情感识别

layout: post
title: Detect Sentiment Polarity
subtitle: Natural Language Processing homework
date: 2018-01-05
author: WeiXiao
header-img: img/tag-bg-o.jpg
catalog: true
tags:
- NLP

- Python

任务定义

- Detecting sentiment polarity: 30 points
  - Given text about movie reviews
  - Can we detect sentiment, like whether a comment is
    - Positive?
    - Negative?
  - Can we tell to what extent is a comment positive of negative?
- Data:
  - 5331 positive snippets
  - 5331 negative snippets
- Other resources:
  - The Subjectivity Lexicon

运行环境

- Mac OS 10.13
- Pycharm
- Python 3.6.1

输入输出

- Specifically:
  - rt-polarity.pos contains 5331 positive snippets
  - rt-polarity.neg contains 5331 negative snippets
- Each line in these two files corresponds to a single snippet (usually containing roughly one single sentence); all snippets are down-cased. The snippets were labeled automatically, as described below (see section "Label Decision").
- 输出

方法描述

- 情感分析只需要判读情感是positive和negative,所以这里采用最简单的朴素贝叶斯分类
  - 基本思想是利用特征词和类别的联合概率来计算给定测试集的类别概率.这里有个不严谨的假设就是词和词在表达本句意思的时候是相互独立的,根据贝叶斯公式

    1、找到一个已知分类的待分类项集合，这个集合叫做训练样本集。

    2、统计得到在各类别下各个特征属性的条件概率估计。即
    $P(a_1|y_1), P(a_2|y_1), ..., P(a_m|y_1); P(a_1|y_2), P(a_2|y_2), ..., P(a_m|y_2); ...; P(a_1|y_n), P(a_2|y_n), ..., P(a_m|y_n)$
    。

    3、如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：
    $$P(y_i|x) = \frac{P(x|y_i)P(y_i)}{P(x)}$$

- 在具体的算法实现上,这里首先把前85%的数据分割为训练集,剩下的作为数据集.
  - 对于训练集,对所有在词性文件中出现过的单词,分别统计他们在正例和反例中出现的比率,

最后再在正例反例之间做归一化处理,最后保存在正反例的dict中
    - 对于测试集,首先分割为单词,然后对于在词性文件中的单词,乘以他们对应的概率,最后当遇到句号或者逗号的时候比较正因子和负因子之间的距离,判断出他们到底属于正例还是反例
    - 由于正例的测试集的大小和反例测试集的大小是相同的,也就是说对于随机的一片评论,他是正例还是反例的概率是相同的,所以这里不用乘以比例系数,直接比较大小就能获得比较高的性能
- 最后在判断结果的时候,因为正例和反例是分开评测的,所以理论上正例的所有分类结果都应该是正例,再计数出实际分为正例的数量,最后算出实际的准确度和召回率,最后再计算出F-score

$$\text{precision} = \frac{true\ positive}{true\ positive + false\ positive} = \frac{true\ positive}{no.of\ predicted\ positive}$$

$$\text{recall} = \frac{true\ positive}{true\ positive + false\ negative} = \frac{true\ positive}{no.of\ actual\ positive}$$

$$\text{F1Score} = 2 \times \frac{precision \times recall}{precision + recall}$$

```
        for line in self.temp_negi_test:#判断反例
            pos_factor = 1.0
            neg_factor = 1.0  # 负因子/正因子
            for voca in line.split():
                if voca in self.subj:
                    try:
                        pos_factor *= self.posi_train[voca]
                        neg_factor *= self.nege_train[voca]
                    except KeyError:
                        print("error!")
            if pos_factor > neg_factor:
                fp += 1
            else:
                tn += 1

        return [tp/(tp+fp),tp/(tp+fn),tn/(tn+fn),tn/(tn+fp)]
```

结果分析

- 对于没有去除停止词的训练集,训练结果如下

```
/Users/weixiao/anaconda/bin/python /Users/weixiao/PycharmProjects/textclass/tc.py 0
Positive Precision: 0.7739975698663426
Positive Recall: 0.79625
Negative Precision: 0.7902187902187903
Negative Recall: 0.7675
Positive F-score: 0.7849661121380159
Negative F-score: 0.7786937222574507
```

- 对去去除停止词的训练集,训练结果如下

```
/Users/weixiao/anaconda/bin/python /Users/weixiao/PycharmProjects/textclass/tc.py 1
Positive Precision: 0.7706093189964157
Positive Recall: 0.8052434456928839
Negative Precision: 0.796078431372549
Negative Recall: 0.7602996254681648
Positive F-score: 0.7875457875457874
Negative F-score: 0.7777777777777778
```

实验思考

- 实验的过程中,遇到的第一个问题就是如何读取文件
  - 最开始看到.pos和.neg文件不知所措,后来请教了已经做完的同学才理解数据好像不怎么受后缀名的影响,最后在改变读取格式之后,成功读取文件
- 对于朴素贝叶斯分类,最开始要先了解数学原理,参考了网上的教程,具体如下
  - 在这里,首先了解了贝叶斯相关理论,并且根据

$$P(x|y_i)P(y_i) = P(a_1|y_i)P(a_2|y_i)...P(a_m|y_i)P(y_i) = P(y_i)\prod_{j=1}^{m} P(a_j|y_i)$$

  - 确定了程序步骤
- 最后在效果评价上,采取了和课本一样的维度,分别是准确率,召回率和F-score来进行度量,具体的公式已经在上图中贴出