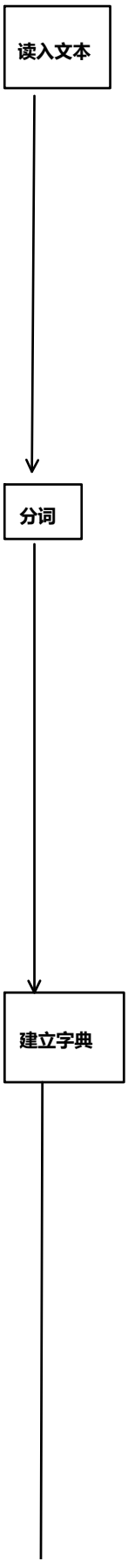
伯禹学习笔记篇贰

2020年2月14日 19:32

文本预处理

1. 预处理步骤




```
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
```

将一个句子划分成若干个词（token），转换为一个词的序列

```
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', ''], ['']]
```

将每个词映射到一个唯一的索引

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

字符转为索引

```
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'
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', '
indices: [20, 21, 22, 23, 24, 16, 25, 26, 27, 28, 29, 30]
```

2. 分词工具

主要是spaCy和NLTK

n元语言模型

1. 基础

假设序列$w_1, w_2, \ldots, w_T$中的每个词是依次生成的，我们有

$$\begin{aligned} P(w_1, w_2, \ldots, w_T) &= \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1}) \\ &= P(w_1)P(w_2 \mid w_1)\cdots P(w_T \mid w_1 w_2 \cdots w_{T-1}) \end{aligned}$$

例如，一段含有4个词的文本序列的概率

$$P(w_1, w_2, w_3, w_4) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)P(w_4 \mid w_1, w_2, w_3).$$

根据样本频率来估计概率：

$$\hat{P}(w_1) = \frac{n(w_1)}{n}$$

$$\hat{P}(w_2 \mid w_1) = \frac{n(w_1, w_2)}{n(w_1)}$$

2. 增加的假设--马尔可夫链

序列长度增加，计算和存储多个词共同出现的概率的复杂度会呈指数级增加。n元语法通过马尔可夫假设简化模型，马尔科夫假设是指一个词的出现只与前面n个词相关，即n阶马尔可夫链（Markov chain of order n），如果$n = 1$，那么有$P(w_3 \mid w_1, w_2) = P(w_3 \mid w_2)$。基于$n-1$阶马尔可夫链，我们可以将语言模型改写为

序列长度增加，计算和存储多个词共同出现的概率的复杂度会呈指数级增加。n元语法通过马尔可夫假设简化模型，马尔科夫假设是指一个词的出现只与前面n个词相关，即n阶马尔可夫链（Markov chain of order n），如果$n=1$，那么有$P(w_3 \mid w_1, w_2) = P(w_3 \mid w_2)$。基于$n-1$阶马尔可夫链，我们可以将语言模型改写为

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1}).$$

以上也叫n元语法（n-grams），它是基于$n-1$阶马尔可夫链的概率语言模型。例如，当$n=2$时，含有4个词的文本序列的概率就可以改写为：

$$\begin{aligned} P(w_1, w_2, w_3, w_4) &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)P(w_4 \mid w_1, w_2, w_3) \\ &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2)P(w_4 \mid w_3) \end{aligned}$$

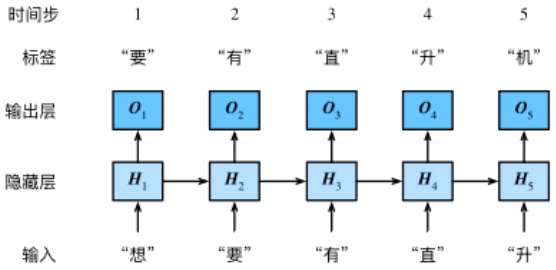
3. 缺陷

如果n很大，通过样本得到的频率会很小，导致数据稀疏；

4. 时序数据的采样方式

- 随机采样： 每个样本是原始序列上任意截取的一段序列，相邻的两个随机小批量在原始序列上的位置不一定相毗邻；
- 相邻采样： 相邻的两个随机小批量在原始序列上的位置相毗邻；

循环神经网络基础



1. 基本结构

在基础神经网络的基础上加入了“历史影响部分”，加上原有的“线性部分”，最后经过非线性部分(激活函数)的处理；

我们先看循环神经网络的具体构造。假设$\boldsymbol{X}_t \in \mathbb{R}^{n \times d}$是时间步$t$的小批量输入，$\boldsymbol{H}_t \in \mathbb{R}^{n \times h}$是该时间步的隐藏变量，则：

$$\boldsymbol{H}_t = \phi(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h).$$

其中，$\boldsymbol{W}_{xh} \in \mathbb{R}^{d \times h}$，$\boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$，$\boldsymbol{b}_h \in \mathbb{R}^{1 \times h}$，$\phi$函数是非线性激活函数。由于引入了$\boldsymbol{H}_{t-1} \boldsymbol{W}_{hh}$，$H_t$能够捕捉截至当前时间步的序列的历史信息，就像是神经网络当前时间步的状态或记忆一样。由于H_t的计算基于H_{t-1}，上式的计算是循环的，使用循环计算的网络即循环神经网络（recurrent neural network）。

在时间步t，输出层的输出为：

$$\boldsymbol{O}_t = \boldsymbol{H}_t \boldsymbol{W}_{hq} + \boldsymbol{b}_q.$$

其中$\boldsymbol{W}_{hq} \in \mathbb{R}^{h \times q}$，$\boldsymbol{b}_q \in \mathbb{R}^{1 \times q}$。

2. one-hot编码

常用的转换类别变量的方法

3. 梯度裁剪

用来避免梯度爆炸的情况，将其2范数限制在一个阈值内

循环神经网络中较容易出现梯度衰减或梯度爆炸，这会导致网络几乎无法训练。裁剪梯度（clip gradient）是一种应对梯度爆炸的方法。假设我们把所有模型参数的梯度拼接成一个向量$\boldsymbol{g}$，并设裁剪的阈值是θ。裁剪后的梯度

$$\min\left(\frac{\theta}{\|\boldsymbol{g}\|}, 1\right)\boldsymbol{g}$$

的L_2范数不超过θ。

4. 困惑度

我们通常使用困惑度（perplexity）来评价语言模型的好坏。回忆一下"softmax回归"一节中交叉熵损失函数的定义。困惑度是对交叉熵损失函数做指数运算后得到的值。特别地，

- 最佳情况下，模型总是把标签类别的概率预测为1，此时困惑度为1；
- 最坏情况下，模型总是把标签类别的概率预测为0，此时困惑度为正无穷；
- 基线情况下，模型总是预测所有类别的概率都相同，此时困惑度为类别个数。

显然，任何一个有效模型的困惑度必须小于类别个数。在本例中，困惑度必须小于词典大小`vocab_size`。