朴素贝叶斯分类器-实验报告

计 74 顾掀宇 2017011401

2020 年 3 月 29 日

目录

1 主要工作

1. 了解并深入贝叶斯学习的基本原理，解决贝叶斯学习的过程中会存在的各种细节问题，实现了基本的朴素贝叶斯分类器，对于垃圾邮件的分类效果好，可以达到 97%+ 的正确率
2. 在基于词袋模型的基础上，构思并添加了一些其它的 Feature，取得了一定的效果，例如：
   (a) 添加停用词
   (b) 对于词做分类，例如含有数字的词、大写词、长单词各自统一作为一类处理
   (c) 过滤某些超高频词汇
   (d) 减少假阳性
   (e) 尝试新的 smoothing 办法
3. 代码工程化，训练过程被完全封装，可以自行设定是否使用某一些 Feature，便于横向对比各种效果
4. 依据实验要求，训练集大小对于最终结果的影响做出了纵向对比

2 实验原理

2.1 朴素贝叶斯

基于贝叶斯学习中的朴素贝叶斯方法。基本思想是根据已有数据（训练集）对先验知识作出修正，以此对新数据进行判断。首先明确一些数学记号：

- $(x_1, y_1), (x_2, y_2), (x_3, y_3)...$ 表示我们的数据，x_1 表示我们的一篇文章的特征向量，y_1 表示它所属的类别，即垃圾/非垃圾
  - 特征向量可以包括词、文章长度、发送时间等等的信息
- $x_k = (x_k^{(1)}, x_k^{(2)}, x_k^{(3)}...x_k^{(n)})$，表示一共有 n 种特征
- ω_1,ω_2 表示样本所属类别，这里只有 spam(1), ham(2) 两种

我们得到贝叶斯公式，左式含义为根据某一篇文章的特征 x_k 得出我们属于某一篇文章 ω_i 的概率，即后验概率。$P(\omega_i)$ 即属于某一类别的先验概率。

$$P(y_k = \omega_i | x_k) = \frac{P(x_k | y_k = \omega_i) P(\omega_i)}{P(x_k)}$$

在我们的实验中，假定不同特征之间的相互独立的，即我们得到

$$P(x_k | y_k = \omega i) = \prod P(x_k^j | y_k = \omega_i)$$

事实上这里 $P(x_k)$ 是和具体类别无关的，判断垃圾邮件也无需得出具体的概率，我们只需要将分子进行比对即可，也就是最大似然估计：

$$\hat{y_k} \leftarrow \underset{\omega_j}{argmax} P(x_k | w_j)$$

2.2 模型评价

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

结束预测过程，我们需要一些统计学指标来评价分类器：Accuracy, Precision, Recall, F1-Measure, 首先需要明确 True Positive, False Positive, False Negative, True Negative 几个概念。我们将识别为垃圾邮件作为 Positive。

- Accuracy，准确率即对于所有样本中正确分类所占比例，对于两类不加区分

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- Precision，计算被判定为垃圾邮件中的确为垃圾邮件的概率

$$Precision = \frac{TP}{TP + FP}$$

- Recall, 召回率即计算所有垃圾邮件中被正确分类的比例

$$Recall = \frac{TP}{TP + FP}$$

- F1-Measure，Precision 和 Recall 各自负责一方面的评价，为了兼顾二者，给出了 F1-Measure 的计算方法

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

其中，我认为 Precision 我认为是一个非常重要的评价方法，所谓“宁缺毋滥”，非垃圾邮件被判断错误的代价远大于垃圾邮件漏过去的代价

3 分类器基础框架实现

```
resource/       # 包含了停用词文件，以及未来或许有用的文件
config.py       # 包含了训练所需Feature类的定义
function.py     # 包含了文本分词、计算概率、获取特征向量的方法
procedure.py    # 包含了贝叶斯分类器训练步骤的定义
main.py         # 运行代码
```

3.1 调用

经过工程化封装以后，在预读取文件路径和分类之后，我们只需要

```
processing(dirs, labels, training_feature, if_shuffle=True, iter=5)
```

即可完成一次实验，其中 dirs 是所有文件的路径，labels 是对应文件的标签。training_feature 是我定义的一些训练模式，feature 由 TrainingFeature 定义，在训练的一开始声明，只需要给每个过程函数传入该训练参数即可。

```
class TrainingFeature:
    def __init__(self, args...):
        self.FEATURE_ADD_STOP_WORDS = add_stop_words            # 是否添加停用词
        self.FEATURE_DROP_FREQUENT_WORDS = drop_frequent_words  # 是否抛弃一些常用词
        self.VOCAB_SIZE = vocab_size                            # 词典大小
        self.FEATURE_USE_EXISTENCE = use_existence              # 是否使用存在来代替词频统计
        self.CATEGORIZE_WORDS =  categorize_words               # 是否对词进行分类
        self.train_proportion = train_proportion                # 训练比例
        self.test_proportion = test_proportion                  # 测试比例
        self.INCREASE_PRECISION = increase_precision            # 是否增加Precision
        self.MODERATE_LAPLACE = moderate_laplace                # 更平滑的Laplace平滑
```

需要说明的是，有一部分文件的编码格式不是 ascii，这些我用 chardet 包判断并跳过了。

3.2 停用词处理

这里一方面我从网上找了一份英文停用词作为 txt 文件存储下来，这个需要预先读取，还有一个是出现在邮件头部的一些词例如 message、sender、HTML 等等没用的信息，这些应当也被列属于停用词里面，一并加入停用词的 set。

```
stop_words = procedure.getStopWords(training_feature)
```

3.3 文本分词

一个非常 naive 的分词办法是直接用 `re.split(r'\W*', text)`分词。最后返还一个词的 List，将所有词进行小写处理。根据是否开启添加停用词的 Feature 判断是否应该加入词向量以计算特征。这里我还对词进行了分类处理，下一节讨论这件事情。

```
function.textParse(text, training_feature))
```

3.4 划分数据集

具体划分数据集测试集的策略是每次随机打乱 dir 和 label，选取前 4/5 作为训练集，然后 1/5 作为测试集。再根据 Feature 中训练集和测试集的比例再进行一次切片，最后计算结果为迭代五轮取平均值计算结果。

```
procedure.getTrainTestSet(dirs, labels, training_feature)
```

3.5 获取词典

遍历训练集中每篇文档的每个单词，然后统计数目，这里处理的方法也有一些细节问题，一方面需要控制词典规模，选取 top-k 出现频次的词汇作为选词依据。另一方面选择的词大可不必是最热门的，这个问题下一节讨论。这里获取的词典是一个 $word \rightarrow index$ 的 dict，由指向的 index 确定在向量表里的位置。

```
vocab = procedure.getVocabularyDict(vocabulary_with_count, training_feature)
```

3.6 训练

训练过程是对于训练集中的每篇文章通过词典构建特征向量，并统计一篇文章的单词总数，这里需要对垃圾/非垃圾的特征分别累加最后取平均。详细而言，对于某一个词，即特征 x_i^j，我们得到了它在垃圾邮件中出现的总次数，以及垃圾邮件的总单词规模，这样就可以得到该特征在垃圾邮件中出现的概率。同理我们得到它在非垃圾邮件中出现的概率，即 $P(x_k^j|y_k = \omega_i)$。这里还有另外一种处理办法，就是统计该词出现/未出现，对于具体数值不做统计。

还有两件事情比较重要，一方面我们计算出的概率值通常较小，我们需要取 log 然后将概率值相加对比，避免数字太小而精度过低。另一方面，为了避免我们计算出的特征概率是 0 的情况（即某个词从未在某一类中出现），我们采用 Laplace 平滑处理，在后面思考题部分有所描述。

```
procedure.training(arg1, arg2...)
```

3.7 测试

对于在测试集的每篇文章，我们需要通过同样的方法进行分词处理。依据下式计算结果，当某个词出现时，即乘以出现概率（取 log 相加）。最后依据垃圾/非垃圾的相对大小作出判断。

$$\hat{y_k} \leftarrow \underset{\omega_j}{argmax} P(x_k|w_j) = \underset{\omega_j}{argmax} \prod P(x_k^j|y_k = \omega_i)$$

```
procedure.validating(arg1, arg2...)
```

3.8 模型评价

依据统计知识分别计算 TP、FN、FP、TN，最后将几次迭代得到的数据取平均即可

4 不同 Feature 实验与对比

未经说明，以下的测试方法都为：首先划分 4/5 作为训练集，根据训练集参与比例再次切片，1/5 作为测试集。每轮 shuffle 一次，测试 5 轮后取平均（并不是标准的 5-fold，但也不会因为一次随机而影响结论）。

4.1 停用词

首先我找到了一个网上的英文停用词表,另外,一篇邮件还有非常多的无效词汇例如 HTML、sender、mail 这样每篇邮件都会有的东西，自然它不能提供什么有效信息，可以加入停用词。下面对比什么 Feature 都没有的以及添加停用词之后的分类器效果，词典规模为 10000。

Feature\\指标	Accuracy	Precision	Recall	F1
不归类、不添加停用词	0.958159	0.989226	0.937554	0.962492
不归类、添加停用词	0.971338	0.989784	0.960122	0.974645

由此，添加停用词有了不小的提升，说明在词典规模相同的情况下，大量的没用的停用词挤占了重要分类词汇的位置。

4.2 词归类

将文章分词完的结果会有非常多的奇怪的词汇比如 AAADDFSFCCCCC，a12bc，FREE 这样的东西。这三者代表了一些重要的分类特征：

- AAADDFSFfadsfasdff，一个长度大于 20 的字符串可能会是非常没有意义的单词，所以给这样的词额外以“_____OVERSIZE_WORD”加入词典
- a12bc,一个含有数字的词汇很有可能是没有意义的,所以这样的词叫做“_____CONTAIN_NUMBER”
- FREE，一个广告垃圾邮件可能会出现很多纯大写单词，我们叫它“_____PURE_UPPER”
- 1999,一些纯数字字符串可能包含了时间等信息,感觉用处不是很大,所以归类为"_____PURE_NUMBER"

为了避免友军误伤的情况，纯大写字母还有含有数字的词汇通常可能是特有名词比如 3D 这样的，所以我们要同时把两个东西都算进某篇文章的词汇。假如真的是没用的词，那么通过次数排序的时候会把它们筛掉。

这些严格来说不是词汇，而是以词汇的方式参与计算，我们得到了某类特征。以下是测试结果（词典规模 10000，训练集 20%）。

Feature\\指标	Accuracy	Precision	Recall	F1
不归类、不添加停用词	0.958159	0.989226	0.937554	0.962492
归类、不添加停用词	0.970712	0.983396	0.965419	0.974177
不归类、添加停用词	0.971338	0.989784	0.960122	0.974645
归类、添加停用词	0.975865	0.988700	0.969198	0.978824

可以看到，再进行词归类之后，确实也起到了很好的效果。足以说明大写字母、含有数字、过长的单词成为进行判断的重要特征。但是同时归类、添加停用词之后效果并不是叠加的，可能是它们正确率已经很高的缘故，想要提升效果就有些困难。我们还可以检查一下两个特征对于小规模测试集（1000 词汇表）的帮助。由此可见这两个 Feature 给 1000 词汇表带来更显著的提升。

Feature\\指标	Accuracy	Precision	Recall	F1
不归类、不添加停用词	0.902444	0.944182	0.883567	0.911742
归类、添加停用词	0.941695	0.954645	0.944298	0.949145

4.3 丢掉热门词

进行这个测试的时候，本身的愿景是，假如某些词汇非常常用，那么它们可能会在垃圾/非垃圾同时出现，就和停用词一样没有什么帮助效果，所以我们可以把最常用的那些词丢掉。下面是测试结果（词典规模 10000，测试集 20%，归类、添加停用词）。

Feature\\指标	Accuracy	Precision	Recall	F1
热门词丢弃	0.82888	0.798315	0.941783	0.863705
热门词不丢弃	0.971566	0.986842	0.963316	0.974774

这里丢弃停用词成为了完完全全的负优化，从这个角度也可以说明，抛掉停用词之后，剩下的热门词汇包含了非常重要的判断信息。

4.4 减少 FP，增加 Precision

正如前面 2.2 节所说，Precision 是一个非常重要的指标，我们需要尽量减少非垃圾邮件判断为垃圾邮件的概率，因为这样会给用户带来更多的困扰。对于某个单词，假如在垃圾邮件中出现的概率为 P^i_{spam}，非垃圾出现的概率是 P^i_{ham}，我们可以把 $2 \cdot P^i_{ham}$ 带入计算，以增加判断为正常邮件的倾向。以下是测试结果（词典规模 10000，测试集 20%, 归类、添加停用词、不丢掉热门词汇）。

Feature\\指标	Accuracy	Precision	Recall	F1
带入 2 * P	0.869746	0.998601	0.774607	0.872035
不带入	0.975865	0.988700	0.969198	0.978824

这里效果还是很显然的，虽然整体其它指标有所下降，但确实减少了正确邮件被判断为垃圾邮件的概率，Precision 有所增加，这对于实际应用还是有价值的。

4.5 词典规模

词典规模影响了特征向量长度并直接影响了训练速度，在维持其它训练参数不变的情况下，我在训练集大小为 20% 的情况下测试了以下词典大小规模：5000，7500，10000，20000（添加停用词，词归类），词典的选取原则是 top-k 出现频次。

词典大小\\指标	Accuracy	Precision	Recall	F1
500	0.908191	0.899079	0.947743	0.922009
1000	0.939624	0.943826	0.953403	0.947935
2000	0.954590	0.969877	0.950635	0.959757
5000	0.971520	0.981576	0.968879	0.975072
7500	0.974700	0.983791	0.972302	0.977923
10000	0.975865	0.988700	0.969198	0.978824
15000	0.977466	0.987835	0.973035	0.980265
20000	0.978047	0.990728	0.970981	0.980736

可以看到的是，随着词典规模的提升，判断效果也越来越好，选取 10000 作为选取规模已经还不错，如果规模再大则会影响训练速度，也不必深究这里的细节问题。

5 思考题

5.1 测试集大小

这个问题我们跑一遍试试就知道了（词典大小 10000，归类、添加停用词，不加倍非垃圾概率，不丢掉常用词），以下为测试结果，可见随着训练集规模的提升，效果越来越好，训练成本也随之增加

训练集规模\\指标	Accuracy	Precision	Recall	F1
5%	0.954326	0.989015	0.931205	0.959091
20%	0.975865	0.988700	0.969198	0.978824
50%	0.978858	0.991764	0.971290	0.981413
100%	0.980728	0.992011	0.974283	0.983063

5.2 零概率问题

5.2.1 Laplace 平滑处理

由于数据中可能存在概率为 0 的特征，也就是说某个词在垃圾/非垃圾中没有出现。如果直接进行计算，那么该类概率则会直接变为 0（况且取 log 的话会造成 math domain error）。所造成的结果就是，原本该类可能在其它特征上表现出了绝对领先，却因为一个零概率而直接打死，为了解决这个问题，这里引入了简单的 Laplace 平滑处理。

Laplace 平滑思想非常简单，给每个类别下所有划分的计数 +1。这里描述了一个普适的公式，K 代表种类的个数，也就是这里词向量的长度。取 $\lambda = 1$ 即可，引入平滑之后，避免出现概率为 0 的情况，保证每个值在 0 ~ 1 范围内，而且保证最后求和为 1。

$$P_\lambda(Y = c_k) = \Sigma \frac{I(y_i = c_k) + \lambda}{N + K\lambda}$$

其实这里面 λ 也是一个参数，也可以进行调参看看究竟有什么效果，因为时间原因这里就不多做文章了。

5.2.2 更加 smoothing 的方法

除了 Laplace，我还想到了一个更加激进的 smoothing 办法，正如上面所说，平滑的目的是为了避免因为某类因为没有某个特征而直接否定的情况出现，同理就算直接 +1 之后的概率也是非常小的，因此我们不妨对于某一特征在垃圾/非垃圾的概率做一下对比，假如有一个小于另一个的 1/9，那么把它调整为其 1/4，也就是避免某类特征上差距过大。对比一下这个效果（词典规模 10000，添加停用词，词分类）。

Feature\\指标	Accuracy	Precision	Recall	F1
更加的平滑处理	0.967258	0.978078	0.965336	0.971455
简单的 Laplace 平滑	0.971566	0.986842	0.963316	0.974774

根据结果来看效果稍微有所下降。大概说明某一项特征如果一个显著、一个不显著就不必照顾弱势一方了，这算是一个失败的 Idea。

5.3 Feature

在这次实验过程中，我给训练过程添加了不同的 Feature 参数，如是否添加停用词、是否进行词归类、丢掉热门词、是否减少假阳性概率以及词典规模等等，总的说来，这里面真正有用的 Feature 是停用词以及对词做分类后提升的效果，以及减少假阳性对于实际应用可能更有价值一些，其它可能算是一些“有用的 Idea”，或许可以在其它数据集上应用。

其实还有很多其它的 Feature 有待挖掘，这里我对于每篇文章的分词方法是 META 信息和邮件内容一并进行分词。因为邮件格式个人感觉不是很整齐所以没有多做尝试。但是 META 信息确实被包含了在了词向量里面，比如有 edu、mit、microsoft、arizona 这些词汇就是来自于邮件 META 信息，我相信这些都是非常重要的信息。

6 小结

这次朴素贝叶斯实验感觉收获颇丰，在这最后就对实验中的收获做一个简单的小结：

1. 将贝叶斯分类器各个部分进行划分，封装，工程化后可以进行统一调用统一测试。而且非常非常感谢助教提供的服务器！我用了 PyCharm 的远程调试器功能，这样可以本地写代码，自动同步到服务器上跑数据，这样也不会占用我电脑太多计算资源，感觉非常方便。
2. 特征分析与实现，最基础的版本是只对文章进行了分词。在此之上我脑补了一篇文章可能会具有的各种特征、可能有帮助的信息，最后通过实验验证。我也只是做了一点微小的工作，作为最经典的朴素贝叶斯学习应用，垃圾邮件分类大有文章可做。
3. 感觉助教的阅读！

参考文献

[1] https://github.com/apachecn/AiLearning/blob/master/docs/ml/4.

[2] http://www.paulgraham.com/better.html

[3] https://zhuanlan.zhihu.com/p/26329951

[4] https://www.youtube.com/watch?v=HZGCoVF3YvMt=5s

[5] https://github.com/PKUanonym/REKCARC-TSC-UHT

[6] Notes from the lecture