华中科技大学计算机科学与技术学院

机器学习报告

专　　业：	计算机科学与技术
班　　级：	
学　　号：	
姓　　名：	[illegible]
成　　绩：	
指导教师：	

完成日期： 2020 年 05 月 20 日

实验二

一、实验题目：朴素贝叶斯（垃圾邮件判别）

二、实验要求

用朴素贝叶斯算法实现垃圾邮件判别，实现要求如下：

1．对邮件进行分词处理以及简单的停用词处理。

2．得到正负样本的特征（对判定贡献最大的词）。

3．从离散特征、多项式特征、连续特征三种参数估计中至少选择一种实现朴素贝叶斯，进行垃圾邮件判别。

4．从混淆矩阵、精确率、召回率、F1 分数以及邮件分类的正确率对这个垃圾邮件判别系统进行评估。

三、算法设计

1．邮件文本处理

邮件文本中只读取中文汉字字符，使用 jieba 库的分词工具（精确模式），最后去掉邮件中所有的停用词。

2．朴素贝叶斯算法设计

统计训练集中所有出现的词和词频，存放在一个字典中，然后将训练集中的垃圾邮件中的词频和非垃圾邮件中的词频分开，分别放在两个字典中，这三个字典的键组成的集合完全相同。然后将垃圾邮件和非垃圾邮件中的词频分别除以总词频就分别得到了 P(spam|word)和 P(ham|word)组成的两个字典。对于测试集中的数据，根据选择按照离散特征或多项式特征对 P(spam|words)和 P(ham|words)进行估计，较大者为预测结果。

3．结果评估

按照混淆矩阵的概念统计真阳例(TP)、真阴例(TN)、假阳例(FP)和假阴例(FN)的数量，并且根据公式

$$\mathrm{precision} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$$

$$\mathrm{recall} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$$

$$F1 = 2 \times precision \times recall \div (precision + recall)$$

$$accuracy = (TP + TN) \div (TP + TN + FP + FN)$$

对测试结果进行评估，然后调用 matplotlib.pyplot 绘制出混淆矩阵图。同时比较离散特征和多项式特征两种参数估计的判别结果。

四、实验环境与平台

本实验使用的语言为 python，版本为 python3.7，实现的开发工具是 Pycharm 2020.1，系统为 Windows 10，CPU 为 i7-8550U，内存 8G，其后的测试也均在此软硬件环境下。

五、程序实现

```
import jieba
import re
import matplotlib.pyplot as plt
import math


def stopwordslist(filePath):
    """
    :param filePath: the file path of the stopwords list
    :return: the list of stopwords
    """
    stopwords = [line.strip() for line in open(filePath, 'r', encoding='utf-8').readlines()]
    return stopwords


def wordslist(filePath):
    """
    :param filePath: the file path of the text
    :return: the words list
    """
    words = []
    email = open(filePath, encoding='gb2312', errors='ignore').read()
    email = email[email.index("\n\n")::]          # to remove the header of the email
    pattern = re.compile("[^\u4e00-\u9fa5]")       # the range of encoded Chinese characters is \u4e00
to \u9fa5
    email = "".join(pattern.split(email)).strip()    # to remove all the non-Chinese characters
    words.extend(list(jieba.cut(email, cut_all=False, HMM=bool)))
    return words
```

```
def getindexes(filePath):
    """
    :param filePath: the file path of the index file
    :return: labels, indexes
    """
    strs = [line.strip() for line in open(filePath, 'r', encoding='utf-8').readlines()]
    labels = []
    indexes = []
    for i in range(len(strs)):
        if "ham" in strs[i]:
            labels.append(0)
            indexes.append("trec06c"+strs[i][6:])
        else:
            labels.append(1)
            indexes.append("trec06c"+strs[i][7:])
    return labels, indexes

def removestopwords(words, stopwords):
    """
    :param words: a words list
    :param stopwords: stopped words
    :return: stopwords removed words list
    """
    for word in words:
        if word in stopwords:
            words.remove(word)

def count2prob(counts, spam_counts, ham_counts):
    """
    :param words: word list
    :return: conditional probability of each word, spam and ham respectively
    """
    ham_probs = {}
    spam_probs = {}
    for word in counts.keys():
        ham_probs[word] = ham_counts[word] / counts[word]
        spam_probs[word] = spam_counts[word] / counts[word]
    return spam_probs, ham_probs

def training(indexes, training_size, stopwords, labels):
    """
    :param indexes: the file path indexes of each email
    :param training_size: the number of emails to be used for training process
    :param stopwords: stopped words
    :param labels: whether it is a spam or ham
    :return: spam_probs, ham_probs
    """
    wordslists = []  # the list of wordslist of all training emails
    words = []  # store all words in one list
    for i in range(training_size):
        tmp = wordslist(indexes[i])
        removestopwords(tmp, stopwords)
        wordslists.append(tmp)
        words += tmp
    counts = {}  # key: word, value: frequency
    for word in words:
        counts[word] = counts.get(word, 0) + 1
    d = counts.keys()
    z = [0 for _ in range(len(d))]
    spam_counts = dict(zip(d, z))  # initialization, key: word in spam, value: frequency in spam
    ham_counts = dict(zip(d,z))  # initialization, key: word in ham, value: frequency in ham
    for i in range(training_size):
        if labels[i]:
            for word in wordslists[i]:
                spam_counts[word] += 1
        else:
            for word in wordslists[i]:
                ham_counts[word] += 1
    spam_probs, ham_probs = count2prob(counts, spam_counts, ham_counts)   # convert frequency to
conditional probability
    return spam_counts, ham_counts, spam_probs, ham_probs

def NBpredict(spamprobs, hamprobs, spamprob, hamprob, stopwords, index, mode):
    """
    :param spamprobs: words in spams with probability
    :param hamprobs: words in hams with probability
    :param spamprob: the true probability of spam
    :param hamprob: the true probability of ham
    :param stopwords: stopped words
    :param index: the file path index of the email to be tested
    :param mode: Discrete or Multinomial
    :return: the prediction
    """
    words = wordslist(index)
    removestopwords(words, stopwords)
    counts = {}
    ham = math.log(hamprob)
    spam = math.log(spamprob)
    for word in words:
        counts[word] = counts.get(word, 0) + 1
    for word in counts.keys():
        if mode == 'Multinomial':  # to choose the Multinomial Feature
            times = counts[word]
        else:                      # to choose the Discrete Feature
            times = 1
        ham += times * math.log(hamprobs.get(word, 0.5)+0.001)
        spam += times * math.log(spamprobs.get(word, 0.5)+0.001)
    if ham > spam:
        return 0
    else:
        return 1

if __name__ == '__main__':
    stopwords = stopwordslist("stopwords_chinese.txt")
    labels, indexes = getindexes("trec06c/full/newindex")
    spamprob = sum(labels)/len(labels)
    hamprob = 1-(sum(labels)/len(labels))
    training_size = 35000
    spamcounts, hamcounts, spamprobs, hamprobs = training(indexes, training_size, stopwords, labels)
    spam_p_order = sorted(spamprobs.items(), key=lambda x: x[1], reverse=True)
    ham_p_order = sorted(hamprobs.items(), key=lambda x: x[1], reverse=True)
    spam_f_order = sorted(spamcounts.items(), key=lambda x: x[1], reverse=True)
    ham_f_order = sorted(hamcounts.items(), key=lambda x: x[1], reverse=True)
    print([key for (key, value) in spam_p_order[:50]])
    print([key for (key, value) in ham_p_order[:50]])
    print([key for (key, value) in spam_f_order[:50]])
    print([key for (key, value) in ham_f_order[:50]])

    predictions = []
    truth = []
    for i in range(15000):
        predictions.append(NBpredict(spamprobs, hamprobs, spamprob, hamprob, stopwords,
indexes[training_size+i], 'Discrete'))
        truth.append(labels[training_size+i])

    # to evaluate the result
    TP = 0
    TN = 0
    FP = 0
    FN = 0
    for i in range(len(predictions)):
        if predictions[i]:
            if truth[i]:
                TP += 1
            else:
                FP += 1
        else:
            if truth[i]:
                FN += 1
            else:
                TN += 1
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    F1 = 2 * precision * recall / (precision + recall)
    accuracy = (TP + TN) / (TP + TN + FP + FN)
    plt.imshow([[TP, FP], [FN, TN]], cmap='binary')
    plt.show()
    print('accuracy=', accuracy)
    print('precision=', precision)
    print('recall=', recall)
    print('F1=', F1)
```

六、实验结果

1．正负样本的特征（对判定贡献最高的 20 个词）

(1)P(spam|word)最高的 10 个词

财务经理、代开、增值税、国税、地税、普通发票、税点、税务局、抵扣、销售税

(2)P(ham|word)最高的 10 个词

贪财、总想、回忆、交代、负的起、旁观、否决、继续下去、勇敢、面对

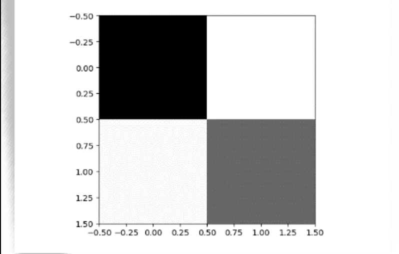
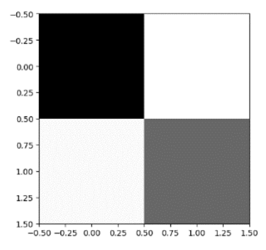
(3)P(word|spam)最高的 10 个词

公司、发票、有限公司、合作、企业、优惠、管理、服务、贵、联系人

(4)P(word|ham)最高的 10 个词（去掉单字）

知道、觉得、现在、时候、工作、喜欢、一起、自己、问题、已经

2．离散特征 VS 多项式特征（训练集为 newindex 前 35000 个索引指向的邮件，测试集为第 35000-50000 个索引指向的邮件）

参数估计方法	离散特征	多项式特征
真阳例(TP)	9056	9052
假阳例(FP)	172	185
假阴例(FN)	324	328
真阴例(TN)	5448	5435
混淆矩阵图		
精确率(precision)	0.9813610749891634	0.9799718523330085
召回率(recall)	0.9654584221748401	0.9650319829424308
F1 分数(F1-score)	0.9733447979363715	0.9724445399366171
正确率(accuracy)	0.9669333333333333	0.9658