(三)朴素贝叶斯运用——文本分类 - Haward - CSDN博客

2018年09月15日 19:29:44 HawardScut 阅读数 1703

1、贝叶斯理论

当我们有样本（包含特征和类别）的时候，我们非常容易通过

$p(x)p(y|x) = p(y)p(x|y)$

统计得到 p(特征|类别) .即

$p(\text{特征})p(\text{类别}|\text{特征}) = p(\text{类别})p(\text{特征}|\text{类别})$

，有

$$p(\text{类别}|\text{特征}) = \frac{p(\text{类别})}{p(}$$

独立假设

特征往往是多维的，

$p(features|class) = p(t_0, t_1, \ldots, t_n|c)$

，这里假设为2维，有

$$p(t_0, t_1|c) = p(t_1|c, t$$

假设特征之间是独立的(朴素贝叶斯的思想)

$$p(t_0, t_1|c) = p(t_1|c)$$

即

$$p(t_0, t_1, \ldots, t_n|c) = \prod$$

贝叶斯分类器

对每个类别计算一个概率

$p(c_i)$

，然后再计算所有特征的条件概率

$p(t_j|c_i)$

，那么分类的时候我们就是依据贝叶斯找一个最可能的类别：

$$p(class_i|t_0, t_1, \ldots, t_n) = \frac{p(c}{p(t_0, t_1}$$

2、文本分类步骤

数据集说明：一条记录为“一段短文本新闻”，lable为“体育，军事，IT等”新闻类别。

（1）分词：把一条记录进行分词，保存为word_list，所有的记录保存在data_list，所有的类别

（2）划分训练集和测试集

（3）统计词频：统计所有**训练集**的每个词出现的次数，即词频，放入all_words_dict字典中，对

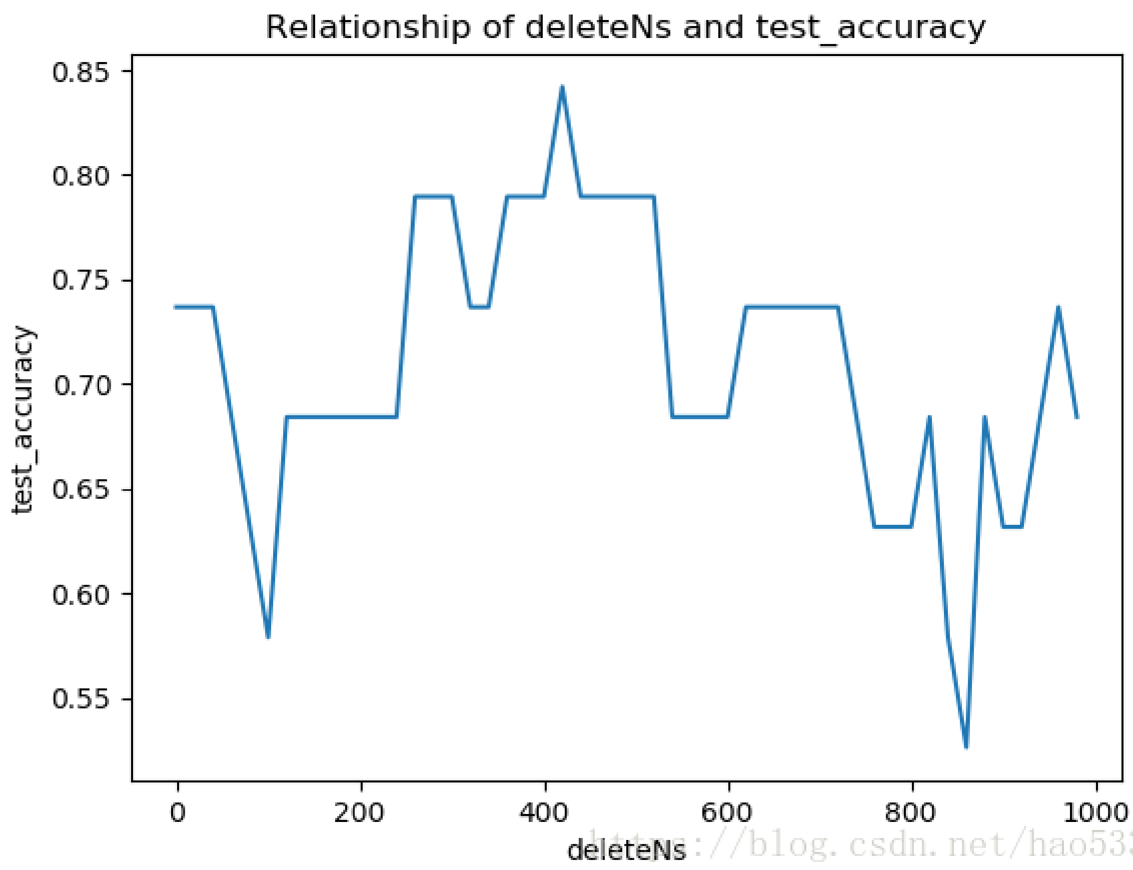
（4）停用词表：载入停用词表stopwords_set

（5）文本特征提取：选取n=1000个特征词（词频高到低依次选）保存在feature_words（选取a
表中的词，排除数字，并且要求词的长度为(1,5)）

（6）每条记录的表示（表示成长度为1000的特征词）：依次遍历feature_words中每个词，对
为1，否则为0。即 `features = [1 if word in text_words else 0 for word in feature_wo`

（7）训练，预测：使用MultinomialNB进行训练，预测

（8）deleteN：删掉前面的deleteN个feature_words词，即最高频的deleteN词不作为特征词，
步骤，得到另一个预测结果。deleteN的取值在某个范围时候，分类效果最好。（如下图，400多



3、代码

```
#coding: utf-8
import os
import time
import random
import jieba
import nltk
import sklearn
from sklearn.naive_bayes import MultinomialNB
import numpy as np
import pylab as pl
import matplotlib.pyplot as plt
#粗暴的词去重
def make_word_set(words_file):
    words_set = set()
    with open(words_file, 'r',encoding='UTF-8') as fp:
        for line in fp.readlines():
            word = line.strip()  #word = line.strip().decode("utf-8")
            if len(word)>0 and word not in words_set: # 去重
                words_set.add(word)
    return words_set


# 文本处理，也就是样本生成过程
def text_processing(folder_path, test_size=0.2):
    folder_list = os.listdir(folder_path)
    data_list = []
    class_list = []

    # 遍历文件夹
    for folder in folder_list:
        new_folder_path = os.path.join(folder_path, folder)
        files = os.listdir(new_folder_path)
        # 读取文件
        j = 1
        for file in files:
            if j > 100:   # 怕内存爆掉，只取100个样本文件
                break
            with open(os.path.join(new_folder_path, file), 'r', encoding='UTF
                raw = fp.read()
            ## jieba中文分词
            #jieba.enable_parallel(4) # 开启并行分词模式，参数为并行进程数，不支持windo
            word_cut = jieba.cut(raw, cut_all=False)  # 精确模式
            word_list = list(word_cut)  # genertor转化为list，每个词unicode格式
            #jieba.disable_parallel() # 关闭并行分词模式

            data_list.append(word_list)   # 训练集list
            class_list.append(folder.encode('utf-8'))  # 类别 class_list.append(fo
            j += 1

    ## 划分训练集和测试集
    data_class_list = list(zip(data_list, class_list)) #data_class_list = zip(da
    random.shuffle(data_class_list)
    index = int(len(data_class_list) * test_size) + 1
    train_list = data_class_list[index:]
    test_list = data_class_list[:index]
    train_data_list, train_class_list = zip(*train_list)
    test_data_list, test_class_list = zip(*test_list)
    # 统计词频放入all_words_dict
    all_words_dict = {}
    for word_list in train_data_list:
        for word in word_list:
            if all_words_dict.get(word)!=None:  #if all_words_dict.has_key(word):
                all_words_dict[word] += 1
            else:
                all_words_dict[word] = 1

    # 词频进行降序排序
    all_words_tuple_list = sorted(all_words_dict.items(), key=lambda f: f[
    all_words_list = list(list(zip(*all_words_tuple_list))[0])  #all_words_lis

    return all_words_list, train_data_list, test_data_list, train_class_lis


def words_dict(all_words_list, deleteN, stopwords_set=set()):
    # 选取特征词
    feature_words = []
    n = 1
    for t in range(deleteN, len(all_words_list), 1):
        if n > 1000:  # feature_words的维度1000
            break

        if not all_words_list[t].isdigit() and all_words_list[t] not in stop
                all_words_list[t]) < 5:
            feature_words.append(all_words_list[t])
            n += 1
    return feature_words

# 文本特征
def text_features(train_data_list, test_data_list, feature_words, flag='nltk'):
    def text_features(text, feature_words):
        text_words = set(text)
        ## -----------------------------------------------------------------------------------
        if flag == 'nltk':
            ## nltk特征 dict
            features = {word:1 if word in text_words else 0 for word in featur
        elif flag == 'sklearn':
            ## sklearn特征 list
            features = [1 if word in text_words else 0 for word in feature_wor
        else:
            features = []
        ## -----------------------------------------------------------------------------------
        return features
    train_feature_list = [text_features(text, feature_words) for text in tr
    test_feature_list = [text_features(text, feature_words) for text in tes
    return train_feature_list, test_feature_list

# 分类，同时输出准确率等
def text_classifier(train_feature_list, test_feature_list, train_class_list, test_class_list, flag='nltk'):
    ## -----------------------------------------------------------------------------------
    if flag == 'nltk':
        ## 使用nltk分类器
        train_flist = zip(train_feature_list, train_class_list)
        test_flist = zip(test_feature_list, test_class_list)
        classifier = nltk.classify.NaiveBayesClassifier.train(train_flist)
        test_accuracy = nltk.classify.accuracy(classifier, test_flist)
    elif flag == 'sklearn':
        ## sklearn分类器
        classifier = MultinomialNB().fit(train_feature_list, train_class_l
        test_accuracy = classifier.score(test_feature_list, test_class_lis
    else:
        test_accuracy = []
    return test_accuracy


print("start")

## 文本预处理
folder_path = './Database/SogouC/Sample'
all_words_list, train_data_list, test_data_list, train_class_list, test_cl

# 生成stopwords_set
stopwords_file = './stopwords_cn.txt'
stopwords_set = make_word_set(stopwords_file)

## 文本特征提取和分类
# flag = 'nltk'
flag = 'sklearn'
deleteNs = range(0, 1000, 20)
test_accuracy_list = []
for deleteN in deleteNs:
    # feature_words = words_dict(all_words_list, deleteN)
    feature_words = words_dict(all_words_list, deleteN, stopwords_set)
    train_feature_list, test_feature_list = text_features(train_data_list,
    test_accuracy = text_classifier(train_feature_list, test_feature_list,
    test_accuracy_list.append(test_accuracy)
print(test_accuracy_list)

# 结果评价
#plt.figure()
plt.plot(deleteNs, test_accuracy_list)
plt.title('Relationship of deleteNs and test_accuracy')
plt.xlabel('deleteNs')
plt.ylabel('test_accuracy')
#plt.show()
plt.savefig('result.png')

print("finished")
```

参考

[1]七月在线
[2]github代码