Structured programming opdracht 3

**Op basis van meest gekochte producten (winkel-wijd) ( postgres\_algorithm\_mostPurchased.py)**

Hier wordt door alle orders geïtereerd, en word bij gehouden hoe vaak ieder product is verkocht.

Dit wordt gedaan door ieder product id op te slaan in een lijst (product\_list), en hoe vaak hij is verkocht wordt opgeslagen in een tweede lijst (counter\_list). De index van het product-id in de eerste lijst komt overeen met zijn frequentie in de tweede lijst.

De lijst wordt daarna gesorteerd op meest naar minst verkochte id. Dit wordt gedaan door telkens het grootste getal uit de counter\_list te pakken, en deze samen met zijn id uit product\_list op te slaan in een nieuwe lijst.

Na het sorteren wordt de lijst in de database geladen, zodat bij het laden van de recommendations niet het hele proces nog een keer hoeft te gebeuren (dit leek ons onnodig, gezien de meest verkochte producten niet zo fluctueren).

**Function: getMostPurchasedProducts**

**Input: The amount of popular products you want to get out of the dataset. (10)**

**Output: The most purchased products. The amount you get from this function is equal to the input.**

Initialize query to search for product\_id’s in previous orders.

row = fetch the first row

While row is not None

If product\_id not in product\_list

add product\_id to product\_list

add 1 to counter\_list

else

add 1 to  the counter\_list on the right index

Fetch the next row

x = 0

While x < 10

Add the most sold product to ret\_list

remove the most sold count from counter\_list

remove the most sold product from product\_list

x +=1

return ret\_list

**Op basis van vergelijkbare visitors (postgres\_algorithm\_similarVisitor.py)**

Allereerst wordt er een profiel opgesteld van de gebruiker op basis van zijn id. Hiervan worden de orders gebruikt en afgewogen tegen de orders van alle andere visitors om te kijken welke andere gebruikers vergelijkbaar koopgedrag tonen. Van de gebruikers die het meest vergelijkbare gedrag vertonen worden producten die zij veel kopen gepakt, en deze worden aangeraden. Als een vergelijkbare gebruiker een product heeft gekocht wat de gebruiker eerder heeft bekeken, word deze met prioriteit aangeraden.

**Function: getMostSimilarVisitors\_order\_based**

**Input: visitor id and what the visitor previously ordered**

**Output: the 5 most similar profiles, order based**

Initialize query to search for visitorid in previous orders.

row = fetch the first row

while row is not None

if buid not in visitor buid and order in visitor\_previously\_ordered and order != none\

if buid not in buid\_list

add buid to buid\_list

add 1  to buid\_counter\_list

else

add 1 to  the counter\_list on the right index

fetch the next row

x = 0

while x < 5

add the most similar buid to most\_similar\_buid

remove the highest buid count from buid\_counter\_list

remove the most similar buid from buid\_list

Initialize query to search for visitor\_buid

row =  fetch the first row

while row is not None

if visitor\_id in buid\_similar\_list

add visitor\_buid to ret\_list

Fetch the next row

return ret\_list

**Function: recommendProductsFromSimilarProfile**

**Input: Visitor profile, similar profile, the size of parent list( so we know how many recommendations are made.)**

**Output: A list with products to recommend from a similar visitor profile**

For x in all the orders of a similar profile

for y in all the orders of similar profile. ( lists in lists)

if similar\_profile[orders] in visitor\_profile[viewed\_before]

add the product id to ret\_list

else

add the product id to temp\_list

Randomize temp\_list

while ret\_list + parent\_ret\_list\_size <= 10

if temp\_list is None

break

add temp\_list to ret\_list untill it’s length is 10.

Return ret\_list

**Function:** getRecommendationFromSimilarVisitors

**Input: Visitor iD**

**Output: List with recommendation from similar profiles.**

**Op basis van vergelijkbare producten (postgres\_algorithm\_similarProduct.py)**

Van een gegeven product id wordt gekeken welke andere producten het meest vergelijkbaar zijn op basis van hun properties en categorie.

Er wordt een lijst met producten van dezelfde categorie opgesteld. Hierbij wordt gekeken naar ((sub-)sub-) categorie, en gender (mits geen unisex). Deze worden teruggegeven.

Ten tweede wordt er een lijst met producten vergelijkbare properties opgesteld: Eerst worden alle properties van het product die niet leeg zijn bij elkaar opgeteld, dit wordt later gebruikt als een target voor hoe vergelijkbaar een ander product moet zijn. Dan worden alle andere producten bekeken en de producten die vergelijkbaar genoeg zijn (target\*weight) worden teruggegeven.

De twee lijsten worden samengevoegd, dan worden de id's in de lijst uniek gemaakt, dan geshuffeld, en als laatst worden de eerste 10 elementen teruggegeven als recommendation.

**Function: getSimilarProducts**

**Input: Product\_id from a product.**

**Output: 10 random similar products from a list.**

list\_a = getProductsWithSameCategory.

list\_b = getProductsWithSameProperties

list\_ab = list\_a + list\_b

cleanUpColorDuplicates(list\_ab)

temp\_set = Creates a set with unique ID’s

For All ID’s in temp\_set

add the ID’s to ret\_list

Shuffle the list

ret\_list = the first 10 items after it’s shuffled

Make ret\_list ready to use on the front-end with getProductsFromList

Return ret\_list

**function getProductsWithSameCategory:**

**Input: product\_id from a product.**

**Output: a list with all the products that have the same category.**

Initialize query to search for the input product\_id in the database

row = fetch the first row

While row is not None:

If product\_id = input:

Save this information in product

else :

fetch the next row

Initialize the query again to compare the sub\_sub\_category from the saved data with other products.

row = fetch the first row

While row is not None:

If row[sub\_sub\_category] == product[sub\_sub\_category] and row != input

   and row[gender] == product[gender] or row[5] == unisex

   add to ret\_list

Fetch the next row

return ret\_list

**Function: getProductsWithSameProperties**

**Input: product\_id from a product.**

**Output:  A list with all the products that share 30% of the properties.**

Initialize query to search for the input product\_id in the database

row = fetch the first row

While row is not None:

If product\_id = input

Save this information in product

else fetch the next row

else: fetch the next row

target points = amount of not empty properties of input product

for x in the second to the last index’ of input product

if product property != None

Target\_points + 1

Initialize the query again to compare the input product with the rest of the products\

row = fetch the first row

While row is not None:

point = amount of not empty properties of the product that is fetched

for x in the second to the last index’ of input product

if row[x] == product[x] and row[x[ != None

Points + 1

if points >= (target\_points \* 0.3)

Add the product\_id from the product it’s currently comparing to ret\_list

Fetch the next row

return ret\_list