程序报告

姓名：秦嘉俊　学号：3210106182

一、问题重述

1.1 背景

垃圾短信 (Spam Messages，SM) 是指未经过用户同意向用户发送不愿接收的商业广告或者不符合法律规范的短信。随着手机的普及，垃圾短信在日常生活日益泛滥，已经严重的影响到了人们的正常生活娱乐，乃至社会的稳定。据 360 公司 2020 年第一季度有关手机安全的报告提到，360 手机卫士在第一季度共拦截各类垃圾短信约 34.4 亿条，平均每日拦截垃圾短信约 3784.7 万条。

大数据时代的到来使得大量个人信息数据得以沉淀和积累，但是庞大的数据量缺乏有效的整理规范；在面对量级如此巨大的短信数据时，为了保证更良好的用户体验，如何从数据中挖掘出更多有意义的信息为人们免受垃圾短信骚扰成为当前亟待解决的问题。

1.2 任务

在这个实验里，我们的任务是根据一段中文文本（200 个中文字符以内），预测这段文本是否为垃圾短信。

具体完成数据读取、基础模型、模型训练等基本代码，搭建并调整模型，将模型训练至最佳状态。

二、设计思想

2.1 数据处理

2.1.1 数据集

数据集，又称为资料集、数据集合或资料集合，是一种由数据所组成的集合。

`label`	`message`(短信内容)	`msg_new`(短信分词后)
0	人们经常是失去了才发现它的珍贵	人们经常是失去了才发现它的珍贵
1	本人现在承办驾驶证业务! 招收学员，一对一教学	本人现在承办驾驶证业务! 招收学员，一对一教学

为了处理读入的数据集，我们这里采用中文分词工具 jieba, 可以将中文句子分为若干个 tokens, 以便我们后面进行判断。

2.1.2 停用词

停用词是指在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词，这些字或词即被称为 **Stop Words(停用词)**

这些停用词都是人工输入、非自动化生成的，生成后的停用词会形成一个停用词库。本次实验中我们采用的是四川大学机器智能实验室停用词库

2.1.3 文本向量化

1. `CountVectorizer`

   目前拥有的数据是长度不统一的文本数据，而绝大多数机器学习算法需要的输入是向量，因此文本类型的数据需要经过处理得到向量。

   我们可以借助 sklearn 中 **CountVectorizer** 来实现文本的向量化，CountVectorizer 实际上是在统计每个词出现的次数 **，这样的模型也叫做**词袋模型**。

2. `TfidfVectorizer`

   与 CountVectorizer 类似的还有 TfidfVectorizer

   TF-IDF 算法是创建在这样一个假设之上的：

   对区别文档最有意义的词语应该是那些在文档中出现频率高的词语，因此选择特征空间坐标系取 TF 词频作为测度，就可以体现同类文本的特点。另外考虑到单词区别不同类别的能力，TF-IDF 法认为一个单词出现的文本频数越小，它区别不同类别文本的能力就越大。因此引入了逆文本频度 IDF 的概念，以 TF 和 IDF 的乘积作为特征空间坐标系的取值测度，并用它完成对权值 TF 的调整，调整权值的目的在于突出重要单词，抑制次要单词。在本质上 IDF 是一种试图抑制噪声的加权，并且单纯地认为文本频率小的单词就越重要，文本频率大的单词就越无用。

   其中 TF、IDF 和 TF-IDF 的含义如下：

   - TF: 词频

   $$TF(w) = \frac{\text{词}w\text{在文档中出现的次数}}{\text{文档的总词数}} \tag{1}$$

   - IDF: 逆向文件频率

     有些词可能在文本中频繁出现，但并不重要，也即信息量小，如 is, of, that 这些单词，这些单词在语料库中出现的频率也非常大，我们就可以利用这点，降低其权重。

     $$IDF(w) = \ln \frac{\text{语料库的总文档数}}{\text{语料库中词}w\text{出现的文档数}} \tag{2}$$

   - TF-ID 综合参数：$TF - IDF = TF * IDF$

2.1.4 划分训练集和测试集

一般的数据集会划分为两个部分：

- 训练数据：用于训练，构建模型
- 测试数据：在模型检验时使用，用于评估模型是否有效

划分比例：

- 训练集：70% 80% 75%
- 测试集：30% 20% 25%

2.2 朴素贝叶斯算法的原理

朴素贝叶斯实现分类的原理是基于贝叶斯公式，给定一个样本，计算该样本条件下每个类别的条件概率。

$$P(y|x_1,...,x_n) = \frac{P(y)P(x_1,...,x_n|y)}{P(x_1,...,x_n)} \tag{3}$$

由于假设每个特征是独立的，所以该公式可以化为：

$$P(x_i|x_1,...x_{i-1},x_{i+1},...,x_n) = P(y)\prod_{i=1}^{n} P(x_i|y) \tag{4}$$

由于分母是确定的，结果只和分子有关。

$$P(x_i|x_1,...,x_n) \propto P(x_i|y) \tag{5}$$

求出最大的条件概率，其对应的类别就是该样本所属的类别。

$$\widehat{y} = arg\max_{y} P(y)\prod_{i=1}^{n} P(x_i|y) \tag{6}$$

2.3 logistics 回归

逻辑回归（Logistic Regression）是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。比如某用户购买某商品的可能性，某病人患有某种疾病的可能性，以及某广告被用户点击的可能性等。注意，这里用的是“可能性”，而非数学上的“概率”，logisitc 回归的结果并非数学定义中的概率值，不可以直接当做概率值来用。该结果往往用于和其他特征值加权求和，而非直接相乘。

我们考虑 1 个输入的维数据，我们对输入数据进行线性加权得到前面说到，logistic 回归用于分类，假设得到的类别为 0 或者 1，那么可以使用 sigmoid 函数处理输入，这个函数类似于阶跃函数但是又是连续型函数

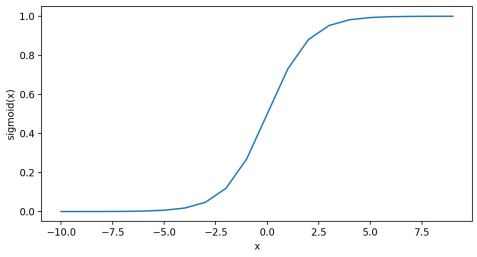

图 1: sigmoid 函数

作为 sigmoid 函数的输入，得到

$$f(x) = \frac{1}{1+e^{-g(x)}} = \sigma(g(x)) = \sigma(w^\top x)$$

这样就得到了输入数据 x 最终属于类别 1 的概率。

我们先考虑使用常规的均方差作为损失函数，这时候的损失函数为

$$L(w) = \frac{1}{2}(y - f(x))^2 = \frac{1}{2}(y - \sigma(w^\top x))^2$$

采用梯度下降的方法对 w 进行更新，那么需要将损失函数对 w 求导得到

$$\frac{\partial L}{\partial w} = (y - \sigma(w^\top x))\sigma^{'}(w^\top x)x$$

这里的梯度更新中包含了 $(w^\top x)$ ，而通过 sigmod 函数可以发现，当 $(w^\top x)$ 位于 0 或者 1 附近的时候，导数值基本趋近于 0，梯度收敛速度极慢。

所以在这种情况下我们可以考虑使用交叉熵作为损失函数。将作为输入数据的输出，对上做个简单的变换

$$\ln \frac{f(x)}{1-f(x)} = w^\top x$$

将 $f(x)$ 视为类后验概率估计 $P(y=1|x)$，则上式可以重写为

$$\ln \frac{P(y=1|x)}{P(y=0|x)} = w^\top x$$

那么从而可以得到

$$P(y|x,w) = [f(x)]^y \cdot [1-f(x)]^{1-y}$$

设输入数据 $X = \{x_1, x_2, \ldots, x_m\}, y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{bmatrix}$ 其中 x_i 表示第 i 个输入数据，上式的似然函数为

$$L(w) = \prod_{i=1}^{n} [f(x)]^{y_i} \cdot [1-f(x)]^{1-y_i}$$

然后我们的目标是求出使这一似然函数的值最大的参数估计，最大似然估计就是求出参数 $w_0, w_1, \ldots, w_n$，使得上式取得最大值，对上式求导得到

$$\ln L(w) = \sum_{i=1}^{m} (y_i \ln[f(x_i)] + (1 - y_i) \ln[1 - f(x_i)])$$

sklearn 实现 logistic 回归使用的是 linear_model 模型。实际使用中可以直接调用这个模型。

2.4 决策树

决策树是一种解决分类问题的算法。决策树算法采用树形结构，使用层层推理来实现最终的分类。决策树由下面几种元素构成：

- 根节点：包含样本的全集
- 内部节点：对应特征属性测试
- 叶节点：代表决策的结果

预测时，在树的内部节点处用某一属性值进行判断，根据判断结果决定进入哪个分支节点，直到到达叶节点处，得到分类结果。这是一种基于 if-then-else 规则的有监督学习算法，决策树的这些规则通过训练得到，而不是人工制定的。

决策树是最简单的机器学习算法，它易于实现，可解释性强，完全符合人类的直观思维，有着广泛的应用。其三个步骤分别是

1. 特征选择
   特征选择决定了使用哪些特征来做判断。在训练数据集中，每个样本的属性可能有很多个，不同属性的作用有大有小。因而特征选择的作用就是筛选出跟分类结果相关性较高的特征，也就是分类能力较强的特征。

   在特征选择中通常使用的准则是：信息增益。

2. 决策树生成
   选择好特征后，就从根节点触发，对节点计算所有特征的信息增益，选择信息增益最大的特征作为节点特征，根据该特征的不同取值建立子节点；对每个子节点使用相同的方式生成新的子节点，直到信息增益很小或者没有特征可以选择为止。

3. 决策树剪枝
   剪枝的主要目的是对抗“过拟合”，通过主动去掉部分分支来降低过拟合的风险。

2.5 逻辑回归和决策树的异同和性能差异

二者都是用来解决分类问题的机器学习基本算法。
从实质上看，决策树和逻辑回归的分歧是：

1. 逻辑回归对数据整体结构的分析优于决策树，而决策树对局部结构的分析优于逻辑回归。
2. 逻辑回归擅长分析线性关系，而决策树对线性关系的把握较差。虽然对付非线性关系是决策树的强项，但是很多非线性关系完全可以用线性关系作为近似，而且效果很好。线性关系在实践中有很多优点：简洁，易理解，可以在一定程度上防止对数据的过度拟合。
3. 逻辑回归对极值比较敏感，容易受极端值的影响，而决策树在这方面表现较好。
4. 执行速度上当数据量很大的时候，逻辑回归的执行速度非常慢，而决策树的运行速度上明显快于逻辑回归。

三、代码内容

3.1 模型的搭建和训练

我们有一个文件 train.py 用来搭建和训练我们的模型，该文件会将训练后的模型导出到 results/pipeline.model, 下面是这个文件的各个部分

3.1.1 停用词的读入

```
import os
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"

# ---------- 停用词库路径，若有变化请修改 ----------
stopwords_path = r'scu_stopwords.txt'
# -----------------------------------------------------

def read_stopwords(stopwords_path):
    """
    读取停用词库
    :param stopwords_path: 停用词库的路径
    :return: 停用词列表，如 ['嘿', '很', '乎', '会', '或']
    """
    stopwords = []
    # ---------- 请完成读取停用词的代码 ----------
```

```
    with open(stopwords_path, 'r', encoding='utf-8') as f:
        stopwords = f.read()
    stopwords = stopwords.splitlines()
    #---------------------------------------------

    return stopwords

# 读取停用词
stopwords = read_stopwords(stopwords_path)
```

这里我们利用 Python 中的 `open` 函数打开文件，其中的参数'r' 代表只读，'utf-8' 代表我们打开文件所用的编码格式，因为是中文文本，所以要使用 utf-8. 接着再使用 `read` 函数，如果文件是以文本模式（非二进制模式）打开的，则 `read()` 函数会逐个字符进行读取；反之，如果文件以二进制模式打开，则 `read()` 函数会逐个字节进行读取。这样我们就读入了停用词。

再用 `splitlines` 方法，去掉文本中的换行符即可。

3.1.2 读入数据集

```
# 导入相关的包
from copy import deepcopy
import warnings
warnings.filterwarnings('ignore')
import os
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"
import pandas as pd
import numpy as np
# 数据集的路径
data_path = "./datasets/5f9ae242cae5285cd734b91e-momodel/sms_pub.csv"
# 读取数据
sms = pd.read_csv(data_path, encoding='utf-8')
# 显示前 5 条数据
sms.head()
# 显示数据集的一些信息
sms.groupby('label').describe()
```

这里的代码 main.ipynb 已给出，不需要做修改

3.1.3 构建训练集和测试集

```
from sklearn.model_selection import train_test_split
```

```
X = np.array(sms.msg_new)
y = np.array(sms.label)


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=66666,
    test_size=0.5)
print("总共的数据大小", X.shape)
print("训练集数据大小", X_train.shape)
print("测试集数据大小", X_test.shape)
```

这里的代码 main.ipynb 也已给出,基本不需要修改。我们可以调节 random_state 和 test_size 的大小来改变我们的采样结果和测试集样本占比。

3.1.4 构建 Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import ComplementNB
from sklearn.naive_bayes import CategoricalNB
# pipline_list用于传给Pipline作为参数
pipeline_list = [
      ('cv', TfidfVectorizer(max_df=0.95, max_features=5000, norm='l2',
          token_pattern=r"(?u)\b\w+\b", stop_words=stopwords,ngram_range=(1,
          2))),
      ('classifier', ComplementNB())
]
# 搭建 pipeline
pipeline = Pipeline(pipeline_list)

# 训练 pipeline
pipeline.fit(X_train, y_train)

# 对测试集的数据集进行预测
y_pred = pipeline.predict(X_test)

# 在测试集上进行评估
from sklearn import metrics
print("在测试集上的混淆矩阵: ")
print(metrics.confusion_matrix(y_test, y_pred))
```

```
print("在测试集上的分类结果报告：")
print(metrics.classification_report(y_test, y_pred))
print("在测试集上的 f1-score ：")
print(metrics.f1_score(y_test, y_pred))
# 在所有的样本上训练一次，充分利用已有的数据，提高模型的泛化能力
pipeline.fit(X, y)
# 保存训练的模型，请将模型保存在 results 目录下
from sklearn.externals import joblib
pipeline_path = 'results/pipeline.model'
joblib.dump(pipeline, pipeline_path)
```

基本框架都已给出,我们可以修改的就是 pipeline_list 部分,我们可以选择 `CountVectorizer` 或者 `TfidfVectorizer` 的方法将文本向量化,同时还可以调节里面的参数。如这里的代码我们使用了 `TfidfVectorizer`, 设置 `max_df=0.95`, 表明出现频次大于 0.95 的字词将会被我们忽略; `max_features=5000` 表明我们对所有关键词的 term frequency 进行降序排序,只取 5000 个作为关键词集; `norm='l2'` 表明用 l2 的方法进行标准化; `ngram_range=2` 表明我们的词组既可以是单字，也可以是两个字组成的词语。(注：此处涉及到调参数的代码只是在实验中的一个例子，用来说明有哪些参数可以调节以及对应的作用，最后提交的代码可能与此不同，是经过反复调整后的结果)

3.2 预测

在 main.py 中，我们主要做的是加载训练好的模型，即导入 pipeline.model 文件，并用来预测短信，代码如下：

```
# 加载训练好的模型
from sklearn.externals import joblib
pipeline_path = 'results/pipeline.model'
pipeline = joblib.load(pipeline_path)
def predict(message):
    """
    预测短信短信的类别和每个类别的概率
    param: message: 经过jieba分词的短信，如"医生 拿 着 我 的 报告单 说 ： 幸亏 你 来 的 早 啊"
    return: label : 整数类型，短信的类别，0 代表正常，1 代表恶意
            proba: 列表类型，短信属于每个类别的概率，如[0.3, 0.7]，认为短信属于 0 的概率为 0.3，
                属于 1 的概率为 0.7
    """
    label = pipeline.predict([message])[0]
    proba = list(pipeline.predict_proba([message])[0])

    return label, proba
```

3.3 正式开始训练

- 当我们使用 TfidfVectorizer 时
  初始我们设置的参数为 (pipeline_list 内):
  ('cv', TfidfVectorizer(max_df=0.95, max_features=1000,norm='l2',
  token_pattern=r"(?u)\b\w+\b", stop_words=stopwords,
  ngram_range=(1, 2))),('classifier', ComplementNB())
  此时的输出:

```
总共的数据大小 (786610,)
训练集数据大小 (393305,)
测试集数据大小 (393305,)
在测试集上的混淆矩阵:
[[330956 22746]
[ 1246 38357]]
在测试集上的分类结果报告:
                    precision recall f1-score support

              0       1.00    0.94    0.97  353702
              1       0.63    0.97    0.76   39603

accuracy                        0.94  393305
macro avg     0.81    0.95    0.86  393305
weighted avg  0.96    0.94    0.94  393305

在测试集上的 f1-score :
0.7617619605584572
```

  Mo 平台上的分类正确比例也只有 6/10
  我认为是 max_features 较小,导致很多有用的字词都被去掉了,于是我将 max_features 调为 10000, 提高得分至 0.8263443826662705, 但对于 Mo 平台系统测试没有明显的帮助。此时我尝试去掉 max_features, 当没有这个参数的时候默认所有字词都要考虑。但这时会发现因为我们的 ngram_range=(1, 2), 单字与双字词的组合过多，我们的程序需要运行漫长的时间。
  在后面的测试中，无论是改变 norm 方式，max_df 还是改变使用的朴素贝叶斯模型，都没有取得好的效果。

- 当我们使用 CountVectorizer 时
  初始我们设置的参数为 (pipeline_list 内):
  ('cv', CountVectorizer(max_df=0.95, max_features=1000,
  token_pattern=r"(?u)\b\w+\b", stop_words=stopwords,

ngram_range=(1, 2))),('classifier', ComplementNB())

此时的输出：

```
总共的数据大小 (786610,)
训练集数据大小 (393305,)
测试集数据大小 (393305,)
在测试集上的混淆矩阵：
[[345221 8481]
[ 2328 37275]]
在测试集上的分类结果报告：
                    precision recall f1-score support

              0       0.99    0.98   0.98  353702
              1       0.81    0.94   0.87   39603

accuracy                       0.97  393305
macro avg     0.90    0.96   0.93  393305
weighted avg  0.98    0.97   0.97  393305

在测试集上的 f1-score ：
0.8733701191438512
```

Mo 平台上的分类正确比例仍然只有 6/10

我用之前的方式调整参数，有了不同的结果：当去掉 max_df 和 max_feature 参数，并将 ngram_range 设为 (1,1) 后，我们在测试集上有了 0.918721999528413 的分数，同时在 Mo 平台的分类正确比例达到了 7/10！

此外，我们还可以设置朴素贝叶斯算法中的 α 值，于是我将其设为 1.0(默认也是 1.0), 同时加入了 norm=True(数据二归一),
即 ComplementNB(alpha=1.0,norm=True) 后发现测试集上的成绩变化不大，但却可以在 Mo 平台上通过 8/10！

值得一提的是，在这个数据集上，ComplementNB 和 MultinomialNB 都有较好的效果,但对于伯努利模型 BernoulliNB, 它虽然可以在我们构建模型时达到 0.9575056836635271 分，但在 Mo 平台上的测试却只能通过 6 个点。

- 此外，在搭建测试集时有两个参数 random_state 和 test_size, 我们发现修改它对于最后系统测试没有影响，只会小幅度影响我们在测试集上的得分，因此这里就不再过多赘述，以上所有尝试均是在相同的 random_state test_size 下进行。

3.4 logistics 回归与决策树实现

在之前代码的基础上，我们只需要从 sklearn 中引入这两种分类算法，更换之前的朴素贝叶斯算法即可。其他部分不需要有变动。

- logistics 回归

```
总共的数据大小 (786610,)
训练集数据大小 (550627,)
测试集数据大小 (235983,)
在测试集上的混淆矩阵：
[[212173 204]
[  769 22837]]
在测试集上的分类结果报告：
                    precision recall f1-score support

              0       1.00    1.00   1.00  212377
              1       0.99    0.97   0.98   23606

accuracy                       1.00  235983
macro avg     0.99    0.98   0.99  235983
weighted avg  1.00    1.00   1.00  235983

在测试集上的 f1-score ：
0.9791412095097219
```

- 决策树实现

```
            总共的数据大小 (786610,)
            训练集数据大小 (550627,)
            测试集数据大小 (235983,)
            在测试集上的混淆矩阵：
            [[210380 1997]
            [ 1655 21951]]
            在测试集上的分类结果报告：
                              precision recall f1-score support

                        0     0.99    0.99    0.99  212377
                        1     0.92    0.93    0.92   23606

                  accuracy                    0.98  235983
            macro avg   0.95    0.96    0.96  235983
            weighted avg  0.98    0.98    0.98  235983
```

```
16
17              在测试集上的 f1-score ：
18              0.9232030954283552
```

经过参数调整后，我们发现这两种模型都只能通过 6/10, 因此在最后的算法里没有使用这两种模型。

四、实验结果

最后我们使用的参数是：

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,
       test_size=0.3) # 构建测试集部分
2
3  ('cv', CountVectorizer(token_pattern=r"(?u)\b\w+\b", stop_words=stopwords,
       ngram_range=(1, 1))),
4        ('classifier', ComplementNB(alpha=1.0, norm=True))# 搭建 pipeline 部分
```

效果如下：

```
总共的数据大小 (786610,)
训练集数据大小 (550627,)
测试集数据大小 (235983,)
在测试集上的混淆矩阵:
[[207660   4717]
 [   549  23057]]
在测试集上的分类结果报告:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    212377
           1       0.83      0.98      0.90     23606

    accuracy                           0.98    235983
   macro avg       0.91      0.98      0.94    235983
weighted avg       0.98      0.98      0.98    235983

在测试集上的 f1-score :
0.8975087582717011
```

(a) 训练模型的分数

(b) Mo 平台系统测试得分

五、总结

首先是朴素贝叶斯算法，实际上 `sklearn.naive_bayes` 模块中有五种贝叶斯算法：伯努利朴素贝叶斯 (BernoulliNB)，类朴素贝叶斯 (CategoricalNB)，高斯朴素贝叶斯 (GaussianNB)、多项式朴素贝叶斯 (MultinomialNB)、补充朴素贝叶斯 (ComplementNB)。通过本次实验亲身经历和网上的资料，我们可以知道 MultinomialNB 假设特征复合多项式分布，是一种非常典型的文本分类模型，模型内部带有平滑参数 α; CNB 的参数估计比 MNB 的参数估计更稳定，比较适用于不平衡的数据集，在文本分类上的结果通常比 MultinomialNB 模型好; BernoulliNB 可能在一些数据集上表现得更好，特别是那些更短的文档。

回看整个实验过程,可以看到 `CountVectorizer` 的效果整体优于 `TfidfVectorizer`. 经过上网搜索，我认为是对于涉及文本的向量空间模型，`TfidfVectorizer` 不适合使用朴素贝叶斯模型，所以我们最后也使用了 `CountVectorizer` 的方式。

虽然做了很多尝试，我们仍然有两个测试点没有通过。在这里我提出我的一些想法：单个模型终究是有限的，如上文所说，各种朴素贝叶斯算法各有优劣，如果我们能结合两种及以上算法，不同的情况用不同的算法，或许能有更高的结果。此外对于 `TfidfVectorizer`, 可以尝试除了朴素贝叶斯以外的模型。

对于逻辑回归和决策树的模型，这里限于自身知识的浅薄没有做过多尝试和优化。理论上我们可以利用原理部分分析的二者的利弊进行融合，结合各自的长处。主要思路是利用决策树对局部数据结构优越的把握能力增加逻辑回归的效力。在具体做法上有几种，一种是从决策树分析中找出数据局部结构，作为在逻辑回归中构建依变量的依据。另一种是在需要对预测因子进行离散化处理时，利用决策树分析决定最佳切分点。还有一种是把决策树分类的最终结果作为预测变量，和其他协变量一起代入回归模型，又称为“嫁接式模型”。从理论上讲，嫁接模型综合了决策树和逻辑回归的优点。最终节点包含了数据中重要的局部结构，而协变量可以拾补被决策树遗漏的数据整体结构。

除此之外，实验中我们使用的是四川大学机器智能实验室停用词库。我们可以尝试更换更好的停用词库，并放在 results 目录下。

总的来说，本次实验主要是通过调整参数来不断提高准确率，几乎不需要自己进行代码的实现。在调参的过程中，每调一次参数都要训练一次，在验证集上要运行一遍整个网络，非常耗时，很难在有限时间内得到非常好的模型，单纯靠“尝试”调参是很难得到一个很好的模型的，需要一些指导性的方针来指引我们朝正确的方向走下去。因此深度学习目前应该发展借助实验和理论发展出一些好的指导原则。

最后，期待下次实验带给我新的惊喜。