自然语言处理导论期中作业实验报告
——基于感知器原理的中文分词工具

信息科学技术学院 1400012776-黎才华 1400012849-李岩昊

一.作业要求简述

实现一个中文自动分词工具。具体来说，根据已有分词结果的训练中文语料，使用感知器模型算法训练一个权重向量并对测试语料进行分词。使用已有的评测工具计算准确率、召回率和F-Score。

二.分词工具实现思路

由于本次作业要求实现的分词工具明确规定了模型必须采用感知器模型算法，因此可供发挥的自由空间不是很充分。在代码撰写前的讨论中，我们曾考虑过最短路径分词和基于点互信息的分词。这些方法都因为与感知器模型的原理不符合而被舍去，例如基于PMI的分词本质上跳过了感知器的训练过程，而认为互信息大于某阈值的两个单字不应该被分开，反之则应该被分开。

因此我们最后采用的算法是老师在课堂上举例讲授的感知器分词算法。首先对训练语料标注标签并提取n-gram特征，将特征标号。然后根据标准感知器算法迭代训练一个感知器，将测试语料的特征放入感知器中计算分数，根据相应的分数给出分词预测序列。最后运行自行编写的shell脚本去调用已有的评测工具给出分词结果评测。

三.源码解释

本次作业使用Java编程实现，详细的代码解释在NLP.java代码的注释当中。此处仅简要介绍关键实现部分。

1. 标签标注

采用B-M-E-S标签进行标注，标注过程发生在读取训练语料时。

```
while((line = bufferedReader.readLine()) != null){
    //append sepTag
    tmpTrainX.append(sepTag);
    tmpTrainY.append(sepTag);
    String[] token = line.split(" ");
    for (int i = 0; i < token.length; ++i){
        int len = token[i].length();
        if (len == 0)
            continue;
        //append the word to trainX
        tmpTrainX.append(token[i]);
        //append sglTag to trainY if it's a single-character word
        if (len == 1)
            tmpTrainY.append(sglTag);
        else{
            // append bgnTag to trainY for its beginning character
            tmpTrainY.append(bgnTag);
            // append bgnTag to trainY for characters within the word
            for (int  j = 1; j < len-1; ++j)
                tmpTrainY.append(midTag);
            // append bgnTag to trainY for its ending character
            tmpTrainY.append(endTag);
        }
    }
    if (Debug >= 2) break;
}
```

2. 特征提取

遍历每一个字符，根据n-gram的定义去提取相关特征。具体实现方式如下，相关数据结构可在NLP.java源码中找到。

```
//else traverse all kind of features
for (int j = 0; j < NofFea; ++j)
{
    StringBuffer buffer = new StringBuffer();
    buffer.append(j);
    int tmp = feature[j].length;
    // extract features and name them
    for (int k = 0; k < tmp; ++k)
    {
        buffer.append('_');
        buffer.append(trainX.charAt(i + feature[j][k]));
    }
    vec[i][j] = new String(buffer);
    hash.put(vec[i][j], new Integer(0));
}
```

3. 训练权重向量

首先，根据B-M-E-S标签的定义，具有词头bgnTag和单字词sglTag的字应当被认为是分词结果的词头文字。为他们赋值1和-1，表明他们应当被感知器分至两侧。之后我们遍历每一个字符，确定在当前权重向量下该字符的分数。大于零的分数被认为是词头字，否则被认为是非词头字。根据分数与训练语料分词结果的相符程度调整权重向量，具体算法与课件中算法一致不再赘述。

```
for (int round = 0; round < NofRound; ++round)
{
    // Printing iteration info
    System.out.println("Iterating: ROUND: " + round);
    int wrongNumber = 0;
    // traverse all characters in the sequence
    for (int i = 0; i < N; ++i)
    {
        char ch = trainY.charAt(i);
        if (ch == sepTag) continue;
        //separate the word if ch is the beginning of the word OR itself forms a word.
        int belong = (ch == bgnTag || ch == sglTag)? 1:-1;
        int score = 0;
        //calc the score
        for (int j = 0; j < NofFea; ++j)
        {
            int index = hash.get(vec[i][j]);
            score += theta[index];
        }
        // see if it is classified correctly
        score = (score > 0)? 1:-1;
        if (score != belong)
        {
            // if not, adjust the vector
            wrongNumber++;
            for (int j = 0; j < NofFea; ++j)
            {
                int index = hash.get(vec[i][j]);
                theta[index] += belong;
            }
        }
    }
    // After a number of iteration, the wrong point percentage will be on a stable stage.
    System.out.printf("Wrong Percentage: %f\n", (double)wrongNumber/N);
}
```

4. 根据权重向量预测分词序列

计算分数方法与训练时完全一样，根据计算的分数判断测试语料中的字是否作为词头字。从而得出测试语料的分词序列，并将结果输出。

```
// see if the word in test.txt is the point to cut
for (int i = 2; i < N-1; ++i)
{
    int score = 0;
    for (int j = 0; j < NofFea; ++j)
    {
        StringBuffer buffer = new StringBuffer();
        buffer.append(j);
        int tmp = feature[j].length;
        //extract features
        for (int k = 0; k < tmp; ++k)
        {
            buffer.append('_');
            buffer.append(testX.charAt(i + feature[j][k]));
        }
        String token = new String(buffer);
        // calc score
        if (hash.containsKey(token))
            score += theta[hash.get(token)];
    }
    // predict. if score > 0 then append space in front of it, which means it should be separated
    if (score > 0)
        result.append("  ");
    result.append(testX.charAt(i));
}
```

5. 自动分词脚本

简单的shell脚本，运行即可得到分词结果和评测分数

```
#!/bin/bash
javac NLP.java
java NLP
perl score train.txt test.answer.txt out.txt > result.txt
```

四. 实验步骤及优化策略调整

实验步骤如下：

1. 读取train.txt。根据训练语料给出的分词标注B-M-E-S标签。
2. 为训练语料中的每个字符确定其n-gram特征向量。并为特征向量标号。
3. 迭代训练感知器。为训练语料中的每个字符计算分数，若分数正负与分词选择不符，则根据感知器算法调整权重向量。迭代次数取150次。将训练好的权重向量存入log.txt中。
4. 为测试语料中的每个字符确定其特征向量，根据训练好的权重向量计算测试语料中每个字符的分数，从而预测其是否作为一个词的开头。
5. 将分词结果输出到result.txt，调用perl脚本进行评测

优化策略调整：

1. 首次实现时未加入Trigram特征只有Unigram&Bigram，加入Trigram特征后分数从92.9提升至93.2。
2. 调整迭代次数，仍旧无法提高分类正确率，分数无提升。

3. 采取平均权重向量，由于迭代次数过多时震荡幅度极小，得到的平均权重向量与标准权重向量几乎无差别，分数无提升。

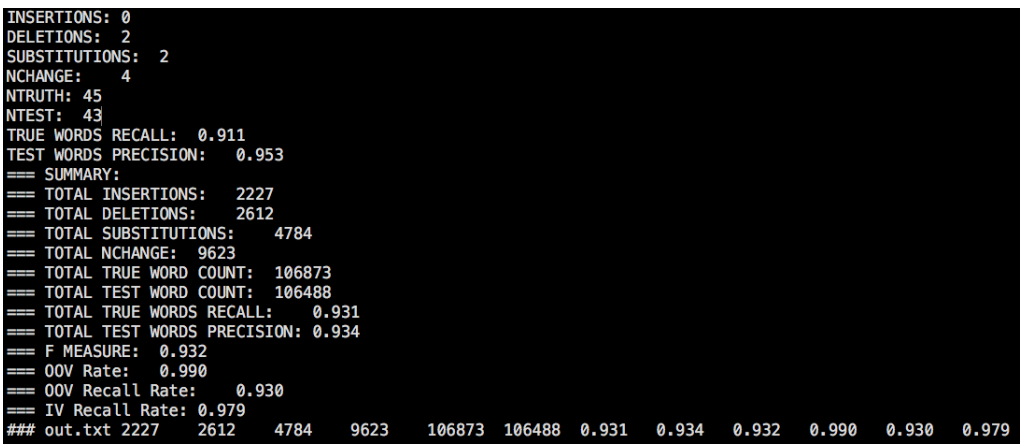
对于结果的思考：
我们未设置收敛阈值而采用简单直接的迭代次数是因为感知器的线性可分类假设太强，在结果中可以显著观察到未被正确分类的点的比例稳定在0.7%左右。而这样的理论缺陷也直接导致了我们分类正确率无法进一步的得到显著提高。
由于迭代次数设置为150，但从20次迭代过后训练语料正确分类比例已经稳定在99.2%-99.3%左右，因此平均感知器求得的权重向量和标准感知器求得的权重向量无显著区别。因此在最终版程序中为了节省内存直接采用了标准感知器算法。

下面给出迭代时错误分类点比例的震荡情况加以说明（Shell中标准输出的迭代信息）：
Iterating: ROUND: 108
Wrong Percentage: 0.007847
Iterating: ROUND: 109
Wrong Percentage: 0.007830
Iterating: ROUND: 110
Wrong Percentage: 0.007866
Iterating: ROUND: 111
Wrong Percentage: 0.007874
Iterating: ROUND: 112
Wrong Percentage: 0.007843
Iterating: ROUND: 113
Wrong Percentage: 0.007846
Iterating: ROUND: 114
Wrong Percentage: 0.007854
Iterating: ROUND: 115
Wrong Percentage: 0.007850
Iterating: ROUND: 116
Wrong Percentage: 0.007854

五.测试结果及组员分工

在Linux环境下运行作业文件夹中的autotest.sh脚本文件，测试结果将会输出到result.txt中。

最终采取的策略为——抽取的特征分别为unigram、bigram、trigram，采用BEMS标签，运用标准感知器算法求权重向量。**测试结果截图如下——**



文字版本如下——

INSERTIONS: 0
DELETIONS: 2
SUBSTITUTIONS: 2
NCHANGE: 4
NTRUTH: 45
NTEST: 43
TRUE WORDS RECALL: 0.911
TEST WORDS PRECISION: 0.953
=== SUMMARY:
=== TOTAL INSERTIONS: 2227
=== TOTAL DELETIONS: 2612
=== TOTAL SUBSTITUTIONS: 4784
=== TOTAL NCHANGE: 9623
=== TOTAL TRUE WORD COUNT: 106873
=== TOTAL TEST WORD COUNT: 106488
=== TOTAL TRUE WORDS RECALL: 0.931
=== TOTAL TEST WORDS PRECISION: 0.934
=== F MEASURE: 0.932
=== OOV Rate: 0.990
=== OOV Recall Rate: 0.930
=== IV Recall Rate: 0.979
out.txt 2227 2612 4784 9623 106873 106488 0.931 0.934 0.932 0.990 0.930 0.979

组员分工（具体代码实现部分在NLP.java中有注释标明作者）：

黎才华——算法设计讨论，感知器模型算法，预测分词算法

李岩昊——算法设计讨论，原始数据处理及自动评测脚本，优化策略实验，报告撰写