决策树实现文本分类报告

刘浚哲
北京大学物理学院 *1500011370*

March 18, 2018

1. 决策树简要介绍[1]

决策树是用样本的属性作为结点, 用属性的取值作为分支的树结构. 决策树的叶结点是样本的类别值. 它是根据特征（feature）的值逐步把数据分类, 直到所有的叶子节点下包含的样本类型属于同一个类型结束. 决策树本质上是一种贪心算法. 决策树是一种知识表示形式, 它是对所有样本数据的高度概括决策树能准确地识别所有样本的类别, 也能有效地识别新样本的类别.

决策树的核心思想是通过不同的特征划分方式将数据逐一通过不同的树枝流向不同的叶子节点, 根据不同的策略对同一个数据集可以构成不同的决策树. 根据信息熵的定义, 产生了 ID3 构建决策树方法. 信息熵的定义为:

$$Entropy(S) = -\sum_{i=1}^{n} p(u_i)\log(p(u_i)) \tag{1}$$

其中 S 是样例的集合，$p(u_i)$ 是类别 i 出现的概率. 另外定义信息增益为:

$$InfoGain(S, f) = Entropy(S) - \sum_{v \in f} \frac{|S_v|}{|S|} Entropy(f) \tag{2}$$

其中 f 为属性, v 是 f 的某一属性值， S_v 是 S 中 f 的值为 v 的样例集合， $|S_v|$ 为 S_v 中所含样例数. 决策树算法每次建立新的节点时, 通过选择信息增益最大的特征 f 作为分类节点, 进行递归操作, 直至分隔到叶节点.

2. 文本分类问题

本次我们需要利用决策树解决的问题是文本分类问题, 即输入一个文本, 需要预测其文本的类别. 一般来说文本分类问题包含以下操作:

2.1. 预处理

预处理一般包含: 分词, 词性标注, 停用词过滤, 词频统计等. 一般来说文本分类使用的是词特征，而文本是一段长字符串的集合, 所以我们需要对文章进行分词, 将文本划分为多个字词的集合. 对于一段文本来说, 比如' 的',' 了', ' 日', ' 月', ' 年' 等不具有任何类别表征能力的字词，是一定需要去除的, 这就需要构建停用词表, 将文本中出现的停用词去除. 而最重要的词频统计则是我们后面进行特征提取、特征权值计算的基础.

2.2. 特征选择

这一步是用词做属性特征 (feature). 注意到决策树所用到的数据类型是结构化的数据; 而对于不同的文本来说, 每篇文章所包含的字词可能是完全不一样的. 如果不做特征选择这一步的话, 很容易出现上万

[1] *Email Address:1500011370@pku.edu.cn*

维、甚至几十万维的属性, 这么多的维度对计算来说可能就是个灾难. 而因为真正对分类起作用的词, 可能就只是一少部分, 所以一般将维度选择在 1000 5000 维, 而这个和特征选择的方法有很大关系.

特征选择有很多方式, 最常用的包括: 信息增益 (Information Gain, IG), 熵 (Entropy), 相对熵 (Relative Entropy),χ^2 统计量 (Chi-Square) 等. 一般来说我们在预处理当中的词频统计的基础上, 可以进一步对统计结果进行特征选择的操作, 而选择各类文本中最具有代表性的字词, 来作为较有区分性的属性, 从而达到降维的作用.

2.3. 训练集, 测试集的划分

在获取文本库之后, 需要对整个文库做训练集与测试集的划分, 并保证二者之间没有交集, 否则并不能达到验证准确率的效果. 划分方式有手动划分, 与交叉验证 (cross validation) 的方法. 交叉验证法是先将数据集 D 划分为 k 个大小相近的互斥子集, 每个子集 D_i 尽可能保持数据分布一致性, 即从 D 中通过分层采样得到. 然后每次用 k 个子集的并集作为训练集, 余下的那个子集作为测试集; 这样可以获得 k 组训练/测试集, 从而进行 k 次训练测试, 最终返回 k 个测试结果的均值.

由于交叉验证法计算量较大, 因此本文使用手动划分后的训练/测试集进行训练与验证.

2.4. 文本结构化

在选择好我们的 feature 并划分完训练集之后, 我们可以对训练集当中的训练文本进行结构化的处理, 即重新统计各个文本在各个 feature 下的值. 通过此步, 我们可以将非结构的文本结构化, 并由此构造一个数据集的矩阵, 后续的决策树算法则由该矩阵进行进一步的操作.

2.5. 分类模型训练与验证

在获取数据集矩阵之后, 我们就可将这个矩阵对我们的模型进行训练. 训练完成之后, 再将测试集当中的文本输入分类模型, 并对比自身的 label 是否与输出的 predication label 相符, 并由计算得到的准确率来描述模型的性能.

3. 决策树实现文本分类模型:

3.1. 文本库来源:

来自于助教所提供的"new-weibo-13638", 下分 9 个类别: 教育, 娱乐, 财经, 健康, 军事, 证券, 体育, 科技. 共 13191 篇文本. 其中教育 445 篇, 娱乐 2255 篇, 财经 2375 篇, 健康 670 篇, 军事 791 篇, 科技 1397 篇, 体育 3325 篇, 证券 1167 篇, 房产 1211 篇.

3.2. 训练集, 测试集的划分:

本文采用手动划分的方法, 将 9 个类别当中的文档随机划分为训练集与测试集, 得到训练集: 教育 400 篇, 娱乐 1900 篇, 财经 2055 篇, 健康 570 篇, 军事 690 篇, 科技 1300 篇, 体育 2900 篇, 证券 990 篇, 房产 1050 篇, 共 11855 篇训练文本; 测试集为: 教育 45 篇, 娱乐 355 篇, 财经 320 篇, 健康 100 篇, 军事 101 篇, 科技 97 篇, 体育 425 篇, 证券 177 篇, 房产 161 篇, 共 1781 篇测试文本. 分别保存在"new-weibo-13638" 下的两个子文件夹: "trainsets" 与"testsets" 下.

3.3. 文本词频统计:

为了提取每个类别的代表词, 来构建一个最终的 feature. 我们需要对训练集当中的各类文本进行词频统计, 将各种类的文本特征词提取出来之后, 再反过来对整个训练集之中的文本重新进行一次特征统计 (向量结构化).

考虑到每个文本都已经分好了词, 我们不需要再额外进行分词的操作, 只需要读取文本 txt 文件, 并以其中的制表符'/t' 为分隔符, 读取文本之中的单词 word, 保存在字典 dict 当中, 其键值 value 就是单词 word 所出现的词频. 实现代码如下:

```
import os
import re
import sys


def process(path):
    catelist = os.listdir(path) # 获取seg_path下的所有子目录, 也就是分类信息
    for category in catelist: # 遍历每一个种类文件夹
        dict = {} # 为每一个种类文件夹都创建一个空字典
        class_path = path + category + "/" # 拼出分类子目录的路径
        file_list = os.listdir(class_path) # 获取class_path下的所有文件
        for file in file_list: # 遍历类别目录下文件
            fullname = class_path + file
            with open(fullname, mode='r', encoding='utf-8') as inFile:
                word = '' # 空字符串以便于连接字符
                for char in inFile.read():
                    if char != '\t': # 工作原理: 一个字符一个字符地读, 将不是制表符的字符丢进 word 字符串当中
                        word += char # 连接字符
                    else: # 读到了制表符, 就从 word 字符串当中读取一个单词, 进行词频统计!
                        if word in dict:
                            dict[word] += 1
                            word = '' # 将word置为空, 否则, word值无限增大
                        else:
                            dict.setdefault(word, 1)
                            word = ''

        wordfreq = "/Users/macbookair/iCloud/Desktop/Daily/Second Major/CS/Machine Learning/hw1/
            new_weibo_13638/" + category + ".txt"
        with open(wordfreq, mode='w', encoding='utf-8') as outFile:
            for word, freq in dict.items():
                s = '{0}\t{1}\n'.format(word, freq)
                outFile.write(s)


if __name__ == "__main__":
    path = "/new_weibo_13638/Train_Set/"
    process(path)
```

最后, 在训练集文件夹下我们可以得到 9 个类别的词频统计文件: category.txt.

3.4. 文本结构化

在获取 9 个类别各自的词频统计之后, 我们还是不能对整个训练集有一个直观的感受. 因为 9 个类别当中出现的词各不尽相同, 在教育类出现多的词" 黑板" 在军事类文本可能不会出现, 而军事类当中的" 坦克" 也不会在证券类文本当中出现. 同样地, 我们也无法了解一个词在各个文本类型当中出现的次数, 例如" 中国" 一词在各类文本当中出现的次数必定不尽相同. 因此我们需要对整个训练集进行统一的统计.

我们利用 excel 表格来创建一个直观的统计表格. python 当中的 xlrd 模板可以方便地读取 xls 类文件, xlwt 模板和 xlsxwriter 可以完成写入 excel 表格数据的功能, 而 openpyxl 则可完成大量的 xlsx 数据读写功能. 我们将 9 个 txt 文件的内容拷贝进入一个 xls 文件中 9 个各自的表格. 用 9 个 dict 字典来保

存每个类型所出现的单词 key 与键值 value(即词频), 并构建一个总的单词表 Words, 利用其不会添加重复元素的特质来将训练集中出现的所有单词进行统计; 在这个过程中我们同时也完成了停用词的去除: 因为在预处理的过程中, 我们可以发现一些诸如" 年, 月, 日" 等无特殊意义的普通词的频数较高, 但其不能表达特殊的类别, 因此我们也将其去掉; 同样地, 我们也不统计出现的数字. 实现代码如下:

```
import os
import xlrd
import xlwt
import xlsxwriter
import openpyxl


stopwords = {'的', '年', '月', '日', '时', '不', '岁', '中', '说', '称', '做', '高', '达', '新', '未', '名'}


def open_excel(file_path):
    file = xlrd.open_workbook(file_path)
    return file


def excel_process(file_path):
    file = open_excel(file_path)

    Words = set()
    dict0 = {}
    dict1 = {}
    dict2 = {}
    dict3 = {}
    dict4 = {}
    dict5 = {}
    dict6 = {}
    dict7 = {}
    dict8 = {}

    sheet = file.sheet_by_index(0)
    nrows = sheet.nrows
    for rownum in range(0, nrows):
        row = sheet.row_values(rownum)
        word = row[0]
        dict0[word] = row[1]
        if word not in stopwords:
            Words.add(word)
    num0 = dictsum(dict0)
    print(num0)

   # 对其余8个类别进行同样处理

    TotalNum = num0 + num1 + num2 + num3 + num4 + num5 + num6 + num7 + num8 # 总频数
    print(TotalNum)
```

在获取了各种类文本各词的词频 $dict_i$, 与总词表 $wordlist$ 之后, 我们可以通过遍历 wordlist 之中的单词, 来计算各词在各类文本中出现的词频, 若不出现该词则初始化为 0. 代码如下:

```
    wordbook = xlwt.Workbook(encoding = 'utf-8')
    wordsheet = wordbook.add_sheet('Aggregate')
```

```
    wordsheet.write(0, 1, 'Education')
    wordsheet.write(0, 2, 'Entertainment')
    wordsheet.write(0, 3, 'Finance')
    wordsheet.write(0, 4, 'Health')
    wordsheet.write(0, 5, 'Military')
    wordsheet.write(0, 6, 'RealEstate')
    wordsheet.write(0, 7, 'Securities')
    wordsheet.write(0, 8, 'Sports')
    wordsheet.write(0, 9, 'Tech')


    n = 1
    for key in Words:
        wordsheet.write(n, 0, key)
        wordsheet.write(n, 1, dict0.get(key, 0))
        wordsheet.write(n, 2, dict1.get(key, 0))
        wordsheet.write(n, 3, dict2.get(key, 0))
        wordsheet.write(n, 4, dict3.get(key, 0))
        wordsheet.write(n, 5, dict4.get(key, 0))
        wordsheet.write(n, 6, dict5.get(key, 0))
        wordsheet.write(n, 7, dict6.get(key, 0))
        wordsheet.write(n, 8, dict7.get(key, 0))
        wordsheet.write(n, 9, dict8.get(key, 0))
        num = dict0.get(key, 0) + dict1.get(key, 0) + dict2.get(key, 0) + dict3.get(key, 0) + dict4.get(key,
              0) + dict5.get(key, 0) + dict6.get(key, 0) + dict7.get(key, 0) + dict8.get(key, 0)
        wordsheet.write(n, 10, num) # 某单词在所有类型的文本中出现词频之和
        n += 1
```

得到的词频统计表格截屏如下:

	A	B	C	D	E	F	G	H	I	J	K	L
1		Education	Entertainment	Finance	Health	Military	RealEstate	Securities	Sports	Tech		
2	中国	31	111	648	27	654	201	213	344	276	2505	
3	公司	10	62	309	2	3	78	393	19	380	1256	
4	北京	17	100	182	23	15	196	28	543	89	1193	
5	新浪	14	150	89	47	35	15	55	591	122	1118	
6	时间	24	80	45	39	30	25	18	641	76	978	
7	视频	1	109	23	2	100	6	9	633	91	974	
8	市场	6	3	237	10	3	251	263	12	134	919	
9	美国	39	68	126	17	276	18	15	170	168	897	
10	分钟	1	9	8	35	12	3	1	742	8	819	
11	体育	3	7	7	0	0	1	16	741	2	777	
12	比赛	0	8	0	0	34	2	2	575	4	625	
13	企业	2	3	345	9	5	49	109	2	72	596	
14	亿	2	23	228	1	1	74	190	34	39	592	
15	万	15	33	203	5	7	76	37	154	42	572	
16	消息	5	66	113	7	26	40	92	96	95	540	
17	发布	11	32	84	8	21	44	93	28	219	540	
18	投资	3	3	178	0	2	98	165	3	82	534	
19	报道	8	85	68	12	108	18	20	117	90	526	
20	前	4	51	96	16	33	40	45	172	55	512	
21	调查	16	39	276	13	3	29	35	20	78	509	
22	亿元	1	2	134	1	3	61	161	0	112	475	
23	手机	7	19	38	15	0	7	57	113	214	470	
24	城市	7	3	178	1	7	229	8	7	28	468	
25	集团	2	6	182	2	2	45	108	4	117	468	
26	近日	10	106	109	10	56	50	23	43	51	458	
27	今日	0	79	90	32	14	17	82	17	124	455	
28	元	20	11	221	6	0	75	67	5	41	446	
29	经济	0	4	247	3	6	72	75	2	27	436	

图 1: 训练集文本词频统计表格示意图, 最右侧一栏为求和

这样我们就得到了训练集中所有出现的词及在各类文本中出现的频数分布. 通过观察表格可知, 这个表格其实无法完全反映一个词是否对一类文本具有较好的区分度, 一方面词频是受到文本个数的影响, 另

一方面词频高也代表着这个词在很多类文本中都有出现, 说明该词并不能很好地用于区分文本. 虽然简单地取排序后的前 900 个词作为 feature 也能完成分类功能, 但其每个节点的区分度必定不会很大, 导致树的高度增加, 分类能力下降. 为此, 我们有必要进一步处理文本, 挑选出每一类文本当中真正的" 特性词".

3.5. 特征提取–χ^2 统计量

为描述某一个词对于某一类文本的" 分类贡献" 大小, 我们采用 χ^2 统计量的方法来定量描述. χ^2 的主要思想是认为词条与类别之间符合 χ^2 分布, 若词条的 χ^2 统计量越大, 则词条与类别之间的独立性越小, 相关性就越强, 即词条对此类别的贡献也就越大. 一般传统的词条 i 对类别 j 的 χ^2 统计量计算公式如下:

$$\chi_{ij}^2 = \frac{n \cdot (n_{11}n_{22} - n_{12}n_{21})^2}{n_{11}^2 \cdot n_{22}^2 \cdot n_{12}^2 \cdot n_{21}^2} \tag{3}$$

其中 n_{11} 表示词条 i 在类别 j 中出现的频数,n_{12} 表示词条 i 在除去类别 j 以外的其他类别中出现的频数,n_{21} 表示除词条 i 之外的其他词条在类别 j 中出现的频数,n_{22} 表示除词条 i 外的其他词条在除类别 j 以外的其他类别中出现的频数,n 表示所有词条的频数总和.

而这种传统的卡方统计量有一个缺陷: 即实际上只能说明词条对某一类别的贡献程度, 而不能说明词条与类别的相关性. 这是由于词条与类别的相关性是有正反两个方向的: 当 $(n_{11}n_{22} - n_{12}n_{21}) > 0$ 时, 词条与类别呈正相关, 卡方统计量的绝对值越大时, 说明当词条 i 出现时, 该文本属于类别 j 的可能性也越大; 反之若 $(n_{11}n_{22} - n_{12}n_{21}) < 0$ 时, 词条则与类别呈负相关, 此时卡方统计量的绝对值越大时, 说明词条 i 出现时, 该文本不属于类别 j 的可能性也越大!

为了描述这种相关性的正负性质, 我们采用修改后的 χ^2 统计量:

$$\chi_{ij}^2 = sign(n_{11}n_{22} - n_{12}n_{21}) \cdot \frac{n \cdot (n_{11}n_{22} - n_{12}n_{21})^2}{n_{11}^2 \cdot n_{22}^2 \cdot n_{12}^2 \cdot n_{21}^2 + 1} \tag{4}$$

其中 $sign(x)$ 函数为一个表示其符号的函数, 若 $x > 0$, 则其值为 1, 若 $x < 0$, 则其值为 -1. 分母 $+1$ 的原因, 则是考虑到了某一个词有可能不在该类文本中出现的可能性, 即 $n_{11} = 0$, 而这会导致分母为 0, 因此我们加一来避免这种情况.

同样地, 我们对每一个类别单独计算 χ^2 统计值之后, 再将各个表汇总得到总体的 χ^2 分布值, 以期得到区分度较大的词. 实现代码如下:

```
    CHI0 = wordbook.add_sheet('Education')
    CHI1 = wordbook.add_sheet('Entertainment')
    CHI2 = wordbook.add_sheet('Finance')
    CHI3 = wordbook.add_sheet('Health')
    CHI4 = wordbook.add_sheet('Military')
    CHI5 = wordbook.add_sheet('RealEstate')
    CHI6 = wordbook.add_sheet('Securities')
    CHI7 = wordbook.add_sheet('Sports')
    CHI8 = wordbook.add_sheet('Tech')
    CHI = wordbook.add_sheet('X^2')

    n = 0
    for key in Words:
        CHI0.write(n, 0, key)
        CHI.write(n, 0, key)
        num = dict0.get(key, 0) + dict1.get(key, 0) + dict2.get(key, 0) + dict3.get(key, 0) + dict4.get(key,
              0) + dict5.get(key, 0) + dict6.get(key, 0) + dict7.get(key, 0) + dict8.get(key, 0)
        n11 = dict0.get(key, 0) # 可能为零! 因为该词条可能不出现在某一种类当中
        n12 = num - n11
        n21 = num0 - n11
        n22 = TotalNum - num0 - num + n11
```

```
21        X = sign(n11*n22-n12*n21)*TotalNum*(n11*n22-n12*n21)*(n11*n22-n12*n21)/(n11*n11*n22*n22*n12*n12*n21*
              n21+1)
22        # 分母+1防止其为0
23        CHI0.write(n, 1, X)
24        CHI.write(n, 1, X)
25        n += 1
26
27        # 对其余8个类别进行同样处理
```

得到各类文本卡方统计量前 20 如下:

	A	B
1	暖文	7.7277E+2
2	卧谈	2.555E+20
3	茶语	1.7436E+20
4	实习	9.4664E+19
5	面试	4.8298E+19
6	校方	4.8298E+19
7	考研	3.0911E+19
8	报考	3.0911E+19
9	中小学生	2.3666E+19
10	夏令营	2.3666E+19
11	班主任	2.3666E+19
12	科大	2.3666E+19
13	小演	1.7387E+19
14	语文	1.7387E+19
15	考题	1.7387E+19
16	投档	1.7387E+19
17	分数线	1.7387E+19
18	论文	1.2074E+19
19	拉手	1.2074E+19
20	培训班	1.2074E+19

(a) 教育类

	A	B
1	上映	2.6251E+21
2	威尼斯	1.25473E+21
3	周星驰	8.46625E+20
4	涉毒	7.74606E+20
5	电影节	7.74606E+20
6	饰演	6.72579E+20
7	周杰伦	6.72579E+20
8	赵薇	5.47746E+20
9	执导	5.18538E+20
10	王全安	4.9013E+20
11	获释	4.35716E+20
12	高虎	4.09709E+20
13	新片	3.60096E+20
14	吴镇宇	3.60096E+20
15	港姐	3.36489E+20
16	柴智屏	3.36489E+20
17	绮	3.13683E+20
18	金贤重	2.91677E+20
19	邓超	2.91677E+20
20	刘诗	2.70472E+20

(b) 娱乐类

	A	B
1	吴英	3.12206E+2
2	4S店	3.12206E+20
3	东阳	2.71966E+20
4	速读	1.67897E+20
5	娃哈哈	1.52981E+20
6	宗庆后	1.38758E+20
7	农夫山泉	1.38758E+20
8	长庆	1.25229E+20
9	吴英案	1.00253E+20
10	东阳市	1.00253E+20
11	张新明	8.88051E+19
12	车主	8.88051E+19
13	中纪委	7.80514E+19
14	怡宝	7.80514E+19
15	吕梁	7.80514E+19
16	物价局	7.80514E+19
17	煤老板	6.79914E+19
18	药典	6.79914E+19
19	长庆油田	5.86253E+19
20	日立	4.99529E+19

(c) 财经类

	A	B
1	特邀	4.8666E+2
2	粥	3.99688E+20
3	性福	3.99688E+20
4	肝脏	3.21272E+20
5	便秘	2.73746E+20
6	主任医师	2.51409E+20
7	清热	2.09587E+20
8	详	1.71567E+20
9	牙	1.53982E+20
10	干燥	1.53982E+20
11	肠道	1.37348E+20
12	葡萄	1.21665E+20
13	润肺	1.21665E+20
14	发病率	1.06932E+20
15	坚果	1.06932E+20
16	银耳	1.06932E+20
17	讲堂	1.06932E+20
18	肠胃	1.06932E+20
19	胃癌	9.31497E+19
20	腹部	9.31497E+19

(d) 健康类

	A	B
1	坦克	2.44767E+2
2	美军	6.82729E+21
3	解放军	6.82729E+21
4	歼	4.40535E+21
5	南海	3.55217E+21
6	潜艇	2.44441E+21
7	空军	1.87883E+21
8	俄军	1.59673E+21
9	96A	1.10134E+21
10	侦察	9.28867E+20
11	防空	8.88043E+20
12	核潜艇	8.88043E+20
13	钓鱼岛	8.09147E+20
14	军演	5.61907E+20
15	侦察机	4.40811E+20
16	礁	3.85767E+20
17	东海	3.85767E+20
18	北约	3.59621E+20
19	驱逐舰	3.59621E+20
20	抵近	3.34392E+20

(e) 军事类

	A	B
1	乐居	3.42129E+2
2	实惠	1.94833E+20
3	早报	1.43143E+20
4	万平	7.46639E+19
5	违建	5.34576E+19
6	抢房	4.41798E+19
7	宋卫平	3.57857E+19
8	房地产商	2.16481E+19
9	普通住宅	1.59047E+19
10	华侨城	1.1045E+19
11	客源	1.1045E+19
12	调整期	1.1045E+19
13	克而瑞	1.1045E+19
14	房企拿地	1.1045E+19
15	中海	1.1045E+19
16	嫂	1.1045E+19
17	度假区	1.1045E+19
18	平方英尺	1.1045E+19
19	全城	1.1045E+19
20	宅地	1.1045E+19

(f) 房产类

图 2.1: 各类文本 χ^2 统计量前 20 词条

	A	B
1	个股	1.34803E+21
2	张晓军	1.25346E+21
3	指报	9.90386E+20
4	三板	6.87768E+20
5	深成指	6.5381E+20
6	老艾	4.40172E+20
7	做市	4.13091E+20
8	转增	4.13091E+20
9	次新股	2.90582E+20
10	湘财	2.90582E+20
11	中签率	2.47597E+20
12	优先股	2.47597E+20
13	定增	2.0805E+20
14	冻结资金	1.71942E+20
15	重仓股	1.39273E+20
16	高开	1.24228E+20
17	盘面	9.67174E+19
18	持仓	9.67174E+19
19	退市	9.67174E+19
20	资产重组	8.42516E+19

(g) 证券类

	A	B
1	破门	5.68277E+22
2	进球	5.25903E+22
3	曼联	4.5812E+22
4	皇马	3.95012E+22
5	赛季	2.88562E+22
6	英超	1.9879E+22
7	切尔西	1.86222E+22
8	阿森纳	1.45464E+22
9	巴萨	1.24431E+22
10	曼城	1.09734E+22
11	助攻	1.01586E+22
12	客场	9.93165E+21
13	米兰	9.59595E+21
14	点球	9.48534E+21
15	比分	6.64846E+21
16	利物浦	6.37432E+21
17	扳平	6.28423E+21
18	亚运	5.50223E+21
19	李娜	4.24025E+21
20	欧冠	4.02191E+21

(h) 体育类

	A	B
1	谷歌	2.8404E+21
2	iOS	7.45162E+20
3	Android	3.31183E+20
4	收为	3.07949E+20
5	利滚利	2.85561E+20
6	Twitter	2.04455E+20
7	Xbox	1.8629E+20
8	出货量	1.8629E+20
9	搜狐	1.8629E+20
10	张朝阳	1.8629E+20
11	铁塔	1.52496E+20
12	张亚勤	1.36866E+20
13	索尼	1.22081E+20
14	网易	1.22081E+20
15	酷派	1.08141E+20
16	孙正义	1.08141E+20
17	惠普	9.50461E+19
18	LTE	8.27957E+19
19	Note	8.27957E+19
20	软银	7.13902E+19

(i) 科技类

图 2.2: 各类文本 χ^2 统计量前 20 词条

从统计结果看来, 利用卡方统计值来计算每个类别的特征词还是刻画的比较好的, 例如娱乐类的” 周杰伦, 邓超”, 军事类的” 坦克, 核潜艇”, 体育类的” 巴萨, 欧冠” 等等.

文本处理到这里, 我就有一些对于特征词 feature 选择的思考了:

1. 单纯地对每个类别分别选择卡方值较大的词, 而后合并起来作为 feature 集合
2. 像之前统计词频一样, 将各个词条对各类别的卡方值统一列成一个表, 处理数据后进行排序, 而后选择 feature

我一开始比较偏向于第二种方法, 是因为第一种方法只片面地考虑了卡方值为正的词条, 而没有考虑为负值的词条, 从而在某些词条出现时, 并不能根据其负卡方值来排除一些类别的可能性. 所以我对于第二种方法进行了尝试. 一个简单的想法是: 将某一词条对 9 个类别的卡方值求和, 所得到的和进行升序排列 (即从负值排到正值), 取最小的前几个词条组成 feature. 这么做有几个好处:

1. 理论上说, 若该词条只出现在某一类别的文本中时, 只有该类别的卡方值强烈地为正, 而其他类别的卡方值强烈地为负, 因此相加之后将会是一个比较大的负值
2. 若该词条有多个类别的卡方值为正值, 但由于其和值强烈地为负, 因此其余几类的卡方值也强烈地为负, 由此可强烈地排除这几类的可能性.

由此操作, 可得到词条卡方值分布如下, 最后一列为和值:

	A	B	C	D	E	F	G	H	I	J	K
1		Education	Entertain	Finance	Health	Military	RealEstate	Securities	Sports	Tech	
2	比赛	-4.12286E+20	-2.13658E-08	-7.49056E+21	-5.66091E+20	-6.18285E-11	-3.4608E-07	-3.59161E-07	9.63957E-09	-8.67818E-08	-8.46894E+21
3	证监会	-1.99718E+20	-1.70471E+21	-8.43927E-09	-2.74224E+20	-4.76492E+20	-1.41411E-07	4.39578E-07	-4.9182E+21	-1.49134E-06	-7.57334E+21
4	房地产	-1.97886E+20	-1.68907E+21	8.91285E-11	-2.71708E+20	-4.7212E+20	6.34271E-09	-2.07335E-09	-4.87308E+21	-3.58093E-07	-7.50386E+21
5	楼市	-1.64677E+20	-1.40561E+21	7.34411E-11	-2.2611E+20	-3.9289E+20	8.71496E-09	-1.2254E-09	-4.05529E+21	-9.37385E+20	-7.18197E+21
6	限购	-1.58074E+20	-1.34925E+21	9.10833E-11	-2.17044E+20	-3.77137E+20	9.93519E-09	-1.15244E-08	-3.89269E+21	-8.998E+20	-6.894E+21
7	柯震东	-1.00776E+20	1.82261E-05	-1.83093E+21	-2.67982E-07	-2.40433E+20	-3.76269E+20	-4.92521E+20	-2.48167E+21	-5.73642E+20	-6.09624E+21
8	亿元	-1.23668E-06	-3.86616E-07	4.7475E-11	-1.27508E-06	-1.32635E-07	1.25206E-10	7.9935E-10	-5.86428E+21	3.12153E-10	-5.86428E+21
9	沪	-1.34516E+20	-1.57616E-06	-2.62367E-11	-1.84698E+20	-3.20932E+20	-7.15897E-09	2.70432E-08	-3.31256E+21	-7.65703E+20	-4.71841E+21
10	同比	-1.09433E+20	-9.34079E+20	4.86705E-11	-1.50258E+20	-2.61089E+20	1.60894E-09	4.133E-10	-2.69488E+21	1.22142E-09	-4.14974E+21
11	A股	-9.24744E+19	-7.89324E+20	-1.32797E-10	-1.26972E+20	-2.20628E+20	-2.51107E-08	5.30757E-08	-2.27725E+21	-3.44578E-07	-3.50665E+21
12	球员	-1.07404E+20	-1.56881E-06	-1.95135E+21	-1.47471E+20	-2.56247E+20	-1.38513E-06	-5.24915E+20	5.08949E-07	-4.65577E-08	-2.98739E+21
13	房价	-6.80935E+19	-5.81218E+20	4.15111E-10	-9.34961E+19	-1.62459E+20	1.72418E-08	-1.34326E-07	-1.67685E+21	-3.87607E+20	-2.96973E+21
14	住房	-6.59657E+19	-5.63056E+20	4.02642E-10	-9.05746E+19	-1.57383E+20	1.89198E-08	-3.26781E-07	-1.62446E+21	-3.75495E+20	-2.87693E+21
15	VS	-7.13485E+19	-6.09002E+20	-1.29629E+21	-9.79654E+19	-7.28674E-09	-2.66396E+20	-3.48702E+20	3.05313E-07	-1.45119E-06	-2.6897E+21
16	体育	-1.26863E-07	-2.95185E-08	-3.5658E-08	-8.7492E+20	-1.52026E+21	-1.44381E-06	-3.89187E-09	1.90484E-08	-3.71432E-07	-2.39518E+21
17	主场	-8.57291E+19	-1.56063E-06	-1.55756E+21	-1.17711E+20	-2.04534E+20	-1.3736E-06	-4.18984E+20	1.56543E-06	-3.42949E-07	-2.38451E+21
18	球	-5.87844E+19	-5.0176E+20	-1.06802E+21	-8.07142E+19	-1.40249E+20	-2.19485E+20	-2.87297E+20	2.51769E-05	-1.4412E-06	-2.35631E+21
19	中超	-5.48666E+19	-4.68319E+20	-9.96836E+20	-7.53348E+19	-1.30902E+20	-2.04857E+20	-1.40297E-06	2.51724E-05	-3.12316E+20	-2.24343E+21
20	国安	-5.43864E+19	-4.6422E+20	-9.88111E+20	-7.46755E+19	-1.29756E+20	-2.03064E+20	-1.30379E-07	2.78255E-06	-3.09582E+20	-2.2238E+21
21	业务	-8.10898E-08	-5.36354E+20	2.18358E-11	-8.62791E+19	-1.27313E-06	-4.60464E-11	3.11048E-09	-1.54742E+21	2.54682E-09	-2.17005E+21
22	投资者	-6.33535E+19	-5.40759E+20	2.65541E-11	-2.49392E-07	-1.27375E-06	-1.4508E-09	9.51679E-09	-1.56013E+21	2.36284E-10	-2.16424E+21
23	上市	-1.63013E+20	-1.39142E+21	-3.13312E-13	-1.13443E-07	-3.88921E+20	-5.44295E-10	9.62878E-10	-4.92593E-07	1.64772E-09	-1.94335E+21
24	反垄断	-5.01593E+19	-4.28139E+20	8.18406E-09	-6.88714E+19	-1.19671E+20	-3.01378E-07	-1.2886E-07	-1.23521E+21	1.64989E-09	-1.90205E+21
25	收益率	-4.92431E+19	-4.20319E+20	-3.90872E-09	-6.76135E+19	-1.17486E+20	-1.33942E-06	-3.27662E-09	-1.21265E+21	1.44979E-07	-1.86731E+21
26	读图	-2.83874E+19	-2.42303E+20	-5.15753E+20	-3.89775E+19	6.52135E-05	-2.79894E-07	-1.38738E+20	-6.99062E+20	-1.61589E+20	-1.82481E+21
27	员工	-6.49145E+19	-1.54721E-08	1.48056E-09	-1.18298E-06	-1.54875E+20	-1.24735E-07	-2.53269E-08	-1.59857E+21	3.45133E-09	-1.81836E+21
28	阿里巴巴	-1.36028E+20	-1.16108E+21	-5.03341E-10	-1.86773E+20	-3.24538E+20	-1.39596E-06	-1.49324E-08	-7.38115E-08	4.58771E-08	-1.80842E+21

图 3：训练集文本卡方值统计表格示意图

如上图所示, 第一个词” 体育” 除去体育类以外的其他类别的卡方值均为负值, 因此若由此值构建决策树的根节点的话, 则表明: 若某一文本包含” 体育” 一词的话, 则其属于体育类. 但这种” 一棒子打死” 的分类方式显然是不合理的, 因为可以看到” 体育” 在体育类里头的卡方值也不是很大, 且有几类的负卡方值的绝对值也比较小, 不能强烈地拒绝它的分类. 同样地, 图中标红的几个词条都是有多个类别的卡方值为正值的情况. 若根据这些词条构建决策树的话: 一方面会造成” 一词定江山” 的情况, 另一方面若某文本包含前几个节点的词的话, 则会立即进入投票系统, 而非继续向下搜索词条.

那有没有更好的方法呢? 我也想过能否用连乘的方式进行排序, 因为乘积绝对值较大的词条, 其卡方值必定是强烈地拒绝, 或是强烈地表明, 该文本属于某一些类别的可能性的. 但同样需要考虑符号的问题, 分析起来比较麻烦. 最终我还是放弃了这个方法, 而采用前文的第一种方法. 但我还是相信第二种方法会有更好的想法能够做出来的.

采用第一种方法, 取各类别的前 100 个词条作为 feature, 共 900 个 feature, 代码如下:

```
def make_sets(file_path):
    Feature = []
    file = open_excel(file_path)

    sheet = file.sheet_by_index(1)
    for rownum in range(0, 100):
        row = sheet.row_values(rownum)
        word = row[0]
        Feature.append(word)
       # 对其余8个种类相同处理
```

3.6. 训练集向量化

选定了 feature 之后, 接下来就需要将训练集中的文本转化为决策树可以处理的结构化的数据. 我们采用的方式为: 将一个文本转化为 list 类型, 将其对应的 feature 值 (是否包含该词) 不断地插入该向量, 最后一个元素插入其类别. 并将该文本向量 (一个 list) 插入 dataset 的 list, 即最后扩充成一个二维 list, 构成一个矩阵的形式. 代码如下:

```
def CreateDataSet(wordlist, corpus_path):
    MATRIX = [] # 11855个训练样本, 900个feature + 1个 label

    catelist = os.listdir(corpus_path) # 获取seg_path下的所有子目录, 也就是分类信息
    for category in catelist: # 遍历每一个种类文件夹
        cate_path = corpus_path + category + "/" # 拼出分类子目录的路径
        filelist = os.listdir(cate_path) # 获取class_path下的所有文件
        for file in filelist:
            FILE = open(cate_path + '/' + file, 'r').read()
            WORDS = set()
            word = ''
            for char in FILE:
                if char != '\t': # 工作原理: 一个字符一个字符地读, 将不是制表符的字符丢进 word 字符串当中
                    word += char # 连接字符
                else: # 读到了制表符, 就从 word 字符串当中读取一个单词, 进行词频统计!
                    WORDS.add(word)
                    word = '' # 清空字符串,等待下一个词的输入
            Feature_Vector = [] # 文本的向量
            for key in wordlist:
                if key in WORDS:
                    Feature_Vector.append('YES')
                else:
                    Feature_Vector.append('NO')
            Feature_Vector.append(category)
            MATRIX.append(Feature_Vector) # 用 List 嵌套 List 来实现矩阵

    return MATRIX
```

3.7. 决策树训练与检验:

构建好数据集 MATRIX 之后, 我们就可以将其运用到决策树算法上去了:

```
def calcShannonEnt(dataSet):
    numEntris = len(dataSet) # labelcounts字典键为类别, 值为该类别样本数量
    labelcounts = {}
    for featVec in dataSet: # 得到dataset中每个样本的最后一个元素, 即类别
        currentlabel = featVec[-1]
        if currentlabel not in labelcounts.keys(): # 当前样本类别labelcounts中没有, 添加
            labelcounts[currentlabel] = 0 # 有则当前样本所属类别数量加一
        labelcounts[currentlabel] += 1

    shannonEnt = 0.0 # 计算香农熵
    for key in labelcounts:
        prob = float(labelcounts[key]/numEntris)
        shannonEnt -= prob * math.log(prob, 2) # numpy 的 log 好像第二个参数是 out array 要用 math 的 log 才会
            是 base
    return shannonEnt


def spiltDataSet(dataSet, axis, value): # 划分数据集 (数据集, 划分特征索引, 特征值)
    # python中函数参数按引用传递, 因此函数中构建参数的复制
    # 防止传入的参数原值被修改
    retDataSet = []
    for featVec in dataSet:
```

```
        if featVec[axis] == value: # 去掉当前这个特征 (因为划分中已用过)
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet


def chooseBestFeatureToSplit(dataset): # 选择最好的划分特征 (数据集)
    numFeatures = len(dataset[0])-1 # 特征数量
    bestEntropy = calcShannonEnt(dataset) # 原始数据集信息熵
    bestInfoGain = 0.0 # 最优的信息增益
    bestFeature = -1 # 最优的特征索引

    for i in range(numFeatures):
        featList = [example[i] for example in dataset] # 取第i个特征
        uniqueVals = set(featList) # set构建集合, 将列表中重复元素合并
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = spiltDataSet(dataset, i, value) # 按照所取当前特征的不同值划分数据集
            prob = len(subDataSet)/float(len(dataset)) # 计算当前划分的累计香农熵
            newEntropy += prob*calcShannonEnt(subDataSet)
        infoGain = bestEntropy-newEntropy # 得到当前特征划分的信息增益

        if infoGain > bestInfoGain: # 选出最大的信息增益特征
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature


def dict2list(dic:dict): # 将字典转化为列表
    keys = dic.keys()
    vals = dic.values()
    lst = [(key, val) for key, val in zip(keys, vals)]
    return lst


# 若特征用完后仍存在同一分支下有不同类别的样本
# 则此时采用投票方式决定该分支隶属类别
# 即该分支下哪个类别最多, 该分支就属哪个类别
def majorityCnt(classList):
    classCount = {}
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote] = 0
        classCount[vote] += 1
    # 字典排序 (字典的迭代器, 按照第1个域排序也就是值而不是键, True是降序)
    sortedClassCount = sorted(dict2list(classCount), key = operator.itemgetter(1), reverse=True)
    # 返回类别
    return sortedClassCount[0][0]


# 递归构建决策树
def createtree(dataset, labels):
    classList = [example[-1] for example in dataset] # 取类别
    # 如果classList中索引为0的类别数量和classList元素数量相等
    # 即分支下都属同一类, 停止递归
    if classList.count(classList[0]) == len(classList):
```

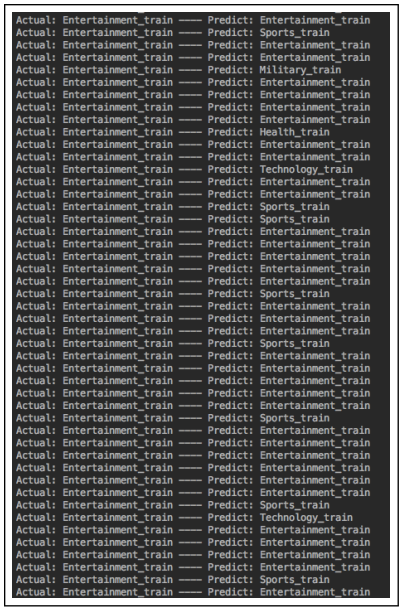
```
        return classList[0]
    # 划分类别的特征已用完，停止递归，返回投票结果
    if len(dataset[0]) == 1:
        return majorityCnt(classList)
    # 选择最具区分度特征
    bestFeat = chooseBestFeatureToSplit(dataset)
    bestFeatLabel = labels[bestFeat]
    # 树用嵌套的字典表示
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataset]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]
        # 递归构建决策树
        myTree[bestFeatLabel][value] = creatertree(spiltDataSet(dataset, bestFeat, value), subLabels)
    return myTree


# 分类函数（决策树，标签，待分类样本）
def classify(inputTree, featLabels, testVec):
    firstSides = list(inputTree.keys())
    # 找到输入的第一个元素
    firstStr = firstSides[0]
    # 这里表明了python3和python2版本的差别，上述两行代码在2.7中为：firstStr = inputTree.key()[0]
    secondDict = inputTree[firstStr]
    # 找到在label中firstStr的下标
    featIndex = featLabels.index(firstStr)
    # for i in secondDict.keys():
        # print(i)

    classLabel = 'Education'

    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]) == dict: # 判断一个变量是否为dict，直接type就好
                classLabel = classify(secondDict[key], featLabels, testVec)
            else:
                classLabel = secondDict[key]
    # 比较测试数据中的值和树上的值，最后得到节点
    return classLabel


def treetest(mytree, wordlist, test_path): # 测试集进行检验
    n = 0
    correct = 0
    catelist = os.listdir(test_path) # 获取seg_path下的所有子目录，也就是分类信息
    for category in catelist: # 遍历每一个种类文件夹
        cate_path = corpus_path + category + "/" # 拼出分类子目录的路径
        filelist = os.listdir(cate_path) # 获取class_path下的所有文件
        for file in filelist:
            FILE = open(cate_path + '/' + file, 'r').read()
            WORDS = set()
            word = ''
            for char in FILE:
                if char != '\t': # 工作原理：一个字符一个字符地读，将不是制表符的字符丢进 word 字符串当中
                    word += char # 连接字符
```

```
136             else: # 读到了制表符, 就从 word 字符串当中读取一个单词, 进行词频统计!
137                 WORDS.add(word)
138                 word = '' # 清空字符串,等待下一个词的输入
139         Feature_Vector = [] # 文本的向量
140         for key in wordlist:
141             if key in WORDS:
142                 Feature_Vector.append('YES')
143             else:
144                 Feature_Vector.append('NO')
145         feature_label = category
146 
147         predict = classify(mytree, feature_label, Feature_Vector)
148        print("Actual:", flabel, "---- Predict:", expct_cate)
149 
150         if predict == feature_label:
151             correct += 1
152         n += 1
153   print('Accuracy:', correct/n)
154 
155 
156 if __name__ == "__main__":
157     wordlist_path = '/Users/macbookair/iCloud/Desktop/Daily/Second Major/CS/Machine Learning/hw1/WORDS.xlsx
          '
158     corpus_path = '/Users/macbookair/iCloud/Desktop/Daily/Second Major/CS/Machine Learning/hw1/
          new_weibo_13638/Train_Set/'
159     test_path = '/Users/macbookair/iCloud/Desktop/Daily/Second Major/CS/Machine Learning/hw1/
          new_weibo_13638/Test_Set/'
160 
161     wordlist = set()
162     file = xlrd.open_workbook(wordlist_path)
163     sheet = file.sheet_by_index(0)
164     Features = []
165     for rownum in range(1, 927):
166         row = sheet.row_values(rownum)
167         word = row[0]
168         wordlist.add(word)
169         Features.append(word)
170 
171     DataSets = CreateDataSet(wordlist, corpus_path)
172     Decision_Tree = createrтree(DataSets, Features)
173     treetest(Decision_Tree, wordlist, test_path)
```

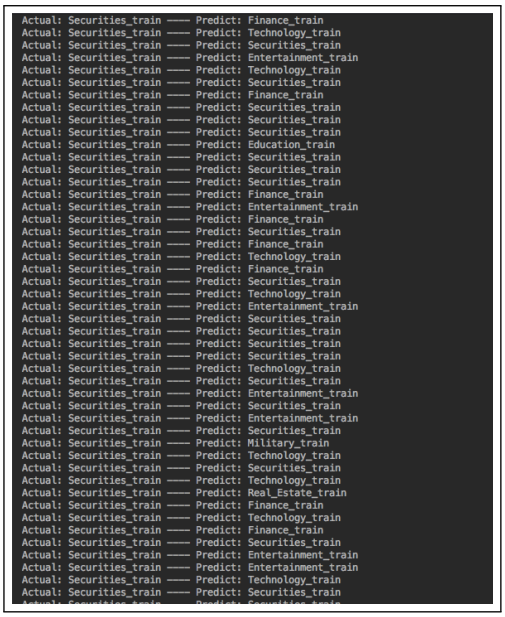
3.8. 结果显示与分析:

利用各类测试集来检验效果: 最终的准确度达到了 65% 左右.

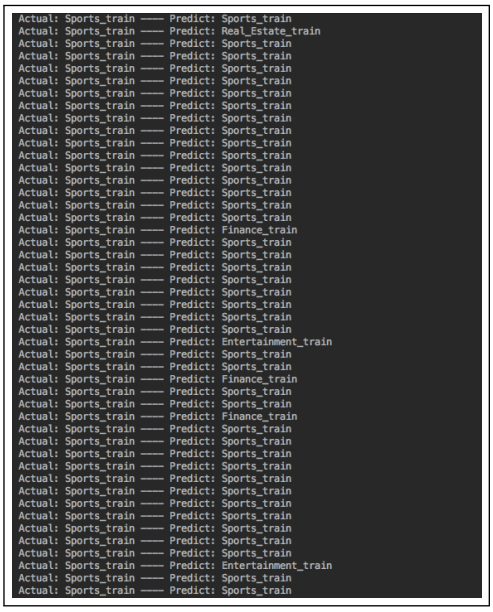
可以看到, 在一些类别上的预测是比较准确的, 而在另外一些类别上运行的效果并不是很好. 造成这种效果不均匀的原因可能是训练集与测试集划分的后果, 从这个意义上说, 做 cross validation 可能可以改善一下最终的结果.



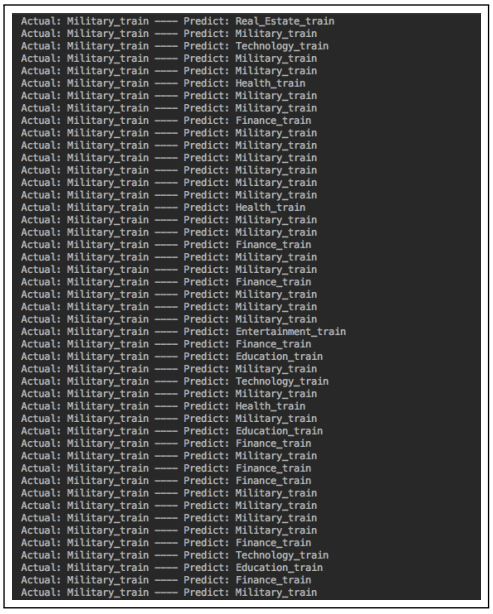
(j) 娱乐类



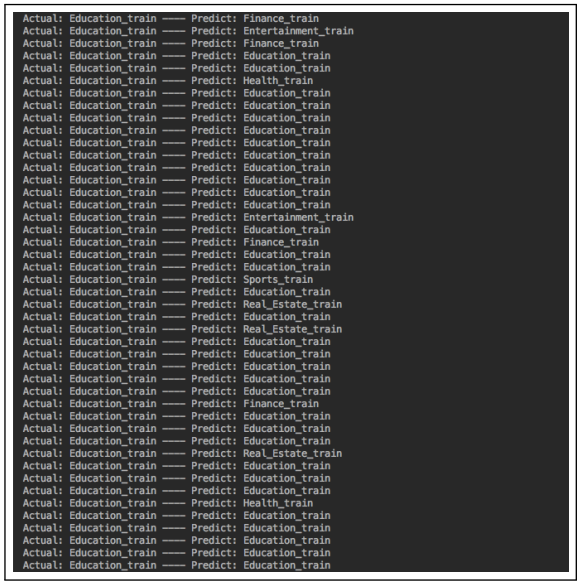
(k) 证券类



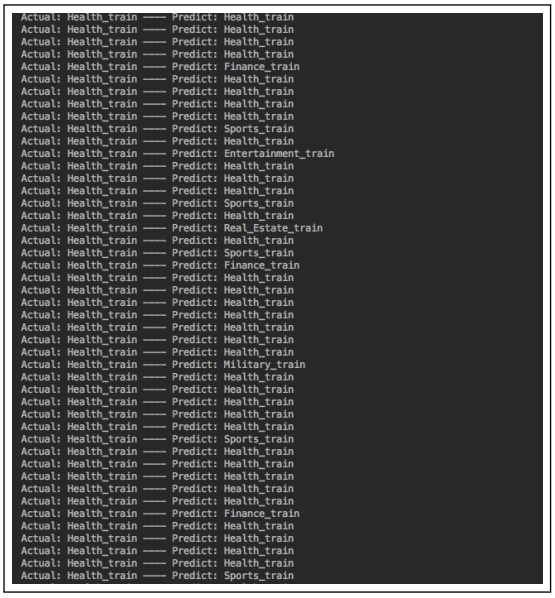
(l) 体育类



(m) 军事类

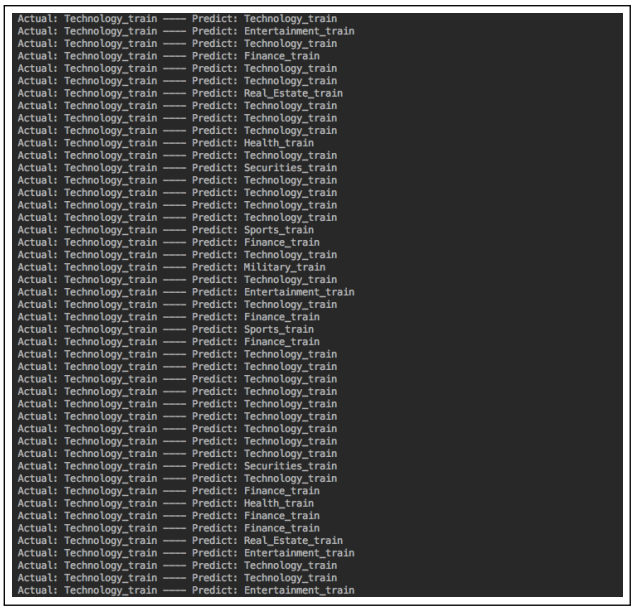


(n) 教育类

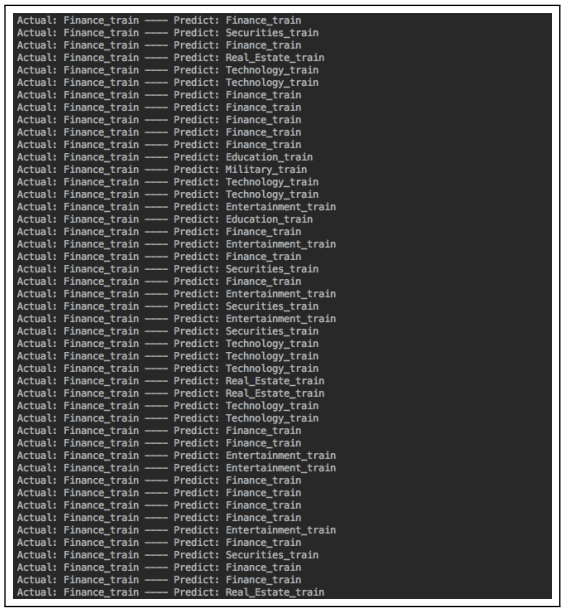


(o) 健康类

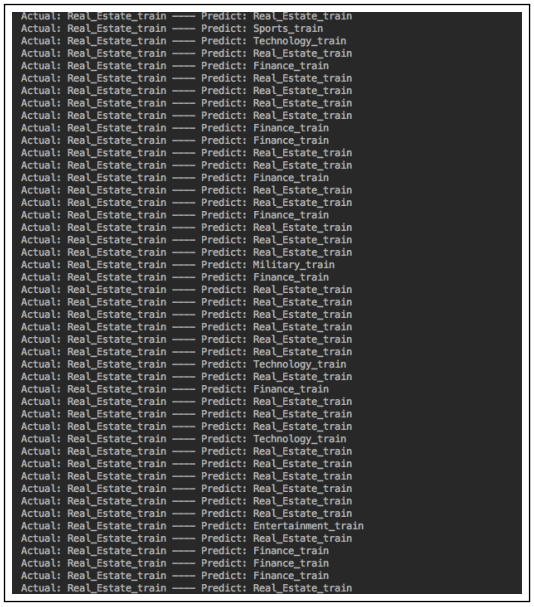
图 4.1：各类测试集检验效果图



(p) 科技类



(q) 财经类



(r) 房产类

图 4.2：各类测试集检验效果图

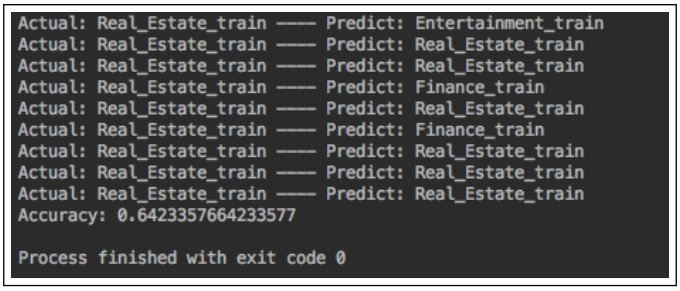


图 5：测试集准确率：Accuracy=0.64

References:

[1] 基于决策树中文文本分类技术的研究与实现 苑擎扬 东北大学信息科学与工程学院 刘辉林 2008.5

[2] 基于决策树的文本分类研究 田苗苗 吉林师范大学信息技术学院 吉林师范大学学报 (自然科学版) 2008.2

[3] 汉字字频统计程序(Python 版) rebillion51 http://blog.csdn.net/rebellion51/article/details/46682933 20015.6.29