程序报告

一、问题重述

（简单描述对问题的理解，从问题中抓住主干，必填）

==========

使用逻辑回归或者决策树模型解决垃圾短信识别问题，并对比两种模型。

二、设计思想

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

==========

本次实验我考虑的因素主要有：

1. 训练形成的向量矩阵是稀疏矩阵。在训练集含有大量短信（词语）的情况下，一条短信可能只含有其中很少的一部分词语。这意味着分词之后形成的矩阵是稀疏矩阵。
2. 垃圾短信往往只通过少数的几个词语或者一句话便可判断出来。这意味着如果用线性模型来进行垃圾短信识别，会有许多系数都是 0。
3. 向量的维度很大，因为整个数据集包含的词语会非常多。出现频率过高和出现频率过低的词语都不足以成为判别垃圾短信的根据，因此可以通过舍去分词形成的词汇表中的这部分词语来适当降低向量的维度。
4. 数据集中不同标签的权重并不一致。在通过 *metrics* 输出训练结果时，可以看到标签 0 的数量远比标签 1 的多，训练时要注意设置 *class_weight* 来减少这种影响。
5. 根据实验指导中的提示对模型进行优化。

三种模型的主要特征如下：

1. 朴素贝叶斯中的朴素是指特征条件独立假设,其以贝叶斯原理为基础，将数据归为后验概率最大的标签。
2. 逻辑回归在线性回归的基础上套上了一层逻辑函数，在工业界得到广泛使用。
3. 决策树模型利用训练数据，根据损失函数最小化的原则建立决策树。

三、代码内容

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

==========

```
import warnings
warnings.filterwarnings('ignore')
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'results/scu_stopwords.txt'
# stopwords_path = r'results/sw.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)

# ----------------- 导入相关的库 -----------------
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import metrics
from sklearn.preprocessing import StandardScaler, MaxAbsScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.externals import joblib
from sklearn.naive_bayes import MultinomialNB

pipeline_path = 'results/pipeline_nb.model'
cv                    =     CountVectorizer(token_pattern=r"(?u)\b\w+\b",
stop_words=stopwords, min_df=1e-5, max_df=0.6, ngram_range=(1, 5))
tfidf           =           TfidfVectorizer(token_pattern=r"(?u)\b\w+\b",
stop_words=stopwords, min_df=1e-5, max_df=0.6, ngram_range=(1, 5))
lr = LogisticRegression(penalty='l1', solver='liblinear',   max_iter =
10000, class_weight='balanced', C=0.01, tol=1e-5)
dt     =     DecisionTreeClassifier(max_features=0.5,     max_depth=50,
class_weight='balanced')
nb = MultinomialNB()
sc = StandardScaler(with_mean = False)
mas = MaxAbsScaler()

# pipline_list 用于传给 Pipline 作为参数
```

```
pipeline_list = [
    #    --------------------------    需 要 完 成 的 代 码
------------------------------

    #    ==========================    以 下 代 码 仅 供 参 考
============================
    # ('tfidf', tfidf),
    ('cv', cv),
    # ('sc', sc),
    # ('mas', mas),
    # ('classifier', lr)
    # ('classifier', dt)
    ('classifier', nb)
    #
=========================================================================
=

    #
-------------------------------------------------------------------------
-
]

# ----------------- 读取数据集 -----------------
import pandas as pd
import numpy as np
import time
from random import randint
from sklearn.model_selection import train_test_split
# 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')
# 构建训练集和测试集
X = np.array(sms.msg_new)
y = np.array(sms.label)

# ----------------- 训练 -----------------
# 搭建 pipeline
pipeline = Pipeline(pipeline_list)
epoch = 20
X_train, X_test, y_train, y_test = None, None, None, None
start = time.time()
# 训练 pipeline
# for i in range(0, epoch):
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42 ,
test_size=0.2)
pipeline.fit(X_train, y_train)
end = time.time()
...
# 保存训练的模型，请将模型保存在 results 目录下
joblib.dump(pipeline, pipeline_path)
```

四、实验结果

（实验结果，必填）

==========

因为不清楚函数的参数设置对其性能的影响有多大，因此逻辑回归、决策树与朴素贝叶斯的对比仅在不设置任何参数、其他条件完全相同的情况下进行。逻辑回归的结果如下：

```
训练时间： 40.28610372543335 s
在测试集上的分类结果报告：
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    141764
           1       0.99      0.97      0.98     15558

    accuracy                           1.00    157322
   macro avg       0.99      0.98      0.99    157322
weighted avg       1.00      1.00      1.00    157322

在测试集上的 f1-score ：
0.9790127875573488
# --------------------------------- #
泛化时间： 54.85858201980591 s
在测试集上的分类结果报告：
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    141764
           1       1.00      0.99      0.99     15558

    accuracy                           1.00    157322
   macro avg       1.00      0.99      1.00    157322
weighted avg       1.00      1.00      1.00    157322

在测试集上的 f1-score ：
0.9910463199405243
```

决策树的结果如下：

```
训练时间： 777.2311425209045 s
在测试集上的分类结果报告：
              precision    recall  f1-score   support
```

```
          0       0.99      0.99      0.99    141764
          1       0.91      0.93      0.92     15558

    accuracy                           0.98    157322
   macro avg       0.95      0.96      0.96    157322
weighted avg       0.98      0.98      0.98    157322

在测试集上的 f1-score ：
0.9196527932339195
# ---------------------------------- #
泛化时间： 1240.0558638572693 s
在测试集上的分类结果报告：
              precision    recall  f1-score   support

          0       1.00      1.00      1.00    141764
          1       1.00      1.00      1.00     15558

    accuracy                           1.00    157322
   macro avg       1.00      1.00      1.00    157322
weighted avg       1.00      1.00      1.00    157322

在测试集上的 f1-score ：
1.0
```

朴素贝叶斯的结果如下：

```
训练时间： 11.949102401733398 s
在测试集上的分类结果报告：
              precision    recall  f1-score   support

          0       1.00      0.98      0.99     70811
          1       0.87      0.98      0.92      7850

    accuracy                           0.98     78661
   macro avg       0.93      0.98      0.96     78661
weighted avg       0.99      0.98      0.98     78661

在测试集上的 f1-score ：
0.9214715878599964
# ---------------------------------- #
泛化时间： 13.23452877998352 s
在测试集上的分类结果报告：
              precision    recall  f1-score   support

          0       1.00      0.98      0.99     70811
```

```
           1       0.87      0.99      0.93      7850

    accuracy                           0.98     78661
   macro avg       0.93      0.99      0.96     78661
weighted avg       0.99      0.98      0.98     78661

在测试集上的 f1-score ：
0.9259767372502237
```

本次实验我主要对逻辑回归以及决策树模型进行了探索，结合教材、网络上的资料以及实验对这两种模型进行一些对比：

1. 两种模型都可以用于分类，而决策树还可以用于回归。
2. 决策树的决策过程是直观且易于理解的，逻辑回归也是可解释性较强的模型。
3. 逻辑回归容易收到极端值影响，而决策树则不然。
4. 速度上逻辑回归相对更快，并且同时可以取得一个不错的 f1-score。
5. 逻辑回归的泛化能力更强，在对没有拟合过的测试集进行预测时，逻辑回归的表现更好。
6. 决策树更看重局部结构，较容易过拟合，而逻辑回归更看重整体。这一点可以从决策树对训练集进行拟合后再预测的 f1-score 为 1 看出。
7. 显然地，逻辑回归适合分析线性关系或者可以用线性关系近似的非线性关系，而决策树更适合用来处理非线性模型。但是识别垃圾短信似乎并不是一个关系非常明确的问题，两个模型都可以在该问题中取得不错的表现。
8. 垃圾短信分类这个问题的维度较大，这对逻辑回归来说是一个不利（不知道该说法是否准确），因为这意味着会有一部分对分类没有意义的数据也被考虑进去了。维度大对决策树来说亦然，这意味着算法最后需要产生一个高度比较大的决策树，影响性能。

综合来看，我个人认为总体上逻辑回归是一个比决策树功能更强大的模型，但是哪种模型的表现会更好与问题的特征息息相关。在后续的模型优化上我主要在优化逻辑回归模型：

1. 实验过程中我并没有感受到 *CountVectorizer* 和 *TfidfVectorizer* 两个文本向量化模型在该问题上展现出了明显区别。考虑到维度大和稀疏等问题，我主要使用 *CountVectorizer* 进行实验。参数设置上主要是设置 *min_df* 来滤去出现频率过低的词语，设置 *max_df* 来滤去出现频率过高的词语，设置 *ngram_range* 来组合词语形成新的词汇表项。实验发现后两个参数可以显著影响拟合的用时以及模型的 f1-score。
2. 将默认的中文停用词表换成了一个具有一千多个项的停用词表。
3. 加入数据归一化处理。我尝试了 *StandardScaler* 和 *MaxAbsScaler*，二者没有表现出太大区别。但是总的来说，加入归一化处理器还是可以提升模型的表现，提高模型的。值得一提的是，*StandardScaler* 必须设置参数 *with_mean* 为 *False*，这提示了文本向量化之后形成的矩阵是稀疏矩阵。
4. 逻辑回归模型的参数设置有：
   (1) 设置 penalty='l1', solver='liblinear'，这是出于倾向于让模型产生稀疏解的考虑。
   (2) 设置 max_iter = 10000，最大迭代次数太低则模型无法拟合。
   (3) 设置 class_weight='balanced'，为了降低标签 0 权重显著大于标签 1 的权重带来的影响。
5. 决策树模型的参数设置有：
   (1) 设置 max_features=0.5, max_depth=50，为了防止使用太多无关的特征来形成决策树，同时也是降低拟合的用时。

(2) 设置 class_weight='balanced'，为了降低标签 0 权重显著大于标签 1 的权重带来的影响。

(3) 我尝试过设置与剪枝有关的参数，但是效果不太好，因此没有再深入探索。

优化后得到的部分表现如下：

```
2022-03-29T06:02:51.160656013Z SYSTEM: Preparing env...
2022-03-29T06:02:51.649860903Z SYSTEM: Running...
2022-03-29T06:04:58.959875823Z 训练时间： 123.29394865036011 s
2022-03-29T06:05:01.007697724Z 在测试集上的分类结果报告：
2022-03-29T06:05:01.23941114Z              precision    recall  f1-score
support
2022-03-29T06:05:01.239455746Z
2022-03-29T06:05:01.239466773Z           0       0.99      0.98      0.98
141764
2022-03-29T06:05:01.23947611Z            1       0.82      0.88      0.85
15558
2022-03-29T06:05:01.239485802Z
2022-03-29T06:05:01.239494274Z     accuracy                           0.97
157322
2022-03-29T06:05:01.239503074Z    macro avg       0.91      0.93      0.92
157322
2022-03-29T06:05:01.23951158Z weighted avg       0.97      0.97      0.97
157322
2022-03-29T06:05:01.23953537Z
2022-03-29T06:05:01.239544369Z 在测试集上的 f1-score ：
2022-03-29T06:05:01.291630554Z 0.8495547667974345
2022-03-29T06:05:01.291658367Z # ---------------------------------- #
2022-03-29T06:07:39.893094902Z 泛化时间： 158.6012396812439 s
2022-03-29T06:07:42.009396593Z 在测试集上的分类结果报告：
2022-03-29T06:07:42.234985204Z                        precision      recall
f1-score   support
2022-03-29T06:07:42.235028721Z
2022-03-29T06:07:42.235038556Z           0       1.00      1.00      1.00
141764
2022-03-29T06:07:42.235047984Z           1       1.00      1.00      1.00
15558
2022-03-29T06:07:42.235057021Z
2022-03-29T06:07:42.235065756Z     accuracy                           1.00
157322
2022-03-29T06:07:42.235075278Z    macro avg       1.00      1.00      1.00
157322
2022-03-29T06:07:42.235083899Z weighted avg       1.00      1.00      1.00
157322
2022-03-29T06:07:42.235092331Z
2022-03-29T06:07:42.235100451Z 在测试集上的 f1-score ：
```

```
2022-03-29T06:07:42.285876612Z 0.9993254312421701
2022-03-29T06:07:51.170610456Z SYSTEM: Finishing...
2022-03-29T06:07:51.722709682Z SYSTEM: Done!
```

```
2022-03-29T07:38:22.623038189Z SYSTEM: Preparing env...
2022-03-29T07:38:23.142362058Z SYSTEM: Running...
2022-03-29T07:39:37.342630659Z 训练时间： 70.40289258956909 s
2022-03-29T07:39:48.029289287Z 在测试集上的分类结果报告：
2022-03-29T07:39:48.514816007Z                     precision      recall
f1-score   support
2022-03-29T07:39:48.514855072Z
2022-03-29T07:39:48.514862752Z           0       0.99      0.99      0.99
283309
2022-03-29T07:39:48.514868872Z           1       0.94      0.95      0.94
31335
2022-03-29T07:39:48.514875337Z
2022-03-29T07:39:48.514880525Z    accuracy                           0.99
314644
2022-03-29T07:39:48.514886431Z   macro avg       0.97      0.97      0.97
314644
2022-03-29T07:39:48.514892485Z weighted avg       0.99      0.99      0.99
314644
2022-03-29T07:39:48.514898813Z
2022-03-29T07:39:48.514904629Z 在测试集上的 f1-score ：
2022-03-29T07:39:48.624715952Z 0.943592029031642
2022-03-29T07:39:48.624752176Z # ---------------------------------- #
2022-03-29T07:41:48.371639145Z 泛化时间： 119.74689269065857 s
2022-03-29T07:41:59.285767299Z 在测试集上的分类结果报告：
2022-03-29T07:41:59.772780922Z                     precision      recall
f1-score   support
2022-03-29T07:41:59.772825361Z
2022-03-29T07:41:59.77284044Z           0       1.00      1.00      1.00
283309
2022-03-29T07:41:59.77284984Z           1       1.00      1.00      1.00
31335
2022-03-29T07:41:59.772857835Z
2022-03-29T07:41:59.772865292Z    accuracy                           1.00
314644
2022-03-29T07:41:59.772872807Z   macro avg       1.00      1.00      1.00
314644
2022-03-29T07:41:59.772880297Z weighted avg       1.00      1.00      1.00
314644
2022-03-29T07:41:59.772888075Z
2022-03-29T07:41:59.772895383Z 在测试集上的 f1-score ：
```

```
2022-03-29T07:41:59.882167934Z 0.9988041138483618
2022-03-29T07:43:26.396479279Z SYSTEM: Finishing...
2022-03-29T07:43:26.937132951Z SYSTEM: Done!
```

在提交测试处的正确率是 10 秒内 7/10。如果使用贝叶斯模型，提交测试处的正确率可达 8/10，但 `f1-score` 没那么高。

五、总结

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

==========

1. 对于模型的调参不太熟练，进行实验时还是有些急躁以及盲目。
2. 实验过程中发现 *max_df* 设置为 0.1 到 0.9 似乎都不太影响流水线的整体表现，还没找到一个合理的解释。
3. 相信模型的表现还有提升的空间，可以通过调参以及补充数据集等方法实现。