Spam Filters and A Star Algorithm

Spam Filter — Bayes

分类

> 根据电子邮件的标题和内容判断其是否为垃圾邮件是一种 **分类** 的应用。
>
> 分类是一个两步走的过程。
>
> 第一步，用类标记已知的 **实例集构建分类器** 。这一步一般发生在训练阶段（学习阶段），由于训练实例的类标记是一致的，所以分类器的构建过程是有导师的学习过程。
>
> 第二步，使用构建好的分类器分类类 **标记未知的实例**。这一步一般发生在测试阶段（工作阶段），一般在分类器被用来预测之前，需要对它的 **分类精度进行评估** 。只有分类准确率达到要求的分类器才可以用来对测试实例进行分类。

朴素贝叶斯

在开始之前，我们先来了解一下下面的几个概念

贝叶斯规则提供了一种计算假设概率的方法，它基于假设的**先验概率**、给定假设下观察到的不同数据的概率以及观察数据本身的先验概率。

> 给定训练实例集 D ，假设空间 H 中最有可能的假设

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

在大多情况下，学习器需要发现给定实例集 D 时可能性最大的假设 h ∈ H (H为候选假设的集合)。这种具有最大可能性的假设被称为**极大后验(Maximum A Posteriori)假设**

$$h_{MAP} = \underset{h \in H}{argmax} P(h|D)$$

朴素贝叶斯（Naive Bayes）分类器假定：在给定实例类标记的前提下，实例的 **属性值之间是相互条件独立** 的。也就是说，在给定实例类标记的情况下，观察到的联合概率正好是 **每个单独属性值概率的乘积。**

> 设每个实例 x 可由属性值的合取描述，而类标记 c 从某有限集合 C 中取值
>
> 根据测试实例 <a1,a2,...,am>，预测它的类标记

$$P(a_1, a_2, \cdots, a_m|c) = \prod_{i=1}^{m} P(a_i|c)$$

综合以上公式，并且由于 P(D) 是一个不依赖于 h 的常量，可以得到以下**朴素贝叶斯分类器的分类公式**

$$c(x) = \underset{c \in C}{argmax} P(c) \prod_{i=1}^{m} P(a_i|c)$$

$$P(a_i|c) = \frac{\sum_{j=1}^{n} \delta(a_{ji}, a_i)\delta(c_j, c)}{\sum_{j=1}^{n} \delta(c_j, c)}$$

> 其中，n 为训练实例的个数；cij 为第 j 个训练实例的类标记；aji为第 j 个训练实例的第 i 个属性
>
> δ(cj , c)是一个二值函数，当cj = c 时其值为 1 ， 否则为 0

在大多情况下，上述这种基于频率比例的方法是对概率的一个良好估计，但是当**接近零频率属性值**出现时，这种概率估计方法就会产生一个有偏差的过低估计概率。为了避免这戏问题，**Laplace估计**常常被用来平滑上述公式得到的概率

$$P(a_i|c) = \frac{\sum_{j=1}^{n} \delta(a_{ji}, a_i)\delta(c_j, c) + 1}{\sum_{j=1}^{n} \delta(c_j, c) + n_i}$$

> ni 为训练实例第 i 个书勋贵的取值个数

课后作业1

> P（正常）= P（垃圾）=50%
>
> 对这封新邮件的内容进行解析，发现其中含有“中奖”这个词，那么这封邮件属于垃圾邮件的概率提高到多少？ 其实就是计算一个条件概率：P(垃圾|中奖)，通过贝叶斯定理 + 全概率公式 可以计算。
>
> 我们假设100封垃圾邮件中有5封包含“中奖”这个词，那么这个概率是5%。 P(中奖|正常)表示所有正常邮件中出现“中奖”的概率，我们假设1000封正常 邮件中有1封包含“中奖”这个词，那么这个概率是0.1%。
> 那么： P(垃圾|中奖) = ？ 误判问题如何解决?

- P(垃圾|中奖)

  $$P(\text{垃圾}|\text{中奖}) = \frac{P(\text{垃圾};\text{中奖})}{P(\text{中奖})} = \frac{P(\text{中奖}|\text{垃圾})P(\text{垃圾})}{P(\text{中奖}|\text{正常})P(\text{正常}) + P(\text{垃圾})P(\text{中奖}|\text{垃圾})P(\text{垃圾})}$$

  将题目中 P(正常) = P(垃圾) = 50% P(中奖|垃圾) = 5% P(中奖|正常) = 0.1%

  所以

  P(垃圾|中奖) = （5% * 50%）/（0.1% * 50% + 5% * 50%）= 0.98039 ≈ 98.04%

- 如何解决误判问题

  在上述的情况下，在邮件包含“中奖”的情况下的推断已经很强了，但是还是存在正常邮件也包含“中奖”的情况。这个时候，我们就需要通过 **多特征判断** 的方法，使我们的推断更加强。当然，在这种情况下，我们需要**朴素贝叶斯**所规定的**属性值之间是相互条件独立**的假设的支持。

垃圾邮件分类模型

根据**朴素贝叶斯分类器的分类公式**（为什么不用除，是因为每种情况的分母一样，一比就会约掉，这利用频率来简单估计）

$$c(x) = \text{垃圾邮件}/\text{正常邮件} \quad \text{两种情况}$$

$$\underset{c \in C}{P(c)} \quad \text{则是在训练时} \quad \text{垃圾邮件}/\text{正常邮件分别的概率}$$

$$\prod_{i=1}^{m} P(a_i|c) \quad \text{则表示当正常}/\text{垃圾时，每封邮件在对应类别邮件会出现该词的概率}$$

Laplace平滑处理

$$\text{对于出现频率不为}0\text{的词语：}$$
$$\text{其}P(a_i|c) = (value + 1)/(2 * spam\_num)$$
$$\text{其}P(a_i|c) = (value + 1)/(2 * normal\_num)$$
$$\text{对于出现频率为}0\text{的词语：}$$
$$\text{其}P(a_i|c) = 1/(2 * spam_num)$$
$$\text{其}P(a_i|c) = 1/(2 * normal_num)$$

进一步优化：

当我们打印最后的概率/频率时，会发现极少数的情况会出现0.0的情况，查看一番原来是因为当该邮件中所有单词出现的频率都不高的时候，那么相乘之后的数会极小，溢出了python的表示范围，这时候我们就需要对所有的频率/概率都做 **log 取对数处理**，实测这样能提高一点正确率

- 结果展示

```
Windows PowerShell
PS E:\PC\桌面\AI> python train.py
train label file:E:\PC\桌面\AI\train-labels.txt
train feature file:E:\PC\桌面\AI\train-features.txt
test label file:E:\PC\桌面\AI\test-labels.txt
test feature file:E:\PC\桌面\AI\test-features.txt
Accuracy:0.980769
PS E:\PC\桌面\AI> python train.py
train label file:E:\PC\桌面\AI\train-labels-50.txt
train feature file:E:\PC\桌面\AI\train-features-50.txt
test label file:E:\PC\桌面\AI\test-labels.txt
test feature file:E:\PC\桌面\AI\test-features.txt
Accuracy:0.542308
PS E:\PC\桌面\AI> python train.py
train label file:E:\PC\桌面\AI\train-labels-100.txt
train feature file:E:\PC\桌面\AI\train-features-100.txt
test label file:E:\PC\桌面\AI\test-labels.txt
test feature file:E:\PC\桌面\AI\test-features.txt
Accuracy:0.915385
PS E:\PC\桌面\AI> python train.py
train label file:E:\PC\桌面\AI\train-labels-400.txt
train feature file:E:\PC\桌面\AI\train-features-400.txt
test label file:E:\PC\桌面\AI\test-labels.txt
test feature file:E:\PC\桌面\AI\test-features.txt
Accuracy:0.957692
PS E:\PC\桌面\AI>
```

可以看到，当我们的训练数据越大的时候，我们的判断就越准确。当训练邮件和测试邮件个数一样的时候，准确率是最低的，仅比 乱猜的50% 的概率要稍高

- 伪代码/代码描述

```
train(lable_file_path,feature_file_pah) //训练的label文件和feature文件的地址
//函数主体
    读取label文件到数组中
```

```
    逐行读取每封邮件每个词语是否出现
        将出现某词语的邮件的次数记录到字典spam_features和normal_features中
        //key为词语，value为出现的邮件的次数
    记录 垃圾邮件和 正常邮件的数量
    循环遍历spam_features和normal_features，将次数转为频率（利用Laplace平滑）

test(test_feature_file_path,test_label_file_path) //测试文件的地址
//函数主体
    读取label文件到数组中
    逐行读取每封邮件每个词语并存到二维数组  //[i][j]，i表示第i封邮件，j表示出现的词语
    循环遍历数组
        根据每封邮件出现的词
            如果词在训练中出现过：            //具体数值请看上一步骤的解释
            如果词没有在训练中出现过：
        分别计算出 c(垃圾邮件) 和 c(正常邮件) 的概率
        比较大小的除本封邮件的判断，与label作比较，得出正确数
    计算正确率
```

垃圾邮件分类模型 version 2

因为上一种情况中，在训练样例较少的情况下，只能获得一个非常差的正确，在上一种情况中，我们只考虑了词语出现在哪些邮件中，并没有考虑到单词出现的次数，所以决定换一个变量，这次我们不以邮件为单位，而是以每个词语为单位。

$$c(x) = \text{垃圾邮件}/\text{正常邮件} \quad \text{两种情况}$$

$$\underset{c\in C}{P(c)} \quad \text{则是在训练时} \quad \text{垃圾邮件}/\text{正常邮件分别的概率}$$

$$\prod_{i=1}^{m} P(a_i|c) \quad \text{则表示当正常}/\text{垃圾时，每个单词在正常}/\text{垃圾词库中出现的概率}$$

Laplace平滑处理

$$\text{对于在对应词库中出现频率不为}0\text{的词语：}$$

$$\text{其}P(a_i|c) = (value+1)/(spam\_word + word\_num)$$

$$\text{其}P(a_i|c) = (value+1)/(normal\_word + word\_num)$$

$$\text{对于在对应词库中出现频率为}0\text{的词语：}$$

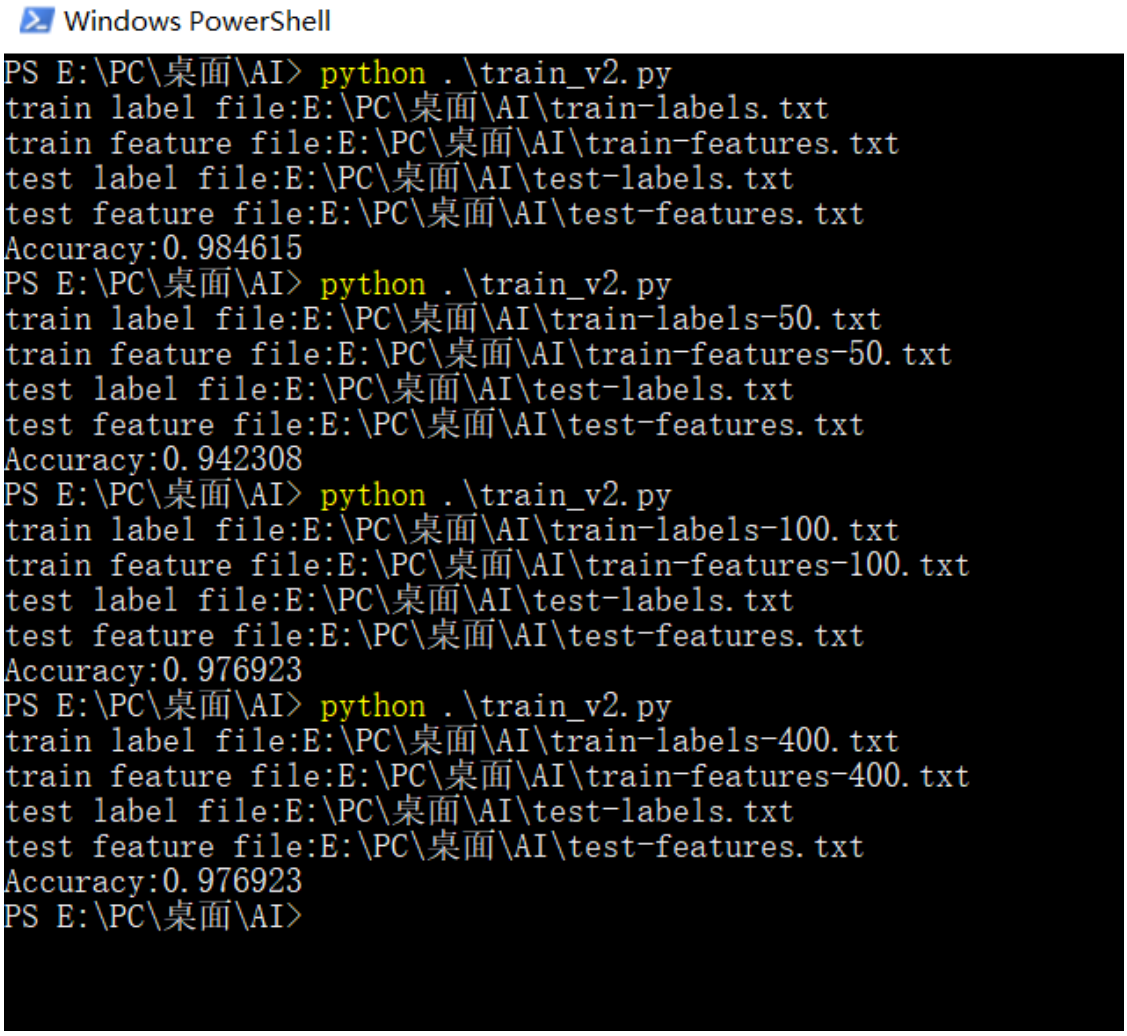
$$\text{其}P(a_i|c) = 1/(spam\_word + word\_num)$$

$$\text{其}P(a_i|c) = 1/(normal\_word + word\_num)$$

最后在计算最后的概率时，因为我们是按单词为单位的，所以这时候也需要把词在每一封邮件的出现次数也算上去，因为我们依然是采用**取对数处理**，所以这里

$$c(x) += numpy.log(P(a_i|c)) * a_i\text{出现的次数}$$

- 结果展示：



可以看到，就算训练样例和测试样例数量差不多，也能得到一个令人比较满意的正确率

- 伪代码（其实与上面区别不大，只是更换了一下计算概率的变量）

```
train(lable_file_path,feature_file_pah) //训练的label文件和feature文件的地址
//函数主体
    读取label文件到数组中
    逐行读取每封邮件每个词语是否出现
        将出现某词语的邮件的次数记录到字典spam_features和normal_features中
        //key为词语，value为该词语的次数
    记录 垃圾邮件和 正常邮件的数量 / 垃圾词库和正常词库数量
    循环遍历spam_features和normal_features，将次数转为频率（利用Laplace平滑）

test(test_feature_file_path,test_label_file_path) //测试文件的地址
//函数主体
    读取label文件到数组中
    逐行读取每封邮件每个词语并存到二维数组  //[i][j]，i表示第i封邮件，j表示出现的词语
    循环遍历数组
        根据每封邮件出现的每一个词
            如果词在训练中出现过：          //具体数值请看上一步骤的解释
            如果词没有在训练中出现过：
        分别计算出 c(垃圾邮件) 和 c(正常邮件) 的概率
        比较大小的除本封邮件的判断，与label作比较，得出正确数
    计算正确率
```

A Star Algorithm

A Star 算法简介

> A Star是一种计算机算法，广泛用于路径查找和图形遍历。该算法有效地在图形上的多个节点或点之间绘制了一条可行的路径。A Star是**Dijkstra算法的扩展**，具有广度优先搜索（BFS）的某些特征。

与Dijkstra一样，A Star通过创建从开始节点到目标节点的最低成本路径树来工作。对于许多搜索来说，不同且更好的地方在于，对于每个节点，A Star 使用一个函数 F(n)，该函数给出使用该节点的路径的总成本的**估计**。因此，A Star是一个启发式函数，所以结果不一定是可证明为正确的。

$$AStar通过以下函数扩展了本来cost已经比较低的路径：$$
$$f(n) = g(n) + h(n)$$
$$f(n)：通过节点n的路径的估计总cost$$
$$g(n)：到目前为止到达节点n的cost$$
$$h(n)：从n到目标的估计cost。这是cost函数的启发式部分，所以它就像一个猜测。$$

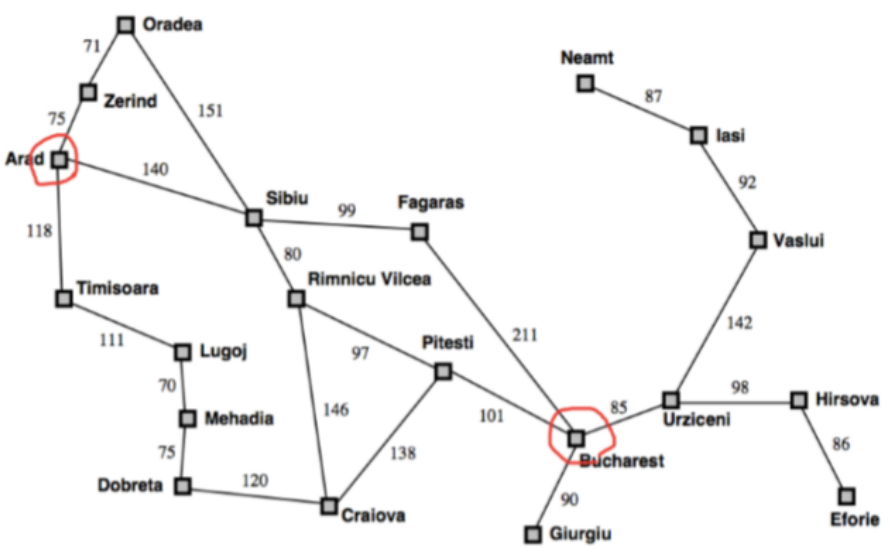
算法步骤总结：

1. 把起点放入openList
2. 取出openList中 F 最小的节点
3. 循环遍历该节点的可行的相邻节点A
    - 如果A在closeList中（表示经过该节点已访问，即存在更短路径），则继续遍历
    - 如果A不在closeList中
        - A已经在openList中，从openList中取出该节点`A'`与`A`的 F 进行比较

          若 当前节点的 g + 当前节点到`A`的cost + h(`A`) < `A'`.F，则更新openList中的该项
        - A不再openList中，则将其添加进去
4. 从第2步开始循环，直到取出目标节点

课后作业2

> 利用A Star算法找到从 A 城市到 B 城市的最短路径，以及代价，其中A Star算法中的h也就是从当前点到目标点的距离已经给出。程序要有通用性，即其中城市A、B可以为下图中任意两个城市。



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- 解决方法

  其实这里只给出了其他城市到Burcharest的直线距离，并没有给出其他各个城市的直线距离，所以我觉得只能以Burcharest为终点，其他任何点作为起点

代码流程参照上一步的算法介绍，几乎完全一样。

这里的h(n)直接用的是 n 到终点 Bucharest 的直线距离