Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2 , 842 16 Bratislava

Zadanie č.2

Information retrieval

Cvičiaci: doc. Ing. Michal Kompan, PhD.
Cvičenie: Ut, 8:00
Autor: Bc. Jozef Varga
Študijný program: Inteligentné softvérové systémy
Predmet: Vyhľadávanie informácií
Akad. rok: 2019/2020

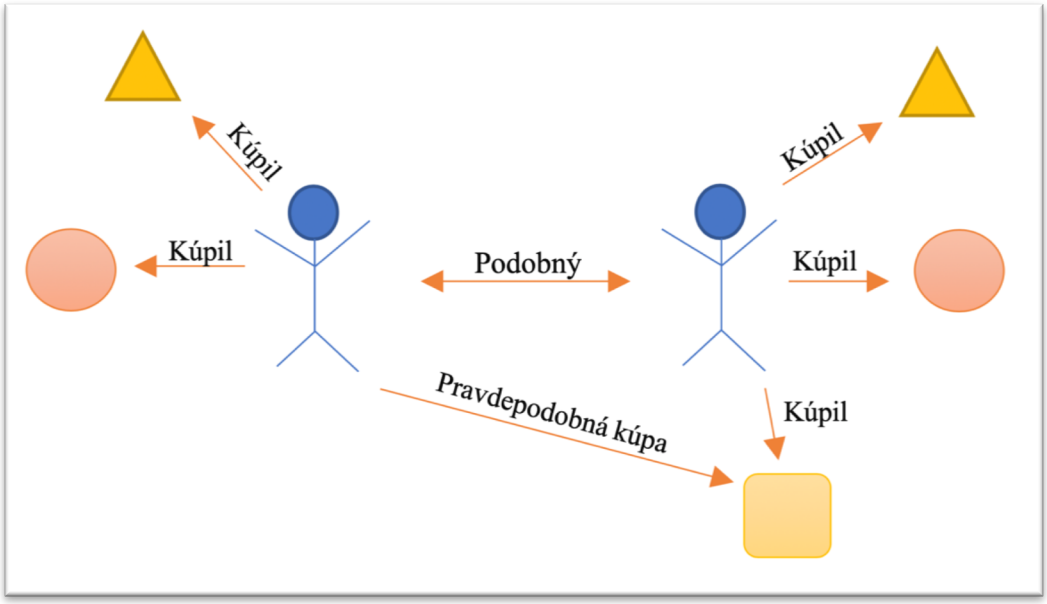
1. Obsah

2. Úvod

Úlohou zadania bolo vytvorenie odporúčania pre zákazníkov ktorý nakupujú v obchode, z ktorého sme dostali dataset. Tieto dáta obsahovali videné a kúpené produkty. Toto zadanie bolo riešené v jazyku Python, na ktorý som použil prostredie jupyter notebook. Dataset som spracoval pomocou knižnice Pandas a knižnica Sklearn mi umožnila využívať k-NN. Nižšie je sú ukázané knižnice, ktoré som v tejto práci využil:

```
from scipy import sparse
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
from pandas.api.types import CategoricalDtype
from sklearn.neighbors import NearestNeighbors
from datetime import datetime
from tqdm import tqdm
from pandas import DataFrame
import pandas as pd
import numpy as np
```

Na tvorbu odporúčania som využil Collaborative Filtering založené na User-base (viď obrázok nižšie)



Obrázok 1 Colaborative Filtering user-base

3. Načítanie a čistenie dát

Základné informácie o datasete

Dataset ktorý sme dostali, bol pomerne veľký. Skladal sa z dvoch súborov a to:

- Purchases_train.csv
    - počet záznamov: 188 712
    - veľkosť súboru: 36MB
- Events_train.csv
    - počet záznamov: 14 614 38
    - veľkosť súboru: 2,77GB

Oba tieto súbory obsahovali dáta s rovnakými stĺpcami a to customer_id, timestamp, event_type, product_id, title, category_name a price. Spolu sa jednalo o dataset s počtom 14 803 097 záznamov. Ako prvé bolo potrebné získať informácie o dátach. Keďže sme z prednášok daného predmetu získali prvotné informácie o stĺpcoch, a v rámci predmetu prebieha taktiež súťaž, ktorá má svoje pravidlá a sleduje predovšetkým product_id. Rozhodol som sa používať iba stĺpce:

- customer_id – id predstavujúceho customera
- timestamp – čas kedy vznikol daný záznam
- event_type – hodnota prezentujúca či bol produkt zakúpený alebo len prehliadaný
    - purchase_item – zakúpený produkt
    - view_item – prehliadaný produkt
- product_id – id predstavujúce produkt.

customer_id	timestamp	event_type	product_id
0d1b7397-7d3c-44c0-9efc-d38bf197828b	2019-07-01 00:05:54.308966 UTC	purchase_item	954f9f2c-d3ca-4236-ac9a-4ea7bcf09305
81ccae7e-e496-4997-a289-4669bf53f33e	2019-07-01 00:20:03.404186 UTC	purchase_item	75b281e5-8a16-42cb-9ae0-9a98db7a2c40
50777c55-8dd6-4309-a5ca-26e66c8a8279	2019-07-01 00:34:35.989935 UTC	purchase_item	0112dec8-47f5-4c2c-9109-571e2dbb6345
50777c55-8dd6-4309-a5ca-26e66c8a8279	2019-07-01 00:34:35.991935 UTC	purchase_item	0fa25a2d-2aa1-4397-82f1-5a64f3b1272d
647d269f-b18f-4558-8653-93369d862ec9	2019-07-01 00:52:53.083698 UTC	purchase_item	81c01216-55a9-4588-a722-bccf0bf35fd5
52eee654-4d66-47ce-9b8e-b6f3a775fb37	2019-07-01 01:12:33.417007 UTC	purchase_item	bd9a157e-7bc8-4f7c-a466-5385f96e48ed
52eee654-4d66-47ce-9b8e-b6f3a775fb37	2019-07-01 01:12:33.419006 UTC	purchase_item	a6cc55b8-04d7-42b0-86d7-b157b2cbd45f
52eee654-4d66-47ce-9b8e-b6f3a775fb37	2019-07-01 01:12:33.421006 UTC	purchase_item	a15a49bf-297b-4be9-a534-1b1eabe441a8
e7f1e5f4-cef2-421d-8aaf-f2f873ec7c19	2019-07-01 01:28:21.032258 UTC	purchase_item	edbb2521-5c0e-4b3b-b259-6dfc86c6b423
c6e8161d-0dfb-40c9-9a1c-95d2786dded6	2019-07-01 01:39:22.923531 UTC	purchase_item	8f2f533f-5700-4839-a978-3d99c93d8950
39944a34-0427-439d-9a4e-533a680d4b34	2019-07-01 01:56:53.681744 UTC	purchase_item	e95855d5-fa06-4501-bcc0-b5008c56519b

Obrázok 2 Ukážka dát

Základné informácie o dátach

Následne bolo nutné získať informácie o dátach. Na začiatok som zistil počty unikátnych zákazníkov a produktov:

Dataset	Typ	Počet
purchases_train	Počet unikátnych zakazníkov	71 566
events_train	Počet unikátnych zakazníkov	2 210 171
spolu	Počet unikátnych zakazníkov	2 211 386
purchases_train	Počet unikátnych produktov	33 613
events_train	Počet unikátnych produktov	123 724
Spolu	Počet unikátnych produktov	123 837

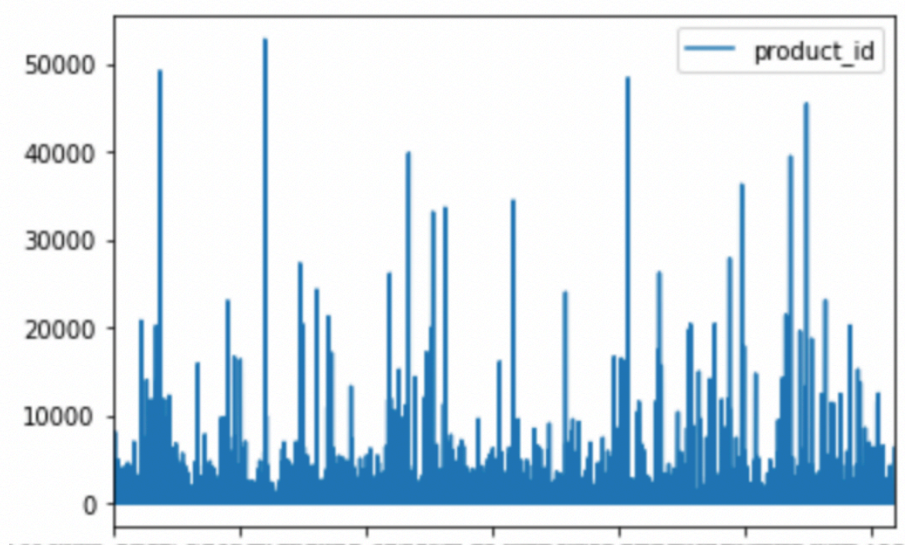
Tabuľka 1 Unikátne hodnoty

Potom som sa pozrel koľko zákazníkov zakúpilo/prezeralo aký počet produktov:

Počet záznamov	events_train	purchases_train	Spolu
1	915242	30843	913643
2	660220	35058	659174
3	598080	24684	597525
4	519336	22480	518884
5	486945	13150	485995
6	434868	13236	434964
7	399560	7266	399378
8	362288	7912	362408
9	341181	5058	342324
10	316090	4800	315480
11 a viac	9580575	24225	9773322

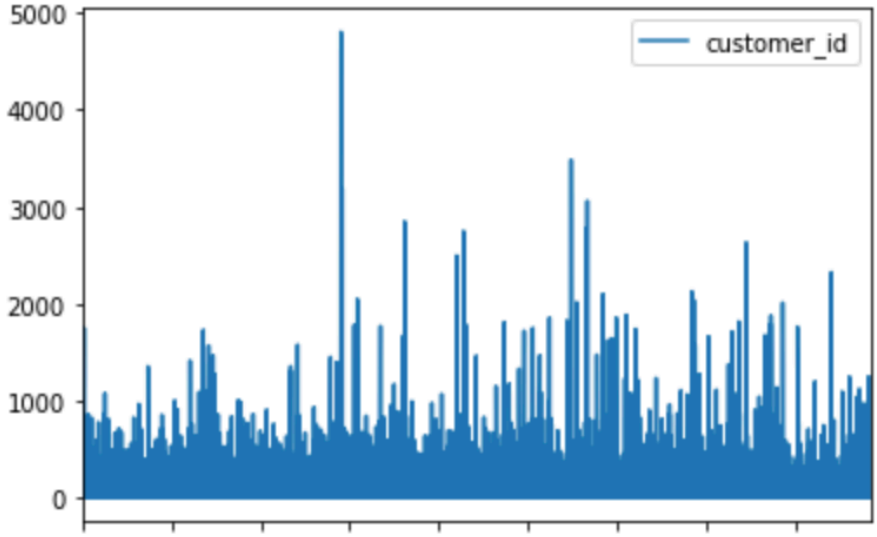
Tabuľka 2 Počet záznamov na jedného používateľa

Čo sa týka počtu produktov, obr. č.2 hovorí o tom, že je pár produktov, ktoré boli veľmi žiadané.



Obrázok 3 Počet, koľko krát sa produkt nachádza v datasete

Taktiež je z obr. č.3 vidieť že existuje veľa zákazníkov, ktorý majú v datasete viac ako 1000 záznamov.



Obrázok 4 Počet, koľko krát sa zákazník nachádza v datasete

Na základe získaných informácií som sa rozhodol odstrániť zákazníkov, ktorý sa nachádzali v datasete menej ako 5 krát kvôli časovej zložitosti. Ďalšie úpravy sú spomenuté v testovacej časti.

```
counts = df3['customer_id'].value_counts()
df3 = df3[~df3['customer_id'].isin(counts[counts < 5].index)]
```

4. Tvorba odporúčaní

Príprava dát pre odporúčania

Najväčšou prekážkou bolo vytvorenie matice, ktorá by obsahovala hodnoty, ktoré sčítavajú hodnotu produktu (koľko krát to videl-1b/kúpil-25b). Vyzerala by nasledovne:

-	Produkt 1	Produkt 2	.....	Produkt x
Customer 1	1	25	.....	Hodnota produktu x pre zákazníka 1
Customer 2	26	0	.....	Hodnota produktu x pre zákazníka 2
.....	.....	.....	.....	.....
Customer y	Hodnota produktu 1 pre zákazníka y	Hodnota produktu 2 pre zákazníka y	.....	Hodnota produktu x pre zákazníka y

Obrázok 5 Zobrazenie predstavy ako by boli uložené dáta

Tvorba tejto matice bola problematická kvôli nedostatku miesta na operačnej pamäti. Riešením nakoniec bolo, vytvoriť sparse_matrix ktorá neukladá hodnoty s obsahom 0. Keďže nie každý zákazník pristúpil ku každému produktu. Nie je potrebné tieto informácie ukladať. Teda je pre nás podstatné iba to, že ktoré produkty boli pre neho zaujímavé.

```
train_row = train.customer_id.astype(train_customer).cat.codes
train_col = train.product_id.astype(train_product).cat.codes
train_sparse_matrix = csr_matrix((train["event_type"], (train_row,
train_col)), shape=(train_customer.categories.size,
train_product.categories.size))
```

Čo sa týka odporúčania. Tu som využil algoritmus K-NN. Na začiatku som si vytvoril sparse maticu, ktorá obsahovala všetkých užívateľov a hodnoty (vzdialenosti), nakoľko sú si používatelia podobný. Teda vyzeralo to nejako takto (avšak v sparse matrix by neboli tie 0 pre ukážku ich nechám):

-	Customer 1	Customer 2	.....	Customer x
Customer 1	0	3	.....	Vzdialenosť zákazníka 1 pre zákazníka y
Customer 2	3	0	.....	Vzdialenosť zákazníka 2 pre zákazníka y
.....	.....	.....	.....	.....
Customer y	Vzdialenosť zákazníka 1 pre zákazníka y	Vzdialenosť zákazníka 2 pre zákazníka y	.....	Vzdialenosť zákazníka x pre zákazníka y

Obrázok 6 Zobrazenie matice zákazníkov a ich vzdialeností

Tvorba odporúčania

Na začiatku si rozdelím dataset na trénovaciu a testovaciu množinu v pomere 80:20. Keďže sú dáta veľmi veľké a k-NN trvá veľmi dlho, rozhodol som sa rozdeliť trenovaciu množinu ešte na 85% a 15%. V tom prípade bolo v trénovacej množine1353 unikátnych zákazníkov. Na rozdelenie som využíval train_test_split. Pri odporúčaní prechádzam každého unikátneho používateľa, ktorý je v testovacej množine. Skúšam zistiť či existuje zákazník s takým ID v trenovacej množine. Ak takýto zákazník neexistuje. Pridám mu zoznam top 10 produktov. Ak existuje, nájdem k (k môže byť ľubovoľné číslo) najpodobnejších susedov, ku ním nájdem ich produkty a všetky vložím do jedného zoznamu. Tu nastáva veľký problém, keďže ak daného zákazníka nemám v trenovacej množine a pridávam mu dáta len z top 10 produktov, toto odporúčanie sa stáva veľmi náhodné a títo zákazníci nám znižujú presnosť odporúčania.

V zozname uchovávam taktiež hodnoty produktov. Následne urobím groupby nad product_id, čím spočítam jednotlivé hodnoty produktov, ak sú v zozname nejaké rovnaké. Následne odstránim z tohoto zoznamu všetky produkty, ktorých hodnota u zákazníka, pre ktorého to odporúčam je väčšia ako 24. To je z toho dôvodu, že ak je táto hodnota 25 je veľký predpoklad že používateľ tento produkt už kúpil, avšak ak je jeho hodnota len 20. Tento produkt si určite nekúpil iba veľa krát pozeral a je dosť pravdepodobné že si ho ešte možno kúpi. Avšak do výsledného odporúčania sa dostane iba v prípade, že ostatný používatelia ho hodnotili viac ako ostatné produkty. Hodnotenie je určené pre kúpene produkty 25 a pre prezeraná 1.

Po pridaní tejto funkcionality sa skóre pri testovaní zvýšilo z približne 0,02 na 0,06. Po zostavení zoznamu produktov, ktoré chceme odporúčať, sa tento zoznam oreže na 10 produktov podľa hodnoty akú mu udelili zákazníci. Ak v zozname nie je dosť produktov, doplnia sa z top 10 produktov ktoré budú opísané nižšie. Tieto produkty sú už výsledné odporúčania pre zákazníka.

Vstupom do funkcie na odporúčanie je list zákazníkov. V testovacej vzorke sú len unikátny zákazníci.

5. Testovanie

Kaggel

Počas tvorby odporúčaní bola nápomocná aj súťaž na stránke koggel. Rozdiel medzi jednotlivými odovzdaniami bol najme z pridaním rôznych funkcionalít a nastavovaní parametrov.

Dátum a čas	Skóre	Vylepšenia
2019-11-26 01:44:49	0.01082	Základný odporúčač iba 10% dát
2019-11-26 18:21:28	0.01774	Zväčšenie počtu trenovacích dát
2019-11-28 09:15:18	0.01961	Odstránenie trénovacých dát ktoré majú menej ako 2 záznamy.
2019-11-29 08:16:00	0.02087	Zmena limitov hodnotenia videných a kúpených
2019-11-30 20:07:06	0.03068	Zväčšenie počtu trénovacých dát
2019-12-07 22:07:45	0.05381	Pridanie možnosti kúpiť už videný produkt

Obrázok 7 Odovzdania na súťaž Kaggel

Následne sa skúšali rôzne zmeny parametrov pričom sa skóre držalo okolo 0,066 až 0,075. Keď som následne odstránil odstránenie zákazníkov, ktorý mali menej ako x záznamov. A do trénovacej zložky vložil celý dataset. Skóre až výsledné skóre sa dostalo na hodnotu 0.13733 (Kaggel = 2019-12-14 18:29:48) Počet podobných používateľov ktorých hľadám bol 10.

Mean average precision (mAP)

Čo sa týka testovanie, využívam mAP (Mean average precision). Vypočíta sa ako podiel súčtu priemerných presností (average precision) a počtu hľadaní. Funkcia mAP sa vyhodnocuje pre viacero hľadaní. Priemerná presnosť pre konkrétne hľadanie sa získa podielom súčtov presností v jednotlivých bodoch vyhľadávania a celkovým počtom relevantných dokumentov. Funkcia nadobúda hodnoty z intervalu 0 - 1 vyššia hodnota značí relevantnejšie výsledky.

```
def map_score(k, Xrecommend_data, Xtest_data):
    global test_customer_uniq
    mapk = 0
    for u in range(len(test_customer_uniq)):
        y_true = Xtest_data[u]
        u_pred = Xrecommend_data[u]
        y_pred = np.sort(u_pred)[::-1][:k]
        n_hit = 0
```

```
        precision = 0
        actual = set(y_true)
        for i, p in enumerate(y_pred, 1):
            if p in actual:
                n_hit += 1
                precision += n_hit / i
        mapk += precision / min(len(actual), k)
    mapk /= len(test_customer_uniq)
    return mapk
```

Normalized Discounted Cumulative Gain (mDCG)

Ako ďalšiu metriku som zvolil NDCG aby som zvážil aj to ako kvalitné je odporúčanie z pohľadu postupnosti. V mojej implementácii pozerám na to, či poradie v dátach čo odporúčam sa zhodujúe v poradí, ktoré je v testovacej zložke. Ak je to tak, tak sa pridá hodnota 3 ak nie tak sa pridá iba hodnota 1 v prípade že sa našla aspoň nejaká zhoda s iným miestom.

```
def dcg_at_k(y_true, y_score, k = 10):
    y_dcg = np.zeros(len(y_true))
    for i in range(len(y_true)):

        for j in range(len(y_score)):
            if y_true[i] == y_score[j]:
                if i == j:
                    y_dcg[i] = 3
                y_dcg[i] = 1

    gains = 2 ** sum(y_dcg) - 1
    discounts = np.log2(np.arange(2, gains.size + 2))
    dcg = np.sum(gains / discounts)
    return dcg

def ndcg_score(Xrecommend_data, Xtest_data, k):
    global test_customer_uniq
    ndcg = 0.0
    for u in range(len(test_customer_uniq)):
        actual = dcg_at_k(sorted(Xtest_data[u], reverse=True),
Xrecommend_data[u], k)
        best = dcg_at_k(sorted(Xtest_data[u], reverse=True),
sorted(Xtest_data[u], reverse=True), k)
        ndcg += actual / best
    avg_ndcg = ndcg / len(test_customer_uniq)
    return avg_ndcg
```

Tabuľka nižšie zobrazuje jedno z testovaní ktoré bolo vykonané. V ňom išlo o zmenu počtu zákazníkov, podľa ktorých sa tvoril zoznam odporúčaní. Teda koľko najpodobnejších zákazníkov berieme do úvahy. Veľkosť trénovacej množiny bola 1353 zákazníkov.

Počet zakazníkov	mAP	nDCG
5	0.0558662028633961	0.03905552277718037
10	0.06356642920643127	0.044219790127584806
20	0.0691789407449493	0.05007445386501801
30	0.07053185655198824	0.052984721011700514
40	0.07370843735923104	0.05451764164022772
50	0.07472750056470547	0.055825160715393216

Vďaka testovaniu je jasne vidieť, že zväčšovaním počtu podobných používateľov sa zlepšuje výsledné skóre. Avšak po určitej hodnote sa toto skóre zväčšuje o menšie hodnoty (viď rozdiel medzi 10-20 a 40-50)

Úprava dát na základe testov

Pri testovaní bolo zistené že rozhodnutie odstrániť zákazníkov, ktorý majú iba 5 záznamov nebolo najvhodnejšie, dôvodom bolo zníženie výsledného skóre, no rapídne nám to ušetrí čas na výpočet keďže sa náš dataset zmenší. Keďže bol dataset pomerne veľký, rozhodol som sa odstrániť dáta staršie ako 8 dní. To nám taktiež umožnilo rýchlejšie spracovať dáta a rozdiel pri testovaní bol malí

```
df3['timestamp'] = pd.to_datetime(df3['timestamp'])
indexNames = df3[df3['timestamp'].max() + pd.Timedelta(days=-8) >
df3['timestamp']].index
df3.drop(indexNames , inplace=True)
```

Následne som odpozoroval že ak som pridal ďalšie dni, skóre na testoch sa zlepšilo. Veľmi dobré skóre som získaval pri 14 a 18 dňoch. Keďže som chcel pri odporúčaní, brať do úvahy aj to, či používateľ produkt pozrel alebo kúpil. Rozhodol som sa tieto hodnoty zameniť za čísla.

```
df3 = df3.replace('purchase_item', 25)
df3 = df3.replace('view_item', 1)
df3['event_type'] = df3['event_type'].astype(int)
```

Zaujímavé bolo sledovať, ako sa mení skóre aj v závislosti od toho, ako vysoko ohodnotím kúpenie produktu oproti prezretiu. Nakoniec som stanovil prezeranie produktu na hodnotu 1 a kúpenie produktu na hodnotu 25. Ak som to zmenil na 15 alebo 10 skóre sa znížilo.

Po rozdelení dát na testovaciu a trénovaciu množinu:

```
train, test = train_test_split(df3, test_size=0.2)
```

som ešte raz aplikoval odstránenie používateľov ktorý majú menej prezeraní / kúpení produktu ako 5.
Pri odporúčaní treba predpokladať že prídu aj používatelia, ktorý nemajú žiaden nákup ani prezeranie produktov. Tým pádom týmto používateľom odporúčam top 10 produktov

```
train_top_10_product =
train.groupby(['product_id']).sum().sort_values(by=['event_type'],ascending
=False).reset_index().head(10)
```

Týchto top 10 som získal ako súčet hodnôt, ktoré reprezentujú, či daný používateľ kúpil, alebo prezeral produkt. Následne som ich zoradil pomocou získaného súčtu a vybral som top 10 z nich.

6. Zhodnotenie

Odporúčania, ktoré vygeneroval program sú pomerne presné. Najvyššie získané skóre bolo 0.13733 na Kaggly a 0.125 na vlastnom nDCG. Toto bolo získané pri pomalšom riešení (veľmi veľa dát). Zatiaľ čo pri niektorých pokusoch, ktoré odstránili veľké množstvo datasetu kôli zrýchleniu, sa toto skóre síce znížilo na cca polovicu. Avšak rýchlosť odporúčania sa znížila o 2/5 času. Toto zistenie viedlo k tomu, že zákazník sa musí rozhodnúť či potrebuje vysokú rýchlosť, alebo mu ide o najpresnejšie odporúčanie.

7. Príloha A – Tvorba odporúčaní

```
def find_recommend(users_data):
    result_recommend =
pd.DataFrame(columns=['customer_id','product_id'])
    for x in tqdm(range(len(users_data))):
        try:
            index_value = train_customer_uniq.index(users_data[x])
        except ValueError:
            index_value = -1
        top10_for_user = train_top_10_product
        top10_for_user['customer_id'] = users_data[x]
        if index_value == -1:
            result_recommend = result_recommend.append(top10_for_user)
        else:
            spam, user_neighbours =
user_knn_distance_matrix[index_value].nonzero()
            user_for_recommend = train.loc[train['customer_id'] ==
train_customer_uniq[index_value]].iloc[:,[3,2]]
            user_for_recommend['customer_id'] = users_data[x]
            user_for_recommend =
user_for_recommend[user_for_recommend.event_type > 24] #pozor
            topNuser = 10
            if len(user_neighbours) < topNuser :
                topNuser = len(user_neighbours)
            for i in range(topNuser):
                if i == 0:
                    recommend = train.loc[train['customer_id'] ==
train_customer_uniq[user_neighbours[i]]].iloc[:,[2,3]]
                else:
                    recommend =
recommend.append(train.loc[train['customer_id'] ==
train_customer_uniq[user_neighbours[i]]].iloc[:,[2,3]])
            recommend = recommend.groupby(['product_id'],
as_index=False).sum()
            recommend['customer_id'] = users_data[x]
            if 'event_type' in recommend.columns:
                recommend =
recommend.sort_values(by=['event_type'],ascending=False)
            recommend.reset_index(drop=True, inplace=True)
            recommend = recommend.append( user_for_recommend
).drop_duplicates(subset='product_id', keep=False).head(10)
            r, c = recommend.shape
            if r < 11:
                recommend = recommend.append(top10_for_user).head(10)
            result_recommend = result_recommend.append(recommend)
    return result_recommend
```