机器学习

编程作业：贝叶斯网络

姓名：陶威

班级：电信1604班

学号：U201613331

指导老师：王邦

# 任务一：使用朴素贝叶斯过滤垃圾邮件

## 实验原理：

基于朴素贝叶斯的思想来分类邮件，即对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。

## 编程思路：

1. 解析数据，将ham和spam中的文件挨个读入，每个文件存为一整个字符串，同时将ham中文件的类别设置为1，表示正常邮件，spam中的文件的类别设置为0，表示垃圾邮件。

*import* os  
*for* ham *in* os.listdir('data/ham'):  
 *with* open('data/ham/' + ham, 'r', encoding='iso-8859-1') *as* f:  
 words\_list.append(text\_parse(f.read()))  
 class\_list.append(1)  
  
*for* spam *in* os.listdir('data/spam'):  
 *with* open('data/spam/' + spam, 'r', encoding='iso-8859-1') *as* f:  
 words\_list.append(text\_parse(f.read()))  
 class\_list.append(0)

1. 创建词汇表，将所有输入的训练集中的单词解析出来，去重并创建一个词汇表。

*def* create\_vocab\_list(dataset):  
 *"""  
 创建一个包含在所有文档中出现的不重复的词的列表。  
 """* vocab\_set = set([])  
 *for* document *in* dataset:  
 vocab\_set = vocab\_set | set(document) *# union of the two sets  
 return* list(vocab\_set)

*# 创建词汇表*vocab\_list = create\_vocab\_list(words\_list)

1. 创建特征向量，将输入的训练集数据根据上一步的词汇表生成特征向量，即统计各个单词的数量并在相应位置上标识。

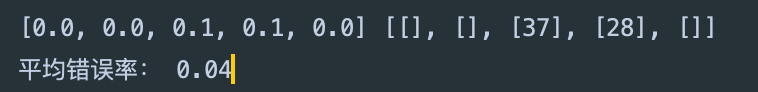
*def* words\_to\_vec(vocab\_list, input\_set):  
 *"""  
 获得文档向量，向量中的数值代表词汇表中的某个单词在一篇文档中的出现次数  
 """* return\_vec = [0] \* len(vocab\_list)  
 *for* word *in* input\_set:  
 *if* word *in* vocab\_list:  
 return\_vec[vocab\_list.index(word)] += 1  
 *return* return\_vec

*# 创建特征向量*vec\_list = [words\_to\_vec(vocab\_list, words) *for* words *in* words\_list]

1. 交叉验证，并计算其错误率。根据cv参数将训练集随机划分为相应的等分，将其中一等分当作测试集，另外部分当作训练集进行训练并计算错误率。

*def* cross\_validation(estimator, X, y, cv=3):  
 *"""  
 交叉验证  
 """  
 import* random  
 *# 生成随机下标* indexes = list(range(len(X)))  
 random.shuffle(indexes)  
 errors\_rate = []  
 errors\_index = []  
  
 *def* map\_data(source, data):  
 *return* [source[val] *for* val *in* data]  
  
 *for* i *in* range(cv):  
 train\_index = [index *for* index *in* indexes *if* index % cv != i]  
 test\_index = [index *for* index *in* indexes *if* index % cv == i]  
 estimator.fit(map\_data(X, train\_index), map\_data(y, train\_index))  
 error\_index = []  
 *for* j *in* test\_index:  
 res = estimator.predict([X[j]])[0] == y[j]  
 *if not* res:  
 error\_index.append(j)  
 errors\_rate.append(len(error\_index) / len(test\_index))  
 errors\_index.append(error\_index)  
 *return* errors\_rate, errors\_index

## 实验结果（五折交叉验证）：



# 任务二：使用朴素贝叶斯对搜狗新闻语料库进行分类

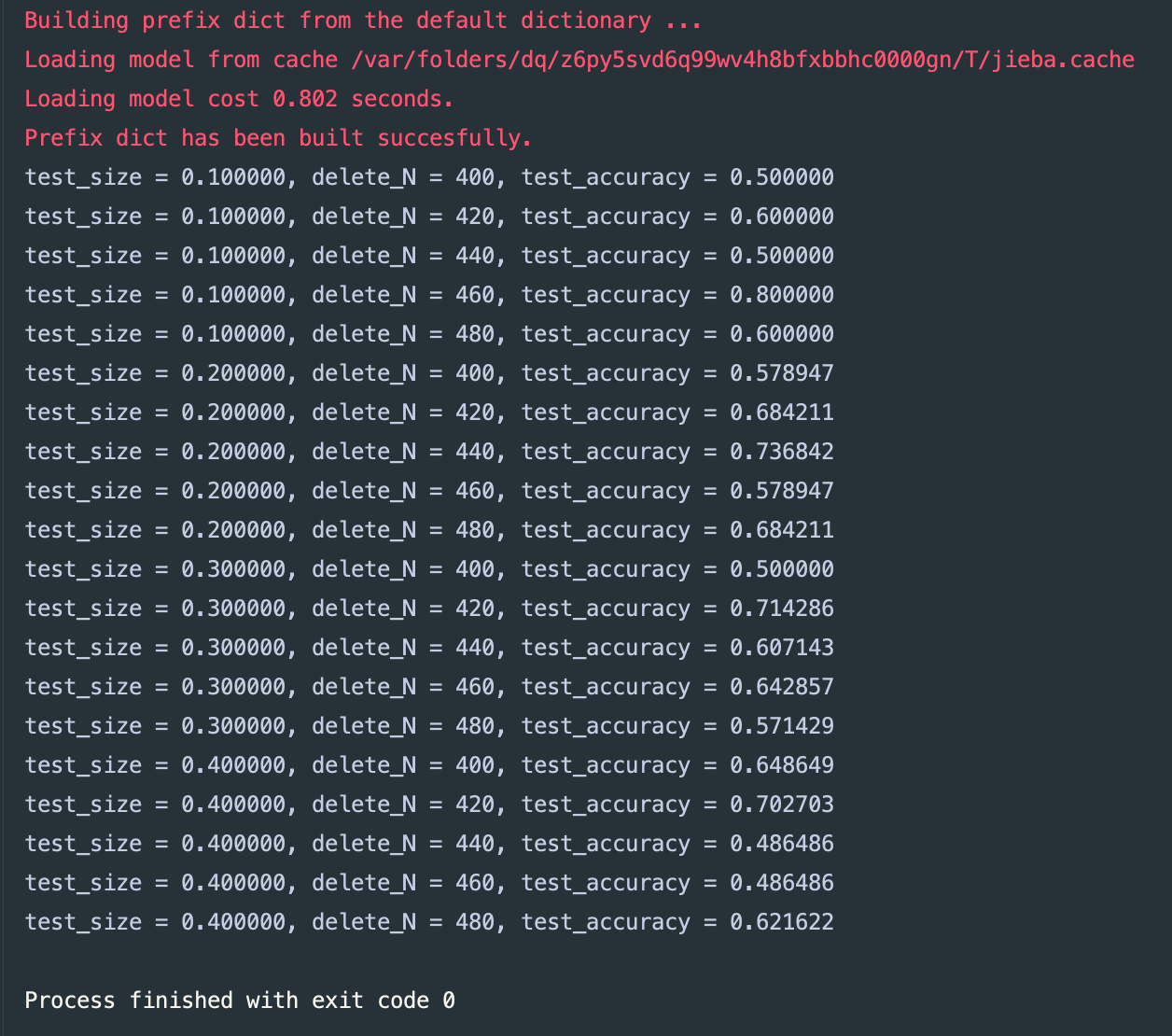
## 实验设置：

设置了不同的测试集比例和删除了不同数量的高频词，来验证分类的准确度

## 编程思路：

1. 载入数据，对中文文本使用jieba分词，获取词集，统计词频
2. 根据原始数据和测试集比例，随机划分训练集和测试集
3. 生成停用词集
4. 根据文本列表删除指定的前N个词频最高的词后生成文本特征集
5. 根据上述生成的特征集将训练数据和测试数据向量化
6. 使用贝叶斯分类器进行预测，计算准确度

## 实验结果：



由上可得，当测试集比例过高时，分类的准确度会下降；当测试集比例一定时，分类准确度随删除高频词的数量大致呈正态分布，在440-460左右分类准确度最高。

# 任务三：使用朴素贝叶斯对电影评论分类

## 编程思路：

1. 载入数据，解析为向量

*with* open('data/train/train\_data.txt', 'r') *as* f:  
 X\_train = [one\_hot\_encoder(10000, line.strip().split(' ')) *for* line *in* f.readlines()]  
  
*with* open('data/train/train\_labels.txt', 'r') *as* f:  
 y\_train = [int(line.strip()) *for* line *in* f.readlines()]

1. 对原始数据进行one-hot编码

*def* one\_hot\_encoder(feature\_num, data):  
 feature = np.zeros(feature\_num, dtype=np.int)  
 *for* num *in* data:  
 feature[int(num)] = 1  
 *return* feature

1. 网格搜索，找到最优超参数

*from* sklearn.model\_selection *import* GridSearchCV

param\_grid = [{  
 'fit\_prior': (*True*, *False*),  
 'alpha': [i \* 0.1 *for* i *in* range(9, 12)]  
}]  
clf = MultinomialNB()  
grid\_search = GridSearchCV(clf, param\_grid, cv=5, n\_jobs=-1)  
grid\_search.fit(X\_train, y\_train)  
print(grid\_search.best\_estimator\_)

1. 集成学习，不同朴素贝叶斯算法投票表决

*from* sklearn.ensemble *import* VotingClassifier

voting\_clf = VotingClassifier(estimators=[  
 ('MultinomialNB', MultinomialNB()),  
 ('GaussianNB', GaussianNB()),  
 ('BernoulliNB', BernoulliNB()),  
 ('ComplementNB', ComplementNB())],  
 voting='hard',  
 n\_jobs=-1  
)  
voting\_clf.fit(X\_train, y\_train)  
print(voting\_clf.score(X\_test, y\_test))

1. 最后载入测试集并保存预测结果

*with* open('data/test/test\_data.txt', 'r') *as* f:  
 X\_test = [one\_hot\_encoder(10000, line.strip().split(' ')) *for* line *in* f.readlines()]  
  
y\_predict = voting\_clf.predict(X\_test)  
*with* open('data/test/predict\_data.txt', 'w') *as* f:  
 f.write('\n'.join([str(num) *for* num *in* y\_predict]))