**华东师范大学数据科学与工程学院实验报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：信息检索 | **年级**：2019 | **上机实践成绩**： |
| **指导教师**：张蓉 | **姓名**：张雯怡 |  |
| **上机实践名称**：NBClassifier | **学号**：10195501425 | **上机实践日期**： |
| **上机实践编号**：4 | **组号**： | **上机实践时间**： |

1. **实验目的**

使用朴素贝叶斯算法进行文本分类

1. **实验任务**

对数据进行分词和统计运算 （可选：词项优化方法）

将分词的结果输出到文件c\_学号.txt中，格式：文件id:类别

示例：49960:alt.atheism

1. **使用环境**

Python 3

1. **实验过程**

首先对文本进行预处理，分词并去除停用词，得到一个词汇表。

def getVocabulary(V):

# Python 代码首先尝试将数据集的词汇表构建为一个集合。

    vocabulary = set()

    cardinality\_of\_examples = 0

    for v in V:

        textfile\_names = os.listdir("./train" + "/" + v)

        cardinality\_of\_examples += len(textfile\_names)

        for textfile in textfile\_names:

            try:

                with open("./train" + "/" + v + "/" + textfile) as f:

                    for line in f:

                        # 使用正则表达式获取每个字母单词来实现的。

                        # 暴力只保留英文单词，除此之外全部去掉

                        raw\_word\_list=re.findall(r"[A-Za-z]+", line)

                        for word in raw\_word\_list:

                            if word not in sr:

                                vocabulary.add(word)

            except UnicodeDecodeError:

                with open("./train" + "/" + v + "/" + textfile, encoding="iso-8859-15") as f:# 区别两种encoding的方式

                    for line in f:

                        raw\_word\_list=re.findall(r"[A-Za-z]+", line)

                        for word in raw\_word\_list:# 使用正则表达式获取每个字母单词来实现的。

                            if word not in sr:

                                vocabulary.add(word)

    return [vocabulary, cardinality\_of\_examples]

然后通过找到 p (w | v)和 p (v)来学习数据集。

def initialise\_pw\_given\_v(vocabulary, V):

    pw\_given\_v = {}

    for v in V:

        pw\_given\_v[v] = {}

        for word in vocabulary:

            pw\_given\_v[v][word] = 0

    return pw\_given\_v

再对于每个 v，我们在字典中记录单词和它们的频率。

def initialise\_count(vocabulary, V): # count[v][wordj] = number of word in v

    count = {}

    for v in V:

        count[v] = {}

        for word in vocabulary:

            count[v][word] = 0

    return count

然后，使用相应单词的计数、词汇集的长度和计算每个文档中单词位置的数量来分配 p (w | v)的每个成员。

def learn\_naive\_bayes\_text(cardinality\_of\_examples, vocabulary, V):

    pw\_given\_v = initialise\_pw\_given\_v(vocabulary, V)

    Pv = {}

    count = initialise\_count(vocabulary, V)

    for v in V:

        textfile\_names = os.listdir("./train" + "/" + v)

        cardinality\_of\_docs = len(textfile\_names)

        Pv[v] = Decimal(cardinality\_of\_docs / cardinality\_of\_examples)

        number\_of\_distinct\_word\_positions = 0

        for textfile in textfile\_names:

            try:

                with open("./train" + "/" + v + "/" + textfile) as f:

                    for line in f:

                        regex = re.findall(r"[A-Za-z]+", line)

                        # regex\_=delete\_stopwords(regex)

                        number\_of\_distinct\_word\_positions += len(regex)

                        for word in regex:

                            if word not in vocabulary:

                                continue

                            else:

                                # print("Counting word " + word + " in 20news-bydate-train" + "/" + v + "/" + textfile + "...")

                                count[v][word] += 1

            except UnicodeDecodeError:

                with open("./train" + "/" + v + "/" + textfile, encoding="iso-8859-15") as f:

                    for line in f:

                        regex = re.findall(r"[A-Za-z]+", line)

                        # regex\_=delete\_stopwords(regex)

                        number\_of\_distinct\_word\_positions += len(regex)

                        for word in regex:

                            if word not in vocabulary:

                                continue

                            else:

                                # print("Counting word " + word + "in 20news-bydate-train" + "/" + v + "/" + textfile + "using encoding iso-8859-15...")

                                count[v][word] += 1

        for vocab\_word in vocabulary: #vocab\_word == wk

            pw\_given\_v[v][vocab\_word] = Decimal((count[v][vocab\_word] + 1) / (number\_of\_distinct\_word\_positions + len(vocabulary)))

            # print("Storing pw\_given\_v[{}][{}] as {}...".format(v, vocab\_word, pw\_given\_v[v][vocab\_word]))

    return Pv, pw\_given\_v, count

其中，单个文档的分类是通过查找当前词典中存在的文档中的所有单词来完成的。

然后我们得到每个 v 的所有 p (w | v)和 p (v)的乘积，并将它们存储在一个字典中，其中概率乘积作为键存储，而 v 作为值存储。

然后，我们对所述结果字典的键使用函数 max () ，并使用该函数值给出分类器对给定文档进行分类的内容。

最终进行分类测试：

def classify\_naives\_bayes\_text(path\_to\_document, pw\_given\_v, vocabulary, V, Pv):

    # print("Classifying textfile at " + path\_to\_document + "...")

    positions = []

    try:

        with open(path\_to\_document) as f:

            for line in f:

                for word in re.findall(r"[A-Za-z]+", line):

                    if word in vocabulary:

                        positions.append(word)

    except UnicodeDecodeError:

        with open(path\_to\_document, encoding="iso-8859-15") as f:

            for line in f:

                for word in re.findall(r"[A-Za-z]+", line):

                    if word in vocabulary:

                        positions.append(word)

    results = {}

    given\_p\_for\_v = 0

    for v in V:

        given\_p\_for\_v = Pv[v]

        for word in positions:

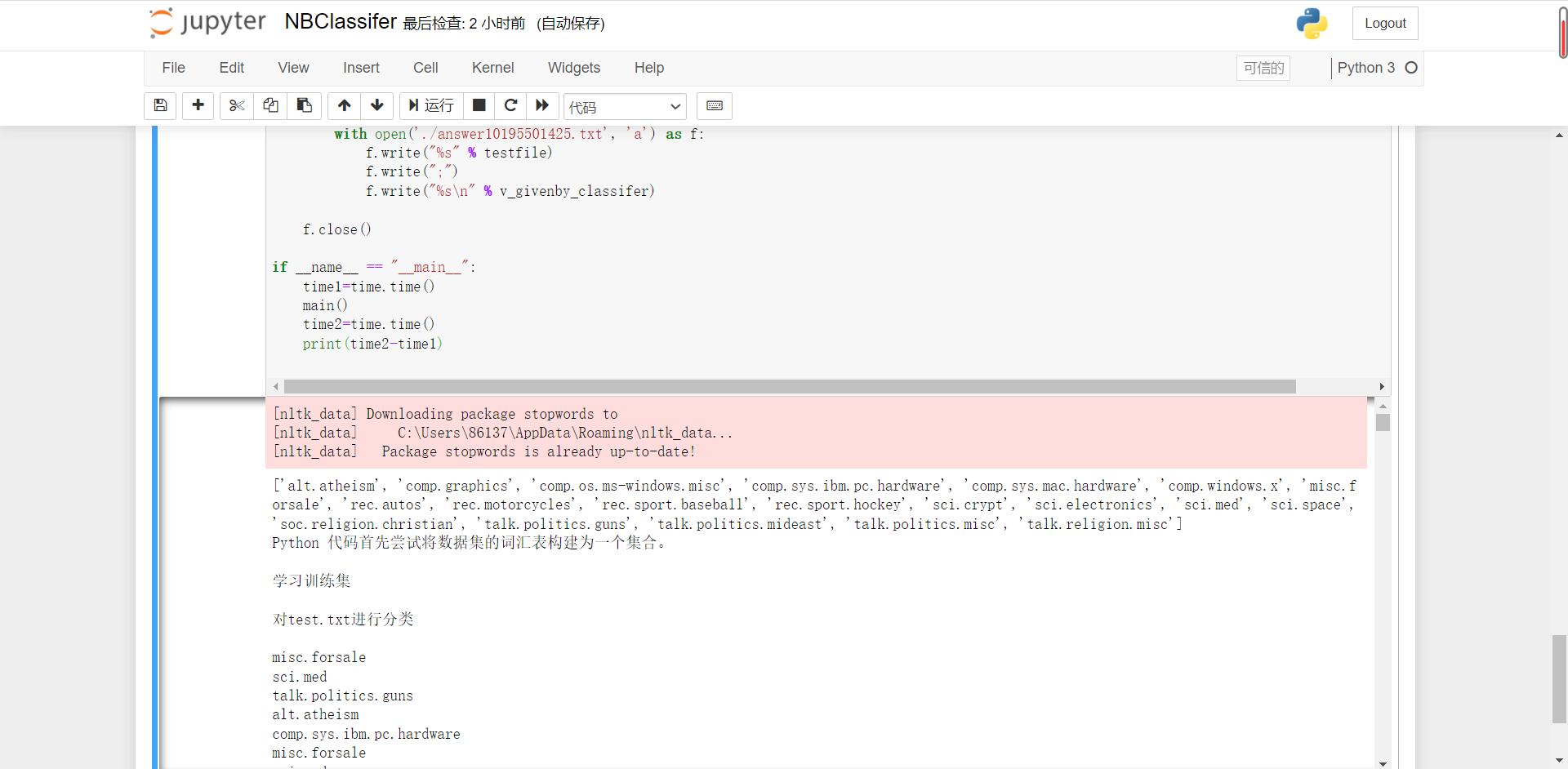
            given\_p\_for\_v \*= pw\_given\_v[v][word]

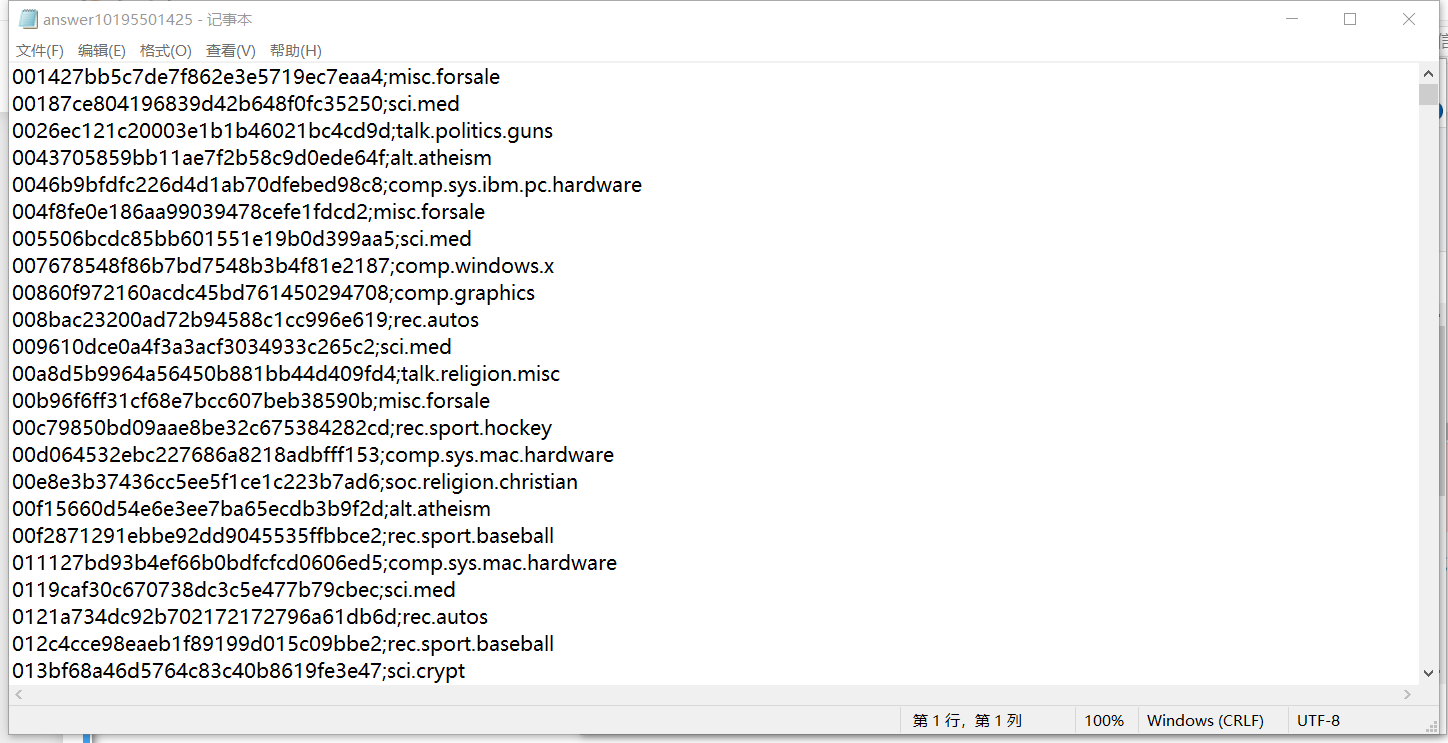
        results[given\_p\_for\_v] = v

    arg\_max = max(results.keys())

    return results[arg\_max]

运行结果：





1. **总结**

朴素贝叶斯分类器（Naïve Bayes）：

朴素贝叶斯分类器基于属性条件独立假设：对于已知类别，假设所有属性相互独立，也就是说，假设每个属性独立地对分类结果产生影响。

设待分类的样本属性集为TT，其中包含nn个属性，分别为t1,t2,t3...tnt1,t2,t3...tn，类别集合为γγ，当前待判断类别为cc，基于属性条件独立性假设，可得：

P(c|T)=P(c)P(T|c)P(T)=P(c)P(T)∏i=1nP(ti|c)

P(c|T)=P(c)P(T|c)P(T)=P(c)P(T)∏i=1nP(ti|c)

因为我们的目标是在当前的属性集TT下，找到一个尽可能正确的类别，也就是说，使得后验概率P(c|T)P(c|T)尽可能大。又因为对于所有类别来说P(T)P(T)相同，所以朴素贝叶斯分类器hnbhnb的计算公式如下：

hnb(T)=argmaxc∈γP(c)∏i=1nP(ti|c)

hnb(T)=argmaxc∈γP(c)∏i=1nP(ti|c)

可以看到上面的公式 ∏ni=1P(ti|c)∏i=1nP(ti|c)中 0≤P(ti|c)≤10≤P(ti|c)≤1，连乘运算很容易得到一个双精度都难以表示的极小的数值，不利于计算。因此一般情况下对上面的公式两边取对数，得到如下公式：

log(hnb(T))=argmaxc∈γ(log(P(c))+∑i=1nlog(P(ti|c)))