自然语言处理导论实验报告

热合玛·阿不力克木 1700017843

实验目标：

实现结构化感知器进行中文分词来完成中文自动分词任务，评估指标是 Precision，Recall，F-score。

实验方法：

结构化感知机模型的主要步骤为，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，直到训练完成。

本实验基于字标注的分词方法，把分词转换为分类过程，将文本语料转化成可用于感知器训练的特征向量，从而进行感知器训练，再用得到的训练模板去对测试文本做预测。

实验环境：

电脑：Windows 10

编译器：pyCharm

环境：python 3.8

实验设置：

该实验使用的是 SBME 模型，S 代表该字符单独成词，B 代表该字是词首，M 代表该字位于词中，E 代表该字为词尾。

特征模板的定义：

特征模板是抽取预料特征的模式，是词特征学习的基础。我的算法使用的特征模板一共有 16 个，如下：

```
def get_features(self, word, word_l1, word_l2, word_r1, word_r2, tag_l1, tag):
    features = ['1' + word,
                '2' + word_l1,
                '3' + word_r1,
                '4' + word_l2 + word_l1,
                '5' + word_l1 + word,
                '6' + word + word_r1,
                '7' + word_r1 + word_r2,
                '8' + word_l1 + word + word_r1,
                '9' + word + tag,
                '11' + tag_l1 + tag,
                '12' + word_l1 + tag,
                '13' + word_l2 + word_l1 + tag,
                '14' + word_r1 + tag,
                '15' + word_l1 + word + tag,
                '16' + word + word_r1 + tag,
                '10' + word_r1 + word_r2 + tag
                ]
    return features
```

word 为当前字，word_l1 为当前字的前一个字，word_l2 为 word_l1 的前一个字，word_r1 为当前字的后一个字，word_r2 为 word_r1 的后一个字，tag 为当前字的标签，tag_l1 为前一个字的标签。为了区别模板，在前面加上了 1~16 的标记。如果某个字对应的某个特征不存在，用'#'代替，这一点在 get_all_features(self, sentence, labels)和 decode(self, sentence)函数中有所体现。

感知器的初始化：

由于实验开始时，不知道总共有多少特征，所以将感知器对每个特征的权重定义为一个空的字典。

```
def __init__(self):

    self.feature_weights = defaultdict(float)
    self.label_type = ['B', 'M', 'E', 'S']
```

感知器的训练：

训练时，首先对每个句子用感知器进行解码预测，代码中用 decode(self, sentence)函数实现。这里运用了维特比算法，求解最优标注结果，这里借鉴了隐马尔科夫模型。运用三个循环，对每一个字，枚举该字的标注（BMES 之一），再枚举前一个字的标注，计算二者状态转移的评分，评分越高越好，从其中选最优路径并记录下来。最后返回总评分最高的一条路径。

```
    for i in range(1, length):  # 每个字
        word_l2 = sentence[i - 2] if i - 2 >= 0 else '#'
        word_l1 = sentence[i - 1] if i - 1 >= 0 else '#'
        word = sentence[i]
        word_r1 = sentence[i + 1] if i + 1 < len(sentence) else '#'
        word_r2 = sentence[i + 2] if i + 2 < len(sentence) else '#'
        for j in range(4):  # 该字的标签

            tag_ = self.label_type[j]

            for k in range(4):  # 前一个字的标签

                tag_l1 = self.label_type[k]
                features = self.get_features(word, word_l1, word_l2, word_r1, word_r2, tag_l1, tag_)

                temp_score = sum(self.feature_weights[mk] for mk in features)
                if temp_score + score[k][i - 1] > score[j][i]:
                    score[j][i] = temp_score + score[k][i - 1]
                    path[j][i] = k
```

接着再分别对预测标注和正确标注进行特征获取，并对感知器进行更新。

```python
        all_features = self.get_all_features(x[j], predict_label)
        all_gold_features = self.get_all_features(x[j], y[j])

        for fid, count in all_gold_features.items():
            self.feature_weights[fid] += count
        for fid, count in all_features.items():
            self.feature_weights[fid] -= count

        correct += sum([1 for (predicted, gold) in zip(predict_label, y[j]) if predicted == gold])
        total += len(y[j])
        if counter % 1000 == 0:
            print(counter)
            print('\tTraining accuracy: %.4f\n\n' % (correct / total))

    weights.update(self.feature_weights)
```

实验步骤：

1. 读入 train 数据，对感知器进行训练。并将训练好的模型保存起来。这里我保存了两组，第一组是迭代 5 次后的模型，第二组是迭代 10 次后的模型。

```python
# train
train_sentences, gold_tags, lines_cnt = pre_pro('train.txt')

# iterations=5
feature_weights = SP.fit(lines_cnt, train_sentences, gold_tags, 5)
# save the model
target_f = open('model3.txt', 'w', encoding='utf-8')
for key in feature_weights.keys():
    target_f.writelines(key + ":" + str(feature_weights[key]) + '\n')
print('write done')
```

运行结果： **下方为训练时的准确率（一次迭代为 7w 行）**

迭代次数为 1：

```
70000
	Training accuracy: 0.9030
```

迭代次数为 5：

```
348000
	Training accuracy: 0.9845


349000
	Training accuracy: 0.9845


350000
	Training accuracy: 0.9846
```

迭代次数为 6：

```
418000
    Training accuracy: 0.9918


419000
    Training accuracy: 0.9919


420000
    Training accuracy: 0.9920
```

迭代次数为 7：

```
488000
    Training accuracy: 0.9947


489000
    Training accuracy: 0.9947


490000
    Training accuracy: 0.9948
```

迭代次数为 8：

```
558000
    Training accuracy: 0.9959


559000
    Training accuracy: 0.9959


560000
    Training accuracy: 0.9960
```

迭代次数为 9：

```
628000
    Training accuracy: 0.9967


629000
    Training accuracy: 0.9967


630000
    Training accuracy: 0.9967
```

迭代次数为 10：

```
699000
    Training accuracy: 0.9971


700000
    Training accuracy: 0.9971
```

由上方运行结果可知，迭代次数到 5 之后，准确率上升的越来越慢，9 次之后基本感觉已经到了稳定状态，很难上升。

2. 用 dev.txt 对模型进行验证。

首先对 dev.txt 进行预处理，去掉分隔符保存成 words.txt 用于脚本测试。

```
words = gen_words('dev.txt')
SP.save(words, 'words.txt')
```

用 5 次迭代的模型进行预测，并将其保存为 predict3.txt 文件。

```
# predict
train_sentences, gold_tags, lines_cnt = pre_pro('dev.txt')
predict_text = SP.predict_(train_sentences)
SP.save(predict_text, 'predict3.txt')
```

用 10 次迭代的模型进行预测，并将其保存为 predict3_1.txt 文件。

```
# predict
train_sentences, gold_tags, lines_cnt = pre_pro('dev.txt')
predict_text = SP.predict_(train_sentences)
SP.save(predict_text, 'predict3_1.txt')
```

用 score 测试脚本去测试，得到上面两个预测结果的准确率分析：

5 次迭代：

F-score = 0.952

Precision = 0.955

Recall = 0.949

OVV Recall = 1.000

IV Recall = 0.949

```
INSERTIONS: 0
DELETIONS:  0
SUBSTITUTIONS:  0
NCHANGE:    0
NTRUTH: 19
NTEST:  19
TRUE WORDS RECALL:  1.000
TEST WORDS PRECISION:   1.000
=== SUMMARY:
=== TOTAL INSERTIONS:   6318
=== TOTAL DELETIONS:    9280
=== TOTAL SUBSTITUTIONS:    14927
=== TOTAL NCHANGE:  30525
=== TOTAL TRUE WORD COUNT:  470304
=== TOTAL TEST WORD COUNT:  467342
=== TOTAL TRUE WORDS RECALL:    0.949
=== TOTAL TEST WORDS PRECISION: 0.955
=== F MEASURE:  0.952
=== OOV Rate:   0.000
=== OOV Recall Rate:    1.000
=== IV Recall Rate: 0.949
### predict3.txt    6318    9280    14927   30525   470304  467342  0.949   0.955   0.952   0.000   1.000   0.949
```

10 次迭代：

F-score = 0.959

Precision = 0.958

Recall = 0.959

OVV Recall = 1.000

IV Recall = 0.959

```
INSERTIONS: 0
DELETIONS:  0
SUBSTITUTIONS:  0
NCHANGE:    0
NTRUTH: 19
NTEST:  19
TRUE WORDS RECALL:  1.000
TEST WORDS PRECISION:   1.000
=== SUMMARY:
=== TOTAL INSERTIONS:   6851
=== TOTAL DELETIONS:    6656
=== TOTAL SUBSTITUTIONS:    12691
=== TOTAL NCHANGE:  26198
=== TOTAL TRUE WORD COUNT:  470304
=== TOTAL TEST WORD COUNT:  470499
=== TOTAL TRUE WORDS RECALL:    0.959
=== TOTAL TEST WORDS PRECISION: 0.958
=== F MEASURE:  0.959
=== OOV Rate:   0.000
=== OOV Recall Rate:    1.000
=== IV Recall Rate: 0.959
### predict3_1.txt  6851    6656    12691   26198   470304  470499  0.959   0.958   0.959   0.000   1.000   0.959
```

由上述结果可知，从 5 到 10 次增加迭代次数虽然正确率有所上升，但并不是很明显。

3. 生成对 test.txt 的分词结果 result.txt。这里使用的是 10 次迭代后生成的模型。

```
# make prediction for test
train_sentences, gold_tags, lines_cnt = pre_pro('test.txt')
predict_text = SP.predict_(train_sentences)
SP.save(predict_text, 'result.txt')
print('predict succeed')
```