案例五:使用朴素贝叶斯实现对非结构化文本进行分类

1. 案例目的

• 理解贝叶斯概率公式；

• 掌握贝叶斯分类算法的原理；

• 掌握非结构化文本的处理；

• 掌握使用朴素贝叶斯实现对非结构化文本进行分类。

2. 案例内容

使用朴素贝叶斯实现对新闻类别分类。

贝叶斯定理由英国数学家贝叶斯 ( Thomas Bayes 1702-1761 )发明，用来描述两个条件概率之间的关系，比如 P(A|B) 和 P(B|A)。朴素贝叶斯模型依据的概率公式为：P(A|B)=P(B|A)\*P(A)/P(B)。

3. 案例知识点

• 贝叶斯概率公式；

• 贝叶斯分类算法的原理；

• 非结构化文本的处理；

• 使用朴素贝叶斯实现对非结构化文本进行分类。

4. 案例时长

共1学时，具体安排如下：

• 数据预处理（0.5学时）

• 构建朴素贝叶斯模型并训练（0.3学时）

• 评估模型（0.2学时）

5. 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）sklearn 0.23

**•开发环境与工具：**

1）Spyder 4

6. 案例分析

本案例主要分为以下3部分：

1）处理数据，为建立朴素贝叶斯模型做准备；

2）构建朴素贝叶斯模型并训练；

3）评估模型性能

7. 案例实验过程

使用朴素贝叶斯实现对新闻进行分类，可分为以下3个步骤：

1、数据预处理；

1.1 导入包

1.2 导入数据集

1.3 文本向量化

2、构建模型；

3、评估模型；

7.1开发准备

在桌面上新建一个目录，名字叫classification。本案例中所有的数据集和代码都放在该目录下。

7.1.1打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至classification目录下。

7.1.2 获取数据

本案例需要的数据集会自动从网上下载。

7.1.3 新建代码文件

确保Spyder的工作目录为classification。使用Spyder新建一个Python文件，命名为classify.py。

至此，整个案例的目录结构如图1所示：



图1 案例的目录结构

7.1.4 了解数据

本案例没有单独的数据集文件。数据集是通过调用库从网上下载的。训练集的数据格式如图2所示。

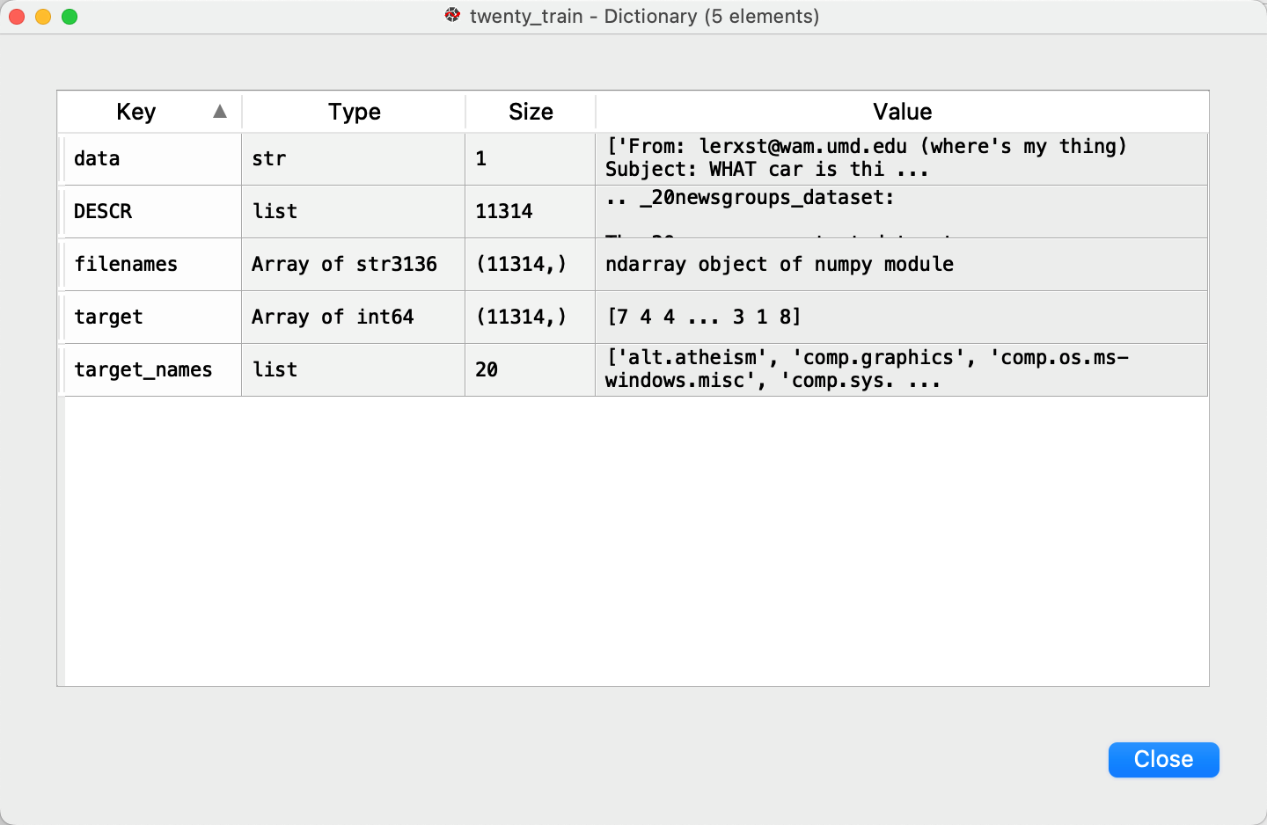


图2 twenty\_train数据格式

测试集的数据格式如图3所示。



图3 twenty\_test数据格式

由上图可见，twenty\_train数据集有11314行，20列. twenty\_test数据集有7532行，20列。

twenty\_train和twenty\_test数据集都包含5个信息，这5个信息分别为：

• data：新闻内容（自变量）

• DESCR：数据集描述

• filenames：新闻内容在哪个文件里面存储

• target：新闻标签（因变量）

• target\_names: 标签名称（因变量列表，共有20个类别）

7.2数据预处理

向classify.py添加代码。

7.2.1 导入包

导入用到的包，代码如下：

【代码7-2-1】classify.py

# -\*- coding: utf-8 -\*-

"""

基于朴素贝叶斯做新闻分类

"""

# 导入包

import numpy as np

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

执行代码块。

7.2.2 导入数据集

本案例没有单独的数据集文件。数据集是通过调用库从网上下载的。

【代码7-2-2】classify.py

# 导入数据集

twenty\_train = fetch\_20newsgroups(subset='train', shuffle=True)

twenty\_test = fetch\_20newsgroups(subset='test', shuffle=True)

print(twenty\_train.data[0]) # 输出第1条新闻内容

print(twenty\_train.target[0]) # 输出第1条新闻标签

执行代码块。

第1条新闻的内容和标签如图4所示。

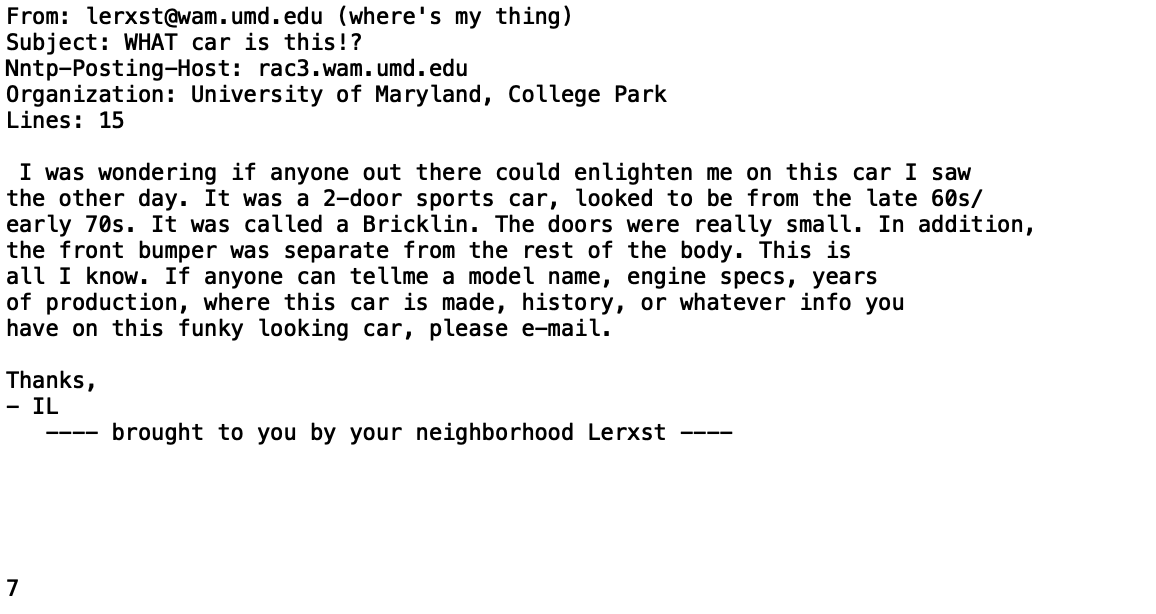


图4 第一条新闻的内容及其标签

7.2.3 文本向量化

由于模型不能直接处理中文，需要首先把中文转换成数值。

首先使用词袋模型进行文本向量化。

【代码7-2-3-1】classify.py

# 词袋模型 文本向量化

count\_vect = CountVectorizer()

X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)

word\_list = count\_vect.get\_feature\_names() # 130107个独一无二的单词

X\_test\_counts = count\_vect.transform(twenty\_test.data)

执行代码块。

执行后发现有130107个独一无二的单词，如图5所示。

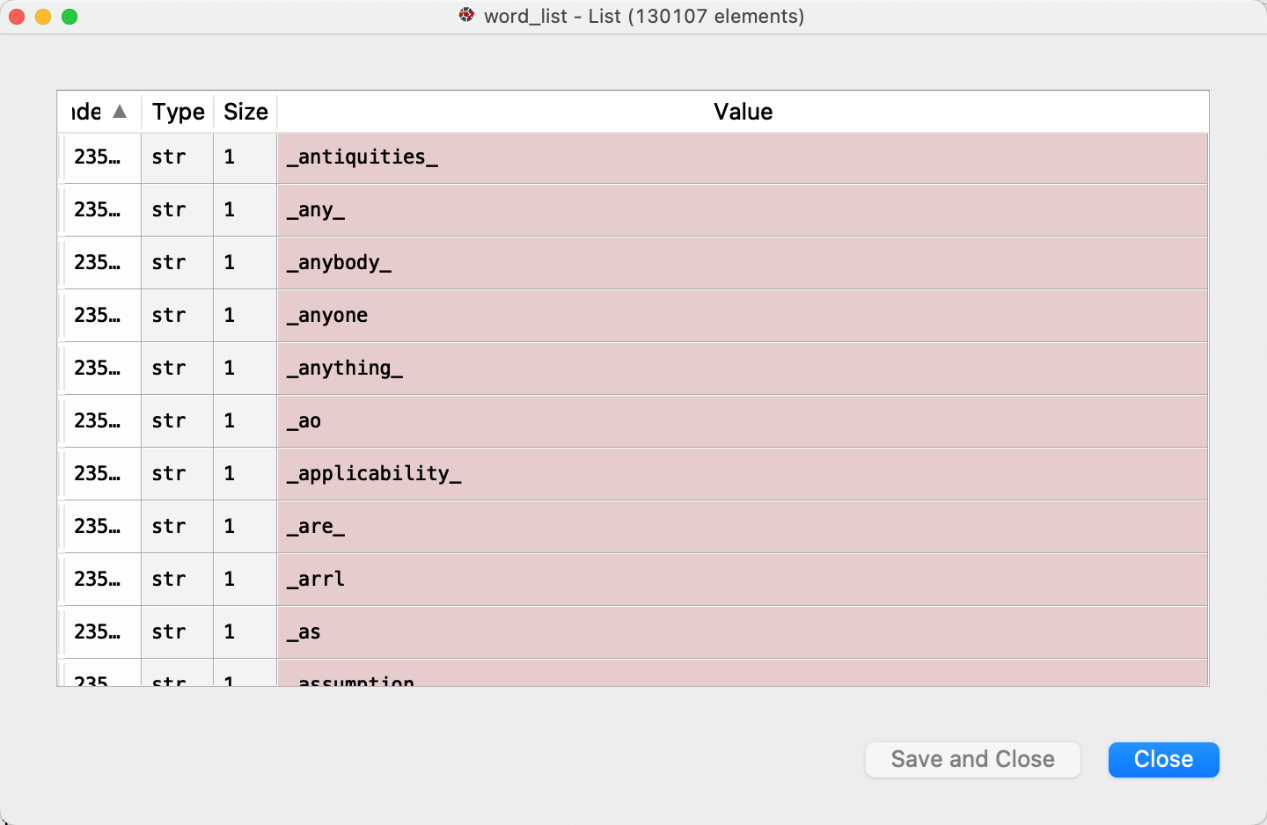


图5 word\_list变量的值

接着，为了使文本向量化更加准确，使用TF-IDF再次进行文本向量化。

【代码7-2-3-2】classify.py

# TF-IDF 文本向量化

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)

X\_test\_tfidf = tfidf\_transformer.fit\_transform(X\_test\_counts)

执行代码块。

7.3构建朴素贝叶斯模型并训练

构建朴素贝叶斯模型并训练。

注：由于这里用于文本分类，所以选择MultinomialNB类，而不是GaussianNB类。

【代码7-3】classify.py

# 构建模型

clf = MultinomialNB(alpha = 0.01, fit\_prior = True)

clf.fit(X\_train\_tfidf, twenty\_train.target)

执行代码块。

7.4评估模型

建立完模型后需要评估一下这个模型的准确性，代码如下：

【代码7-4】classify.py

# 评估模型

predicted = clf.predict(X\_test\_tfidf)

correct\_count = np.sum(predicted == twenty\_test.target)

accuracy = correct\_count/X\_test\_tfidf.shape[0]

print('accuracy is %.2f%s' %(accuracy\*100,'%'))

执行代码块。

执行后结果如图6所示。



图6 模型准确率

模型准确率为83.36%，可以看出模型性能还不错。

8. 案例代码

【案例代码】classify.py

# -\*- coding: utf-8 -\*-

"""

基于朴素贝叶斯做新闻分类

"""

# 导入包

import numpy as np

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

# 导入数据集

twenty\_train = fetch\_20newsgroups(subset='train', shuffle=True)

twenty\_test = fetch\_20newsgroups(subset='test', shuffle=True)

print(twenty\_train.data[0]) # 输出第1条新闻内容

print(twenty\_train.target[0]) # 输出第1条新闻标签

# 词袋模型 文本向量化

count\_vect = CountVectorizer()

X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)

word\_list = count\_vect.get\_feature\_names() # 130107个独一无二的单词

X\_test\_counts = count\_vect.transform(twenty\_test.data)

# TF-IDF 文本向量化

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)

X\_test\_tfidf = tfidf\_transformer.fit\_transform(X\_test\_counts)

# 构建模型并训练

clf = MultinomialNB(alpha = 0.01, fit\_prior = True)

clf.fit(X\_train\_tfidf, twenty\_train.target)

# 评估模型

predicted = clf.predict(X\_test\_tfidf)

correct\_count = np.sum(predicted == twenty\_test.target)

accuracy = correct\_count/X\_test\_tfidf.shape[0]

print('accuracy is %.2f%s' %(accuracy\*100,'%'))