自然语言处理导论 期中作业报告

徐晟 1500012780

实验方法：

本次作业给定一些已经分词的训练语料和一个测试数据，需要给出测试数据的中文分词（Chinese word segmentation）结果。

分别实现了一个非结构化感知器和结构化感知器进行分词，代码分别在 nsp.py 文件和 sp.py 文件中。结构化感知器模型的主要步骤即为：随机初始化一个超平面，逐个扫描训练数据，如果预测结果错误，则相应更新模型参数。直到训练完或者达到收敛条件退出。

代码使用方法：

代码运行环境为 python3，均需结合命令行参数使用，具体方法如下：

--train train.txt 指定训练语料
--predict test.txt 指定需要测试的语料
--result output.txt 指定测试语料分词结果的输出文件
--iteration 10 指定训练轮数，默认为 5 轮
--save nsp.model 指定是否保存训练完的模型，需要结合--train 使用
--load nsp.model 指定是否加载某个已经训练完的模型，会覆盖--train 的训练结果

例如：

```
python3 nsp.py --train train.txt --predict test.txt --result out.txt
```

非结构化感知器：

用非结构化感知器切词采用的是 01 分词模型，1 表示这个字需要和前面一个字分开，0 表示不分。考虑到采用的非结构化感知器最后输出的是 1 和-1，为了方便，于是把 01 修改为-1 和 1。

对于该模型，定义了两个辅助函数 **seg_to_sent** 和 **sent_to_seg**，分别实现分词结果（segmentation）和句子（sentence）的 x y 向量的互相转换（x 即是该句子的字符串，y 是每个字符的 1 和-1 的标记结果）。

对于每一个字符 x[i]，提取了 8 个特征，分别是 x[i-1]， x[i]， x[i+1]这三个 unigram 和 x[i-2]x[i-1]， x[i-1]x[i]， x[i]x[i+1]， x[i+1]x[i+2]这四个 bigram，还有 x[i-1]x[i]x[i+1]这个 trigram 如果对于字符不存在，则用字符#代替。为了区分某个特征属于这七个特征中的哪一种，在这些特征之前分别加上了字符 1-8。

实现细节：

由于训练开始之前不知道总共有多少特征，于是先把感知器对每个特征的权重定义为一个空的字典，每次访问时，如果 key 不存在，则默认返回 0，否则返回相应的 key 对于的值。每次训练一个句子，如果感知器输出和标准输出不同，则需要更新感知器各个权重，如果某个 key 从未在感知器中出现过，则先将该 key 对应的 value 初始化为 0，再更新。即，每次加入一个新的特征则在字典里动态加入该特征，不用一开始就生成所有的特征的字典。

一开始的想法是：为了让训练结果尽量稳定，采用了平均感知器。但是每次训练完一个句子，就要累加词典中所有 key-value 对到权重累加词典 `acc`，这样处理太耗时间（之前尝试的时候大概需要六个小时才能训练完一轮）。于是定义了一个新的字典 `last_step`，记录每一个 key 最后一次更新是在训练第几个句子（即是第几步）。在下一次更新前，这个 key 对应的权重值其实一直都没有变过，所以只需要在再次更新该 key 的权重时，检查其上一次更新的 step，同时用两次步数之差乘以该 key 旧的权重值更新 `acc` 词典中该 key 的权重即可。这样可以节省大量时间。

不过实际测试中，平均化感知器效果没有普通的感知器好，于是又用回普通的感知器。猜想原因可能是采用了多轮训练，这样多轮训练完之后得到的参数已经比较稳定，采用平均感知器反而可能积累很多训练前期不太好的参数，导致结果没有普通感知器好。

为了支持把模型保存下来，将所有特征权重组织成一个词典，模型最后训练的结果即为词典的 `key-value` 对。保存或加载模型只需保存或加载 `key-value` 对。分别使用了 `json` 库的 `dump` 和 `load` 函数。

测试结果：

训练 1 轮的结果：

```
=== SUMMARY:
=== TOTAL INSERTIONS:   2772
=== TOTAL DELETIONS:    2978
=== TOTAL SUBSTITUTIONS:        5968
=== TOTAL NCHANGE:      11718
=== TOTAL TRUE WORD COUNT:      106873
=== TOTAL TEST WORD COUNT:      106667
=== TOTAL TRUE WORDS RECALL:    0.916
=== TOTAL TEST WORDS PRECISION: 0.918
=== F MEASURE:  0.917
=== OOV Rate:   0.945
=== OOV Recall Rate:    0.912
=== IV Recall Rate:     0.997
```

训练 5 轮的结果：

```
=== SUMMARY:
=== TOTAL INSERTIONS:   1804
=== TOTAL DELETIONS:    2434
=== TOTAL SUBSTITUTIONS:        4140
=== TOTAL NCHANGE:      8378
=== TOTAL TRUE WORD COUNT:      106873
=== TOTAL TEST WORD COUNT:      106243
=== TOTAL TRUE WORDS RECALL:    0.938
=== TOTAL TEST WORDS PRECISION: 0.944
=== F MEASURE:  0.941
=== OOV Rate:   0.945
=== OOV Recall Rate:    0.935
=== IV Recall Rate:     0.997
```

训练 20 轮的结果：

```
=== SUMMARY:
=== TOTAL INSERTIONS:   1696
=== TOTAL DELETIONS:    2232
=== TOTAL SUBSTITUTIONS:        3833
=== TOTAL NCHANGE:      7761
=== TOTAL TRUE WORD COUNT:      106873
=== TOTAL TEST WORD COUNT:      106337
=== TOTAL TRUE WORDS RECALL:    0.943
=== TOTAL TEST WORDS PRECISION: 0.948
=== F MEASURE:  0.946
=== OOV Rate:   0.945
=== OOV Recall Rate:    0.940
=== IV Recall Rate:     0.997
```

可见，在训练轮数较少时，增加训练轮数可以比较明显地提高准确率和召回率。但当训练轮数逐渐增大之后，增大训练轮数对这些指标的提升程度会变小。

结构化感知器：

结构化感知器采用的是 SBME 模型，S 表示该字符单个字符组成一个词，B 表示字符是一个词的第一个字符，M 表示是一个词中间的字符，E 表示是一个词最后的字符。

和非结构化感知器类似，也定义了两个辅助函数，**seg_to_sent** 和 **sent_to_seg**，功能类似。特征的提取也采用了和非结构化感知器一样的 8 个特征。采用了平均感知器的模型（实际测试中平均感知器和普通的感知器结果差不多）。

为了实现隐式马尔科夫模型的解码问题，实现了维特比算法。大致步骤即是得到所有隐含层状态之间的转移概率，以及隐层状态到显层的生成概率，然后用动态规划的算法计算出使概率最大化的序列。

测试结果：

训练 1 轮的结果：

```
=== SUMMARY:
=== TOTAL INSERTIONS:   1726
=== TOTAL DELETIONS:    2155
=== TOTAL SUBSTITUTIONS:        3978
=== TOTAL NCHANGE:      7859
=== TOTAL TRUE WORD COUNT:      106873
=== TOTAL TEST WORD COUNT:      106444
=== TOTAL TRUE WORDS RECALL:    0.943
=== TOTAL TEST WORDS PRECISION: 0.946
=== F MEASURE:  0.945
=== OOV Rate:   0.945
=== OOV Recall Rate:    0.939
=== IV Recall Rate:     0.998
```

训练 5 轮的结果：

```
=== SUMMARY:
=== TOTAL INSERTIONS:   1337
=== TOTAL DELETIONS:    1828
=== TOTAL SUBSTITUTIONS:        3168
=== TOTAL NCHANGE:      6333
=== TOTAL TRUE WORD COUNT:      106873
=== TOTAL TEST WORD COUNT:      106382
=== TOTAL TRUE WORDS RECALL:    0.953
=== TOTAL TEST WORDS PRECISION: 0.958
=== F MEASURE:  0.955
=== OOV Rate:   0.945
=== OOV Recall Rate:    0.951
=== IV Recall Rate:     0.998
```

训练 20 轮的结果：

```
=== SUMMARY:
=== TOTAL INSERTIONS:   1246
=== TOTAL DELETIONS:    1701
=== TOTAL SUBSTITUTIONS:        2895
=== TOTAL NCHANGE:      5842
=== TOTAL TRUE WORD COUNT:      106873
=== TOTAL TEST WORD COUNT:      106418
=== TOTAL TRUE WORDS RECALL:    0.957
=== TOTAL TEST WORDS PRECISION: 0.961
=== F MEASURE:  0.959
=== OOV Rate:   0.945
=== OOV Recall Rate:    0.955
=== IV Recall Rate:     0.998
```

可见，结构化感知器的正确率和召回率都比非结构化感知器要高（只训练一轮得到的结果就和非结构化感知器得到的结果很接近）。但是加大训练轮数同样对性能提升不明显。

结果汇总：

F 值

	训练 1 轮	训练 5 轮	训练 20 轮
非结构化感知器	0.917	0.941	0.946
结构化感知器	0.945	0.955	0.959

总结与反思：

本次实验中存在一些不足。

特征的提取还存在改进的地方，比如只使用了一个 trigram，这是考虑到内存等因素，所以一开始没有加很多 trigram，但实际运行中对内存的占用也不是很多。

只测试了训练 1 轮、5 轮、20 轮的情况。在 20 轮的时候训练结果可能已经过拟合。

总的来说，本次实验结果还是比较让人满意的，一开始也没有想到这么简单的模型可以得到这样的效果。