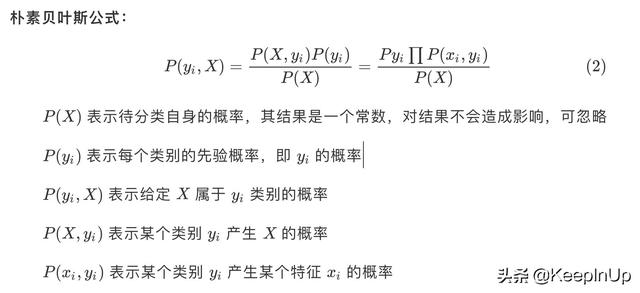
朴素贝叶斯

朴素贝叶斯中的朴素是指假设各个特征之间相互独立，不会互相影响，所以称为朴素贝叶斯。正是因为这个假设，使得算法的模型简单且容易理解，虽然牺牲了一点准确性，但是如果模型训练的好，也能得到不错的分类效果。

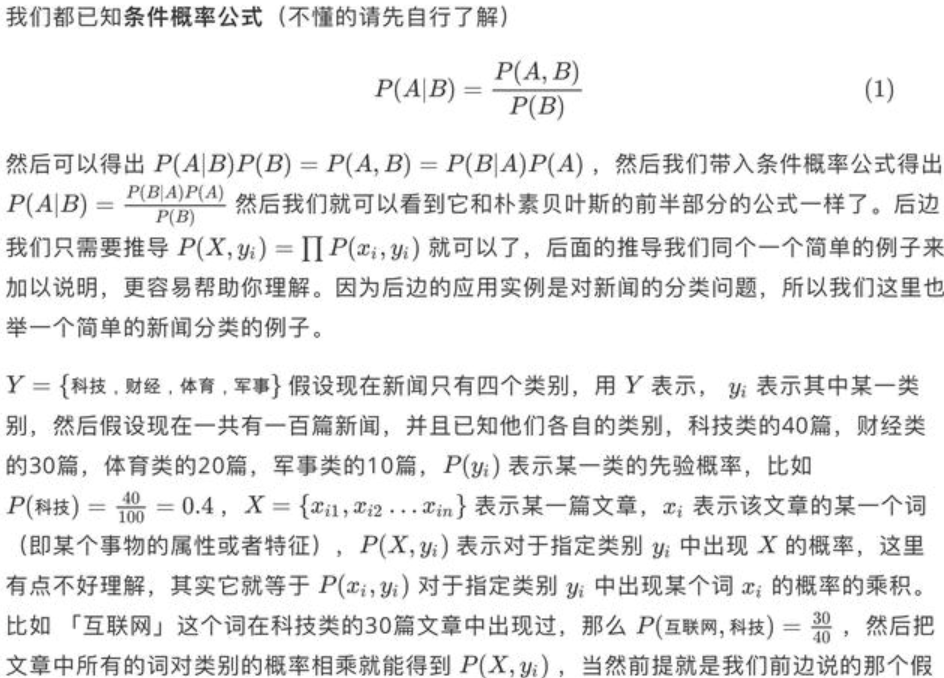
 





公式简单推导

下面我们简单看一下公式的推导过程



假。



表中表示实际上科技类的文章有 40 篇，财经类的有 30 篇，然而预测的结果科技类的有 35 篇，其中 30 篇预测正确了，有 5 篇预测错误了；预测结果财经类的有 35 篇，其中 25 篇预测正确了，10 篇预测错误了。

准确度

表示预测正确的文章数比上总的文章数：(30+25)/(40+30)=78%

精确率

表示每一类预测正确的数量比上预测的该类文章总数量，比如科技类精确率：30/(30+5)=85%

召回率

表示每一类预测正确的数量比上实际该类的总数量，比如科技类：30/40=75%

应用实例

上边我们已经了解了朴素贝叶斯公式及推导过程，下边我们来看一下在实际的新闻分类中的应用。

元数据的准备，我们的元数据是网上找来的一些各类的新闻，这里为了简单，我们只选取了科技、财经和体育三类数量不等的新闻，并且都已知他们的类别。然后通过中文结巴分词

下面我们来看一下代码的具体实现。

首先我们先把汉字的文章转成每个词所对应的数字id的形式，方便我们后边的操作和计算。

Convert.py

import os

import sy

import rando

import re

inputPath = sys.argv[1]

outputFile = sys.argv[2]

#训练集所占百分比

trainPercent = 0.8

wordDict = {

wordList = []

trainOutputFile = open('%s.train' % outputFile, "w")

testOutputFile = open('%s.test' % outputFile, "w")

for fileName in os.listdir(inputPath)

tag =

if fileName.find('technology') != -1:

tag = 1

elif fileName.find('business') != -1:

tag =

elif fileName.find('sport') != -1:

tag =

outFile = trainOutputFile

rd = random.random()

if rd >= trainPercent:

outFile = testOutputFile

inputFile = open(inputPath+'/'+fileName, "r")

content = inputFile.read().strip()

content = content.decode('utf-8', 'ignore')

content = content.replace('

content = re.sub(r1, '', content)

outFile.write(str(tag)+' ')

words = content.split(' '

or word in words:

if word not in wordDict:

wordList.append(word)

wordDict[word] = len(wordList)

outFile.write(str(wordDict[word]) + ' ')

inputFile.close()

trainOutputFile.close()

testOutputFile.close()

朴素贝叶斯实现过程

NB.py

#Usage:

#Training: NB.py 1 TrainingDataFile ModelFile

#Testing: NB.py 0 TestDataFile ModelFile OutFile

​

import sys

import os

import math

​

​

DefaultFreq = 0.1

TrainingDataFile = "nb\_data.train"

ModelFile = "nb\_data.model"

TestDataFile = "nb\_data.test"

TestOutFile = "nb\_data.out"

ClassFeaDic = {}

ClassFreq = {}

WordDic = {}

ClassFeaProb = {}

ClassDefaultProb = {}

ClassProb = {}

​

#加载数据

def LoadData():

i =0

infile = open(TrainingDataFile, 'r')

sline = infile.readline().strip()

while len(sline) > 0:

pos = sline.find("#")

if pos > 0:

sline = sline[:pos].strip()

words = sline.split(' ')

if len(words) < 1:

print("Format error!")

break

classid = int(words[0])

if classid not in ClassFeaDic:

ClassFeaDic[classid] = {}

ClassFeaProb[classid] = {}

ClassFreq[classid] = 0

ClassFreq[classid] += 1

words = words[1:]

for word in words:

if len(word) < 1:

continue

wid = int(word)

if wid not in WordDic:

WordDic[wid] = 1

if wid not in ClassFeaDic[classid]:

ClassFeaDic[classid][wid] = 1

else:

ClassFeaDic[classid][wid] += 1

i += 1

sline = infile.readline().strip()

infile.close()

print(i, "instances loaded!")

print(len(ClassFreq), "classes!", len(WordDic), "words!")

​

#计算模型

def ComputeModel():

sum = 0.0

for freq in ClassFreq.values():

sum += freq

for classid in ClassFreq.keys():

ClassProb[classid] = (float)(ClassFreq[classid])/(float)(sum)

for classid in ClassFeaDic.keys():

sum = 0.0

for wid in ClassFeaDic[classid].keys():

sum += ClassFeaDic[classid][wid]

newsum = (float)(sum + 1)

for wid in ClassFeaDic[classid].keys():

ClassFeaProb[classid][wid] = (float)(ClassFeaDic[classid][wid]+DefaultFreq)/newsum

ClassDefaultProb[classid] = (float)(DefaultFreq) / newsum

return

​

#保存模型

def SaveModel():

outfile = open(ModelFile, 'w')

for classid in ClassFreq.keys():

outfile.write(str(classid))

outfile.write(' ')

outfile.write(str(ClassProb[classid]))

outfile.write(' ')

outfile.write(str(ClassDefaultProb[classid]))

outfile.write(' ' )

outfile.write('

')

for classid in ClassFeaDic.keys():

for wid in ClassFeaDic[classid].keys():

outfile.write(str(wid)+' '+str(ClassFeaProb[classid][wid]))

outfile.write(' ')

outfile.write('

')

outfile.close()

​

#加载模型

def LoadModel():

global WordDic

WordDic = {}

global ClassFeaProb

ClassFeaProb = {}

global ClassDefaultProb

ClassDefaultProb = {}

global ClassProb

ClassProb = {}

infile = open(ModelFile, 'r')

sline = infile.readline().strip()

items = sline.split(' ')

if len(items) < 6:

print("Model format error!")

return

i = 0

while i < len(items):

classid = int(items[i])

ClassFeaProb[classid] = {}

i += 1

if i >= len(items):

print("Model format error!")

return

ClassProb[classid] = float(items[i])

i += 1

if i >= len(items):

print("Model format error!")

return

ClassDefaultProb[classid] = float(items[i])

i += 1

for classid in ClassProb.keys():

sline = infile.readline().strip()

items = sline.split(' ')

i = 0

while i < len(items):

wid = int(items[i])

if wid not in WordDic:

WordDic[wid] = 1

i += 1

if i >= len(items):

print("Model format error!")

return

ClassFeaProb[classid][wid] = float(items[i])

i += 1

infile.close()

print(len(ClassProb), "classes!", len(WordDic), "words!")

​

#预测类别

def Predict():

global WordDic

global ClassFeaProb

global ClassDefaultProb

global ClassProb

​

TrueLabelList = []

PredLabelList = []

i =0

infile = open(TestDataFile, 'r')

outfile = open(TestOutFile, 'w')

sline = infile.readline().strip()

scoreDic = {}

iline = 0

while len(sline) > 0:

iline += 1

if iline % 10 == 0:

print(iline," lines finished!

")

pos = sline.find("#")

if pos > 0:

sline = sline[:pos].strip()

words = sline.split(' ')

if len(words) < 1:

print("Format error!")

break

classid = int(words[0])

TrueLabelList.append(classid)

words = words[1:]

for classid in ClassProb.keys():

scoreDic[classid] = math.log(ClassProb[classid])

for word in words:

if len(word) < 1:

continue

wid = int(word)

if wid not in WordDic:

continue

for classid in ClassProb.keys():

if wid not in ClassFeaProb[classid]:

scoreDic[classid] += math.log(ClassDefaultProb[classid])

else:

scoreDic[classid] += math.log(ClassFeaProb[classid][wid])

i += 1

maxProb = max(scoreDic.values())

for classid in scoreDic.keys():

if scoreDic[classid] == maxProb:

PredLabelList.append(classid)

sline = infile.readline().strip()

infile.close()

outfile.close()

print(len(PredLabelList),len(TrueLabelList))

return TrueLabelList,PredLabelList

​

#计算准确度

def Evaluate(TrueList, PredList):

accuracy = 0

i = 0

while i < len(TrueList):

if TrueList[i] == PredList[i]:

accuracy += 1

i += 1

accuracy = (float)(accuracy)/(float)(len(TrueList))

print("Accuracy:",accuracy)

​

#计算精确率和召回率

def CalPreRec(TrueList,PredList,classid):

correctNum = 0

allNum = 0

predNum = 0

i = 0

while i < len(TrueList):

if TrueList[i] == classid:

allNum += 1

if PredList[i] == TrueList[i]:

correctNum += 1

if PredList[i] == classid:

predNum += 1

i += 1

return (float)(correctNum)/(float)(predNum),(float)(correctNum)/(float)(allNum)

​

#main framework

if sys.argv[1] == '1':

print("start training:")

LoadData()

ComputeModel()

SaveModel()

elif sys.argv[1] == '0':

print("start testing:")

​

LoadModel()

TList,PList = Predict()

i = 0

outfile = open(TestOutFile, 'w')

while i < len(TList):

outfile.write(str(TList[i]))

outfile.write(' ')

outfile.write(str(PList[i]))

outfile.write('

')

i += 1

outfile.close()

Evaluate(TList,PList)

for classid in ClassProb.keys():

pre,rec = CalPreRec(TList, PList,classid)

print("Precision and recall for Class",classid,":",pre,rec)

else:

print("Usage incorrect!")

​