自然语言理解

基于 CRF 的汉语自动分词系统

姓名: **燕东**
学号:201428015059074
培养单位: 软件研究所

2015 年 5 月 31 日

目录

1 项目分工与任务安排

本次项目由我一人完成，在这个项目中我首先掌握了利用 CRF（条件随机场）进行汉语分词的基本原理，然后对 CRF 进行了一些初步的实现。在这个项目中我实现了对语料特征的提取、参数的训练，以及分词的预测。在实现过程中遇到了一些的困难，比如说参数的训练这一块，我利用的是最大似然模型，然后初始化参数，利用拟牛顿法进行最优解的选取，后来发现当语料规模变大的时候，我的方程是无解的，苦思无解，于是我采用了折中的办法，利用当前比较流行的开源工具 CRF++ 进行训练，然后我提取它的参数，除此之外，都是我自己的思路。

2 国内外的相关工作

中文分词是中文信息处理中的一个关键的环节，国内外从上世纪 80 年代甚至更早就开始了研究，提出了很多的解决方法。目前的解决方法主要分为以下的几类。

2.1 基于词典的分词方法

这个方法是通过人工构造词典的方式进行分词，主要的方法有最大匹配法（包括正向最大匹配法、逆向最大匹配法、双向最大匹配法）：这种方法的实现简单，但是很难处理歧义的问题；最短路径法：这种方法设想求得最小的分词数来进行分词。

2.2 基于统计的分词方法

随着统计机器学习的发展，国内外研究人员提出了利用机器学习中文信息规则的方式，主要的方法有 N 元语言模型，这个是自然语言处理的基础、隐形马尔科夫链，可以有效的进行分词和词性标注、最大熵模型是在已知信息的条件下求得熵最大的概率分布模型；条件随机场，一般使用的是线性链条件随机场。

3 条件随机场与中文分词

3.1 条件随机场

以字构词的分词方法可以由线性链条件随机场解决。即每个汉字有几种特定的标记，比如 B（代表词的开头），M（代表词中），E（代表词尾），S（代表单独成词）。

首先定义 $f_k(y_{i-1}, y_i, x, i)$，它是针对语料库定义的特征函数，算法就是通过提取语料库这些特征，然后进行训练后得到模型，然后再根据模型进行预测。在定义 $f_k(y_{i-1}, y_i, x, i)$ 之前首先要定义一个实值函数 [1]

$$B(x,i) = \begin{cases} 1, & x\text{具有某个特征} \\ 0, & \text{其他} \end{cases}$$

这个 $B(x,i)$ 是一个二值函数，那么同样可以得到 $f_k(y_{i-1},y_i,x,i)$ 的定义：

$$f_k(y_{i-1},y_i,x,i)=\begin{cases}B(x,i), & y_{i-1}\text{是某个特定的标记,}y_i\text{是某个特别的标记}\\0, & \text{其他}\end{cases}$$

上面的定义是比较的正规的定义，在实际的处理过程中，我把这两者结合了起来，即定义的特征函数 $f_k(y_{i-1},y_i,x,i)$ 只保留使它不为 0 的那些取值，并且省略了 $B(x,i)$ 的定义，$f_k(y_{i-1},y_i,x,i)$ 新的定义是：

$$f_k(y_{i-1},y_i,x,i)=\begin{cases}1, & y_{i-1}\text{是某个特定的标记,}y_i\text{是某个特别的标记,}x\text{在 i 处具有某种特征}\\0, & \text{其他}\end{cases}$$

这样在计算机中存储模型的时候会节省很多的空间。

最后建立出来的线性链条件随机场的模型是

$$P_w(y|x)=\frac{exp\left(\sum_{i=1}^{n}w_if_i(x,y)\right)}{\sum_y exp\left(\sum_{i=1}^{n}w_if_i(x,y)\right)} \tag{1}$$

学习的优化目标函数是

$$\min_{\omega\in\mathbf{R}^n} f(\omega)=\sum_x \widetilde{P}(x)\log\sum_y exp\left(\sum_{i=1}^{n}\omega_if_i(x,y)\right)-\sum_{x,y}\widetilde{P}(x,y)\sum_{i=1}^{n}w_if_i(x,y) \tag{2}$$

其中 $f_k(y,x)=\sum_{i=1}^{n}f_k(y_{i-1},y_i,x,i),\qquad k=1,2,\cdots,K$

优化的参数是 ω，其中 ω 是 $f_k(y_{i-1},y_i,x,i)$ 对应的参数，求得这个参数后，就可以根据输入的测试语料从中选择最优的分类，然后根据分类输出分词的结果。

4 思路和实现方法

利用 CRF 进行中文分词，需要特征的提取、特征的训练以及模型的预测等步骤。我理解的条件随机场进行中文分词就是从原始的语料中提取出字的特征，这些特征的选择非常灵活，然后针对建立的模型，对公式 (2)进行训练得到优化后的参数值，设为 ω。然后进行模型预测的时候，先按照和原始语料相同的办法提取特征，并利用这些特征从不同的标记序列中选择出可能性最大的标记序列，然后根据这些标记序列进行一些处理后就可以输出分词的结果。所以可以很明显的看出条件随机场是判别式模型，他是根据当前得到的特征直接对模型进行预测。下面介绍主要的实现步骤。

4.1 特征的提取

4.1.1 特征模板的定义

CRF 对特征的选择非常的灵活，比如说可以定义当前字的前一个字，前两个字，后一个字，后两个字等。CRF 模型中定义的特征有两类 Unigram 和 Bigram，Unigram 之和当前的标记有关，Bigram 不仅和当前的标记有关，而且还和前一个字的标记有关。为了实现灵活的特征选取，首先就要定义特征模板，特征模板的定义我参照的是 CRF++ 的一个定义方式，特征定义类似于以下的情形：

```
# feature model
# Unigram
U00:%x[0,0]
U01:%x[-1,0]/%x[0,0]/
U02:%x[1,0]
# Bigram
B
```

其中 Unigram 定义的是提取 Unigram 的特征，Bigram 提取的是 Bigram 的特征，每个特征的标记中%x[a,b] 代表从相对于当前的第 a 行中取出第 b 列作为该字在该位置处的特征，a 是一个相对量，b 是一个绝对量。Bigram 中的 B 代表只处理标记之间的转换的可能性。在实际的实现中，我假设 Unigram 和 Bigram 的区别只是是否考虑前一个字的标记状态，所以这两者可以合在一起考虑，并且我还得到，只利用 Unigram 并且选择合适的特征，也可以达到很好的分类结果。

4.1.2 特征的提取

我利用的语料库是北大的人民日报语料库，所以之前需要对语料库进行预处理，转换成有标记的情形，并且处于简单考虑，没有考虑时间、标点以及专有名词等。提取的时候，针对某一个词去除它的词性标记，如果词为单字词标记为 S，如果为字首标记为 B，字中标记为 M，字尾标记为 E。并且分行进行存储，每个行之间有一个区分标记，在此设置的是 <BOS><EOS> 和空行的方式（为 CRF++ 做了适配）。

转化完成后根据定义好的特征模板读取转化后的语料的特征，在此我对特征进行了编码，每一个特征对应一个整数值，并且由专门的一个类来进行特征值的分配和管理，对应特征值我存储了三个文件，feature_ dict txt、feature_ learned.txt 和 total_ feature_ with_ tags.txt。这三个文件存储的就是从语料库中提取到的特征信息，之后的训练算法就是根据后两个文件进行的学习。feature_ dict.txt 存储的是对特征值和特征编码的映射关系，feature_ learned.txt 存储的是每一行所提取到的特征，其中首先是每一行没有标记的特征，然后是添加了标记后的特征。对添加了标记的特征表示的方法是 tag_ coding，其中 tag 对应的标记的编码，coding 对应的是上述特征的编码。total_ feature_ with_ tags.txt 存储的是有标签特征的编号，此处是为了进行预测的时候能够保证编号的唯一。

在特征的提取中遇到了很多的问题，经过很多次的调试才得以成功，对于那些提取相邻行的特征和未来行的特征，我采用了队列的方式进行存储，因为队列是先进先出，并且为了节省空间，我实现的是具有固定容量的 queue，这个设定基于的是一般一个字的特征相关性只由其相近的字的特征描述。所以对应于历史，我建立了 history_ queue，这个历史当每次处理完一个之后，将此字入队列，当遇到

下一行时队列清空;future_ queue，当每次一行开始后进行预读，每次处理一个字的时候，先进行出队列的操作，然后再进行预读。

4.2 模型的学习

4.2.1 拟牛顿法

拟牛顿方法是用矩阵 B_k 来对 $\nabla^2 f(x_k)$ 进行近似，并且保证 B_k 是正定的，这样就能保证下降方向，且 B_k 的计算比较容易，拟牛顿方法的收敛速度是超线性的。B_k 在构造中需要满足一定的条件。

由 $f(x_k + \alpha_k p_k)$ 在 x_k 处的展开式 $f(x_k + \alpha_k p_k) = f(x_k) + \alpha_k p_k \nabla f(x_k) + \mathbf{O}(\alpha_k p_k)$ 又由 $x_{k+1} = x_k + \alpha_k p_k \Rightarrow \alpha_k p_k = x_{k+1} - x_k$ 所以可以得到:

$$\nabla^2 f(x_{k+1})\,(x_{k+1} - x_k) \;=\; \nabla f(x_{k+1}) \;-\; \nabla f(x_k) \tag{3}$$

所以 B_k 理想的取法就是满足:

$$B_{k+1}\, s_k = y_k \tag{4}$$

$$s_k \;=\; x_{k+1} - x_k \qquad y_k \;=\; \nabla f(x_{k+1}) - \nabla f(x_k) \tag{5}$$

上面的条件被称作 **secant equation**，常用的 B_k 的生成方法为 BFGS 准则，它的公式是：

$$B_{k+1} \;=\; B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \;+\; \frac{y_k y_k^T}{y_k^T s_k} \tag{6}$$

实际在计算的时候要先计算除 B_k 然后在计算它的逆，所以可以直接计算出 B_k^{-1}:

$$H_{k+1} \;=\; (I - \rho_k s_k y_k^T)^T H_k (I - \rho_k s_k y_k^T) + \rho_k s_k s_k^T \tag{7}$$

$$\rho_k = \frac{1}{y_k^T s_k} \tag{8}$$

4.2.2 L-BFGS 准则

当前需要求解的问题是数据量非常大的问题，当问题规模变的很大的时候，近似矩阵的计算和存储也会占用很大的开销，所以要使用 L-BFGS 选择搜索方向。

L-BFGS 算法的思想是用最近几次迭代时的信息来计算海森矩阵的近似，因为当前计算的近似值可能只是和最近的迭代信息有关，所以要存储最近 m 次 (m 取 3-20 会有比较好的效果) [2] 迭代中的 $\{s_i, y_i\}$ 的值，根据这些历史数据，计算当前的 $H_k \nabla f(x_k)$，计算完成后，移出进入最早的那一对 $\{s_i, y_i\}$，并将 $\{s_k, y_k\}$ 移入。其中 H_k 的计算可由下面推导得出：

可以对公式(7)做如下的变形：

$$H_{k+1} \;=\; V_k^T H_k V_k + \rho_k s_k s_k^T \tag{9}$$

$$V_k = I - \rho_k s_k y_k^T \tag{10}$$

那么可以得出递推公式： [3]

$$\begin{aligned} H_{k+1} &= V_k^T(V_{k-1}^T H_{k-1} V_{k-1} + \rho_{k-1} s_{k-1} s_{k-1}^T) V_k + \rho_k s_k s_k^T \\ &= V_k^T V_{k-1}^T H_{k-1} V_{k-1} V_k + V_k^T \rho_{k-1} s_{k-1} s_{k-1}^T V_k + \rho_k s_k s_k^T \\ &= \cdots\cdots \\ &= V_k^T V_{k-1}^T \cdots V_{k-m}^T H_k^0 V_{k-m} \cdots V_{k-1} V_k \\ &\quad + V_{k-1}^T \cdots V_{k-m+1}^T \rho_{k-m} s_{k-m} s_{k-m}^T V_{k-m+1} \cdots V_{k-1} \\ &\quad + V_{k-1}^T \cdots V_{k-m+2}^T \rho_{k-m+1} s_{k-m+1} s_{k-m+1}^T V_{k-m+2} \cdots V_{k-1} \\ &\quad + \cdots \\ &\quad + \rho_k s_k s_k^T \end{aligned} \tag{11}$$

可以用下面的算法高效的计算 $H_k \nabla f(x_k)$ 在实际中使用的 L-BFGS 算法的形式如下:

Algorithm 1 Computing L-BFGS $H_k \nabla f(x_k)$

$q \leftarrow \nabla f(x_k)$
for $i = k-1, k-2, \cdots, k-m$ **do**
    $\alpha_i \leftarrow \rho_i s_i^T q$
    $q \leftarrow q - \alpha_i y_i$
end for
$\gamma \leftarrow H_k^0 q$
for $i = k-m, k-m+1, \cdots, k-1$ **do**
    $\beta \leftarrow \rho_i y_i^T \gamma$
    $\gamma \leftarrow \gamma + s_i(\alpha_i - \beta)$
end for
return γ

上面的算法中预设了搜索步长是 1，因为大多数情况下都满足条件，这是牛顿方法的一个优点，只有当不满足条件的时候才会重新计算步长。

4.2.3 具体的实现

目前程序中我采用了两种方式实现，第一种就是自己编写程序实现了训练算法，但是效果不佳，尤其是会在一定情况下优化函数出现无界的情况，所以我就采用了 CRF++ 进行训练，之后我提取出 CRF++ 生成的模型文件中的特征对应的参数值，并且根绝我之前提取出的特征值进行编码，编码完成后我将自己的模型重新导出到模型文件。第一种算法的实现：已知目标函数是公式(2), 经过转化就可以得到下面的公式： [2]

$$L(\omega) = \sum_{(x,y)} \sum_{i=1}^{N} \sum_{k=1}^{K} \omega_k f_k(x_i, y_{i-1}, y_i) - \sum_{(x,y)} \log Z(x) - \sum_{k=1}^{K} \frac{{\omega_k}^2}{2\delta^2} \tag{12}$$

前面的都好计算，我在提取特征值的时候，就已经将每一句中的特征出现次数统计出来了，所以只需要对这些次数进行相加即可，然后对于 $Z(x)$ 这是一个归一化变量，计算的是针对输入 x，它所对应的

Algorithm 2 L-BFGS

Require: ω_0, m

1: $k \leftarrow 0$
2: **repeat**
3: calculate ∇L_k
4: **if** a_{k-1} satisfy Wolfe condition **then**
5: discard the earliest yi ans si
6: $s_k = \omega_k - \omega_{k-1}$
7: $y_k = \nabla L_k - \nabla L_{k-1}$
8: store s_k y_k
9: **else**
10: calculate a_{k-1} which satisfy the Wolfe condition
11: $\omega_k \leftarrow \omega_{k-1} + a_{k-1}p_{k-1}$
12: goto 4
13: **end if**
14: caculate p_k use algorithm 1
15: $a_k \leftarrow 1$
16: $\omega_{k+1} \leftarrow \omega_k + a_k p_k$
17: $k \leftarrow k + 1$
18: **until** Convergence

所有标记可能性的总和。所以在这里采用动态规划的思想进行求解 $Z(x)$，定义递推变量：

$$\vec{\alpha_i}^T = \begin{cases} \vec{\alpha_{i-1}}^T \Psi_i, & 1 \le i \le n \\ \vec{1}^T, & i = 0 \end{cases}$$

$$\Psi_[i,j] = exp(\sum_{k=1}^{K} \lambda_k f_k(\vec{x}, i, s_i, s_j)) \tag{13}$$

其中 Ψ_i 是 x 在 i 的位置处不同的标签之间相互转化的概率，则根据每一步中的 x 首先计算 Ψ_i，然后就可以把 α_i 计算出来，知道了目标函数，可以调用 l-bfgs 求解器，来求解最优值。

第二种算法的实现，在 crf++ 中的 crf_learn 程序如果添加 -t 参数的话，它会输出编码前特征的概率值，由于，我在前面所取得的特征和 crf++ 定义的特征是一样的，所以我只要分析这个模型文件，先从中搜索制定的特征，然后就可以找到它的概率值，而这个概率值是我需要的，只需要复制过来即可。crf++ Model 文件的格式为：

参数说明
....
标签列表
....
特征模板
....
提取出的特征（格式为 id:feature）
....
每个特征对应的概率，以 id 顺序排列

....

其 id 是每隔 4 个 id 存储的，也就是间隔了标签的顺序，所以我只要找到这个特征，得到它的 id，就可以顺着找到它对应的 unigram 的概率值。

最后我把训练的结果存为了 model.txt，文件的格式为:feature:probability

4.3 模型的预测

模型的预测我采用的是维特比算法，首先根据输入的文字按行为单位进行分解，每一行为一个处理单位。针对每一行，定义变量：

$$\delta_i(j) = \max_{1 \le k \le 4} \delta_{i-1}(k) + exp(\omega_j f(x, i, s_k, s_j)) \tag{14}$$

$$\Phi_i(j) = argmax_(\delta_{i-1}(k) + exp(\omega_j f(x, i, s_k, s_j)) \tag{15}$$

通过搜索 Φ 就可以得到最优的标签序列

5 实验结果

特征模板

模板	意义
U00:%x[0,0]	当前字
U01:%x[-1,0]/%x[0,0]/	上一个字/当前字/
U02:%x[1,0	下一个字
U03:%x[0,0]/%x[1,0]/	当前字/下一个字/

实验结果采用了北京大学的人民日报切分的语料库 [4]

指标	数量
训练句子数量	16000
预测句子数量	6722
正确率	68%
召回率	76%
F-measure	72%

特征模板

实验结果采用了北京大学的人民日报切分的语料库 [4]

模板	意义
U00:%x[0,0]	当前字
U01:%x[-1,0]/%x[0,0]/	上一个字/当前字/
U02:%x[1,0	下一个字
U03:%x[0,0]/%x[1,0]/	当前字/下一个字/
U02:%x[-1,0]/%x[0,0]/%x[1,0]/	前一个字/当前字/后一个字/

指标	数量
训练句子数量	16000
预测句子数量	6722
正确率	66.6%
召回率	79.5%
F-measure	72.5%

参考文献

[1] 李航, *统计学习方法*, 清华大学出版社, 2012.

[2] 陈天缘, “线性链条件随机场训练算法优化的研究,” M.S. thesis, 复旦大学, 2010.

[3] miaoweiletter, “拟牛顿法及其相关解法,” http://blog.pfan.cn/miaowei/52925.html, 2011.

[4] 北京大学, “人民日报语料库,” .