Naïve Bayes Classifier

石大川 <sdc17@mails.tsinghua.edu.cn>

2020 年 3 月 30 日

目录

1 Introduction

模型的环境，正确的目录结构和运行方式，在 README.md 中有说明。以下所有涉及定量分析的实验结果都是按照 README.md 中的设置且将 random 的 seed 置为本人学号 2017013632 得到的，改变模型的参数，seed 或者物理实验环境可能会产生不同的实验结果。

2 Implement

2.1 K-Fold

首先在 $tools.py$ 中实现了对数据集做 K-Fold 划分。本次实验中 $K = 5$，因此每次在其中 4 个 fold 上训练，在另一个 flod 上测试，实验结果取 5 次训练和测试的平均值。

具体实现上，$tools.py$ 中的 $kflod.py$ 函数会逐行读取 $'./trec06p/label/index'$，对于每一项随机生成一个 $0 \sim 4$ 范围内的整数编号作为其在哪个 flod 中的标识符，编号的结果存储在 $'./DataSet/kflod.yaml'$。

2.2 Training

在 $NaiveBayes.py$ 中实现了训练的过程:根据上一步生成的 $'./trec06p/label/index'$，每次取其中 4 个 flod 训练，训练生成的结果位于 $'./Prob/prob\_fold\_\{0,1,2,3,4\}.yaml'$，文件名中的数字代表之后测试时使用的 fold, 例如 $'./Prob/prob_fold_1.yaml'$ 意味着在 fold 0, 2, 3, 4 上训练并在 flod 1 上测试。

具体实现上，对每轮训练 (共 5 轮，每轮包含 4 个 flods):

Basic Words 使用正则

$$r'[a-zA-Z]+'$$

匹配文本中的 words, 并根据相应的 label 写入到词频模型文件。

Select Top Rank Valid Features 根据统计的词频，挑选出前 $rank\%$ 的 words 作为有效的 features。这样做的意义有两点：首先是出现次数太少的 words 噪声比较强，将其剔除对提升模型的泛化性能是有帮助的。其次，词频信息是要序列化后记录下来的，词频模型过大会使得序列化文件的生成和后续预测阶段文件的读取花费很大的时间成本。

经过多次实验，确定了 $rank = 20$ 是一个较为合适的值。

Delete Fuzzy features 根据统计的词频，剔除掉满足

$$(1-fuzzy) \times \frac{\#\{y=spam\}}{\#\{y=ham\}} < \frac{\#\{y=spam, x_i=k\}}{\#\{y=ham, x_i=k\})} < (1+fuzzy) \times \frac{\#\{y=spam\}}{\#\{y=ham\}}$$

条件的 words，其中 $fuzzy$ 是 $0 \sim 1$ 之间的系数。举例来说，如果有 10000 封 spam 和 20000 封 ham, 且 big 在 spam 中出现 100 次, 而在 ham 中出现 205 次。如果取 $fuzzy = 0.1$，那么可以认为 big 出现在 spam 和 ham 中的概率是相近的。因此一方面 big 这个 word 并不会给模型对于 spam 和 ham 的区别带来较大的参考价值，另一方面同上一个小节的理由：这样做进一步较减少了特征维度，再次压缩了序列化后的词频文件大小，节省了时间和存储成本。

经过多次实验，确定了 $fuzzy = 0.25$ 是一个较为合适的值。

Ignore Stop Words 在网上找了一张英文停用词表放在了 $'./StopWords/stopwords.txt'$，去掉了里面一些实际有用的词例如 http, html 等，一共剩下 852 个停用词，这些 word 在词频文件中的存储的词频都被置为了 0。理由也是这些词区分 spam 和 ham 的能力不强。

Delete Long Nonsense Words 实际训练发现统计的词频中有一些长度惊人的 word，到原数据中找了一下发现是 Content-Type 为 image 或者 application 时，会在邮件末尾附上一长串无意义的字符串，例如Figure 1。从右下角的红框中可以看出这些字符串后续还有很长。

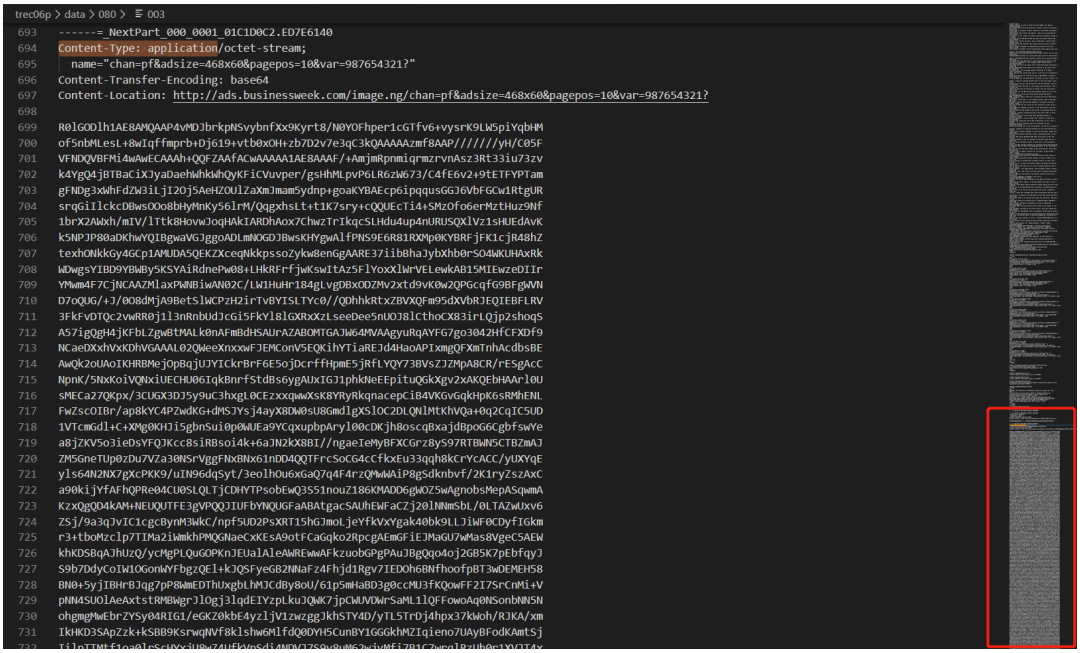

Figure 1: Example for nonsense words, from trec06p/data/080/003

这些字符串作为邮件的附件内容，对于 spam 和 ham 的区分影响不大，所以用正则：

$$r'\backslash S20,'$$

去掉了长度超过 20 的 words，去掉之后词频统计结果得到了进一步的压缩且看起来规整了很多，与此同时模型的准确率也有略微提升。

Extract Http Pattern 基于 spam 中往往带有较多 url 的假设，使用正则：

$$r'http[s]? : //(? : [a-zA-Z]|[0-9]|[\$-\_@.\&+]|[!*\backslash(\backslash),]|(? : \%[0-9a-fA-F][0-9a-fA-F]))+'$$

提取出邮件中的 url。

Extract Html Pattern 基于 spam 中往往带有较多 html 标签的假设，使用正则：

$$r' < /?\backslash w + [^{>}] * >'$$

提取出邮件中的 html 标签。

Extract Received From Pattern 基于大量 spam 可能来自同一个或少数几个发件人的假设，使用正则：

$$r'Received:(?:\s)+from\s[a-zA-Z0-9]*?\s'$$

以及：

$$email.split('from')[-1].strip('\quad').strip('\n').strip('\t').split('.')$$

提取出 meta data 中的发件人数据。

Extract Mailer Pattern 基于大量 spam 可能来自同一个或少数几个 Mailer 的假设，使用正则：

$$r'Mailer:\s(.*?)\s'$$

提取出 meta data 中的 Mailer 数据。

Decode 对于不能用 utf-8 解码的邮件抛出 UnicodeDecodeError 然后跳过。

Process Parallel 用 ProcessPoolExecutor 套了个并行，进程数默认为处理器最大核数，加速训练过程。

最后每轮 training 生成 spam 和 ham 的特征维度分别在 $10632 \sim 10715$ 和 $16970 \sim 17809$ 之间，每个词频模型文件 (包含 4 个 fold) 的大小在 $424 \sim 438$KB 之间，5 轮训练的时间一共在 $35 \sim 40$s 之间。

2.3 Predicting

在 $Predict.py$ 中实现了预测的过程：根据上一步生成的 $'./Prob'$ 下的词频统计文件，分别在其对应测试集的 flod 上做预测。取五轮预测结果的平均值作为最终结果，默认输出到命令行，修改 $Script.py$ 中的 record 参数为 True 可以将预测结果记录到 $'./Record/'$ 文件夹下。

Naïve Bayes 基于条件独立性假设

$$P(x|c_k)=P(a_1|c_k)P(a_2|c_k)\cdots P(a_m|c_k)=\prod_{l=1}^{m}P(a_l|c_k),\quad k=spam,ham$$

预测问题转化为求

$$\hat{c_k}=\arg\max_{c_k}P(c_k)\prod_{l=1}^{m}P(a_l|c_k)$$

其中 $P(c_k)$ 和 $P(a_l|c_k)$ 根据 2.2 节生成的词频模型文件分别由下列公式可得：

$$P(Y=c_k)=\frac{\sum_{i=1}^{n}I(y_i=c_k)+\alpha}{N+K\alpha},\quad k=spam,ham$$

$$P(X^j=a_{jl}|Y=c_k)=\frac{\sum_{i=1}^{n}I(x_i^j=a_{jl},Y=c_k)+\alpha}{\sum_{i=1}^{n}I(y_i=c_k)+M\alpha},\quad \alpha>0$$

其中 K 为 label 类别数，即 2。M 为特征的维数，即各轮训练中统计出的 words 字典的 keys 数目。α 为平滑值，在后续的 Issue 部分详细讨论。

Basic Words 同 2.2, 匹配出基本单词。

Ignore Stop Words 同 2.2, 词频统计文件中 value 为 0 的是停用词，预测阶段跳过。

Delete Long Nonsense Words 同 2.2，去掉文本中长度超过 20 的 word。

Extract Http Pattern 同 2.2，增加 http 的权重：

$$\hat{logc_k} = \arg\max_{c_k}(logP(c_k) + log \sum_{l=1,a_l \neq http}^{m} P(a_l|c_k) + \omega_{http} \times logP(a_l = http|c_k))$$

其中 ω_{http} 是为 http 增加的权重，这里把连乘取了对数是为了防止出现很多小概率的 word 连乘导致结果过小，造成精度损失。

实测中发现调整 ω_{http} 的值对预测准确率几乎没什么影响，于是去查看了词频文件发现原因在于统计出的 http 在 spam 和 ham 中占的比例是接近的，所以最终去掉了关于 http 的权重。

Extract Html Pattern 同 2.2，增加 html 的权重：

$$\hat{logc_k} = \arg\max_{c_k}(logP(c_k) + log \sum_{l=1,a_l \neq html}^{m} P(a_l|c_k) + \omega_{html} \times logP(a_l = http|c_k))$$

其中 ω_{html} 是为 html 增加的权重. 经过多次实验，确定了 $\omega_{html} = 10$ 是一个较为合适的权重。

Extract Received From Pattern 同 2.2，增加发件人的权重：

$$\hat{logc_k} = \arg\max_{c_k}(logP(c_k) + log \sum_{l=1,a_l \neq html,from}^{m} P(a_l|c_k) + \omega_{html} \times logP(a_l = http|c_k) + \omega_{from} \times logP(a_l = from|c_k))$$

其中 ω_{from} 是为发件人增加的权重. 经过多次实验，确定了 $\omega_{from} = 2$ 是一个较为合适的权重。

Extract Mailer Pattern 同 2.2，增加 Mailer 的权重：

$$\hat{logc_k} = \arg\max_{c_k}(logP(c_k) + log \sum_{l=1,a_l \neq html,from,mailer}^{m} P(a_l|c_k) + \omega_{html} \times logP(a_l = http|c_k) + \omega_{from} \times logP(a_l = from|c_k) + \omega_{mailer} \times logP(a_l = mailer|c_k))$$

其中 ω_{mailer} 是为 Mailer 增加的权重. 经过多次实验，确定了 $\omega_{mailer} = 200$ 是一个较为合适的权重。

Decode 同 2.2，跳过不能用 utf-8 解码的文件。

Process Parallel 同 2.2，加速预测过程。

5 轮预测的时间一共在 30 ~ 35s 之间。

3 Evaluate

对模型的评价使用了 accuracy, precision, recall, f1 四个指标。定义 spam 为正样本，ham 为负样本，那么：

True Positive: 将 spam 预测为 spam

False Positive: 将 ham 预测为 spam

True Negative: 将 ham 预测为 ham

False Negative: 将 spam 预测为 ham

相应地有：

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

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

$$recall = \frac{TP}{TP + FN}$$

$$f1 = \frac{2 \times precision \times recall}{precision + recall}$$

最终模型的表现为（见文件$'/Record/record_ver_41.yaml'$）：

Index	Accuracy	Precision	Recall	F1
Mean	0.981116945366739	0.9941509052458477	0.9751890077718119	0.984578403552655
Min	0.9806638246041413	0.9937841869716559	0.9741116751269036	0.984334525718515
Max	0.982089552238806	0.9948159668221876	0.9767618085375095	0.985474006116208

表 1: Final performance given by accuracy, precision, reacall and f1

实际上除了提取基本词并设置平滑值为 $\alpha = 1e-30$ 外，不做任何特征提取和特征过滤，保留最大规模的词频统计模型，就可以得到在四个评价指标意义上相对上面的模型更好的结果（见文件$'./Record/record_ver_3.yaml'$）：

Index	Accuracy	Precision	Recall	F1
Mean	0.9911468179273782	0.9988397983761399	0.9868252157168896	0.9927961072941031
Min	0.9899984846188816	0.9982771351218311	0.9856622114216282	0.991929567131328
Max	0.9916807887844709	0.9992542878448919	0.9874723655121592	0.9933283914010379

表 2: Better performance given by accuracy, precision, reacall and f1

这样虽然实现起来简单且指标更漂亮，但是问题在于会生成巨大的词频模型文件，从而将训练过程和预测过程的主要时间成本转移到了词频数据文件的序列化，反序列化和文件读写上，而涉及磁盘文件读写的操作时间成本通常是很大的。结果是整个 kflod → training → predicting 的 pipline 打开并行也要 10 分钟，同时还会生成几十 M 的词频统计文件，这也增加了模型的空间成本。

所以综合考虑模型性能和时间空间成本，在之后的分析和问题讨论中都统一使用前一个模型。

4 Analyze & Issue

4.1 Issue1

分别从 training set 中取出 1%('/Record/record_ver_45.yaml'), 5% ('/Record/record_ver_44.yaml'), 20%('/Record/record_ver_43.yaml'), 50% ('/Record/record_ver_42.yaml') 和 100%('/Record/record_ver_41.yaml') 进行训练。

预测的结果见Figure 2

sample=1%

Index	Accuracy	Precision	Recall	F1
Mean	0.9278985762226732	0.9571333995703301	0.9254045185516576	0.9404795739937377
Min	0.8973812423873325	0.931901391261899	0.8634253603713657	0.9129423921467321
Max	0.9392976082349379	0.9684845163058372	0.9643849456933569	0.9504019789734075

sample=5%

Index	Accuracy	Precision	Recall	F1
Mean	0.9643497494881856	0.9781797659965281	0.963833591144005	0.9709295768310922
Min	0.9592794429306691	0.9739504299443601	0.9565879664889566	0.9668760004925502
Max	0.9675700365408039	0.9854166666666667	0.9727203839353372	0.9739194318599241

sample=20%

Index	Accuracy	Precision	Recall	F1
Mean	0.9796185430168626	0.9886120961488359	0.9782993199834499	0.9834246828010148
Min	0.9765364819860732	0.9861932938856016	0.9743589743589743	0.9809932556713672
Max	0.9813450384072738	0.9912190082644629	0.9817258883248731	0.9848504137492042

sample=50%

Index	Accuracy	Precision	Recall	F1
Mean	0.9824897599291594	0.9922623894251655	0.9793040636714843	0.9857394869163979
Min	0.981501802790406	0.9915579432079816	0.9774111675126903	0.9849104859335038
Max	0.9844701583434835	0.9927835051546392	0.9828976301001711	0.9874815905743742

sample=100%

Index	Accuracy	Precision	Recall	F1
Mean	0.981116945366739	0.9941509052458477	0.9751890077718119	0.984578403552655
Min	0.9806638246041413	0.9937841869716559	0.9741116751269036	0.984334525718515
Max	0.982089552238806	0.9948159668221876	0.9767618085375095	0.985474006116208

Figure 2: Performance given by 1%, 5%, 20%, 50%, 100% training set

作出折线图Figure 3。其中所有实线代表均值，点迹代表最大值，点线代表最小值，红色为 Accuracy，橙色为 Precision，绿色为 Recall，青色为 F1。

可见总体上从各项指标来看，模型的表现是随着 training set 规模的增大而增强的，且 20% ~ 100% 规模的 training set 模型表现上差别不大，当规模继续下降到 5% 时，模型表现出现了明显的下滑，继续下降到到 1% 时，模型表现出现了大幅下滑。

此外，对比橙色和绿色线可知模型的 Precision 是高于 Recall 的，也就是说把 ham 误分为 spam 的概率和小于把 spam 误分为 ham 的概率。在实际的 spam 检测系统中，第一种情况是用户更不希望发生的，而第二种情况相对来是更能被用户容忍的。因此从这个

角度来看，模型的表现与实际 spam 检测系统的用户需求是一致的。当 sample 大于等于 50% 时，模型的 Precision 最小值都是大于 99% 的，也即 100 多封 ham 中才会出现一封被误判为 spam，这是可以接受的。

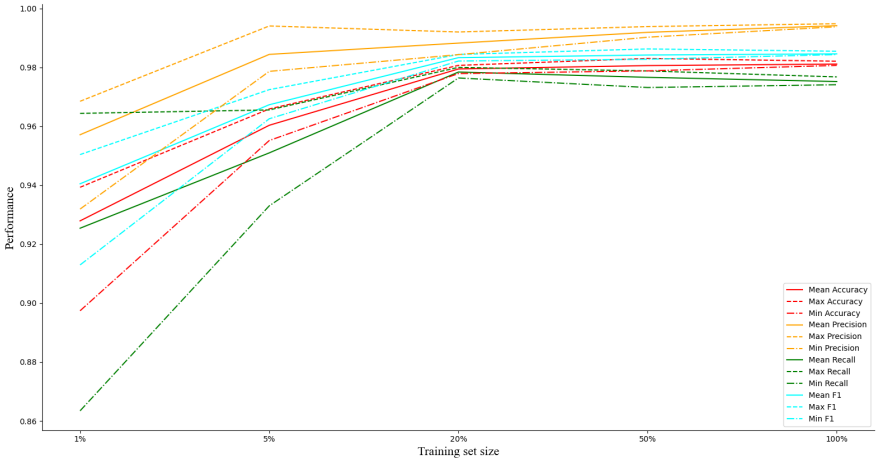

Figure 3: Performance given by 1%, 5%, 20%, 50%, 100% training set

4.2 Issue2

零概率问题是指在预测的文本中包含某一个 word，但是在某些类别中训练阶段没有出现过这个 word，其词频为 0。因为用贝叶斯公式算后验概率时，各个 word 出现的概率是连乘的，只要有一个概率为 0，那么整个连乘的结果就是 0，这意味着直接将当前的类别排除出预测结果了。

因此在计算 word 出现的概率时需要在分子分母上加上一个值 α 作为平滑，分母的 α 前可以乘上一个系数 M，一般为特征维度，这里就是 words 字典 keys 的数目。

$$P(X^j = a_{jl}|Y = c_k) = \frac{\sum_{i=1}^{n} I(x_i^j = a_{jl}, Y = c_k) + \alpha}{\sum_{i=1}^{n} I(y_i = c_k) + M\alpha}, \quad \alpha > 0$$

当 $\alpha = 0$ 时就等价于极大似然估计，$0 < \alpha < 1$ 时是 Lidstone smooth，$\alpha = 1$ 时是 Laplace smooth。

此外，后验概率的连乘还有一项是算 $P(Y = c_k)$，多类别分类任务时也可能出现某类别在 training set 中频数为 0 的情况，因此也需加上 α 做平滑，分母上的系数 K 可以取类别的数目。

$$P(Y = c_k) = \frac{\sum_{i=1}^{n} I(y_i = c_k) + \alpha}{N + K\alpha}, \quad k = spam, ham$$

当然对于本次实验来说两个类别出现的先验概率都不是 0，所以这里做不做平滑都没有关系。

以 accuracy 为性能指标考虑 α 取值的影响：

从Figure 4可见当 α 的取值小于 $1e-20$ 时，模型的 accuracy 变化不大，当 α 取 $1e-10$ 时，accuracy 开始出现明显下降，大于 $1e-10$ 时大幅下降。

4.3 Issue3

主要的思路和理由在 Implement 的部分介绍过了，这里补充一些实验结果。

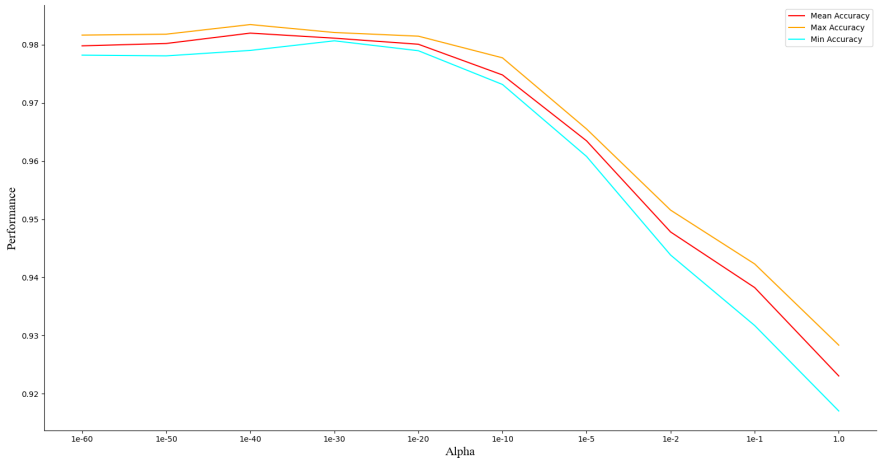

Figure 4: Performance given by 1e-60 ∼ 1.0 alpha

Select Top Rank Valid Features 这项主要是权衡模型表现与时间空间成本，效果是类似于 Issue1 中调整 sample 比例的，不再详细展开。

Delete Fuzzy features 同上。

Ignore Stop Words 使用与不使用 Stop Words 的对比Figure 5：

Without Stop Words

Index	Accuracy	Precision	Recall	F1
Mean	0.978960027787201	0.9945847767774749	0.9712462488076232	0.9827757715780632
Min	0.9780502573418105	0.99425	0.9695431472081218	0.982096555130263
Max	0.9809897879025923	0.9950507944777286	0.974993685274059	0.9845682948603494

Using Stop Words

Index	Accuracy	Precision	Recall	F1
Mean	0.981116945366739	0.9941509052458477	0.9751890077718119	0.984578403552655
Min	0.9806638246041413	0.9937841869716559	0.9741116751269036	0.984334525718515
Max	0.982089552238806	0.9948159668221876	0.9767618085375095	0.985474006116208

Figure 5: Performance with stop words and without stop words

可见使用 Stop Words 是有助于提升模型性能的。

Delete Long Nonsense Words 删除或者不删除 Long Nonsense Words 对比Figure 6：
可见删除 Long Nonsense Words 是有助于提升模型性能的。

Extract Http Pattern 实验发现对模型表现影响不大，略去不表。

Extract Html Pattern 以 accuracy 为性能指标考虑 ω_{html} 权重的影响Figure 7：
可见以 $\omega_{html} = 10$ 为权重可以较好地提升模型表现。

Preserving Long Nonsense words

Index	Accuracy	Precision	Recall	F1
Mean	0.9799839443874967	0.995364687699133	0.9721543843076107	0.9836188257722828
Min	0.9771124000627057	0.9942913874410524	0.9675126903553299	0.9812097812097812
Max	0.9824402058734484	0.997145081754477	0.9773115393998536	0.9857283464566928

Deleting Long Nonsense words

Index	Accuracy	Precision	Recall	F1
Mean	0.981116945366739	0.9941509052458477	0.9751890077718119	0.984578403552655
Min	0.9806638246041413	0.9937841869716559	0.9741116751269036	0.984334525718515
Max	0.982089552238806	0.9948159668221876	0.9767618085375095	0.985474006116208

Figure 6: Performance with deleting long nonsense words and preserving long nonsense words

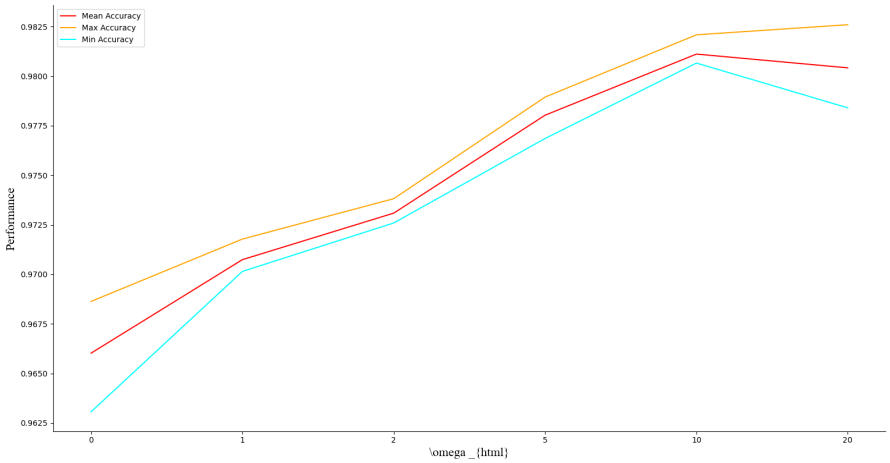

Figure 7: Performance given by html weight

Extract Received From Pattern　以 accuracy 为性能指标考虑 ω_{from} 权重的影响Figure 8：

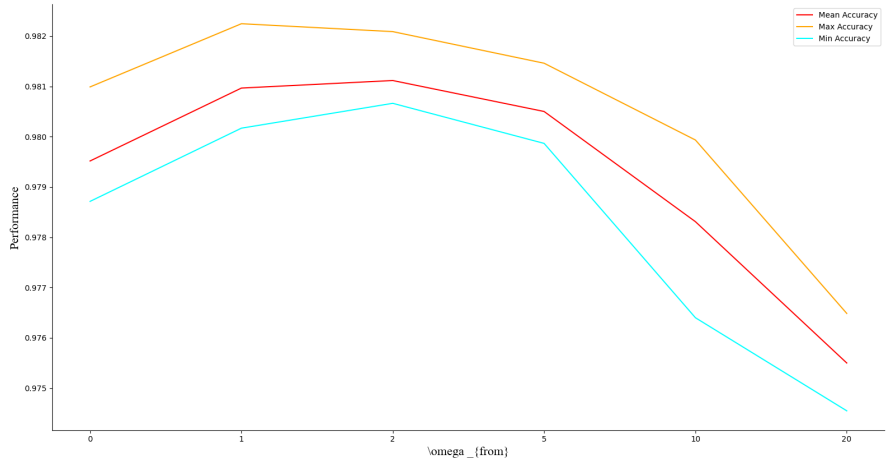

Figure 8: Performance given by from weight

可见以 $\omega_{from} = 2$ 为权重可以较好地提升模型表现。

Extract Mailer Pattern　以 accuracy 为性能指标考虑 ω_{mailer} 权重的影响Figure 9：

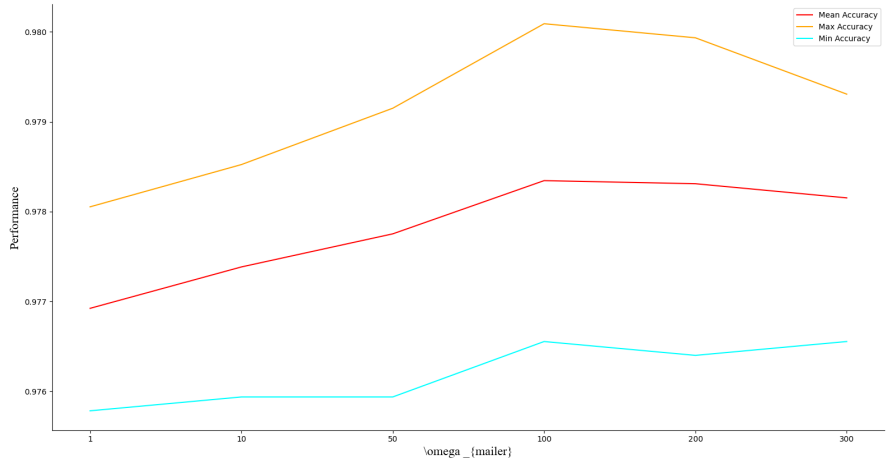

Figure 9: Performance given by mailer weight

可见以 $\omega_{mailer} = 200$ 为权重可以较好地提升模型表现。

5 Conclusion

在实现 Naïve Bayes Classifier 过程中，我一方面充分考虑了模型性能与时间空间成本的关系，以较小的时间空间代价取得了较高的分类准确度。另一方面我也在维持模型泛化能力的基础上对一些特殊的具有高区分度的特征做了权重加强的处理，也使得模型对于 meta data 的利用更加充分。

但是，我实现的模型仍有很多不足之处。首先是模型超参数众多，容易造成过拟合。超参数的取值依赖于在有限的 training set 和 testing set 上实验，背后缺乏严谨的数学推导。其二是可供利用的 specific features 还有很多，我对于此发掘地还不够深入。