20377199 赵芮箐 第5周作业

实现文本处理的Tokenizer类

- 深度学习处理自然语言时，会常常用到Tokenizer（https://huggingface.co/transformers/tokenizer_summary.html）。简单来说，就是按照预先定义好的词典，把文本编码成整数序列的过程。深度学习模型进行文本挖掘任务时会经常需要处理这种编码过的序列。在构建过程中，有时候我们希望句子的长度能够整齐，所以会规定一个特殊的号码[PAD]=0，代表填充位。具体地，例如词典为 {'[PAD]': 0, '我':1, '是': 2, '北京': 3, '大学生': 4, '的': 5 }，那么"我是北京的大学生" 将被编码为 [1, 2, 3, 5, 4]；如果需要句子长度为8，那么我们在后面填充[PAD]，得到 [1,2,3,5,4,0,0,0]。

任务一：实现一个基础的中文Tokenizer类

1.Tokenizer类实现

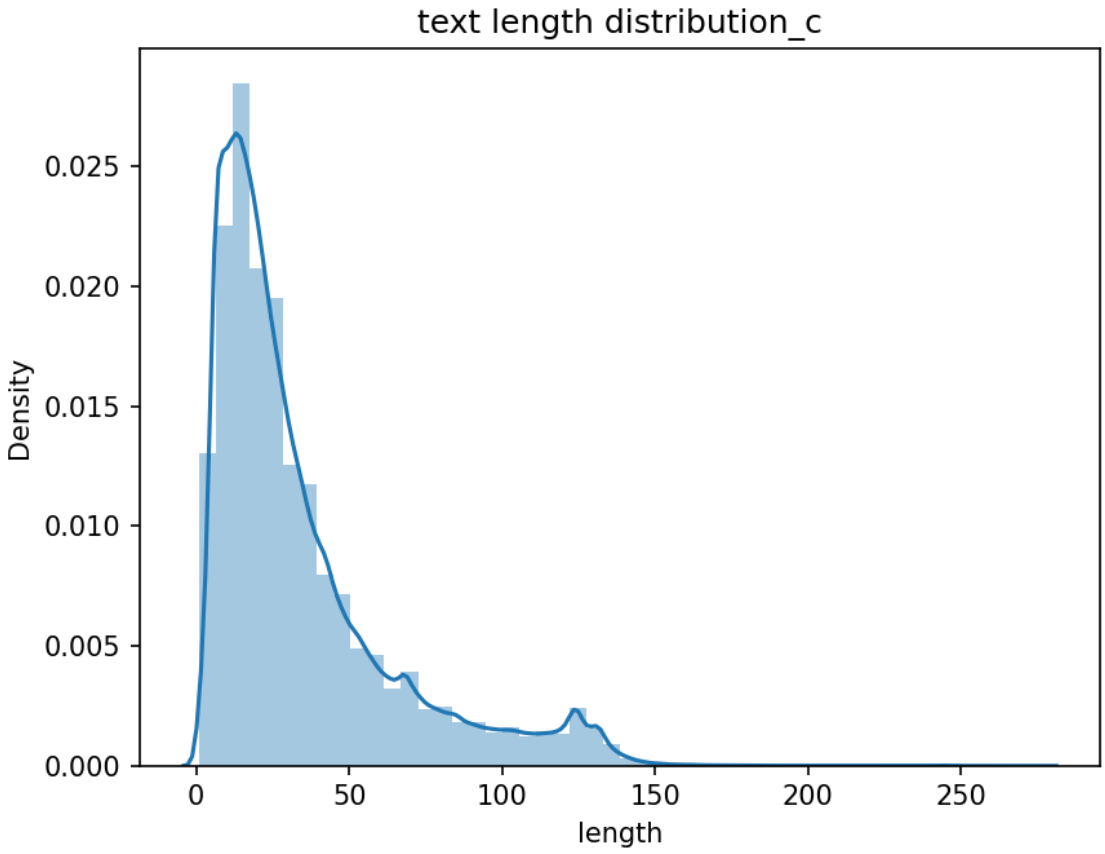
```
class Tokenizer:
    def __init__(self, chars, coding='c', PDA=0):
        """
        初始化函数：初始化类，并且构建编码词典
        """
        self._chars = chars
        self._coding = coding
        self._PDA = PDA

        dic = {}
        dic['[PAD]'] = PDA
        i = 0                                       #从0开始编码
        if coding=='c':
            for char in chars:
                for ch in char:
                    if ch not in dic.keys():
                        if i == PDA: i+=1           #跳过PDA的值
                        dic[ch] = i; i+=1
        elif coding=='w':
            for char in chars:
                for ch in jieba.lcut(char):
                    if ch not in dic.keys():
                        if i == PDA: i+=1
                        dic[ch] = i; i+=1
        self._dic = dic

    def tokenize(self, sentence):
        """
        分词（字）函数：输入一句话，返回分词（字）后的字符列表
        """
        list_of_chars = []
        if self._coding == 'c':
            for char in sentence:
                list_of_chars.append(char)
        elif self._coding == 'w':
```

```
            list_of_chars = jieba.lcut(sentence)
        return list_of_chars

    def encode(self, list_of_chars):
        """
        编码函数：输入字符列表，返回转化后的数字列表
        """
        tokens = []
        for char in list_of_chars:
            tokens.append(self._dic[char])
        return tokens

    def get_seq_len(self):
        """
        得长函数:观察句子长度分布，确定一个合适的seq_len
        """
        text = self._chars
        text_len = []
        if self._coding == 'c':
            for txt in text:
                text_len.append(len(txt))
        elif self._coding  == 'w':
            for txt in text:
                text_len.append(len(jieba.lcut(txt)))
        sns.distplot(text_len)
        plt.title('text length distribution_{}'.format(self._coding))   # 标题
        plt.xlabel('length')                                            # x轴名
        #plt.savefig('text length distribution_{}.png'.format(self._coding))
        plt.show()
        seq_len = np.percentile(text_len, 75)       #用75%分位数作为seq_len的值
        return int(seq_len)

    def trim(self, tokens, seq_len):
        """
        整长函数：输入数字列表tokens，整理数字列表的长度。不足seq_len的部分用PAD补足，超过的部分截断。
        """
        if len(tokens) >= seq_len:
            return tokens[:seq_len]
        else:
            add_len = seq_len - len(tokens)
            tokens_trim = tokens + [self._PDA]*add_len
            return tokens_trim

    def decode(self, tokens):
        """
        翻译函数：将模型输出的数字列表翻译回句子，如果有PAD，输出'[PAD]'
        """
        key = list(self._dic.keys())
        value = list(self._dic.values())
        chars = []
        for code in tokens:
            if code == self._PDA:
                chars.append('[PDA]')
            else:
```

```
                ch = key[value.index(code)]
                chars.append(ch)
        print("".join(chars))

    def encode_all(self, seq_len):
        """
        编码文本：返回所有文本(chars)的长度为seq_len的tokens
        """
        tokens_list = []
        for sen in self._chars:
            sen_token = self.trim(self.encode(self.tokenize(sen)), seq_len)
            tokens_list.append(sen_token)
        return tokens_list
```

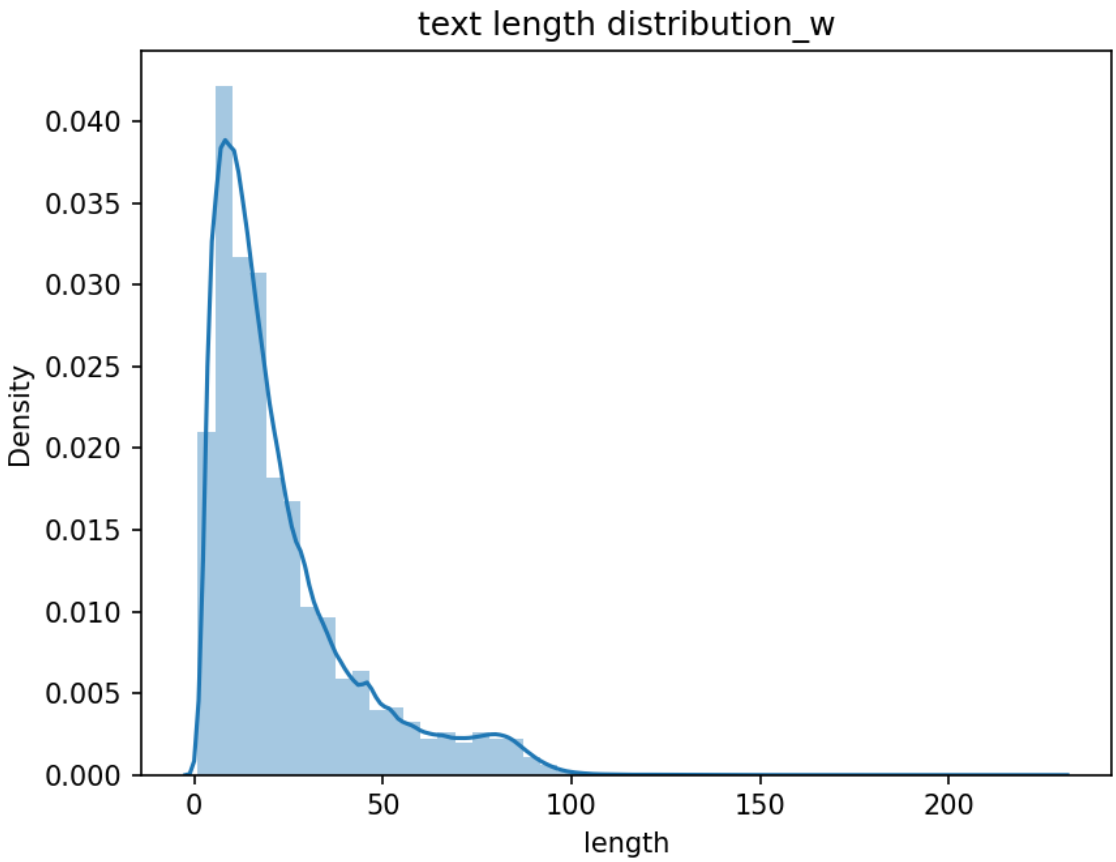
2.确定合适的长度（seq_len）

- 按字划分：



返回的75%分位数是44，则**取44为按字划分该文本下的seq_len**

- 按词划分：



返回的75%分位数是30，则**取30为按词划分该文本下的seq_len**

3.结果展示

- 构建的字典（部分）
  - 按字：


```
{'[PAD]': 0, '若': 1, '苦': 2, '追': 3, '求': 4, '过': 5, '却': 6, '还': 7, '是': 8, '一': 9, '无': 10, '所': 11, '获': 12, ',': 13, '又': 14, '何': 15, '必': 16, '硬': 17, '要': 18, '去': 19, '强': 20, '呢': 21, '?': 22, '不': 23, '妨': 24, '这': 25, '样': 26, '安': 27, '慰': 28, '自': 29, '己': 30, ':': 31, '该': 32, '你': 33, '的': 34, '躲': 35, '也': 36, ';': 37, '来': 38, '。': 39, '费': 40, '尽': 41, '心': 42, '思': 43, '绞': 44, '脑': 45, '汁': 46, '地': 47, '占': 48, '有': 49, '那': 50, '些': 51, '原': 52, '本': 53, '属': 54, '于': 55, '东': 56, '西': 57, ' ': 58, '很': 59, '久': 60, '没': 61, '情': 62, '弄': 63, '大': 64, '餐': 65, '今': 66, '晚': 67, '雅': 68, '兴': 69, '!': 70, '顺': 71, '便': 72, '发': 73, '图': 74, '国': 75, '博': 76, '电': 77, '信': 78, '3': 79, 'G': 80, '太': 81, '让': 82, '人': 83, '抓': 84, '狂': 85, '了': 86, '到': 87, '最': 88, '后': 89, '几': 90, '幅': 91, '画': 92, '微': 93, '导': 94, '览': 95, '收': 96, '只': 97, '能': 98, '回': 99, '家': 100, '再': 101, '听': 102, '～': 103, '计': 104, '划': 105, '看': 106, '五': 107, '个': 108, '展': 109, '逛': 110
```


  - 按词：


```
{'[PAD]': 0, '若': 1, '苦苦': 2, '追求': 3, '过': 4, '却': 5, '还是': 6, '一无所获': 7, ',': 8, '又': 9, '何必': 10, '硬要': 11, '去': 12, '强求': 13, '呢': 14, '?': 15, '不妨': 16, '这样': 17, '安慰': 18, '自己': 19, ':': 20, '该是': 21, '你': 22, '的': 23, '躲': 24, '也': 25, '不过': 26, ';': 27, '不是': 28, '求': 29, '不来': 30, '。': 31, '要': 32, '费尽心思': 33, '绞尽脑汁': 34, '地去': 35, '占有': 36, '那些': 37, '原本': 38, '不': 39, '属于': 40, '东西': 41, ' ': 42, '很久没': 43, '心情': 44, '弄': 45, '大餐': 46, '今晚': 47, '雅兴': 48, '!': 49, '顺便': 50, '发图': 51, '国博': 52, '电信': 53, '3G': 54, '太': 55, '让': 56, '人': 57, '抓狂': 58, '了': 59, '到': 60, '最后': 61, '还有': 62, '几幅': 63, '画': 64, '微信': 65, '导览': 66, '没收': 67, '只能': 68, '回家': 69, '再': 70, '听': 71, '～': 72, '原': 73, '计划': 74, '看': 75, '五个': 76, '展': 77, '逛': 78, '一': 79, '下午': 80, '只': 81, '三个': 82, '《': 83, '列支敦士登': 84, '王室': 85, '收藏': 86, '》': 87, '、': 88, '嘉德': 89, '20': 90, '年': 91, '大美木艺': 92, '地中海': 93, '文明': 94, '下次': 95, '今天': 96, '前所未有': 97, '人山人海': 98, '[': 99, ']': 100, '一张': 101, '桌子': 102, '大家': 103, '可以': 104, '猜猜': 105, '用途': 106, '"': 107, '我會講': 108,
```


- 编码及解码
  - 按字：

```
-----长度大于seq_len-----
#text#
若苦苦追求过却还是一无所获，又何必硬要去强求呢？不妨这样安慰自己：该是你的，躲也躲不过；不是你的，求也求不来。又何必要费尽心思绞尽脑汁地去占有那些原本不属于的东西呢？
#encode#
[1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 4, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 8, 33, 34, 13, 35, 36, 35, 23, 5, 37]
#decode#
若苦苦追求过却还是一无所获，又何必硬要去强求呢？不妨这样安慰自己：该是你的，躲也躲不过；
-----长度小于seq_len-----
#text#
很久没心情弄大餐，今晚雅兴！顺便发图！
#encode#
[59, 60, 61, 42, 62, 63, 64, 65, 13, 66, 67, 68, 69, 70, 71, 72, 73, 74, 70, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#decode#
很久没心情弄大餐，今晚雅兴！顺便发图！ [PDA]
```

  - 按词：

```
-----长度大于seq_len-----
#text#
若苦苦追求过却还是一无所获，又何必硬要去强求呢？不妨这样安慰自己：该是你的，躲也躲不过；不是你的，求也求不来。又何必要费尽心思绞尽脑汁地去占有那些原本不属于的东西呢？
#encode#
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 8, 24, 25, 24, 26, 27, 28]
#decode#
若苦苦追求过却还是一无所获，又何必硬要去强求呢？不妨这样安慰自己：该是你的，躲也躲不过；不是
-----长度小于seq_len-----
#text#
很久没心情弄大餐，今晚雅兴！顺便发图！
#encode#
[43, 44, 45, 46, 8, 47, 48, 49, 50, 51, 49, 42, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#decode#
很久没心情弄大餐，今晚雅兴！顺便发图！ [PDA]
```

4.思考：tokenizer方法与one-hot方法编码之间的区别和优劣

- 区别：

  tokenizer是通过对每个出现的字进行唯一对应的字（词）与编码的字典构建，来对句子进行编码。

  one-hot是通过对特征词的筛选，然后判断句子是否包含特征词来进行编码。

- 优劣：

  tokenizer是有序的，one-hot是无序的，所以tokenizer能确保句子的唯一性

  tokenizer的维数会比one-hot少，储存空间会比one-hot小很多。而且one-hot储存的向量，零向量很多，矩阵很稀疏，tokenizer避免浪费了这么多的空间。

任务二：(附加) 使用BERT预训练模型中提供的tokenizer

1.对本次样本中的文本进行编码

```
from transformers import BertTokenizer

bert_name = 'bert-base-chinese'
tokenizer = BertTokenizer.from_pretrained(bert_name)

input_ids = tokenizer.encode(
                        text = data[1],
                        text_pair = data[4],       # 可以编码一个/两个句子
                        truncation=True,           # 当句子长度大于max_length时截断
                        padding='max_length',      # 补pad到max_length长度
                        add_special_tokens=True,   # 添加special tokens,也就是CLS和
SEP
                        max_length=30,             # 设定最大文本长度
                        return_tensors=None        # 可取值tf,pt,np,默认为返回list
                    )
```

```
---text---
 很久没心情弄大餐，今晚雅兴！顺便发图！
---text_pair---
  我太喜欢它了
---input_ids---
 [101, 2523, 719, 3766, 2552, 2658, 2462, 1920, 7623, 8024, 791, 3241, 7414, 1069, 8013, 7556, 912, 1355, 1745, 8013, 102, 2769, 1922, 1599
, 3614, 2124, 749, 102, 0, 0]
---decode---
 [CLS] 很 久 没 心 情 弄 大 餐 ， 今 晚 雅 兴 ！ 顺 便 发 图 ！ [SEP] 我 太 喜 欢 它 了 [SEP] [PAD] [PAD]
```

这里用的就是最简单的编码函数，还有增强的编码函数、批量编码句子函数和批量编码成对句子函数！很好用了就是说。还可以看到的是这个tokenizer划分句子是按字划分的。

> BERT模型的tokenizer除了[PAD]这个特殊编码外，还有另外俩特殊编码。
>
> [CLS]：将对应的输出作为文本的语义表示，会用于文本分类任务
>
> [SEP]：对输入的两句话用一个[SEP]符号作分割，并分别对两句话附加两个不同的文本向量以作区分。

2.使用其预训练模型得到句子的表征向量

```
from transformers import BertModel

bert_name = 'bert-base-chinese'
bert_model = BertModel.from_pretrained(bert_name)

tokens = tokenizer.encode_plus(text=data[1], return_tensors='pt')
sen_vec = bert_model(**tokens)
```

```
BaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=tensor([[[-0.1920,  0.8330,  0.1294,  ...,  0.2733,  0.7649, -0.1034],
         [ 0.0556,  0.5174,  1.2869,  ..., -1.0265, -0.2023, -0.2523],
         [-0.5605,  0.2495,  0.1715,  ...,  0.0415,  0.5819, -0.2613],
         ...,
         [-0.1377, -0.2830,  0.1659,  ...,  0.0930,  1.3227,  0.0645],
         [-0.3193,  0.1122,  0.4227,  ..., -0.1177,  0.7251, -0.3740],
         [-0.5363,  0.4137,  0.3142,  ..., -0.3300,  0.7210, -0.3559]]],
```

> 这里输出的是是模型最后一层输出的隐藏状态，实际上是每个词会被转换成**768维**的向量表示，然后再共同组成句子的表征向量

3.抽取本次数据中的一条文本，并查找其相似文本

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

sbert_model = SentenceTransformer(bert_name)

sentence_embeddings = sbert_model.encode(data)
#计算第一句向量和其余句子的余弦相似度，即语义相似度
comp_sim = cosine_similarity([sentence_embeddings[1]],sentence_embeddings[1:])
```

```
---余弦相似度---
 [0.8611549139022827, 0.6553544998168945, 0.6455197334289551, 0.7414307594299316, 0.7746103405952454, 0.7600514888763428, 0.7775270342826843, 0.6947007775306702, 0.8047978281974792, 0.7866750955581665, 0.7798540592193604, 0.8264040946960449, 0.7295289635658264, 0.6968473196029663, 0.7662327289581299, 0.821499228477478, 0.8196771740913391, 0.7152970433235168, 0.7918980717658997, 0.7038840055465698, 0.8128366470336914, 0.6415748000144958, 0.6498414278030396, 0.8166165351867676, 0.7863913178443909, 0.77877277135849, 0.5513163208961487, 0.7256449460983276, 0.6691147089004517, 0.6965814232826233, 0.8224854469299316, 0.7724271416664124, 0.7539234757423401, 0.8024623394012451, 0.8008068799972534, 0.7487412095069885, 0.5659757256507874, 0.6850701570510864, 0.6580640077590942, 0.7499516010284424, 0.8031402826309204, 0.7746409177780151, 0.7300928831100464, 0.8124337196350098, 0.6538482904434204, 0.6728290319442749, 0.7425083518028259, 0.8124809265136719, 0.767279744148254, 0.4412759542465210, 0.7359265089035034, 0.8625633120536804, 0.4218427836894989, 0.791566789150238, 0.8003641366958618, 0.7315494418144226, 0.7935746908187866, 0.7852552533149719, 0.7058371305465698, 0.5890829563140869, 0.6155953407287598, 0.8393773436546326, 0.8268294334411621, 0.2591176927089691, 0.7685687541961670, 0.7032935619354248, 0.8031611442565918, 0.8095548748970032, 0.7087712287902832, 0.8263974189758301, 0.7830874919891357, 0.7565965056419373, 0.6834869384765625, 0.7598683834075928, 0.7544331550598145, 0.7824113368988037, 0.8099573850631714, 0.6728398799896240, 0.4245623946189880, 0.7537526488304138, 0.6247129440307617, 0.8468209505081177, 0.8206266164779663, 0.8282279968261719, 0.8266761302947998, 0.7234483361244202, 0.8490976095199585, 0.4459100365638733, 0.5352753996849060, 0.6718251109123230, 0.8214111328125, 0.7962392568588257, 0.8171333074569702, 0.6900241374969482, 0.7123084068298340, 0.7868522405624390, 0.7683006525039673, 0.7339091300964355]
---原文本---
 很久没心情弄大餐，今晚雅兴！顺便发图！
---最相似文本---
 很久没有刷过微博了，今晚刷一下，顺便祈祷天气快点好起来。
```

> PS：就选了前100个句子进行分析

4.思考：与第二周中使用word2wec得到的结果相比较，分析两者优劣

- 无论是Bert还是word2vec，对词语的表征都利用了周围词的信息。但Bert比word2vec更好在还考虑了词序
- 对于语义相反词的问题，Bert处理的会更好，因为其预训练中，同时训练了next sentence prediction，相比word2vec而言，就会避免句子几乎相似只存在一个反义词的问题。

代码：

https://github.com/rachhhhing/mp2022_python/blob/master/week5/week5.py

Ref:

- BERT：https://zhuanlan.zhihu.com/p/98855346
- huggingface使用

  视频教程：https://www.bilibili.com/video/BV1a44y1H7Jc

  对应代码：https://github.com/lansinuote/Huggingface_Toturials
- BERT生成句向量+文本相似度：https://zhuanlan.zhihu.com/p/375495030