УДК 004

Разработка API бота для обработки больших данных социальной сети Вконтакте

Вихляев Дмитрий Романович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье содержится описание скрипта API бота Вконтакте по нахождению общих групп среди подписчиков одного сообщества. Программа полностью написана на языке программирования python, с использованием методов общедоступного API Вконтакте. В результате исследования будут описаны способы получения, хранения и отображения больших данных, а также результат работы и исходный код программы.
Ключевые слова: vk api, python, парсинг.

Development of a bot API for processing big data of the Vkontakte social network

Vikhlyaev Dmitry Romanovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article contains a description of the script of the Vkontakte bot API for finding common groups among subscribers of the same community. The program is completely written in the python programming language, using the methods of the Vkontakte public API. As a result of the research, the methods of obtaining, storing and displaying big data, as well as the result of the work and the source code of the program will be described.
Keywords: vk api, python, parsing.

1 Введение

1.1 Актуальность

Парсинг данных в социальных сетях часто используется для поиска аудитории в коммерческих целях. Используя парсеры, можно быстро собирать и обрабатывать большие объёмы данных, в том числе находить потенциальных клиентов или сообщества для размещения рекламы. Чтобы найти потенциальных клиентов среди большой аудитории, необходимо знать их предпочтения. Однако поиск каждого пользователя в отдельности займёт много времени и сил. Лучше всего найти сообщества, в которых уже собраны группы потенциальных клиентов, а затем провести поиск на пересечение их интересов. Скорее всего, большинство пользователей будут подписаны на

другие схожие по теме сообщества. Группы, в которых количество пересечений будут наибольшими, являются самыми удачными для ретаргетинга рекламы.

1.2Обзор исследований

С.Н.Широбокова, В.С.Холодков, А.М.Бейбалаев провели исследование api-запросов социальной сети "Вконтакте" для мониторинга активности профориентационной деятельности консультантов университета [1]. А.А.Корепанова, И.Н.Москаленко описали выгрузку данных по api вконтакте [2]. Д.Р.Вихляев, В.А.Глаголев реализовали парсинг данных сообщества "Вконтакте" с помощью vk api [3]. Б.Р.Авхадеев Л.И.Воронова Е.П.Охапкина разработали рекомендательную систему на основе данных из профиля социальной сети "Вконтакте" [4]. Л.И.Тонких В.Г.Юрасов показали детали разработки информационной распределенной системы получения данных, построения и анализа социальных графов из сети вконтакте [5]. Р.Р.Галимов, И.А.Стефанова продемонстрировали способы получения данных из социальных сетей для поиска потенциальных абитуриентов [6].

1.3Цель исследования

Цель исследования – разработать программу поиска общих групп у участников одного сообщества Вконтакте.

2 Материалы и методы

Для реализации программы парсинга данных из группы используются общедоступный API Вконтакте, язык программирования python, модуль удобного обращения к методам API «vk_api» и графический модуль «tkinter».

3 Результаты и обсуждения

API Вконтакте предполагает использование get запросов на сервер и возвращение результата клиенту. В параметрах запроса необходимо указывать метод из документации API, а также ключ доступа и версию используемого API.

В языке программирования python существует библиотека vk_api позволяющая использовать простой интерфейс для обращения к методам. Помимо этого, в данном модуле имеется метод генерирующий ключ доступа и устанавливающий соединение сессию с а API Вконтакте. Таким образом, ускоряется подготовка к началу использования API, так как отсутствует необходимость создавать приложение и регистрировать ключ доступа самостоятельно. Необходимо создать приложение и получить ключ доступа. Для использования метода vk api, который позволяет извлечь данные о пользователя сообщества можно использовать серверный ключ доступа пользователя или сообщества.

Ниже приведён пример обращения к методу «VkApi», который принимает в качестве параметров логин и пароль, соответствующие учётной

записи Вконтакте и методу «auth» позволяющий получить доступ к методам API (рис.1).

```
def Auth(self,login='+7-xxx-xxx-xx-xx',password='vkpass'):
    self.session=vk_api.VkApi(login,password)

    # установление сессии по токену
    try:
        self.session.auth()
        #получение доступа к методам
        return self.session.get_api()
    except:
        return False
```

Рис. 1. Авторизация пользователя в API Вконтакте

Интерфейсом программы служит окно, созданное с помощью модуля «tkinter». В начале работы открывается окно с полем ввода текста для идентификатора группы и кнопкой запускающей функцию обработки. Для удобства, текстовому полю присвоены обработчики копирования и вставки.

Для хранения общих групп, используется пользовательская структура данных «CommonGroups», инициализированная в самом начале. В качестве параметров принимается максимальная вместимость идентификаторов группы. Так как среднее количество сообществ пользователя Вконтакте вирируется от 50 до 100, то для обработки группы с 10000 пользователей придётся использовать список или словарь с 1000000 элементов. Однако большая часть сообществ будет просто висеть в оперативной памяти из-за малого количества повторений у других пользователей. Дабы ускорить обработку данных, ведено ограничение на 1000 элементов (рис.2).

```
vk=MethodVk()
vk.Auth()

cg=CommonGroups(1000)

winvk=tk.Tk()
winvk.title("iouiou")
winvk.geometry("1000x500")

group_id=tk.StringVar()

logLabel = ttk.Label(text="input vk group id")
logLabel.pack()

vkentry = ttk.Entry(textvariable = group_id)
vkentry.pack()

vkentry.event_add('<<Paste>>', '<Control-igrave>')
vkentry.event_add("<<Copy>>", "<Control-ntilde>")

btn = ttk.Button(text="SEARCH", command=lambda :launch(group_id.get()))
btn.pack()

INFO=ttk.Label(text="")
INFO.pack()

winvk.mainloop()
```

Рис. 2. Отображение интерфеса

При инициализации объявляются все элементы структуры.

«buffer» –максимально возможное количество алиментов структуры.

«size» – текущий размер структуры.

«list_UC» – список групп и количество их повторений в порядке убывания числа повторов.

«dict_UI» – словарь групп и их индекс в «list_UC».

«dict_CI» – словарь количества повторений и диапазон индексов в «list_UC».

Данная структура позволяет вставлять, находить и перемещать данные групп за единицу времени (рис.3).

```
class CommonGroups():

    def __init__(self,count):
        self.buffer=count
        self.size=0
        self.list_UC=[]
        self.dict_UI={}
        self.dict_CI={}
```

Рис. 3. Инициализация структуры

При добавлении группы в структуру происходит проверка на существования этой группы в контейнере «dict_UI». Если такая группа уже добавлена, то изменяются данные об этой группе во всех контейнерах. В противном случае либо добавляется новый элемент или при наполненности буфера часть старых данных заменяются (рис.4).

```
def Add(self,group_id):
    if(self.dict_UI.get(group_id)!=None):
        index=self.dict_UI[group_id]
        self.list_UC[index][1]+=1
        self._changeToEveryContainer(index)

    else:
        newRepit=1
        if(self.size<self.buffer):
            self.size+=1

            #добавление во все контейнеры
            self._addToEveryContainer(group_id,newRepit)

        else:
            #Замена последних необновляющихся данных
            self._replaceToEveryContainer(group_id,newRepit)
```

Рис. 4. Метод добавления данных в структуру

Добавление новой группы в структуру происходит путём добавления соответствующих данных в каждый контейнер (рис.5).

```
def _addToEveryContainer(self,index,repit):
        self.list_UC.append([index,repit])

        self.dict_UI[index]=self.size-1

        if(self.dict_CI.get(repit)):
            self.dict_CI[repit][1]+=1
        else:
            self.dict_CI[repit]=[self.size-1,self.size]
```

Рис. 5. Метод добавления данных группы во все контейнеры

Замена старых данных на новые обеспечивает уменьшение количества элементов заполненной структуры путём удаления групп с самым маленьким количеством повторений. Данный промежуток хранится в контейнере «dict_CI» (рис.6).

```
def _replaceToEveryContainer(self,index,repit):
    oldRepit=self.list_UC[-1][1]

    start,stop=self.dict_CI[oldRepit]

    for u,i in self.list_UC[start:]:
        del self.dict_UI[u]

    del self.list_UC[start:]
    del self.dict_CI[oldRepit]
    self.size-=(stop-start-1)
    self._addToEveryContainer(index,repit)
```

Рис. 6. Метод замены старых данных групп новой группой

При обновлении данных уже существующей группы, увеличивается количество повторений на единицу и меняется расположение данной группы в «list_UC» (рис. 7).

```
def _changeToEveryContainer(self,index):

    repit=self.list_UC[index][1]
    if(self.dict_CI[repit-1][0]==index):
        if(self.dict_CI.get(repit)):
            self.dict_CI[repit][1]+=1
            self.dict_CI[repit-1][0]+=1
            if(self.dict_CI[repit-1][0]==self.dict_CI[repit-1][1]):
                del self.dict_CI[repit-1]
        else:
            self.dict_CI[repit]=[index,index+1]
            self.dict_CI[repit-1][0]+=1
            if(self.dict_CI[repit-1][0]==self.dict_CI[repit-1][1]):
                del self.dict_CI[repit-1]
    else:
        if(self.dict_CI.get(repit)):
            #обмен
            tmpRepit=self.dict_CI[repit-1][0]
            tmpGroup_id1=self.list_UC[tmpRepit][0]
            tmpGroup_id2=self.list_UC[index][0]
            self.list_UC[index],self.list_UC[tmpRepit]=self.list_UC[tmpRepit],self.list_UC[index]
            self.dict_UI[tmpGroup_id1],self.dict_UI[tmpGroup_id2]=self.dict_UI[tmpGroup_id2],self.dict_UI[tmpGroup_id1]

            self.dict_CI[repit][1]+=1
            self.dict_CI[repit-1][0]+=1

        else:
            #обмен
            tmpRepit=self.dict_CI[repit-1][0]
            tmpGroup_id1=self.list_UC[tmpRepit][0]
            tmpGroup_id2=self.list_UC[index][0]
            self.list_UC[index],self.list_UC[tmpRepit]=self.list_UC[tmpRepit],self.list_UC[index]

            self.dict_UI[tmpGroup_id1],self.dict_UI[tmpGroup_id2]=self.dict_UI[tmpGroup_id2],self.dict_UI[tmpGroup_id1]
```

Рис. 7. Обновление данных во всех контейнерах

В начале функции обработки проверяется возможность обращения к данной группе и получения у неё списка пользователей. При удачном выполнении запроса возвращается список идентификаторов пользователей и полное количество пользователей сообщества. Чтобы извлечь пользователей сообщества используется метод «groups.getMembers». За одно обращение можно получить не более 1000 пользователей.

Затем вызывается пользовательская функция «search», которая находит общие группы у первого списка участников.

Если группа имеет более 1000 участников, тогда вновь вызывается метод «groups.getMembers» для получения пользователей.

API Вконтакте имеет метод «execute» с помощью которого можно за раз обратится к 25 методам и получить двумерный список их результатов. Таким образом, за раз можно получить до 25000 участников сообщества (рис. 8,9,10,13).

```
if(vk):
    try:
        items=vk.count_users(group_id)
        INFO["text"]="processing {}%".format(STATUS)
    except:
        INFO["text"]="нет доступа к группе"
        sys.exit()


    items=[items]
    cgl=ttk.Label(text="количество участников {}".format(vk.gCount))
    cgl.pack()

    search(items)

    for offset in range(1000,vk.gCount,25000):
        api=MethodVk.metod_getMembers(group_id,offset,vk.gCount)
        items=vk.execute(api)
        search(items)
```

Рис. 8. Запуск функции обработки

```
def count_users(self,group_id=166301116):
    users=self.session.method("groups.getMembers",{"group_id": group_id})
    self.gCount=users["count"]
    return users["items"]
```

Рис. 9. Метод получения участников сообщества и их количества

```
def metod_getMembers(group_id,start,count):
    api = "API.groups.getMembers({{'group_id':{},'offset':{}}}).items".format(group_id,start)
    stop=start+25000 if count>start+25000 else count
    for i in range(start+1000,stop,1000):
        api+=",API.groups.getMembers({{'group_id':{},'offset':{}}}).items".format(group_id,i)
    return api
```

Рис. 10. Создание 25 методов «groups.getMembers»

Функция «search» принимает двумерный список идентификаторов пользователей. Используя методы «get_groups» и «execute» получает список всех групп 25 участников, после чего отправляет их на добавление в структуру «CommonGroups».

Контроль исключений используется чтобы предотвратить ошибки сервера на получение запросов. Ошибка может возникнуть из-за ограничения

API на количество запросов в секунду, их должно быть не более трёх (рис.11,12,13).

```
def search(items):
    global STATUS
    for k in range(len(items)):
        for w in range(0,len(items[k]),25):
            api=MethodVk.get_groups(items,k,w)
            tryrepit=True
            while(tryrepit):
                try:
                    resp=vk.execute(api)
                    for i in resp:
                        if(i==None):
                            continue
                        for j in i:
                            cg.Add(j)
                    tryrepit=False
                except:
                    sleep(0.5)
```

Рис. 11. Получение групп пользователей и добавление в структуру

```
def get_groups(users,index,start):
    api = "API.groups.get({{'user_id':{}}}).items".format(users[index][start])
    stop = start+25 if len(users[index])>start+25 else len(users[index])
    for i in range(start+1,stop):
        api+=",API.groups.get({{'user_id':{}}}).items".format(users[index][i])
    return api
```

Рис. 12. Создание 25 методов «get_groups» с разными идентификаторами пользователей

```
def execute(self,api):
    reaponse=self.session.method("execute",{"code": "return [{}];".format(api)})
    #print(users)
    return reaponse
```

Рис. 13. Вызов метода «execute»

Выгрузка данных выводится в виде таблицы, с помощью виджита «Treeview».

```
lg = ttk.Treeview(winvk)
lg['columns'] = ('group_id','img','name','reference','common_user')
lg["show"]="headings"
lg.heading('group_id', text='group_id', anchor=tk.CENTER)
lg.heading('img', text='img', anchor=tk.CENTER)
lg.heading('name', text='name', anchor=tk.CENTER)
lg.heading('reference', text='reference', anchor=tk.CENTER)
lg.heading('common_user', text='common_user', anchor=tk.CENTER)
```

Рис. 14. Создание виджета-таблицы для выгрузки данных

Структура «CommonGroups» предназначена, чтобы хранить идентификаторы сообществ в порядке убывания по количеству встречаемости у пользователей, поэтому чтобы вывести общие группы достаточно сделать срез первых нескольких элементов списка «list_UC».

Дополнительную информацию о группах по идентификаторам можно получить, используя метод «groups.getById». С помощью данного метода

можно получить информацию сразу у нескольких групп (до 500 сообществ за раз). Однако при подстановке списка идентификаторов, параметр «group_ids» отправит информацию только о последнем. Необходимо преобразовать список, в строку разделив элементы запятой. Для этого используются методы «map» и «join». Используя данный метод, получено название, адрес и аватар сообществ.

Далее циклически заполняется таблица через метод «Treeview.insert», данные присваиваются параметру «values»(рис.15,16).

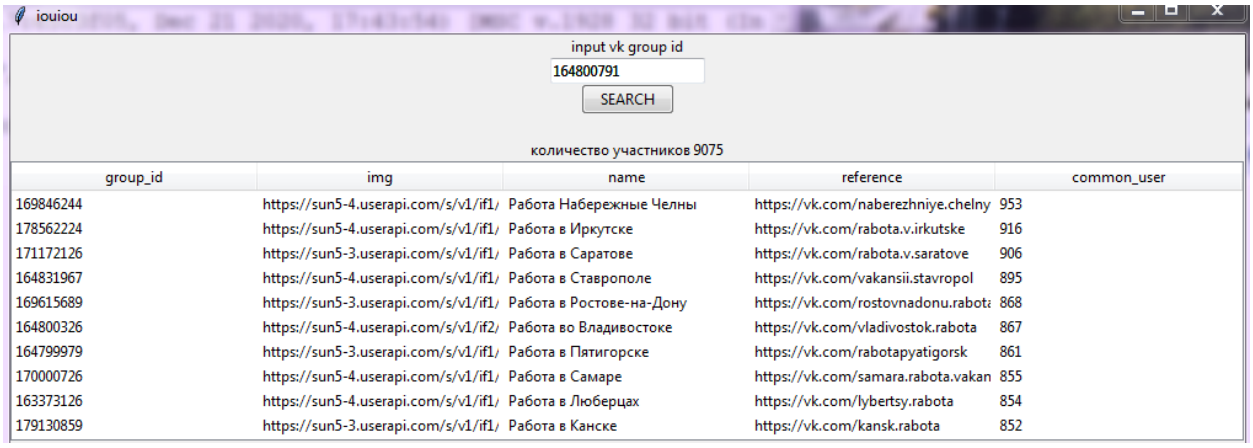
```
gids=[]
for i,v in cg.list_UC[:25]:
    if(i==group_id):
        continue
    gids.append(i)

dg=vk.get_data_groups(gids)

for i in range(25):
    lg.insert('',i,text='',
        values=(cg.list_UC[i][0],dg[i]['photo_50'],dg[i]['name'],
              'https://vk.com/{}'.format(dg[i]['screen_name']),cg.list_UC[i][1]))

lg.pack()
```

Рис. 15. Заполнение таблицы данными

```
def get_data_groups(self,group_ids):
    dg=self.session.method("groups.getById",
                           {"group_ids": ",".join(map(str,group_ids)),
                            "fields":"name,screen_name,photo_50"})
    return dg
```

Рис. 16. Вызов метода «groups.getById»

Ниже приведён пример работы программы на сообществе «Работа в Хабаровске» с 9075 участниками (рис. 17).



group_id	img	name	reference	common_user
169846244	https://sun5-4.userapi.com/s/v1/if1/	Работа Набережные Челны	https://vk.com/naberezhniye.chelny	953
178562224	https://sun5-4.userapi.com/s/v1/if1/	Работа в Иркутске	https://vk.com/rabota.v.irkutske	916
171172126	https://sun5-3.userapi.com/s/v1/if1/	Работа в Саратове	https://vk.com/rabota.v.saratove	906
164831967	https://sun5-4.userapi.com/s/v1/if1/	Работа в Ставрополе	https://vk.com/vakansii.stavropol	895
169615689	https://sun5-3.userapi.com/s/v1/if1/	Работа в Ростове-на-Дону	https://vk.com/rostovnadonu.rabota	868
164800326	https://sun5-4.userapi.com/s/v1/if2/	Работа во Владивостоке	https://vk.com/vladivostok.rabota	867
164799979	https://sun5-3.userapi.com/s/v1/if1/	Работа в Пятигорске	https://vk.com/rabotapyatigorsk	861
170000726	https://sun5-4.userapi.com/s/v1/if1/	Работа в Самаре	https://vk.com/samara.rabota.vakan	855
163373126	https://sun5-4.userapi.com/s/v1/if1/	Работа в Люберцах	https://vk.com/lybertsy.rabota	854
179130859	https://sun5-3.userapi.com/s/v1/if1/	Работа в Канске	https://vk.com/kansk.rabota	852

Рис. 16. Вывод общих сообществ у подписчиков одной группы

В результате данной статьи, были продемонстрированы необходимые методы API Вконтакте и реализован графический интерфейс пользователя. Также описана структура, для хранения значимых и перезаписью не влияющих на результат данных, с быстрым поиском, удалением и перемещением.

Работу данной программы можно сравнить с одним из парсеров, которые доступны на сервисе поиска целевой аудитории в социальных сетях, по адресу: https://vk.barkov.net/auditory.aspx.

Библиографический список

1. Широбокова С.Н., Холодков В.С., Бейбалаев А.М. Использование api-запросов социальной сети "Вконтакте" для мониторинга активности профориентационной деятельности консультантов университета // Новая наука: Проблемы и перспективы. 2017. № 1-2. С. 185-187.
2. Корепанова А.А., Москаленко И.Н. Выгрузка данных по api Вконтакте // В сборнике: Региональная информатика и информационная безопасность. Сборник трудов XII Санкт-Петербургской межрегиональной конференции. Санкт-Петербург, 2021. С. 314-317.
3. Вихляев Д.Р., Глаголев В.А. Парсинг данных сообщества "Вконтакте" с помощью vk api // Постулат. 2021. № 10 (72).
4. Авхадеев Б.Р., Воронова Л.И., Охапкина Е.П. Разработка рекомендательной системы на основе данных из профиля социальной сети "Вконтакте" // Вестник Нижневартовского государственного университета. 2014. № 3. С. 68-76.
5. Тонких Л.И., Юрасов В.Г. Разработка информационной распределенной системы получения данных, построения и анализа социальных графов из сети вконтакте // Информация и безопасность. 2015. Т. 18. № 4. С. 488-495.
6. Галимов Р.Р., Стефанова И.А. Способы получения данных из социальных сетей для поиска потенциальных абитуриентов // Инновации в науке. 2016. № 57-1. С. 37-41
7. Диков М.Е., Жевакин Д.М., Широбокова С.Н. О Реализации подхода к поиску пользователей из определенного города в социальной сети в условиях ограничений api методов // Молодой исследователь Дона. 2021. № 2 (29). С. 20-24.
8. Каляуш В.В. Подсистема выбора активных участников групп с анализом интересов для сегментированой таргетинговой рекламы информационной системы управления рекламным кабинетом в социальной сети "вконтакте" // В сборнике: Информационные и измерительные системы и технологии. Сборник научных статей по материалам еженедельного научно-технического семинара. 2016. С. 26-32.