華東理工大學

模式识别大作业

题　　目	使用朴素贝叶斯对短信进行分类
学　　院	信息科学与工程
专　　业	控制科学与工程
组　　员	胡阳、胡云松
指导教师	赵海涛

完成日期：　2019 年 12 月 5 日

基于朴素贝叶斯的短信分类

组员：胡阳、胡云松

一、基于朴素贝叶斯的短信分类问题的简介

朴素贝叶斯分类是一种基于贝叶斯定理的简单概率贝叶斯分类，具有很强的(所谓朴素的)假设独立性。这种受监督的机器学习算法在线性分类器家族中非常流行。例如，垃圾邮件过滤器在电子邮件应用程序中的功能就是基于这种算法的。在 Kaggle 的这道题目中，我们将使用包含千条短信消息来创建一个模型来预测接收的消息是否是垃圾短信，使用的就是朴素贝叶斯算法。

二、数据描述统计

2.1 数据结构分析

训练集和测试集数据是从 Kaggle 上下载，训练数据 train.csv 和测试数据 test_cleaned.csv。部分训练集部据的如下所示：

表 2.1 训练集数据

V1	V2
Ham	Go until jurong point,crazy. Available only in bugis n great world…
Ham	Ok lar… Joking wif u oni…
Spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.
Ham	U dun say so early hor… U c already then say…
Ham	Nah I don’t think he goes to usf, he lives around here though
Spam	FreeMsg Hey there darling it’s been 3 week’s now and no word back!...
……	……

训练集一共有 4471 组训练数据，其中 V1 代表着标签：Ham 表示非垃圾短信，Spam 表示垃圾短信。V2 的就是短信的内容。

2.2 数据预处理

数据预处理由以下几部分完成：

（1）将 V1 标签中的 Ham 变成 0，标签中的 Spam 变成 1；

（2）将非字母的字符替换成空格从而将 V2 标签内的短信进行分割。

（3）将分割以后的单词全部转换成小写，避免相同的单词因大小写不同而被识别成不同的单词。

（4）去除一些停用词：这一步就是把一些常用的词类似“你”、“我”、“是”等词，因为无论是在中文还是英文中，这些词在语句中也只是起到了支撑句子结构的作用，其本身对于表达句子的含义并没有帮助。

这部分我们使用 python 进行编程，代码如下：

```
import re
import pandas as pd

class Fileoperate:
    def __init_(self, data_path):
        data = pd.read_excel(data_path)
        self.data = data.iloc[:, :2]
    def load_data(self):
    x = list()
    y = list()

    for i in range(len(self.data)):
        x.append(FileOperate.__clean_data(self.data.iloc[i, 1]))
        # 将标签为 spam 的转换成 1，标签为 ham 的转换成 0
        y.append(1 if self.data.iloc[i, 0] == ‘sapm’ else 0)
    return x, y

    def __clean_data(origin_info):
    ’’’
    清洗数据，去掉非字母的字符和长度小于 2 的单词
    ’’’
    # 先装换成小写
    # 把标点符号都换成空格
    if type(origin_info) != type(‘,’):
```

```
            origin_info = str(origin_info)
        temp_info = re.sub('W', ' ', origin_info.lower())
        # 根据空格来进行分割
        words = re.split(r'\s+', temp_info)
    # 保留长度大于等于 3 的单词
    return list(filter(lambda x: len(x) >= 3, words))
```

数据预处理之后：

x = [['until', 'jurong', 'point', 'crazy', 'available', 'only'…],

['lar', 'joking', 'wif', 'oni'], ['free', 'entry', 'wkly', 'comp', …]…]

y = [0, 0, 1, 0, 0, 1…]

接下来就是通过训练集进行训练：

（1）首先通过训练集统计训练集中的垃圾短信和非垃圾短信的数量，为之后计算先验概率做准备。

（2）然后针对两个类别建立两个字典，这两个字典分别统计在垃圾短信和非垃圾短信中各个单词出现的次数。

（3）需要注意的是，测试集中可能会有一些单词在训练集中没有出现过，那时频数为 0，频率也为 0，于是连乘之后的积就为 0，而这显然是不合理的，所以我们就是用了拉普拉斯修正。

```
class NaiveBayes:


    def __init__(self):
        self.__ham_count = 0    # 非垃圾短信数量
        self.__spam_count = 0    # 垃圾短信数量

        self.__ham_words_count = 0    # 非垃圾短信单词总数
        self.__spam_words_count = 0    # 垃圾短信单词总数

        self.__ham_words = list()    # 非垃圾短信单词列表
        self.__spam_words = list()    # 垃圾短信单词列表

        # 训练集中不重复单词集合
        self.__word_dictionary_set = set()
        self.__word_dictionary_size = 0
```

```
        self.__ham_map = dict()   # 非垃圾短信的词频统计
        self.__spam_map = dict()   # 垃圾短信的词频统计

        self.__ham_probability = 0
        self.__spam_probability = 0
    # fit()用来训练
    def fit(self, X_train, y_train):
        self.build_word_set(X_train, y_train)
        self.word_count()

    def predict(self, X_train):
        return [self.predict_one(sentence) for sentence in X_train]

    def build_word_set(self, X_train, y_train):
        '''
        第 1 步：建立单词集合
        :param X_train:
        :param y_train:
        :return:
        '''
        for words, y in zip(X_train, y_train):
            if   y == 0:
                # 统计训练集中非垃圾短信的数量以及各个词出现的频数
                self.__ham_count += 1
                self.__ham_words_count += len(words)
                for word in words:
                    self.__ham_words.append(word)
                    self.__word_dictionary_set.add(word)
            if   y == 1:
                # 统计训练集中垃圾短信的数量以及各个词出现的频数
                self.__spam_count += 1
                self.__spam_words_count += len(words)
                for word in words:
                    self.__spam_words.append(word)
```

```
                    self.__word_dictionary_set.add(word)

        # print('非垃圾短信数量', self.__ham_count)
        # print('垃圾短信数量', self.__spam_count)
        # print('非垃圾短信单词总数', self.__ham_words_count)
        # print('垃圾短信单词总数', self.__spam_words_count)
        # print(self.__word_dictionary_set)
        self.__word_dictionary_size = len(self.__word_dictionary_set)

    def word_count(self):
        # 第 2 步：获取不同类别下的词频
        for word in self.__ham_words:
            self.__ham_map[word] = self.__ham_map.setdefault(word, 0) + 1

        for word in self.__spam_words:
            self.__spam_map[word] = self.__spam_map.setdefault(word, 0)+1

        # 下面两行计算先验概率
        # 非垃圾短信的概率
                    self.__ham_probability = self.__ham_count /
            (self.__ham_count + self.__spam_count)
        # 垃圾短信的概率
                    self.__spam_probability = self.__spam_count /
            (self.__ham_count + self.__spam_count)

    def predict_one(self, sentence):
        ham_pro = 0
        spam_pro = 0
    # 使用拉普拉斯修正方法
        for word in sentence:
            # print('word', word)
            ham_pro += math.log(
                        (self.__ham_map.get(word,  0)  +  1)  /
          (self.__ham_count + self.__word_dictionary_size))
```

```
                spam_pro += math.log(
                                (self.__spam_map.get(word,  0)    +    1)    /
            (self.__spam_count + self.__word_dictionary_size))

            ham_pro += math.log(self.__ham_probability)
            spam_pro += math.log(self.__spam_probability)
            # print('垃圾短信概率', spam_pro)
            # print('非垃圾短信概率', ham_pro)
        # 如果垃圾短信的概率高于非垃圾短信的概率，则返回 1，反之返回 0。
            return int(spam_pro >= ham_pro)
```

三、原理公式

3.1 贝叶斯公式

设 x 为数据样本，c_i 为样本类别，贝叶斯条件概率公式为：

$$P(c_i|x)=\frac{P(x|c_i)P(c_i)}{P(x)} \tag{1}$$

其中 i=1,2，表示 ham 和 spam 两种类别。朴素贝叶斯假设样本特征独立同分布，即

$$P(x|c_i)=P\left(x_1|c_i\right)P\left(x_2|c_i\right)\cdots P\left(x_n|c_i\right) \tag{2}$$

根据（1）式，对所有数据来说，$P\left(x\right)$都相同。$P\left(c_i\right)$是先验概率，能直接得到。因此，需要对（2）式的两个类别做词频统计。

3.2 拉普拉斯修正

训练集上，很多样本的取值可能并不在其中，但是这不并代表这种情况发生的概率为 0，因为未被观测到，并不代表出现的概率为 0 。因此，进行拉普拉斯修正是必须的。

在训练集中总共的分类数，用 N 表示；di 属性可能的取值数用 Ni 表示，因此原来的先验概率 P(c) 的计算公式由：

$$P\left(c\right)=\frac{D_c}{D} \tag{3}$$

被拉普拉斯修正为：

$$P(c)=\frac{D_c+1}{D+N} \tag{4}$$

类的条件概率 $P(x|c)$ 的计算公式由：

$$P(c)=\frac{D_{c,x_i}}{D_c} \tag{5}$$

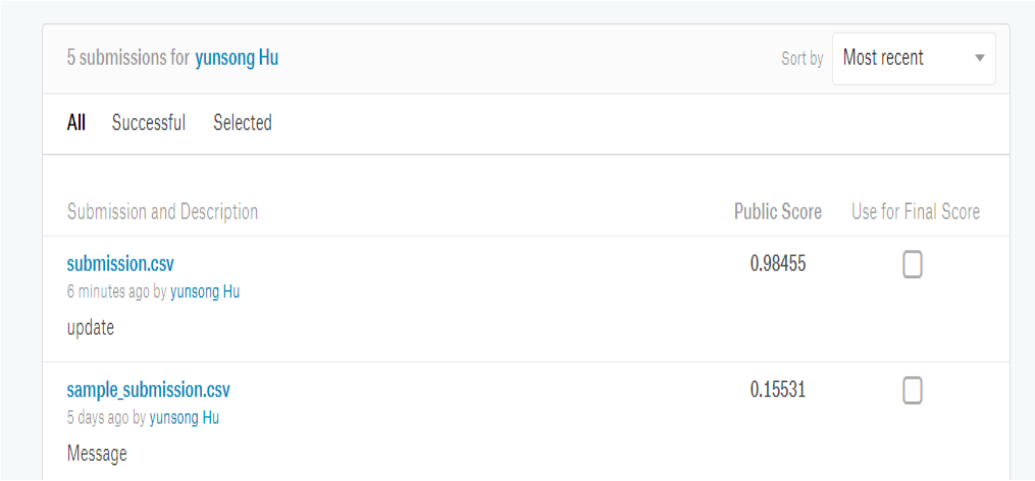
被拉普拉斯修正为：

$$P(c)=\frac{D_{c,x_i}+1}{D_c+N_i} \tag{6}$$

四、实验结果

```
if    name ==’main ’
        f = FileOperate(r’train.xlsx’)
        NB = NaiveBayes()
        x_train, y_train = f.load_data()
        NB.fit(x_train, y_train)
        f_test = FileOperate(r’test_cleanned.xlsx’)    # 将测试数据输入进行预测
        x_test = f_test.load_data_test()
        prediction = NB.predict(x_test)
        dataframe = pd.DataFrame(prediction, column=[‘prediction’])
        dataframe.to_csv(r”test_prediction.csv”)    # 将预测出的数据转换成 csv 文件输出
```

将从 kaggle 下载的测试数据输入，预测的数据通过 csv 文件的形式输出。我们将预测的数据复制到 kaggle 给定格式的提交文件中，然后提交。



我们第一次提交发现得分只有 0.15531，这个得分和其他的参赛人员比起来相差很多。通过查阅相关资料，我们发现关键之处在于我们预处理数据时我们去掉非字母的字符以及长度小于 2 的单词，以这样的规则来预处理数据不彻底。之后我们通过 nltk 模块里面的禁用词来处理数据就取得了非常好的效果，最后的得分是 0.98455。

```
stop_words = nltk.corpus.stopwords('english')
# return list(filter(lambda x: len(x) >= 3), words)   # 替换之前
return list(filter(lambda x: x if x not in set(stop_words) else '',words))   # 使用禁用词
```

五、总结

通过这次大作业令我们更加细致地理解了朴素贝叶斯分类以及拉普拉斯修正的原理。除此之外，我们还学习了 python 这门语言，python 拥有很多的模块，十分适合处理数据，今后有必要进一步深入地学习。而这次大作业令我收获最大的是锻炼了我们独自处理问题的能力，当拿到一个问题之后，如何通过网上的资源去解决它。Github 是如今最大的开源社区，里面有着无数的程序员的开源代码供我们学习，我们正是通过在 Github 学习别人的代码才能解决这个问题。最后还要感谢赵海涛老师，是您让我们对模式识别这个领域有了初步的了解，而这也令我们受益匪浅。