1、垃圾短信分类

(1) 数据读取：

代码：

```
import os
import re
import jieba
import numpy as np
import pandas as pd
# from scipy.misc import imread
import imageio.v2 as imageio
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix, classification_report
# 读取数据
data = pd.read_csv('../data/message80W.csv', encoding='utf-8',
index_col=0, header=None, nrows=2000)
data.columns = ['类别', '短信']
data.类别.value_counts()
```

(2) 文本预处理：

代码：

```
temp = data.短信
temp.isnull().sum()
# 去重
data_dup = temp.drop_duplicates()
# 脱敏
l1 = data_dup.astype('str').apply(lambda x: len(x)).sum()
data_qumin = data_dup.astype('str').apply(lambda x: re.sub('x', '', x))
l2 = data_qumin.astype('str').apply(lambda x: len(x)).sum()
print('减少了' + str(l1-l2) + '个字符')
# 加载自定义词典
jieba.load_userdict('../data/newdic1.txt')
# 分词
data_cut = data_qumin.astype('str').apply(lambda x: list(jieba.cut(x)))
# 去停用词
stopword = pd.read_csv('../data/stopword.txt', sep='ooo',
encoding='gbk',header=None, engine='python')
stopword = [' '] + list(stopword[0])
l3 = data_cut.astype('str').apply(lambda x: len(x)).sum()
```

```
data_qustop = data_cut.apply(lambda x: [i for i in x if i not in stopword])
l4 = data_qustop.astype('str').apply(lambda x: len(x)).sum()
print('减少了' + str(l3-l4) + '个字符')
data_qustop = data_qustop.loc[[i for i in data_qustop.index if
data_qustop[i] != []]]
```

结果：

```
减少了2221个字符
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\ADan\AppData\Local\Temp\jieba.cache
Loading model cost 0.599 seconds.
Prefix dict has been built successfully.
减少了58897个字符
```

(3) 词频统计：

代码：

```
lab = [data.loc[i, '类别'] for i in data_qustop.index]
lab1 = pd.Series(lab, index=data_qustop.index)

def cipin(data_qustop, num=10):
    temp = [' '.join(x) for x in data_qustop]
    temp1 = ' '.join(temp)
    temp2 = pd.Series(temp1.split()).value_counts()
    return temp2[temp2 > num]

data_gar = data_qustop.loc[lab1 == 1]
data_nor = data_qustop.loc[lab1 == 0]
data_gar1 = cipin(data_gar, num=5)
data_nor1 = cipin(data_nor, num=30)

# 绘制垃圾短信词云图
back_pic = imageio.imread('../data/background.jpg')
wc = WordCloud(font_path='C:/Windows/Fonts/simkai.ttf',  # 字体
               background_color='white',    # 背景颜色
               max_words=2000,   # 最大词数
               mask=back_pic,   # 背景图片
               max_font_size=200,  # 字体大小
               random_state=1234)  # 设置多少种随机的配色方案
gar_wordcloud = wc.fit_words(data_gar1)
plt.figure(figsize=(16, 8))
plt.imshow(gar_wordcloud)
plt.axis('off')
plt.savefig('../tmp/spam.jpg')
```

```
plt.show()

# 绘制非垃圾短信词云图
nor_wordcloud = wc.fit_words(data_nor1)
plt.figure(figsize=(16, 8))
plt.imshow(nor_wordcloud)
plt.axis('off')
plt.savefig('../tmp/non-spam.jpg')
plt.show()

# 代码8-3 数据采样
num = 100
adata = data_gar.sample(num, random_state=123)
bdata = data_nor.sample(num, random_state=123)
data_sample = pd.concat([adata, bdata])
cdata = data_sample.apply(lambda x: ' '.join(x))
lab = pd.DataFrame([1] * num + [0] * num, index=cdata.index)
my_data = pd.concat([cdata, lab], axis=1)
my_data.columns = ['message', 'label']
```

结果：

(4) 分类：

代码：

```
# 代码8-4
# 划分训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(
    my_data.message, my_data.label, test_size=0.2, random_state=123)  #
构建词频向量矩阵
# 训练集
cv = CountVectorizer()  # 将文本中的词语转化为词频矩阵
train_cv = cv.fit_transform(x_train)  # 拟合数据，再将数据转化为标准化格式
train_cv.toarray()
train_cv.shape  # 查看数据大小
cv.vocabulary_  # 查看词库内容
# 测试集
cv1 = CountVectorizer(vocabulary=cv.vocabulary_)
test_cv = cv1.fit_transform(x_test)
test_cv.shape
# 朴素贝叶斯
nb = MultinomialNB()  # 朴素贝叶斯分类器
nb.fit(train_cv, y_train)  # 训练分类器
pre = nb.predict(test_cv)  # 预测
```

(5) 模型评价：

代码：

```
cm = confusion_matrix(y_test, pre)
cr = classification_report(y_test, pre)
print(cm)
print(cr)
```

结果：

```
[[ 5 19]
 [ 1 15]]
              precision    recall  f1-score   support

           0       0.83      0.21      0.33        24
           1       0.44      0.94      0.60        16

    accuracy                           0.50        40
   macro avg       0.64      0.57      0.47        40
weighted avg       0.68      0.50      0.44        40
```

2、新闻文本聚类

(1) 数据读取：

代码：

```
# 代码8-9
import re
import os
import json
import jieba
import pandas as pd
from sklearn.cluster import KMeans
import joblib
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

# 代码8-6
# 数据读取
files = os.listdir('../data/json/')  # 读取文件列表
train_data = pd.DataFrame()
test_data = pd.DataFrame()
for file in files:
    with open('../data/json/' + file, 'r', encoding='utf-8') as load_f:
        content = []
        while True:
            load_f1 = load_f.readline()
            if load_f1:
                load_dict = json.loads(load_f1)
                content.append(re.sub('[\t\r\n]', '',
load_dict['contentClean']))
            else:
                break
```

```
        contents = pd.DataFrame(content)
        contents[1] = file[:len(file) - 5]
    # 划分训练集与测试集
    train_data = train_data._append(contents[:400])
    test_data = test_data._append(contents[400:])
```

(2) 文本预处理：

代码：

```
def seg_word(data):
    corpus = []  # 语料库
    stop = pd.read_csv('../data/stopwords.txt', sep='bucunzai', encoding
='utf-8', header=None, engine="python")
    stopwords = [' '] + list(stop[0])  # 加上空格符号
    for i in range(len(data)):
        string = data.iloc[i, 0].strip()
        seg_list = jieba.cut(string, cut_all=False)  # 结巴分词
        corpu = []
        # 去除停用词
        for word in seg_list:
            if word not in stopwords:
                corpu.append(word)
        corpus.append(' '.join(corpu))
    return corpus
train_corpus = seg_word(train_data)  # 训练语料
test_corpus = seg_word(test_data)  # 测试语料
```

(3) 特征提取：

代码：

```
# 代码8-8
# 将文本中的词语转换为词频矩阵，矩阵元素a[i][j]表示j 词在i 类文本下的词频
vectorizer = CountVectorizer()
# 统计每个词语的tf-idf 权值
transformer = TfidfTransformer()
# 第一个fit_transform 是计算tf-idf，第二个fit_transform 是将文本转为词频矩阵
train_tfidf =
transformer.fit_transform(vectorizer.fit_transform(train_corpus))
test_tfidf =
transformer.fit_transform(vectorizer.fit_transform(test_corpus))
# 将tf-idf 矩阵抽取出来，元素w[i][j]表示j 词在i 类文本中的tf-idf 权重
train_weight = train_tfidf.toarray()
test_weight = test_tfidf.toarray()
```

(4) 聚类：

代码：

```
clf = KMeans(n_clusters=4, random_state=4)  # 选择4个中心点
# clf.fit(X)可以将数据输入到分类器里
clf.fit(train_weight)
# 4个中心点
print('4个中心点为:' + str(clf.cluster_centers_))
# 保存模型
joblib.dump(clf, 'km.pkl')
train_res = pd.Series(clf.labels_).value_counts()
s = 0
for i in range(len(train_res)):
    s += abs(train_res[i] - 400)
acc_train = (len(train_res) * 400 - s) / (len(train_res) * 400)
print('\n训练集准确率为：' + str(acc_train))
print('\n每个样本所属的簇为')
for i in range(len(clf.labels_)):
    print(i + 1, ' ', clf.labels_[i])
```

结果：

```
4个中心点为:[[ 2.72785850e-04  5.42101086e-20  1.08420217e-19 ...  7.35980515e-05
   1.35638360e-04  4.94996451e-05]
 [ 1.15059076e-03 -4.06575815e-20  1.08420217e-19 ...  6.09863722e-20
  -1.15196481e-19 -8.47032947e-20]
 [ 5.72074283e-04  4.90657502e-04  1.32978163e-04 ...  6.77626358e-20
  -1.01643954e-19 -6.77626358e-20]
 [ 7.87817769e-04 -5.42101086e-20  6.77626358e-20 ...  6.77626358e-21
  -9.48676901e-20 -7.11507676e-20]]

训练集准确率为：0.60875

每个样本所属的簇为
1   0
2   0
3   0
4   1
5   1
6   0
7   1
8   0
9   1
10   1
11   1
12   1
13   1
14   0
15   1
```

……

```
1591   2
1592   2
1593   2
1594   2
1595   2
1596   2
1597   2
1598   2
1599   0
1600   2
```

(5) 模型评价：

代码：

```
# 代码8-10
test_res = pd.Series(clf.fit_predict(test_weight)).value_counts()
s = 0
for i in range(len(test_res)):
    s += abs(test_res[i] - 100)
acc_test = (len(test_res) * 100 - s) / (len(test_res) * 100)
print('测试集准确率为：' + str(acc_test))
```

结果：

```
测试集准确率为：0.535
```