

Weibo Text Classification Prediction Based on Machine Learning Algorithm

by Tao Chen

FILE	25396_TAO_CHEN_WEIBO_TEXT_CLASSIFICATION_PREDICTION_BASED_ON _MACHINE_LEARNING_ALGORITHM_309231_1909706590.DOCX (284.54K)		
TIME SUBMITTED	30-NOV-2020 01:31AM (UTC+0000)	WORD COUNT	4228
SUBMISSION ID	137779450	CHARACTER COUNT	23550

Weibo Text Classification Prediction Based on Machine Learning Algorithm

Data Science and Analytics Tao Chen 2006607

Abstract: With the wide rise of social media, a series of application products such as micro-blog are developing rapidly. Until 2020, according to official statistics of Weibo, its daily active users had already reached more than 160 million, and its daily visits had reached 10 billion levels. How to extract important information from these data and how to provide consumers with what they want to see more quickly and accurately has become a top priority. In this paper, based on microblog news data, the classification of Chinese short text is studied by using two machine learning algorithms, SVM, Naive Bayes Classifier. Word segmentation is the process of recombining consecutive subsequences into word sequences in accordance with certain specifications. In English, spaces are delimiters between words, but Chinese is fairly complicated. Generally speaking, the division of words, sentences and paragraphs is relatively simple, but the division of words in Chinese has no obvious signs, so it is more difficult to segment Chinese words. Python's powerful word segmentation library jieba is used for word segmentation of Chinese text.

Keywords: Machine Learning, Chinese Text Classification, SVM, Naïve Bayes, Microblog(weibo)

1 Introduction

1.1 Weibo

Weibo is a general term for microblogging, but normally understood as Chinese-based mini-blogging services, including social chat sites and platform sharing. Weibo services makes it possible for internet users to set up real-time information sharing communities individually, and upload and update information. Weibo services use a format similar to the American-based Twitter-service, but is used almost exclusively by Chinese language speakers. Weibos are a major source of commentary on a wide range of topics.

1.2 Jieba^[1]

"Jieba" (Chinese for "to stutter") Chinese text segmentation: built to be the best Python Chinese word segmentation module.

Chinese word segmentation is a basic step of Chinese text processing, and it is also a basic module of Chinese human-computer natural language interaction. When performing Chinese natural language processing, word segmentation is usually required first.

1.2.1 Support three types of segmentation mode:

- 1.Accurate Mode attempts to cut the sentence into the most accurate segmentations, which is suitable for text analysis.
- 2.Full Mode gets all the possible words from the sentence. Fast but not accurate.
- 3.Search Engine Mode, based on the Accurate Mode, attempts to cut long words into several short words, which can raise the recall rate. Suitable for search engines.

This program uses the (default) accurate mode.

1.2.2 Usage

- 1.Fully automatic installation: `easy_install jieba` or `pip install jieba`
- 2.Semi-automatic installation: Download <http://pypi.python.org/pypi/jieba/> , run `python setup.py install` after extracting.
- 3.Manual installation: place the jieba directory in the current directory or python site-packages directory.
- 4.`import jieba`.

1.2.3 Algorithm

1. Based on a prefix dictionary structure to achieve efficient word graph scanning. Build a directed acyclic graph (DAG) for all possible word combinations.
2. Use dynamic programming to find the most probable combination based on the word frequency.
3. For unknown words, a HMM-based model is used with the Viterbi algorithm.

1.2.4 Main Functions

1. The **jieba.cut** function accepts three input parameters: the first parameter is the string to be cut; the second parameter is **cut_all**, controlling the cut mode; the third parameter is to control whether to use the Hidden Markov Model.
2. **jieba.cut_for_search** accepts two parameter: the string to be cut; whether to use the Hidden Markov Model. This will cut the sentence into short words suitable for search engines.
3. The input string can be an unicode/str object, or a str/bytes object which is encoded in UTF-8 or GBK. Note that using GBK encoding is not recommended because it may be unexpectedly decoded as UTF-8.
4. **jieba.cut** and **jieba.cut_for_search** returns an generator, from which you can use a for loop to get the segmentation result (in unicode).
5. **jieba.lcut** and **jieba.lcut_for_search** returns a list.
6. **jieba.Tokenizer(dictionary=DEFAULT_DICT)** creates a new customized Tokenizer, which enables you to use different dictionaries at the same time. **jieba.dt** is the default Tokenizer, to which almost all global functions are mapped.

1.2.5 Code example: segmentation

```
#encoding=utf-8
import jieba

seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print("Full Mode: " + "/" .join(seg_list))  # Full Mode
```

```
seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print("Default Mode: " + "/" .join(seg_list)) # Default Mode

seg_list = jieba.cut("他来到了网易杭研大厦")
print(", ".join(seg_list))

seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造") # Searching Engine Mode
print(", ".join(seg_list))
```

Output:

[Full Mode]: 我/ 来到/ 北京/ 清华/ 清华大学/ 华大/ 大学

[Accurate Mode]: 我/ 来到/ 北京/ 清华大学

[Unknown Words Recognize] 他, 来到, 了, 网易, 杭研, 大厦

[Search Engine Mode]: 小明, 硕士, 毕业, 于, 中国, 科学, 学院, 科学院, 中国科学院, 计算, 计算所, 后, 在, 日本, 京都, 大学, 日本京都大学, 深造

1.3 Classifier

1.3.1 Multinomial Naive Bayes.

A generative model estimates the conditional probability of a class given input data. In particular, Naive Bayes assumes the input features are independent each other (conditional independence). Multinomial Naive Bayes is a specialized version of Naïve Bayes mostly used for document and text classification^[2]

1.3.2 SVM (Support Vector Machines)

Support Vector Machines are a particularly powerful and flexible class of supervised algorithms for classification and regression^[3]. SVM is a powerful and versatile machine learning model, capable of performing linear or non-linear classification, regression and even outliers detection. The algorithm makes the classification creating a linear hyperplane of maximum margin that separates two classes. This margin makes there are few possibilities of separating the data from the sample, and thus, there is little chance of misclassifying new instances.

2 Weibo text classification method

Step1 Corpus selection: get the corpus that have been labeled

Step2 The data pre-processing, remove noise, for example, Format conversion, Remove symbol,

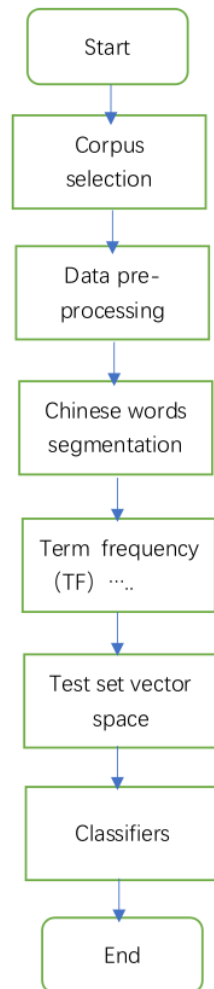
Step3 Chinese words segmentation: Split the article into words

Step4 Construct training set vector space, get terms frequency (TF), generate word vector space for each text, and remove Chinese stop words

Step5 Construct the test set vector space, the method is the same as above

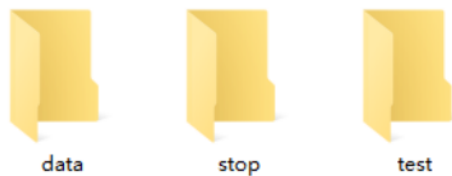
Step6 Use classifiers(naïve bayes or SVM) to predict text classification

The process is shown in the flowchart below



2.1 Corpus Selection

Select corpus, data set, stop word set



The corpus contains four types (women, sports, literature, campus)

-  女性
-  体育
-  文学出版
-  校园

Sufficient data

-  1315.txt
-  1316.txt
-  1317.txt
-  1318.txt
-  1319.txt
-  1320.txt
-  1321.txt
-  1322.txt
-  1323.txt
-  1324.txt
-  1325.txt
-  1326.txt
-  1327.txt
-  1328.txt
-  1329.txt
-  1330.txt
-  1331.txt
-  1332.txt
-  1333.txt
-  1334.txt
-  1335.txt
-  1336.txt
-  1337.txt

2.2 The data pre-processing

Data pre-processing is an important step in the data mining process. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running any analysis^[4]. Often, data preprocessing is the most important phase of a machine learning project. Data preprocessing includes cleaning, Instance selection, normalization, transformation, feature extraction and selection, etc. **The product of data preprocessing is the final training set.**

In this experiment, we removed various symbols, tabs, spaces, carriage returns and other unnecessary interference factors.

Codes show as below:

```
def getStopWord(inputFile):
    stopWordList = readFile(inputFile).splitlines()
    return stopWordList

def segText(inputPath, resultPath):
    fatherLists = os.listdir(inputPath) # 主目录
    for eachDir in fatherLists: # 遍历主目录中各个文件夹
        eachPath = inputPath + eachDir + "/" # 保存主目录中每个文件夹目录，便于遍历二级文件
        each_resultPath = resultPath + eachDir + "/" # 分词结果文件存入的目录
        if not os.path.exists(each_resultPath):
            os.makedirs(each_resultPath)
        childLists = os.listdir(eachPath) # 获取每个文件夹中的各个文件
        for eachFile in childLists: # 遍历每个文件夹中的子文件
            eachPathFile = eachPath + eachFile # 获得每个文件路径
            # print(eachFile)
            content = readFile(eachPathFile) # 调用上面函数读取内容
            # content = str(content)
            result = (str(content)).replace("\r\n", "").strip() # 删除多余空行与空格
            # result = content.replace("\r\n", "").strip()

            cutResult = jieba.cut(result) # 默认方式分词，分词结果用空格隔开
            saveFile(each_resultPath + eachFile, " ".join(cutResult)) # 调用上面函数保存文件
```

2.3 TF-IDF algorithm (Term Frequency–Inverse Document Frequency)

2.3.1 Instruction

TF-IDF score represents the relative importance of a term in the document and the entire corpus. The importance of a word increases proportionally with the number of times it appears in the file, and decreases inversely with its frequency in other files in the corpus. That is to say, a word appears more frequently in a certain document, and

other documents do not appear, indicating that the word is very important for the classification of the document. However, if there are many other documents, it means that the word is not distinguishable, so use IDF to reduce the weight of the word. TF-IDF score is composed by two terms: the first computes the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$

$IDF(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it})$

$TF\text{-}IDF = TF \times IDF^{[5]}$

2.3.2 Stop Words

Stop words refer to words that do not play a decisive role and are not helpful to document classification, but appear many times in the text. In text analysis, in order to save storage space and improve efficiency, these words need to be filtered out before processing the text.

2.3.3 TF-IDF Steps for Usage

Step1 Call the segmented document

Step2 Construct word frequency matrix

Step3 Filter stop words

Step4 Extract the calculated TF-IDF vector

2.3.4 Classifier selection

In this paper, two classifiers are used to predict the category of the text respectively, and the classification results are compared and then the better classifier is used.

3 Result

- We use 80% of the data set as the training set.
- Choose the remaining 20% as the test set
- In order to test the accuracy of the code, we deliberately put some documents in

- the corpus of the wrong category.
- The result shows that the document type can be detected correctly.

Naïve bayes classifier:

```
./split/test_split/体育/1376.txt :实际类别: 体育 -->预测类别: 校园
./split/test_split/体育/1390.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1391.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1400.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1416.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1421.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1441.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1445.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1447.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/女性/959.txt :实际类别: 女性 -->预测类别: 体育
./split/test_split/女性/986.txt :实际类别: 女性 -->预测类别: 文学出版
./split/test_split/文学出版/773.txt :实际类别: 文学出版 -->预测类别: 体育
./split/test_split/文学出版/791.txt :实际类别: 文学出版 -->预测类别: 体育
./split/test_split/校园/250.txt :实际类别: 校园 -->预测类别: 体育
./split/test_split/校园/252.txt :实际类别: 校园 -->预测类别: 女性
./split/test_split/校园/256.txt :实际类别: 校园 -->预测类别: 文学出版
./split/test_split/校园/260.txt :实际类别: 校园 -->预测类别: 文学出版
./split/test_split/校园/263.txt :实际类别: 校园 -->预测类别: 体育
error rate: 9.0 %
```

```
error rate: 9.0 %
```

As shown in the figure above, the error rate is 9.0%

SVM Classifier:

```
./split/test_split/体育/1365.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1376.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1390.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1391.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1395.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1400.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1404.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1441.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1444.txt :实际类别: 体育 -->预测类别: 文学出版
./split/test_split/体育/1445.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1446.txt :实际类别: 体育 -->预测类别: 女性
./split/test_split/体育/1448.txt :实际类别: 体育 -->预测类别: 校园
./split/test_split/女性/959.txt :实际类别: 女性 -->预测类别: 校园
./split/test_split/文学出版/789.txt :实际类别: 文学出版 -->预测类别: 校园
./split/test_split/文学出版/791.txt :实际类别: 文学出版 -->预测类别: 体育
./split/test_split/文学出版/793.txt :实际类别: 文学出版 -->预测类别: 体育
./split/test_split/文学出版/794.txt :实际类别: 文学出版 -->预测类别: 体育
./split/test_split/校园/250.txt :实际类别: 校园 -->预测类别: 体育
./split/test_split/校园/252.txt :实际类别: 校园 -->预测类别: 女性
./split/test_split/校园/262.txt :实际类别: 校园 -->预测类别: 体育
./split/test_split/校园/263.txt :实际类别: 校园 -->预测类别: 体育
./split/test_split/校园/264.txt :实际类别: 校园 -->预测类别: 文学出版
svm model error rate: 11.0 %
```

```
svm model error rate: 11.0 %
```

As shown in the figure above, the error rate is 11.0%

In this section, we present the results of the experiment and discuss its implications. We conducted one experiment to determine the best combination of Machine Learning algorithm and feature extraction technique to classify text. Obviously, naïve bayes classifier had a great performance with the error rate of 9.0%.

4 Conclusion

The innovation of this paper was to apply Chinese text classification to the Weibo text classification, and uses two machine learning algorithms to predict the type of text. Through the comparison of the error rate, the conclusion is that the naive Bayes classifier More suitable for Weibo text classification.

Reference

- 1 jieba,[online].available: <https://github.com/fxsjy/jieba>
- 2 Vergara, D.; Hernández, S.; Jorquera, F. Multinomial Naive Bayes for real-time gender recognition.In Proceedings of the 2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA),Bucaramanga, Colombia, 31 August–2 September 2016; pp. 1–6.
- 3 VanderPlas, J. Python Data Science Handbook; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2016.
- 4 Pyle, D., 1999. Data Preparation for Data Mining. Morgan Kaufmann Publishers, Los Altos, California.
- 5 TF-IDF [online].available:<https://en.wikipedia.org/wiki/Tf-idf>

Appendix I

Naïve bayes classifier

```
import jieba
from numpy import *
import pickle
import os
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets.base import Bunch
from sklearn.naive_bayes import MultinomialNB

def readFile(path):
    with open(path, 'r', errors='ignore') as file:
        content = file.read()
        file.close()
    return content

def saveFile(path, result):
    with open(path, 'w', errors='ignore') as file:
        file.write(result)
        file.close()

def segText(inputPath, resultPath):
    fatherLists = os.listdir(inputPath) # 主目录
    for eachDir in fatherLists: # 遍历主目录中各个文件夹
        eachPath = inputPath + eachDir + "/" # 保存主目录中每个文件夹目录，
        便于遍历二级文件
        each_resultPath = resultPath + eachDir + "/"
```

```

if not os.path.exists(each_resultPath):
    os.makedirs(each_resultPath)
childLists = os.listdir(eachPath)
for eachFile in childLists:
    eachPathFile = eachPath + eachFile # 获得每个文件路径
    # print(eachFile)
    content = readFile(eachPathFile) # 调用上面函数读取内容
    # content = str(content)
    result = (str(content)).replace("\r\n", "").strip()
    # result = content.replace("\r\n", "").strip()

    cutResult = jieba.cut(result)
    saveFile(each_resultPath + eachFile, " ".join(cutResult)) # 调用上面
函数保存文件

```

```

def bunchSave(inputFile, outputFile):
    catelist = os.listdir(inputFile)
    bunch = Bunch(target_name=[], label=[], filenames=[], contents=[])
    bunch.target_name.extend(catelist)
    for eachDir in catelist:
        eachPath = inputFile + eachDir + "/"
        fileList = os.listdir(eachPath)
        for eachFile in fileList:
            fullName = eachPath + eachFile
            bunch.label.append(eachDir)
            bunch.filenames.append(fullName)
            bunch.contents.append(readFile(fullName).strip())
    with open(outputFile, 'wb') as file_obj: # 持久化必须用二进制访问模式打开
        pickle.dump(bunch, file_obj)
    # pickle.dump(obj, file, [,protocol])函数的功能 : 将 obj 对象序列化存入已
已经打开的 file 中。
    # obj : 想要序列化的 obj 对象。
    # file: 文件名称。

```

protocol : 序列化使用的协议。如果该项省略, 则默认为 0。如果为负值或 HIGHEST_PROTOCOL, 则使用最高的协议版本

```
def readBunch(path):
    with open(path, 'rb') as file:
        bunch = pickle.load(file)

    return bunch

def writeBunch(path, bunchFile):
    with open(path, 'wb') as file:
        pickle.dump(bunchFile, file)

def getStopWord(inputFile):
    stopWordList = readFile(inputFile).splitlines()
    return stopWordList

def getTFIDFMat(inputPath, stopWordList, outputPath,
                 tftfidfspace_path, tfidfspace_arr_path, tfidfspace_vocabulary_path):
    # 求得 TF-IDF 向量
    bunch = readBunch(inputPath)
    tfidfspace = Bunch(target_name=bunch.target_name, label=bunch.label,
                      filenames=bunch.filenames, tdm=[],
                      vocabulary={})

    "读取 tfidfspace"
    tfidfspace_out = str(tfidfspace)
    saveFile(tftfidfspace_path, tfidfspace_out)

    vectorizer = TfidfVectorizer(stop_words=stopWordList, sublinear_tf=True,
```

```

max_df=0.5)
    transformer = TfidfTransformer()

    tfidfspace.tdm = vectorizer.fit_transform(bunch.contents)
    tfidfspace_arr = str(vectorizer.fit_transform(bunch.contents))
    saveFile(tfidfspace_arr_path, tfidfspace_arr)
    tfidfspace.vocabulary = vectorizer.vocabulary_ # 获取词汇
    tfidfspace_vocabulary = str(vectorizer.vocabulary_)
    saveFile(tfidfspace_vocabulary_path, tfidfspace_vocabulary)
    '''over'''
    writeBunch(outputPath, tfidfspace)

def getTestSpace(testSetPath, trainSpacePath, stopWordList, testSpacePath,
                 testSpace_path, testSpace_arr_path, trainbunch_vocabulary_path):
    bunch = readBunch(testSetPath)

    testSpace = Bunch(target_name=bunch.target_name, label=bunch.label,
                      filenames=bunch.filenames, tdm=[],
                      vocabulary={})
    '''
    读取 testSpace
    '''
    testSpace_out = str(testSpace)
    saveFile(testSpace_path, testSpace_out)

    trainbunch = readBunch(trainSpacePath)

    vectorizer = TfidfVectorizer(stop_words=stopWordList, sublinear_tf=True,
                                max_df=0.5,
                                vocabulary=trainbunch.vocabulary)

    transformer = TfidfTransformer()
    testSpace.tdm = vectorizer.fit_transform(bunch.contents)
    testSpace.vocabulary = trainbunch.vocabulary

```

```

testSpace_arr = str(testSpace.tdm)
trainbunch_vocabulary = str(trainbunch.vocabulary)
saveFile(testSpace_arr_path, testSpace_arr)
saveFile(trainbunch_vocabulary_path, trainbunch_vocabulary)
# 持久化
writeBunch(testSpacePath, testSpace)

def bayesAlgorithm(trainPath, testPath, tfidfspace_out_arr_path,
                  tfidfspace_out_word_path, testspace_out_arr_path,
                  testspace_out_word_apth):
    trainSet = readBunch(trainPath)
    testSet = readBunch(testPath)
    clf = MultinomialNB(alpha=0.001).fit(trainSet.tdm, trainSet.label)

    # print(shape(trainSet.tdm)) #输出单词矩阵的类型
    # print(shape(testSet.tdm))
    "处理 bat 文件"
    tfidfspace_out_arr = str(trainSet.tdm) # 处理
    tfidfspace_out_word = str(trainSet)
    saveFile(tfidfspace_out_arr_path, tfidfspace_out_arr) # 矩阵形式的
train_set.txt
    saveFile(tfidfspace_out_word_path, tfidfspace_out_word) # 文本形式的
train_set.txt

    testspace_out_arr = str(testSet)
    testspace_out_word = str(testSet.label)
    saveFile(testspace_out_arr_path, testspace_out_arr)
    saveFile(testspace_out_word_apth, testspace_out_word)

    "处理结束"
    predicted = clf.predict(testSet.tdm)
    total = len(predicted)
    rate = 0

```

```

for flabel, fileName, expct_cate in zip(testSet.label, testSet filenames, predicted):
    if flabel != expct_cate:
        rate += 1
        print(fileName, ":实际类别：", flabel, "-->预测类别：", expct_cate)
print("error rate:", float(rate) * 100 / float(total), "%")

#
if __name__ == '__main__':
    datapath = "./data/" #原始数据路径
    stopWord_path = "./stop/stopword.txt"#停用词路径
    test_path = "./test/" #测试集路径

    test_split_dat_path = "./test_set.dat" #测试集分词 bat 文件路径
    testspace_dat_path = "./testspace.dat" #测试集输出空间矩阵 dat 文件
    train_dat_path = "./train_set.dat"
    tfidfspace_dat_path = "./tfidfspace.dat" #tf-idf 词频空间向量的 dat 文件
    """
    以上四个为 dat 文件路径，是为了存储信息做的，不要打开
    """

    test_split_path = './split/test_split/' #测试集分词路径
    split_datapath = "./split/split_data/" # 对原始数据分词之后的数据路径
    """

    以上两个路径是分词之后的文件路径，大家可以生成之后自行打开查阅学习
    """

    tfidfspace_path = "./tfidfspace.txt" # 将 TF-IDF 词向量保存为 txt，方便查看
    tfidfspace_arr_path = "./tfidfspace_arr.txt" # 将 TF-IDF 词频矩阵保存为 txt，
    方便查看

    tfidfspace_vocabulary_path = "./tfidfspace_vocabulary.txt" # 将分词的词汇统
    计信息保存为 txt，方便查看

    testSpace_path = "./testSpace.txt" #测试集分词信息

```

```
testSpace_arr_path = "/testSpace_arr.txt" #测试集词频矩阵信息
trainbunch_vocabulary_path = "/trainbunch_vocabulary.txt" #所有分词词频信息
```

```
tfidfspace_out_arr_path = "/tfidfspace_out_arr.txt" #tfidf 输出矩阵信息
tfidfspace_out_word_path = "/tfidfspace_out_word.txt" #单词形式的 txt
testspace_out_arr_path = "/testspace_out_arr.txt" #测试集输出矩阵信息
testspace_out_word_path = "/testspace_out_word.txt" #测试集单词信息
'''
```

以上 10 个文件是 dat 文件转化为 txt 文件，大家可以查询信息，这是 NLP（自然语言处理）非常珍贵的资源

```
'''

#输入训练集
segText(datapath,#读入数据
        split_datapath)#输出分词结果
bunchSave(split_datapath,#读入分词结果
          train_dat_path) # 输出分词向量
stopWordList = getStopWord(stopWord_path) # 获取停用词表
getTFIDFMat(train_dat_path, #读入分词的词向量
            stopWordList, #获取停用词表
            tfidfspace_dat_path, #tf-idf 词频空间向量的 dat 文件
            tfidfspace_path, #输出词频信息 txt 文件
            tfidfspace_arr_path, #输出词频矩阵 txt 文件
            tfidfspace_vocabulary_path) #输出单词 txt 文件
```

```
#输入测试集
segText(test_path,
        test_split_path) # 对测试集读入文件，输出分词结果
bunchSave(test_split_path,
          test_split_dat_path) #
getTestSpace(test_split_dat_path,
            tfidfspace_dat_path,
            stopWordList,
            testspace_dat_path,
```

```
testSpace_path,  
testSpace_arr_path,  
trainbunch_vocabulary_path)# 输入分词文件 ,停用词 ,词向量 ,  
输出特征空间(txt,dat 文件都有)  
bayesAlgorithm(tfidfspace_dat_path,  
testspace_dat_path,  
tfidfspace_out_arr_path,  
tfidfspace_out_word_path,  
testspace_out_arr_path,  
testspace_out_word_apth)
```

Appendix II

SVM classifier

```
from sklearn.multiclass import OneVsRestClassifier  
import sklearn.svm as svm  
from sklearn.linear_model import LogisticRegression  
import jieba  
from numpy import *  
import pickle # 持久化
```

```

import os
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets.base import Bunch
from sklearn.naive_bayes import MultinomialNB

def readFile(path):
    with open(path, 'r', errors='ignore') as file:
        content = file.read()
        file.close()
    return content

def saveFile(path, result):
    with open(path, 'w', errors='ignore') as file:
        file.write(result)
        file.close()

def segText(inputPath, resultPath):
    fatherLists = os.listdir(inputPath) # 主目录
    for eachDir in fatherLists: # 遍历主目录中各个文件夹
        eachPath = inputPath + eachDir + "/" # 保存主目录中每个文件夹目录，便于遍历二级文件
        each_resultPath = resultPath + eachDir + "/"
        if not os.path.exists(each_resultPath):
            os.makedirs(each_resultPath)
        childLists = os.listdir(eachPath)
        for eachFile in childLists:
            eachPathFile = eachPath + eachFile # 获得每个文件路径
            # print(eachFile)
            content = readFile(eachPathFile) # 调用上面函数读取

```

内容

```
# content = str(content)
result = (str(content)).replace("\r\n", "").strip()
# result = content.replace("\r\n", "").strip()

cutResult = jieba.cut(result)
saveFile(each_resultPath + eachFile, " ".join(cutResult))
# 调用上面函数保存文件
```

```
def bunchSave(inputFile, outputFile):
    catelist = os.listdir(inputFile)
    bunch = Bunch(target_name=[], label=[], filenames=[], contents=[])
    bunch.target_name.extend(catelist)
    for eachDir in catelist:
        eachPath = inputFile + eachDir + "/"
        fileList = os.listdir(eachPath)
        for eachFile in fileList:
            fullName = eachPath + eachFile # 二级目录子文件全路
            bunch.label.append(eachDir)
            bunch.filenames.append(fullName)
            bunch.contents.append(readFile(fullName).strip())
```

径

```
with open(outputFile, 'wb') as file_obj: # 持久化必须用二进制访问模式打开
    pickle.dump(bunch, file_obj)
    # pickle.dump(obj, file, [,protocol])函数的功能：将 obj 对象序列化存入已经打开的 file 中。
    # obj：想要序列化的 obj 对象。
    # file:文件名称。
```

```

def readBunch(path):
    with open(path, 'rb') as file:
        bunch = pickle.load(file)
        # pickle.load(file)
        # 函数的功能：将 file 中的对象序列化读出。
    return bunch

def writeBunch(path, bunchFile):
    with open(path, 'wb') as file:
        pickle.dump(bunchFile, file)

def getStopWord(inputFile):
    stopWordList = readFile(inputFile).splitlines()
    return stopWordList

def getTFIDFMat(inputPath, stopWordList, outputPath,

tftfidfspace_path,tftfidfspace_arr_path,tftfidfspace_vocabulary_path):    #
求得 TF-IDF 向量
    bunch = readBunch(inputPath)
    tfidfspace      =      Bunch(target_name=bunch.target_name,
label=bunch.label, filenames=bunch.filenames, tdm=[],
                                vocabulary={})

    '''读取 tfidfspace'''
    tfidfspace_out = str(tfidfspace)
    saveFile(tftfidfspace_path, tfidfspace_out)

    vectorizer      =      TfidfVectorizer(stop_words=stopWordList,
sublinear_tf=True, max_df=0.5)
    transformer = TfidfTransformer()

```

```

tfidfspace.tdm = vectorizer.fit_transform(bunch.contents)
tfidfspace_arr = str(vectorizer.fit_transform(bunch.contents))
saveFile(tfidfspace_arr_path, tfidfspace_arr)
tfidfspace.vocabulary = vectorizer.vocabulary_ # 获取词汇
tfidfspace_vocabulary = str(vectorizer.vocabulary_)
saveFile(tfidfspace_vocabulary_path, tfidfspace_vocabulary)
"over"
writeBunch(outputPath, tfidfspace)

def getTestSpace(testSetPath, trainSpacePath, stopWordList,
testSpacePath,

testSpace_path, testSpace_arr_path, trainbunch_vocabulary_path):
    bunch = readBunch(testSetPath)

    testSpace = Bunch(target_name=bunch.target_name,
label=bunch.label, filenames=bunch.filenames, tdm=[],
vocabulary={})
    """
    读取 testSpace
    """
    testSpace_out = str(testSpace)
    saveFile(testSpace_path, testSpace_out)

    trainbunch = readBunch(trainSpacePath)

    vectorizer = TfidfVectorizer(stop_words=stopWordList,
sublinear_tf=True, max_df=0.5,

vocabulary=trainbunch.vocabulary)
    transformer = TfidfTransformer()
    testSpace.tdm = vectorizer.fit_transform(bunch.contents)
    testSpace.vocabulary = trainbunch.vocabulary

```

```

testSpace_arr = str(testSpace.tdm)
trainbunch_vocabulary = str(trainbunch.vocabulary)
saveFile(testSpace_arr_path, testSpace_arr)
saveFile(trainbunch_vocabulary_path, trainbunch.vocabulary)
# 持久化
writeBunch(testSpacePath, testSpace)

def bayesAlgorithm(trainPath, testPath, tfidfspace_out_arr_path,
                  tfidfspace_out_word_path, testspace_out_arr_path,
                  testspace_out_word_apth):
    trainSet = readBunch(trainPath)
    testSet = readBunch(testPath)
    clf = MultinomialNB(alpha=0.001).fit(trainSet.tdm, trainSet.label)

    # print(shape(trainSet.tdm)) #输出单词矩阵的类型
    # print(shape(testSet.tdm))
    '''处理 bat 文件'''
    tfidfspace_out_arr = str(trainSet.tdm) # 处理
    tfidfspace_out_word = str(trainSet)
    saveFile(tfidfspace_out_arr_path, tfidfspace_out_arr) # 矩阵形式
    的 train_set.txt
    saveFile(tfidfspace_out_word_path, tfidfspace_out_word) # 文本
    形式的 train_set.txt

    testspace_out_arr = str(testSet)
    testspace_out_word = str(testSet.label)
    saveFile(testspace_out_arr_path, testspace_out_arr)
    saveFile(testspace_out_word_apth, testspace_out_word)

    '''处理结束'''
    predicted = clf.predict(testSet.tdm)
    total = len(predicted)
    rate = 0

```

```

        for flabel, fileName, expct_cate in zip(testSet.label, testSet filenames,
predicted):
            if flabel != expct_cate:
                rate += 1
                print(fileName, ":实际类别：", flabel, "-->预测类别：",
expct_cate)
            print("bayes model erroe rate:", float(rate) * 100 / float(total), "%")

```

```

def svm_algorithm(trainPath, testPath, tfidfspace_out_arr_path,
                  tfidfspace_out_word_path, testspace_out_arr_path,
                  testspace_out_word_apth):

```

```

    trainSet = readBunch(trainPath)

```

```

    testSet = readBunch(testPath)

```

```

    """改动点"""

```

```

    model = OneVsRestClassifier(svm.SVC(kernel='linear'))#选择线性
SBM 多分类组合器

```

```

    clf = model.fit(trainSet.tdm, trainSet.label)

```

```

    """处理 bat 文件"""

```

```

    tfidfspace_out_arr = str(trainSet.tdm) # 处理

```

```

    tfidfspace_out_word = str(trainSet)

```

```

    saveFile(tfidfspace_out_arr_path, tfidfspace_out_arr) # 矩阵形式
的 train_set.txt

```

```

    saveFile(tfidfspace_out_word_path, tfidfspace_out_word) # 文本
形式的 train_set.txt

```

```

    testspace_out_arr = str(testSet)

```

```

    testspace_out_word = str(testSet.label)

```

```

    saveFile(testspace_out_arr_path, testspace_out_arr)

```

```

    saveFile(testspace_out_word_apth, testspace_out_word)

```

```

"""处理结束"""

predicted = clf.predict(testSet.tdm)
total = len(predicted)
rate = 0
for flabel, fileName, expct_cate in zip(testSet.label, testSet filenames,
predicted):
    if flabel != expct_cate:
        rate += 1
        print(fileName, ":实际类别：", flabel, "-->预测类别：",
expct_cate)
print("svm model error rate:", float(rate) * 100 / float(total), "%")

#
if __name__ == '__main__':
    datapath = "/data/" #原始数据路径
    stopWord_path = "/stop/stopword.txt"#停用词路径
    test_path = "/test/" #测试集路径

    test_split_dat_path =  "/test_set.dat" #测试集分词 bat 文件路径
    testspace_dat_path = "/testspace.dat" #测试集输出空间矩阵 dat
文件
    train_dat_path = "/train_set.dat"

    tfidfspace_dat_path = "/tfidfspace.dat" #tf-idf 词频空间向量的 dat
文件
    """
    以上四个为 dat 文件路径，是为了存储信息做的，不要打开
    """
    test_split_path = '/split/test_split/' #测试集分词路径
    split_datapath = "/split/split_data/" # 对原始数据分词之后的数

```

据路径

'''

以上两个路径是分词之后的文件路径,大家可以生成之后自行打开查阅学习

'''

tfidfspace_path = "./tfidfspace.txt" # 将 TF-IDF 词向量保存为 txt, 方便查看

tfidfspace_arr_path = "./tfidfspace_arr.txt" # 将 TF-IDF 词频矩阵保存为 txt, 方便查看

tfidfspace_vocabulary_path = "./tfidfspace_vocabulary.txt" # 将分词的词汇统计信息保存为 txt, 方便查看

testSpace_path = "./testSpace.txt" #测试集分词信息

testSpace_arr_path = "./testSpace_arr.txt" #测试集词频矩阵信息

trainbunch_vocabulary_path = "./trainbunch_vocabulary.txt" #所有分词词频信息

tfidfspace_out_arr_path = "./tfidfspace_out_arr.txt" #tfidf 输出矩阵信息

tfidfspace_out_word_path = "./tfidfspace_out_word.txt" #单词形式的 txt

testspace_out_arr_path = "./testspace_out_arr.txt" #测试集输出矩阵信息

testspace_out_word_apth = "./testspace_out_word.txt" #测试界单词信息

'''

以上 10 个文件是 dat 文件转化为 txt 文件, 大家可以查询信息, 这是 NLP (自然语言处理) 非常珍贵的资源

'''

#输入训练集

segText(datapath,#读入数据

split_datapath)#输出分词结果

bunchSave(split_datapath,#读入分词结果

train_dat_path) # 输出分词向量

stopWordList = getStopWord(stopWord_path) # 获取停用词表

```

getTFIDFMat(train_dat_path, #读入分词的词向量
            stopWordList,    #获取停用词表
            tfidfspace_dat_path, #tf-idf 词频空间向量的 dat 文件
            tfidfspace_path, #输出词频信息 txt 文件
            tfidfspace_arr_path, #输出词频矩阵 txt 文件
            tfidfspace_vocabulary_path) #输出单词 txt 文件

#输入测试集
segText(test_path,
        test_split_path) # 对测试集读入文件，输出分词结果
bunchSave(test_split_path,
          test_split_dat_path) #
getTestSpace(test_split_dat_path,
             tfidfspace_dat_path,
             stopWordList,
             testspace_dat_path,
             testSpace_path,
             testSpace_arr_path,
             trainbunch_vocabulary_path)# 输入分词文件，停
用词，词向量，输出特征空间(txt,dat 文件都有)
svm_algorithm(tfidfspace_dat_path,
             testspace_dat_path,
             tfidfspace_out_arr_path,
             tfidfspace_out_word_path,
             testspace_out_arr_path,
             testspace_out_word_apth)

```

Weibo Text Classification Prediction Based on Machine Learning Algorithm

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

63/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

GRADING FORM: SMU GENERIC_GJ

TAO CHEN

62.50

RESEARCH-INFORMED LITERATURE



References - too few. Also use school standard.
Don't use superscript. Essential areas covered but
the algorithms used could be described in a little
more detail.

5

KNOWLEDGE AND UNDERSTANDING



Good contextualisation of problem and quite a lot of pre-processing required before algorithms could be used. More detail of algorithms and their strengths in this area could have been included. Methods used for evaluation need to be better documented.

17

ANALYSIS



You do not provide a detailed evaluation. The accuracy figures 9% and 11% are a start. but F1 scores, AUC, etc would help here. Your dataset could be better analysed - check/correct for biases. Have you applied cross-validation/ tuned your parameters and hyperparameters?

20

PRACTICAL APPLICATION AND DEP.



Some very good work here in pre-processing your dataset. Algorithms applied but you need to document your choice of parameters/hyperparameters.

18

SKILLS FOR PROFESSIONAL PRAC.



Comments in code are in Chinese - I can't read Chinese. Refs poor. One or two typos.

2.5