Лабораторная работа №3

Поиск ассоциативных правил

Поиск ассоциативных правил представляет собой весьма часто применяемый метод выявления зависимостей между записями в больших наборах данных.

Ассоциативные правила обычно используются при анализе продуктовых корреляций (Market Basket Analysis) и выглядят примерно так: «если в чеке клиента есть молоко, то в 80% случаев там будет и хлеб».

Набор транзакций Instacart

Компания Instacart (сайт https://www.instacart.com (https://www.instacart.com)) – розничный продавец овощей в США и Канаде (слоган «Groceries delivered in as little as 1 hour»).

Для исследователей на сайте Instacart был выложен набор данных “The Instacart Online Grocery Shopping Dataset 2017”, адрес набора https://www.instacart.com/datasets/grocery-shopping-2017 (https://www.instacart.com/datasets/grocery-shopping-2017).

Набор данных анонимизирован и состоит из следующих таблиц.

`orders` (3.4m rows, 206k users):

- `order_id` : order identifier
- `user_id` : customer identifier
- `eval_set` : which evaluation set this order belongs in (see `SET` described below)
- `order_number` : the order sequence number for this user (1 = first, n = nth)
- `order_dow` : the day of the week the order was placed on
- `order_hour_of_day` : the hour of the day the order was placed on
- `days_since_prior` : days since the last order, capped at 30 (with NAs for `order_number` = 1)

`products` (50k rows):

- `product_id` : product identifier
- `product_name` : name of the product
- `aisle_id` : foreign key
- `department_id` : foreign key

`aisles` (134 rows):

- `aisle_id` : aisle identifier
- `aisle` : the name of the aisle

`deptartments` (21 rows):

- `department_id` : department identifier
- `department` : the name of the department

`order_products__SET` (30m+ rows):

- `order_id` : foreign key
- `product_id` : foreign key
- `add_to_cart_order` : order in which each product was added to cart
- `reordered` : 1 if this product has been ordered by this user in the past, 0 otherwise

где `SET` - один из следующих наборов (`eval_set` в таблице `orders`):

- `"prior"` : orders prior to that users most recent order (~3.2m orders)
- `"train"` : training data supplied to participants (~131k orders)
- `"test"` : test data reserved for machine learning competitions (~75k orders)

Для работы с набором данных Instacart будем использовать встроенную в Python СУБД `sqlite` .

SQL

База данных (БД) - это данные, которые хранятся в соответствии с определенной схемой, описывающей соотношения между данными.

Язык БД SQL (structured query language) - используется для описания структуры БД, управления данными (добавление, изменение, удаление, получение), управления правами доступа к БД и ее объектам, управления транзакциями.

Система управления базами данных (СУБД) - это программные средства, которые дают возможность управлять БД, поддерживая соответствующий язык (языки) для управления БД.

Язык SQL подразделяется на следующие категории:

`DDL` (Data Definition Language) - язык описания данных

- `CREATE` - создание новой таблицы, СУБД, схемы
- `ALTER` - изменение существующей таблицы, колонки
- `DROP` - удаление существующих объектов из СУБД

`DML` (Data Manipulation Language) - язык манипулирования данными

- `SELECT` - выбор данных
- `INSERT` - добавление новых данных
- `UPDATE` - обновление существующих данных
- `DELETE` - удаление данных

`DCL` (Data Control Language) - язык определения доступа к данным

- `GRANT` - предоставление пользователям разрешения на чтение/запись определенных объектов в СУБД
- `REVOKE` - отзыв ранее предоставленных разрешений

`TCL` (Transaction Control Language) - язык управления транзакциями

- `COMMIT` - применение транзакции
- `ROLLBACK` - откат всех изменений, сделанных в текущей транзакции

SQLite

SQLite — компактная встраиваемая СУБД с открытым исходным кодом, написанная на языке C.

SQLite не использует парадигму клиент-сервер, а именно, SQLite представляет собой библиотеку, с которой компонуется программа, использующая SQLite. SQLite является составной частью Python.

Утилита sqlite3

В комплект поставки SQLite входит утилита sqlite3 для работы с SQLite в командной строке. С помощью sqlite3 можно вручную выполнять команды SQL.

Для доступа к базе данных Instacart при помощи sqlite3 нужно выполнить следующую команду в окне терминала:

```
sqlite3 instacart.db
```

В sqlite3 можно выполнять команды SQL или так называемые метакоманды (или dot-команды).

К метакомандам относятся несколько специальных команд для работы с SQLite. Они относятся только к утилите sqlite3, а не к SQL языку. В конце этих команд ; ставить не нужно. Примеры метакоманд:

- .help - подсказка со списком всех метакоманд
- .exit или .quit - выход из сессии sqlite3
- .databases - показывает присоединенные БД
- .tables - показывает доступные таблицы

При выполнении в sqlite3 команды SQL ее нужно завершить символом `;` , например:

```
select count(*) from orders;
```

Модуль sqlite3

Для работы с SQLite в Python используется модуль sqlite3.

Объект `Connection` - это подключение к конкретной БД. Пример создания подключения:

In [ ]:
```
import sqlite3

conn = sqlite3.connect('temp.db')
```

In [ ]:
```
type(conn)
```

Если база данных temp.db не существует, то она будет создана. После создания соединения надо создать объект Cursor - это основной способ работы с БД. Создается курсор из соединения с БД:

In [ ]:
```
cursor = conn.cursor()

type(cursor)
```

Выполнение команд SQL

Для выполнения команд SQL в модуле есть несколько методов:

- `execute()` - метод для выполнения одного выражения SQL
- `executemany()` - метод позволяет выполнить одно выражение SQL для последовательности параметров (или для итератора)
- `executescript()` - метод позволяет выполнить несколько выражений SQL за один раз

Метод execute()

Метод execute() позволяет выполнить одну команду SQL (при условии, что уже созданы соединение и курсор).

Создадим таблицу user с помощью метода execute():

In [ ]:
```
cursor.execute("""create table user
    (userid text not NULL primary key,
    username text,
    room text,
    phone text)""")
```

Выражения SQL могут быть параметризированы - вместо данных можно подставлять специальные значения. За счет этого можно использовать одну и ту же команду SQL для передачи разных данных.

Например, таблицу user нужно заполнить данными из списка data:

In [ ]:
```
data = [
    ('001', 'Иванов И.И.',  '123', '(985)1234567'),
    ('002', 'Петров П.П.',  '234', '(903)9876543'),
    ('003', 'Сидоров С.С.', '345', '(495)1357900')]
```

Для этого можно использовать запрос вида:

In [ ]:
```
query = "INSERT into user values (?, ?, ?, ?)"
```

Знаки вопроса в команде используются для подстановки данных, которые будут передаваться методу execute.

Теперь можно передать данные таким образом:

In [ ]:
```
for row in data:
    cursor.execute(query, row)
```

Второй аргумент, который передается методу execute, должен быть кортежем. Если нужно передать кортеж с одним элементом, используется запись вида (value,).

Чтобы изменения были применены, нужно выполнить commit() (обратите внимание, что метод commit() вызывается у соединения):

In [ ]:
```
conn.commit()
```

Обратимся к сохраненным данным:

In [ ]:
```
for row in cursor.execute("SELECT * FROM user"):
    print(row)
```

Метод executemany()

Метод executemany() позволяет выполнить одну команду SQL для последовательности параметров (или для итератора).

С помощью метода executemany() в таблицу user можно добавить аналогичный список данных одной командой.

Например, в таблицу user надо добавить данные из списка data2:

In [ ]:
```
data2 = [
    ('004', 'Васильев В.В.', '456', '(901)1112233'),
    ('005', 'Сергеев С.С.',  '567', '(905)1231212')]
```

Для этого нужно использовать аналогичный запрос вида:

In [ ]:
```
query = "INSERT into user values (?, ?, ?, ?)"
```

Теперь можно передать данные методу executemany():

In [ ]:
```
cursor.executemany(query, data2)

conn.commit()
```

Метод executemany() подставил соответствующие кортежи в команду SQL, и все данные добавились в таблицу.

In [ ]:
```
for row in cursor.execute("SELECT * FROM user"):
    print(row)
```

Метод executescript()

Метод executescript() позволяет выполнить несколько выражений SQL за один раз:

In [ ]:
```
cursor.executescript('''
delete from user;
drop table user;
''')
conn.close()
```

Получение результатов запроса

Для получения результатов запроса в sqlite3 есть несколько способов:

- использование методов fetch...() - в зависимости от метода возвращаются одна, несколько или все строки
- использование курсора как итератора - возвращается итератор

Метод fetchone()

Метод fetchone() возвращает одну строку данных.

Пример получения информации из базы данных instacart.db:

In [ ]:
```
conn = sqlite3.connect('instacart.db')

cursor = conn.cursor()

cursor.execute('select * from departments')

cursor.fetchone()
```

Обратите внимание, что хотя запрос SQL подразумевает, что запрашивалось всё содержимое таблицы, метод fetchone() вернул только одну строку.

Если повторно вызвать метод, он вернет следующую строку:

In [ ]:
```
cursor.fetchone()
```

Аналогичным образом метод будет возвращать следующие строки. После обработки всех строк метод начинает возвращать None.

За счет этого метод можно использовать в цикле, например, так:

In [ ]:
```
cursor.execute('select * from departments')

while True:
    next_row = cursor.fetchone()
    if next_row:
        print(next_row)
    else:
        break
```

Метод fetchmany()

Метод fetchmany() возвращает список строк данных. Синтаксис метода:

```
cursor.fetchmany([size=cursor.arraysize])
```

С помощью параметра size можно указывать, какое количество строк возвращается. По умолчанию параметр size равен значению cursor.arraysize:

In [ ]:
```
print(cursor.arraysize)
```

Например, таким образом можно возвращать по три строки из запроса:

In [ ]:
```
cursor.execute('select * from aisles')

while True:
    three_rows = cursor.fetchmany(3)
    if three_rows:
        print(three_rows)
    else:
        break
```

Метод выдает нужное количество строк, а если строк осталось меньше, чем параметр size, то оставшиеся строки.

Метод fetchall()

Метод fetchall() возвращает все строки в виде списка:

In [ ]:
```
cursor.execute('select * from departments')

cursor.fetchall()
```

Важный аспект работы метода - он возвращает все оставшиеся строки.

То есть, если до метода fetchall() использовался, например, метод fetchone(), то метод fetchall() вернет оставшиеся строки запроса.

Cursor как итератор

Если нужно построчно обрабатывать результирующие строки, лучше использовать курсор как итератор. При этом не нужно использовать методы fetch.

При использовании методов execute возвращается курсор. А, так как курсор можно использовать как итератор, можно использовать его, например, в цикле for:

In [ ]:
```
result = cursor.execute('select * from departments')

for row in result:
    print(row)
```

Аналогичный вариант отработает и без присваивания переменной:

In [ ]:
```
for row in cursor.execute('select * from departments'):
    print(row)
```

Использование sqlite3 без создания курсора

Методы execute доступны и в объекте Connection, и в объекте Cursor, а методы fetch доступны только в объекте Cursor. При использовании методов execute с объектом Connection курсор возвращается как результат выполнения метода execute. Его можно использовать как итератор и получать данные без методов fetch. За счет этого при работе с модулем sqlite3 можно не создавать курсор.

In [ ]:
```
data = [
    ('001', 'Иванов И.И.', '123', '(985)1234567'),
    ('002', 'Петров П.П.', '234', '(903)9876543'),
    ('003', 'Сидоров С.С.', '345', '(495)1357900'),
    ('004', 'Васильев В.В.', '456', '(901)1112233'),
    ('005', 'Сергеев С.С.', '567', '(905)1231212'),
]
conn2 = sqlite3.connect('temp.db')
conn2.execute('''
    create table user
        (userid text not NULL primary key,
        username text,
        room text,
        phone text)
    '''
)
query = 'INSERT into user values (?, ?, ?, ?)'
conn2.executemany(query, data)
conn2.commit()
for row in conn2.execute('select * from user'):
    print(row)
#conn2.close()
```

Обработка исключений

Посмотрим на пример использования метода execute при возникновении ошибки. В таблице user поле mac должно быть уникальным. И, если попытаться записать пересекающийся MAC-адрес, возникнет ошибка:

In [ ]:
```
query = "INSERT into user values ('005', 'Антонов А.А.', '678', '(9

conn2.execute(query)
```

Соответственно, можно перехватить исключение:

In [ ]:
```
try:
    conn2.execute(query)
except sqlite3.IntegrityError as e:
    print("Произошла ошибка: ", e)

conn2.close()
```

Обратите внимание, что надо перехватывать исключение sqlite3.IntegrityError, а не IntegrityError.

SQL запросы к базе данных Instacart

SQL запросы к базе данных Instacart будут предусматривать соединение различных таблиц базы данных. Например, подсчитаем количество заказов в разбивке по департаментам:

In [ ]:
```
for row in cursor.execute("""
        SELECT count(DISTINCT ord.order_id),dept.department
        FROM order_products__train as ord,products as prod,departme
        WHERE ord.product_id=prod.product_id AND prod.department_id
        GROUP BY dept.department
        """):
    print(row)
```

Получим количество заказов в разбивке по дням недели:

In [ ]:
```
for row in cursor.execute("""
    select
      count(order_id) as total_orders,
      (case
         when order_dow = '0' then 'Sunday'
         when order_dow = '1' then 'Monday'
         when order_dow = '2' then 'Tuesday'
         when order_dow = '3' then 'Wednesday'
         when order_dow = '4' then 'Thursday'
         when order_dow = '5' then 'Friday'
         when order_dow = '6' then 'Saturday'
       end) as day_of_week
      from orders
     group by order_dow
     order by total_orders desc
     """):
    print(row)
```

Получим количество заказов в разбивке по часам дня:

In [ ]:
```
for row in cursor.execute("""
    select
      count(order_id) as total_orders,
      order_hour_of_day as hour
      from orders
     group by order_hour_of_day
     order by order_hour_of_day
         """):
    print(row)
```

```
Получим 10 наиболее популярных продуктов:
```

In [ ]:
```
for row in cursor.execute("""
    select count(opp.order_id) as orders, p.product_name as popular
    from order_products__train opp, products p
    where p.product_id = opp.product_id
    group by popular_product
    order by orders desc
    limit 10
    """):
    print(row)
```

In [ ]:
```
conn.close()
```

Модуль itertools

Модуль itertools расширяет функционал Python, связанный с созданием последовательностей объектов и манипулированием ими.

Функция combinations()

Функция combinations() позволяет комбинировать отдельные элементы последовательности и принимает два аргумента. Первый позволяет задать определенный объект, а второй – количество значений, которые будут присутствовать в каждом новом элементе.

In [ ]:
```
from itertools import combinations
data = list(combinations('РУДН', 2))
print(data)
```

Как видно из кода, функция получает строку `РУДН`, которая впоследствии раскладывается на отдельные символы. Далее происходит группировка по 2 буквы так, чтобы каждая новая выборка отличалась от всех существующих. Функция print выводит полученный список data на экран, отображая все сформированные пары символов `Р`, `У`, `Д`, `Н`.

Аналогично функция combinations работает и со списками:

In [ ]:
```
data2 = list(combinations([1,2,3,4,5], 3))
print(data2)
```

Функция chain()

Функция chain() выполняет объединение списков, возвращая итератор, как это показано в следующем примере. Итоговый массив содержит все элементы данных последовательностей.

In [ ]:
```
from itertools import chain
list1 = ['Р', 'У', 'Д', 'Н']
list2 = [1, 2, 3, 4, 5]
data = list(chain(list1, list2))
print(data)
```

Функция chain.from_iterable()

Функция работает аналогично функции chain, выполняя объединение списков. Отличие заключается в том, что аргумент только один – вложенный список со списками, которые надо объединить.

In [ ]:
```
from itertools import chain
list0 = [['Р', 'У', 'Д', 'Н'], [1, 2, 3, 4, 5]]
data = list(chain.from_iterable(list0))
print(data)
```

Множество всех подмножеств

Для построения множества всех подмножеств данного множества (в теоретико-множественном смысле) можем использовать следующую функцию:

In [ ]:
```
from itertools import chain, combinations

def powerset(iterable):
    xs = list(iterable)
    # возвращаем итератор, а не список
    return chain.from_iterable(combinations(xs,n) for n in range(le

print(list(powerset([1,2,3,4])))
```

Поиск ассоциативных правил в учебном наборе транзакций

Будем использовать следующий учебный набор транзакций:

In [ ]:
```
D_train = [
    [ 1, {"A","B","D","E"} ],
    [ 2, {"B","C","E"} ],
    [ 3, {"A","B","D","E"} ],
    [ 4, {"A","B","C","E"} ],
    [ 5, {"A","B","C","D","E"} ],
    [ 6, {"B","C","D"} ],
]
```

Чтобы посчитать поддержку заданного набора предметов, будем использовать следуюшую функцию:

In [ ]:
```
def ComputeSupport( X, D ):
    supX = 0
    for _,itemset in D:
        if X.issubset( itemset ):
            supX += 1
    return supX
```

Например,

In [ ]:
```
X = {"B","C","E"}

print("\nsup(", X, ") =", ComputeSupport(X, D_train))
```

Для построения множества всех подмножеств используем функцию powerset():

In [ ]:
```
I = {"A","B","C", "D", "E"}

print("\nМножество всех подмножеств множества", I, ":")
for itemset in powerset( I ):
    print( itemset )
```

Построение популярных наборов предметов

Алгоритм BruteForce может быть реализован следующим образом:

In [ ]:
```
def BruteForce( D, I, minsup ):
    F = []
    for X in powerset( I ):
        if len( X ) > 0:
            supX = ComputeSupport( set( X ), D )
            if supX >= minsup:
                F.append( [ X, supX ] )
    return F
```

Построим популярные наборы для заданного уровня минимальной поддержки:

In [ ]:
```
minsup = 3

print("\nПопулярные наборы предметов для minsup =", minsup, ":")

for itemset in BruteForce( D_train, I, minsup ):
    print( itemset )
```

Замечание. При построении дерева префиксов в алгоритмах Apriori, Eclat и dEclat нужно разделять названия предметов (товаров) каким-либо символом (разделителем), который не встречается в названиях.

Построение ассоциативных правил

Будем считать, что задан популярный набор предметов и задача состоит в получении ассоциативных правил с заданным минимальным уровнем достоверности.

In [ ]:
```
F_set, _ = BruteForce( D_train, I, minsup )[-1]
F_set
```

Алгоритм AssociationRules для случая построения ассоциативных правил по заданному популярному набору предметов может быть реализован следующим образом:

In [ ]:
```
def powersetk(iterable,k):
    xs = list(iterable)
    # возвращаем итератор, а не список
    return chain.from_iterable(combinations(xs,n) for n in range(k,

def AssociationRules(D, Z_set, minconf):
    A_rules = []
    supZ = ComputeSupport(set(Z_set), D)
    A_set = list(powersetk(Z_set,1))[:-1]
    # print("\nA_set:",A_set)
    while len(A_set)>0:
        X_set = A_set[-1]
        #print("\nX_set:",X_set)
        A_set.pop()
        conf = supZ/ComputeSupport(set(X_set), D)
        if conf >= minconf:
            Y_set = sorted(list(set(Z_set)-set(X_set)))
            A_rules.append([X_set, Y_set, supZ, conf])
        else:
            for W_set in powersetk(X_set,1):
                if W_set in A_set:
                    A_set.remove(W_set)
    return A_rules

AssociationRules(D_train, F_set, 0.9)
```

Задание на лабораторную работу №3

Задание (10 баллов)

Для закрепленного за Вами варианта лабораторной работы:

1. При помощи модуля sqlite3 откройте базу данных Instacart в файле instacart.db.
2. При помощи запроса SELECT извлеките из таблицы order_products__train записи, соответствующие указанным в индивидуальном задании дню недели (поле order_dow таблицы orders) и коду департамента (поле department_id таблицы products). Определите количество записей в полученном наборе и определите количество товаров (поле order_id таблицы order_products__train) в транзакциях набора.
3. Определите количество покупок (транзакций) для пяти наиболее популярных товаров в наборе.
4. Постройте транзакционную базу данных для поиск ассоциативных правил из полученного набора записей таблицы order_products__train, используя в качестве идентификатора транзакции поле order_id, а в качестве названий товаров - поле product_name из таблицы products, соответствующее полю product_id.
5. Реализуйте указанный в индивидуальном задании метод построения популярных наборов предметов (Apriori/Eclat/Declat) (3 балла) или используйте метод BruteForce (0 баллов). Протестируйте корректность реализации алгоритма на учебном наборе данных из материалов лекции.
6. При помощи указанного в индивидуальном задании метода или метода BruteForce постройте популярные наборы товаров с минимальной поддержкой, равной половине среднего количества покупок пяти наиболее популярных товаров. В случае нехватки вычислительных ресурсов (слишком долгой работы программы) при построении популярных наборов товаров оставьте в наборе данных транзакции с 10 наиболее популярными товарами и сокращайте число записей (например, методом деления пополам).
7. Для какого-либо из полученных популярных наборов товаров постройте набор ассоциативных правил.
8. Для построенного набора ассоциативных правил вычислите показатели: support, confidence, lift, leverage, conviction и выведите на экран.

In [ ]: