**朴素贝叶斯文档分类**

|  |
| --- |
| """  Created on 下午5:28 22 03 2017  bayes algorithm: classify a words as good or bad   [text classify]  @author: plushunter  """  from numpy import \*  class Naive\_Bayes:      def \_\_init\_\_(self):          self.\_creteria = "NB"      #创建不重复词集      def \_creatVocabList(self,dataSet):          vocabSet = set([])  # 创建一个空的SET          for document in dataSet:              vocabSet = vocabSet | set(document)  # 并集          return list(vocabSet)  # 返回不重复词表（SET的特性）      #文档词集向量模型      def \_setOfWordToVec(self,vocabList, inputSet):          """          功能:给定一行词向量inputSet，将其映射至词库向量vocabList，出现则标记为1，否则标记为0.          """          returnVec = [0] \* len(vocabList)          for word in inputSet:              if word in vocabList:                  returnVec[vocabList.index(word)] = 1          return returnVec      #文档词袋模型      def \_bagOfsetOfWordToVec(self,vocabList, inputSet):          """          功能：对每行词使用第二种统计策略，统计单个词的个数，然后映射到此库中          输出：一个n维向量，n为词库的长度，每个取值为单词出现的次数          """          returnVec = [0] \* len(vocabList)          for word in inputSet:              if word in vocabList:                  returnVec[vocabList.index(word)] += 1 #更新此处代码          return returnVec      def \_trainNB0(self,trainMatrix, trainCategory):          """          输入：训练词矩阵trainMatrix与类别标签trainCategory,格式为Numpy矩阵格式          功能：计算条件概率p0Vect、p1Vect和类标签概率pAbusive          """          numTrainDocs = len(trainMatrix)#样本个数          numWords = len(trainMatrix[0])#特征个数，此处为词库长度          pAbusive = sum(trainCategory) / float(numTrainDocs)#计算负样本出现概率（先验概率）          p0Num = ones(numWords)#初始词的出现次数为1，以防条件概率为0，影响结果          p1Num = ones(numWords)#同上          p0Denom = 2.0#类标记为2，使用拉普拉斯平滑法,          p1Denom = 2.0          #按类标记进行聚合各个词向量          for i in range(numTrainDocs):              if trainCategory[i] == 0:                  p0Num += trainMatrix[i]                  p0Denom += sum(trainMatrix[i])              else:                  p1Num += trainMatrix[i]                  p1Denom += sum(trainMatrix[i])          p1Vect = log(p1Num / p1Denom)#计算给定类标记下，词库中出现某个单词的概率          p0Vect = log(p0Num / p0Denom)#取log对数，防止条件概率乘积过小而发生下溢          return p0Vect, p1Vect, pAbusive      def \_classifyNB(self,vec2Classify, p0Vec, p1Vec, pClass1):          """          该算法包含四个输入:          vec2Classify表示待分类的样本在词库中的映射集合，          p0Vec表示条件概率P(wi|c=0)P(wi|c=0)，          p1Vec表示条件概率P(wi|c=1)P(wi|c=1)，          pClass1表示类标签为1时的概率P(c=1)P(c=1)。          p1=ln[p(w1|c=1)p(w2|c=1)…p(wn|c=1)p(c=1)]          p0=ln[p(w1|c=0)p(w2|c=0)…p(wn|c=0)p(c=0)]          log取对数为防止向下溢出          功能:使用朴素贝叶斯进行分类,返回结果为0/1          """          p1 = sum(vec2Classify \* p1Vec) + log(pClass1)          p0 = sum(vec2Classify \* p0Vec) + log(1 - pClass1)          if p1 > p0:              return 1          else:              return 0      #test      def testingNB(self,testSample):          "step1：加载数据集与类标号"          listOPosts, listClasses = loadDataSet()          "step2：创建词库"          vocabList = self.\_creatVocabList(listOPosts)          "step3：计算每个样本在词库中出现的情况"          trainMat = []          for postinDoc in listOPosts:              trainMat.append(self.\_bagOfsetOfWordToVec(vocabList, postinDoc))          p0V, p1V, pAb = self.\_trainNB0(trainMat, listClasses)          "step4：测试"          thisDoc = array(self.\_bagOfsetOfWordToVec(vocabList, testSample))          result=self.\_classifyNB(thisDoc, p0V, p1V, pAb)          print testSample, 'classified as:', result          # return result  ###  # 加载数据集  def loadDataSet():      postingList = [['my', 'dog', 'has', 'flea', 'problems', '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', 'my', 'steak', 'how', 'to', 'stop', 'him'],                     ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]      classVec = [0, 1, 0, 1, 0, 1]  # 1 is abusive, 0 not      return postingList, classVec  #测试  if \_\_name\_\_=="\_\_main\_\_":      clf = Naive\_Bayes()      testEntry = [['love', 'my', 'girl', 'friend'],                   ['stupid', 'garbage'],                   ['Haha', 'I', 'really', "Love", "You"],                   ['This', 'is', "my", "dog"],                   ['maybe','stupid','worthless']]      for item in testEntry:          clf.testingNB(item) |

### 使用朴素贝叶斯过滤垃圾邮件

|  |
| --- |
| """  Created on 下午8:47 22 03 2017  Email\_Classify  @author: plushunter  """  import re  import Bayes  from numpy import \*  # mysent='This book is the best book on Python or M.L I have ever laid eyes upon.'  # regEx = re.compile('\\W\*')  # listOfTokens=regEx.split(mysent)  # tok=[tok.upper() for tok in listOfTokens if len(tok)>0]  # print tok  #  # emailText=open('email/ham/6.txt').read()  # listOfTokens=regEx.split(emailText)  # print listOfTokens  def textParse(bigString):      import re      listOfTokens=re.split(r'\w\*',bigString)      return [tok.lower() for tok in listOfTokens if len(tok)>2]  def spamTest():      clf = Bayes.Naive\_Bayes()      docList=[]      classList=[]      fullText=[]      for i in range(1,26):          wordList=textParse(open('email/spam/%d.txt'%i).read())          docList.append(wordList)          fullText.extend(wordList)          classList.append(1)          wordList=textParse(open('email/ham/%i.txt'%i).read())          docList.append(wordList)          fullText.extend(wordList)          classList.append(0)      vocabList=clf.\_creatVocabList(docList)      trainingSet=range(50);testSet=[]      for i in range(10):          randIndex=int(random.uniform(0,len(trainingSet)))          testSet.append(trainingSet[randIndex])          del(trainingSet[randIndex])      trainMatix=[];trainClasses=[]      for docIndex in trainingSet:          trainMatix.append(clf.\_bagOfsetOfWordToVec(vocabList,docList[docIndex]))          trainClasses.append(classList[docIndex])      p0V,p1V,pSpam=clf.\_trainNB0(array(trainMatix),array(trainClasses))      errorCount = 0      for docIndex in testSet:          wordVector = clf.\_bagOfsetOfWordToVec(vocabList,docList[docIndex])          if clf.\_classifyNB(array(wordVector), p0V, p1V, pSpam)!=classList[docIndex]:              errorCount+=1      print 'the error rate is :',float(errorCount)/len(testSet) |