大作业：Twitter数据情态分析

钱庭涵
10152130122
上海市普陀区中山北路 3663 号华东师范大学
10152130122@stu.ecnu.edu.cn

概要

本次实验针对Twitter数据进行情态分析，根据提供的训练集（ID、标签、数据文本），采取合适的算法处理数据，去除噪音，提取特征，对测试集数据进行情感分类（三分类），为防止训练集过拟合，使用验证集进行验证。在本报告中，采取的分类算法为朴素贝叶斯，并辅以其他符号处理来进行数据清洗，最后结果是训练集准确率为66.52%，验证集准确率为 47.94%，训练集F-score为 98.00%，验证集F-score为 71.97%。

1 介绍

1.1 背景

现在自然语言处理（NLP）、机器学习（ML）等领域可谓是火爆异常，而随着社交网络的爆发式发展，自然也有很多研究者对社交媒体上的文本分析产生了兴趣。中文的语法十分繁杂，构成文字的结构与种类也繁多，英文则相对简单，只由 26 个英文字母构成所有表达词义的单词。在以英文为主的社交媒体中，Twitter可谓是使用最多最广泛了，本次实验就采用Twitter上的数据文本进行分析。

1.2 目的

本次实验的目的是对Twitter上的数据文本进行情态分析，处理数据，去除噪音，提取特征，使用合适的算法将文本段分为positive（积极的）、negative（消极的）、neutral（中性的）三类。

2 相关工作

2.1 实验要求

需要自己处理数据，去除噪音，并且自己提取特征，用学过的分类算法或者自己学过的其他算法进行情感分类。

2.2 提供数据

dazuoye_train-A.txt
dazuoye_dev-A.txt
dazuoye_test-A.txt

dazuoye_train-A.txt
第一列表示句子的唯一标识，即ID，第二列表示句子的情感标签，一共有三种标签，分别是positive、negative和 neutral，接下来是Twitter数据文本。

dazuoye_dev-A.txt
数据格式与训练集相同，验证集的作用是防止模型过拟合，对于训练集训练好的模型，应用验证集先验证一下效果。

dazuoye_test-A.txt
第一列表示句子的唯一标识，即ID，接下来是Twitter数据文本，需要根据这些文本对句子进行分类。

2.3 结果格式

句子的唯一标识+句子的情感标签，标识和标签之间按“空格”隔开。
比如：264238274963451904 negative

3 方法

3.1 数据文本分析

在搜索观察Twitter数据文本，我发现数据格式十分混乱，格式种类过多，以下列举的是发现的数据问题：

- 句子ID与标签之间、标签与接下来的文本之间，间隔它们的是一个退格键’\t’，而不是空格，这个在存入list中时需要注意。
- 单词的大小写格式不统一，在后续分类上没法归为一类。
- 出现’\u2019’和’\u002c’这样的unicode编码上的问题。
- 各种符号跟单词文本连在一起，很难区分开符号和单词，在后续分类上没法将这样的单词归为一类。
- 一些比较重要的带有情感色彩的符号，单个符号的有’?’和’!’，还有各种类型的颜文字，比如’:)’、’:-)’。
- 两个单词之间的连接符’-’，使得这两个单词没法归到该有的类别中去。
- 中括号’[’和’]’，小括号’(’和’)’，尖括号’<’和’>’，冒号’:’，分号’;’，句号’.’，逗号’,’，斜杠’/’和’\’，星

号’*’，双引号’”’，单引号’’’，等号’=’这些符号，连接在句子或单词的头和尾，存在对分类番干扰。

- 单词简写上的连接符，比如’I’m’中的单引号’’’干扰了被连接的两个单词的归类。
- Twitter上常出现的’@’用于提醒想要提醒的用户，’@’后面往往跟的是那个用户的昵称或ID，这与情态分析无关，应该去除。
- Twitter上出现的连接符’&’，当它将两个单词连接在一起时，会干扰这两个单词的归类，还有当它被编码成’&amp’时，也应该去除。
- Twitter上的话题会用’#’开头来表示，话题与情态分析无关，应该去除。
- Twitter上以’RT’开头的表示转发，这个’RT’是无关特征，应该去除。
- 以’http://’和’https://’开头的表示网址链接，这些可能也与情态分析无关，需要注意。
- 数据文本中可能出现邮箱地址，邮箱地址与情态分析无关，应该去除。

最后，经过数据清洗之后希望能得到一个纯净的文本，便于分类，但要注意有些符号去除之后反而会使情态分析更不准确，哪些符号应该去除，哪些符号应该分隔开，需要反复尝试实验。

3.2 分类算法——朴素贝叶斯

需要注意：本次实验需要用到的是三分类，因此需在二分类朴素贝叶斯算法上稍作修改。

贝叶斯定理

假设对于某个数据集，随机变量C表示样本为C类的概率，F1 表示测试样本某特征出现的概率，套用基本贝叶斯公式，则如下所示：

$$P(C \mid F_1) = \frac{P(CF_1)}{P(F_1)} = \frac{P(C) \cdot P(F_1 \mid C)}{P(F_1)}$$

上式表示对于某个样本，特征F1 出现时，该样本被分为C类的条件概率。那么如何用上式来对测试样本分类呢？

举例来说，有个测试样本，其特征F1 出现了（F1=1），那么就计算P(C=0|F1=1)和P(C=1|F1=1)的概率值。前者大，则该样本被认为是 0 类；后者大，则分为 1 类。

对该公示，有几个概念需要熟知：

先验概率（Prior）。P(C)是C的先验概率，可以从已有的训练集中计算分为C类的样本占所有样本的比重得出。

证据（Evidence）。即上式P(F1)，表示对于某测试样本，特征F1 出现的概率。同样可以从训练集中F1 特征对应样本所占总样本的比例得出。

似然（likelihood）。即上式P(F1|C)，表示如果知道一个样本分为C类，那么他的特征为F1 的概率是多少。

对于多个特征而言，贝叶斯公式可以扩展如下：

$$P(C \mid F_1F_2 \cdots F_n) = \frac{P(C) \cdot P(F_1F_2 \cdots F_n \mid C)}{P(F_1F_2 \cdots F_n)}$$

$$= \frac{P(C) \cdot P(F_1 \mid C) \cdot P(F_2 \cdots F_n \mid CF_1)}{P(F_1F_2 \cdots F_n)}$$

$$= \cdots$$

$$= \frac{P(C) \cdot P(F_1 \mid C) \cdot P(F_2 \mid CF_1) \cdots P(F_n \mid CF_1 \cdots F_{n-1})}{P(F_1F_2 \cdots F_n)}$$

分子中存在一大串似然值。当特征很多的时候，这些似然值的计算是极其痛苦的。现在该怎么办？

朴素的概念

为了简化计算，朴素贝叶斯算法做了一假设：“朴素的认为各个特征相互独立”。这么一来，上式的分子就简化成了：

P(C)P(F1|C)P(F2|C)...P(Fn|C)。

这样简化过后，计算起来就方便多了。

这个假设是认为各个特征之间是独立的，看上去确实是个很不科学的假设。因为很多情况下，各个特征之间是紧密联系的。然而在朴素贝叶斯的大量应用实践实际表明其工作的相当好。

其次，由于朴素贝叶斯的工作原理是计算P(C=0|F1...Fn)和P(C=1|F1...Fn)，并取最大值的那个作为其分类。而二者的分母是一模一样的。因此，我们又可以省略分母计算，从而进一步简化计算过程。

另外，贝叶斯公式推导能够成立有个重要前期，就是各个证据（evidence）不能为 0。也即对于任意特征Fx，P(Fx)不能为 0。而显示某些特征未出现在测试集中的情况是可以发生的。因此实现上通常要做一些小的处理，例如把所有计数进行+1（加法平滑 additive smoothing，又叫拉普拉斯平滑 Laplace smothing）。而如果通过增加一个大于 0 的可调参数 alpha 进行平滑，就叫 Lidstone 平滑。

3.3 去除并分隔单词的方法——调用re模块

string类型的split()方法只能实现针对一个字符进行字符串的分隔，想要使用针对多个字符进行字符串的分隔，需要调用re模块，返回分隔后的字符串列表。

调用re模块的split()方法，如：

```
temp2 = re.split('[][():;&/.,*"=< \t\\\]+', temp1)
```

3.4 替换字符/字符串的方法——replace()

- 调用re模块的split()方法，对某些字符进行处理的话，可能会破坏一些重要的特征（如颜文字）。
- 有时候需要去除一个字符但又不能使这个单词被分隔开。
- 有时候只是想要将这个字符的前段字符串分隔开，如将’about@abc’分成’about’和’@abc’。
- ……

像以上这些情况，就可以使用string类型的replace()方法，直接替换，调用后返回替换后的新字符串（若没有替换，则返回原字符串），比如对于颜文字，可以将这个颜文字替换成一个固定的单词：

```
temp1 = temp1.replace(':)', ' emotion1 ')
```

比如去除一个字符：

```
temp1 = temp1.replace('\'', '')
```

比如将这个字符的前段字符串分隔开：

```
temp1 = temp1.replace('@', ' @')
```

3.5 不考虑含有某个字符的字符串的方法——find()

只要字符串中含有某个字符，就不考虑这个字符串。

想要实现这种判断，可以调用string类型的find()方法，找到返回true，找不到返回false，若返回true，则跳过这个字符串，如：

```
if word.find('@') != -1:
    continue
```

3.5 F-score的计算方法

$$P_{pos} = \frac{true\_pos}{true\_pos + false\_pos\_true\_neg + false\_pos\_true\_neu}$$

$$R_{pos} = \frac{true\_pos}{true\_pos + false\_neg\_true\_pos + false\_neu\_true\_pos}$$

$$P_{neg} = \frac{true\_neg}{true\_neg + false\_neg\_true\_pos + false\_neg\_true\_neu}$$

$$R_{neg} = \frac{true\_neg}{true\_neg + false\_pos\_true\_neg + false\_neu\_true\_neg}$$

$$P_{neu} = \frac{true\_neu}{true\_neu + false\_neu\_true\_pos + false\_neu\_true\_neg}$$

$$R_{neu} = \frac{true\_neu}{true\_neu + false\_pos\_true\_neu + false\_neg\_true\_neu}$$

$$F - score = \frac{P_{pos} \times R_{pos}}{P_{pos} + R_{pos}} + \frac{P_{neg} \times R_{neg}}{P_{neg} + R_{neg}} + \frac{P_{neu} \times R_{neu}}{P_{neu} + R_{neu}}$$

4 实验

详细过程见代码，报告中只阐述关键部分。

4.1 数据清洗部分

将文本中所有大小字母转换成小写

```
temp1 = line.strip('\n').lower()
```

将unicode中\u的编码问题转换成原来的字符

```
temp1 = temp1.replace('\\u2019', '\'')
temp1 = temp1.replace('\\u002c', ',')
```

将’!’和’?’分割开

’?’和’!’是比较重要的带有情感色彩的符号，将其与其他字符串隔离开来，且不能去除。

```
temp1 = temp1.replace('!', ' ! ')
temp1 = temp1.replace('?', ' ? ')
```

对文本中常见的颜文字表情进行替换

考虑到符号对单词归类的干扰比较严重，但直接将符号去掉的话会破坏带有情感色彩的表情特征值，因此在此处将常见的表情替换成单词，并同时在前后加空格保证与其他单词隔开，之后再进行符号的去除。

被替换的表情有：:) ;) :-) :o) =) ;-) :0) :D :P :(:-(^^ ^_^ ^^ ^_^ :-0 -_- <3333 <333 <33 <3 共20个，替换方法如下：

```
temp1 = temp1.replace(':)', ' emotion1 ')
```

对于对结果影响较大的符号，将其隔开

```
temp1 = temp1.replace('-', ' - ')
temp1 = temp1.replace('>', ' > ')
```

单引号对归类影响较大，但将其作为分隔符会产生过多错误的分类，因此将其去除，即替换为空

```
temp1 = temp1.replace('\'', '')
```

使用’@’时后面常跟着想要提醒的用户的昵称或ID，但前面可能还有一些有用的特征信息，因此将其前面的字符串隔开

```
temp1 = temp1.replace('@', ' @')
```

将unicode编码中的’&amp’替换成’&’，方便后续去除，避免产生’amp’这个多余的特征

```
temp1 = temp1.replace('&amp', '&')
```

去除大部分符号，提高归类准确率

```
temp2 = re.split('[][():;&/.,*"=< \t\\\]+', temp1)
```

去除符号后可能造成空字符串的产生，因此先做筛选

```
if len(word) == 0:
    continue
```

排除’#’开头的话题字符串和表示转发的字符串’RT’

```
if word[0] == '#' or word == 'RT':
    continue
```

排除带有’@’的无用特征

```
if word.find('@') != -1:
    continue
```

4.2 朴素贝叶斯分类部分

详细部分见代码，这里只阐述需要注意的部分：

需要做拉普拉斯平滑处理

```
for key in positive_word_dict:
    if positive_word_dict[key] == 0:
        positive_word_dict[key] = 1 / (positive_number + 1) * 1.0
    else:
        positive_word_dict[key] = positive_word_dict[key] / positive_number * 1.0
```

最后的比率可能会过小而导致无法比较，因此需做对数变换

```
if word not in positive_word_dict.keys():
    positive_condition_rate = positive_condition_rate + math.log(1 / (positive_number + 1) * 1.0)
    negative_condition_rate = negative_condition_rate + math.log(1 / (negative_number + 1) * 1.0)
    neutral_condition_rate = neutral_condition_rate + math.log(1 / (neutral_number + 1) * 1.0)
else:
    positive_condition_rate = positive_condition_rate + math.log(positive_word_dict[word] * 1.0)
    negative_condition_rate = negative_condition_rate + math.log(negative_word_dict[word] * 1.0)
    neutral_condition_rate = neutral_condition_rate + math.log(neutral_word_dict[word] * 1.0)
```

4.3 测试部分

编写测试代码，对训练集和验证集进行准确率和F-score的测试，按照方法中F-score的公式进行计算。

最后再对测试集进行分类，输出得到最终结果。

5 结果与讨论

5.1 测试结果

测试训练集train：

```
P_total = 0.6652209830648492
P_pos = 0.7909252669039146
R_pos = 0.7326923076923076
P_neg = 0.3751378169790518
R_neg = 0.9334705075445816
P_neu = 0.8994038748137109
R_neu = 0.5263846489315307
F-score = 0.9799948194625564
```

测试验证集dev：

```
P_total = 0.47944377267230953
P_pos = 0.6578947368421053
R_pos = 0.5217391304347826
P_neg = 0.313953488372093
R_neg = 0.8735294117647059
P_neu = 0.7777777777777778
R_neu = 0.2652232746955345
F-score = 0.7197083296789746
```

5.2 分析

在本课程的lab3 中，我们已经学会了如何使用朴素贝叶斯算法对垃圾邮件进行分类，但当时提供的数据是一个纯净的文本，比较容易分类，而本次实验的数据则含有较多噪声，因此如何对数据进行清洗是关键。

在方法中我已经描述了数据文本中出现的问题，此处则给出这些问题会对字符串归类造成干扰的原因：

- 若不处理退格键，则字符串分隔之后list中的第一个字符串将会是’ID 标签 第一个字符串’，即不便于提取。
- 单词的大小写格式的统一能使更多相同但大小写不同的单词归到一类。
- 去除’\u2019’和’\u002c’的问题能使更多单词被分隔开来，也方便了后续对逗号和引号的去除。
- 两个单词之间的连接符’-’使得这两个单词没法归到该有的类别中去，因此需要替换，在连接符前后各加一个空格。
- 单词简写上的连接符，比如’I’m’中的单引号’’’干扰了被连接的两个单词的归类。
- Twitter上常出现的’@’用于提醒想要提醒的用户，’@’后面往往跟的是那个用户的昵称或ID，这与情态分析无关，应该去除。
- Twitter上出现的连接符’&’，当它将两个单词连接在一起时，会干扰这两个单词的归类，还有当它被编码成’&amp’时，也应该去除。
- Twitter上的话题会用’#’开头来表示，话题与情态分析无关，应该去除。
- Twitter上以’RT’开头的表示转发，这个’RT’是无关特征，应该去除。
- 以’http://’和’https://’开头的表示网址链接，这些可能也与情态分析无关，我搜索了数据文本之后发现，网址格式均为’http://t.co/xxxxxxxx’和’https://t.co/xxxxxxxx’，因此我尝试了多种方法，比如直接去掉网址，或只去掉网址前面重复的几个字符，或按照’/’将其分隔开，或按照’/’和’.’将其分隔开……最后我选取了准确率最高的一种，即将其分隔成’http’、’https’、’t’、’co’、’xxxxxxxx’。
- 数据文本中可能出现邮箱地址，邮箱地址与情态分析无关，应该去除，但想要匹配邮箱，则不能先去除’@’和’.’。我尝试之后发现，使用正则匹配排除邮箱、但保留这两个符号，和直接去除这两个符号、放弃匹配邮箱相比，直接去除符号的准确率更高，因此我放弃了匹配邮箱的处理。
- 数据文本的关键问题在于各种符号跟单词文本连在一起，很难区分开符号和单词，在后续分类上没法将这样的单词归为一类，因此需要做符号的去除，但必须考虑到一些特殊的符号不能直接去除，比如带有情感色彩的符号（’?’、’!’、颜文字表情），对于剩下的符号，是直接去除，还是作为分隔符，还是替换成前后加空格，则需要多次尝试，选择准确率最高的那种方法。

最后我去除了大部分符号，得到了一个较为干净的文本。再对这些文本使用朴素贝叶斯进行分类，虽然最后的准确率并不高，但F-score还差强人意。

6 结论

本次实验针对Twitter数据进行情态分析，我在本次实验中主要针对数据清洗做了较多的搜索、替换、修改、排除，做出有选择性的改动，以求尽量获得一个干净且不失去过多特征值的文本，再使用朴素贝叶斯算法进行分类，最后的结果大体上让我满意。

但是还存在许多可以提升的地方，比如针对数据清洗后，字典中key的出现概率进行降序排序，选取概率更高的前n个key值作为特征，其余的则不做考虑；

比如给key分类，分成几种词性或者几个主题；

比如去除那些没有什么实际意义的介词‘for’、‘to’等；

比如将意义相近的key归到同一类，如’no’与’not’意义相近；

……

可能提高情态分析的准确率的方法还有很多，未来我还需要学习更多的知识，努力提升自己的能力，去更有效的解决这个问题。

7 参考

Aaronji1222，“利用机器学习算法进行特朗普twitter的主题分析”

http://blog.csdn.net/Aaronji1222/article/details/78153269?locationNum=1&fps=1

AI研习社，“手把手教你如何用 Python 做情感分析”

https://www.leiphone.com/news/201706/YXVb0apveG0yYDeT.html

房海朔，“分类：情感分析”

http://blog.csdn.net/qq_36954426/article/details/71105134

liugallup，“六、机器学习系统设计笔记之分类II情感分析”

http://blog.csdn.net/liugallup/article/details/49046423

AI研习社，“详解基于朴素贝叶斯的情感分析及 Python 实现”

https://www.leiphone.com/news/201707/VyUNGYnEy3kXnkVb.html