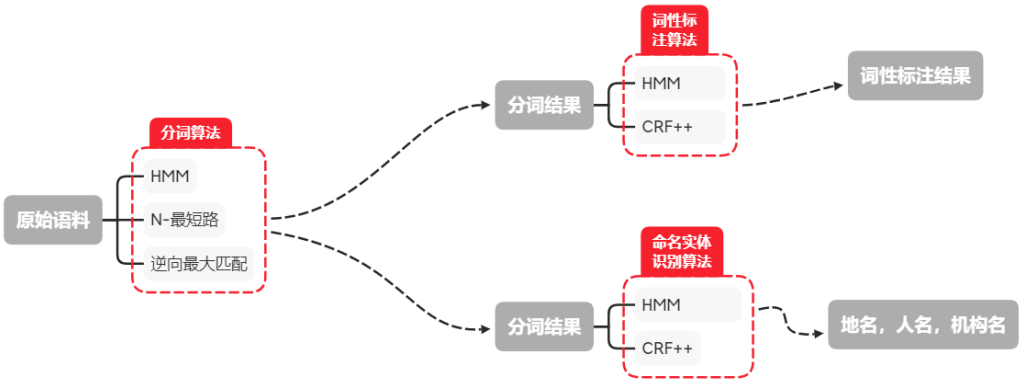
自然语言理解 课程大作业一

分词与词性标注

07111805 1120181319 崔晨曦

一、项目概况



本项目是一个使用 python 实现的词法分析工具包，集成了分词，词性标注以及命名实体识别等功能。项目处理流程如上图所示。

对于分词，本项目提供了三种不同的算法实现：隐马尔可夫(HMM)，N-最短路和逆向最大匹配。其中隐马尔可夫是基于统计的算法，而 N-最短路及逆向最大匹配均是基于词典的算法。

对于词性标注，本项目提供了两种算法实现，HMM 和条件随机场(CRF++)。其中 CRF++使用 python 作为胶水语言，调用底层的开源 C++接口，拥有较高的效率。

对于命名实体识别(NER)，本项目同样提供了 HMM 和 CRF++的两种实现。

二、项目实现

2.1 数据及预处理

2.1.1 数据集

本项目的数据集使用的是“北大语料库加工规范：切分·词性标注·注音”(2003 规范)，https://klcl.pku.edu.cn/gxzy/231686.htm。大小约 9MB，共 23268 行，内容来源于 1998 年的人民日报。语料已经完成切分和词性标注，含 100 多种词性标签。

本项目使用的词典通过对训练语料处理计数后得到。此外还使用了 hanlp 中内置的 CoreNatureDictionary.txt 作为对照，其大小约 2MB，共收录 153091 个词。

将数据集的前 20869 行作为训练集，后 2398 行作为测试集，比例约为 8.7：1。

2.1.2 数据预处理

由于原始数据集中一行内含有多个句子，故预处理的第一步便是按句进行分行，保证前后词之间具有语义上的关联性，这对于 HMM 和 CRF++这种统计模型较为重要，按句分行

后训练集约有 49000 个句子，测试集约有 4300 个句子。

对于分词模块的数据，主要使用 split 方法进行字符串分割，辅以相关操作，得到按词划分，不含词性标签的训练数据集和测试数据集，存放于项目的./corpus/segment 路径下。

对于词性标注模块的数据，首先对原始数据集中出现的 100 多个词性标签按词性相近程度进行合并，有助于算法运行速度和准确率的提升。合并后共计 33 个标签：

标签	词性	标签	词性
f	方位词	nr	人名
z	状态词	r	代词
n	名词	y	语气词
o	拟声词	h	前接成分
q	量词	an	名形词
c	连词	d	副词
ad	副形词	s	处所词
nx	非汉字的字符串	t	时间词
vd	副动词	v	动词
u	助词	k	后接成分
l	习用语	j	简称
i	成语	m	数词
p	介词	e	叹词
nz	其他专有名词	nt	机构名
ns	地名	a	形容词
b	区别词	vn	名动词
w	标点符号		

之后进行分割，分离文本和标签等操作，处理后的数据文件保存在./corpus/pos 路径下。

对于 NER 模块的数据，使用正则表达进行匹配，记录命名实体在句子中的起始终止位置，形成元组。本项目主要关注人名，地名，机构名等三类命名实体，处理后的数据文件保存在./corpus/NER 路径下。

2.2 分词

2.2.1 基于隐马尔可夫的分词

使用基于 HMM 的分词算法，首先按字符为数据打上 BMES 标签，B(begin)代表词汇的第一个字符，M(middle)代表词汇中间的字符，E(end)代表词汇末尾的字符，S(single)代表单字成词。

随后训练模型得到转移矩阵，发射矩阵和初始状态向量。然后对矩阵中的参数进行正则化处理，对发射矩阵进行平滑，将原本为 0 的项进行加 1 处理，最后再对所有参数取对数，目的是将之后的累乘操作用累加来替代，避免概率值趋近于 0.

运用动态规划的维特比算法找到概率值最大的路径，回溯得到结果。

2.2.2 基于 N-最短路的分词

对于长度为 n 的句子，N-最短路分词首先构造一条有 n+1 个节点，n 条边的链。句子中的每个单字依次构成组成链的边。

然后，枚举句子中所有的起止位置，并在词典中搜索，若起止位置之间的字符串能构成词，则在之前得到的图中新添加一条边，从词语的第一个字的对应边的起始节点出发，指向词语末字的对应边所指向的节点。

图构建完成后，简单起见，为每条边赋权值为 1。随后在图上运行 dijkstra 算法，得到一条最短路径，该路径对应了一种分词结果。

此外本项目中还稍作改进，在 dijkstra 算法中加入随机性，使得结果为多条最短路中的任意一条，不同次分词的结果可能有所不同。

2.2.3 基于逆向最大匹配的分词

运用贪心的思想，从后向前，每次均选择从词典中匹配到的最长的词进行划分，得到分词结果。

2.2.4 算法性能分析

	准确率	召回率	F1	效率 (kb/s)	未登录词召回率	登录词召回率
HMM	80.39	80.56	80.47	207.39	52.35	81.66
N-最短路(使用数据集统计字典)	89.98	93.52	91.72	233.08	2.11	97.12
随机化 N-最短路(使用数据集统计字典)	89.92	93.46	91.66	225.03	2.11	97.05
逆向最大匹配(使用数据集统计字典)	89.88	93.46	91.63	697.26	2.11	97.06

可以看出，在分词模块中，基于字典的分词方法的总体准确率，召回率，以及效率均要高于基于统计的分词方法，其中逆向最大匹配更是凭借简单的算法获得了极高的效率。

然而 HMM 对于未登录词的召回率远高于基于词典的方法，这说明基于字典的方法对于之前没有出现过的新词基本上是无能为力的，而 HMM 却有着相对不错的效果，这便是基于统计的方法的优势。

2.3 词性标注

2.3.1 基于隐马尔可夫的词性标注

大致流程及算法原理与 HMM 分词相同，只是此时处理的最小单位为词汇，标签数也提升为 33 个，对应 33 种不同的词性。

2.3.2 基于条件随机场的词性标注

首先训练 CRF++模型，将训练数据调整为 CRF++的输入格式，每个词占一行，第一列为词汇，第二列为词性，句子之间以空行划分。

取特征模板为：

```
# Unigram
U00:%x[-2,0]
U00:%x[-1,0]
U00:%x[-0,0]
U00:%x[1,0]
U00:%x[2,0]
U00:%x[-1,0]/%x[-0,0]
U00:%x[0,0]/%x[1,0]

# Bigram
B
```

该模板涉及到窗口大小为 5 的空间内的词语对当前位置词性的影响。

命令行训练：crf_learn -f 3 -p 8 -c 3 template.txt crftrain.txt model，只考虑出现次数大于等于 3 的特征，开启 8 线程。

经过 477 轮迭代，模型训练完成，用时 8.29h。

最后在本项目中使用 CRFPP python 接口载入模型参数，进行词性标注工作。

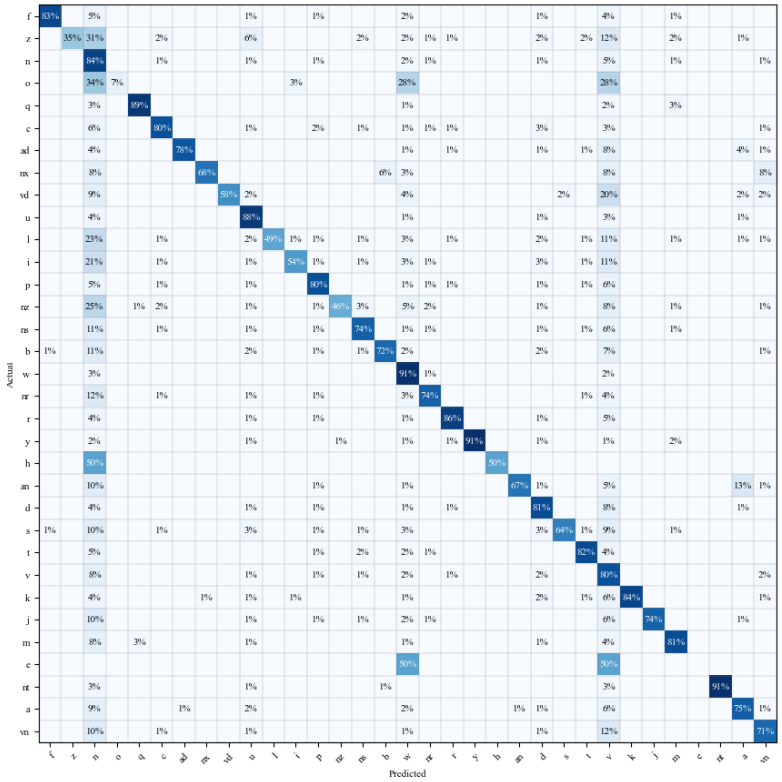
2.3.3 算法性能分析

	总体准确率	效率(kb/s)
CRF++	80.98	263.95
HMM	90.69	10.61

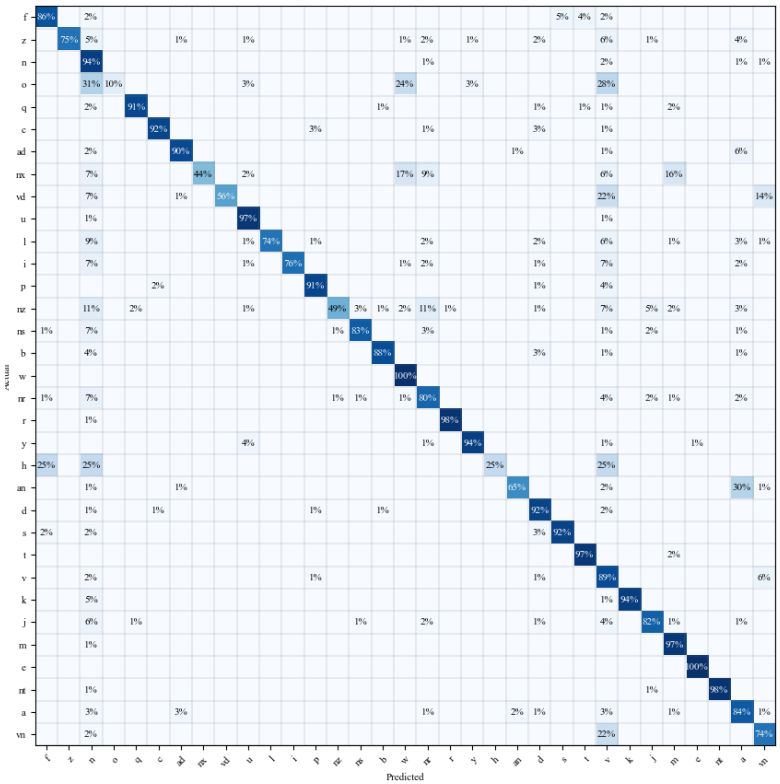
可以看出 HMM 在标注准确率上有一定的优势，但 CRF++由于底层通过 C++实现，效率约为 HMM 的 25 倍。下面分别给出 CRF++和 HMM 标注的混淆矩阵，详细的各词性准确率和召回率将在附录中给出。

如下方混淆矩阵所示，CRF++和 HMM 在标点(w)，语气词(y)，助词(u)，量词(q)等词性较为明确的类别上拥有较好的识别效果(90%左右)。HMM 在对动词(v)，名词(n)，连词(c)的识别效果上较 CRF++有不少的优势(超出 10%左右)。

然而，无论是 CRF++还是 HMM，对于兼性词的识别效果都不够理想，譬如动名词(vn)，副动词(vd)，名形词(an)。这说明以上的两种标注算法对于语义的理解还不够深入，只是掌握了较为浅层的统计规律。



图表 1 CRF++混淆矩阵



图表 2 HMM 混淆矩阵

2.4 命名实体识别

2.4.1 基于隐马尔可夫的命名实体识别

此处基于 HMM 的 NER，同之前的基于 HMM 词性标注几乎一致，只是从预测标签中抽出 nt，nr，ns 作为命名实体的识别结果。

2.4.2 基于 CRF++的命名实体识别

基于条件随机场的 NER，对训练数据进行了重新的预处理，设定 7 个标签：nt-B，nt-I，nr-B，nr-I，ns-B，ns-I，O。其中 O 表示非命名实体，x-x 前面的字符表示实体的类型 nr(人名)，ns(地名)，nt(机构名)，后面表示该词汇在命名实体中的位置，B 代表开头，I 代表之后的位置。

构造特征模板：

```
# Unigram
U01:%x[-2,0]
U02:%x[-1,0]
U03:%x[0,0]
U04:%x[1,0]
U05:%x[2,0]
U06:%x[-2,0]/%x[-1,0]/%x[0,0]
U07:%x[-1,0]/%x[0,0]/%x[1,0]
U08:%x[0,0]/%x[1,0]/%x[2,0]
U09:%x[-1,0]/%x[0,0]
U10:%x[0,0]/%x[1,0]

# Bigram
B
```

该模板描述了在窗口大小为 5 的情况下，相邻的一个词汇，两个词汇，以及三个词汇对当前位置词汇标签的影响。

命令行训练：crf_learn -f 3 -p 16 -c 3 template4.txt crfNERtrain.txt NERmodel3
训练用时 251.53 s

2.4.3 算法性能分析

	人名			地名			机构名		
	准确率	召回率	F1	准确率	召回率	F1	准确率	召回率	F1
CRF++	81.68	63.48	71.44	91.15	64.07	75.25	96.89	56.77	71.59
HMM	59.71	56.07	57.83	90.77	68.08	77.80	96.93	57.55	72.22

可以看出，CRF++在人名实体的识别方面的效果远高于 HMM，其原因在于 HMM 只考虑了 2 元语法，而很多人名为三字实体，而为 CRF++设定的特征模板弥补了 HMM 的不足。

此外 CRF++的运行效率仍远高于 HMM。

三、程序使用说明

3.1 环境配置

numpy 1.18.5
matplotlib 3.2.2
sklearn.metrics
CRFPP

3.1.1 CRFPP 安装

将作业文件夹中的 crfpp_0.58_python_edition.zip 解压，使用命令行窗口进入到 crfpp_0.58_python_edition\crfpp_0.58\python 路径下，依次输入命令 python setup.py build，python setup.py install。最后将当前文件夹中的 libcrfpp.dll 复制到 Anaconda3\Lib\site-packages 路径下(若不使用 Anaconda，请复制到 python 存储外部包的目录下)。

3.2 使用方法

打开项目文件夹，可以看到 demo.ipynb，用 Jupyter Notebook 将其打开或使用 PyCharm 打开整个项目文件夹。demo.ipynb 中有对本项目所有功能的详细展示和介绍，运行之即可。

3.3 API 介绍

3.3.1 分词：

```
def DictSegment(s,mode):
    '''''
    基于词典的分词算法
    :param s: 输入字符串
    :param mode: 模式参数，可选 RMM(逆向最大匹配)，DAG(N-最短路)，RDAG(随机化 N-最短路)
    :return: 分词结果
    '''
```

```
def StatisticSegment(s):
    '''''
    基于统计的分词算法
    :param s: 输入字符串
    :return: 分词结果
    '''
```

```
def BatchSeg(mode,path,outpath=None):
```

```
    '''''
    工具包调用接口
    对整个文件进行分词
    @param path: 原始文本路径
    @param outpath: 分词结果保存位置
    @param mode: 模式参数 可选 HMM,RMM,DAG,SDAG
    @return:
    '''
```

```
def DemoSeg():
    '''''
    分词算法性能测评
    @return:
    '''
```

3.3.2 词性标注

```
def Tagging(s,mode):
    """
    单句标注调用接口
    @param s: 需要进行词性标注的预分词字符串
    @param mode: 模式参数 可选 HMM,CRF
    """
```

```
def BatchTagging(mode,path,outpath=None):
    '''''
    工具包外部调用接口
    对整个预分词文件进行标注
    @param mode: 模式参数 可选 HMM,CRF
    @param path: 预分词文件路径
    @param outpath: 标注结果保存位置
    @return:
    '''
```

```
def PosTagDemo():
    '''''
    词性标注算法性能分析
    @return:
```

```
    '''
```

3.3.3 命名实体识别

```
def NER(s,mode):
    '''''
    工具包的 NER 单句调用接口
    @param s: 预分词字符串
    @return:
    '''
```

```
def BatchNER(mode,path,outpath=None):
    '''''
    对整个文件进行命名实体识别
    :param mode: 模式参数 可选 CRF,HMM
    :param path: 输入的预分词文件所在路径
    :param outpath: 输出的 NER 结果保存路径
    :return:
    '''
```

```
def DemoCRFNER():
    '''''
    CRF NER 性能测评
    :return:
    '''

def DemoHMMNER():
    '''''
    HMM NER 性能测评
    :return:
    '''
```

四、总结与感悟

该项目是对本学期“自然语言理解初步”这门专业选修课中词法分析部分的一次总结与实践。通过完成这个项目，我对知识的理解更近一步，不仅停留于对算法原理的了解，更学会了如何真正地将其实现，在保证算法正确性的情况下亦要考虑算法运行的效率。

在完成项目的过程中我也遇到了不少困难，譬如原始的数据集中存在一些格式错误，导致后期算法运行的过程中发生一些不匹配的现象，对这些错误的debug过程相当考验耐心。

此外，我对本项目的初衷起始是完成一个类似于 jieba 的轻量级词法分析工具包，可无奈自己对于 python 语法中 import 的原理理解不够深入，原始的项目频繁出现头文件引用错误，封装成工具包后载入内置参数时路径报错。由于时间有限，在最后阶段我也只得放弃，将项目重构，成为了现在的样子。这是令我十分遗憾的一点，我也计划在本学期结束后对该项目进行进一步完善，最终实现一个便捷使用，可在其他项目中 import，开箱即用的词法分析工具包。

附录：

词性标注详细准确率和召回率

```
CRFPosTag complete in 0m 3s
CRFPosTag speed is 242.92455882835952 kb/s
The whole precision is 80.9759937484917.
Tag f:
Precision:0.8084808946877913    Recall:0.8277671755725191
Tag z:
Precision:0.8020304568527918    Recall:0.3534675615212528
Tag n:
Precision:0.8035211554939002    Recall:0.8444172883972215
Tag o:
Precision:1.0   Recall:0.06896551724137931
Tag q:
Precision:0.8166318719554627    Recall:0.8880060537268256
Tag c:
Precision:0.7230818826563508    Recall:0.8027916964924839
Tag ad:
Precision:0.7701525054466231    Recall:0.7803532008830022
Tag nx:
Precision:0.9506172839506173    Recall:0.6754385964912281
Tag vd:
Precision:0.8666666666666667    Recall:0.5777777777777777
Tag u:
Precision:0.7976219965320783    Recall:0.8805031446540881
Tag l:
Precision:0.9113300492610837    Recall:0.49377224199288255
Tag i:
Precision:0.9035019455252918    Recall:0.5387470997679814
Tag p:
Precision:0.7469825659365221    Recall:0.8025936599423631
Tag nz:
Precision:0.778175313059034 Recall:0.46178343949044587
Tag ns:
Precision:0.8744302308484047    Recall:0.7446781868269472
Tag b:
Precision:0.7956630525437864    Recall:0.7156789197299325
Tag w:
Precision:0.8525525186356449    Recall:0.9137513618206028
Tag nr:
Precision:0.8442655935613682    Recall:0.7443675714032286
Tag r:
Precision:0.8813381022512098    Recall:0.860340932429657
Tag y:
```

```
Precision:0.9615384615384616    Recall:0.9114583333333334
Tag h:
Precision:1.0   Recall:0.5
Tag an:
Precision:0.8577235772357723    Recall:0.6741214057507987
Tag d:
Precision:0.7547542978852883    Recall:0.8115491575331262
Tag s:
Precision:0.7614285714285715    Recall:0.635280095351609
Tag t:
Precision:0.9038776624795194    Recall:0.8164775530340405
Tag v:
Precision:0.7425574881115237    Recall:0.8013638440718479
Tag k:
Precision:0.7134146341463414    Recall:0.8357142857142857
Tag j:
Precision:0.8768161718256475    Recall:0.7410571276027763
Tag m:
Precision:0.8797307404637248    Recall:0.8136413945766464
Tag e:
Precision:0 Recall:0.0
Tag nt:
Precision:0.9426585577758471    Recall:0.910234899328859
Tag a:
Precision:0.8408001667013961    Recall:0.7501394311210262
Tag vn:
Precision:0.8213403589659147    Recall:0.7138972377272077


HMMPosTag complete in 0m 60s
HMMPosTag speed is 10.442919839785048 kb/s
The whole precision is 90.69283719647433.
Tag f:
Precision:0.8656145384983261    Recall:0.8635496183206107
Tag z:
Precision:0.9436619718309859    Recall:0.7494407158836689
Tag n:
Precision:0.9399225220983242    Recall:0.9415144498756539
Tag o:
Precision:1.0   Recall:0.10344827586206896
Tag q:
Precision:0.9082326283987915    Recall:0.9099508134695422
Tag c:
```

```
Precision:0.911000355998576 Recall:0.9158911954187545
Tag ad:
Precision:0.8009803921568628    Recall:0.9017660044150111
Tag nx:
Precision:0.819672131147541 Recall:0.43859649122807015
Tag vd:
Precision:0.625 Recall:0.5555555555555556
Tag u:
Precision:0.9731138545953361    Recall:0.9699207000273449
Tag l:
Precision:0.9519725557461407    Recall:0.7406583629893239
Tag i:
Precision:0.9975609756097561    Recall:0.7591647331786543
Tag p:
Precision:0.887330528284245 Recall:0.9116234390009607
Tag nz:
Precision:0.7678275290215588    Recall:0.49150743099787686
Tag ns:
Precision:0.978742249778565 Recall:0.8302028549962435
Tag b:
Precision:0.8389166072701354    Recall:0.8829707426856714
Tag w:
Precision:0.9786666666666667    Recall:0.9995763224791188
Tag nr:
Precision:0.8182646737149106    Recall:0.7963455738868193
Tag r:
Precision:0.9699309223892727    Recall:0.9804888067364962
Tag y:
Precision:0.8294930875576036    Recall:0.9375
Tag h:
Precision:0.125 Recall:0.25
Tag an:
Precision:0.7038327526132404    Recall:0.645367412140575
Tag d:
Precision:0.8862923462986199    Recall:0.9244233600523475
Tag s:
Precision:0.8004136504653567    Recall:0.9225268176400476
Tag t:
Precision:0.9560975609756097    Recall:0.9669462259496794
Tag v:
Precision:0.8637981789552954    Recall:0.8870259060072411
Tag k:
Precision:0.8098159509202454    Recall:0.9428571428571428
Tag j:
```

```
Precision:0.7480430528375733    Recall:0.816337426588361
Tag m:
Precision:0.9402122799946259    Recall:0.9681793027116768
Tag e:
Precision:0.6666666666666666    Recall:1.0
Tag nt:
Precision:0.9660314830157415    Recall:0.9781879194630873
Tag a:
Precision:0.812331475822398 Recall:0.8401189812232757
Tag vn:
Precision:0.7103712837374982    Recall:0.7420924574209246
```