“文本分类”项目报告

李拙 李泽斌 刘子衡

项目描述

文本分类，对于给定的文本数据集，分别实现基于概率统计的、基于决策树的和基于神经网络的分类算法，并分析这三种算法的理论复杂度和实际开销。

项目地址

代码上传在 GitHub 中，地址为 https://github.com/Lizhmq/News_Classify .

数据集

使用搜狐新闻数据(SogouCS)，地址为：

http://www.sogou.com/labs/resource/cs.php

数据介绍：

来自搜狐新闻 2012 年 6 月至 7 月期间国内、国际、体育、社会、娱乐等 18 个频道的新闻数据，提供 URL 和正文信息。

数据格式：

```
<doc>
<url>页面URL</url>
<docno>页面ID</docno>
<contenttitle>页面标题</contenttitle>
<content>页面内容</content>
</doc>
```

数据预处理

下载的数据是 GBK 编码的，需要转换为 utf-8 编码。

```
cat news_tensite_xml.dat | iconv -f gbk -t utf-8 -c | grep "<content>"  > corpus.txt
```

数据的格式较为规整，可以方便地从中提取出 url、标题和内容等信息，用 python 或者 shell 命令处理。

```
for i in range(0,len(lines),6):
    news={}
    news["title"] = lines[i+3][14:-16]
    news["content"] = lines[i+4][9:-11]
    news["label"] = urltolabel(lines[i+1])
    News.append(news)
```

去除标签：一般地，需要去除网页内容中的 html 标签，此处的数据页面内容不含 html 标签，因此不需要处理。

数据清洗：原数据中有很多我们不关心的无用符号，比如数字、表情符号和颜文字等，需要清除。

```
rubbish="[0-9\\s+\\.\\!\\/_,$%^*()?;；:-【】+\"\']+|[+——！，；：。？、~@#￥%……&*（）A-Za-z0-9*%=／+]+"
```

分词：由于使用的是中文数据集，需要分词以进行后续处理。在这里使用 jieba 库可以方便地得到分词结果。

```
def newsprocess(news):
    news = news.replace("?"," ")
    news = re.sub(rubbish," ",news)
    news = '|'.join(jieba.cut(news)).replace("| ","")
    # print(news)
    return news
```

停用词：为得到更有意义的数据，我们讲内容中的停用词去掉，在 GitHub 下载常用的停用词表，筛选出内容中出现的停用词并删除。

```
with open('stopwords/stopwords.txt', encoding='utf-8') as f:
    stopwords = f.read()
stopwords = stopwords.split('\n')
stopwords = set(stopwords)
# remove stopwords
words = [word for word in words if word not in stopwords]
```

最终，我们通过对 url 信息的抽取提取了数量较多的八个类，各选择了 2000 条数据，作为我们的训练集和测试集。分别是以下八类：汽车、健康、IT、学习、体育、财经、旅游和娱乐。

朴素贝叶斯

基于贝叶斯公式（后验概率）的机器学习，理论较简单，可参考周志华老师的《机器学习》。

针对文本分类问题，设x是某个文本，y_i是类别i，取最大化P(y_i|x)的类别作为预测结果，用统计学的方法，以频率估计概率来计算。将文本x看作词x_i组成的向量。贝叶斯公式：

$$P(y_i|X) = \frac{P(Xy_i)}{P(X)} = \frac{P(X|y_i)P(y_i)}{P(X)}$$

问题在于文本的特征空间过大，训练样本不足以实现概率估计。因此，引入简化问题的独立性假设（朴素贝叶斯中朴素二字的由来）：

$$P(y_i|X) = P(y_i|x_1) \dots P(y_i|x_n)$$

对于文本分类问题，提出两个模型：伯努利模型、多项式模型。

伯努利模型

将训练集中任何一个单词都视为一个特征（某个文本含有或者不含这个单词），每个文本表示为一个n维 0-1 向量，因此统计时单词的重复出现不用计数。

$P(y_i) = y_i$类文本数/总文本数

$P(x_i|y_i)$ =类y_i下含单词x_i的文档个数 / 类y_i下文档总数

为避免一个类中某个单词不存在而导致整个类被否定的情况，引入拉普拉斯平滑：

$P(x_i|y_i)$ =(类y_i下含单词x_i的文档个数+1) / (类y_i下文档总数+2)

多项式模型

伯努利模型将文本视为个体进行统计，而多项式模型将单词视为个体进行统计，所以这里一个文本中单词的重复出现需要统计。

$P(y_i)$ =类y_i下单词数 / 所有训练集单词总数

$P(x_i|y_i)$=类y_i下文档中x_i出现的次数之和 / 类y_i下单词总数

同样地，引入拉普拉斯平滑：

$P(x_i|y_i)$=(类y_i下文档中x_i出现的次数之和+1) / (类y_i下单词总数 + V)

其中 V 为训练样本中不同的词类别数。

二者在计算后验概率时略有不同。对于一个文档 doc，多项式模型中，只有在 doc 中出现过的单词，才会参与后验概率计算，伯努利模型中，没有在 doc 中出现，但是在全局单词表中出现的单词，也会参与计算，不过是作为“反方”参与的。

分别对这两个统计模型，编写代码，对文本分类问题进行测试。为了避免每次重复地冗余训练，我们实现了统计数据的保存与读取。

除自己编写统计模型外，我们还尝试调用 sklearn 的库函数，效率和准确率都要高于我们自己编写的代码。

复杂度分析

设训练集的词个数为 M，文本类别数为|Y|。

这里我们假设 python 的字典访问时间是常数时间的（实际可能不是，但差距不会过大），那么，训练（也就是统计）的时间复杂度是 O(M)，测试一个文本（长度为 m）的时间复杂度为 O(|Y|*m)的，算法的空间消耗是 O(M)的。我认为朴素贝叶斯是能预见的最高效的算法，而且简单，易于理解。

并且，在实际测试中，算法给出了非常好的结果:

correctness: 94.10368519675203%
train time: 7.988204958024698
test time: 30.226281481481486

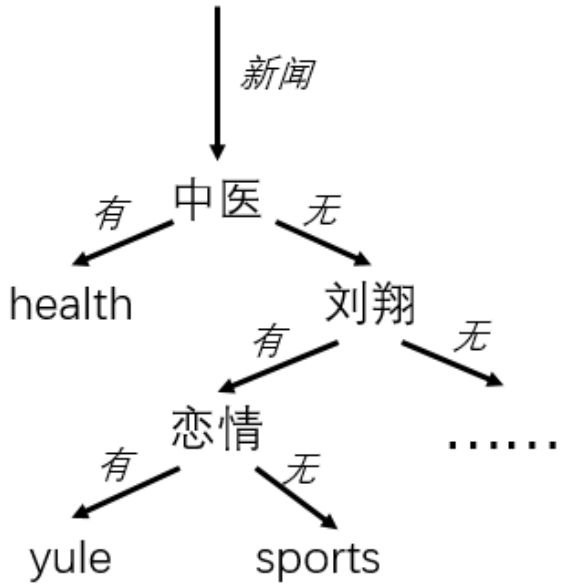
在实际测试中，由训练集和测试集的划分：M:m = 7:3， 类别数|Y|=8，考虑到测试时访问字典次数略多于训练，常数也略大。可以看到，实际测试与理论分析吻合地不错。

而调用 sklearn 库函数的结果更好，训练和测试时间更短，准确率更是达到了

95%-96%，更加体现了朴素贝叶斯法的高效性以及—简单的哲学。

决策树

决策树是一种树形结构，其中每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。新闻分类问题里，决策树的结构大致如下图所示。



模型的参数

1、alpha 参数：选择信息增益最大的词时，若计算所有词，时间开销过大。所以只计算出现次数前 alpha 大的词汇。

2、beta 参数：当构建决策树时，一个节点新闻数量小于等于 beta 时，就不再继续细分。即节约了时间，又防止了过拟合现象。

实验内容

取 alpha=100,200,400，beta=10,20,30,40,50 进行 15 组测试，测试内容为训练准确率、测试准确率、树的节点个数、树的深度、训练时间、测试时间，每组都测 5 次取平均值。

实验结果

alpha	beta	train accuracy	test accuracy	nodes	depth	train time	test time
100	10	97.44%	89.62%	2742.6	563.4	114.02	0.3021
100	20	96.13%	89.88%	2107.0	539.6	109.51	0.2870
100	30	95.23%	90.16%	1788.2	531.6	111.74	0.2836
100	40	94.94%	90.12%	1547.0	521.2	95.37	0.2589
100	50	94.73%	90.09%	1410.6	516.2	99.90	0.2640
200	10	97.87%	90.57%	2562.6	624.0	237.05	0.3288
200	20	96.90%	91.06%	2059.8	620.6	233.10	0.3191

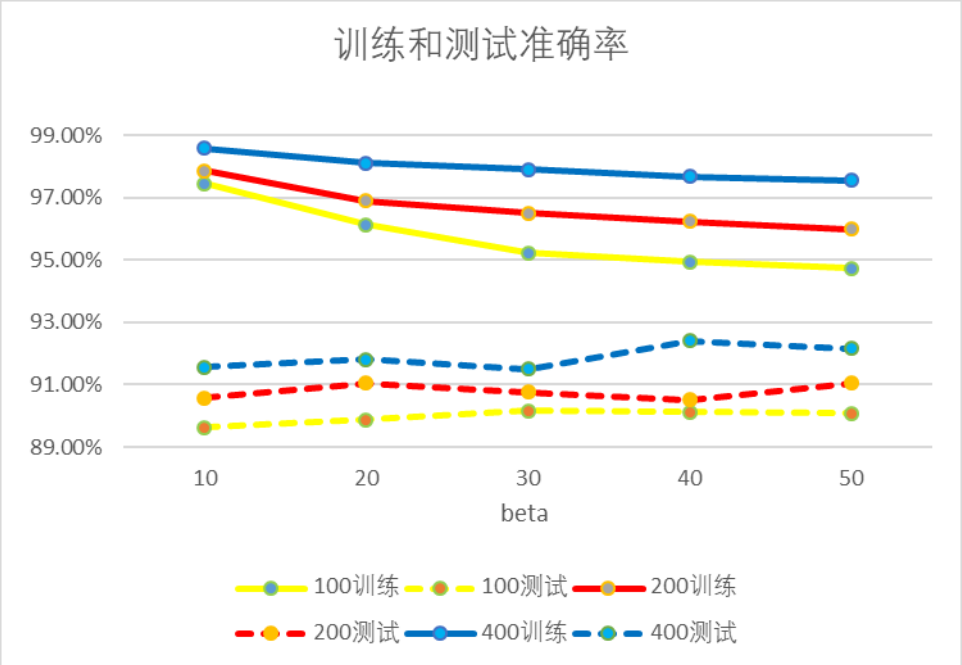
200	30	96.50%	90.77%	1744.6	609.6	229.31	0.3222
200	40	96.24%	90.51%	1537.8	603.0	224.50	0.3208
200	50	95.98%	91.06%	1427.8	590.2	231.73	0.3495
400	10	98.58%	91.57%	2436.2	737.6	597.01	0.4229
400	20	98.11%	91.82%	1979.0	732.4	632.42	0.4186
400	30	97.91%	91.50%	1730.6	713.4	625.00	0.4320
400	40	97.67%	92.40%	1630.6	713.4	707.29	0.4654
400	50	97.56%	92.16%	1543.4	692.6	786.99	0.4903

数据分析

1、可以看出训练准确率和测试准确率和 alpha 正相关，alpha 越大，准确率越高。因为 alpha 代表每次挑选信息增益最大词的候选词数，故 alpha 大使得选词更为精准，符合预期。

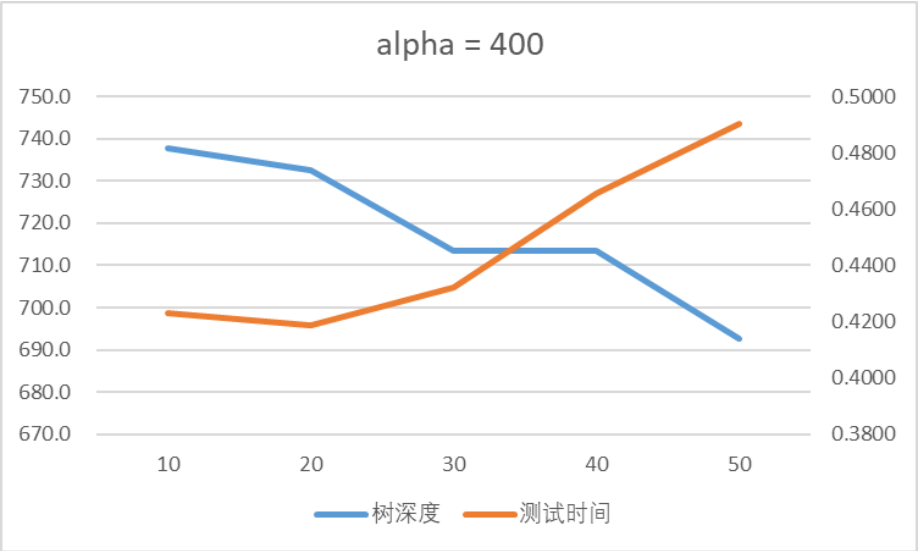
2、训练准确率与 beta 负相关，而测试准确率往往实在某个 beta 的中间值取得最大值。beta 代表着剪枝的力度，所以增大 beta 可以减少过拟合的现象，实验结果符合预期。

3、决策树的节点个数与 alpha 无关，与 beta 负相关，这也很好理解，剪枝使得节点减少。同理，决策树的深度也与 beta 负相关，但与 alpha 正相关，也就是说 alpha 虽然不影响决策树的大小，但是会使树的深度增大，这是因为在选词上变得精确，能选出极具代表性的词，使得含有这个词的子节点成为叶节点。计算 log(nodes)/depth 即可估计是决策树的疏密程度，得 alpha =100，200，400 时，这个值分别为 0.0203，0.0178，0.0151。



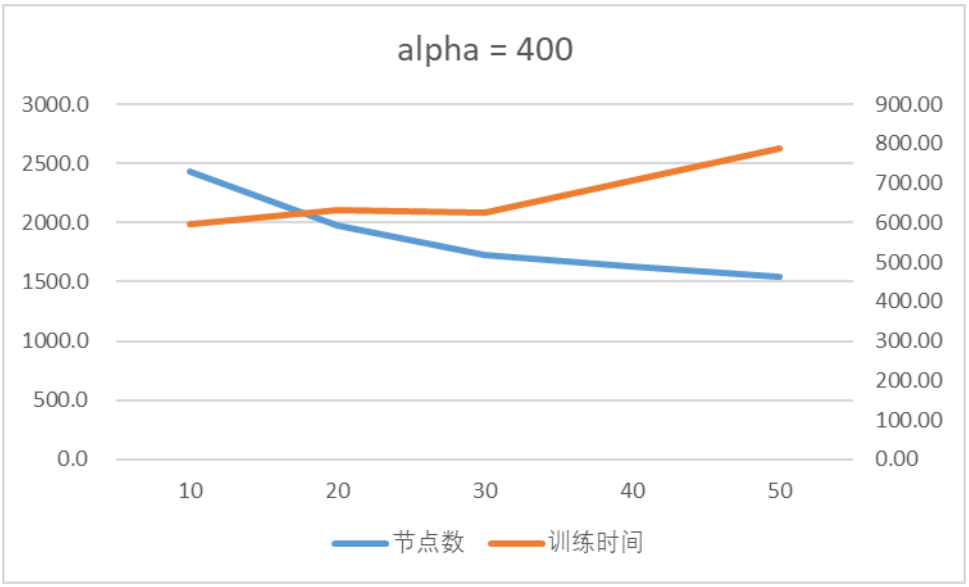
4、训练时间与 alpha 成线性关系，意味着训练的主要开销就是在选择信息增益最大的词上。

5、猜想训练时间也应与树的节点个数正相关，测试时间与树的深度正相关。但是实验数据并不十分符合，目前还不确定原因。



总结

决策树最后的准确率可以达到 92%左右，准确率相对另两种模型较低。但决策树具有结构简单、测试时间短和空间占用小的优点。采用的剪枝策略较为简单，对准确率有略微提升，若要优化可以从其他剪枝策略入手。



神经网络

最初的选择——FastText

FastText 是 Facebook 于 2016 年开源的一个词向量计算和文本分类工具，在学术上并没有太大创新。但是它的优点也非常明显，在文本分类任务中，FastText（浅层网络，三层）往往能取得和深度网络相媲美的精度，却在训练时间上比深度网络快许多数量级。

FastText t 算法思想：将整篇文档的词及 n-gram 向量叠加平均得到文档向量，然后使用文档向量做 softmax 多分类。

基本模型：改进的 CBOW（三层）+ Softmax

改进之处：

●字符级别的 N-grams

●分层 Softmax 以降低模型训练时间

FastText 的架构和 word2vec 中的 CBOW 的架构类似，因为它们的作者都是 Facebook 的科学家 Tomas Mikolov，而且确实 FastText 也算是 words2vec 所衍生出来的。

但是通过 python 调用，表现并无特别之处，正确率与训练时间都不尽人意。经过思考权衡后后，结论如下：

1.数据集及数据种类太少无法体现 fasttext 算法优越性

2.封装函数本身对过拟合现象考虑不够，封装函数又难以直接改进

3.n-grams 技术复杂度高

基于以上思考，在进一步了解 softmax 与 CBOW 的基础上，我们决定采用 word2vec+softmax 的算法框架。

Word2Vec 原理

Word2Vec 是从大量文本语料中以无监督的方式学习语义知识的一种模型，它被大量地用在自然语言处理（NLP）中。Word2Vec 其实就是通过学习文本来用词向量的方式表征词的语义信息，即通过一个嵌入空间使得语义上相似的单词在该空间内距离很近。

Skip-gram 模型通过统计一个词的上下文出现的词的概率，来训练出一个可以预测一个词上下文出现某个词概率的模型。但这个模型的最终结果并不重要，重要的是得到的中间隐藏层，就是代表每个词的词向量。

直觉上的理解是如果两个不同的单词有着非常相似的“上下文”，那么通过我们的模型训练，这两个单词的嵌入向量将非常相似。

算法实现

对完整的搜狗新闻数据集（1.59G）进行转码、清洗，调用 jieba 切词处理，再调用 gensim 库的 word2vec 模型进行训练，得到每个词对应的 400 维向量。训练全部的语料需要花费一个小时左右。

将训练集和测试集的每条新闻转化成 400 维向量，即对一条新闻内所有词对应的向量取平均值，来代表整条新闻。

调用 keras，训练一个单层神经网络，激活函数为 softmax，将 400 维的向量转化为一个 8 维向量。8 维向量即对应该新闻属于某种类别的概率。

```
50    model = Sequential()
51    model.add(Dense(8,input_dim = 400,kernel_initializer='he_normal'))
52    model.add(Activation('softmax'))
53    model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =['accuracy'])
54    model.fit(trainX, trainY, epochs=10, batch_size=50, verbose=1)
55    loss, accuracy = model.evaluate(testX, testY)
```

最终结果

测试准确率:	0.9636212278900981
单层神经网络训练时间:	28.880191051851853
测试时间:	0.2973137382716686
载入 word2vec 模型时间:	15.2997147654321
将新闻转化为向量的时间:	35.30542704197528
训练 word2vec 的时间:	一个半小时左右

改进方向

- 考虑特征的依赖关系，实现半朴素贝叶斯分类器或搭建贝叶斯网
- 实现更复杂的剪枝算法
  - 详细介绍：https://www.cnblogs.com/starfire86/p/5749334.html
- 运用 TF-IDF 方法从文本提取特征
- 实现 Word2Vec 模型
- 使用其他的数据集训练模型并进行测试

项目收获

- 简单即优美--《数学之美》by 吴军
- 良好的开发风格：好的命名习惯，必要的代码注释，友好便于阅读的文档
- 使用规范的编码格式—数据集、含中文注释的 python 代码的使用中都出现编码的问题，统一为 utf-8
- 注意平台和软件版本的不同—在 Windows 和 Linux 兼容性上出现一些问题，以及 python2 和 python3 中一些细微的差别
- 工欲善其事，必先利其器—利用正则表达式，高效提取文本信息和清洗数据；在服务器上训练模型，更加高效
- 阅读源代码，查询 API，学习并利用已有的成果—如 Google 的 Word2Vec 和 Facebook 的 fastText 开源代码

参考资料

- 搜狗实验室--搜狐新闻数据集
- 《机器学习》—周志华
- Facebook 的 GitHub 开源代码
  - https://github.com/facebookresearch/fastText
- 关于词嵌入学习(Word Embedding)和 Word2Vec 与 FastText 的论文
  - https://github.com/Lizhmq/News_Classify/tree/master/Word2Vec&NN/Papers
- CSDN 博客