机器学习学习报告

周珊琳

上海电力大学

上海, 2019-06-28, 中国

第一节　算法概述

算法名称：FP-tree

算法原理：FP-growth算法不同于Apriori算法生成候选项集再检验是否频繁的“产生-测试”方法，而是使用一种称为频繁模式树（FP-tree）的紧凑数据结构组织数据，并直接从该结构中提取频繁项集。FP-树是一种输入数据的压缩表示，通过逐个读入事务，并把每个事务映射到FP-树中的一条路径来构造。由于不同的事务可能会有若干个相同的项，因此它们的路径可能部分重叠。路径相互重叠越多，使用FP-tree结构获得的压缩效果越好。如果FP-tree足够小，能够存放在内存中，就可以直接从这个内存中的结构提取频繁项集，而不必重复地扫描放在硬盘上的数据，从而提高处理效率。

算法中所用的到数学公式

支持度公式：

$$Support(A \to B) = support_count(A \bigcup B)/N \tag{1.1}$$

置信度公式：

$$confidence(A \to B) = support_count(A \bigcup B)/support_count(A) \tag{1.2}$$

第二节　算法设计

2.1　算法流程

Algorithm 1 FP-growth 算法

```
收集数据
准备数据：由于存储的是集合，需要离散数据，即连续型数据需要离散化
分析数据
训练算法：构建一个FP树，对其进行挖掘（抽取频繁项集)
使用算法
```

Algorithm 2 FP-growth 抽取频繁项集的步骤:

```
从FP中获取条件模式基
利用条件模式基，构建一个条件FP树
迭代重复前两个步骤，直到树包含一个元素为止
```

2.2　核心代码

源代码 1:

```

class treeNode:
    def __init__(self, nameValue, numOccur, parentNode):
        self.name =nameValue
        self.count =numOccur
        self.nodeLink =None
        self.parent =parentNode    #needs to be updated
        self.children ={}

    def inc(self, numOccur):
        self.count +=numOccur

    def disp(self, ind=1):
        print( ’ ’*ind, self.name, ’ ’, self.count)
        for child in self.children.values():
            child.disp(ind+1)

def createTree(dataSet, minSup=1): #create FP-tree from dataset but don’t mine
    headerTable ={}
    #go over dataSet twice
    for trans in dataSet:#first pass counts frequency of occurance
        for item in trans:
            headerTable[item] =headerTable.get(item, 0) +dataSet[trans]
    for k in headerTable.keys(): #remove items not meeting minSup
        if headerTable[k] <minSup:
            del(headerTable[k])
    freqItemSet =set(headerTable.keys())
    #print ’freqItemSet: ’,freqItemSet
    if len(freqItemSet) ==0: return None, None #if no items meet min support -->get out
    for k in headerTable:
        headerTable[k] =[headerTable[k], None] #reformat headerTable to use Node link
    #print ’headerTable: ’,headerTable
    retTree =treeNode(’Null Set’, 1, None) #create tree
    for tranSet, count in dataSet.items(): #go through dataset 2nd time
        localD ={}
        for item in tranSet: #put transaction items in order
            if item in freqItemSet:
                localD[item] =headerTable[item][0]
        if len(localD) >0:
            orderedItems =[v[0] for v in sorted(localD.items(), key=lambda p: p[1], reverse=True)]
            updateTree(orderedItems, retTree, headerTable, count)#populate tree with ordered freq itemset
    return retTree, headerTable #return tree and header table

def updateTree(items, inTree, headerTable, count):
    if items[0] in inTree.children:#check if orderedItems[0] in retTree.children
        inTree.children[items[0]].inc(count) #incrament count
    else:  #add items[0] to inTree.children
        inTree.children[items[0]] =treeNode(items[0], count, inTree)
        if headerTable[items[0]][1] ==None: #update header table
            headerTable[items[0]][1] =inTree.children[items[0]]
        else:
            updateHeader(headerTable[items[0]][1], inTree.children[items[0]])
    if len(items) >1:#call updateTree() with remaining ordered items
        updateTree(items[1::], inTree.children[items[0]], headerTable, count)

def updateHeader(nodeToTest, targetNode): #this version does not use recursion
    while (nodeToTest.nodeLink !=None): #Do not use recursion to traverse a linked list!
        nodeToTest =nodeToTest.nodeLink
    nodeToTest.nodeLink =targetNode

def ascendTree(leafNode, prefixPath): #ascends from leaf node to root
    if leafNode.parent !=None:
        prefixPath.append(leafNode.name)
        ascendTree(leafNode.parent, prefixPath)

def findPrefixPath(basePat, treeNode): #treeNode comes from header table
    condPats ={}
    while treeNode !=None:
        prefixPath =[]
        ascendTree(treeNode, prefixPath)
        if len(prefixPath) >1:
            condPats[frozenset(prefixPath[1:])] =treeNode.count
        treeNode =treeNode.nodeLink
    return condPats


def mineTree(inTree, headerTable, minSup, preFix, freqItemList):
    bigL =[v[0] for v in sorted(headerTable.items(), key=lambda p: p[1])]#(sort header table)
    for basePat in bigL: #start from bottom of header table
        newFreqSet =preFix.copy()
        newFreqSet.add(basePat)
        #print ’finalFrequent Item: ’,newFreqSet #append to set
        freqItemList.append(newFreqSet)
        condPattBases =findPrefixPath(basePat, headerTable[basePat][1])
        #print ’condPattBases :’,basePat, condPattBases
        #2. construct cond FP-tree from cond. pattern base
        myCondTree, myHead =createTree(condPattBases, minSup)
        #print ’head from conditional tree: ’, myHead
        if myHead !=None: #3. mine cond. FP-tree
            #print ’conditional tree for: ’,newFreqSet
            #myCondTree.disp(1)
            mineTree(myCondTree, myHead, minSup, newFreqSet, freqItemList)

def loadSimpDat():
    simpDat =[[’r’, ’z’, ’h’, ’j’, ’p’],
              [’z’, ’y’, ’x’, ’w’, ’v’, ’u’, ’t’, ’s’],
              [’z’],
              [’r’, ’x’, ’n’, ’o’, ’s’],
              [’y’, ’r’, ’x’, ’z’, ’q’, ’t’, ’p’],
              [’y’, ’z’, ’x’, ’e’, ’q’, ’s’, ’t’, ’m’]]
    return simpDat

def createInitSet(dataSet):
    retDict ={}
    for trans in dataSet:
        retDict[frozenset(trans)] =1
    return retDict

import twitter
from time import sleep
import re

def textParse(bigString):
    urlsRemoved =re.sub(’(http:[/][/]|www.)([a-z]|[A-Z]|[0-9]|[/.]|[~])*’, ’’, bigString)
    listOfTokens =re.split(r’\W*’, urlsRemoved)
    return [tok.lower() for tok in listOfTokens if len(tok) >2]

def getLotsOfTweets(searchStr):
    CONSUMER_KEY =’’
    CONSUMER_SECRET =’’
    ACCESS_TOKEN_KEY =’’
    ACCESS_TOKEN_SECRET =’’
    api = twitter.Api(consumer_key=CONSUMER_KEY, consumer_secret=CONSUMER_SECRET,
                      access_token_key=ACCESS_TOKEN_KEY,
                      access_token_secret=ACCESS_TOKEN_SECRET)
    #you can get 1500 results 15 pages * 100 per page
    resultsPages =[]
    for i in range(1,15):
        print( "fetching page %d" % i)
        searchResults =api.GetSearch(searchStr, per_page=100, page=i)
        resultsPages.append(searchResults)
        sleep(6)
    return resultsPages

```

```
def mineTweets(tweetArr, minSup=5):
    parsedList =[]
    for i in range(14):
        for j in range(100):
            parsedList.append(textParse(tweetArr[i][j].text))
    initSet =createInitSet(parsedList)
    myFPtree, myHeaderTab =createTree(initSet, minSup)
    myFreqList =[]
    mineTree(myFPtree, myHeaderTab, minSup, set([]), myFreqList)
    return myFreqList

from numpy import *
import numpy as np
from sklearn.datasets import load_iris,load_wine
import pandas as pd

#iris
#data=load_iris()
#dataSet = data.data
#labels = data.feature_names
#target = data.target
#r=[]
#for i in target:
#    if i==0:
#        i=’Iris-setosa’
#    elif i==1:
#        i=’Iris-versicolor’
#    else:
#        i=’Iris-virginica’
#    r.append(i)
#simpDat = []
#for i in range(4):
#    tmp=pd.cut(dataSet[:,1],3,labels=[’a’+str(i),’b’+str(i),’c’+str(i)])
#    simpDat.append(tmp)
#simpDat.append(r)
#simpDat = np.array(list(zip(*simpDat)))

#wine
#data=load_wine()
#dataSet = data.data
#r = data.target
#simpDat = []
#for i in range(13):
#    tmp=pd.cut(dataSet[:,1],3,labels=[’a’+str(i),’b’+str(i),’c’+str(i)])
#    simpDat.append(tmp)
#simpDat.append(r)
#simpDat = np.array(list(zip(*simpDat)))

def loadDataSet(fileName):
    dataMat =[]
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =line.strip().split(’,’)
        # print(lineArr)
        if lineArr[0] ==’vhigh’:
            lineArr[0] =’a1’
        if lineArr[0] ==’high’:
            lineArr[0] =’a2’
        if lineArr[0] ==’med’:
            lineArr[0] =’a3’
        if lineArr[0] ==’low’:
            lineArr[0] =’a4’

        if lineArr[1] ==’vhigh’:
            lineArr[1] =’b1’
        if lineArr[1] ==’high’:
            lineArr[1] =’b2’
        if lineArr[1] ==’med’:
            lineArr[1] =’b3’
        if lineArr[1] ==’low’:
            lineArr[1] =’b4’

        if lineArr[2]==’2’:
            lineArr[2]=’c1’
        if lineArr[2]==’3’:
            lineArr[2]=’c2’
        if lineArr[2]==’4’:
            lineArr[2]=’c3’
        if lineArr[2]==’5more’:
            lineArr[2]=’c4’

        if lineArr[3]==’2’:
            lineArr[3]=’d1’
        if lineArr[3]==’4’:
            lineArr[3]=’d2’
        if lineArr[3]==’more’:
            lineArr[3]=’d3’

        if lineArr[4]==’small’:
            lineArr[4]=’e1’
        if lineArr[4]==’med’:
            lineArr[4]=’e2’
        if lineArr[4]==’big’:
            lineArr[4]=’e3’

        if lineArr[5]==’low’:
            lineArr[5]=’f1’
        if lineArr[5]==’med’:
            lineArr[5]=’f2’
        if lineArr[5]==’high’:
            lineArr[5]=’f3’

        dataMat.append(lineArr)

    return dataMat
#car
simpDat =loadDataSet(’car.data’)

```

```

#initSet = createInitSet(simpDat)
#print(initSet)
#myFPtree, myHeaderTab = createTree(initSet, minSup)
#myFPtree.disp()
#myFreqList = []
#mineTree(myFPtree, myHeaderTab, minSup, set([]), myFreqList)
#print(myFreqList)

minSup=300
#simpDat = loadSimpDat()
#print(simpDat)
initSet =createInitSet(simpDat)
#print(initSet)
myFPtree, myHeaderTab =createTree(initSet, minSup)
myFPtree.disp()
myFreqList =[]
mineTree(myFPtree, myHeaderTab, minSup, set([]), myFreqList)
print(myFreqList)

#parsedDat = [line.split() for line in open('kosarak.dat').readlines()]
#initSet = createInitSet(parsedDat)
#myFPtree, myHeaderTab = createTree(initSet, 100000)
#myFPtree.disp()
#myFreqList = []
#mineTree(myFPtree, myHeaderTab, 100000, set([]), myFreqList)
#print(myFreqList)
```

第三节　选用数据

iris行数：150 列数：5

列属性及取值:

1)萼片长度cm，数值型

2)萼片宽度cm，数值型

3)花瓣长度cm，数值型

4)花瓣宽度cm数值型

类别：

Iris Setosa

Iris Versicolour

Iris Virginica

car，行数：1728，列数：6

列属性及取值：

1)buying: vhigh, high, med, low.

2)maint: vhigh, high, med, low.

3)doors: 2, 3, 4, 5more.

4)persons: 2, 4, more.

5)lugboot: small, med, big.

6)safety: low, med, high.

类别：

unacc, acc, good, vgood

wine，行数：178，列数：13

属性：

1) Alcohol

2) Malic acid

3) Ash

4) Alcalinity of ash

5) Magnesium

6) Total phenols

7) Flavanoids

8) Nonflavanoid phenols

9) Proanthocyanins

10) Color intensity

11) Hue

12) OD280/OD315 of diluted wines

13) Proline

类别：

Alcohol 1，2，3

news新闻网站点击流：

列属性：

新闻报道编号

第四节 实验结果截图

第五节 实验分析和比较

对于iris，wine来说使用该算法的意义不大，只能找出哪类超过了3+；相对于前两个数据集car本身是离散型的数据，所以结果还行，找出来了哪类车超过了500+。因为car原来的不同属性列直接有相同的取值，因此需要事先进行处理，而iris和wine因为是连续型数据，因此需要提前离散化处理。后选择了一个比较适合这个算法的新闻网站点击流，找出有哪些新闻报道曾被10万+人浏览过。

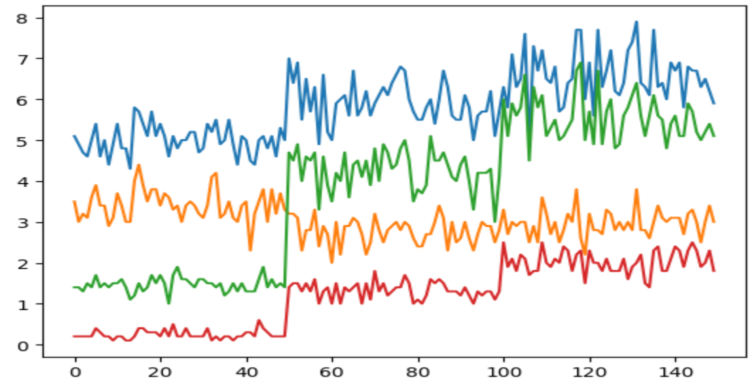


图 1: iris数据展示

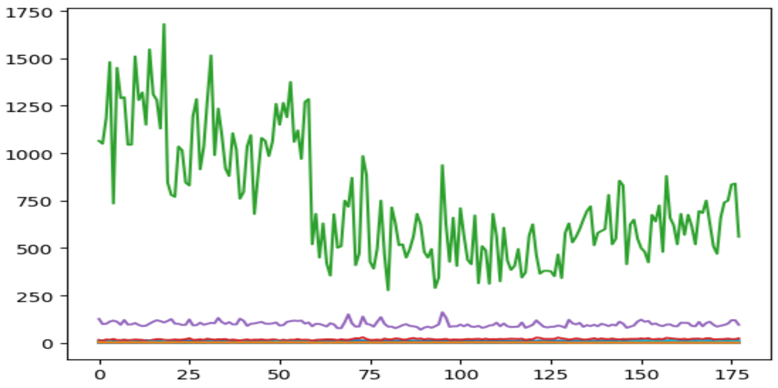
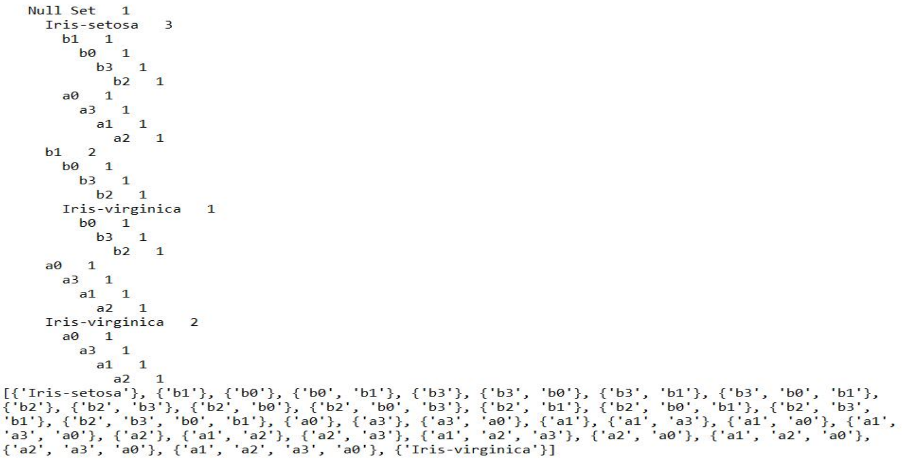
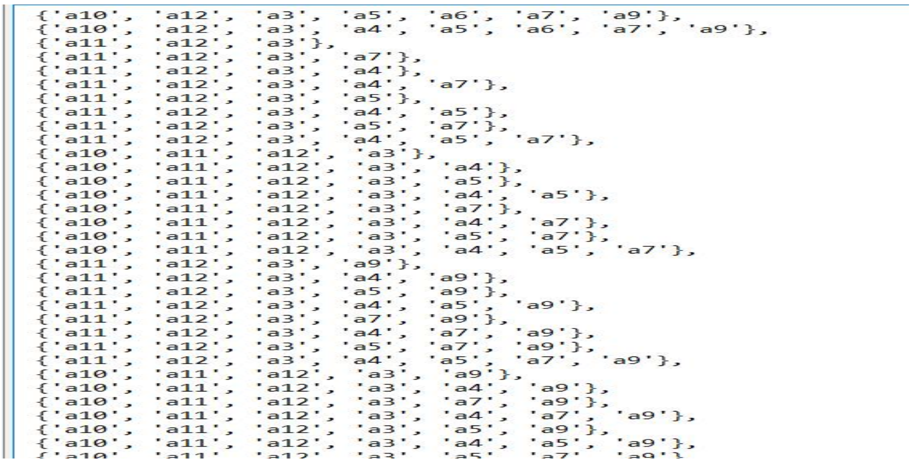


图 2: wine数据展示



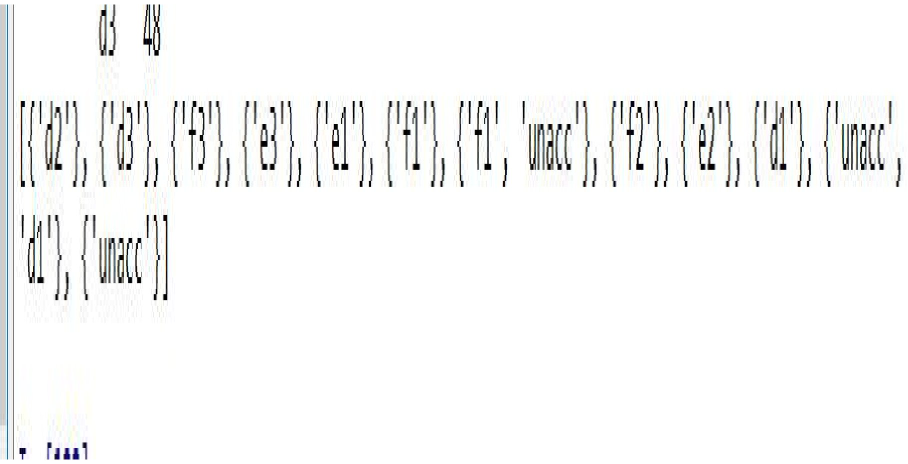
```
    Null Set   1
      Iris-setosa   3
        b1   1
          b0   1
            b3   1
              b2   1
        a0   1
          a3   1
            a1   1
              a2   1
      b1   2
        b0   1
          b3   1
            b2   1
        Iris-virginica   1
          b0   1
            b3   1
              b2   1
      a0   1
        a3   1
          a1   1
            a2   1
      Iris-virginica   2
        a0   1
          a3   1
            a1   1
              a2   1
[{'Iris-setosa'}, {'b1'}, {'b0'}, {'b0', 'b1'}, {'b3'}, {'b3', 'b0'}, {'b3', 'b1'}, {'b3', 'b0', 'b1'},
{'b2'}, {'b2', 'b3'}, {'b2', 'b0'}, {'b2', 'b0', 'b3'}, {'b2', 'b1'}, {'b2', 'b0', 'b1'}, {'b2', 'b3',
'b1'}, {'b2', 'b3', 'b0', 'b1'}, {'a0'}, {'a3'}, {'a3', 'a0'}, {'a1'}, {'a1', 'a3'}, {'a1', 'a0'}, {'a1',
'a3', 'a0'}, {'a2'}, {'a1', 'a2'}, {'a2', 'a3'}, {'a1', 'a2', 'a3'}, {'a2', 'a0'}, {'a1', 'a2', 'a0'},
{'a2', 'a3', 'a0'}, {'a1', 'a2', 'a3', 'a0'}, {'Iris-virginica'}]
```

图 3: iris运行结果



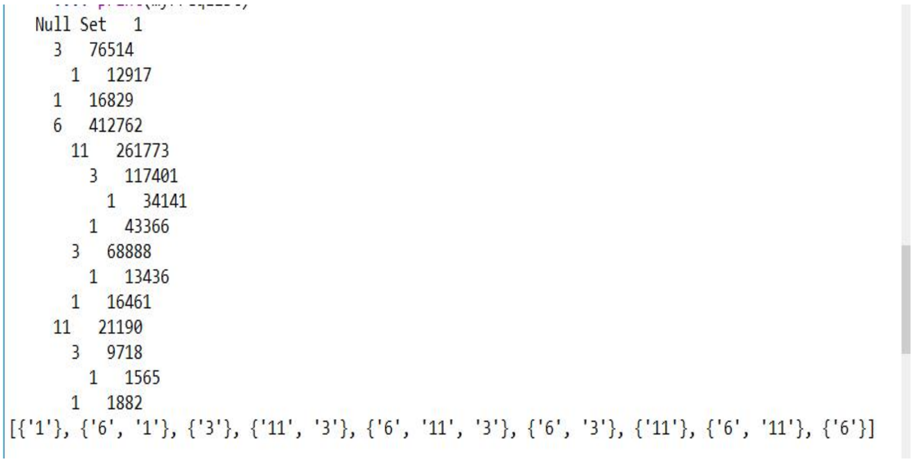
```
 {'a10',  'a12',  'a3',  'a5',  'a6',  'a7',  'a9'},
 {'a10',  'a12',  'a3',  'a4',  'a5',  'a6',  'a7',  'a9'},
 {'a11',  'a12',  'a3'},
 {'a11',  'a12',  'a3',  'a7'},
 {'a11',  'a12',  'a3',  'a4'},
 {'a11',  'a12',  'a3',  'a4',  'a7'},
 {'a11',  'a12',  'a3',  'a5'},
 {'a11',  'a12',  'a3',  'a4',  'a5'},
 {'a11',  'a12',  'a3',  'a5',  'a7'},
 {'a11',  'a12',  'a3',  'a4',  'a5',  'a7'},
 {'a10',  'a11',  'a12',  'a3'},
 {'a10',  'a11',  'a12',  'a3',  'a4'},
 {'a10',  'a11',  'a12',  'a3',  'a5'},
 {'a10',  'a11',  'a12',  'a3',  'a4',  'a5'},
 {'a10',  'a11',  'a12',  'a3',  'a7'},
 {'a10',  'a11',  'a12',  'a3',  'a4',  'a7'},
 {'a10',  'a11',  'a12',  'a3',  'a5',  'a7'},
 {'a10',  'a11',  'a12',  'a3',  'a4',  'a5',  'a7'},
 {'a11',  'a12',  'a3',  'a9'},
 {'a11',  'a12',  'a3',  'a4',  'a9'},
 {'a11',  'a12',  'a3',  'a5',  'a9'},
 {'a11',  'a12',  'a3',  'a4',  'a5',  'a9'},
 {'a11',  'a12',  'a3',  'a7',  'a9'},
 {'a11',  'a12',  'a3',  'a4',  'a7',  'a9'},
 {'a11',  'a12',  'a3',  'a5',  'a7',  'a9'},
 {'a11',  'a12',  'a3',  'a4',  'a5',  'a7',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a4',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a7',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a4',  'a7',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a5',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a4',  'a5',  'a9'},
 {'a10',  'a11',  'a12',  'a3',  'a5',  'a7',  'a9'}
```

图 4: wine运行结果



```
        d3   48
[{'d2'}, {'d3'}, {'f3'}, {'e3'}, {'e1'}, {'f1'}, {'f1', 'unacc'}, {'f2'}, {'e2'}, {'d1'}, {'unacc',
'd1'}, {'unacc'}]
```

图 5: car运行结果



```
 Null Set   1
   3   76514
     1   12917
   1   16829
   6   412762
     11   261773
       3   117401
         1   34141
       1   43366
     3   68888
       1   13436
     1   16461
   11   21190
     3   9718
       1   1565
     1   1882
[{'1'}, {'6', '1'}, {'3'}, {'11', '3'}, {'6', '11', '3'}, {'6', '3'}, {'11'}, {'6', '11'}, {'6'}]
```

图 6: news运行结果

第六节　遇到的问题及解决方法，实践心得

一开始对如何自底向上回溯找频繁项集不知道如何实现，后来参考了《机器学习实战》。还有就是对数据的处理，尤其是car，后来才发现car存在不同属性列直接有相同的取值这个问题，所以需要进行处理。每个数据都有其适合的算法，有时候硬套算法，可能意义不大。