- 题目描述
  - 背景介绍
    - 中文分词方法介绍
      - 基于字符串匹配的分词方法
      - 基于统计的分词方法
      - 基于深度学习的文本分词方法
    - 任务实现
  - 双向最大匹配算法
    - 性能评估
  - CRF分词算法
    - 模型搭建
      - 实验环境
      - 实验数据
    - 训练结果
  - 基于Bert的文本分词方法
  - 子词压缩
    - 输入样本：
    - 参考分词
    - 分词结果
    - 子词压缩
  - 附录

题目描述

1. 利用已经学习过的理论方法和北京大学标注的《人民日报》 分词和词性标注语料，设计实现至少两种不同的汉语词语自动切分方法，进行性能测试和分析。
2. 利用不同类型的网络文本测试你的分词系统，对比分析分词方法和不同测试样本的性能变化。
3. 在1、2得到的分词结果的基础上，实现子词压缩。

背景介绍

中文分词(Chinese Word Segmentation) 指的是将一个汉字序列切分成一个一个单独的词。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。

中文分词方法介绍

现有的分词方法可分为三大类：基于字符串匹配的分词方法、基于理解的分词方法和基于统计的分词方法。

基于字符串匹配的分词方法

基于字符串匹配的分词方法又称机械分词方法，它是按照一定的策略将待分析的汉字串与一个“充分大的”机器词典中的词条进行配，若在词典中找到某个字符串，则匹配成功（识别出一个词）。

按照扫描方向的不同，字符串匹配分词方法可以分为正向匹配和逆向匹配；按照不同长度优先匹配的情况，可以分为最大（最长）匹配和最小（最短）匹配；按照是否与词性标注过程相结合，可以分为单纯分词方法和分词与词性标注相结合的一体化方法。常用的字符串匹配方法有如下几种：

- 正向最大匹配法（从左到右的方向）；
- 逆向最大匹配法（从右到左的方向）；
- 最小切分（每一句中切出的词数最小）；
- 双向最大匹配（进行从左到右、从右到左两次扫描）

这类算法的优点是速度快，时间复杂度可以保持在O（n）,实现简单，效果尚可；但对歧义和未登录词处理效果不佳。

基于统计的分词方法

基于统计的分词方法是在给定大量已经分词的文本的前提下，利用统计机器学习模型学习词语切分的规律（称为训练），从而实现对未知文本的切分。例如最大概率分词方法和最大熵分词方法等。随着大规模语料库的建立，统计机器学习方法的研究和发展，基于统计的中文分词方法渐渐成为了主流方法

主要的统计模型有：N元文法模型（N-gram），隐马尔可夫模型（Hidden Markov Model，HMM），最大熵模型（ME），条件随机场模型（Conditional Random Fields，CRF）等。

在实际的应用中，基于统计的分词系统都需要使用分词词典来进行字符串匹配分词，同时使用统计方法识别一些新词，即将字符串频率统计和字符串匹配结合起来，既发挥匹

配分词切分速度快、效率高的特点，又利用了无词典分词结合上下文识别生词、自动消除歧义的优点。

基于深度学习的文本分词方法

基于深度学习的自然语言处理（NLP）分词方法通常使用神经网络模型来学习词语的切分规则。这种方法的基本原理是将分词任务视为序列标注问题，即为输入序列中的每个字符标注一个标签，表示该字符是一个词的开始、中间、结束或者是单独成词。 例如：BiLSTM-CRF的输入层是一个embedding层，经过双向LSTM网络编码，输出层是一个CRF层。BiLSTM-CRF经过双向LSTM网络输出的实际上是当前位置对于各词性的得分，CRF层的意义是对词性得分加上前一位置的词性概率转移的约束，其好处是引入一些语法规则的先验信息。

任务实现

为了完成这个任务，我共使用了三种文本分词方法，分别是双向最大匹配算法，条件随机场分词方法，基于bert网络的文本分词方法。它们的实现和结果分析如下：

双向最大匹配算法

双向最大匹配算法是一种基于词典的分词方法，原理也十分简单，我们要做的就是从语料库中得到词典，再实现前向和后向最大匹配算法，最后整合为双向最大匹配算法即可。

得到词典的方式和上一个实验中获得词及词频文件的处理方式一致，只需要将其中的词频信息删除即可。

```
def return_dict(word_count_file,encoding='utf-8'):
    word_dict = set()
    with open(word_count_file, 'r', encoding=encoding) as f:
        for line in f:
            word = line.split('/')[0]  # 提取词语
            if word.startswith('[') and len(word) > 1:  # 如果词语以"["开头，并且长度
大于1
                word = word[1:]  # 去除左方括号
            word_dict.add(word)
    return word_dict
```

双向最大匹配的伪代码如下：

```
定义双向最大匹配算法(BiMM):
    输入: 待分词的句子sentence, 词典word_dict, 最大词长max_len
    输出: 分词结果

    定义前向最大匹配算法(FMM):
        输入: 待分词的句子sentence, 词典word_dict, 最大词长max_len
        输出: 分词结果
        初始化空列表result
        当sentence不为空时:
            对于长度i从max_len到1:
                如果sentence的前i个字符在word_dict中:
                    将sentence的前i个字符添加到result中
                    将sentence的前i个字符从sentence中删除
                    跳出循环
        返回result

    定义后向最大匹配算法(BMM):
        输入: 待分词的句子sentence, 词典word_dict, 最大词长max_len
        输出: 分词结果
        初始化空列表result
        当sentence不为空时:
            对于长度i从max_len到1:
                如果sentence的后i个字符在word_dict中:
                    将sentence的后i个字符添加到result的开头
                    将sentence的后i个字符从sentence中删除
                    跳出循环
        返回result

    使用FMM对sentence进行分词, 得到result_fmm
    使用BMM对sentence进行分词, 得到result_bmm

    如果result_fmm和result_bmm相同:
        返回result_fmm
    否则:
        如果result_fmm的分词数量少于result_bmm的分词数量:
            返回result_fmm
        否则如果result_bmm的分词数量少于result_fmm的分词数量:
            返回result_bmm
        否则:
            返回result_fmm
```

性能评估

性能评测结果如下：

对测试样本进行分词，可以看到双向最大匹配算法对词典中的已有的词划分结果很好，但是如果存在未知词就难以划分出正确结果。对于随机摘出的测试样本，双向最大匹配算法的precison,recall,f1分别为：

- **precision**: 0.959823

- **recall**: 0.971837
- **f1**: 0.965793

与课件中预期的结果一致，具有一定效果，但是在面对未知词和具有歧义的语料时难以得到正确的结果。
测试文本：

- EXAMPLE1

```
Input text: 北京大学生爱喝进口红酒
Result: 北京/大学生/爱/喝/进口/红/酒
```

- EXAMPLE2

```
Input text: 小华为了考试早晨买了一杯小米粥喝，让黄飞鸿蒙题目中有几个苹果，但是郭麒麟刷牙选中华为的就是干净，速度快，每次只挤5g就够用。我喜欢在大城市生活流浪地球不爆炸我就不退缩，平时也看看《东吴京剧》、《大战狼人》、《鸿蒙至尊》等经典电视剧。\我用中华为的就是便宜实惠，而且每次只用5g，我最喜欢的画家是达芬奇，尤其喜欢他的代表作 佛罗伦萨画派蒙娜丽莎。\秦始皇派蒙恬还原神舟十二对接并顺便一提瓦特改良了蒸汽机。

Result: 小/华/为了/考试/早晨/买/了/一/杯/小米/粥/喝/，/让/黄/飞/鸿/蒙/题目/中/有/几/个/苹果/，/但是/郭/麒麟/刷牙/选中/华为/的/就是/干净/，/速度/快/，/每次/只/挤/5/g/就/够/用/。/我/喜欢/在/大/城市/生活/流浪/地球/不/爆炸/我/就/不/退缩/，/平时/也/看看/《/东吴/京剧/》/、/《/大战/狼/人/》/、/《/鸿/蒙/至尊/》/等/经典/电视剧/。/\/我/用/中/华为/的/就是/便宜/实惠/，/而且/每次/只用/5/g/，/我/最/喜欢/的/画家/是/达/芬/奇/，/尤其/喜欢/他/的/代表作/佛/罗/伦/萨/画派/蒙/娜/丽/莎/。/\/秦始皇/派/蒙/恬/还原/神/舟/十二/对接/并/顺便/一/提/瓦/特/改良/了/蒸汽/机/。
```

CRF分词算法

条件随机场Conditional Random Field:CRF）是用来标注和划分序列结构数据的概率化结构模型。对于给定的输出，标识序列Y和观测序列X，条件随机场通过定义条件概率$P(Y|X)$，而不是联合概率分布$P(X,Y)$来描述模型。条件随机场试图对多个随机变量（它们代表标记序列）在给定观测序列的值之后的条件概率进行建模：令$X=X1,X2,\cdots,Xn$为观测变量序列，$Y=Y1,Y2,\cdots,Yn$为对应的标记变量序列。条件随机场的目标是构建条件概率模型P(Y丨X)。

即：已知观测变量序列的条件下，标记序列发生的概率。标记随机变量序列Y的成员之间可能具有某种结构：

- 在自然语言处理的词性标注任务中，观测数据为单词序列，标记为对应的词性序列（即动词、名词等词性的序列），标记序列具有线性的序列结构。

- 在自然语言处理的语法分析任务中，观测数据为单词序列，标记序列是语法树，标记序列具有树形结构。

令$G =< V , E >$表示与观测变量序列X和标记变量序列Y对应的无向图，Y_v表示与结点v对应的标记随机变量，$n(v)$表示结点v的邻接结点集。若图G中结点对应的每个变量Y_v都满足马尔可夫性，即：$P(Y_v \mid X, Y_{V-v}) = P(Y_v \mid X, Y_n(v))$。则$(Y , X)$构成了一个条件随机场。

图G可以具有任意结构，只要能表示标记变量之间的条件独立性关系即可。但在现实应用中，尤其是对标记序列建模时，最常用的是链式结构，即链式条件随机场chain-structured CRF。条件随机场使用势函数和团来定义条件概率$Y_{i-1}, Y_i, X, i = 2, \cdots , n$。采用指数势函数，并引入特征函数feature function，定义条件概率：

$$P(y/x) = \frac{1}{Z} exp(\sum_{j} \sum_{i=1}^{n-1} \lambda_j t_j(y_{i+1}, y_i, x, i) + \sum_{j} \sum_{i=1}^{n-1} \mu_k s_k(y_i, x, i))$$

其中：

• $t_j(Y_i, Y_{i+1}, X, i)$：在已知观测序列情况下，两个相邻标记位置上的转移特征函数transition feature function。它刻画了相邻标记变量之间的相关关系，以及观察序列X对它们的影响。位置变量i也对势函数有影响。比如：已知观测序列情况下，相邻标记取值(代词，动词)出现在序列头部可能性较高，而(动词，代词)出现在序列头部的可能性较低。

• $s_k(Y_i, X, i)$：在已知观察序列情况下，标记位置i上的状态特征函数status feature function。它刻画了观测序列X对于标记变量的影响。位置变量i也对势函数有影响。比如：已知观测序列情况下，标记取值名词出现在序列头部可能性较高，而动词出现在序列头部的可能性较低。

• λ_j, μ_k为参数Z为规范化因子（它用于确保上式满足概率的定义）。

特征函数通常是实值函数，用来刻画数据的一些很可能成立或者预期成立的经验特性。

中文分词指的是中文在基本文法上有其特殊性而存在的分词 。分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。我们知道，在英文的行文中，单词之间是以空格作为自然分界符的，而中文只是字、句和段能通过明显的分界符来简单划界，唯独词没有一个形式上的分界符，虽然英文也同样存在短语的划分问题，不过在词这一层上，中文比之英文要复杂得多、困难得多。因此CRF进行文本分词的基本思想就是：每

个字在构造一个特定的词语时都占据着一个确定的构词位置（即构词位）。常用的四位构词位：词首（B），词中（M）,词尾（E），独立词（S）。

我们需要做的就是定义特征模板，并将训练数据转换为CRF训练可用的数据格式。本次实验中使用的是利用词的上下文信息来定义特征模板，代码如下：

```
def word2features(self, sent, i):
        word = sent[i][0]
        # 构造特征字典
        features = {
            'bias': 1.0,
            'word': word,
            'word.isdigit()': word.isdigit(),
        }
        # 该字的前一个字
        if i > 0:
            word1 = sent[i - 1][0]
            words = word1 + word
            features.update({
                '-1:word': word1,
                '-1:words': words,
                '-1:word.isdigit()': word1.isdigit(),
            })
        else:
            # 添加开头的标识 BOS(begin of sentence)
            features['BOS'] = True
        # 该字的前两个字
        if i > 1:
            word2 = sent[i - 2][0]
            word1 = sent[i - 1][0]
            words = word1 + word2 + word
            features.update({
                '-2:word': word2,
                '-2:words': words,
                '-3:word.isdigit()': word2.isdigit(),
            })
        # 该字的前三个字
        if i > 2:
            word3 = sent[i - 3][0]
            word2 = sent[i - 2][0]
            word1 = sent[i - 1][0]
            words = word1 + word2 + word3 + word
            features.update({
                '-3:word': word3,
                '-3:words': words,
                '-3:word.isdigit()': word3.isdigit(),
            })
        # 该字的后一个字
        if i < len(sent) - 1:
            word1 = sent[i + 1][0]
            words = word1 + word
            features.update({
                '+1:word': word1,
                '+1:words': words,
```

```
            '+1:word.isdigit()': word1.isdigit(),
        })
    else:
        # 句子的结尾添加对应的标识end of sentence
        features['EOS'] = True
    # 该字的后两个字
    if i < len(sent) - 2:
        word2 = sent[i + 2][0]
        word1 = sent[i + 1][0]
        words = word + word1 + word2
        features.update({
            '+2:word': word2,
            '+2:words': words,
            '+2:word.isdigit()': word2.isdigit(),
        })
    # 该字的后三个字
    if i < len(sent) - 3:
        word3 = sent[i + 3][0]
        word2 = sent[i + 2][0]
        word1 = sent[i + 1][0]
        words = word + word1 + word2 + word3
        features.update({
            '+3:word': word3,
            '+3:words': words,
            '+3:word.isdigit()': word3.isdigit(),
        })
    return features
```

之后将训练数据输入到CRF模型中得到训练结果。

模型搭建

实验环境

实验中使用到的CRF模型是用`skelearn_crfsuite`的python搭建的。 sklearn_crfsuite库提供了许多有用的特性，包括：

- scikit-learn兼容的API：可以像使用scikit-learn的其他模型一样使用sklearn_crfsuite.CRF类。这意味着可以使用scikit-learn的工具，如交叉验证、网格搜索等，来训练和优化CRF模型。
- 特性函数：sklearn_crfsuite支持使用特性函数来定义CRF模型。
- 模型持久化：可以使用sklearn_crfsuite.CRF类的save和load方法来保存和加载你的模型。这使得你可以在不同的Python会话中使用同一个模型，或者在不同的机器上共享模型。

实验数据

得到训练用的的数据的过程也很简单，只需要将每个词语提取出去再对其中的每个单字进行BSME标注即可。接下来，将字符串中的每个词利用特征函数进行转化，然后标签即可每个词对应的BSME标注。最后就可以用训练数据在CRF中参与训练，得到最后的结果。

```
def load_data(self, file_path, encoding="gbk"):
        with open(file_path, 'r', encoding=encoding) as f:
            lines = f.readlines()

        bmes_data = []
        bmes_label = []

        for line in lines:
            words = line.split()
            words = [word.split('/')[0] for word in words]
            sentence = "".join(words)
            bmes_data.append(sentence)

            line_label = []
            for word in words:
                word = word.split('/')[0]
                if len(word) == 1:
                    line_label.append('S')
                else:
                    line_label.append('B')
                    for char in word[1:-1]:
                        line_label.append('M')
                    line_label.append('E')
            bmes_label.append(line_label)
        self.train_data = [self.sent2features(sent) for sent in bmes_data]
        self.train_label = bmes_label
```

训练结果

CRFs模型的分词效果很好，并且能够很好地面对未知样本，也能取得不错的效果。CRFs的precison,recall,f1分别为：

- **precision**: 0.999558
- **recall**: 0.999117
- **f1**: 0.999337

看到CRFs对于三个指标都取得了非常不错的效果，在面对未知样本时也有不错的分词效果。 测试文本：

- EXAMPLE1

```
Input text:  北京大学生爱喝进口红酒
Result: 北京大学生/爱/喝/进口/红酒
```

- EXAMPLE2

```
Input text:  小华为了考试早晨买了一杯小米粥喝，让黄飞鸿蒙题目中有几个苹果，但是郭麒麟刷牙选中华为的就是干净，速度快，每次只挤5g就够用。我喜欢在大城市生活流浪地球不爆炸我就不退缩，平时也看看《东吴京剧》、《大战狼人》、《鸿蒙至尊》等经典电视剧。\我用中华为的就是便宜实惠，而且每次只用5g，我最喜欢的画家是达芬奇，尤其喜欢他的代表 作佛罗伦萨画派蒙娜丽莎。\秦始皇派蒙恬还原神舟十二对接并顺便一提瓦特改良了蒸汽机。

Result:  小华/为了/考试/早晨/买/了/一/杯/小米粥喝/，/让/黄/飞鸿/蒙题目/中/有/几/个/苹果/，/但是/郭/麒麟/刷/牙选/中华/为/的/就/是/干净/，/速度/快/，/每次/只/挤/5g/就/够/用/。/我/喜欢/在/大/城市/生活/流浪/地球/不/爆炸/我/就/不/退缩/，/平时/也/看看/《/东吴/京剧/》/、/《/大战/狼人/》/、/《/鸿蒙至尊/》/等/经典/电视剧/。/\/我/用/中 华为/的/就/是/便宜/实惠/，/而且/每次/只/用/5g/，/我/最/喜欢/的/画家/是/达/芬奇/，/尤其/喜欢/他/的/代表作/佛罗伦萨/画派/蒙娜/丽莎/。/\/秦始皇派/蒙恬/还/原/神舟/十二/对/接并/顺便/一提瓦特/改良/了/蒸汽机/。
```

基于**Bert**的文本分词方法

BERT（Bidirectional Encoder Representations from Transformers）是由Google在2018年提出的一种预训练语言模型。它的主要特点是使用了Transformer的编码器结构，并且采用了双向的自注意力机制（bidirectional self-attention mechanism）。

BERT的主要创新之处在于它是完全无监督的学习，主要使用了两种训练策略：

- Masked Language Model（MLM）：在输入序列中随机遮盖一些单词，然后预测这些被遮盖的单词。
- Next Sentence Prediction（NSP）：给定两个句子，预测第二个句子是否是第一个句子的下一句。

通过这两种训练策略，BERT能够学习到词汇之间以及句子之间的深层次语义关系。在预训练完成后，BERT可以通过添加少量的任务特定层，对各种NLP任务进行微调，如情感分析、命名实体识别、问答系统等。

利用bert来做中文文本分词任务，也是一个Seq2Seq任务，给定输入文本，输出对应的BMES标注序列，从而得到分词结果。基于这个思路，需要做的就是构造训练集，搭建模型和进行训练，具体流程请参考{}一书第七章。

测试结果： 对测试样本进行分词，可以看到bert的训练结果并不是很好，很重要的原因我猜想是因为语料库有限，模型无法充分地学习到语义之间的相关性。对于随机摘出的测试样本，双向最大匹配算法的precison,recall,f1分别为：

- **precision**: 0.660485
- **recall**: 0.491135
- **f1**: 0.56335

这个结果与预期结果相差甚远，最大区别就在于我进行训练的语料库仅仅只是人民日报一个月的内容，原文使用的是lcwb2语料。并且由于机能有限，进行训练时设置的batch_size也有限，这在一方面可能也导致了较差的效果。

测试文本：

- EXAMPLE1

```
Input text: 北京大学生爱喝进口红酒
Result: 北京/大学/生/爱/喝/进口/红/酒
```

- EXAMPLE2

```
Input text: 小华为了考试早晨买了一杯小米粥喝，让黄飞鸿蒙题目中有几个苹果，但是郭麒麟刷牙选中华为的就是干净，速度快，每次只挤5g就够用。我喜欢在大城市生活流浪地球不爆炸我就不退缩，平时也看看《东吴京剧》、《大战狼人》、《鸿蒙至尊》等经典电视剧。我用中华为的就是便宜实惠，而且每次只用5g，我最喜欢的画家是达芬奇，尤其喜欢他的代表作佛罗伦萨画派蒙娜丽莎。秦始皇派蒙恬还原神舟十二对接并顺便一提瓦特改良了蒸汽机。

Result: 小华/为/了/考试/早/晨买/了/一/杯/小/米/粥/喝/，/让/黄飞/鸿蒙/题/目/中/有/几个/苹果/，/但/是/郭/麒/麟刷/牙/选/中华/为/的/就/是/干净/，/速/度/快/，/每/次只/挤/5g/就/够用/。/我/喜欢/在/大城/市/生/活/流/浪/地/球/不/爆/炸/我/就/不/退/缩/，/平/时/也/看/看/《/东/吴/京剧/》/、/《/大/战/狼/人/》/、/《/鸿/蒙/至尊/》/等/经典/电视/剧/。/我用/中华/为/的/就/是/便/宜/实惠/，/而/且/每/次/只/用/5g/，/我/最/喜/欢/的/画/家/是/达/芬/奇/，/尤/其/喜欢/他/的/代/表作/佛/罗/伦/萨/画派/蒙/娜/丽/莎/。/秦/始皇/派/蒙/恬/还/原/神/舟/十/二/对/接/并/顺便/一/提/瓦/特/改良/了/蒸/汽/机/。
```

训练后的bert分词结果比较怪异，并且很少出现M的词性标注。这可能也是因为原本语料库中这样的词汇较少，并且许多词都是单词或者双字词，语料信息也较少，最终导致结果不佳。

子词压缩

子词压缩（Subword Tokenization）是一种在自然语言处理中常用的技术，它的主要目标是将词汇分解为更小的、有意义的组成部分，这些部分被称为子词或子词单元。

子词压缩的主要优点是可以有效地处理稀有词和词汇表外的词。通过将这些词分解为已知的子词单元，模型可以更好地理解和处理这些词。此外，子词压缩还可以减少词汇表的大小，从而提高模型的计算效率。

常见的子词压缩算法包括Byte Pair Encoding（BPE）、Unigram Language Model Tokenization、SentencePiece等。这些算法的具体实现方式各不相同，但都是通过统计语料库中的词频来确定最常见的子词单元。

以下是使用BPE算法来实现子词压缩功能：

输入样本：

Input text: 在京剧发展的历史上，企业、企业家的参与是曾经产生过重要作用的。众所周知，最有名的，曾培养了梅兰芳、周信芳、马连良、谭富英、裘盛戎、袁世海等艺术大师的富 连成科班，就是老一代企业家，吉林牛子厚和叶春善、肖长华等老一代京剧教育家共同创办的。而梅兰芳、余叔岩两位大师的成长和梅派、余派的艺术建设又得到了当年的中国银行冯耿兴、盐业银行张伯驹的支持和帮助。北京同仁堂、瑞蚨祥等企业，都曾积极地参与过京剧界的各种活动，为京剧艺术的发展做出了贡献。这些，都是历史佳话，也是优良传统，作为现代企业应当珍视这一传统，继承这一传统。

参考分词

tokenized text: 在 京剧 发展 的 历史 上 ， 企业 、 企业家 的 参与 是 曾经 产生 过 重要 作用 的 。 众所周知 ， 最 有名 的 ， 曾 培养 了 梅 兰芳 、 周 信芳 、 马 连良 、 谭 富英 、 裘 盛戎 、 袁 世海 等 艺术 大师 的 富连成 科班 ， 就是 老 一 代 企业家 ， 吉林 牛 子厚 和 叶 春善 、 肖 长华 等 老 一 代 京剧 教育家 共同 创办 的 。 而 梅 兰芳 、 余 叔岩 两 位 大师 的 成长 和 梅派 、 余派 的 艺术 建 设 又 得到 了 当年 的 中国银行 冯 耿兴 、 盐业 银行 张 伯驹 的 支持 和 帮助 。 北京 同仁堂 、 瑞蚨祥 等 企业 ， 都 曾 积 极 地 参与 过 京剧界 的 各种 活动 ， 为 京剧 艺术 的 发展 做出 了 贡献 。 这些 ， 都 是 历史 佳话 ， 也 是 优良 传统 ， 作为 现代 企业 应当 珍视 这 一 传统 ， 继承 这 一 传统 。

分词结果

Result: 在/京剧/发展/的/历史/上/，/企业/、/企业家/的/参与/是/曾经/产生/过/重要/作用/的/。/众所周知/，/最/有名/的/，/曾/培养/了/梅/兰芳/、/周/信芳/、/马/连良/、/谭/富英/、/裘/盛戎/、/袁/世海/等/艺术/大师/的/富连成/科班/，/就是/老/一/代/企业家/，/吉林/牛/子厚/和/叶/春善/、/肖/长华/等/老/一/代/京剧/教育家/共同/创办/的/。/而/梅/ 兰芳/、/余/叔岩/两/位/大师/的/成长/和/梅派/、/余派/的/艺术/建设/又/得到/了/当年/的/中国银行/冯/耿兴/、/盐业/银行/张/伯驹/的/支持/和/帮助/。/北京/同仁堂/、/瑞蚨祥/等/企业/，/都/曾/积极/地/参与/过/京剧界/的/各种/活动/，/为/京剧/艺术/的/发展/做出/了/贡献/。/这些/，/都/是/历史/佳话/，/也/是/优良/传统/，/作为/现代/企业/应当/珍视/这/一/传统/，/继承/这/一/传统/。/

子词压缩

子词压缩结果：（/为分隔符，加粗与分词结果不一致处） zip: 在/京剧/发展/的/历史/上/，/企业/、/企业家/的/参与/是/曾经/产生/过/重要/作用/的。/众所周知/，/最/有名/的/，/曾/培养/了/梅兰芳/、/周/信芳/、/马/连良/、/谭/富英/ 、/裘/盛戎/、/袁/世海/等/艺术/大师的/富连成/科班/，/就是/老一代/企业家/，/吉林/牛/子厚/和/叶/春善/、/肖/长华/等/老一代/京剧/教育家/共同/创办/的。/而/梅兰芳/、/余/叔 岩/两/位/大师的/成长/和/梅派/、/余派/的/艺术/建设/又/得到/了/当年/的/中国银行/冯/耿兴/、/盐业/银行/张/伯驹/的/支持/和/帮助/。/北京/同仁堂/、/瑞蚨祥/等/企业/，都/曾/积极/地/参与/过/京剧界/的/各种/活动/，/为/京剧/艺术/的/发展/做出/了/贡献/。/这些/，都/是/历史/佳话/，/也/是/优良/传统，/作为/现代/企业/应当/珍视/这一/传统，/继承/这一/传统/。

不难看出，分词结果中对新出现的词语如“老一代”等没有进行很好的分割。而子词压缩的结果则较好的将其进行了区分，这一压缩节省了表达信息所需的复杂程度。可以预见的是，随着压缩规模的扩大，压缩的效果也会有一定的提升。

bert模型及搭建均参考《自然语言处理基础与大模型》第七章，但未使用飞浆框架，而是等效替换为了pytorch框架函数。

附录

文件名	说明
homework2	
├── checkpoint	#Bert模型训练保存的参数位置
├── data	

文件名	说明
│ ├── word_count.txt	#生成的语料计数文件
│ ├── test.txt	#随机提取出来的测试文件
│ └── ...	各类模型中间生成的数据集保存位置
├── PKU_TXT	
│ └── ChineseCorpus199801.TXT	#语料库文件
├── pics #各类结果图片	
│ └── ...	
├── Bert	#Bert模型配置文件
│ ├── BEPT.py	#Bert模型训练文件
│ ├── Dataloader.py	#数据集构建文件
│ ├── main.py	#主程序，执行测试
│ └── net_dataset.py	#数据集预处理文件
├── bi_MM.py	#双向最大匹配算法
├── main.py	#主程序，训练或评估
├── CRFs.py	#调节随机场模型
├── handle_train_data.py	#模型数据集构建
├── bpe.py	# 子词压缩算法
└── main.py	# 主程序，比对各种模型效果并进行子词压缩