基于关联算法分析用户用户上网行为特征报告

一.前言

由于用户上网行为存在浏览新闻网页，观看视频，下载音乐及聊天等行为特征，通过分析用户上网行为中电脑数据包，通过获取本地IP传输信息，可简单分析出用户上行为特征，即经常使用的某些应用所访问的特定一类网页，得出该用户平时的上网特征，本报告主要简单将其安装数据量传输的多少归类为观看视频，浏览网页，聊天通信三大类型，因为网络中各种数据传输的复杂性，只对其获取的主要数据信息进行分析，以便得到比较好的数据分析结果。进而为后续分析该用户上级节点路由器中传输的IP的种类，选择适当的传输路径。

1. 研究主要内容

运用关联规则的挖掘算法，使用到支持度、置信度、相关度这三个参数，对经常访问的网站的IP进行分析，得出用户经常访问IP集合。

三.问题可行性研究

1.获取本地网络包可行性

通过查阅相关资料，和各种本，硕士关于网络优化论文及阅读《IP/TCP协议》书得知，在Linux平台下，可以利用python的相关包获取本机上网信息包，通过python编程提取数据以特定格式以文本保存到本地，进而为后续进行关联分析用户上网时数据做预处理。

2.关联分析与数据挖掘的发展状况：

数据挖掘与知识发现是人工智能、机器学习与数据库技术相结合的产物。机器学习(Machine Learning)是用计算机模拟人类学习的一门科学，始于60年代末，真正的发展是在70年代末。由于在专家系统开发中存在知识获取的“瓶颈”现象，所以就用机器学习来完成知识的自动获取。1980年，在美国召开了第一届国际机器学习研讨会;1984年《机器学习》杂志问世。我国很快跟上了国际步伐，于1987年召开了第一届全国机器学习研讨会;1989年成立了以中国科技大学蔡庆生教授为理事长的理事会;1995年在国防科技大学召开了第五届机器学习研讨会。

3.关联分析算法描述：关联分析，即从一个数据集中发现项之间的隐藏关系。本篇文章Apriori算法主要是基于频繁集的关联分析，所以本文中所出现的关联分析默认都是指基于频繁集的关联分析。有了一个感性的认识，我们来一段理性的形式化描述：

令项集I={i1,i2,...in}且有一个数据集合D，它其中的每一条记录T，都是I的子集那么关联规则都是形如A->B的表达式，A、B均为I的子集，且A与B的交集为空这条关联规则的支持度：support = P(A并B)

这条关联规则的置信度：confidence = support(A并B）/suport(A)

如果存在一条关联规则，它的支持度和置信度都大于预先定义好的最小支持度与置信度，我们就称它为强关联规则。强关联规则就可以用来了解项之间的隐藏关系。所以关联分析的主要目的就是为了寻找强关联规则，而Apriori算法则主要用来帮助寻找强关联规则。

注意：因为频率=事件出现次数/总事件次数，为了方便，我们在以下都用事件出现的频数而非频率来作为支持度。

1. 关联分析Apriori算法介绍：

算法主要利用了如下性质：如果一个项集是频繁的，则它的所有子集一定也是频繁的（这个性质也称支持度度量的反单调性）。也就是说如果当前的项集不是频繁的，那么它的超集也不在是频繁的。（该算法的计算复杂度依赖于数据中的项数和事物的平均长度等性质）

算法步骤：（1）算法初始通过单遍扫描数据集，确定每个项的支持度。一旦完成这一步，就得到所有频繁1项集的集合F1；（2）接下来，该算法使用上一次迭代发现的频繁(k-1)项集，产生新的候选k项集；（3）为了对候选项的支持度计数，算法需要再次扫描一遍数据库，使用子集函数确定包含在每一个事物t中的Ck中的所有候选k项集；（4）计算候选项的支持度计数后，算法将删除支持度计数小于minsup的所有候选项集；（5）当没有新的频繁项集产生时，算法结束。

Apriori算法第一它是逐层算法，第二它使用产生—测试策略来发现频繁项集。注意：在由k-1项集产生k项集的过程中有以下几点注意：

（1）新产生的k项集先要确定它的所有的k-1项真子集都是频繁的（其实如果k个子集中的m个用来产生候选项集，则在候选项集剪枝时只需检查剩下的k-m个子集），如果有一个不是频繁的，那么它可以从当前的候选项集中去掉。

（2）候选项集的产生方法：A)蛮力法：从2项集开始以后所有的项集都从1项集完全拼出来。如：3项集有3个一项集拼出（要列出所有的3个一项集拼出的可能）。然后再按照剪枝算法剪枝（确定当前的项集的所有k-1项集是否都是频繁的）。；B）Fk-1\*F1法：顾名思义由1项集和k-1项集生成k项集，然后再剪枝。（这种方法是完全的，因为每一个k频繁项集都是由一个频繁k-1项集和一个频繁1项集产生的）但是，这种方法会产生大量重复的k频繁项集（由于顺序的关系）。避免产生重复的候选项集的一种方法是确保每个频繁项集中的项以字典序存储，每个频繁k-1项集X只用字典序比X中的所有项都大的频繁项进行扩展。（还可以使用一些启发式方法进一步减少候选项集的数量，如：对于每一个幸免于剪枝的候选k项集，它的每一个项必须至少在k-1个k-1项集中出现）；C）：Fk-1\*Fk-1法：由两个频繁k-1项集生成候选k项集,但是两个频繁k-1项集的前k-2项必须相同，最后一项必须相异。（由于每个候选是由一对频繁k-1项集合并而成，所以需要附加的候选剪枝步骤来确保该候选的其余k-2个子集是频繁的）。

注意：（1）在支持度计数时除了传统的方法外还可以将候选项集划分为不同的桶，并存放在Hash树中。在支持度计数期间，包含在事物中的项集也散列到相应的桶中。（2）随着频繁项集最大长度的增加，算法需要扫描数据库的次数也将增多。

5.关联分析基于本应用算法介绍

通过研读周志华《机器学习》，和《机器学习实战》中关联分析算法实现。本论文将分析本地一百次数据文档，因为每次文档中都有相关的IP信息，可以得出相关的频繁数据项集，不频繁项集将自动剔除，减少计算复杂度。由于用户每次的上网时都会经常访问到某些IP，计算出每一个IP 的数据支持度，然后求出IP的关联度表。即可以将某些特定的IP归为一类，简单概括为某一个应用。

四.python代码实现

本例子主要函数如下：

def loadDataSet():

#return dataMat，ipCountMap，ipCountTime，ipMap

本函数将对从本机中加载的ip数据报通过dict类型进行相应映射数值化，挖掘出数据集对应的关系。

def createC1(dataSet):

#return use frozen set so we can use it as a key in a dict .

def scanD(D, Ck, minSupport):

#return retList， supportData

通过设置的最小支持度，筛选出support > minsupport值的频繁项集和相应的support。

def aprioriGen(Lk, k):

#return L，supportData

运用关联分析基类算法，快速将一组k-2元集合，和并成k元集合。

def apriori(dataSet, minSupport = 0.5):

#return L,supportData

通过调用aprioriGen（）方法，求出一组1元集合所形成的2的n次方集合项的支持度。（频繁集 < minsupport 和其超类自动剔除以减少计算量）

def generateRules(L, supportData, minConf=0.7):

#supportData is a dict coming from scanD

#L is a List coming from apriori

根据：关联度=支持度1/支持度2，求出每组关联度值。

def calcConf(freqSet, H, supportData, brl, minConf=0.7):

#将每两组集和的关联度结果放在brl中。

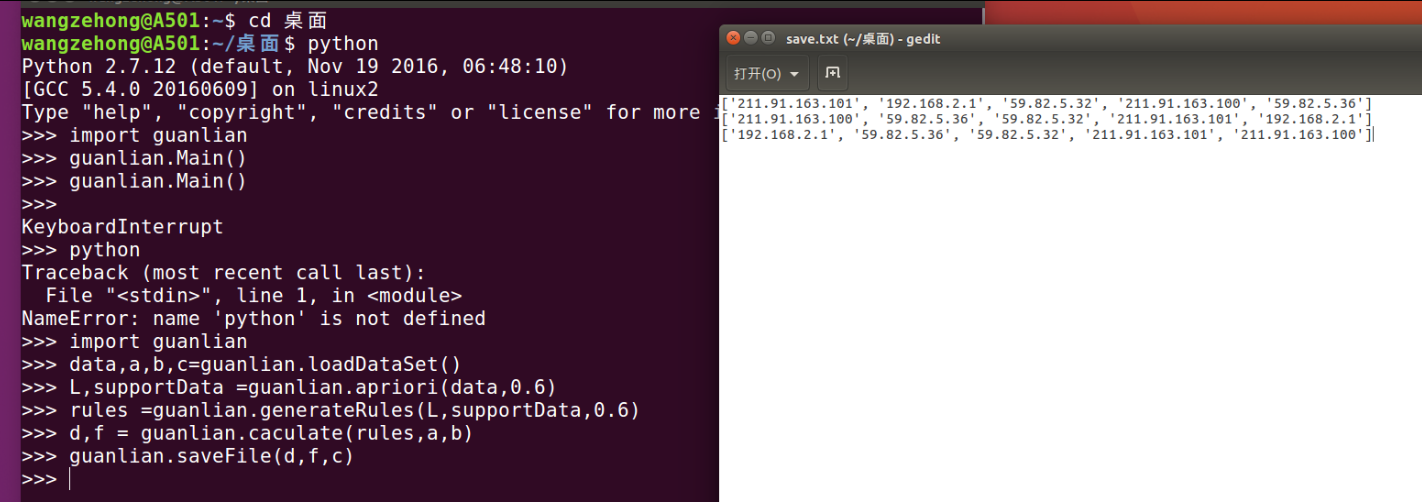
def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):

#将H中数据项非两组其合并成两组，便于求其关联度。

def caculate(bigRuleList,ipCountMap,ipCountTime,setNumber = 3):

#按照平均访问数据量多少排序，将数据从大到小排列，选出setNumber个集合，并将对应IP保存到本地文件。

1. 运行结果



1. 参考文献

《TCP/IP详解卷1：协议》是2000年由[机械工业出版社](http://baike.baidu.com/view/156089.htm" \t "http://baike.baidu.com/_blank)出版的图书，作者是LawrenceBerkeley。

《机器学习.pdf》周志华，机器工业出版社。

《python基础教程第2版》 Magnus Lie Hetland 著

《机器学习实战》是2013年由人民邮电出版社出版的书籍，作者是Peter Harrington。

1. 代码附录

'''

Created on Dec 19, 2016

@author: Ulysses

'''

from numpy import \*

def Main():

data,a,b,c=loadDataSet()

L,supportData =apriori(data,0.6)

rules = generateRules(L,supportData,0.6)

d,f = caculate(rules,a,b)

saveFile(d,f,c)

#print supportData

def loadDataSet():

dataMat = []; timeSet = set(); ipSet = set(); ipMap = {}; lineData = [];ipCountMap = {};ipCountTime = {}

count = 0

file = open('test.txt')

for line in file.readlines():

lineArr = line.strip().split()

if int(lineArr[1]) < 100:

continue

if lineArr[2] not in timeSet:

if len(lineData) > 0:

dataMat.append(lineData)

lineData = []

timeSet.add(lineArr[2])

if lineArr[0] not in ipSet:

ipSet.add(lineArr[0])

ipMap[lineArr[0]] = count

ipCountMap[count] = int(lineArr[1])

ipCountTime[count] = 1

count += 1

else:

ipCountMap[ipMap[lineArr[0]]] += int(lineArr[1])

ipCountTime[ipMap[lineArr[0]]] += 1

lineData.append(ipMap[lineArr[0]])

if len(lineData) > 0:

dataMat.append(lineData)

return dataMat,ipCountMap,ipCountTime,ipMap

#,timeSet,ipSet

#return

def createC1(dataSet):

C1 = []

for transaction in dataSet:

for item in transaction:

if not [item] in C1:

C1.append([item])

C1.sort()

return map(frozenset, C1)#use frozen set so we

#can use it as a key in a dict

def scanD(D, Ck, minSupport):

ssCnt = {}

for tid in D:

for can in Ck:

if can.issubset(tid):

if not ssCnt.has\_key(can): ssCnt[can]=1

else: ssCnt[can] += 1

numItems = float(len(D))

retList = []

supportData = {}

for key in ssCnt:

support = ssCnt[key]/numItems

if support >= minSupport:

retList.insert(0,key)

supportData[key] = support

return retList, supportData

def aprioriGen(Lk, k): #creates Ck

retList = []

lenLk = len(Lk)

for i in range(lenLk):

for j in range(i+1, lenLk):

L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]

L1.sort(); L2.sort()

if L1==L2: #if first k-2 elements are equal

retList.append(Lk[i] | Lk[j]) #set union

return retList

def apriori(dataSet, minSupport = 0.5):

C1 = createC1(dataSet)

D = map(set, dataSet)

L1, supportData = scanD(D, C1, minSupport)

L = [L1]

k = 2

while (len(L[k-2]) > 0):

Ck = aprioriGen(L[k-2], k)

Lk, supK = scanD(D, Ck, minSupport)#scan DB to get Lk

supportData.update(supK)

L.append(Lk)

k += 1

return L, supportData

def generateRules(L, supportData, minConf=0.7): #supportData is a dict coming from scanD

bigRuleList = []

for i in range(1, len(L)):#only get the sets with two or more items

for freqSet in L[i]:

H1 = [frozenset([item]) for item in freqSet]

if (i > 1):

rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)

else:

calcConf(freqSet, H1, supportData, bigRuleList, minConf)

return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):

prunedH = [] #create new list to return

for conseq in H:

conf = supportData[freqSet]/supportData[freqSet-conseq] #calc confidence

if conf >= minConf:

#print freqSet-conseq,'-->',conseq,'conf:',conf

brl.append((freqSet-conseq, conseq, conf))

prunedH.append(conseq)

return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):

m = len(H[0])

if (len(freqSet) > (m + 1)): #try further merging

Hmp1 = aprioriGen(H, m+1)#create Hm+1 new candidatesa

Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)

if (len(Hmp1) > 1): #need at least two sets to merge

rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)

def caculate(bigRuleList,ipCountMap,ipCountTime,setNumber = 3):

meanNumber = {}; a = 0.0;maxkey = -1; judge = True; time = []; line = [];dataMat = []

for item in ipCountMap.keys():

meanNumber[item] = float(ipCountMap[item] \* 1.0 / ipCountTime[item])

#print meanNumber

while (len(time) != setNumber):

if len(meanNumber) == 0:

break

a = 0.0;judge = 1;maxkey = -1;line = []

for item in meanNumber.keys():

if meanNumber[item] > a:

a = meanNumber[item]

maxkey = item

for x in bigRuleList:

if frozenset([maxkey]) == x[0]:

judge = False

line.append(x[1])

#print 'x[1]',x[0],'--->',x[1]

del meanNumber[maxkey]

if frozenset([maxkey]) in dataMat or judge:

continue

if len(line) > 0:

dataMat.append(line)

time.append(maxkey)

return time, dataMat

def saveFile(time, dataMat, ipMap):

dataSet = [];dataList = []

for lineData in dataMat:

setLine = set()

for setData in lineData:

setLine.update(set(setData))

dataSet.append(setLine.copy())

file=open("save.txt","w")

index = 0

for lineData in dataSet:

dataList.append(findIndex(ipMap,time[index]))

for data in lineData:

dataList.append(findIndex(ipMap,data))

file.writelines(str(dataList)+'\n')

dataList = []

index += 1

def findIndex(Map,value):

for keys in Map.keys():

if value == Map[keys]:

return keys