



《人工智能原理实验》 实验报告

(Assignment 1)

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 17 软件工程 5 班

学 生 姓 名 : 张淇

学 号 : 17343153

时 间 : 2019 年 9 月 23 日

成绩：

Assignment 1 : Spam Filters、A*

一. 实验内容

- Bayesians Spam Filters
- A* 算法寻路

二. 实验原理

- Bayesians Spam Filters
 1. 使用两组已经识别好的邮件（一组是正常邮件，另一组是垃圾邮件，这两组邮件的规模越大，训练效果就越好）
 2. 解析所有的邮件，提取每一个词，并计算每个词语在正常邮件和垃圾邮件中出现的频率。
 3. 对于一封未标识的邮件，我们假定它是垃圾邮件的概率为50%。（S = Spam, H = Health）

$$P(S) = P(H) = 50\%$$

4. 使用W表示邮件中的某个词，那么问题就变成了计算P(S|W)的值：

$$P(S|W) = \frac{P(W|S)P(S)}{P(W|S)P(S) + P(W|H)P(H)}$$

5. 因为邮件中往往会有多个词语，为了更准确地判断，我们

选出出现邮件中出现频率最高若干个词（例如15个），然后计算它们的联合概率：

$$P = \frac{P_1 P_2 \cdots P_{15}}{P_1 P_2 \cdots P_{15} + (1 - P_1)(1 - P_2) \cdots (1 - P_{15})}$$

其中：

$$P_n = P(S | W_n)$$

6. 最后，我们再将 P_n 与我们设定好的阈值 P 进行比较进行判断这封邮件是否为垃圾邮件。

● A*算法

伪代码：

- * 初始化open_set和close_set；
- * 将起点加入open_set中，并设置优先级为0（优先级最高）；
- * 如果open_set不为空，则从open_set中选取优先级最高的节点n：
 - * 如果节点n为终点，则：
 - * 从终点开始逐步追踪parent节点，一直达到起点；
 - * 返回找到的结果路径，算法结束；
 - * 如果节点n不是终点，则：
 - * 将节点n从open_set中删除，并加入close_set中；
 - * 遍历节点n所有的邻近节点：
 - * 如果邻近节点m在close_set中，则：
 - * 跳过，选取下一个邻近节点
 - * 如果邻近节点m也不在open_set中，则：
 - * 设置节点m的parent为节点n
 - * 计算节点m的优先级
 - * 将节点m加入open_set中

三. 实验器材

Ubuntu 18.04 Terminal、 VSCode

四. 实验过程与结果

- Bayesians Spam Filters

1. 训练分类器: 将标定好的样本中的信息使用数组存起来(单词编号 -> 索引, 出现次数 -> 索引对应的值), 并计算 $P(W_n|S), P(W_n|H)$, 从而求出 $P(S|W_n)$ 。对应函数为:

```
void init();
```

2. 使用分类器: 读取测试样本, 对测试样本中出现的词按出现次数进行从大到小排序, 再对前 *TEST_SAMPLE_NUM* 个词语求联合概率 X 。若 X 大于规定好的阈值 P , 则认为此邮件为垃圾邮件; 反之则认为这是一封正常邮件。对应函数为:

```
void test();
```

3. 比较上述输出与真实答案, 并统计判断错误的个数, 求出判断垃圾邮件的准确率。对应函数为:

```
void calculate();
```

4. 实验结果:

```

record.txt
25 10 0.4 0.8 13
26 10 0.4 0.82 13
27 10 0.4 0.84 11
28 10 0.4 0.86 11
29 10 0.4 0.88 10
30 10 0.4 0.9 11
31 10 0.4 0.92 9
32 10 0.4 0.94 9
33 10 0.4 0.96 9
34 10 0.4 0.98 7
35 10 0.4 0.99 6
36 10 0.4 0.999 8
37 10 0.4 0.9999 17
38 10 0.4 1 130
39
40 -BEST
41 10 0.1 0.99 4

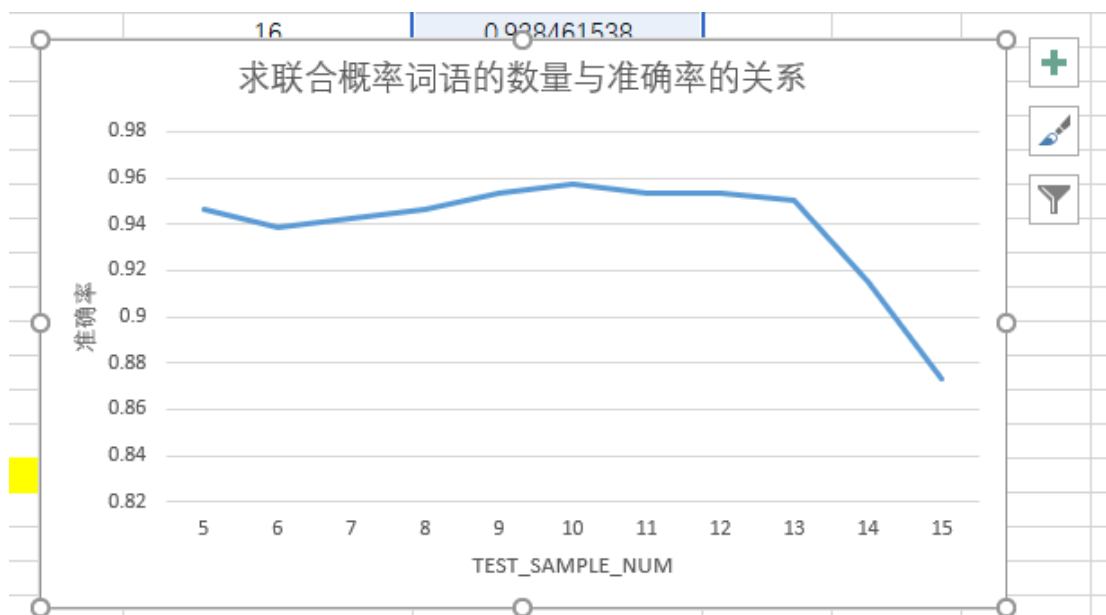
```

5. 在上述过程中，一共有三个参数，下面分析它们对结果的影响：

初始值 (10, 0.4, 0.9) , 准确率0.957692308

1) 求联合概率的词语的数量 $TEST_SAMPLE_NUM$:

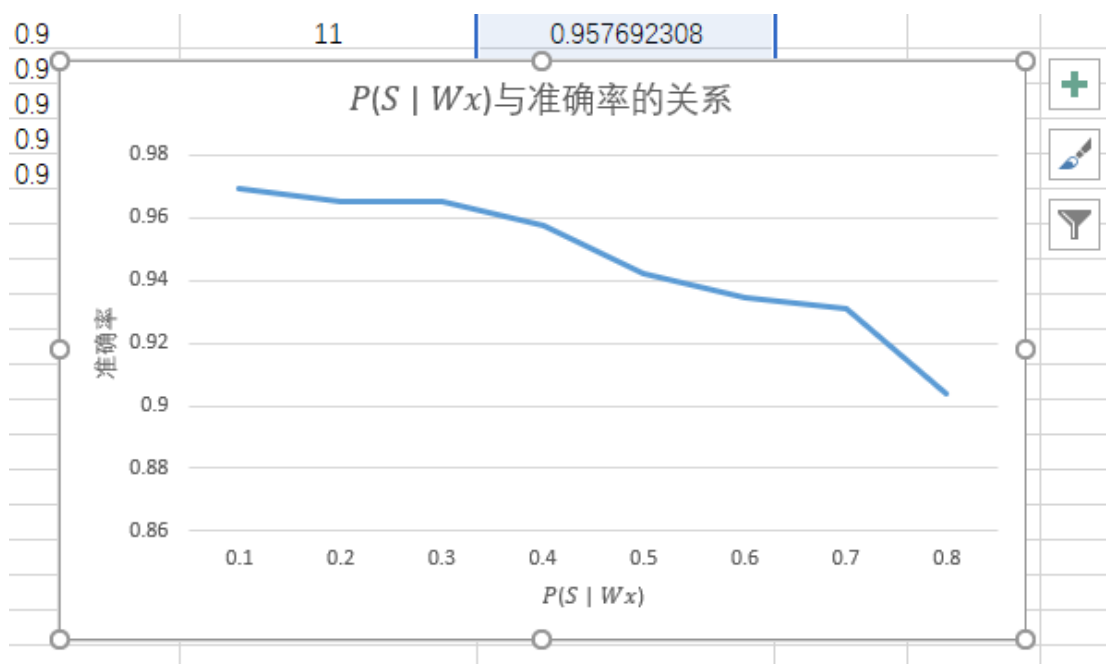
通过控制变量法得出其对准确率的影响，拟合的曲线如下图所示：



2) 当样本中未出现过词语 W_x 时，假设有这个词语的情况下邮件未垃圾邮件的概率 $P(S | W_x)$

通过控制变量法得出其对准确率的影响，拟合的曲线如

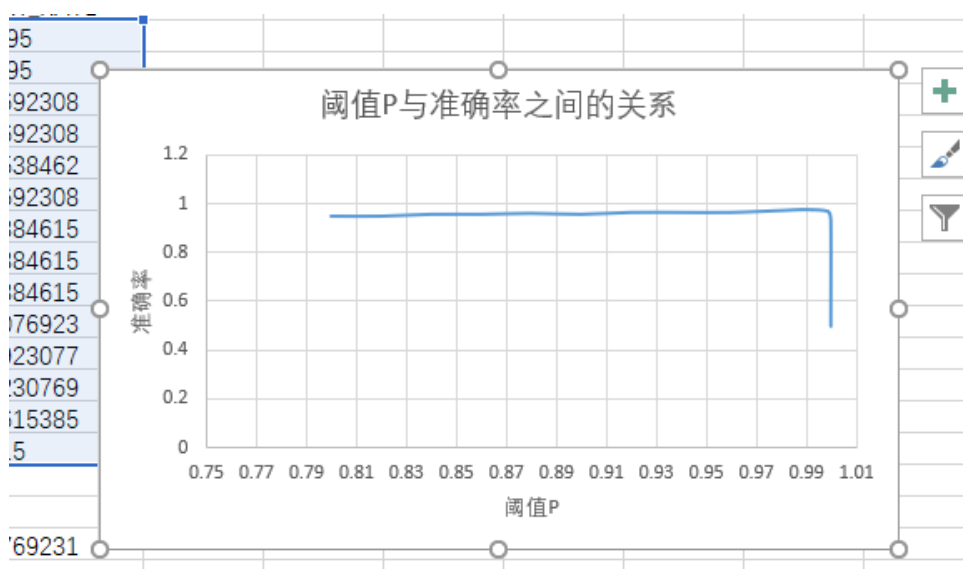
下图所示：



3) 与联合概率比较进行判断的阈值P

通过控制变量法得出其对准确率的影响，拟合的曲线如

下图所示：

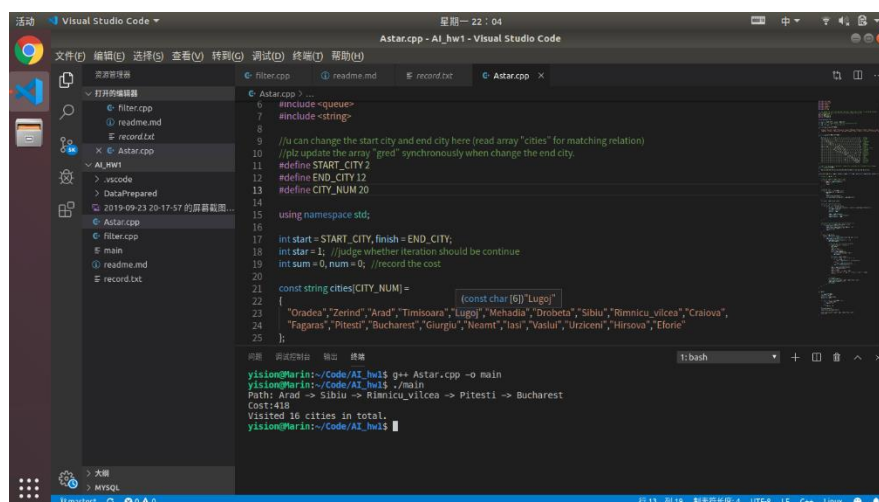


(注：因为程序中计算概率中涉及到大量的浮点运算，存在误差累积，因此 $P == 1$ 时准确度会直线下降但不为0)

4) 通过三个参数测试得出的数据,我们将控制变量法中表现最好的三个参数整合在一起 (10, 0.1, 0.99) , 最后得出的准确率为: 0.980769231。

● A* 算法寻路

1. 初始化: 使用二维数组存放各个城市之间的“真实距离”, 使用一维数组存放各个城市到 *Bucharest* 的“估计距离”。
2. 将出发城市装入vector中、并压入栈, 将与其相邻的城市加入集合S中。
3. 计算到达S中各个城市的距离 $f(n)$, 以及加上到达 *Bucharest* 的“估计距离” $g(n)$ 得出 $cost$, 将 $cost$ 值最小对应的城市压入栈中, 更新集合S为与此城市相邻的城市。
4. 循环步骤3直到抵达目的城市 (*Bucharest*) 中, 输出路线、总共的 $cost$ 以及过程中访问的点的数量。
5. 实验结果:



The screenshot displays the Visual Studio Code editor with a C++ file named `Astar.cpp` open. The code implements the A* search algorithm for finding the shortest path between cities. It includes headers for `vector`, `queue`, and `string`. The `main` function initializes the start city as `START_CITY` (2) and the end city as `END_CITY` (12). It defines a constant array of city names and a function to calculate the cost of a path. The output window shows the execution results:

```
yision@Marin:~/Code/AI_hw1$ g++ Astar.cpp -o main
yision@Marin:~/Code/AI_hw1$ ./main
Path: Arad -> Sibiu -> Rimnicu_vilcea -> Pitesti -> Bucharest
Cost: 418
Visited 16 cities in total.
yision@Marin:~/Code/AI_hw1$
```