게임 추천 프로젝트

컴퓨터정보통신공학과
182571
윤현서

목차

0. 요약

1. 정보수집
(1) 웹 크롤링
(2) 수집 정보
(3) 결과

2. 정보 처리
(1) 데이터 전처리
(2) 최종 데이터 파일 추출

3. 정보 분석
(1) 연관 규칙 분석
(2) 추천 알고리즘 적용

4. 정보 표현 (구현X)
(1) R의 그래픽 툴을 이용
(2) 웹 또는 앱 구현

0. 요약

(1) 웹크롤링을 통해 스팀 프로필 정보의 게임 이름과 플레이시간 수집
(2) 임의의 프로필 정보 입력 시 플레이할 가능성이 높은 게임 추천
((3). 추천 정보와 데이터 시각화를 웹 또는 앱에 적용)

1. 정보 수집

(1) 웹 크롤링

- 파이썬을 활용한 웹 크롤링을 통해 Steam 이용자 프로필 수집
- 코드(Python)

```
import re
import requests
import json
import datetime
import time
import csv
from fake_useragent import UserAgent

def steam_crawling(steamID, ID_Range, fileNum):
    userCount=int(fileNum)
    for n in range(ID_Range):
        time.sleep(35)
        steamID = steamID+1
        string_ID = str(steamID)

        ua = UserAgent()
        headers = {'User-Agent':str(ua.random)}

        url='https://steamcommunity.com/profiles/'+string_ID+'/games/?tab=all&sort=playtime'
        print(url)

        try:
            html=requests.get(url, headers=headers).text

            if 'An error was encountered' in html:
                #steamID = steamID + 1
                print('429 Error')
                time.sleep(1800)
                continue


            htmlSearch = re.search(r'var rgGames =(.+?);', html, re.S)

            gameList = htmlSearch.group(1)

            SteamGame = json.loads(gameList)
```

```
        except:
            continue

        if (str(SteamGame) != "[]" and 'hours_forever' in gameList):  #Private library
Check
            userCount = userCount+1
            f = open('D:/Project/'+str(userCount)+'n'+'.csv', 'w', newline='')
            wr=csv.writer(f)

            for course in SteamGame:
                try:
                    gameName = '{name}'.format(**course)
                    gameName = gameName.replace(',',' ') #if 'comma' inside GameName,
replace 'space'
                    gameTime = '{hours_forever}'.format(**course)
                    gameTime = gameTime.replace(',','') #if'comma' inside playtime,
replace 'space'
                    print(gameName)
                    print(gameTime)
                    wr.writerow([string_ID, gameName, gameTime])

                except:
                    continue

            f.close()

        else:
            print('Private account')
            continue


steamID=76561198293008336

now = datetime.datetime.now()

steam_crawling(steamID, 369, 49)
```

(2) 수집 정보

- 유저고유코드, 게임, 플레이시간

(3) 결과

- 노트북 : 3824명, 개인PC : 명, 총 ?명의 유저 중 비공개 유저를 제외하여
- 총 373개의 유저 데이터 수집

2. 정보 처리

(1) 데이터 전처리

- 중복 데이터 제거
- 이상치 데이터 제거 (과도한 플레이시간)
- 총 185개의 유저 데이터

(2) 최종 데이터 파일 추출

- 분할된 csv파일을 하나의 csv파일로 변환

3. 정보 분석

(1) 연관 규칙 분석

- 연관규칙 분석(유저고유코드 + 플레이한 게임)
- R studio 사용
- support=0.2, confidence = 0.20, minlen = 2로 연관 규칙 분석 결과 19개의 규칙 발견
- 코드(R)

```
# userdata Association Rule Analysis
# userdata(ID, Gamename)

# set working directory
setwd("D:/Project/test")

# association rule analysis package
library(arules)

# data import-> make transaction data
udata1<-read.csv("merge_new_userdata_notime.csv")
head(udata1)

udata.list<-split(udata1$Game,udata1$ID)

udata.trans<-as(udata.list,"transactions")
```

warning(s) : In asMethod(object) : removing duplicated items in transactions

udata.trans

# summary of userdata
summary(udata.trans)
#density : 0.016, 373*1104 cell -> 1.6%

# for running dvdtras data
# apriori(transaction, parameter=list(support=list(support=0.0#, confidence=0.##))
udata_rule<-apriori(udata.trans,parameter = list(support=0.2,confidence = 0.20,minlen = 2))

udata_rule
19 rules

inspect(udata_rule)

summary(udata_rule)

# Bar chart for support>0.2
itemFrequencyPlot(udata.trans,support=0.2,main="item for support>=0.2", col="blue")

- 발견된 규칙 (19가지)
lhs rhs support confidence lift count
[1] {Z1 Battle Royale} => {Counter-Strike: Global Offensive}
0.2037534 0.8837209 1.2030216 76
[2] {Counter-Strike: Global Offensive} => {Z1 Battle Royale}
0.2037534 0.2773723 1.2030216 76
[3] {Grand Theft Auto V} => {Counter-Strike: Global Offensive}
0.2359249 0.8979592 1.2224043 88
[4] {Counter-Strike: Global Offensive} => {Grand Theft Auto V}
0.2359249 0.3211679 1.2224043 88
[5] {Wallpaper Engine} => {Counter-Strike: Global Offensive}
0.2198391 0.8200000 1.1162774 82
[6] {Counter-Strike: Global Offensive} => {Wallpaper Engine}
0.2198391 0.2992701 1.1162774 82
[7] {Team Fortress 2} => {Counter-Strike: Global Offensive}
0.2198391 0.8723404 1.1875291 82
[8] {Counter-Strike: Global Offensive} => {Team Fortress 2}
0.2198391 0.2992701 1.1875291 82
[9] {Unturned} => {Counter-Strike: Global Offensive}

0.2091153 0.8666667 1.1798054 78
[10] {Counter-Strike: Global Offensive} => {Unturned}
0.2091153 0.2846715 1.1798054 78
[11] {PLAYERUNKNOWN'S BATTLEGROUNDS} => {Dota 2}
0.2707775 0.5706215 1.0433422 101
[12] {Dota 2} => {PLAYERUNKNOWN'S BATTLEGROUNDS}
0.2707775 0.4950980 1.0433422 101
[13] {PLAYERUNKNOWN'S BATTLEGROUNDS} => {Counter-Strike: Global Offensive}
0.3887399 0.8192090 1.1152006 145
[14] {Counter-Strike: Global Offensive} => {PLAYERUNKNOWN'S BATTLEGROUNDS}
0.3887399 0.5291971 1.1152006 145
[15] {Dota 2} => {Counter-Strike: Global Offensive}
0.3860590 0.7058824 0.9609274 144
[16] {Counter-Strike: Global Offensive} => {Dota 2}
0.3860590 0.5255474 0.9609274 144
[17] {Dota 2, PLAYERUNKNOWN'S BATTLEGROUNDS}
=> {Counter-Strike: Global Offensive}
0.2171582 0.8019802 1.0917468 81
[18]{Counter-Strike: Global Offensive,
PLAYERUNKNOWN'S BATTLEGROUNDS}
=> {Dota 2}
0.2171582 0.5586207 1.0213996 81
[19] {Counter-Strike: Global Offensive, Dota 2}
=> {PLAYERUNKNOWN'S BATTLEGROUNDS}
0.2171582 0.5625000 1.1853814 81

(2) 추천 알고리즘 적용

- 유사도와 KNN을 활용
- K Nearest Neighbors(KNN) 가중치 예측 기법 : 유사도가 구해지면 평점을 예측하고자 하는 사용자(또는 상품)와 유사도가 큰 k 개의 사용자(또는 상품) 벡터를 사용하여 가중 평균을 구해서 가중치를 예측

① 대상과 가장 유사도가 높은 k의 대상의 플레이시간과 유사도를 통해 추측평점((유사도 x (타인의)플레이시간)을 구한다.

② 추측 플레이시간의 총합을 구한 후, 추측평점 총합계/유사도 합계를 통해 예상 플레이시간을 뽑아낼 수 있다.

- 코드 (Python)

```
import math
import openpyxl
import re
import requests
import json
```

```
import datetime
import time
from fake_useragent import UserAgent

# web crawling
def steam_crawling(baseUrl):
        time.sleep(30)

        newGame = dict()

        ua = UserAgent()
        headers = {'User-Agent':str(ua.random)}

        #url='https://steamcommunity.com/profiles/'+string_ID+'/games/?tab=all&sort=playtime'
        url = baseUrl+'games/?tab=all&sort=playtime'
        print(url)

        try:
            html=requests.get(url, headers=headers).text

            if 'An error was encountered' in html:
                print('429 Error')



            htmlSearch = re.search(r'var rgGames =(.+?);', html, re.S)

            gameList = htmlSearch.group(1)

            SteamGame = json.loads(gameList)


        except:
            print('error')


        if (str(SteamGame) != "[]" and 'hours_forever' in gameList):  #Private library
Check

            for course in SteamGame:
                try:
```

```
                    gameName = '{name}'.format(**course)
                    gameName = gameName.replace(',',' ') #if 'comma' inside GameName,
replace 'space'
                    gameTime = '{hours_forever}'.format(**course)
                    gameTime = gameTime.replace(',','') #if'comma' inside playtime,
replace 'space'
                    print(gameName)
                    print(gameTime)
                    newGame[gameName] = float(gameTime)

                except:
                    print('error')


        else:
            print('비공개 계정입니다. 계정을 공개로 변경해주세요!')


        return newGame


def sim_msd(data, name1, name2):
    sum = 0
    count = 0
    for games in data[name1]:
        if games in data[name2]: #같은 게임을 했다면
            sum += pow(data[name1][games]- data[name2][games], 2)
            count += 1

    return 1 / ( 1 + (sum / count) )


def sim_cosine(data, name1, name2):
    sum_name1 = 0
    sum_name2 = 0
    sum_name1_name2 = 0
    count = 0
    for games in data[name1]:
        if games in data[name2]: #같은 게임을 했다면
            sum_name1 += pow(data[name1][games], 2)
            sum_name2 += pow(data[name2][games], 2)
```

```
                sum_name1_name2 += data[name1][games]*data[name2][games]

    return sum_name1_name2 / (math.sqrt(sum_name1)*math.sqrt(sum_name2))

def sim_pearson(data, name1, name2):
    avg_name1 = 0
    avg_name2 = 0
    count = 0
    for games in data[name1]:
        if games in data[name2]: #같은 게임을 했다면
            avg_name1 = data[name1][games]
            avg_name2 = data[name2][games]
            count += 1

    if(count == 0):
        return 0

    avg_name1 = avg_name1 / count
    avg_name2 = avg_name2 / count

    sum_name1 = 0
    sum_name2 = 0
    sum_name1_name2 = 0
    count = 0



    for games in data[name1]:
        if games in data[name2]: #같은 영화를 봤다면
            sum_name1 += pow(data[name1][games] - avg_name1, 2)
            sum_name2 += pow(data[name2][games] - avg_name2, 2)
            sum_name1_name2 += (data[name1][games] - avg_name1) *
(data[name2][games] - avg_name2)




    if( sum_name1_name2 == 0):
        return 0;

    return (sum_name1_name2 / (math.sqrt(sum_name1)*math.sqrt(sum_name2)))

def top_match(data, name, index=3, sim_function=sim_pearson):
```

```
    li=[]
    for i in data: #딕셔너리를 돌고
        if name!=i: #자기 자신이 아닐때만
            li.append((sim_function(data,name,i),i)) #sim_function()을 통해 상관계수를 구하
고 li[]에 추가
    li.sort() #오름차순
    li.reverse() #내림차순
    return li[:index]

def getRecommendation (data, person, k=3, sim_function=sim_pearson):

    result = top_match(data, person, k)

    score = 0 # 플레이 시간 합을 위한 변수
    li = list() # 리턴을 위한 리스트
    score_dic = dict() # 유사도 총합을 위한 dic
    sim_dic = dict() # 플레이 시간 총합을 위한 dic

    for sim, name in result: # 튜플이므로 한번에
        print(sim, name)
        #simuser = name
        if sim <= 0 : continue #유사도가 양수인 사람만
        for game in data[name]:
            if game not in data[person]: #name이 플레이 시간을 게임
                score += sim * data[name][game] # 그사람의 플레이 시간 * 유사도
                score_dic.setdefault(game, 0) # 기본값 설정
                score_dic[game] += score # 합계 구함

                # 조건에 맞는 사람의 유사도의 누적합을 구한다
                sim_dic.setdefault(game, 0)
                sim_dic[game] += sim

            score = 0  #게임이 바뀌었으니 초기화한다

    for key in score_dic:
        score_dic[key] = score_dic[key] / sim_dic[key] # 플레이 시간 총합/ 유사도 총합
        li.append((score_dic[key],key)) # list((tuple))의 리턴을 위해서.
    li.sort() #오름차순
    li.reverse() #내림차순
    #return li
    return name
```

```
userdata = { }
playGame = { }

preUser = 1
playGame = { }

# 엑셀파일 열기
wb = openpyxl.load_workbook('userdata.xlsx')

# 현재 Active Sheet 얻기
ws = wb.active

for r in ws.rows:
    userID = r[0].value

    if( preUser != userID ):
        userdata[preUser] = playGame
        playGame = { }
        preUser = userID

    game = r[1].value
    playtime = r[2].value
    playGame[game] = playtime

userdata[preUser] = playGame

baseUrl = input('스팀 프로필 주소를 입력해주세요!')

print('탐색 유저')

user = steam_crawling(baseUrl)

userdata[1000] = user

#print(user)
print()
# 한명만 가능
print('결과 유저')
print(userdata[getRecommendation(userdata, 1000, k=1, sim_function=sim_pearson)])
```

```
wb.close()
```

4. 정보 표현 (구현X)

(1) R의 그래픽 툴을 이용

(2) 웹 또는 앱 구현