实体在知识库中的ID”, “idx”: [实体开始位置, 实体结束位置]}]}]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文句子> | str |
| <实体在文中的表述> | str |
| <实体在知识库中的名称> | str |
| <实体类型> | str enumeration |
| <实体在知识库中的ID > | str |
| <实体开始/结束位置> | int |
| 解释 | 每个文章包含若干个句子，句子按照其出现的顺序排列。每个句子有若干个实体，同名同指实体若在句子中出现多次则也会在“entities”字段占有多个位置，“entities”字段以实体的结束位置进行排序，若实体字段为空则说明本句无实体。实体类型只可能是PER,LOC,ORG,REG,OTH之一。实体在知识库的ID以Q开头，若ID为空字符串则说明该实体在知识库中找不到对应项目。实体的开始/结束位置使得后续算法可以通过 原文句子[开始位置:结束位置] 得到该实体在原文的表述。 | |

## 实现逻辑

### 输入读取

定义专门的类继承torch.Dataset类进行数据读取。得到输入的文章集合后，将每篇文章的每个句子提取出来，合并原句的分词，组成总列表，等待算法进行批处理。输入的其余字段舍去。

模型预测阶段，将句子总列表中的每个句子进行长度统一，长度不足128的句子用[PAD]字符补足到128，长度超过128的句子截断超出部分。再将所有字符进行数字化编号，转化为整形Tensor，输入到实体识别算法模型中。

### 实体识别算法模型

待定

### 实体链接算法模型

待定

### 输出格式转化

待定

## 实现逻辑

### 输入读取

定义专门的类继承torch.Dataset类进行数据读取。得到输入的文章集合后，将每篇文章的每个句子提取出来，合并原句的分词，组成总列表，等待算法进行批处理。输入的其余字段舍去。

模型预测阶段，将句子总列表中的每个句子进行长度统一，长度不足128的句子用[PAD]字符补足到128，长度超过128的句子截断超出部分。再将所有字符进行数字化编号，转化为整形Tensor，输入到实体识别算法模型中。

### 实体识别算法模型

待定

### 输出格式转化

待定

# 实体识别+链接模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: “原文句子”, “entities”: [{“form”: “实体在文中的表述”, “type”: “实体类型”, “idx”: [实体开始位置, 实体结束位置]}]}]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文句子> | str |
| <实体在文中的表述> | str |
| <实体类型> | str enumeration |
| <实体开始/结束位置> | int |
| 解释 | 每个文章包含若干个句子，句子按照其出现的顺序排列。每个句子有若干个实体，同名同指实体若在句子中出现多次则也会在“entities”字段占有多个位置，“entities”字段以实体的结束位置进行排序，若实体字段为空则说明本句无实体。实体类型只可能是PER,LOC,ORG,REG,OTH之一。实体的开始/结束位置使得后续算法可以通过 原文句子[开始位置:结束位置] 得到该实体在原文的表述。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: “原文句子”, “entities”: [{“form”: “实体在文中的表述”, “name”: “实体在知识库中的名称”, “type”: “实体类型”, “wikiID”: “实体在知识库中的ID”, “idx”: [实体开始位置, 实体结束位置]}]}]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文句子> | str |
| <实体在文中的表述> | str |
| <实体在知识库中的名称> | str |
| <实体类型> | str enumeration |
| <实体在知识库中的ID > | str |
| <实体开始/结束位置> | int |
| 解释 | 每个文章包含若干个句子，句子按照其出现的顺序排列。每个句子有若干个实体，同名同指实体若在句子中出现多次则也会在“entities”字段占有多个位置，“entities”字段以实体的结束位置进行排序，若实体字段为空则说明本句无实体。实体类型只可能是PER,LOC,ORG,REG,OTH之一。实体在知识库的ID以Q开头，若ID为空字符串则说明该实体在知识库中找不到对应项目。实体的开始/结束位置使得后续算法可以通过 原文句子[开始位置:结束位置] 得到该实体在原文的表述。 | |

## 实现逻辑

### 输入读取

定义专门的类继承torch.Dataset类进行数据读取。得到输入的实体识别结果后，将整篇文章的每个句子及其包含的实体表述提取出来，组成总列表，等待算法进行批处理。输入的其余字段舍去。

模型预测阶段，将句子总列表中的每个句子进行长度统一，长度不足128的句子用[PAD]字符补足到128，长度超过128的句子截断超出部分。再将所有字符进行数字化编号，转化为整形Tensor，输入到实体识别算法模型中。

### 实体链接算法模型

待定

### 输出格式转化

待定

# 开放式关系抽取模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: “原文句子”, “entities”: [{“form”: “实体在文中的表述”, “name”: “实体在知识库中的名称”, “type”: “实体类型”, “wikiID”: “实体在知识库中的ID”, “idx”: [实体开始位置, 实体结束位置]}]}]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文句子> | str |
| <实体在文中的表述> | str |
| <实体在知识库中的名称> | str |
| <实体类型> | str enumeration |
| <实体在知识库中的ID > | str |
| <实体开始/结束位置> | int |
| 解释 | 每个文章包含若干个句子，句子按照其出现的顺序排列。每个句子有若干个实体，同名同指实体若在句子中出现多次则也会在“entities”字段占有多个位置，“entities”字段以实体的结束位置进行排序，若实体字段为空则说明本句无实体。实体类型只可能是PER,LOC,ORG,REG,OTH之一。实体在知识库的ID以Q开头，若ID为空字符串则说明该实体在知识库中找不到对应项目。实体的开始/结束位置使得后续算法可以通过 原文句子[开始位置:结束位置] 得到该实体在原文的表述。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章ID”: [{“sentence”: “原文句子”, “triples”: [  {“entity1”: {“form”: “实体在文中的表述”, “name”: “实体在知识库中的名称”, “type”: “实体类型”, “wikiID”: “实体在知识库中的ID”, “idx”: [实体开始位置, 实体结束位置]},  “entity2”: {“form”: “实体在文中的表述”, “name”: “实体在知识库中的名称”, “type”: “实体类型”, “wikiID”: “实体在知识库中的ID”, “idx”: [实体开始位置, 实体结束位置]},  “relation”: 关系词在文中的表述,  “triple\_id”: 实体关系三元组在知识库的ID}]}]} | |
| 数据类型 | 整体 | dict |
| <文章ID> | str |
| <原文句子> | str |
| <实体在文中的表述> | str |
| <实体在知识库中的名称> | str |
| <实体类型> | str enumeration |
| <实体在知识库中的ID > | str |
| <实体开始/结束位置> | int |
|  | <关系词在文中的表述> | str |
| <实体关系三元组在知识库的ID > | str |
| 解释 | 每个句子中可能存在多个实体，通过组合两个实体作为实体对，对实体对进行开放式关系抽取。对于一个句子，可能有多个实体对，则对应有多个开放式关系。 | |

## 实现逻辑

### 输入读取

### 开放式关系抽取算法模型

### 输出格式转化

待定

# 相关事件模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“文章路径” } | |
| 数据类型 | “文章路径” | str |
| 解释 | 相关事件模块通过文件路径读取文件夹下的全部文章，将文章内容按照聚类算法聚类出包含的事件。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“事件ID”：[ “文章ID”]} } | |
| 数据类型 | 整体 | dict |
| <事件ID> | str |
| <文章ID> | str |
| 解释 | 每个事件由多篇相关文章组成，文章按照其报道的时间排序，返回该事件包含的所有文章ID。 | |

## 实现逻辑

### 输入读取

读取给定的文件夹下所有的文章内容。包括文章报道的时间和文章的具体内容。

模型预测阶段：将待聚类的所有文章分词，并获得利用TFIDF获得文本数据的空间向量表示。

### 事件算法模型

在该系统中文章数据较多，因此采用Single-Pass聚类方法，该算法高效、简单、快捷。对于大规模数据只需要遍历一遍就可以完成聚类，并且Single-pass聚类算法不需要指定类目数量，可以通过设定相似度阈值来限定聚类数量。

算法流程：

（1） 接收文章的文本向量d；

（2） d逐一与已有的事件中各报道进行相似度计算，并取最大者作为与该时间的相似度（single-link策略）；

（3） 在所有话题间选出与d相似度最大的一个，以及此时的相似度值；

（4） 如果相似度大于阈值TC，d所对应的互联网文本被分配给这个事件模型文本类，跳至（6）；

（5） 如果相似度值小于阈值TC，d所对应的文本不属于已有的事件，创建新事件，同时把这篇文本归属创建的新事件模型文本类；

（6） 本次聚类结束，等待文本到来。

注：阈值在[0,1]之间，阈值设置的越高，得到的簇粒度越小（簇内文本数量少），簇的个数越多；相反，阈值设置的越低，得到的簇粒度越大（簇内文本数量多），簇的个数越少。

### 输出格式转化

待定

1. **文本摘要模块**

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“事件ID”: 事件包含的文章ID} | |
| 数据类型 | 整体 | dict |
| <事件ID> | str |
| <时间包含的文章ID> | list |
| 解释 | 文本摘要模块每次需要处理当前事件包含的所有文章ID，文章以{id: 内容}的键值对组织表示为dict数据类型。文章按照报道的时间顺序排列。文章内容的每个自然段之间用’\n’分隔。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“事件ID”：{“摘要”：“抽取出的事件摘要”，“文章”：[ “文章ID”]} } | |
| 数据类型 | 整体 | dict |
| <事件ID> | str |
| <抽取出的事件摘要> | str |
| <文章ID> | str |
| <文章摘要> | str |
| 解释 | 每个事件由多篇相关文章组成，文章按照其报道的时间排序，“摘要”根据多篇文章组合内容抽取出事件的摘要。同时保留源文章信息。 | |

## 实现逻辑

### 输入读取

首先按照事件的ID获取有序的相关文章列表，对获取列表中包含的文章的内容，将每一篇文章的第一段内容清洗后作为文章的信息（由新闻的写作特点决定）组成事件内容。同时对事件内容进行分句处理。将所有待处理的事件按上述步骤处理。等待算法进行批处理。

模型预测阶段：将每个事件内容控制在512个字符，用ALBERT的分词工具进行分词和数字化编码，转化为模型输入形式。

### 文本摘要算法模型

事件内容是包含多个句子的文章，针对ALBERT预训练时输入数据的特点，在每个句子开始部分添加［CLS］，表示句子的信息，结束部分添加［SEP］作为句子结束标记。同时为了充分应用散布在整个ALBERT层的语言信息，将每一层的句子嵌入表示，输入Bｉ－LSTM得到模型对当前层的语言信息理解，然后获得每一层的信息所占权重，最终每一层的句子嵌入加权作为ALBERT模型输出的补充表示，然后输入全连接层进行句子的预测。

### 输出格式转化

待定

# 人格预测预处理模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“用户ID” : [“帖子内容”]} | |
| 数据类型 | 整体 | dict |
| <用户ID> | str |
| <帖子内容> | str |
| 解释 | 人格预测模块每次处理当前需要进行人格分析的用户的所有帖子内容，数据保存为dict格式，每一条数据有用户id，以及该用户所有帖子内容。每个用户有若干篇帖子，帖子之间以‘<sep>’作为分隔。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“用户ID” : [“规范化的帖子内容”]} | |
| 数据类型 | 整体 | 数据类型 |
| <用户ID> | str |
| <规范化的帖子内容> | str |
| 解释 | 每个用户的原始帖子文本存在许多特殊符号、表情等无关内容，需要针对每个用户的<帖子内容>进行清洗，去除噪声文本。重新保存为dict格式，每个用户的帖子之间仍然以‘<sep>’作为分隔。 | |

## 实现逻辑

首先针对每个用户的帖子文本内容进行数据清洗，对于中文数据使用正则表达式‘[^\u4e00-\u9fa5]’筛除非中文字符，然后使用结巴分词对文本进行切分，词之间用空格分隔；对于英文数据，首先使用正则表达式去除特殊符号，然后使用NLTK python库过滤每个帖子，提取动词词干，最后仍然使用<sep>对帖子进行切分。

# 人格预测模块

## 输入格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“用户ID” : [“规范化的帖子内容”]} | |
| 数据类型 | 整体 | 数据类型 |
| <用户ID> | str |
| <规范化的帖子内容> | str |
| 解释 | 人格预测模块每次处理当前需要人格预测的所有用户，输入数据格式与人格预测预处理模块的输出相同。 | |

## 输出格式

|  |  |  |
| --- | --- | --- |
| 数据格式 | {“用户ID”: {“ope”:开放性分值, “con”:尽责性分值, “ext”:外向性分值, “agr”:宜人性分值, “neu”:神经质分值}} | |
| 数据类型 | 整体 | dict |
| <用户ID> | str |
| <开放性分值> | float |
| <尽责性分值> | float |
| <外向性分值> | float |
| <宜人性分值> | float |
| <神经质分值 > | float |
| 解释 | 针对用户的所有输入帖子内容，输出用户五个维度的对应分值。 | |

## 实现逻辑

### 输入读取

在规范化后的文本输入之前，首先对其进行相关特征的抽取工作，根据构建好的繁体LIWC心理词典文件‘tw\_liwc\_dict.json’，每类特征（除字符总数外）对应一个词类别，对清洗后的文本统计该文本在每类词下面的词频情况，若该用户文本中某类词频情况高于平均水平，则该用户在这个词类下的特征值为1，反之则为-1。针对每个用户的所有帖子具体需要统计以下特征：

|  |  |
| --- | --- |
| 特征名 | 数据类型 |
| <字符总数> | int |
| <代词> | int |
| <第一人称单数> | int |
| <第一人称复数> | int |
| <第二人称> | int |
| <第三人称单数> | int |
| <第三人称复数> | int |
| <介词> | int |
| <助动词> | int |
| <副词> | int |
| <连词> | int |
| <否定词> | int |
| <数词> | int |
| <积极情绪词> | int |
| <消极情绪词> | int |
| <社交类词> | int |
| <认知过程> | int |
| <洞察力> | int |
| <因果性> | int |
| <差异动词> | int |
| <猜想性> | int |
| <可能性> | int |
| <转折> | int |
| <归属性> | int |
| <成就> | int |
| <权力> | int |
| <过去时态> | int |
| <现在时态> | int |
| <将来时态> | int |
| <动作短语> | int |
| <空间> | int |
| <时间> | int |
| <大词> | int |

针对输入数据中每个用户的各个特征值（+1或-1），将用户作为节点加入到构建好的词—语言风格（word—style，W—S）图中，若一个用户U的某个词类特征W的特征值为+1，则构建U—W+的关联关系，反之构建U—W-的关联关系，从而得到U—W—S的层级图，将所有节点数字化编号，进而获取用户节点与词类节点的邻接矩阵Adj\_UW, 以及词类节点与风格节点的邻接矩阵Adj\_WS和风格节点之间的邻接矩阵Adj\_SS,

得到用户与词类节点的邻接矩阵Adj\_UW后，定义专门的类继承torch.Dataset类进行数据读取，并转化为Tensor类便于模型处理，接着输入到人格预测算法模型中。

### 人格预测算法模型

待定

### 输出格式转化

待定