

机器学习概论实验一实验报告

一、实验目的

- 熟悉贝叶斯学习的基本原理，对贝叶斯学习的各种简化变形有所了解；
- 通过编写利用Naive Bayes进行垃圾邮件识别，对于Bayes学习在实际工程项目中的应用有所熟悉，同时能够体会到实现过程中的一些细节；
- 在实际项目中，感受Bayes学习的优劣势。

二、实验内容

对于64620个已经分词过的邮件文档(已打过标签)，采用Naive Bayes方法，进行模型的训练和测试，同时在其中针对以下几个话题进行讨论：

1. 训练数据集大小的影响。通过采用不同规模的训练集，观察最后评价指标的变化；
2. 零概率情境下的平滑处理。通过采用不同的平滑取值策略，体会它们的差异，感受平滑的效果；
3. 一些其他特征的作用。除了文档本身的信息，还可以提取出来其他特征，观察这些特征对最后模型结果的影响。
4. 其他有意义的话题，例如我还讨论了停用词表过滤的效果，选取的特征维数变化对模型效果的影响。

三、实验原理

本次实验所基于的理论基础为贝叶斯学习中的朴素贝叶斯方法。为了完成预期的任务，这里需要解决两个问题：贝叶斯决策以及贝叶斯模型的参数估计。其中，贝叶斯模型参数估计部分的原理与实践我在第四部分实验过程的第5小节模型的参数训练上会有详细的说明，于是这里我们仅针对贝叶斯决策这部分进行原理描述。

针对这个问题，我们定义以下术语便于后面表示：

样本点： $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$ 共N个样本点，每个样本点由 (x_k, y_k) 这样一组属性对构成。其中， x_k 表示第k个样本的特征构成的向量，当然，在这个问题中，特征可以是某个词项出现的频次，也可以是某个词项是否出现在文档中（布尔类型），还可以是发送邮箱类型、发送时间、发送方平台这些其他信息。而 y_k 则为样本k所属的类别，该问题中 $y_k \in \{spam, ham\}$

类别： $\omega_1, \omega_2, \dots, \omega_c$ ，即样本点所属于的类别信息，该问题中只有spam和ham两种。

特征向量： $x_k = (x_k^{(1)}, x_k^{(2)}, \dots, x_k^{(n)})$ ，其中，该样本的特征组成的向量的维数为n

据此，我们可以得到概率论中的Bayes公式可表示为

$$P(y_k = \omega_i | x_k) = \frac{P(x_k | y_k = \omega_i)P(\omega_i)}{P(x_k)} = \frac{P(x_k | y_k = \omega_i)P(\omega_i)}{\sum_j P(x_k | y_k = \omega_j)P(\omega_j)} \quad (1)$$

其中， $P(\omega_i)$ 表示该样本点属于第i类别的先验概率。

在该次实验中，由于我们使用的是Naive Bayes，即假设不同特征之间在已知类别时是相互独立的，即

$$P(x_k|y_k = \omega_i) = \prod_{j=1}^n P(x_k^{(j)}|y_k = \omega_i) \quad (2)$$

根据Naive Bayes的条件独立性假设，我们可以得到Naive Bayes的公式可表示为

$$P(y_k = \omega_i|x_k) = \frac{P(\omega_i) \prod_{l=1}^n P(x_k^{(l)}|y_k = \omega_i)}{\sum_j P(x_k|y_k = \omega_j) \prod_{l=1}^n P(x_k^{(l)}|y_k = \omega_j)} \quad (3)$$

有了上面的计算公式，我们就可以计算出在已知该样本点的特征向量的情况下，该样本点属于各种类别的后验概率。为了保证总体的错误率最小，我们选取后验概率最大的类别作为我们决策的该样本的分类，即

$$\hat{y}_k \leftarrow \arg \max_{\omega_j} P(x_k|\omega_j) \quad (4)$$

四、实验过程

1. 数据集认识与预处理

实验所提供的数据来自于中文邮件数据集：<https://plg.uwaterloo.ca/~gvcormac/treccorpus06/>，当然，助教也在网络学堂上提供了一个已经经过分词的版本。

我们可以发现，所提供的邮件文档共有64620个，文件所采用的编码格式为utf-8，打开数据文件，我们可以发现，它包含两个部分：前一部分是邮件头，其中包含了该邮件的相关信息，包括发送方、接收方、发送时间、邮件主题等等；而后一部分则是邮件具体内容。这两部分之间的特征就是中间有一个空行，因此我在数据使用过程中，也采用了这一方法进行提取。

同时我们发现，提供的数据中，还有一个label/index文件，我们发现其中包含了每个文档的相对路径和标签，标签格式为spam或ham，表示垃圾邮件或非垃圾邮件。通过简单的计数，我们发现，在所给的所有文档中，共有42854个垃圾邮件，21766个非垃圾邮件，两者之比约为2:1。同时，针对这个index文件，我进行了一下预处理，主要是为了在项目中将其引入正确的相对路径中，由于我采用的编程语言是java，IDE为Eclipse，其默认的根本目录为项目的根目录，因此，首先我需要将index中的../替换为input/，另外，由于我使用的是data_cut这个已经经过分词的数据集，因此我还需要将index中的data/替换为data_cut/（后面添加一个/的目的是防止连续替换）。并且，以示区分，我将新的文件命名为index_cut。

下面，我们针对具体的六万多个文档进行一个初步的认识。通过遍历一下所有的文档，我们发现，在邮件正文中，共出现了24962173个term，其中不相同的term共有213406个。所有这些文档所占的总的存储空间约为306MB。

2. 特征的提取与筛选

在这里，我们采用最为简单的one-hot方法进行特征提取。但是，从上一部分，我们也发现，一共存在着个213406不同的term（词项），如果我们将每个term都作为一维特征进行存储，那么为了将所有文档的特征记为一个特征矩阵，假定特征均采用一个整型变量(int)进行存储，最终所需消耗的内存大约为55G，这显然是不能接受的。因此，在这里，我们首先要进行特征的筛选操作。

我所筛选的具体流程分为以下三步：频次截断、去停用词、最大信息增益筛选。下面依次介绍这三个步骤：

(1) 频次截断

这里我做出一个基本假设：如果一个term出现的频次过低时，那么它所能提供的特征信息量是很少的，这主要是因为它出现的偶然性太大了，因此噪声会比较大。另外，它所出现的文档数量很少，因此对于大多数文档基本上不会有任何信息贡献。

为此，我选取了频次最高的前12000个term，并将结果写入topword.txt中，可以发现，出现频次最高的term为“，”，出现的频次为988865次，相对于平均每个文档出现了15次左右。而排名在第12000名的term为共模，出现频次为119次，可以看出，出现数量已经很少了。

(2) 去停用词

进行了频次截断之后，我们把出现频次很低的词汇给去掉了（低于119个），但是我们也会发现一个很严峻的问题：有很多term其实信息量是很低的，比如说频次排名前五的term分别为“，”、“-”、“的”、“-”、“、”，可以看出，基本上它们不会提供什么信息量。这种词我们一般称之为stop word（停用词）。为此，我的第二步筛选就是去停用词，我采用了网上所找的停用词表

（url:<https://blog.csdn.net/shijiebei2009/article/details/39696571>），并在其中补充了一些标点符号。然后采用逐一对比法去掉了停用词表中出现的term，同时，我还去除了为整数的term，这主要是因为整数主要是在分词过程中残留下来的电话号码的片段，因此，它们也基本没有提供信息量。最终，我将这部分筛选出的结果写入topfilterword.txt中。

去掉停用词和整数后，我们发现，存留下来的term的个数又从12000个降到了10435个，相当于去掉了1565个停用词！这个效果还是很可观的。下图展示的是去停用词前后出现频次最高的二十多个term，可以发现，前面的很多无效的标点符号和单字词都被去除了。

去停用词前	去停用词后
1 , 988865	1 公司 110427
2 - 931215	2 com 66506
3 的 875194	3 管理 64754
4 - 751342	4 企业 56685
5 、 453511	5 元 51825
6 。 427784	6 说 46255
7 273200	7 发票 44363
8 : 273110	8 月 36399
9 = 262727	9 有限公司 35453
10 . 250224	10 工作 34510
11 : 239078	11 http 34433
12 我 217578	12 服务 32093
13 / 207614	13 中 31867
14 了 173816	14 中国 31482
15 是 164205	15 年 30968
16 , 138430	16 做 28965
17 在 130011	17 www 28508
18 ! 123024	18 合作 28046
19 你 119539	19 请 27061
20 (119048	20 网站 26868
21) 114851	21 信息 24990
22) 113124	22 培训 24002
23 和 111395	23 优惠 23607
24 公司 110427	24 咨询 23480
25 有 107375	25 想 23047
26 (104537	26 日 22338
27 - 81389	27 贵 22255
	28 公 22105

(3) 最大信息增益筛选

经过前面的两步筛选后，我们仍然还有一万多个term，这个规模依然是很大的。为此，我们需要定义一个规则将这些term进行重排序(reorder)，按照我们的需求筛选出top k的结果。首先，我们发现，按照频次排序其实并不是一个很好的判断准则。例如，如果一个词它出现在所有的文档中，它的频次大约为六万多，但其实它的分类效果其实很差；而如果另一个词它只在垃圾邮件中出现，它的频次虽然只有四万多，但它的分类效果要比前者好太多。

根据上面那个例子，让我联想到了一个前面刚学的决策树学习中的一个概念：信息增益(information gain)。的确，在这个问题中，由于我们做了特征间独立性的假设(Naive Bayes Assumption)，因此，仅考虑每个term自己的信息增益，来评估其总体的分类效果，其实是合理的。关于信息增益，下面给出其计算公式：

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \quad (5)$$

在这里， S 表示所有的垃圾邮件组成的集合， A 表示邮件中是否含有该term，而 $Values(A)=\{\text{含有}, \text{不含有}\}$ 。 $Entropy(S)$ 表示 S 的信息熵，其计算公式如下：

$$Entropy(S) = - \sum_j P(\omega_j) \log_2 P(\omega_j) \quad (6)$$

在这里， ω_j 表示样本所有可能的取值，在这里它属于集合 $\{\text{spam}, \text{ham}\}$ 。

按照信息增益，我们可以根据前两步所筛选出来的一万多个term进行信息增益的降序排序，最终选取前top k的term作为特征进行提取。如下图是信息增益值排名前30位的term-增益值-频次对应值，我们可以发现，尽管它们并非一定是出现频次最多的一批词，但是它们所出现的频次并不少，最少的也有九千多。这主要是因为出现频次太少，未达到伤筋动骨的效果，对于信息的增益值不会太大。而其中，对于信息的增益值最大的是com,达到了0.32，我觉得主要是在很多非垃圾邮件中，为了方便对方联系自己，一般会设置“签名”，其中有一项则是自己的邮箱地址，里面就会含有com。

1	com	0.3204920785748213	66506
2	公司	0.22833738037581874	110427
3	说	0.21468231955058092	46255
4	有限公司	0.20663579831770407	35453
5	您好	0.18101436029699047	17194
6	发票	0.17159344700213142	44363
7	优惠	0.1612044785342207	23607
8	服务	0.14422683791890722	32093
9	联系人	0.1389380945940618	15553
10	广告	0.13368920404723028	17998
11	http	0.12800844764967378	34433
12	深圳市	0.12786624563143967	16825
13	代开	0.12575569150532606	15655
14	运输	0.12194381826616851	12142
15	实业	0.11767416325678803	14960
16	增值税	0.1135952607189068	12572
17	贵	0.11289856698765999	22255
18	合作	0.11278236714593082	28046
19	经理	0.10749652460609649	14881
20	财务	0.10662403428462552	15210
21	负责人	0.10467325034859687	11849
22	商祺	0.09838010435150024	9739
23	进项	0.09691353142600445	10549
24	信息	0.09561976744543832	24990
25	点数	0.09541573232513101	13349
26	顺祝	0.09315827105132879	9273
27	电	0.09161471147148703	11694
28	www	0.09129273974571694	28508
29	承诺	0.09023627097992781	11201
30	我司	0.09002890649206241	16332

3. 特征矩阵的建立

选取了我们需要使用的词项特征之后，下一步就要建立特征矩阵了。特征矩阵相当于一个索引表(index)，通过它，我们就可以将具体数据与最后的学习过程之间实现隔绝，从而达到比较好的通用性效果。

在这里，矩阵中每一行，即每一个文档中的特征（即每一列）对应于上一步选取出来的top k的term在文档中出现的频次，因此可以用一个整型数据进行存储(int)，而文档的列表(class)则为垃圾邮件或非垃圾邮件两类，也可以用一个整型存储（当然更经济地也可以使用Boolean类型）。这样，我们就把矩阵抽象出来了：其行数和列数分别为文档个数(64620个)与term个数加一(通常我选取term个数为1000，因此列数为1001)。编程实现也比较简单，基本上就是遍历一下每个文档，给出其特征的向量，然后写入磁盘中即可。

4. 训练集与测试集的划分

划分训练集与测试集的一个要求就是两者之间不能存在交叉，因为那样会使得测试准确率“虚高”，降低了模型的未知样本预测能力。另一个不成文的规定是要使得训练样本尽可能多些，训练样本的数量是模型好坏的一个重要保证，一种极端的处理方法被称之为“留一法”，但是在这里我并不适用。因为留一法的情况下，你需要抽样数据集规模量级次，然后得到平均的测试准确率，但在我这个问题中，那样做成本太高，反而得不偿失。为此，我最终选取的训练样本的全规模为45000个，其他样本作为测试集。

当然，在训练集与测试集的划分中，我采用了随机划分的策略，具体为随机洗牌策略(shuffle)，洗牌后抽取前45000个样本作为训练集。这样一方面保证消除了样本顺序之间的bias，同时实现起来也比较简单(Java的Collections类中有shuffle方法可以直接使用)。

5. 模型的参数训练

参数训练步骤，就是我们根据训练数据集，去训练得到一些参数的值。在这里，我需要训练得到的参数包括以下：

$$P(\omega_j), P(x_k^{(i)} | y_k = \omega_j) \quad (7)$$

其中， $P(\omega_j)$ 为第 j 类别的先验概率，我采用频率估计概率的方式进行训练，得到参数的估计，具体公式如下

$$P(\omega_j) = \frac{\sum_{k=1}^M I(y_k = \omega_j)}{M} \quad (8)$$

其中， M 为训练集的大小， $I(\cdot)$ 表示示性函数，当函数中的式子成立时结果为1，不成立时则为0。

而另一部分， $P(x_k^{(i)} | y_k = \omega_j)$ 可以直接运用条件概率公式，并根据频率估计概率来进行计算即

$$P(x_k^{(i)} | y_k = \omega_j) = \frac{\sum_{l=1}^M I(x_l^{(i)} = x_k^{(i)}, y_l = \omega_j)}{\sum_{l=1}^M I(y_l = \omega_j)} \quad (9)$$

运用上面这两个公式，我们就可以通过训练得到模型的参数。

6. 模型的预测

模型的预测我所依据的是最小错误率贝叶斯决策。即依据公式

$$\hat{y}_k \leftarrow \arg \max_{\omega_j} P(x_k | \omega_j) \quad (10)$$

当然，在这里，由于后验概率 $P(x_k | \omega_j)$ 是特别多个概率值相乘的结果，因此结果数值会非常小，为了防止概率的数值过小而跨越下界，因此在这里我采取了取对数后的等价的公式如下

$$\hat{y}_k \leftarrow \arg \max_{\omega_j} \log(P(x_k | \omega_j)) \quad (11)$$

当然在这个问题中，预测时，针对后验概率的计算，我提出了一下三种模型：

(1) 只考虑词项存在性的简单模型

在这个模型中，我只考虑了在文档中出现了的term，对于未出现的term的信息并未使用。同时对于term，只考虑了其出现与否这个信息，忽略了其出现的频次信息，其所对应的公式可表示为：

$$\log P(x_k | \omega_j) = \sum_{i=1}^n I(x_k^{(i)} > 0) \cdot \log P(x_k^{(i)} | \omega_j) \quad (12)$$

(2) 考虑了词项出现频次的模型

在这个模型中，我们只考虑在文档中出现了的term，不过在对这些term分析时，我们考虑到了该词项的频次信息，即我们认为每一次出现的词项都将作为一个需要利用的信息进行考虑，这样我们得到最终的公式如下：

$$\log P(x_k|\omega_j) = \sum_{i=1}^n I(x_k^{(i)} > 0) \cdot x_k^{(i)} \cdot \log P(x_k^{(i)}|\omega_j) \quad (13)$$

(3) 考虑了反向信息的存在性模型

在这个模型中，我们考虑了在文档中某个词项出现与未出现这两方面的信息，当然，因为在这里考虑了词项未出现的情况，因此对于出现了的词项，其出现频次的信息就显得不是那么的重要了。于是最终我们可以得到公式如下：

$$\log P(x_k|\omega_j) = \sum_{i=1}^n I(x_k^{(i)} > 0) \cdot \log P(x_k^{(i)}|\omega_j) + I(x_k^{(i)} = 0) \cdot \log(1 - P(x_k^{(i)}|\omega_j)) \quad (14)$$

7. 模型的评价

结束了预测过程，我们就要对学习的效果进行模型的评价了。在这里，我使用了Accuracy，Precision，Recall这些指标来评估各分类器。为此，我们需要首先定义True Positive, True Negative，False Positive，False Negative这几个概念，注意到在这里我们希望能够正确分类的类别为spam，即识别出来哪些邮件是垃圾邮件，因此我们将这一类别Positive。于是我们可以得到这几类的对应表如下：

	模型判为spam	模型判为ham
真实类别为spam	True Positive	False Negative
真实类别为ham	False Positive	True Negative

为此，相关的评价指标可以按照如下方式来定义

(1) Accuracy

准确率即对于所有样本中正确分类所占的比率，这里对于两类样本的分类不加以区分，即

$$Accuracy = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative}} \quad (15)$$

(2) Precision

精确率即对于被判为垃圾邮件的样本点其中的确为垃圾邮件的比率，公式表示如下

$$Precision = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (16)$$

(3) Recall

召回率即对于真实类别为垃圾邮件的样本点其中被正确找出的比率，公式表示如下

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{17}$$

(4) F1-Measure

由于精确率和召回率所考虑的信息都只是一方面，我们所需使用的好的学习机器应该是需要兼顾这两方面性能，因此就有了F1-Measure这个指标，其取值为Precision和Recall的调和平均，即

$$\text{F1-Measure} = \frac{2 \text{ Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{18}$$

五、实验结果与话题分析

具体程序的运行顺序可以参考第七部分的源码说明，这里我们给出实验结果以及有关于话题的分析，同时在其中也会嵌入代码中参数的配置方法。

1. 关于预测模型的选择问题

在这里，为了控制变量，我们不加入邮箱类别、发送时间、XMailer类型这些额外的信息，都进行滤词操作，特征维数大小我们都选择1000，训练集大小均选择100%，并且平滑系数 $\alpha = \frac{1}{\sqrt{\text{train size}}}$ ，（随后这些配置被称之为默认配置）仅仅是改变预测模型的类型（即上一部分提到的三种模型，在这里为了描述简单就称之为模型1,2,3）为了改变预测模型的类型，在代码中我们只需要修改NaiveBayes.java中的预测模型语句为如下三句之一：

```
1 int[][] table = bayes.testModel1(low, high); //采用模型1进行预测
2 int[][] table = bayes.testModel2(low, high); //采用模型2进行预测
3 int[][] table = bayes.testModel3(low, high); //采用模型3进行预测
```

我们分别对它们运行5组，求出各个性能的平均值，分别得到它们的测试集的结果如下：

【注：完整版的实验运行结果见提交的excel文件"实验运行结果.xlsx"中，这里只展示5组运行的平均值、最大值和最小值，之后的结果展示亦然】

(1) 考虑平均值

指标	模型1	模型2	模型3
Accuracy	0.96454	0.96338	0.92718
Precision	0.95609	0.95572	0.98017
Recall	0.99206	0.99078	0.90825
F1-Measure	0.97374	0.97293	0.94284

(2) 考虑最大值

指标	模型1	模型2	模型3
Accuracy	0.96498	0.96437	0.92813
Precision	0.95703	0.95698	0.98164
Recall	0.99257	0.99104	0.90952
F1-measure	0.97423	0.97372	0.94340

(3) 考虑最小值

指标	模型1	模型2	模型3
Accuracy	0.96407	0.96218	0.92614
Precision	0.95496	0.95424	0.97921
Recall	0.99148	0.99038	0.90672
F1-measure	0.97317	0.97197	0.94193

我们可以发现，三种模型从总体性能来看，模型1最佳，也就是我们当时认为的最basic的模型，而模型2与模型1之间的差别并不大，最终效果略差一些。而模型3结果差别则较大，从总体来看模型3比模型1、2要差，主要差距是在于召回率上，模型1、2的召回率可以达到99%以上，而模型3只有90%，也就是说10封垃圾邮件中就有大约1封邮件会找不出来。但模型3也是有它的优势的，主要则是在精确率上，它能够达到98%，也就是说它判别为垃圾邮件的邮件中，它被分错的概率是最低的。我觉得可能在大多数应用场景中，垃圾邮件分类的精确率的确应该也是最重要的一项指标，因此我们也不能一棍子打死说模型3比模型1、2要差。

同时，我们也可以发现，平均值、最大值与最小值这三个评价指标上差别不是很大，因此在接下来的部分中我仅展示平均值这一个指标，其他可以提交的excel文件。

2. 关于去停用词效果的分析

这里，我们控制其他参数为默认配置（模型默认使用模型1），仅改变是否去停用词。而改变是否采用停用词只需要调用WordIndexer中的方法如下：

```
1 index1.setFilter(true); //进行去停用词操作
2 index1.setFilter(false); //不进行去停用词操作
```

通过选择其中一行进行操作即可达到去或不去停用词的效果。

当然，为了后面使用的方便，针对是否使用去停用词，最终得到的top words和feature matrix我所存储的路径也有所区别，以示区分。而在NaiveBayes类中，则不加以区分，只需要给出相应的向量矩阵，就可以对其进行学习，因此在这里只需要相对应即可。

下面，我们来看一下两种情况下得到的测试集的测试结果（性能平均值）如下：

指标	不去除停用词	去除了停用词
Accuracy	0.95712	0.96454
Precision	0.96201	0.95609
Recall	0.97364	0.99206
F1-measure	0.96779	0.97374

可以看出，从总体性能角度看，去除了停用词与不去除相比，准确度提升了0.6%，这其实还算一个比较大的提升了。不过，在Precision这一项中，反而是不去除停用词这一项中占有优势。这也说明，在停用词所提供的信息中，有更强的倾向性让模型分类到ham类中。

3. Issue 1: 训练集规模对结果的影响

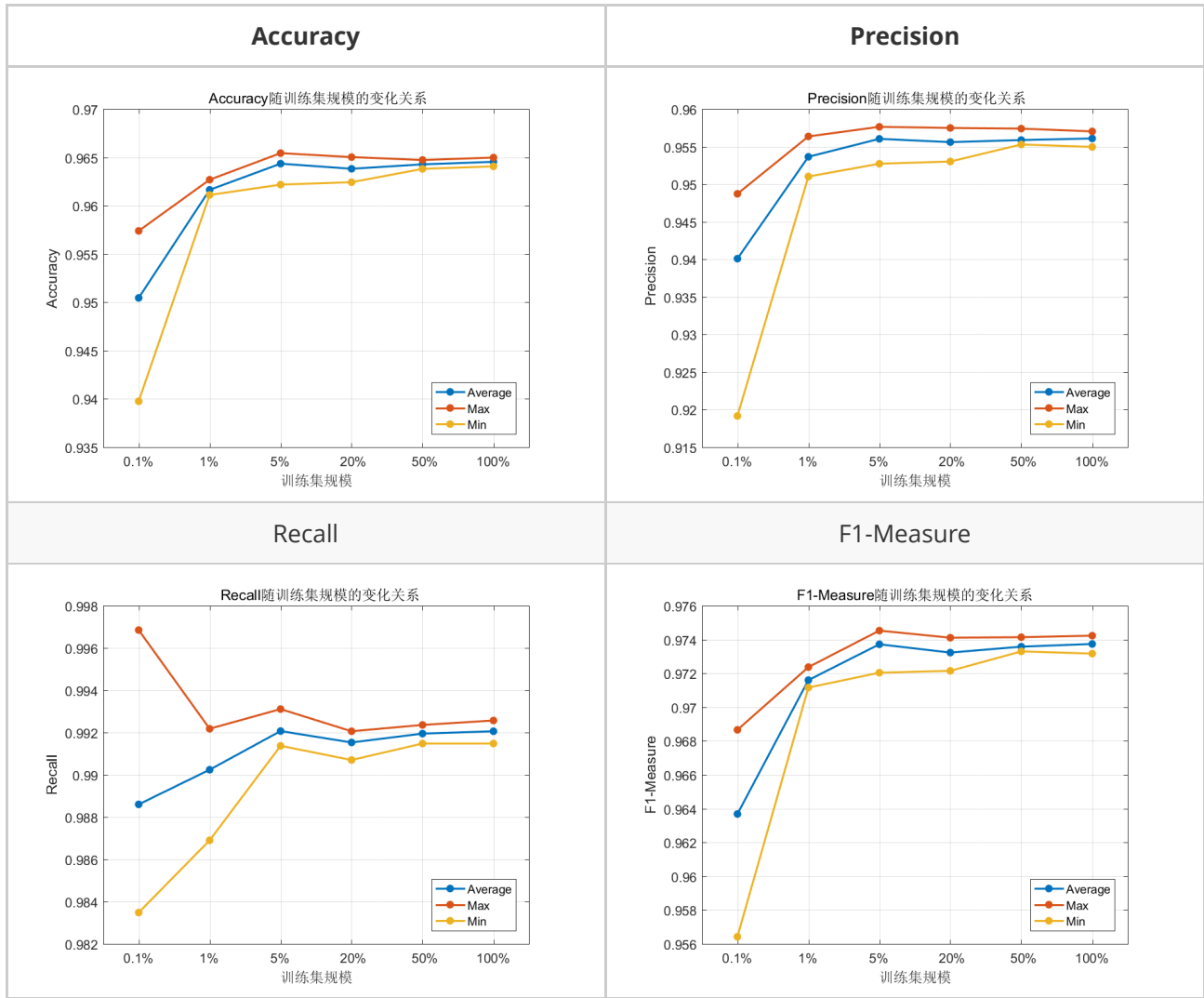
在我的模型中，所选取的100%的训练集规模为45000，而测试集都是从随机洗牌后的第45001个样本到最后这一万七千多个样本组成。因此，我们可以调整如下语句：

```
1 public static final int trainSize = 45000; //训练集数据的大小
```

改变初始给trainSize常量的赋值，就可以来控制训练集规模的大小。当然，45000个已经是最大规模了，因此规模取值只能比100%取值更小。为此，我们选取0.1%，1%，5%，20%，50%，100%取值下进行测试，得到测试集结果的相关指标（平均值），如下表所示：

训练集规模	Accuracy	Precision	Recall	F1-measure
0.1%	0.95045	0.94009	0.98860	0.96368
1%	0.96164	0.95365	0.99024	0.97160
5%	0.96435	0.95604	0.99207	0.97372
20%	0.96382	0.95560	0.99153	0.97323
50%	0.96428	0.95588	0.99195	0.97358
100%	0.96454	0.95609	0.99206	0.97374

据此，我们可以绘制出各指标随训练集规模变化的关系图像如下：



我们可以发现，随着训练集规模的增大，模型的稳定性也是在不断的提高，当训练集规模较小时，训出来模型的效果上下波动很大。但在50%规模下的训练集上，其性能与100%训练集数据规模上已几乎没有差异。另外，如果我们只考虑平均值大小，大约当训练集达到全规模的5%时，就基本上性能上与全集区别不大。综上可以看出，模型的鲁棒性上还是很不错的。

4. Issue 2: 零概率平滑问题

由于数据中可能存在着概率为零的特征，如果直接计算，这样可能会导致就是因为这一个term的特征，使得原本后验概率上占据绝对优势的类别直接因为一个零概率从而概率降为0。为了避免这种情况的诞生，我们拟采用平滑处理的方法，利用下式对于某维特征上的概率进行平滑处理：

$$P(x_k^{(i)} | y_k = \omega_j) = \frac{\alpha + \sum_{l=1}^M I(x_l^{(i)} = x_k^{(i)}, y_l = \omega_j)}{M\alpha + \sum_{l=1}^M I(y_l = \omega_j)} \quad (19)$$

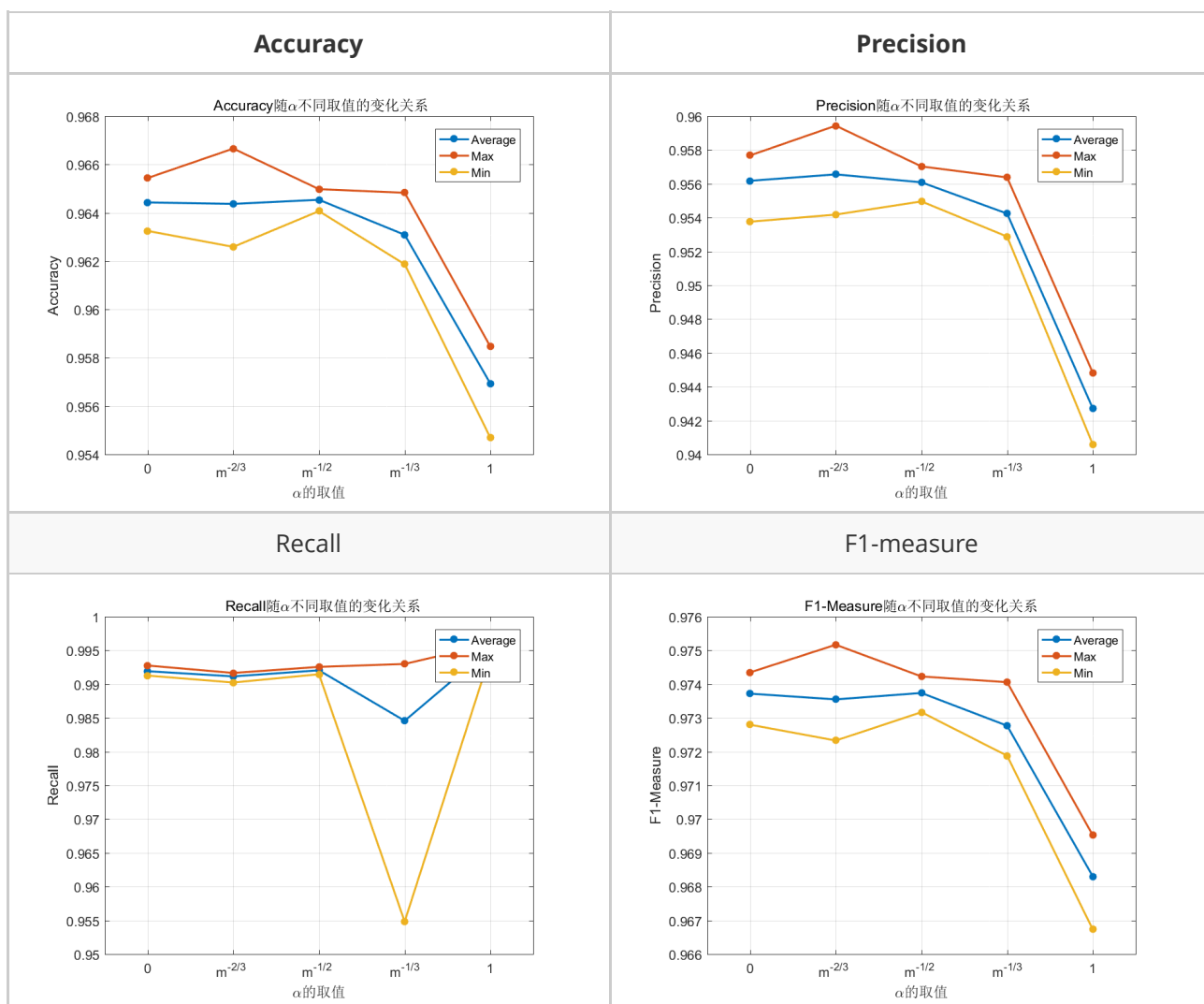
其中，在实验中，M我统一选取训练集的数据规模，例如在100%训练集情况下为45000，而 α 则也是一个关于 **train size**（令为 m ）的函数 $f(m)$ ，一般情况下我们认为 $0 \leq f(m) \leq 1$ ，否则它对数值的影响就将大于真实的样本点，弄成了“喧宾夺主”的副效果。在这里，我们选取 $f(m)$ 分别为 $0, m^{-2/3}, m^{-1/2}, m^{-1/3}, 1$ 进行实验，实验中，我们只需要修改NaiveBayes.java中如下行的代码：

```
1 public double alpha = 1.0 / Math.sqrt(trainSize); //设置平滑系数
```

将变量alpha的取值修改为所需的表达式即可。最终我们得到最后的结果（性能平均值）如下：

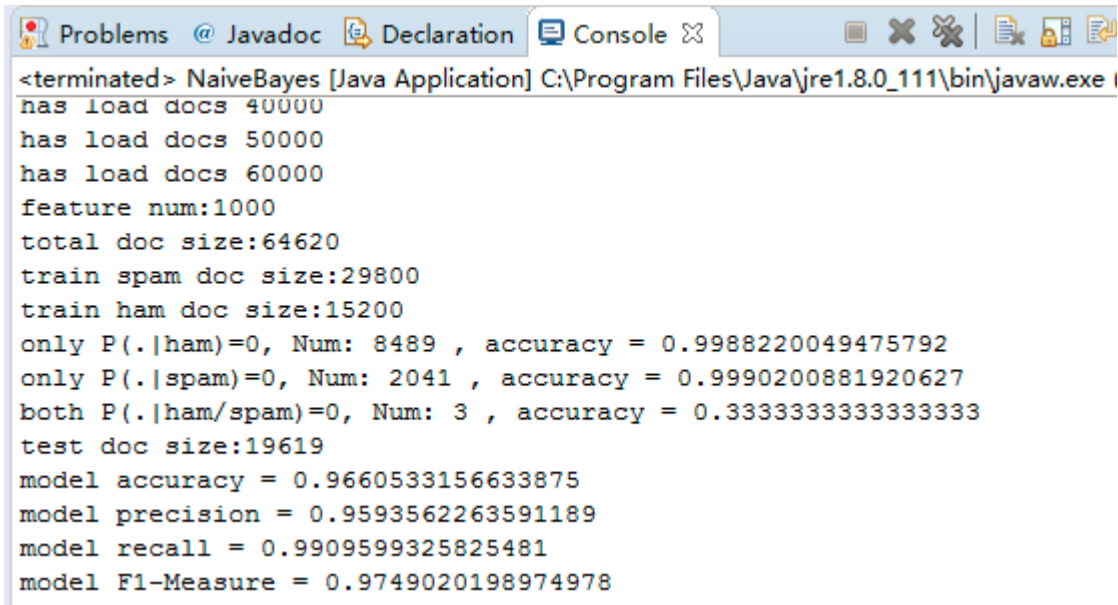
α	Accuracy	Precision	Recall	F1-Measure
0	0.96443	0.95618	0.99193	0.97372
$m^{-2/3}$	0.96437	0.95656	0.99116	0.97355
$m^{-1/2}$	0.96454	0.95609	0.99206	0.97374
$m^{-1/3}$	0.96309	0.95424	0.98458	0.97277
1	0.95693	0.94271	0.99533	0.96830

据此，我们可以绘制出各指标 α 取值不同的变化关系图像如下：



可以发现，和我们之前预期的结果不太一样！从图中我们看到，当 α 取值不大于 $m^{-1/2}$ 时，最后的结果没有太大的区别！而当 $\alpha = m^{-1/3}$ 时，性能开始出现小幅下降，当 $\alpha = 1$ 时，下降就相当明显了。

下面，我们针对 $\alpha = 0$ 时仍能保持较高性能这一点进行一定的分析。要说到 $\alpha = 0$ 对性能若要有极大的影响，主要就是概率为0造成的，尤其是当两类的概率均为0时，最后的选择其实是盲目的，下面，我们就对 $\alpha = 0$ 时对测试数据进行计算时出现概率为零的样本点的数量进行计数（在我的代码中就相当于 $\log P$ 取到负无穷）。下面记录的是一次运行后的结果，其中命令行截图如下：



```
<terminated> NaiveBayes [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe |
has load docs 40000
has load docs 50000
has load docs 60000
feature num:1000
total doc size:64620
train spam doc size:29800
train ham doc size:15200
only P(.|ham)=0, Num: 8489 , accuracy = 0.9988220049475792
only P(.|spam)=0, Num: 2041 , accuracy = 0.9990200881920627
both P(.|ham/spam)=0, Num: 3 , accuracy = 0.3333333333333333
test doc size:19619
model accuracy = 0.9660533156633875
model precision = 0.9593562263591189
model recall = 0.9909599325825481
model F1-Measure = 0.9749020198974978
```

整理并统计到表格中数据如下：

类别	样本点个数	Accuracy
全部测试样本点	19619	0.96605
ham为0，spam不为0	8489	0.99882
spam为0，ham不为0	2041	0.99902
spam、ham都为0	3	0.33333

我们发现，对于我们所忌惮的spam、ham两类全为0的情况，出现的其实是极少的，在这次运行过程中，只出现了3个这样的测试样本点。而对于ham或spam只有一个为0时，我们发现它的分类准确率竟然高达99.9%左右，这说明这种情况下，如果我们加大了为零特征项所能提供的信息，或许可以获得更高的性能提升。

于是，在下一部分中，我们就拟采用这种方法改进我们的模型，获得更高的Accuracy。

5. 利用零概率信息进行模型的改进

在上一部分中，我们发现，当spam和ham中在取 $\alpha = 0$ 时，有一个出现概率为0的情况，这个时候它的判别准确率是相当高的，基本上都在99.5%以上。因此，在我们采取一般的 α 值时，提前判断是不是有一边概率可能出现零的情况，从而利用到刚才得到的这块准确率相当高的信息，就可以提高模型整体的预测的精确度。

为此，我又在NaiveBayes.java中专门写了一个方法testModel1_1(int, int)，将这种方法实现了出来，下面我们来看一下其运行五次的性能评估（依然选取 α 为默认值），并与之前的模型结果进行比较（依然是只看平均值）：

指标	改进前	改进后
Accuracy	0.96454	0.96603
Precision	0.95609	0.95812
Recall	0.99206	0.99218
F1-measure	0.97374	0.97485

可以看出，这种改进后的方法在每项指标中性能都有所提升！总体来看提升大约为0.15%，可以看出这部分信息还是很有价值的。

6. Issue 3: 特殊特征的提取

在这里，我一共提取了三个特殊的特征，分别为发送邮箱类型、发送时间和发送平台，下面针对这三个特征的提取方法和结果进行说明和分析。

(1) 发送邮箱类型特征

我们发现，发送邮箱类型的信息一般都在第一行给出，例如下面这个就是一个典型的第一行内容

Received: from sohu.com ([219.134.0.1]) (20)

可以发现，从其中我们就能够得知该邮件的发送邮箱为sohu。根据题目中所给数据的一些常用邮箱信息，我将这一特征划分为一个9维信息，这九维特征具体如下：

mail type= [edu, 126.com, 163.com, qq, sina, sohu, yahoo, microsoft, other] (21)

对于前八个维度，如果该邮件的发送邮箱属于它，那么这一维度的值就为1，否则则为0。而如果该邮件都不属于前八种类型，则other维度置为1，否则置为0。

这里说明一下：其实这里的维度划分与Naive Bayes的特征条件独立性相矛盾，但由于最终我们实际只是用的到一个维度的信息，因此不会影响到实际使用的效果。

另外，解释一下为何对于126和163我在后面要加个.com。这主要是为了方便后期处理，否则这很容易与后面IP地址中可能出现的数字相冲突，这样追加之后可以直接使用126.com或163.com进行子串匹配。

最终，我们得到的ham和spam分别属于各个维度的数量向量如下：

ham = [13173, 1569, 740, 0, 79, 834, 313, 0, 5058] (22)
spam = [94, 4147, 6315, 249, 927, 514, 1275, 211, 29122]

我们可以发现，邮箱类型所能提供的分类信息还是很可观的，尤其是对于QQ邮箱和微软邮箱而言，在这个数据集中所有它们发送的邮件全是垃圾邮件。

针对邮箱类型特征，我们可以将其加入到数据中，并给这部分特征分配一个权重 w ，并取 $w = 0, 1, 10, 100$ ，来观察不同取值下测试数据中分类性能（比较平均值）的变化，具体控制操作在于以下两句赋值语句：

```

1 public boolean useMailInfo = true; //是否使用邮箱信息
2 public double alphaMail = 10; //邮箱提供信息的权重

```

以下是在不同权重之下的运行结果：

指标	w=0	w=1	w=10	w=100
Accuracy	0.96454	0.96839	0.97215	0.88818
Precision	0.95609	0.96205	0.96405	0.86371
Recall	0.99206	0.99140	0.99510	0.98748
F1-measure	0.97374	0.97650	0.97933	0.92145

可以发现，随着所选取的权重 w 的增大，系统的性能呈现出先提高后降低的趋势，在选取的四个权重数值中，我们可以发现， $w=10$ 的时候性能是最好的，可以达到0.97215左右，比baseline要出0.7%左右，这还是一个相当大的提升。而随后，再随着 w 的进一步增大，性能下降很厉害。最终，经过不断尝试，我发现当 w 选取在6附近时，性能是最好的。

(2) 发送时间特征

有了前面那个特殊特征提取的详细例子，在这里对于后两个新特征，我就做一下简要的介绍。对于发送时间这个特征，由于它基本上都出现在第三行，因此我直接在第三行，通过字符串拆解的方式一步步逼近我所需要的发送小时的信息，并最终将其提取出来。

与邮箱特征类似，这里可以根据小时数，将发送时间特征化为一个24维的向量，分别为该邮件是否在0时，1时，…，23时内发送，基于此，我们可以得到ham和spam分别属于各个维度的数量向量如下：

$$\begin{aligned}
 \text{ham} &= [881, 931, 854, 818, 793, 769, 801, 827, 777, 895, 879, 882, \\
 &\quad 882, 936, 959, 945, 978, 983, 966, 987, 1022, 957, 944, 952] \\
 \text{spam} &= [1698, 1777, 1733, 1704, 1780, 1809, 1714, 1771, 1712, 1893, 1827, 1776, \\
 &\quad 1754, 1792, 1719, 1778, 1857, 1792, 1746, 1784, 1727, 1732, 1725, 1768]
 \end{aligned} \tag{23}$$

可以看出，两种类别基本上都是均匀分摊到各个维度上的，也就是说发送时间这个特征其实提供的信息量并不大（甚至可以认为没有），于是在这里，我们选取 $w = 0, 1, 5, 10$ ，来观察不同取值下测试数据中分类性能（比较平均值）的变化，具体控制操作在于以下两句赋值语句：

```

1 public boolean useTimeInfo = true; //是否使用时间信息
2 public double alphaTime = 10; //时间提供信息的权重

```

以下是在不同权重之下的运行结果：

指标	w=0	w=1	w=5	w=10
Accuracy	0.96454	0.96475	0.96318	0.96016
Precision	0.95609	0.95622	0.95558	0.95570
Recall	0.99206	0.99229	0.99044	0.98571
F1-measure	0.97374	0.97392	0.97270	0.97047

可以发现，当 $w=1$ 时效果不是很明显，而当 $w=5,10$ 时，分类的性能开始出现下降。这与我们刚开始的设想区别不大，因为这部分的信息的确很少，给最终模型所带来的更多的是噪声。

(3) 发送平台特征

在邮件头的信息中，还有一个常用的信息叫XMailer，例如一个普通的邮件中的XMailer信息呈现如下：

X-Mailer: Microsoft Outlook Express 6.00.2800.1106 (24)

例如这个邮件就是从微软的Outlook程序中进行发送的。根据题中所给的数据，我提取出的发送平台的特征共分为4维，这四维特征如下：

XMailer = [Outlook, FoxMail, VolleyMail, None] (25)

与前面的特征取值方式类似，如果发送平台属于某一类，则其取值为1，否则为0.但对最后一维特征None，则是如果没有发送平台时，其取值为1，否则为0。当然，这个地方要注意，这里的None与发送邮箱类型中的Other不完全相同。这里的None含义是，如果该邮件头中没有XMailer信息，则定义为None，而并不是只要不属于前三位特征就归为这一类。

针对这四维特征，我们可以得到ham和spam分别属于各个维度的数量向量如下：

ham = [7761, 164, 0, 9760]
spam = [19857, 9385, 2130, 2444] (26)

可以看到，XMailer信息对于两类邮件的区分力度的还是很不错的，尤其是对于VolleyMail，在数据中它全属于spam类型。

下面，我们选取 $w = 0, 1, 5, 10$ ，来观察不同取值下测试数据中分类性能（比较平均值）的变化，具体控制操作在于以下两句赋值语句：

```
1 public boolean useXMailerInfo = true; //是否使用平台信息
2 public double alphaXMailer = 10; //平台提供信息的权重
```

以下是在不同权重之下的运行结果：

指标	w=0	w=1	w=5	w=10
Accuracy	0.96454	0.96556	0.96737	0.96847
Precision	0.95609	0.95892	0.96184	0.96373
Recall	0.99206	0.99056	0.99013	0.98968
F1-measure	0.97374	0.97448	0.97578	0.97525

可以发现，权重从0不断增加到10的过程中，模型的性能在逐步地提升，不过提升幅度没有前面的邮箱类型信息那么大，顶峰的时候提升幅度大致为0.4%。但我们还是可喜地看到这部分信息对于模型的性能还是有正向地影响的，并根据不断尝试，我发现，当权值w在8附近时，提升效果最为显著。

7. 选取特征维数对结果的影响

由于我这里所选取的特征是top k维的term，因此只需要改变k即可改变最终选取的特征的维数。要改变特征维数，首先需要改变WordIndexer.java主方法中的语句

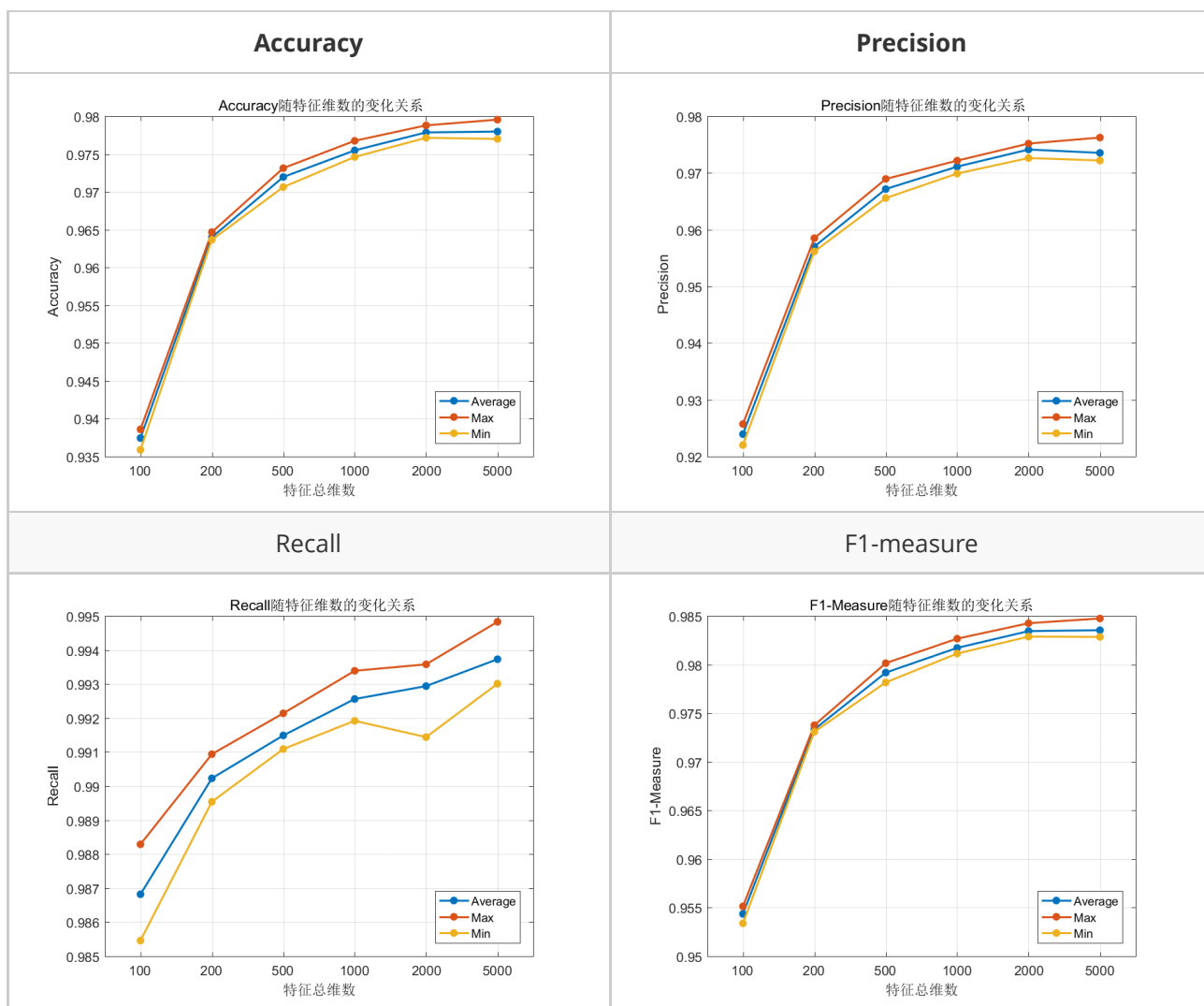
```
1 index1.reorderTopword(500);
```

其中，调用方法中的参数即为你所选取的特征的维数。随后，还需要在MatrixIndexer.java中生成该维数所对应的特征矩阵，之后就可以运行程序，进行模型的训练和测试了。

下面，我们依次选取100,200,500,1000,2000,5000维特征进行分析，观察模型性能随着特征维数的变化关系，得到的结果（平均值）如下所示。当然同时为了测试上一部分其他特征对于结果的实际效果，在这里，我设定使用邮箱类型和发件平台这两方面信息，并选取邮箱类型的权值为6，发件平台的权值为8，在这种情况下进行实验。

特征维数	Accuracy	Precision	Recall	F1-measure
100	0.93743	0.92394	0.98682	0.95435
200	0.96400	0.95702	0.99023	0.97334
500	0.97197	0.96717	0.99149	0.97918
1000	0.97549	0.97112	0.99256	0.98172
2000	0.97787	0.97414	0.99294	0.98345
5000	0.97798	0.97355	0.99373	0.98354

以下是程序运行所得到的变化关系图像：



从上面这些图像中我们可以看出，随着特征维度的增加，模型的预测准确度也随着逐步提升，这一点在最开始100维到200维之间的提升反映的最为明显。另外，我们也看到了邮箱类型和发件平台那总共加起来只有13维特征的危害之处，它能够使得1000维的结果提升大约1.1%！，同时，它加上200维原始特征的效果近乎等价于1000维原始特征的效果。

六、实验小结

在第四和第五部分，我们分别针对实验中具体的实现流程和实验最后结果进行了详细地描述和分析，在这里，针对我实验中的收获进行一个简单的小结：

1. 数据预处理和特征筛选的重要性，从最初一共20万多维特征到最后我只采用的1000维特征之间，我进行了三次筛选，这对于性能和效率的提升效果还是很明显的。特征筛选除了提高程序运行效率外，另一个重要作用就是降低噪声。而数据预处理则为后面的处理提供便利和基石。
2. 关键特征的挖掘和提取有时真的很重要，从最后实例我们也能看到，邮箱类型和发件平台总共才13维特征，就把从201维到1000维这800维特征所达到的效果近乎抗衡了，这足以说明这些隐式的特征可能是需要我们去挖掘、去筛选的。
3. 小规模数据调试，大规模数据测试。当我们最开始编写代码时，最好还是使用较小规模的数据进行调试，这样可以降低过程中训练模型，等待出错debug的时间消耗。而当模型在小规模数据上没问题后，再转而去大规模数据测试。

模数据中测试，验证程序是否handle这一规模的数据。当然，在这之前，针对模型的复杂度也需要有一定的估计，避免写出那种理论正确，但实际完成不了的“天文”代码。

七、实验源码说明

本次实验，我使用的编程语言为java，总代码长度1300多行，代码见我提交的codes/文件夹中，下面对于代码中各类的功能以及正确的执行顺序进行一定的说明：

1. 各类的功能描述

类名	功能描述
WordIndexer	实现词项筛选过程的第一步和第三步，筛选出有效词项的框架类
WordInfo	记录在词项筛选时各词项所需要存储的各种信息，并进行简单处理
Filter	实现去停用词和数字的滤词操作
MatrixIndexer	生成并导出相应维度的特征矩阵
FeatureExtract	提取额外的三类信息（邮箱类型、时间、平台），并导出它们相应的特征矩阵
NaiveBayes	实现模型训练和测试的核心类

2. 程序正确执行顺序

1. 运行WordIndexer.java，在其中你可以选择是否使用去停用词，选择输出的词项的数量以及导出的文件名；
2. 运行MatrixIndexer.java，在其中你需要指定上一步的输出文件和这一步你希望导出的特征矩阵的文件名；
3. 运行FeatureExtract.java，导出额外特征信息；
4. 运行NaiveBayes.java，进行模型训练和测试，导入的文件名需与第二步的导出特征矩阵文件名相对应，其中你也可以设置各种参数，具体参数设置见第五部分的实验结果与话题分析。

八、参考文献

李航《统计学习方法》

<https://blog.csdn.net/shijiebei2009/article/details/39696571>

http://blog.sina.com.cn/s/blog_90bb1f2001019cki.html

<https://www.cnblogs.com/sddai/p/5696870.html>

<https://blog.csdn.net/u011606714/article/details/79091281>