文本预处理

2020年2月14日　0:23

文本是一类序列数据，一篇文章可以看作是字符或单词的序列，本节将介绍文本数据的常见预处理步骤，预处理通常包括四个步骤：

1. 读入文本
2. 分词
3. 建立字典，将每个词映射到一个唯一的索引（index）
4. 将文本从词的序列转换为索引的序列，方便输入模型

读入文本

我们用一部英文小说，即H. G. Well的Time Machine，作为示例，展示文本预处理的具体过程。

In [1]:

```
import collections
import re
def read_time_machine():
    with open('/home/kesci/input/timemachine7163/timemachine.txt', 'r') as f:
        lines = [re.sub('[^a-z]+', ' ', line.strip().lower()) for line in f]
    return lines
lines = read_time_machine()
print('# sentences %d' % len(lines))
# sentences 3221
```

分词

我们对每个句子进行分词，也就是将一个句子划分成若干个词（token），转换为一个词的序列。

In [2]:

```
def tokenize(sentences, token='word'):
    """Split sentences into word or char tokens"""
    if token == 'word':
        return [sentence.split(' ') for sentence in sentences]
    elif token == 'char':
        return [list(sentence) for sentence in sentences]
    else:
        print('ERROR: unkown token type '+token)
tokens = tokenize(lines)
tokens[0:2]
Out[2]:
[['the', 'time', 'machine', 'by', 'h', 'g', 'wells', ''], ['']]
```

建立字典

为了方便模型处理，我们需要将字符串转换为数字。因此我们需要先构建一个字典（vocabulary），将每个词映射到一个唯一的索引编号。

建立字典步骤：

1. 去重，统计词频；
2. 视情况增删词；
3. 增加特殊token；
4. 建立索引；

In [3]:

```
class Vocab(object):
    def __init__(self, tokens, min_freq=0, use_special_tokens=False):
        counter = count_corpus(tokens)  #:  函数count_corpus统计词频
```

```
class Vocab(object):
    def __init__(self, tokens, min_freq=0, use_special_tokens=False):
        counter = count_corpus(tokens)  #:  函数count_corpus统计词频
        self.token_freqs = list(counter.items())
        self.idx_to_token = []
        if use_special_tokens:
            # padding, begin of sentence, end of sentence, unknown
            self.pad, self.bos, self.eos, self.unk = (0, 1, 2, 3)
            self.idx_to_token += ['', '', '', '']
        else:
            self.unk = 0
            self.idx_to_token += ['']
        self.idx_to_token += [token for token, freq in self.token_freqs
                              if freq >= min_freq and token not in self.idx_to_token]
        self.token_to_idx = dict()
        for idx, token in enumerate(self.idx_to_token):
            self.token_to_idx[token] = idx
    def __len__(self):
        return len(self.idx_to_token)
    def __getitem__(self, tokens):
        if not isinstance(tokens, (list, tuple)):
            return self.token_to_idx.get(tokens, self.unk)
        return [self.__getitem__(token) for token in tokens]
    def to_tokens(self, indices):
        if not isinstance(indices, (list, tuple)):
            return self.idx_to_token[indices]
        return [self.idx_to_token[index] for index in indices]
def count_corpus(sentences):
    tokens = [tk for st in sentences for tk in st]
    return collections.Counter(tokens)  # 返回一个字典，记录每个词的出现次数
```

3. 增加特殊token；
4. 建立索引；

Special token:

- pad -padding，在短句后根据长句补充pad，使整齐；
- bos -begin of sentence，表示句子开始；
- eos -end of sentence，表示句子结束；
- unk -unknown，表示语料库中未出现的词（未登录词）

我们看一个例子，这里我们尝试用Time Machine作为语料构建字典

In [4]:

```
vocab = Vocab(tokens)
print(list(vocab.token_to_idx.items())[0:10])
```

```
[('', 0), ('the', 1), ('time', 2), ('machine', 3), ('by', 4), ('h', 5), ('g', 6), ('wells', 7), ('i', 8), ('traveller', 9)]
```

将词转为索引

使用字典，我们可以将原文本中的句子从单词序列转换为索引序列

In [5]:

```
for i in range(8, 10):
    print('words:', tokens[i])
    print('indices:', vocab[tokens[i]])
```

```
words: ['the', 'time', 'traveller', 'for', 'so', 'it', 'will', 'be', 'convenient', 'to', 'speak', 'of', 'him', '']
indices: [1, 2, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0]
words: ['was', 'expounding', 'a', 'recondite', 'matter', 'to', 'us', 'his', 'grey', 'eyes', 'shone', 'and']
indices: [20, 21, 22, 23, 24, 16, 25, 26, 27, 28, 29, 30]
```

用现有工具进行分词

我们前面介绍的分词方式非常简单，它至少有以下几个缺点:

1. 标点符号通常可以提供语义信息，但是我们的方法直接将其丢弃了
2. 类似“shouldn't", "doesn't"这样的词会被错误地处理
3. 类似"Mr.", "Dr."这样的词会被错误地处理

   我们可以通过引入更复杂的规则来解决这些问题，但是事实上，有一些现有的工具可以很好地进行分词，我们在这里简单介绍其中的两个：spaCy和NLTK。

   下面是一个简单的例子：

   In [6]:

```
text = "Mr. Chen doesn't agree with my suggestion."
```

   spaCy:

   In [7]:

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(text)
print([token.text for token in doc])
```

```
['Mr.', 'Chen', 'does', "n't", 'agree', 'with', 'my', 'suggestion', '.']
```

   NLTK:

   In [8]:

```
from nltk.tokenize import word_tokenize
from nltk import data
data.path.append('/home/kesci/input/nltk_data3784/nltk_data')
print(word_tokenize(text))
```

```
['Mr.', 'Chen', 'does', "n't", 'agree', 'with', 'my', 'suggestion', '.']
```