智能信息处理的统计方法课程报告
Homework 1
Naïve Bayes 分类器

HSJMDMG

October 10, 2016

1 基本原理

1.1 贝叶斯公式

定理 1 (贝叶斯公式)

$$P(a|b) = \frac{P(b|a) * P(a)}{P(b)}$$

对于文本分类问题，集合 $C = \{c_1, c_2, \ldots c_n\}$ 为所有的类别的集合，其中 c_i 为第 i 个具体的类别，用集合 $W = \{w_1, w_2, \ldots w_n\}$ 来描述一个文本，其中 w_i 为第 i 个单词

根据贝叶斯公式，则有：

$$P(C|W) = \frac{P(W|C) * P(C)}{P(W)}$$

称其中 $P(C)$ 为 C 的先验概率, $P(W|C)$ 为类条件概率, $P(C|W)$ 为 C 的后验概率

1.2 Naïve Bayes

在文本分类问题中，对文本 W 进行分类，相当于求 C 的后验概率 $P(C|W)$。

贝叶斯公式：

$$P(C|W) = \frac{P(W|C) * P(C)}{P(W)}$$

具体地，对每个类别 c_i 有：

$$P(c_i|W) = \frac{P(W|c_i) * P(c_i)}{P(W)}$$

又对于 W 来说，

$$c_{MAP} = argmaxP(c_i|d) = argmax\frac{P(W|c_i) * P(c_i)}{P(W)}, \quad for\ c_i \in C$$

对于文本 W 来说，P(W) 为一常量。也即：

$$c_{MAP} = argmaxP(W|c_i) * P(c_i), \quad for\ c_i \in C$$

如果可以得到类条件概率 $P(W|C)$ 和 C 的先验概率 $P(C)$，就可以得到 c_{MAP} 其中 P(C) 通过统计训练集中计算含 c_i 标签的样本数与总样本数的比例来估计，

对于 P(W|C)，Naïve Bayes 算法假设所有单词的出现都是都是独立的，并且假设所有单词的位置信息是不重要的，即可得到：

$$P(W|c_i) = \prod_{j=0}^{|W|} P(w_j|c_i)$$

其中 $P(w_j|c_i)$ 可以通过统计训练集中单词 w_j 在类别 c_j 中出现的频率来求。

所以对于文本 W 来说：

$$c_{MAP} = argmax\prod_{j=0}^{|W|} P(w_j|c_i) * P(c_i), \quad for\ c_i \in C$$

1.3 拉普拉斯平滑

为了避免先验概率 $P(c_i) = 0$ 和类条件概率 $Pw_j|ci = 0$，我们对其进行加一平滑 (add-one smoothing)，具体细节见相关模型。

1.4 多项式模型

这是一种以“单词”为核心的模型，效果取决于所有训练（测试）集中总单词数的多少和单词在词集中的统计规律，和文档的多少没有必然的联系。

设某一文档 $doc = (w_1, w_2, \ldots, w_n)$, 其中 w_i 为文本 doc 中出现的单词，允许存在 $w_i = w_j, (i \neq j)$，对于先验概率 $P(c_i)$ 和条件概率 $P(w_j|c_j)$ 采用统计频率的方法进行测量，则从直观上有：

$$P(c_i) = \frac{count(all\ words\ in\ c_i)}{count(all\ words\ in\ training\ set)}$$

$$P(w_j|c_i) = \frac{count(w_j\ in\ c_i)}{count(all\ words\ in\ c_i)}$$

具体地，为了防止 $P(c_i) = 0$，$P(w_j|c_i) = 0$, 对其进行 Laplace Smooth 操作，则有：

$$P(c_i) = \frac{count(all\ words\ in\ c_i) + 1}{count(all\ words\ in\ training\ set) + |C|}$$

$$P(w_j|c_i) = \frac{count(w_j \ in \ c_i + 1)}{count(all \ words \ in \ c_i) + |V_t|}$$

其中 C 为类别的集合，为 V_t 为训练集的词汇表

1.5 伯努利模型

这是一种以“文档”为核心的模型，对于单个单词而言，只有“出现”和” 不出现“的区别是有意义的，而出现次数的多寡是无意义的。

设某一文档 $doc = (w_1, w_2, \ldots, w_n)$, 其中 w_i 为文本 doc 中出现的单词，允许存在 $w_i = w_j, (i \neq j)$，对于先验概率 $P(c_i)$ 和条件概率 $P(w_j|c_j)$ 采用统计频率的方法进行测量，则从直观上有：

$$P(c_i) = \frac{count(all \ documents \ in \ c_i)}{count(all \ documents \ in \ training \ set)}$$

$$P(w_j|c_i) = \frac{count(documents \ that \ containing \ w_j \ in \ c_i)}{count(all \ words \ in \ c_i)}$$

具体地，为了防止 $P(c_i) = 0$，$P(w_j|c_i) = 0$, 对其进行 Laplace Smooth 操作，则有：

$$P(c_i) = \frac{count(all \ documents \ in \ c_i) + 1}{count(all \ documents \ in \ training \ set) + |C|}$$

$$P(w_j|c_i) = \frac{count(documents \ that \ containing \ w_j \ in \ c_i) + 1}{count(all \ words \ in \ c_i) + 2}$$

其中 C 为类别的集合

并没有实现这种模型

2 代码分析

文件说明见 README.md
实现的代码中采用了多项式模型
虽然代码是按照《机器学习实战》上的经典方法来写的，但是感觉多项式模型的 TraningMatrix 不一定需要按照文本个数来添加样本，可以直接做成一个大的 1 * |V| 的向量

2.1 文本处理

2.1.1 过滤非文字的字符

通常情况下，非单词类的字符不带有倾向性，将其过滤，并进行分词

```
TokenList = re.split(r'\W*', text)
```

2.1.2 过滤过短的字符

考虑到 $len(w_i) \leq 2$ 的单词（如 a, is 等）通常并不带有倾向性，故将其过滤，并将所有单词都转换为小写，便于处理

```
if len(token) > 2:
        WordList.append(token.lower())
```

2.2 词袋模型

将单个文本转化成一个 1 * $|V|$ 的向量 $\vec{v}$，其中 V 为词汇表。使用向量来描述文本可以使得实现代码的时候操作较为简便，可以对整个文本进行矩阵的运算。

在词袋模型中，向量 $\vec{v}$ 的第 i 个分量 v_i 描述了单词词汇表 $|V|$ 中第 i 个单词 w_i 出现的次数。由于采用的是多项式模型，这种描述方法便于计算单词出现的次数。

```
def CreateBOWVec(vocabulary, doc):
    vector = zeros(len(vocabulary))
    for word in doc:
        if word in vocabulary:
            vector[vocabulary.index(word)] += 1
    return vector
```

2.3 训练

2.3.1 训练矩阵

利用词袋模型可以将训练集中的所有文本整理成 $|W| * |V|$ 的训练矩阵 T，T_i 为第 i 个文本的词袋向量

2.4 训练过程

训练过程即计算：

$$P(c_i) = \frac{count(all\ words\ in\ c_i) + 1}{count(all\ words\ in\ training\ set) + |C|}$$

$$P(w_j|c_i) = \frac{count(w_j\ in\ c_i + 1)}{count(all\ words\ in\ c_i) + |V_{train}|}$$

虽然这部分代码比较长，但是没有什么特殊的技巧，细节参考代码，在此不再赘述。

为了矩阵计算方便和减少乘法中产生的误差，存储 $log(P(c_i))$ 和 $log(P(w_j|c_i))$，将乘法转换为加法

2.4.1 分类器

分类过程利用:

$$c_{MAP} = argmax\prod_{j=0}^{|W|} P(w_j|c_i) * P(c_i), \quad for\ c_i \in C$$

在这个过程中，我们只关注相对大小，而并不关心具体地数值，所以可以通过取对数的方法将乘法转换为加法，也即：

$$c_{MAP} = argmax\sum_{j=0}^{|W|} log(P(w_j)|c_i) + log(P(c_i)), \quad for\ c_i \in C$$

通过训练已经得到了 $log(P(C))$ 和 $log(P(W|C))$。这里需要指出的是，我们将测试文本也利用 BOW 转化成向量，这样计算 $\sum_{j=0}^{|W|} log(P(w_j)|c_i) + log(P(c_i))$ 可以直接利用向量 $\overrightarrow{log(P(W|c_i))}$ 、向量 $\overrightarrow{W}$、向量 $\overrightarrow{P(C)}$ 进行运算

```
for i in range(ClassNum):
  temp = sum(vector * pWC[i]) + pCi[i]
  if ((prediction == -1) or (temp > maximum)):
    maximum = temp;
     prediction = i;
```

3 测试结果与分析

3.1 测试结果

为了预估分类误差，对训练集提取一小部分，用作交叉验证，email 数据共 50 个训练文档，从中随机抽取了 10 个文档作交叉验证，CSDMC2010_SPAM 数据集中抽取了 $\frac{1}{10}$（共 432 个）文档作交叉验证。

由于 email 数据集的样本个数比较少，所以重复进行了 100 次的 SpamTest 实验，而 CSDMC2010_SPAM 数据集较大，所以一共进行了 10 次的 SpamTest

从实际意义上来看，大家通常都比较关心分类的正确率，以及发生严重错误（把正确的邮件分进了垃圾邮件）里的概率，所以对测试数据统计分类错误概率 P(error) 和严重错误的发生概率 P(H2S|error)

.	P(erro)	P(H2S|error)	运行时间
email	0.0 / 0.1/ 0.2	–	–
100 * email	0.032	0.219	2.01 sec
CSDMC2010_SPAM	0.016	0.714	736.99 sec
10 * CSDM2010_SPAM	0.0243	0.495	5576.234104 sec

Table 1: email 数据集和 CSDM2010 数据集对比

3.1.1 结果分析

通过测试我们可以发现当训练集中的样本个数较少时，仅仅进行一次 SpamTest 所得到的错误率是不够精准的，有的时候 error rate 甚至会到 0.2(这种情况下通常是交叉验证集中选到了两个近乎一样的文档作样本)，需要通过多次测试求平均值来得到一个较为准确的错误率

运行了 10 次的 CSDM2010 中的结果可以发现，H(H2P) 接近 0.5，反映出该算法针对普通邮件被当成垃圾邮件这一问题的处理，并没有特殊的效果
对比 email 数据集合 CSDM2010_SPAM 数据集的错误率,可以发现 CSDM2010_SPAM 数据集的错误率近乎为 email 数据集的 $\frac{1}{2}$。训练样本越丰富，提取出的统计特征就越全面，分类的错误率就越低。

分类算法的错误率还和文本处理时候的规则选择有一定的关系。

比如在 TextParse 过程中，如果把过滤规则改成

```
TokenList = re.split(r'[^a-z0-9]', text)
```

email 数据集的错误率会降低到 0.014 左右，而 CSDMC2010 为 0.024 差别不大。

有的邮件仅仅从英文单词的角度出发是很难判断是不是垃圾邮件的，比如 CSDMC2010 中的这封邮件：

Email Content:
=?KOI8-R?B?QWxla3NleSBWIFN1cmlrb3 YgzsXUIM7BINLBws/UxS4=?=D Qrtxc7RI M7FIMLVxMXUIM7BINLBws/UxSDTICAx Ny4wNS4yMDEwINDP IDE0LjA2LjIwMTAu DQoNCvDPINfP2tfSwd3FzsnJINEgz9TXx d7VIM7BINfB 28Ug08/Pwt3FzsnFLg==

这显然是一封邮件里只包含了一个网址。这样的邮件单从内容上来说，是很难判断是否是垃圾邮件的，判断这样的邮件还需要额外的信息，比如发件人的邮箱地址，比如这个网址里的具体内容等等。

简单的二元分类可能会丢失很多信息，或者增加了一些错误的信息。比如说 TRAIN_02833.eml，虽然这是一封介绍莎士比亚的邮件，但是前后都夹杂了一些和垃圾邮件相似度非常高的小广告，那么仅仅用垃圾邮件/非垃圾邮件来描述这样一封邮件的时候，总是会出现一定的偏差。