英文文档评测数据预处理

readme

在英文文本中，输入和输出的句子是不能直接拿来计算EM(exact match)或者说是F1score的，往往需要一定的处理，本篇文档则记录了这种文本怎么处理，学习本篇文档前需要先学习正则表达式。

相关python库的导入

```python
from collections import defaultdict
from typing import Any, Dict, List, Set, Tuple, Union, Optional
import json
import argparse
import string
import re
```

- `collections.defaultdict`：`defaultdict`是 Python 标准库中的一部分，它是 `dict` 的一个子类，提供了一个默认值，这样在查找不存在的键时不会抛出 KeyError。如果尝试访问一个不存在的键，`defaultdict` 会为你返回一个默认值，而不是抛出异常。比如，`defaultdict(int)` 会返回 0，`defaultdict(list)` 会返回一个空列表。这在处理数据时非常有用。
- `typing`：`typing` 模块是 Python 3.5 引入的标准库，用于支持类型提示。它包含了一系列用于类型注解的类和函数，比如 `Any`，`Dict`，`List`，`Set`，`Tuple`，`Union`，`Optional` 等，这些可以帮助开发者在代码中明确指定变量、函数参数和返回值的类型，提高代码的可读性和可维护性。
  - `Any`：任何形式的数据
  - `Dict`：字典
  - `List`：列表
  - `Set`：集合
  - `Tuple`：元组
  - `Union`：`Union` 用于指定一个变量可以是多种类型中的一种。例如，`Union[int, float]` 表示一个变量可以是整数或浮点数。

    - `Optional` 用于指定一个变量可以是某种特定类型，或者是 `None`。例如，`Optional[str]` 表示一个变量可以是字符串类型，也可以是 `None`。
- `json`：`json` 是 Python 的一个内置模块，用于处理 JSON 格式的数据。它提供了将 Python 数据结构转换为 JSON 格式（序列化）以及将 JSON 格式转换为 Python 数据结构（反序列化）的功能。
- `argparse`：`argparse` 是 Python 标准库中用于解析命令行参数的模块。它可以帮助开发者编写易于使用的命令行接口，包括定义命令行选项、参数和子命令，以及生成帮助信息。
- `string`：`string` 模块包含了许多有用的字符串相关的常量和函数，比如标点符号、数字和字母等。这些可以帮助开发者进行字符串操作时更加方便和高效。
- `re`：`re` 是 Python 的正则表达式模块，提供了对正则表达式的支持，包括模式匹配、替换、分割等功能。通过 `re` 模块，开发者可以使用正则表达式来处理文本数据，进行复杂的模式匹配和替换操作。

数据预处理

- 学习完后再使用可以直接复制一键运行版

大写字母转小写

```python
def _lower(text: str) -> str:
    return text.lower()
```

去掉冠词

```python
def _remove_articles(text: str) -> str:
    regex = re.compile(r"\b(a|an|the)\b", re.UNICODE)
    return re.sub(regex, " ", text)

_remove_articles('I have an apple')
```

- output: 'I have apple'

去除多余的空格

```python
def _white_space_fix(text: str) -> str:
    return " ".join(text.split())

_white_space_fix("""你好
今天星期几?
""")
```

- output: '你好 今天星期几? '
- 可以把多余的空格比如：连着两个的空格、换行符、制表符消除

判断一个字符串是不是数字

```python
def _is_number(text: str) -> bool:
    try:
        float(text)
        return True
    except ValueError:
        return False

_is_number('5')
# output: True

_is_number('a')
# output: False
```

对数字型文本统一格式

```python
def _normalize_number(text: str) -> str:
    if _is_number(text):
        return str(float(text))
    else:
        return text
_normalize_number('5.000')
# output: '5.0'
```

- 作用是消除多余小数点后面的 `0`

去除标点符号

```
EXCLUDE = set(string.punctuation)
EXCLUDE
#标点符号有 !"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~

def _remove_punc(text: str) -> str:
    if not _is_number(text):
        return "".join(ch for ch in text if ch not in EXCLUDE)
    else:
        return text
_remove_punc('hello, world')
```

- output: hello world

分词(粒度为单词)

```
def _tokenize(text: str) -> List[str]:
    return re.split(" |-", text)
_tokenize('hello world-how')
```

- output: ['hello', 'world', 'how']

合并上面的六种处理

```
def _normalize_answer(text: str) -> str:
    """Lower text and remove punctuation, articles and extra whitespace."""

    parts = [_white_space_fix(_remove_articles(_normalize_number(_remove_punc(_lower(token))))) for token in _tokenize(text)]
    # 下面代码的作用是将列表 `parts` 中的非空字符串元素连接成一个字符串，并去除首尾的空白字符后返回。
    parts = [part for part in parts if part.strip()]
    normalized = " ".join(parts).strip()
    return normalized
```

```
9
10 _normalize_answer('''Hello World,
11                     how| are
12                     you
13
14                     ''')
```

- Output : hello world how are you

一键运行版

- 别忘记了把要洗的语句换成自己的

```python
def _lower(text: str) -> str:
    return text.lower()

def _remove_articles(text: str) -> str:
    regex = re.compile(r"\b(a|an|the)\b", re.UNICODE)
    return re.sub(regex, " ", text)

def _white_space_fix(text: str) -> str:
    return " ".join(text.split())

def _is_number(text: str) -> bool:
    try:
        float(text)
        return True
    except ValueError:
        return False

def _normalize_number(text: str) -> str:
    if _is_number(text):
        return str(float(text))
    else:
        return text

EXCLUDE = set(string.punctuation)
def _remove_punc(text: str) -> str:
    if not _is_number(text):
        return "".join(ch for ch in text if ch not in EXCLUDE)
    else:
        return text

def _tokenize(text: str) -> list[str]:
```

```
    return re.split(" |-", text)

def _normalize_answer(text: str) -> str:
    """Lower text and remove punctuation, articles and extra whitespace."""

    parts = [_white_space_fix(_remove_articles(_normalize_number(_remove_punc(_lower(token))))) for token in _tokenize(text)]
    # 下面代码的作用是将列表 `parts` 中的非空字符串元素连接成一个字符串，并去除首尾的空白字符后返回。
    parts = [part for part in parts if part.strip()]
    normalized = " ".join(parts).strip()
    return normalized

_normalize_answer('''Hello World,
                  how|are
                  you

                  ''')
```

数据集处理

数据集统一进行处理，提取单词集合

```
def _answer_to_bags(answer: Union[str, List[str], Tuple[str, ...]]) -> Tuple[List[str], List[Set[str]]]:
    # 判断是否是list或tuple，不是的话将str数据加上[]变成list
    # raw_spans为原先未处理的数据，可以是包含很多个句子的列表
    if isinstance(answer, (list, tuple)):
        raw_spans = answer
    else:
        raw_spans = [answer]

    # normalized_spans用于储存处理过后的数据
    normalized_spans: List[str] = []

    # token_bags用于储存一句话里面所有单词的集合
    token_bags = []
    for raw_span in raw_spans:
        normalized_span = _normalize_answer(raw_span)
        normalized_spans.append(normalized_span)
        token_bags.append(set(normalized_span.split()))
```

```
    return normalized_spans, token_bags
normalized_spans, token_bags = _answer_to_bags(['Hello, how are you','I am fine, Thank you, and you?'])

print(normalized_spans)
print(token_bags)
```

- output:
- normalized_spans:
- ['hello how are you', 'i am fine thank you and you']
- token_bags:
- [{'hello', 'are', 'how', 'you'}, {'fine', 'i', 'thank', 'and', 'am', 'you'}]

计算精确率、召回率、F1score

- **nlp**中的精确率、召回率、F1Score计算公式
- 预测语句和真实答案的单词集合交集为 intersection
- 预测语句的单词集合长度为len_predict
- 真实语句的单词集合长度为len_truth
- 精确率：$precision = intersection / len\_predict$
- 召回率：$recall = intersection / len\_truth$
- $F1score = \dfrac{2 * precision * recall}{precision + recall}$

```
# predicted_bag: 预测语句的单词集合set
# gold_bag: 答案语句的单词集合set
def _compute_f1(predicted_bag: Set[str], gold_bag: Set[str]) -> Tuple[float, float, float]:

    # intersection: 两个集合set之间的交集
    intersection = len(gold_bag.intersection(predicted_bag))
    if not predicted_bag:
        precision = 1.0
    else:
        precision = intersection / float(len(predicted_bag))
    if not gold_bag:
        recall = 1.0
    else:
        recall = intersection / float(len(gold_bag))
```

```
    f1 = (2 * precision * recall) / (precision + recall) if not (precision == 0.0 and recall == 0.0) else 0.0
    return (f1, precision, recall)

_compute_f1(['今天','星期五','是'],set(['今天','星期六','不是','是']))
```

- output: (0.5714285714285715, 0.6666666666666666, 0.5)
- 分别对应 *F1 score*，精确率和召回率