湖南大学

机器学习实验报告

班　　级：	计科 1907
学　　号：	201908010705/201908010718/201908030319
姓　　名：	杨杰/邓谨/开博
项目名称：	微博情感分析
指导老师：	刘益萍

目 录

微博情感分析实验报告

一、引言

情感分析（Sentimentanalysis），也称意见挖掘(OpinionMining)，主要是对带有感情色彩的主观性文本进行分析、处理、归纳然后进行推理的过程，例如对产品，话题，政策的意见。利用这些分析的结果，消费者可以深入了解商品的实用性，从而优化购买的决策，同时，生产者和经销商可以改善自己的服务，从而赢得竞争的优势。随着信息时代的到来，越来越多的公司开始组建数据分析团队对自身公司的数据进行挖掘、分析。比如某服装公司想调查自己制作的服装的受喜爱程度，就可以从服装的评论入手，挖据文本内容，判断留下评论的用户对服装的喜好态度，积极的、消极的或者是中性的评价。

微博的强大影响力已经深深的吸引了更多的人加入。而对微博的情感分析，不仅可以获取网民的此时的心情，对某个事件或事物的看法，还可以获取其潜在的商业价值，还能对社会的稳定做出一定的贡献。

二、问题定义

通过对微博评论进行预处理、分词以及特征选择等，建立特征词典，构建每条评论的特征向量。之后利用分类算法，如朴素贝叶斯、SVM等，针对训练集的特征向量以及类标签进行训练，得到分类模型，并通过计算在测试集上的预测准确率、召回率等对分类器的分类效果以及不同参数影响进行性能评估。

三、研究思路及算法过程

算法思想

朴素贝叶斯

朴素贝叶斯是在独立性假设的前提下实现的，即在给定目标值时，假设特征之间是相互独立的。

朴素贝叶斯的公式表示为：

$$v_{NB} = \arg\max_{v_j \in V} p(v_j) \prod_i p(a_i \mid v_j)$$

因为这部分需要独立编写代码实现，所以我详细了解并掌握了朴素贝叶斯的流程，并总结如下：

对已知的特征矩阵和标签矩阵，我们定义：

$x = [f1, f2, f3, ..., fn]$ 为一个样本特征向量，其中 fi 是每个特征的特征值。

$y = [y1, y2, y3, ..., yn]$ 为每个样本的真实所属类别。

$X = [x1, x2, x3, ..., xn]$ 为样本集，其中每个样本x_i是x的构成形式。

假设这些样本总共分为了3类，分别为class1,class2,class3。

第一步：

通过样本集的类别分布，对每个类别计算先验概率。也就是需要计算出这些样本分属3个类别各自的比例，比如：

$$p(class1)=\frac{\text{所属class1的样本数目}}{\text{所有样本的总数}}$$

这样，分别计算出 $p(class1), p(class2), p(class3)$ 。

第二步：

计算每个类别下每个特征属性值的出现概率。比如，当为类别1时，特征1的值为 $f1_1$ 时的概率 $p(f1_1 \mid class1)$ 为：

$$p(f1_1 \mid class1)=\frac{\text{在属于class1的样本中特征1的值为}f1_1\text{的样本数目}}{\text{所属class1的样本数目}}$$

第三步：

计算每个样本所属每个类别的概率。比如对于第一个样本，根据它每个特征值的情况，依次从第二步的结果中提取对应值并相乘，得到它分别属于这3个类别的概率，即

$$p(class1|x1) = p(class1)*p(f1_a|class1)*p(f2_b|class1)*\ldots*p(fn_q|class1)/p(x1)$$
$$p(class2|x1) = p(class2)*p(f1_a|class2)*p(f2_b|class2)*\ldots*p(fn_q|class2)/p(x1)$$
$$p(class3|x1) = p(class3)*p(f1_a|class3)*p(f2_b|class3)*\ldots*p(fn_q|class3)/p(x1)$$

第四步：

比较样本预测属于这三个类别的概率值，找出最大概率对应的分类作为这个样本最终的预测分类结果。

研究思路

本实验，在设计过程中主要考虑以下几点：

①文本分类属于有监督的学习，需要整理样本，确定样本数目以及记录样本标签。

②针对样本需要进行分词操作得到评论的词语表示。

③因为分词后每条评论中包含的词语是很多的，这些词并不都是表征能力强的词，所以需要根据词性、词长短等过滤掉大部分的无关词。

④如何表征评论呢？在本实验中，我采用的特征提取模型是向量空间模型(VSM)，即将样本转换为向量。为了能实现这种转换，需要进行确定特征词典和得到特征向量的过程。

⑤虽然可以将所有样本的词都提取出来作为词典，但随着样本数目的增多，词典规模可能达到万级、千万级甚至亿级，这么大的维度可能会带来维度灾难，因此就要想办法从大量的特征中选择一些有代表性的特征而又不影响分类的效果，这个环节，我采用了目前领域内认为比较好的卡方检验方法得到每类中的关键词。

实验步骤

1. 数据预处理

原始数据集中存在重复标记的情况，这部分数据属于噪声，针对噪声的处理，简单的方法有直接去除和随机保留一条数据。本实验中采取直接去除。

相关代码如下：

```
from collections import Counter
dataset = open("dataset.txt", encoding='UTF-8')
lines = []
for line in dataset.readlines():
    lines.append(line[2:])
dataset.close()
newlines = dict(Counter(lines))
# ls = [key for key, value in newlines.items() if value > 1]  # 只展示重复元素
dic = {key: value for key, value in newlines.items() if value > 1}  # 展现重复元素和重复次数
```

2. 分词和过滤

2.1 分词

jieba分词是国内程序员用Python开发的一个中文分词模块，其主要处理思路如下：

1、加载词典dict.txt。结巴分词自带了一个叫做dict.txt的词典，里面有2万多条词，包含了词条出现的次数(这个次数是于作者自己基于人民日报语料等资源训练得出来的)和词性。

2、依据词典dict.txt，基于Trie树结构实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图(DAG)。

3、对于词典中未收录词，采用了基于汉字成词能力的HMM模型，使用viterbi算法预测分词。

4、已收录词和未收录词全部分词完毕后，使用动态规划寻找DAG的最大概率路径，找出基于词频的最大切分组合。

5、输出分词结果。

jieba分词有专门的python包——jieba，通过安装及import即可使用。虽然jieba分词效果上不如中科院的ICTCLAS和哈工大的ltp，但其由python编写，代码清晰，扩展性好，所以在本实验中，采用jieba包进行分词。

相关代码如下：

```
file = open(dataFileName, encoding='UTF-8')
# 读取文件中的所有行
contents = file.readlines()
file.close()
for line in contents:
    classVec.append(int(line[0]))
    # 利用正则表达式u'[\u4e00-\u9fa5]+'过滤掉输入数据中的所有非中文字符；
    contentstr = "".join(re.findall(u'[\u4e00-\u9fa5]+', line[2:]))
    content = (" ".join(jieba.cut(contentstr))).strip('\n').split(' ')
```

2.2 去除停用词

对于分词后的每个词语，都需要和停用词列表进行对比来决定是否保留。

相关代码如下：

```
stop = [row.strip() for row in open('stop.txt', 'r', encoding='utf-8').readlines()]  # 停用词
content = [item for item in content if item not in stop]
```

2.3 一字词语过滤

分词结果中会出现一些一字名词词语，如“油”、“板”、“风”等，这些词的表征能力较差，基本不能从中理解到关于评论类别的信息，所以在本实验中，也对这种一字词语进行了过滤。

相关代码如下：

```
content = [item for item in content if len(item) > 1]
```

3. 特征提取和表达

特征提取和特征选择是减少特征数量、降维的过程，主要目的是使模型泛化能力更强，减少过拟合的发生。所以，如何选择更具表征能力的词语作为特征词，既降低特征维度，减小计算开销，又能很好地表现同类别评论中词语的相关性以及不同类别评论中词语的差别是特征选择的关键。

3.1 统计词频信息

针对上一步得到的每条评论中的保留词，为了方便之后进行卡方检验以及tfidf的计算，这里我统计了两种词频信息，分别为：

1、每个类别中出现词语的词频信息。(wordtimes)

2、每个类别中出现的词各自在多少篇评论中出现的信息。(classtimes)

这里需要注意，并不只是简单的统计。在统计完某篇评论中出现词语的词频之后，还要做进一步判断，如果某个词在这篇评论中仅仅出现一次，我们就认为这种词语对评论所属类别决定意义不大，可以忽略。而且，jieba分词会把空格也看做一个词语，所以这里还需要忽略分词结果中的空格词。

python中的字典是一种可变容器模型，可存储任意类型对象，当然也就可以存储文本字符串，它具有键key和值上value结构，便于快速定位词语及对应值。考虑到词典结构的优势，在本实验中，我利用dictionary结构来统计文本信息，其中.key存储词语，.value存储词语的词频信息。

相关代码如下：

```
word_fd = FreqDist()  # 可统计所有词的词频
    con_word_fd = ConditionalFreqDist()  # 可统计积极文本中的词频和消极文本中的词频
    for word in label0Words:
        word_fd[word] += 1
        con_word_fd['0'][word] += 1
    for word in label1Words:
        word_fd[word] += 1
        con_word_fd['1'][word] += 1
    for word in label2Words:
        word_fd[word] += 1
        con_word_fd['2'][word] += 1
    for word in label3Words:
        word_fd[word] += 1
        con_word_fd['3'][word] += 1
```

3.2 卡方检验

根据卡方检验的原理，假设当前这个词为kv，在本实验中特征选择中的几个观察值分别为：

①在该类别中包括这个词的文档数目。命名为kv_in_class。

②在该类别外包括这个词的文档数目。命名为kv_out_class。

③在该类别中不包含这个词的文档数目。命名为not_kv_in_class。

④在该类别外不包含这个词的文档数目。命名为not_kv_out_class。

对应于独立样本四格表各部分表示如图5.11所示。

特征选择	属于此类别c	不属于此类别c	总计
包含当前词t	A *kv_in_class*	B *kv_out_class*	A+B
不包含当前词t	C *not_kv_in_class*	D *not_kv_out_class*	C+D
总计	A+C	B+D	N

图5.11 独立样本四格表示意图

“词t与类别c有关系”的卡方检验的计算公式是这样的：

$$x^2(t,c)=\frac{N(AD-BC)^2}{(A+C)(A+B)(B+D)(C+D)}$$

因为A+C、B+D、N对于每个词的计算值是一样的，而且我们主要是想通过比较CHI 的大小，找到与类别c更有关或者说在类别c中更具表征性的词语，以此筛选关键特征词，对求CHI的准确计算值并没有要求。所以在本实验中，为了减少计算量，提高计算效率，我将CHI的计算比较公式简化成如下形式：

$$x^2(t,c)=\frac{(AD-BC)^2}{(A+B)(C+D)}$$

关于CHI的计算，主要根据之前得到的各类别中词各自在多少条评论中出现的信息得到。

相关代码如下：

```
    for word, freq in word_fd.items():
        label0_score = BigramAssocMeasures.chi_sq(con_word_fd['0'][word], (freq,

label0_word_count), total_word_count)
        label0_word[word] = label0_score
        label1_score = BigramAssocMeasures.chi_sq(con_word_fd['1'][word], (freq,

label1_word_count), total_word_count)
        label1_word[word] = label1_score
        label2_score = BigramAssocMeasures.chi_sq(con_word_fd['2'][word], (freq,

label2_word_count), total_word_count)
        label2_word[word] = label2_score
        label3_score = BigramAssocMeasures.chi_sq(con_word_fd['3'][word], (freq,

label3_word_count), total_word_count)
        label3_word[word] = label3_score
```

3.3 利用CHI筛选每类中的特征词

遍历每个类别中的每个保留词，计算该词的CHI值，存储于CHI_dic[kv]中，通过sorted(CHI_dic,key=CHI_dic.get,reverse=True)，根据CHI值大小对本类别中出现的所有词进行降序排列，存储于相应列表中。

因为每类评论的数目并不相等，这样也会导致在训练和测试时各类评论数目不平均，所以在本实验中，每类中保留的特征词的数目根据评论数目多少决定。

相关代码如下：

```
vocabList = []
    ele = []

    ele.append(sorted(label0_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[0] * number)])
    ele.append(sorted(label1_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[1] * number)])
    ele.append(sorted(label2_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[2] * number)])
    ele.append(sorted(label3_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[3] * number)])
```

3.4 构建特征词典和特征向量

把每类中的保留词取并集即为此时的特征词典。很明显，此时的特征维度已经大大降低了。用特征词典对每条评论中的保留词进行筛选，也就是说，每条评论中和本类主题关系不大的词都已经被忽略了，然后构建特征向量，将在特征词典中出现的保留词置1，否则置0。

相关代码如下：

```
    for item in ele:
        for w, f in item:
            vocabList.append(w)
    vocabList = list(set(vocabList))

    return vocabList

def Words_to_vec(vocabList, wordSet):
    """
    1.函数说明：根据vocabList词汇表 将每个评价分词后再进行向量化 即出现为1 不出现为0
    2.vocablit: 词汇表
    3.wordSet: 生成的词向量
    return：返回的词向量
    """
    # print("生成文本向量")
    featureVec = [0] * len(vocabList)

    for word in wordSet:
        if word in vocabList:
            # 如果在词汇表中的话 便将其所在位置赋为1
            featureVec[vocabList.index(word)] = 1
        else:
            pass

    return featureVec
```

4. 训练过程

在本实验中，我采用的是常用的K折交叉验证，它的思想是将原始数据分成K组(一般是均分)，将每个子集数据分别做一次验证集，其余的K-1组子集数据作为训练集,这样会得到K个模型，用这K个模型最终的验证集的分类准确率的平均数作为此K-CV下分类器的性能指标。

本实验中，评论共有4个类别，根据交叉验证分好训练集和测试集之后，需要针对训练集得到一些关键值，完成模型建立。

①根据训练样本的真实标签情况，分别记录训练样本中分属不同类别的评论的数目，存储于列表class_amount的class_amount[0]，class_amount[1]，…… ，class_amount[3]。

②根据①得到的训练样本中分属不同类别的评论的数目，分别记录训练样本分属不同类别的评论数目占训练样本总数的概率，存储于列表p_class_amount的p_class_amount[0]，p_class_amount[1]，…… ，p_class_amount[3]。

③遍历每个类别下的所有特征属性，分别记录针对当前类别的评论，每个特征属性的分布情况，在这里，我只考虑了属性值等于0和不等于0两种情况，也就是说，每个特征属性只有两种取值。

这样分别得到每个类别内的训练样本中，每个特征属性为1时的样本数目，存储在列表feature_list_class0_1，feature_list_class1_1，……，feature_list_class3_1中。其中，每个列表为特征数目×1的存储结构。

并将所占此类别内样本的比例存储在列表p_feature_list_class0_1，p_feature_list_class1_1，……，p_feature_list_class3_1中。其中，每个列表为特征数目×1的存储结构。

这里需要说明，多数情况下，该比例是对概率的一个良好的估计。但因为特征向量稀疏，甚至存在在某个类别的样本中，某个特征值并没有等于1的情况，这将导致其对应的概率为0，这样在测试中，会造成如下影响：

- 将来的查询此概率项将会在贝叶斯分类器中占统治地位，因为贝叶斯公式中计算的其他所有概率项都将乘以此0值。

为了避免此问题，所以需要采用一种平滑技术，在本实验中，我采用的是相对简单的拉普拉斯平滑，其原理是假定了统一先验概率。例如由

$$p(f1_1 \mid class1) = \frac{\text{在属于class1的样本中特征1的值为}f1_1\text{的样本数目}}{\text{所属class1的样本数目}}$$

修改为：

$$p(f1_1 \mid class1) = \frac{\text{在属于class1的样本中特征1的值为}f1_1\text{的样本数目}+1}{\text{所属class1的样本数目}+2}$$

简化即为：

$$p = \frac{r+1}{n+2}$$

事实上，分子加1和分母加2背后的基本原理是这样的：在执行实际的试验之前，我们假设已经有两次试验，一次成功和一次失败。

由以上步骤，完成朴素贝叶斯的训练过程。即针对训练集的样本和特征属性的关系进行了学习。

相关代码如下：

```
def trainNB(trainMat, trainLabel):
    """
```

```
    1.函数说明：朴素贝叶斯训练函数
    2.trainMat：训练文本的词向量矩阵
    3.trainLable：训练数据的类别标签
    4.return:
        pVecList:
            p0vec:label为0的评论
            p1vec:label为1的评论
            p2vec:label为2的评论
            p3vec:label为3的评论
        pList:对应概率
    """
    # 训练集的数量
    numTraindocs = len(trainMat)
    # 单词数
    numWords = len(trainMat[0])
    # 各类情感类评论数量及概率
    p0Num = 0
    p1Num = 0
    p2Num = 0
    p3Num = 0

    for label in trainLabel:
        if label == 0:
            p0Num = p0Num + 1
        elif label == 1:
            p1Num = p1Num + 1
        elif label == 2:
            p2Num = p2Num + 1
        else:
            p3Num = p3Num + 1

    p0 = p0Num / float(numTraindocs)
    p1 = p1Num / float(numTraindocs)
    p2 = p2Num / float(numTraindocs)
    p3 = p3Num / float(numTraindocs)

    label0Num = np.ones(numWords)
    label1Num = np.ones(numWords)
    label2Num = np.ones(numWords)
    label3Num = np.ones(numWords)

    for i in range(numTraindocs):
        if trainLabel[i] == 0:
            label0Num += trainMat[i]
        elif trainLabel[i] == 1:
            label1Num += trainMat[i]
        elif trainLabel[i] == 2:
            label2Num += trainMat[i]
        else:
            label3Num += trainMat[i]

    p0vec = label0Num / (p0Num + 2)
    p1vec = label1Num / (p1Num + 2)
    p2vec = label2Num / (p2Num + 2)
    p3vec = label3Num / (p3Num + 2)

    return [p0vec, p1vec, p2vec, p3vec], [p0, p1, p2, p3]
```

5. 测试过程

根据交叉验证得到的测试集，依次将测试集中的每个样本送入训练阶段得到的朴素贝叶斯分类模型中，得到每个样本最可能所属的类别。具体过程如下：

①针对每个测试样本，首先初始化测试样本分属每个类别的概率，存储于列表belong_classn中，其中，列表为类别数目×1的存储结构。

②分别将训练过程中得到的训练样本分属不同类别的评论数目占训练样本总数的概率p_class_amount依次存储到belong_classn中。

③得到当前测试样本中，特征属性值等于1的特征的索引。

④根据训练过程中得到的每个类别内的训练样本中，每个特征属性为1时的样本数目占此类别内样本的比例p_feature_list_class0_1，p_feature_list_class1_1，.....，p_feature_list_class3_1，将③中索引对应的概率值相乘并更新对应的belong_classn。

⑤得到当前测试样本中，特征属性值等于0的特征的索引。

⑥根据训练过程中得到的每个类别内的训练样本中，每个特征属性为1时的样本数目所占此类别内样本的比例，通过(1-③中索引对应的概率值)得到相应特征属性为0时的样本数目所占此类别内样本的比例，相乘并更新对应的belong_classn。

⑦考虑到很多概率值都很小，这导致最终求得的列表belong_classn中的值特别小，甚至可能最终小到不可比，所以在这里，我采用了取log的方式，便于之后分属各个类别的概率值之间的大小比较。

⑧找到分属各个类别概率值的最大值，作为当前测试样本的预测分类结果。

相关代码如下：

```
def classifyNB(vec2Classify, pVec, p):
    """
    1.函数说明：分类 比较p0、p1、p2、p3的大小 并返回相应的预测类别
    2.vec2Classify:返回的词汇表对应的词向量
    """

    p0 = sum(np.log(vec2Classify * pVec[0] + (1 - vec2Classify) * (1 -
pVec[0]))) + np.log(p[0])
    p1 = sum(np.log(vec2Classify * pVec[1] + (1 - vec2Classify) * (1 -
pVec[1]))) + np.log(p[1])
    p2 = sum(np.log(vec2Classify * pVec[2] + (1 - vec2Classify) * (1 -
pVec[2]))) + np.log(p[2])
    p3 = sum(np.log(vec2Classify * pVec[3] + (1 - vec2Classify) * (1 -
pVec[3]))) + np.log(p[3])

    pmax = max(p0, p1, p2, p3)
    if p0 == pmax:
        return 0
    elif p1 == pmax:
        return 1
    elif p2 == pmax:
        return 2
    else:
        return 3
```

四、实验结果与讨论

性能评估

交叉验证

要评估分类效果的好坏，对于原始数据我们要将其划分为train data和test data。train data用于训练，test data用于测试正确率(validation error)。但是为了避免偶然性的影响，不能只做出随机一次划分，得到一个validation error，就作为衡量这个算法好坏的标准。必须进行多次随机的划分，分别在其上面计算出各自的validation error。这样通过某种结合方式有效利用这一组validation，就可以较好的、准确的衡量算法的好坏。所以，在本实验中用到了交叉验证的方法，可以有效消除一次检验所带来的波动，得出比较合理的分类正确率。

交叉验证是用来验证分类器的性能的一种统计分析方法，基本思想是把在某种意义下将原始数据(dataset)进行分组，一部分作为训练集(train set)，另一部分作为验证集(validation set)，首先用训练集对分类器进行训练，再利用验证集来测试训练得到的模型(model)。

使用交叉验证方法的目的有 3 个：

①从有限的学习数据中获取尽可能多的有效信息；

②从多个方向开始学习样本， 可以有效的避免陷入局部最小值；

③无论是训练样本还是测试样本都得到了尽可能多的学习，可以在一定程度上避免过拟合问题。

在本实验中，我采用的是常用的K折交叉验证。

正确率、精确率和召回率

通常，我们利用正确率(accuracy)来评价分类算法。正确率确实是一个很好很直观的评价指标，但是有时候正确率高并不能代表一个算法就好。因为在本实验中我们的数据分布不均衡，类别为0的评论很多，而类别为1、2、3的评论很少，完全错分类别1、2、3依然可以达到很高的正确率却忽视了我们关注的东西。所以，在本实验中，我采用正确率、精确率和召回率三者一同来评价分类效果。

关于正确率、精确率和召回率的概念，这里会涉及到几个模型评价术语，现在假设我们的分类目标只有两类，则会得到四种情况：

1)True positives(TP):真实为正类，预测为正类的样本数；

2)False positives(FP):真实为负类，预测为正类的样本数；

3)False negatives(FN):真实为正类，预测为负类的样本数；

4)True negatives(TN):真实为负类，预测为负类的样本数。

由此得到四者的关系如图所示。

		predicted class	
		Class=Yes	Class=No
ground truth	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

图1.1 混淆矩阵示意图

正确率、精确率、召回率、F1值的计算公式分别为：

1)正确率(accuracy)

accuracy = (TP+TN)/(TP+FN+FP+TN)

即被分对的样本数除以所有的样本数，通常来说，正确率越高，分类器越好。

2)精确率(precision)

precision=TP/(TP+FP)

即被分为正例的示例中实际为正例的比例。是针对我们**预测结果**而言的。

3)召回率(recall)

recall=TP/(TP+FN)

即得到有多少个正例被分为正例的比例。是针对我们原来的**样本**而言的。

4)F1值(F-score)

F-score=precision*recall

实验结果

```
准确率为：
Label0 0.9345002051141802
Label1 0.246367656348705
Label2 0.1619718309859155
Label3 0.18181818181818182
Total 0.7811663991439273
召回率为：
Label0 0.9345002051141802
Label1 0.246367656348705
Label2 0.1619718309859155
Label3 0.18181818181818182
Total 0.38116446856674563
```

算法优势

时间复杂度分析：朴素贝叶斯对训练集和测试集的大小而言是线性的，这在某种意义上是最优的。

五、主要结论

通过此次实验，让我对朴素贝叶斯有了更深刻的理解，原本只是了解基本的先验概率公式。实验过程中学习了中文的分词以及停用词的使用，使分类更加的准确，也认识到了贝叶斯广阔的实用空间，对于机器学习这门课的兴趣也更加浓厚。

六、参考文献

- https://blog.csdn.net/qwe1110/article/details/103391632

- https://blog.csdn.net/qq_38364053/article/details/85784648
- https://www.sohu.com/a/313188111_787107
- http://en.wikipedia.org/wiki/Curse_of_dimensionality
- Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47.
- Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze, Introduction to Information Retrieval

七、附录

项目地址

本项目开源：https://gitee.com/gunshi3/weibo-sentiment-analysis

完整代码

```
import jieba
import numpy as np
import re
import random
import time
from nltk.probability import FreqDist, ConditionalFreqDist
from nltk.metrics import BigramAssocMeasures


def data_split(full_list, ratio):
    n_total = len(full_list)
    offset = int(n_total * ratio)
    random.shuffle(full_list)
    sublist_1 = full_list[:offset]
    sublist_2 = full_list[offset:]
    return sublist_1, sublist_2


def split_train_test(dataFileName, ratio):
    dataset = open(dataFileName, encoding='UTF-8')
    contents = dataset.readlines()
    trainls, testls = data_split(contents, ratio)
    trainset = open("train.txt", "w", encoding='UTF-8')
    for row in trainls:
        trainset.write(row)
    trainset.close()
    testset = open("test.txt", "w", encoding='UTF-8')
    for row in testls:
        testset.write(row)
    testset.close()


def dataProsessing(dataFileName):
    file = open(dataFileName, encoding='UTF-8')
    # 读取文件中的所有行
    contents = file.readlines()
    file.close()
```

```
    # 分词处理
    classVec = []
    contentList = []
    label0Words = []
    label1Words = []
    label2Words = []
    label3Words = []
    labelNum = [0, 0, 0, 0]

    for line in contents:
        classVec.append(int(line[0]))
        # 利用正则表达式u'[\u4e00-\u9fa5]+'过滤掉输入数据中的所有非中文字符；
        contentstr = "".join(re.findall(u'[\u4e00-\u9fa5]+', line[2:]))
        content = (" ".join(jieba.cut(contentstr))).strip('\n').split(' ')
        stop = [row.strip() for row in open('stop.txt', 'r', encoding='utf-
8').readlines()]  # 停用词
        content = [item for item in content if len(item) > 1]
        content = [item for item in content if item not in stop]
        contentList.append(content)
        if line[0] == '0':
            labelNum[0] += 1
            for item in content:
                label0Words.append(item)
        elif line[0] == '1':
            labelNum[1] += 1
            for item in content:
                label1Words.append(item)
        elif line[0] == '2':
            labelNum[2] += 1
            for item in content:
                label2Words.append(item)
        elif line[0] == '3':
            labelNum[3] += 1
            for item in content:
                label3Words.append(item)

    return contentList, classVec, label0Words, label1Words, label2Words,
label3Words, labelNum


# 获取信息量较高(前number个)的词的特征(卡方统计)
def jieba_feature(number, label0Words, label1Words, label2Words, label3Words,
labelNum):
    word_fd = FreqDist()  # 可统计所有词的词频
    con_word_fd = ConditionalFreqDist()  # 可统计积极文本中的词频和消极文本中的词频
    for word in label0Words:
        word_fd[word] += 1
        con_word_fd['0'][word] += 1
    for word in label1Words:
        word_fd[word] += 1
        con_word_fd['1'][word] += 1
    for word in label2Words:
        word_fd[word] += 1
        con_word_fd['2'][word] += 1
    for word in label3Words:
        word_fd[word] += 1
        con_word_fd['3'][word] += 1
```

```
    label0_word_count = con_word_fd['0'].N()  # label0词的数量
    label1_word_count = con_word_fd['1'].N()  # label1词的数量
    label2_word_count = con_word_fd['2'].N()  # label2词的数量
    label3_word_count = con_word_fd['3'].N()  # label3词的数量

    # 一个词的信息量等于积极卡方统计量加上消极卡方统计量
    total_word_count = label0_word_count + label1_word_count + label2_word_count
+ label3_word_count
    label0_word = {}
    label1_word = {}
    label2_word = {}
    label3_word = {}

    for word, freq in word_fd.items():
        label0_score = BigramAssocMeasures.chi_sq(con_word_fd['0'][word], (freq,

label0_word_count), total_word_count)
        label0_word[word] = label0_score
        label1_score = BigramAssocMeasures.chi_sq(con_word_fd['1'][word], (freq,

label1_word_count), total_word_count)
        label1_word[word] = label1_score
        label2_score = BigramAssocMeasures.chi_sq(con_word_fd['2'][word], (freq,

label2_word_count), total_word_count)
        label2_word[word] = label2_score
        label3_score = BigramAssocMeasures.chi_sq(con_word_fd['3'][word], (freq,

label3_word_count), total_word_count)
        label3_word[word] = label3_score

    vocabList = []
    ele = []

    ele.append(sorted(label0_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[0] * number)])
    ele.append(sorted(label1_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[1] * number)])
    ele.append(sorted(label2_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[2] * number)])
    ele.append(sorted(label3_word.items(), key=lambda item: item[1],
reverse=True)[:int(labelNum[3] * number)])
    for item in ele:
        for w, f in item:
            vocabList.append(w)
    vocabList = list(set(vocabList))

    return vocabList


def loadTestFile(dataFileName):
    file = open(dataFileName, encoding='UTF-8')
    # 读取文件中的所有行
    contents = file.readlines()
    file.close()

    # 分词处理
    classVec = []
```

```
    contentList = []

    for line in contents:
        classVec.append(int(line[0]))
        # 利用正则表达式u'[\u4e00-\u9fa5]+'过滤掉输入数据中的所有非中文字符；
        contentstr = "".join(re.findall(u'[\u4e00-\u9fa5]+', line[2:]))
        content = (" ".join(jieba.cut(contentstr))).strip('\n').split(' ')
        content = [item for item in content if len(item) > 1]
        contentList.append(content)

    return contentList, classVec


def Words_to_vec(vocabList, wordSet):
    """
    1.函数说明：根据vocabList词汇表 将每个评价分词后再进行向量化 即出现为1 不出现为0
    2.vocablit: 词汇表
    3.wordSet: 生成的词向量
    return：返回的词向量
    """
    # print("生成文本向量")
    featureVec = [0] * len(vocabList)

    for word in wordSet:
        if word in vocabList:
            # 如果在词汇表中的话 便将其所在位置赋为1
            featureVec[vocabList.index(word)] = 1
        else:
            pass

    return featureVec


def trainNB(trainMat, trainLabel):
    """
    1.函数说明：朴素贝叶斯训练函数
    2.trainMat：训练文本的词向量矩阵
    3.trainLable：训练数据的类别标签
    4.return:
        pVecList:
            p0vec:label为0的评论
            p1vec:label为1的评论
            p2vec:label为2的评论
            p3vec:label为3的评论
        pList:对应概率
    """
    # 训练集的数量
    numTraindocs = len(trainMat)
    # 单词数
    numWords = len(trainMat[0])
    # 各类情感类评论数量及概率
    p0Num = 0
    p1Num = 0
    p2Num = 0
    p3Num = 0

    for label in trainLabel:
        if label == 0:
```

```
            p0Num = p0Num + 1
        elif label == 1:
            p1Num = p1Num + 1
        elif label == 2:
            p2Num = p2Num + 1
        else:
            p3Num = p3Num + 1

    p0 = p0Num / float(numTraindocs)
    p1 = p1Num / float(numTraindocs)
    p2 = p2Num / float(numTraindocs)
    p3 = p3Num / float(numTraindocs)

    label0Num = np.ones(numWords)
    label1Num = np.ones(numWords)
    label2Num = np.ones(numWords)
    label3Num = np.ones(numWords)

    for i in range(numTraindocs):
        if trainLabel[i] == 0:
            label0Num += trainMat[i]
        elif trainLabel[i] == 1:
            label1Num += trainMat[i]
        elif trainLabel[i] == 2:
            label2Num += trainMat[i]
        else:
            label3Num += trainMat[i]

    p0vec = label0Num / (p0Num + 2)
    p1vec = label1Num / (p1Num + 2)
    p2vec = label2Num / (p2Num + 2)
    p3vec = label3Num / (p3Num + 2)

    # model = open("model.txt", "w", encoding="UTF-8")
    # model.write(str(p0vec) + "\n")
    # model.write(str(p1vec) + "\n")
    # model.write(str(p2vec) + "\n")
    # model.write(str(p3vec) + "\n")
    # model.write(str(p0) + "\n")
    # model.write(str(p1) + "\n")
    # model.write(str(p2) + "\n")
    # model.write(str(p3) + "\n")
    # model.close()

    return [p0vec, p1vec, p2vec, p3vec], [p0, p1, p2, p3]


def classifyNB(vec2Classify, pVec, p):
    """
    1.函数说明：分类 比较p0、p1、p2、p3的大小 并返回相应的预测类别
    2.vec2Classify:返回的词汇表对应的词向量
    """

    p0 = sum(np.log(vec2Classify * pVec[0] + (1 - vec2Classify) * (1 -
pVec[0]))) + np.log(p[0])
    p1 = sum(np.log(vec2Classify * pVec[1] + (1 - vec2Classify) * (1 -
pVec[1]))) + np.log(p[1])
```

```
    p2 = sum(np.log(vec2Classify * pVec[2] + (1 - vec2Classify) * (1 -
pVec[2]))) + np.log(p[2])
    p3 = sum(np.log(vec2Classify * pVec[3] + (1 - vec2Classify) * (1 -
pVec[3]))) + np.log(p[3])

    pmax = max(p0, p1, p2, p3)
    if p0 == pmax:
        return 0
    elif p1 == pmax:
        return 1
    elif p2 == pmax:
        return 2
    else:
        return 3


def main():
    split_train_test("quchong.txt", 0.8)
    trainList, trainLable, label0Words, label1Words, label2Words, label3Words,
labelnum = dataProsessing("train.txt")
    vocabList = jieba_feature(0.1, label0Words, label1Words, label2Words,
label3Words, labelnum)
    i = 0
    file = open("vocabList.txt", 'w', encoding='UTF-8')
    for item in vocabList:
        file.write(item + " ")
        i = i + 1
        if i % 20 == 0:
            file.write('\n')
    file.close()
    print("创建词汇表完成")

    trainMat = []
    cnt = 0
    for train in trainList:
        cnt += 1
        # print("正在处理第%s条训练数据" % cnt)
        trainMat.append(Words_to_vec(vocabList, train))
    print("训练集数据处理完毕")
    pVec, p = trainNB(np.array(trainMat, dtype='float16'), np.array(trainLable,
dtype='float16'))
    print("生成训练集指标")
    print("训练样本数 %s" % cnt)
    print("特征维度 %s" % len(vocabList))
    print(pVec)
    print(p)

    # 加载测试集数据进行测试
    testList, testLable = loadTestFile("test.txt")
    predictLable = []
    nn = 0
    for test in testList:
        doc = np.array(Words_to_vec(vocabList, test))
        nn += 1
        # print("正在处理第%s条数据" % nn)
        if classifyNB(doc, pVec, p) == 0:
            predictLable.append(0)
        elif classifyNB(doc, pVec, p) == 1:
```

```
            predictLable.append(1)
        elif classifyNB(doc, pVec, p) == 2:
            predictLable.append(2)
        else:
            predictLable.append(3)

    TP = [0, 0, 0, 0]
    labelNum = [0, 0, 0, 0]
    for i in range(len(testLable)):
        if testLable[i] == 0:
            labelNum[0] += 1
            if predictLable[i] == 0:
                TP[0] += 1
        elif testLable[i] == 1:
            labelNum[1] += 1
            if predictLable[i] == 1:
                TP[1] += 1
        elif testLable[i] == 2:
            labelNum[2] += 1
            if predictLable[i] == 2:
                TP[2] += 1
        else:
            labelNum[3] += 1
            if predictLable[i] == 3:
                TP[3] += 1
    print("准确率为：")
    print("Label0 " + str(TP[0] / labelNum[0]))
    print("Label1 " + str(TP[1] / labelNum[1]))
    print("Label2 " + str(TP[2] / labelNum[2]))
    print("Label3 " + str(TP[3] / labelNum[3]))
    print("Total " + str(sum(TP) / sum(labelNum)))
    print("召回率为：")
    print("Label0 " + str(TP[0] / labelNum[0]))
    print("Label1 " + str(TP[1] / labelNum[1]))
    print("Label2 " + str(TP[2] / labelNum[2]))
    print("Label3 " + str(TP[3] / labelNum[3]))
    print("Total " + str((TP[0] / labelNum[0] + TP[1] / labelNum[1] + TP[2] /
labelNum[2] + TP[3] / labelNum[3]) / 4))


if __name__ == "__main__":
    start = time.perf_counter()
    main()
    print("耗时{}s".format(time.perf_counter() - start))
```