

**课 程 报 告**

**题目：机器学习自动判别垃圾邮件方法初**

**探**

**院（系）： 计算机系**

**课程名称： 人工智能导论（专选课）**

**学 号：**

**姓 名：**

**任课老师：**

**报告日期： 2020/6/13**

**目 录**

**1绪论** 3

1.1**课题背景与意义** 3

1.2**国内外研究现状** 3

2 **算法应用** 4

2.1**朴素贝叶斯分类器** 4

2.2 **KNN过滤算法** 5

2.3**决策树分类算法** 6

**3 总结与展望** 8

3.1 **总结** 8

3.2 **展望** 8

**参考文献** 9

**附录** 10

**1绪论**

**1.1课题背景与意义**

当前我们的生活中网络无处不在，如影随形的还有大量的垃圾邮件，他们不仅占用网络资源、浪费网络用户时间，还可能传播病毒，对国家、企业和个人的信息安全造成极大的威胁本文旨在回顾复习人工智能导论所学内容，并且结合网络机器学习论文报告文章初探机器学习在垃圾邮件分类方面的应用。

**1.2国内外研究现状**

根据网上的资料了解，当下主要的垃圾邮件分类算法大致分为三种：黑白名单过滤方法、基于规则的垃圾邮件过滤方法和基于统计的垃圾邮件过滤算法。

1. 黑白名单过滤：大致为黑白名单，若一个IP发送垃圾邮件的数目达到阈值，就将该IP放入黑名单中，之后的邮件都将被归类为垃圾邮件；而白名单与之相反。当然实际中大多非法邮件地址可能会处于变化状态，因而这种过滤算法也是有局限性的。
2. 基于规则的过滤技术：典型即是决策树。可以用在增量式学习上，算法可以根据数据的增多加以调整。根本原理是根据与假设规则比较的结果来判断垃圾邮件。
3. 基于统计的过滤技术：这种算法应用较广，最常见的算法有朴素贝叶斯算法和SVM算法，算法速度较快、效率高，易于实现、思路简单。但是这种技术也依赖大量的样本，对于贝叶斯算法，算法准确率不是很高，至于SVM算法，计算量以及空间复杂度也较高。

同时也有行为识别技术，并将基于规则和基于统计的过滤技术统归为基于内容的过滤技术。而行为识别技术是指基于邮件收发行为，通过研究发信人的行为来推断所发送的邮件是否为垃圾邮件。

**2 算法应用**

相较而言，对于普通用户，利用已有的有限邮件来训练数据集以实现分类器是较为简易的。因此优先选择了朴素贝叶斯算法、KNN算法进行研究，并且研究了基于规则的决策树分类器。在网络上参考资料时发现其实有很多改进的方案，例如可以使用TSVM-NB算法（改进的朴素贝叶斯），或者基于MapReduce对贝叶斯进行改进，但毕竟时间能力有限，只是粗浅研究了最为基础的算法思想。

**2.1 朴素贝叶斯分类器**

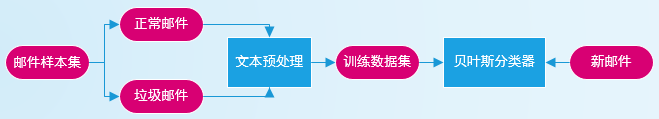


图2-1贝叶斯分类器过程

1. 初始化数据集：

毕竟能够获得的样本是有限的，那么可以先人为地依靠经验设置一个初始具有句子特征的词汇表，对其进行分类：0代表垃圾邮件代表文字，1代表正常文字。生成一个词汇表集合。

1. 求解先验概率和条件概率：

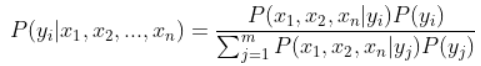
依照训练的文本以及词汇表，可以训练得出“垃圾文字”的先验概率，即等于“垃圾文字“的数量/邮件的总数量；以及条件概率，即是每个文字在正常和”垃圾“条件下的概；垃圾邮件中词汇表中单词出现的概率，及正常邮件中词汇表单词出现的频率。

我们输入已经用向量表示的训练集、邮件的类别、训练集的数目以及训练集的长度，输出垃圾邮件中频率——P(Wi|垃圾邮件)表，以及正常邮件中频率——P(Wj|正常邮件)表。

1. 邮件分类（主函数执行）：

可以先判断一下数据集当中的文字是否存在于新邮件当中，当然也可以不这样做，只是如果不包含数据集词汇可以节约一下时间。

利用算法



输入数据集中的分量、正常类和垃圾类下单词的条件概率、垃圾邮件的概率，输出得到一个概率，若是该概率高于50%，则该邮件被认为是垃圾邮件

**2.2 KNN过滤算法**

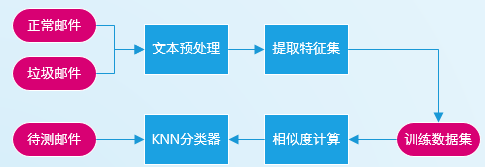


图2-2 KNN算法过程

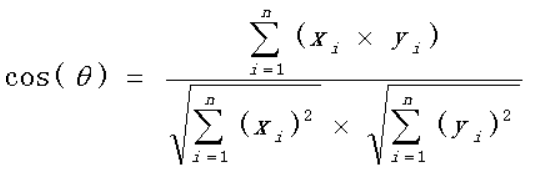
KNN算法是一种基于统计的算法，其实思想与朴素贝叶斯相同，区别在于训练数据集的算法区别。

1. 文本预处理：

将文本用向量空间模型表示，然后对词进行特征选择，试着提高运行效率。

1. 相似度计算：

利用向量空间余弦相似度算法衡量：



待邮件转化为特征向量后，计算测试集邮件与训练集邮件中的文本相似程度，右侧即输入向量的各维，n即是整个向量的维数，得到左侧的测试集和样本集的相似度。通过对比相似度，可以挑选出cos值最高的K封邮件，用这些邮件组成样本。利用权重算法：



依次计算正常邮件和垃圾邮件的权值，然后根据权值大小将测试邮件分类到所属邮件类型之中。

KNN算法对于K值依赖性极大，需要经过算法评估来不断确定K值，主要评判标准有正确率、召回率、代价因子等。网络上也有基于KNN改进的算法，改进后K值的影响降低，但同时时间复杂度上升。

**2.3决策树分类算法**

基于规则的垃圾邮件是通过对拥有的邮件进行训练，得到规则后利用该规则对邮件进行过滤，但是此规则难以提取，邮件基数需求极大，对于普通用户，实际效果并不佳。

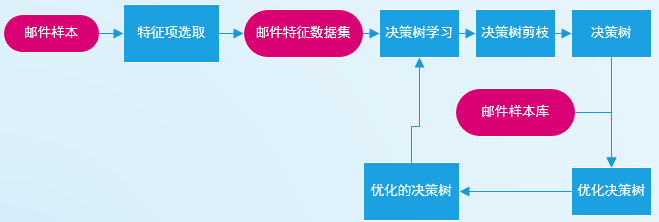


图2-3 决策树分类过程

1. 邮件预处理：

对训练集邮件进行预处理，提取出特征信息，然后从特征信息中再度选出最为代表垃圾邮件的特征，构建垃圾邮件特征集。特征集中的元素应该能够充分代表目标，出现频率应该足够高，同时应该与其他各个元素有明显区分，类别没有模糊性。

1. 存储数据：

邮件已经由向量空间模型表示，特征项以及邮件属性都应存储于数据库当中。

1. 构造决策树：

基于ID5R算法【1】，1)根结点为空then将实例放人根结点的实例集,并把该实例的分类作为根结点的分类.此时,根结点的实例集中只有一个例子,并且此时的根结点可以被看成是一个叶结点.2)else将实例沿决策树进行匹配,直到到达一个叶结点.3) if实例和该叶子结点中的其他实例是同一分类then将这个实例加人该叶结点的实例集.4) else:①将实例加人该叶结点的实例集中，随意选择一个属性对该结点进行划分.②对所有结点的所有属性,根据新增加的训练实例更新正反例计数.③if某个结点已用于划分的属性对该点来说信息增益值不是最大then调用上拉过程使该结点用对它米说信息增益值最大的属性进行划分;对它的各子树递归地进行更新,使每个子结点用于划分的属性对该结点米说都是最优的,其中不包括新加人实例的结点.对新加入实例的结点,递归地更新该结点的子树,必要时可以生成新的分枝.

1. 剪枝：

经过衡量标准如正确率、遗漏率等进行评价，然后根据决策树规模进行剪枝。采用预剪枝算法可以决定树的高度，将子树去除并且用叶结点替换，并且在相应的叶子上标出所属类所占实例比例。之后可以利用分类器判断样本类别，根据结果判断正确率重整决策树。

**3总结与展望**

**3.1总结**

完成本文的过程中有诸多挫折，虽然尽力解决，但由于对于机器学习方面仍有很多不了解之处，有些问题依然未能得到解答，在探索过程中却也是受益匪浅。比如一开始选题其实选择的是图像分类，但是在探究过程中发现对于python掌握过浅，SVM的算法理解得也并不透彻，结合网上资料有些部分也不好理解，无法自己编写，便放弃了这个课题。

最初的探究也是搞得一头雾水，但是在回顾魏老师的课程内容，以及学习理解网络上的论文报告之后，也是有所启发，最终是基础粗浅地对监督学习算法在邮件分类应用上有些许理解。

由于疫情原因，很遗憾未能在学校学习这门课，而且学时不长，很多知识只能带过无法深入，但是老师的课程也是足够清晰明了，学习之后希望能够以后有机会继续学习机器学习以及人工智能方面的内容。感谢老师的付出！

**3.2展望**

不仅是在垃圾邮件分类这一课题上，机器学习在当今社会应用十分广泛。但是当今的很多算法在某些方面都存在着部分弊端，例如前文所提到过的SVM算法计算量以及空间复杂度较高。而将来垃圾邮件的存在形式必不止于文字文本，还需要能够针对视频、图片、音频等类别的算法分类器。人工智能前辈们也已经对现存的各类算法做出各种基于速度或者正确率的改进，但是算法发展前景依然巨大，能够解决新社会问题的最佳方法或许仍有待发现。

这次学习也发现机器学习其实是一个基于各项学科诸如微积分、概率论、离散数学的综合项目，目前个人在这方面的学习也极大地受到学科知识方面的限制，很多更为高级有效的算法没有能力掌握。希望日后在能力增进之后能够在人工智能学科上走得更远。

**参考文献** (黑体小2号加粗居中)

[1]包理群，柴世红.决策树在垃圾邮件过滤中的应用. https://www.ixueshu.com/document/5aee60bc15f00b98668957a126b98c55318947a18e7f9386.html

[2]朴素贝叶斯. https://www.cnblogs.com/huangyc/p/10327209.html.

[3]jerry81333.KNN\_mail.py.https://github.com/jerry81333/SpamSMSFiltering/blob/master/KNN\_mail.py

**附 录**

1.朴素贝叶斯算法代码实现【2】

posting\_list = [

['my', 'dog', 'has', 'flea', 'problem', 'help', 'please'],

['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],

['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],

['stop', 'posting', 'stupid', 'worthless', 'garbage'],

['mr', 'licks', 'ate', 'ny', 'steak', 'how', 'to', 'stop', 'him'],

['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']

]

labels = [0, 1, 0, 1, 0, 1]

def createVocabList(dataSet):

vocabSet = set([])

for document in dataSet:

vocabSet = vocabSet | set(document)

return list(vocabSet)

def bagOfWords2Vec(vocabList, inputSet):

returnVec = [0] \* len(vocabList)

for word in inputSet:

if word in vocabList:

returnVec[vocabList.index(word)] += 1

else:

print("the word: %s is not in my vocabulary!" % word)

return returnVec

def fit(self, trainMatrix, trainCategory):

numTrainDocs = len(trainMatrix)

numWords = len(trainMatrix[0])

self.pAbusive = sum(trainCategory) / float(numTrainDocs)

p0Num = np.ones(numWords)

p1Num = np.ones(numWords)

p0Denom = 2.0

p1Denom = 2.0

for i in range(numTrainDocs):

if trainCategory[i] == 1:

p1Num += trainMatrix[i]

p1Denom += sum(trainMatrix[i])

else:

p0Num += trainMatrix[i]

p0Denom += sum(trainMatrix[i])

self.p1Vect = np.log(p1Num / p1Denom)

self.p0Vect = np.log(p0Num / p0Denom)

return self

def predict(self, testX):

p1 = np.sum(testX \* self.p1Vect) + np.log(self.pAbusive)

p0 = np.sum(testX \* self.p0Vect) + np.log(1 - self.pAbusive)

if p1 > p0:

return 1

else:

return 0

2.KNN算法代码实现【3】

from numpy import \*

import random

from sklearn import neighbors

import time

def creatVocabList(Data):

List = set([])

for document in Data:

List=List | set(document)

return list(List)

def textList(bigString):

import re

listOfTokens=re.split(r'\W\*', bigString)

return [tok.lower() for tok in listOfTokens if len(tok) > 2]

def dataToList(vocabList, inputSet):

returnVec = [0]\*len(vocabList)

for word in inputSet:

if word in vocabList:

returnVec[vocabList.index(word)] += 1

return returnVec

def trainKNN(trainData,trainLabel):

knn = neighbors.KNeighborsClassifier(n\_neighbors=3,algorithm='ball\_tree')

#print(type(trainData))

return knn.fit(trainData,trainLabel)

def mailfilter(K):

txtList=[]; classList = []; fullText =[]

for i in range(1,K+1):

wordList = textList(open('spam/%d.txt' % i).read())

#print (wordList)

txtList.append(wordList)

fullText.extend(wordList)

classList.append(1)

wordList=textList(open('ham/%d.txt' % i).read())

txtList.append(wordList)

fullText.extend(wordList)

classList.append(0)

vocabList=creatVocabList(txtList) #create vocabulary

trainingSet=list(range(2\*K))

#create test set

testSet=[]

for i in range(int(K/2)):

randIndex = int(random.uniform(0,len(trainingSet)))

testSet.append(trainingSet[randIndex])

del(trainingSet[randIndex])

trainMat=[];trainClasses =[]

start=time.clock()

for i in trainingSet:#train the classifier (get probs) naiveBayesTrain

trainMat.append(dataToList(vocabList, txtList[i]))

trainClasses.append(classList[i])

clf=trainKNN(array(trainMat),array(trainClasses))

right=0

errorCount=0

shouldSpam=0

shouldHam=0

for i in testSet: #classify the remaining items

wordVector = dataToList(vocabList, txtList[i])

if(clf.predict(array(wordVector)) == classList[i]):

right+=1

elif (clf.predict(array(wordVector))==0 and classList[i]==1):

errorCount += 1

shouldSpam += 1

#print ("Classification error",txtList[i])

elif (clf.predict(array(wordVector))==1 and classList[i]==0):

errorCount+=1

shouldHam+=1

#print("Classification error",txtList[i])

Recall=right/(right+shouldSpam)

Precision=right/(right+shouldHam)

Error=errorCount/len(testSet)

#print('Recall:',Recall)

#print('Precision:',Precision)

#print ('Error rate: ',Error)

end=time.clock()

Time=end-start

#print("Time:",Time,"s")

#return vocabList,fullText

return Recall,Precision,Error,Time

i=0

MaxRecall=0

MaxPrecision=0

MinError=1

AveRecall=0

AvePrecision=0

AveError=0

AveTime=0

K=10

while K<=80:

while i<10:

Recall,Precision,Error,Time=mailfilter(K)

AveRecall=AveRecall+Recall

AvePrecision=AvePrecision+Precision

AveError=AveError+Error

AveTime=AveTime+Time

if(Recall>MaxRecall):

MaxRecall=Recall

if(Precision>MaxPrecision):

MaxPrecision=Precision

if(MinError>Error):

MinError=Error

i+=1

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('K=',K)

print('MaxRecall:',MaxRecall)

print('MaxPrecision:',MaxPrecision)

print('MinError:',MinError)

print('AveRecall:',AveRecall/i)

print('AvePrecision:',AvePrecision/i)

print('AveError:',AveError/i)

print('AveTime:',AveTime/i)

i=0

K+=10

MaxRecall=0

MaxPrecision=0

MinError=1

AveRecall=0

AvePrecision=0

AveError=0

AveTime=0