Amazon Apparel Recommendations

[4.2] Data and Code:

https://drive.google.com/open?id=0BwNkduBnePt2VWhCYXhMV3p4dTg (https://drive.google.com/open?id=0BwNkduBnePt2VWhCYXhMV3p4dTg)

[4.3] Overview of the data

In [1]:

```
#import all the necessary packages.

from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import math
import time
import re
import os
import seaborn as sns
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import pairwise_distances
from matplotlib import gridspec
from scipy.sparse import hstack
import plotly
import plotly.figure_factory as ff
from plotly.graph_objs import Scatter, Layout

plotly.offline.init_notebook_mode(connected=True)
warnings.filterwarnings("ignore")
```

In [2]:

```
# we have give a json file which consists of all information about
# the products
# loading the data using pandas' read_json file.
data = pd.read_json('tops_fashion.json')
```

In [3]:

```
print ('Number of data points : ', data.shape[0], \
       'Number of features/variables:', data.shape[1])
```

```
Number of data points :  183138 Number of features/variables: 19
```

Terminology:

What is a dataset?

Rows and columns

Data-point

Feature/variable

In [4]:

```
# each product/item has 19 features in the raw dataset.
data.columns # prints column-names or feature-names.
```

Out[4]:

```
Index(['asin', 'author', 'availability', 'availability_type', 'brand',
'color',
       'editorial_reivew', 'editorial_review', 'formatted_price',
       'large_image_url', 'manufacturer', 'medium_image_url', 'model',
       'product_type_name', 'publisher', 'reviews', 'sku', 'small_imag
e_url',
       'title'],
      dtype='object')
```

Of these 19 features, we will be using only 6 features in this workshop.

```
1. asin  ( Amazon standard identification number)
2. brand ( brand to which the product belongs to )
3. color ( Color information of apparel, it can contain many colors as   a
 value ex: red and black stripes )
4. product_type_name (type of the apperal, ex: SHIRT/TSHIRT )
5. medium_image_url  ( url of the image )
6. title (title of the product.)
7. formatted_price (price of the product)
```

In [5]:

```
data = data[['asin', 'brand', 'color', 'medium_image_url', 'product_type_name', 'ti
```

In [6]:

```
print ('Number of data points : ', data.shape[0], \
       'Number of features:', data.shape[1])
data.head() # prints the top rows in the table.
```

```
Number of data points :  183138 Number of features: 7
```

Out[6]:

	asin	brand	color	medium_image_url	product_type_name	title	fori
0	B016I2TS4W	FNC7C	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	Minions Como Superheroes Ironman Long Sleeve R...	
1	B01N49AI08	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	FIG Clothing Womens Izo Tunic	
2	B01JDPCOHO	FIG Clothing	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	FIG Clothing Womens Won Top	
3	B01N19U5H5	Focal18	None	https://images-na.ssl-images-amazon.com/images...	SHIRT	Focal18 Sailor Collar Bubble Sleeve Blouse Shi...	
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featherlite Ladies' Long Sleeve Stain Resistan...	

[5.1] Missing data for various features.

Basic stats for the feature: product_type_name

In [7]:

```
# We have total 72 unique type of product_type_names
print(data['product_type_name'].describe())

# 91.62% (167794/183138) of the products are shirts,
```

```
count     183138
unique        72
top        SHIRT
freq      167794
Name: product_type_name, dtype: object
```

In [8]:

```
# names of different product types
print(data['product_type_name'].unique())
```

```
['SHIRT' 'SWEATER' 'APPAREL' 'OUTDOOR_RECREATION_PRODUCT'
 'BOOKS_1973_AND_LATER' 'PANTS' 'HAT' 'SPORTING_GOODS' 'DRESS' 'UNDERW
EAR'
 'SKIRT' 'OUTERWEAR' 'BRA' 'ACCESSORY' 'ART_SUPPLIES' 'SLEEPWEAR'
 'ORCA_SHIRT' 'HANDBAG' 'PET_SUPPLIES' 'SHOES' 'KITCHEN' 'ADULT_COSTUM
E'
 'HOME_BED_AND_BATH' 'MISC_OTHER' 'BLAZER' 'HEALTH_PERSONAL_CARE'
 'TOYS_AND_GAMES' 'SWIMWEAR' 'CONSUMER_ELECTRONICS' 'SHORTS' 'HOME'
 'AUTO_PART' 'OFFICE_PRODUCTS' 'ETHNIC_WEAR' 'BEAUTY'
 'INSTRUMENT_PARTS_AND_ACCESSORIES' 'POWERSPORTS_PROTECTIVE_GEAR' 'SHI
RTS'
 'ABIS_APPAREL' 'AUTO_ACCESSORY' 'NONAPPARELMISC' 'TOOLS' 'BABY_PRODUC
T'
 'SOCKSHOSIERY' 'POWERSPORTS_RIDING_SHIRT' 'EYEWEAR' 'SUIT'
 'OUTDOOR_LIVING' 'POWERSPORTS_RIDING_JACKET' 'HARDWARE' 'SAFETY_SUPPL
Y'
 'ABIS_DVD' 'VIDEO_DVD' 'GOLF_CLUB' 'MUSIC_POPULAR_VINYL'
 'HOME_FURNITURE_AND_DECOR' 'TABLET_COMPUTER' 'GUILD_ACCESSORIES'
 'ABIS_SPORTS' 'ART_AND_CRAFT_SUPPLY' 'BAG' 'MECHANICAL_COMPONENTS'
 'SOUND_AND_RECORDING_EQUIPMENT' 'COMPUTER_COMPONENT' 'JEWELRY'
 'BUILDING_MATERIAL' 'LUGGAGE' 'BABY_COSTUME' 'POWERSPORTS_VEHICLE_PAR
T'
 'PROFESSIONAL_HEALTHCARE' 'SEEDS_AND_PLANTS' 'WIRELESS_ACCESSORY']
```

In [9]:

```
# find the 10 most frequent product_type_names.
product_type_count = Counter(list(data['product_type_name']))
product_type_count.most_common(10)
```

Out[9]:

```
[('SHIRT', 167794),
 ('APPAREL', 3549),
 ('BOOKS_1973_AND_LATER', 3336),
 ('DRESS', 1584),
 ('SPORTING_GOODS', 1281),
 ('SWEATER', 837),
 ('OUTERWEAR', 796),
 ('OUTDOOR_RECREATION_PRODUCT', 729),
 ('ACCESSORY', 636),
 ('UNDERWEAR', 425)]
```

Basic stats for the feature: brand

In [10]:

```
# there are 10577 unique brands
print(data['brand'].describe())

# 183138 - 182987 = 151 missing values.
```

```
count     182987
unique     10577
top         Zago
freq         223
Name: brand, dtype: object
```

In [11]:

```
brand_count = Counter(list(data['brand']))
brand_count.most_common(10)
```

Out[11]:

```
[('Zago', 223),
 ('XQS', 222),
 ('Yayun', 215),
 ('YUNY', 198),
 ('XiaoTianXin-women clothes', 193),
 ('Generic', 192),
 ('Boohoo', 190),
 ('Alion', 188),
 ('Abetteric', 187),
 ('TheMogan', 187)]
```

Basic stats for the feature: color

In [12]:

```
print(data['color'].describe())


# we have 7380 unique colors
# 7.2% of products are black in color
# 64956 of 183138 products have brand information. That's approx 35.4%.
```

```
count     64956
unique     7380
top       Black
freq      13207
Name: color, dtype: object
```

In [13]:

```
color_count = Counter(list(data['color']))
color_count.most_common(10)
```

Out[13]:

```
[(None, 118182),
 ('Black', 13207),
 ('White', 8616),
 ('Blue', 3570),
 ('Red', 2289),
 ('Pink', 1842),
 ('Grey', 1499),
 ('*', 1388),
 ('Green', 1258),
 ('Multi', 1203)]
```

Basic stats for the feature: formatted_price

In [14]:

```
print(data['formatted_price'].describe())

# Only 28,395 (15.5% of whole data) products with price information
```

```
count      28395
unique      3135
top       $19.99
freq         945
Name: formatted_price, dtype: object
```

In [15]:

```
price_count = Counter(list(data['formatted_price']))
price_count.most_common(10)
```

Out[15]:

```
[(None, 154743),
 ('$19.99', 945),
 ('$9.99', 749),
 ('$9.50', 601),
 ('$14.99', 472),
 ('$7.50', 463),
 ('$24.99', 414),
 ('$29.99', 370),
 ('$8.99', 343),
 ('$9.01', 336)]
```

Basic stats for the feature: title

In [16]:

```
print(data['title'].describe())

# All of the products have a title.
# Titles are fairly descriptive of what the product is.
# We use titles extensively in this workshop
# as they are short and informative.
```

```
count                                                 183138
unique                                                175985
top       Nakoda Cotton Self Print Straight Kurti For Women
freq                                                      77
Name: title, dtype: object
```

In [17]:

```
data.to_pickle('pickels/180k_apparel_data')
```

We save data files at every major step in our processing in "pickle" files. If you are stuck anywhere (or) if some code takes too long to run on your laptop, you may use the pickle files we give you to speed things up.

In [18]:

```
# consider products which have price information
# data['formatted_price'].isnull() => gives the information
#about the dataframe row's which have null values price == None|Null
data = data.loc[~data['formatted_price'].isnull()]
print('Number of data points After eliminating price=NULL :', data.shape[0])
```

```
Number of data points After eliminating price=NULL : 28395
```

In [19]:

```
# consider products which have color information
# data['color'].isnull() => gives the information about the dataframe row's which h
data =data.loc[~data['color'].isnull()]
print('Number of data points After eliminating color=NULL :', data.shape[0])
```

```
Number of data points After eliminating color=NULL : 28385
```

We brought down the number of data points from 183K to 28K.

We are processing only 28K points so that most of the workshop participants can run this code on thier laptops in a reasonable amount of time.

For those of you who have powerful computers and some time to spare, you are recommended to use all of the 183K images.

In [20]:

```
data.to_pickle('pickels/28k_apparel_data')
```

In [21]:

```
images=data['medium_image_url']
```

In [ ]:

```

```

In [ ]:

```
# You can download all these 28k images using this code below.
# You do NOT need to run this code and hence it is commented.


from PIL import Image
import requests
from io import BytesIO

for index, row in data.iterrows():
        try:
            url = row['medium_image_url']
            response = requests.get(url)
            img = Image.open(BytesIO(response.content))
            img.save('images/28k_images/'+row['asin']+'.jpeg')
        except OSError:
            pass
```

In [93]:

```
data.head()
```

Out[93]:

	asin	brand	color	medium_image_url	product_type_name	title	for
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	featherlite ladies long sleeve stain resistant...	
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	womens unique 100 cotton special olympics wor...	
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	featherlite ladies moisture free mesh sport sh...	
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	supernatural chibis sam dean castiel neck tshi...	
43	B0079BMKDS	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	APPAREL	featherlite ladies silky smooth pique white xl	

[5.2] Remove near duplicate items

[5.2.1] Understand about duplicates.

In [4]:

```
# read data from pickle file from previous stage
data = pd.read_pickle('pickels/28k_apparel_data')

# find number of products that have duplicate titles.
print(sum(data.duplicated('title')))
# we have 2325 products which have same title but different color
```

```
2325
```

These shirts are exactly same except in size (S, M,L,XL)

:B00AQ4GMCK :B00AQ4GMTS

:B00AQ4GMLQ :B00AQ4GN3I

These shirts exactly same except in color

:B00G278GZ6 :B00G278W6O

:B00G278Z2A :B00G2786X8

In our data there are many duplicate products like the above examples, we need to de-dupe them for better results.

[5.2.2] Remove duplicates : Part 1

In [5]:

```
# read data from pickle file from previous stage
data = pd.read_pickle('pickels/28k_apparel_data')
```

In [6]:

```
data.head()
```

Out[6]:

	asin	brand	color	medium_image_url	product_type_name	title	fo
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	Featherlite Ladies' Long Sleeve Stain Resistan...	
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	Women's Unique 100% Cotton T - Special Olympic...	
11	B001LOUGE4	Fitness Etc.	Black	https://images-na.ssl-images-amazon.com/images...	SHIRT	Ladies Cotton Tank 2x1 Ribbed Tank Top	
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	FeatherLite Ladies' Moisture Free Mesh Sport S...	
21	B014ICEDNA	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	Supernatural Chibis Sam Dean And Castiel Short...	

In [7]:

```
# Remove All products with very few words in title
data_sorted = data[data['title'].apply(lambda x: len(x.split())>4)]
print("After removal of products with short description:", data_sorted.shape[0])
```

```
After removal of products with short description: 27949
```

In [8]:

```
# Sort the whole data based on title (alphabetical order of title)
data_sorted.sort_values('title',inplace=True, ascending=False)
data_sorted.head()
```

Out[8]:

	asin	brand	color	medium_image_url	product_type_name	titl
61973	B06Y1KZ2WB	Éclair	Black/Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	Éclai Women' Printe Thin Stra Blous Black..
133820	B010RV33VE	xiaoming	Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaomin Women Sleeveles Loos Long T shirts..
81461	B01DDSDLNS	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaomin Women' Whit Lon Sleev Singl Brea..
75995	B00X5LYO9Y	xiaoming	Red Anchors	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaomin Stripe Tan Patch/Bea Sleev Anchor..
151570	B00WPJG35K	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	xiaomin Sleev Shee Loos Tasse Kimon Woma..

Some examples of dupliacte titles that differ only in the last few words.

```
Titles 1:
16. woman's place is in the house and the senate shirts for Womens XXL Whit
e
17. woman's place is in the house and the senate shirts for Womens M Grey

Title 2:
25. tokidoki The Queen of Diamonds Women's Shirt X-Large
```

```
26. tokidoki The Queen of Diamonds Women's Shirt Small
27. tokidoki The Queen of Diamonds Women's Shirt Large

Title 3:
61. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Anima
l Print Head Shirt for woman Neon Wolf t-shirt
62. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Anima
l Print Head Shirt for woman Neon Wolf t-shirt
63. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Anima
l Print Head Shirt for woman Neon Wolf t-shirt
64. psychedelic colorful Howling Galaxy Wolf T-shirt/Colorful Rainbow Anima
l Print Head Shirt for woman Neon Wolf t-shirt
```

In [9]:

```
indices = []
for i,row in data_sorted.iterrows():
    indices.append(i)
```

In [10]:

```
data_sorted
```

Out[10]:

	asin	brand	color	medium_image_url	product_type_name
61973	B06Y1KZ2WB	Éclair	Black/Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT
133820	B010RV33VE	xiaoming	Pink	https://images-na.ssl-images-amazon.com/images...	SHIRT
81461	B01DDSDLNS	xiaoming	White	https://images-na.ssl-images-amazon.com/images...	SHIRT

In [36]:

```
import itertools
stage1_dedupe_asins = []
i = 0
j = 0
num_data_points = data_sorted.shape[0]
while i < num_data_points and j < num_data_points:

    previous_i = i

    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Quee
    a = data['title'].loc[indices[i]].split()

    # search for the similar products sequentially
    j = i+1
    while j < num_data_points:

        # store the list of words of jth string in b, ex: b = ['tokidoki', 'The', '
        b = data['title'].loc[indices[j]].split()

        # store the maximum length of two strings
        length = max(len(a), len(b))

        # count is used to store the number of words that are matched in both strin
        count  = 0

        # itertools.zip_longest(a,b): will map the corresponding words in both stri
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c','d'), (
        for k in itertools.zip_longest(a,b):
            if (k[0] == k[1]):
                count += 1

        # if the number of words in which both strings differ are > 2 , we are cons
        # if the number of words in which both strings differ are < 2 , we are cons
        if (length - count) > 2: # number of words in which both sensences differ
            # if both strings are differ by more than 2 words we include the 1st st
            stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[i]])


            # start searching for similar apperals corresponds 2nd string
            i = j
            break
        else:
            j += 1
    if previous_i == i:
        break
```

In [28]:

```
data = data.loc[data['asin'].isin(stage1_dedupe_asins)]
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-28-f98a1a948378> in <module>()
----> 1 data = data.loc[data['asin'].isin(stage1_dedupe_asins)]

NameError: name 'stage1_dedupe_asins' is not defined
```

We removed the dupliactes which differ only at the end.

In [38]:

```
print('Number of data points : ', data.shape[0])
```

```
Number of data points :  17592
```

In [11]:

```
data.to_pickle('pickels/17k_apperal_data')
```

[5.2.3] Remove duplicates : Part 2

```
In the previous cell, we sorted whole data in alphabetical order of  title
s.Then, we removed titles which are adjacent and very similar title

But there are some products whose titles are not adjacent but very similar.

Examples:

Titles-1
86261.  UltraClub Women's Classic Wrinkle-Free Long Sleeve Oxford Shirt, Pi
nk, XX-Large
115042. UltraClub Ladies Classic Wrinkle-Free Long-Sleeve Oxford Light Blue
XXL

TItles-2
75004.  EVALY Women's Cool University Of UTAH 3/4 Sleeve Raglan Tee
109225. EVALY Women's Unique University Of UTAH 3/4 Sleeve Raglan Tees
120832. EVALY Women's New University Of UTAH 3/4-Sleeve Raglan Tshirt
```

In [12]:

```
data = pd.read_pickle('pickels/17k_apperal_data')
```

In [ ]:

```
# This code snippet takes significant amount of time.
# O(n^2) time.
# Takes about an hour to run on a decent computer.

indices = []
for i,row in data.iterrows():
    indices.append(i)

stage2_dedupe_asins = []
while len(indices)!=0:
    i = indices.pop()
    stage2_dedupe_asins.append(data['asin'].loc[i])
    # consider the first apperal's title
    a = data['title'].loc[i].split()
    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Quee
    for j in indices:

        b = data['title'].loc[j].split()
        # store the list of words of jth string in b, ex: b = ['tokidoki', 'The', '

        length = max(len(a),len(b))

        # count is used to store the number of words that are matched in both strin
        count  = 0

        # itertools.zip_longest(a,b): will map the corresponding words in both stri
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c','d'), (
        for k in itertools.zip_longest(a,b):
            if (k[0]==k[1]):
                count += 1

        # if the number of words in which both strings differ are < 3 , we are cons
        if (length - count) < 3:
            indices.remove(j)
```

In [ ]:

```
# from whole previous products we will consider only
# the products that are found in previous cell
data = data.loc[data['asin'].isin(stage2_dedupe_asins)]
```

In [36]:

```
print('Number of data points after stage two of dedupe: ',data.shape[0])
# from 17k apperals we reduced to 16k apperals
```

```
Number of data points after stage two of dedupe:  17592
```

In [37]:

```
data.to_pickle('pickels/16k_apperal_data')
# Storing these products in a pickle file
# candidates who wants to download these files instead
# of 180K they can download and use them from the Google Drive folder.
```

6. Text pre-processing

In [54]:

```
data = pd.read_pickle('pickels/16k_apperal_data')

# NLTK download stop words. [RUN ONLY ONCE]
# goto Terminal (Linux/Mac) or Command-Prompt (Window)
# In the temrinal, type these commands
# $python3
# $import nltk
# $nltk.download()
```

In [56]:

```
data = data[:16042]
```

In [57]:

```
# we use the list of stop words that are downloaded from nltk lib.
stop_words = set(stopwords.words('english'))
print ('list of stop words:', stop_words)

def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        for words in total_text.split():
            # remove the special chars in review like '"#$@!%^&*()_+-~?>< etc.
            word = ("".join(e for e in words if e.isalnum()))
            # Conver all letters to lower-case
            word = word.lower()
            # stop-word removal
            if not word in stop_words:
                string += word + " "
        data[column][index] = string
```

```
list of stop words: {'herself', "shan't", 're', 'here', 'on', 'off',
'where', 'such', 'haven', 'ourselves', 'which', 'his', 'into', 'unde
r', 'not', 'with', 'until', "needn't", 'me', 's', 'shouldn', 'ours',
'over', 'about', 'at', 'do', 'there', 'just', 'was', 'both', "have
n't", 'that', 'or', 'above', 'the', 'those', 'ma', 'shan', 'am', 'havi
ng', 'itself', 'through', 'mustn', 'y', 'further', 'myself', 'betwee
n', 'm', "shouldn't", 'when', 'them', 'should', 'isn', 'so', "don't",
'than', 'they', 'too', "you're", "doesn't", 'against', "isn't", "must
n't", 'be', 'is', 'yours', "hadn't", 'other', 'out', 'have', 'o', 'n
o', 'again', 'couldn', 'who', 'i', 'we', 've', "that'll", "you've", "y
ou'll", 'your', 'hers', 'why', 'if', "didn't", 'will', 'after', 'fro
m', 'he', "wouldn't", "aren't", 'to', 'doesn', 't', 'it', 'nor', 'was
n', 'as', 'of', 'during', 'doing', 'our', 'some', 'aren', 'won', "yo
u'd", 'd', "hasn't", 'she', "it's", 'now', "should've", 'mightn', 'ho
w', 'in', 'while', "weren't", 'most', "she's", 'but', 'and', 'my', 'ha
s', "couldn't", 'more', 'only', 'down', 'hadn', 'any', 'yourself', 'hi
m', "mightn't", 'weren', 'its', 'a', 'had', 'few', 'being', "wasn't",
'whom', 'because', 'own', 'can', "won't", 'up', 'hasn', 'himself', 'be
low', 'were', 'll', 'for', 'then', 'been', 'their', 'each', 'by', 'the
se', 'same', 'before', 'once', 'yourselves', 'you', 'very', 'did', 'he
r', 'theirs', 'themselves', 'all', 'didn', 'are', 'does', 'an', 'ain',
'what', 'this', 'needn', 'wouldn', 'don'}
```

In [58]:

```
start_time = time.clock()
# we take each title and we text-preprocess it.
for index, row in data.iterrows():
    nlp_preprocessing(row['title'], index, 'title')
# we print the time it took to preprocess whole titles
print(time.clock() - start_time, "seconds")
```

```
7.009756999999979 seconds
```

In [59]:

```
data.head()
```

Out[59]:

	asin	brand	color	medium_image_url	product_type_name	title	fo
4	B004GSI2OS	FeatherLite	Onyx Black/ Stone	https://images-na.ssl-images-amazon.com/images...	SHIRT	featherlite ladies long sleeve stain resistant...	
6	B012YX2ZPI	HX-Kingdom Fashion T-shirts	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	womens unique 100 cotton special olympics wor...	
15	B003BSRPB0	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	SHIRT	featherlite ladies moisture free mesh sport sh...	
27	B014ICEJ1Q	FNC7C	Purple	https://images-na.ssl-images-amazon.com/images...	SHIRT	supernatural chibis sam dean castiel neck tshi...	
43	B0079BMKDS	FeatherLite	White	https://images-na.ssl-images-amazon.com/images...	APPAREL	featherlite ladies silky smooth pique white xl	

In [60]:

```
data.to_pickle('pickels/16k_apperal_data_preprocessed')
```

Stemming

In [5]:

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
print(stemmer.stem('arguing'))
print(stemmer.stem('fishing'))


# We tried using stemming on our titles and it didnot work very well.
```

```
argu
fish
```

[8] Text based product similarity

In [6]:

```
data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
data.shape
```

Out[6]:

```
(16042, 7)
```

In [7]:

```
# Utility Functions which we will use through the rest of the workshop.


#Display an image
def display_img(url,ax,fig):
    # we get the url of the apparel and download it
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    # we will display it in notebook
    plt.imshow(img)

#plotting code to understand the algorithm's decision.
def plot_heatmap(keys, values, labels, url, text):
        # keys: list of words of recommended title
        # values: len(values) ==  len(keys), values(i) represents the occurence of
        # labels: len(labels) == len(keys), the values of labels depends on the moc
                # if model == 'bag of words': labels(i) = values(i)
                # if model == 'tfidf weighted bag of words':labels(i) = tfidf(keys(
                # if model == 'idf weighted bag of words':labels(i) = idf(keys(i))
        # url : apparel's url

        # we will devide the whole figure into two parts
        gs = gridspec.GridSpec(2, 2, width_ratios=[4,1], height_ratios=[4,1])
        fig = plt.figure(figsize=(25,3))

        # 1st, ploting heat map that represents the count of commonly ocurred words
        ax = plt.subplot(gs[0])
        # it displays a cell in white color if the word is intersection(lis of word
        ax = sns.heatmap(np.array([values]), annot=np.array([labels]))
        ax.set_xticklabels(keys) # set that axis labels as the words of title
        ax.set_title(text) # apparel title

        # 2nd, plotting image of the the apparel
        ax = plt.subplot(gs[1])
        # we don't want any grid lines for image and no labels on x-axis and y-axis
        ax.grid(False)
        ax.set_xticks([])
        ax.set_yticks([])

        # we call dispaly_img based with paramete url
        display_img(url, ax, fig)

        # displays combine figure ( heat map and image together)
        plt.show()

def plot_heatmap_image(doc_id, vec1, vec2, url, text, model):

    # doc_id : index of the title1
    # vec1 : input apparels's vector, it is of a dict type {word:count}
    # vec2 : recommended apparels's vector, it is of a dict type {word:count}
    # url : apparels image url
    # text: title of recomonded apparel (used to keep title of image)
    # model, it can be any of the models,
        # 1. bag_of_words
        # 2. tfidf
        # 3. idf
    # we find the common words in both titles, because these only words contribute
    intersection = set(vec1.keys()) & set(vec2.keys())

    # we set the values of non intersecting words to zero, this is just to show the
    for i in vec2:
        if i not in intersection:
            vec2[i]=0

    # for labeling heatmap, keys contains list of all words in title2
    keys = list(vec2.keys())
    #  if ith word in intersection(lis of words of title1 and list of words of titl
    values = [vec2[x] for x in vec2.keys()]

    # labels: len(labels) == len(keys), the values of labels depends on the model w
        # if model == 'bag of words': labels(i) = values(i)
        # if model == 'tfidf weighted bag of words':labels(i) = tfidf(keys(i))
        # if model == 'idf weighted bag of words':labels(i) = idf(keys(i))

    if model == 'bag_of_words':
        labels = values
    elif model == 'tfidf':
        labels = []
        for x in vec2.keys():
            # tfidf_title_vectorizer.vocabulary_ it contains all the words in the c
            # tfidf_title_features[doc_id, index_of_word_in_corpus] will give the t
            if x in  tfidf_title_vectorizer.vocabulary_:
                labels.append(tfidf_title_features[doc_id, tfidf_title_vectorizer.v
            else:
                labels.append(0)
    elif model == 'idf':
        labels = []
        for x in vec2.keys():
            # idf_title_vectorizer.vocabulary_ it contains all the words in the cor
            # idf_title_features[doc_id, index_of_word_in_corpus] will give the idf
            if x in  idf_title_vectorizer.vocabulary_:
                labels.append(idf_title_features[doc_id, idf_title_vectorizer.vocab
            else:
                labels.append(0)

    plot_heatmap(keys, values, labels, url, text)


# this function gets a list of wrods along with the frequency of each
# word given "text"
def text_to_vector(text):
    word = re.compile(r'\w+')
    words = word.findall(text)
    # words stores list of all words in given string, you can try 'words = text.spl
    return Counter(words) # Counter counts the occurence of each word in list, it r



def get_result(doc_id, content_a, content_b, url, model):
    text1 = content_a
    text2 = content_b

    # vector1 = dict{word11:#count, word12:#count, etc.}
    vector1 = text_to_vector(text1)

    # vector1 = dict{word21:#count, word22:#count, etc.}
    vector2 = text_to_vector(text2)
    plot_heatmap_image(doc_id, vector1, vector2, url, text2, model)
```

[8.2] Bag of Words (BoW) on product titles.

In [8]:

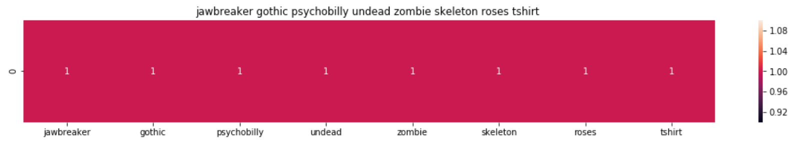
```
from sklearn.feature_extraction.text import CountVectorizer
title_vectorizer = CountVectorizer()
title_features   = title_vectorizer.fit_transform(data['title'])
title_features.get_shape() # get number of rows and columns in feature matrix.
# title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(corpus) returns
# the a sparase matrix of dimensions #data_points * #words_in_corpus

# What is a sparse vector?

# title_features[doc_id, index_of_word_in_corpus] = number of times the word occure
```

Out[8]:

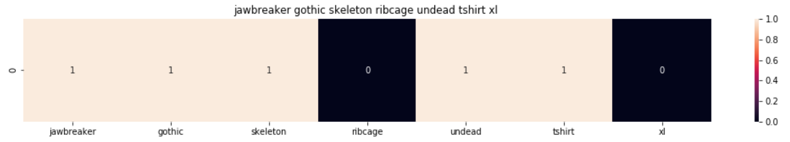
```
(16042, 12406)
```

In [9]:

```
def bag_of_words_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaini
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(title_features,title_features[doc_id])

    # np.argsort will return indices of the smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0,len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title'].loc[d
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print ('Brand:', data['brand'].loc[df_indices[i]])
        print ('Title:', data['title'].loc[df_indices[i]])
        print ('Euclidean similarity with the query image :', pdists[i])
        print('='*60)

#call the bag-of-words model for a product to get similar products.
bag_of_words_model(12566, 20) # change the index if you want to.
# In the output heat map each value represents the count value
# of the label word, the color represents the intersection
# with inputs title.

#try 12566
#try 931
```



```
ASIN : B01D3R5VXC
Brand: Jawbreaker
Title: jawbreaker gothic psychobilly undead zombie skeleton roses ts
hirt
Euclidean similarity with the query image : 0.0
============================================================
```



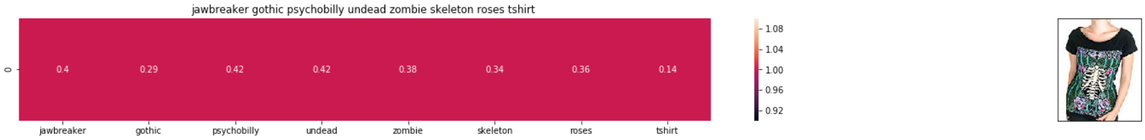
[8.5] TF-IDF based product similarity

In [10]:

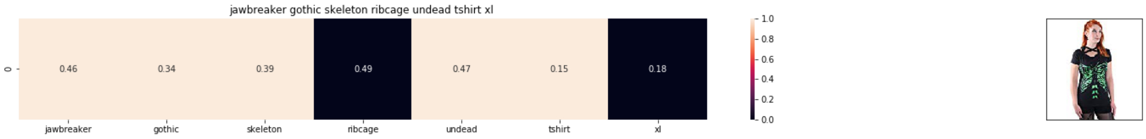
```
tfidf_title_vectorizer = TfidfVectorizer(min_df = 1)
tfidf_title_features = tfidf_title_vectorizer.fit_transform(data['title'])
# tfidf_title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(courpus) returns the a sparase matrix of dimensio
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the word
```

In [11]:

```
def tfidf_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaini
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(tfidf_title_features,tfidf_title_features[do

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0,len(indices)):
        # we will pass 1. doc_id, 2. title1, 3. title2, url, model
        get_result(indices[i], data['title'].loc[df_indices[0]], data['title'].loc[
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('Eucliden distance from the given image :', pdists[i])
        print('='*125)
tfidf_model(12566, 20)
# in the output heat map each value represents the tfidf values of the label word,
```



```
ASIN : B01D3R5VXC
BRAND : Jawbreaker
Eucliden distance from the given image : 0.0
====================================================================
=========================================================
```



```
ASIN : B072YV132Q
BRAND : Jawbreaker
```

[8.5] IDF based product similarity

In [12]:

```
idf_title_vectorizer = CountVectorizer()
idf_title_features = idf_title_vectorizer.fit_transform(data['title'])

# idf_title_features.shape = #data_points * #words_in_corpus
# CountVectorizer().fit_transform(courpus) returns the a sparase matrix of dimensic
# idf_title_features[doc_id, index_of_word_in_corpus] = number of times the word oc
```

In [13]:

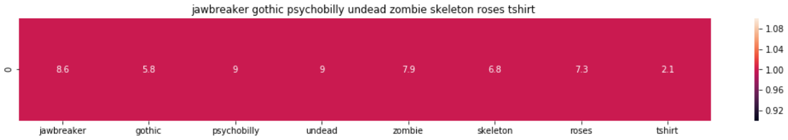
```
def n_containing(word):
    # return the number of documents which had the given word
    return sum(1 for blob in data['title'] if word in blob.split())

def idf(word):
    # idf = log(#number of docs / #number of docs which had the given word)
    return math.log(data.shape[0] / (n_containing(word)))
```

In [15]:

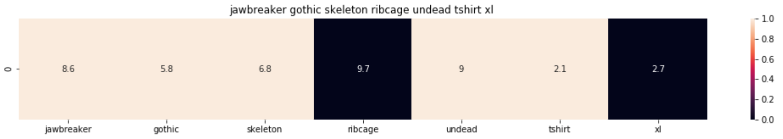
```
# we need to convert the values into float
idf_title_features  = idf_title_features.astype(np.float)

for i in idf_title_vectorizer.vocabulary_.keys():
    # for every word in whole corpus we will find its idf value
    idf_val = idf(i)

    # to calculate idf_title_features we need to replace the count values with the
    # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will
    for j in idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0

        # we replace the count values of word i in document j with  idf_value of wo
        # idf_title_features[doc_id, index_of_word_in_courpus] = idf value of word
        idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val
```

In [16]:

```
def idf_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaini
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(idf_title_features,idf_title_features[doc_id

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0,len(indices)):
        get_result(indices[i],data['title'].loc[df_indices[0]], data['title'].loc[d
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from the given image :', pdists[i])
        print('='*125)



idf_model(12566,20)
# in the output heat map each value represents the idf values of the label word, th
```



```
ASIN : B01D3R5VXC
Brand : Jawbreaker
euclidean distance from the given image : 0.0
====================================================================
=========================================================
```



```
ASIN : B072YV132Q
Brand : Jawbreaker
```

[9] Text Semantics based product similarity

In [23]:

```
# credits: https://www.kaggle.com/c/word2vec-nlp-tutorial#part-2-word-vectors
# Custom Word2Vec using your own text data.
# Do NOT RUN this code.
# It is meant as a reference to build your own Word2Vec when you have
# lots of data.

'''
# Set values for various parameters
num_features = 300    # Word vector dimensionality
min_word_count = 1    # Minimum word count
num_workers = 4       # Number of threads to run in parallel
context = 10          # Context window size
downsampling = 1e-3   # Downsample setting for frequent words

# Initialize and train the model (this will take some time)
from gensim.models import word2vec
print ("Training model...")
model = word2vec.Word2Vec(sen_corpus, workers=num_workers, \
            size=num_features, min_count = min_word_count, \
            window = context)

'''
```

Out[23]:

```
'\n# Set values for various parameters\nnum_features = 300    # Word v
ector dimensionality                      \nmin_word_count = 1    # Mi
nimum word count                        \nnum_workers = 4       # Numb
er of threads to run in parallel\ncontext = 10          # Context wind
ow size
\ndownsampling = 1e-3   # Downsample setting for frequent words\n\n# I
nitialize and train the model (this will take some time)\nfrom gensim.
models import word2vec\nprint ("Training model...")\nmodel = word2vec.
Word2Vec(sen_corpus, workers=num_workers,             size=num_feature
s, min_count = min_word_count,             window = context)\n    \n'
```

In [24]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.
'''
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', bin

'''
#if you do NOT have RAM >= 12GB, use the code below.
with open('word2vec_model', 'rb') as handle:
    model = pickle.load(handle)
```

In [25]:

```
# Utility functions

def get_word_vec(sentence, doc_id, m_name):
    # sentence : title of the apparel
    # doc_id: document id in our corpus
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of w
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(wor
    vec = []
    for i in sentence.split():
        if i in vocab:
            if m_name == 'weighted' and i in  idf_title_vectorizer.vocabulary_:
                vec.append(idf_title_features[doc_id, idf_title_vectorizer.vocabula
            elif m_name == 'avg':
                vec.append(model[i])
        else:
            # if the word in our courpus is not there in the google word2vec corpus
            vec.append(np.zeros(shape=(300,)))
    # we will return a numpy array of shape (#number of words in title * 300 ) 300
    # each row represents the word2vec representation of each word (weighted/avg) i
    return  np.array(vec)

def get_distance(vec1, vec2):
    # vec1 = np.array(#number_of_words_title1 * 300), each row is a vector of lengt
    # vec2 = np.array(#number_of_words_title2 * 300), each row is a vector of lengt

    final_dist = []
    # for each vector in vec1 we caluclate the distance(euclidean) to all vectors i
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between vector
            dist.append(np.linalg.norm(i-j))
        final_dist.append(np.array(dist))
    # final_dist = np.array(#number of words in title1 * #number of words in title2
    # final_dist[i,j] = euclidean distance between vectors i, j
    return np.array(final_dist)


def heat_map_w2v(sentence1, sentence2, url, doc_id1, doc_id2, model):
    # sentance1 : title1, input apparel
    # sentance2 : title2, recommended apparel
    # url: apparel image url
    # doc_id1: document id of input apparel
    # doc_id2: document id of recommended apparel
    # model: it can have two values, 1. avg 2. weighted

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighte
    s1_vec = get_word_vec(sentence1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighte
    s2_vec = get_word_vec(sentence2, doc_id2, model)

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in title2
    # s1_s2_dist[i,j] = euclidean distance between words i, j
    s1_s2_dist = get_distance(s1_vec, s2_vec)
    # devide whole figure into 2 parts 1st part displays heatmap 2nd part displays
    gs = gridspec.GridSpec(2, 2, width_ratios=[4,1],height_ratios=[2,1])
    fig = plt.figure(figsize=(15,15))

    ax = plt.subplot(gs[0])
    # ploting the heap map based on the pairwise distances
    ax = sns.heatmap(np.round(s1_s2_dist,4), annot=True)
    # set the x axis labels as recommended apparels title
    ax.set_xticklabels(sentence2.split())
    # set the y axis labels as input apparels title
    ax.set_yticklabels(sentence1.split())
    # set title as recommended apparels title
    ax.set_title(sentence2)

    ax = plt.subplot(gs[1])
    # we remove all grids and axis labels for image
    ax.grid(False)
    ax.set_xticks([])
    ax.set_yticks([])
    display_img(url, ax, fig)

    plt.show()
```

In [26]:

```
# vocab = stores all the words that are there in google w2v model
# vocab = model.wv.vocab.keys() # if you are using Google word2Vec

vocab = model.keys()
# this function will add the vectors of each word and returns the avg vector of giv
def build_avg_vec(sentence, num_features, doc_id, m_name):
    # sentace: its title of the apparel
    # num_features: the lenght of word2vec vector, its values = 300
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of w
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(wor

    featureVec = np.zeros((num_features,), dtype="float32")
    # we will intialize a vector of size 300 with all zeros
    # we add each word2vec(wordi) to this fetureVec
    nwords = 0

    for word in sentence.split():
        nwords += 1
        if word in vocab:
            if m_name == 'weighted' and word in  idf_title_vectorizer.vocabulary_:
                featureVec = np.add(featureVec, idf_title_features[doc_id, idf_titl
            elif m_name == 'avg':
                featureVec = np.add(featureVec, model[word])
    if(nwords>0):
        featureVec = np.divide(featureVec, nwords)
    # returns the avg vector of given sentance, its of shape (1, 300)
    return featureVec
```

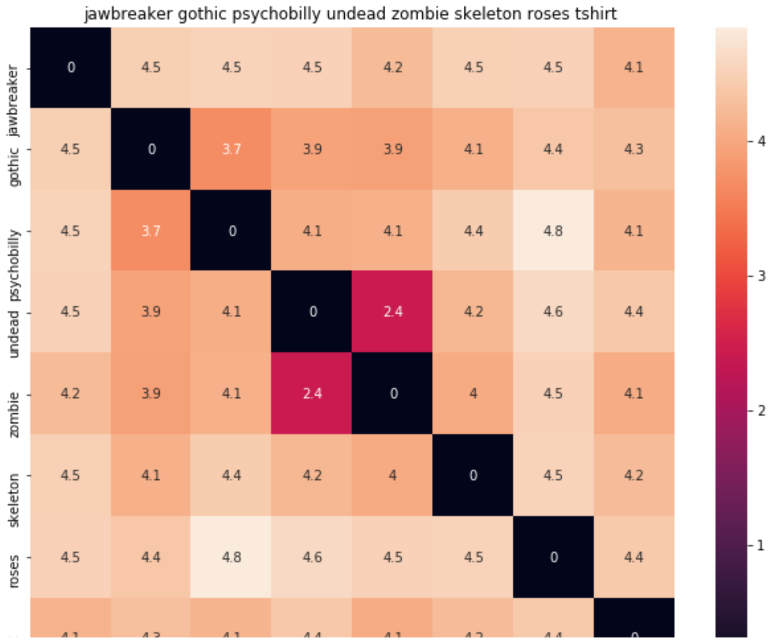
[9.2] Average Word2Vec product similarity.

In [27]:

```
doc_id = 0
w2v_title = []
# for every title we build a avg vector representation
for i in data['title']:
    w2v_title.append(build_avg_vec(i, 300, doc_id,'avg'))
    doc_id += 1

# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a
w2v_title = np.array(w2v_title)
```

In [28]:

```
def avg_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
    pairwise_dist = pairwise_distances(w2v_title, w2v_title[doc_id].reshape(1,-1))

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('BRAND :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from given input image :', pdists[i])
        print('='*125)


avg_w2v_model(12566, 20)
# in the give heat map, each cell contains the euclidean distance between words i,
```



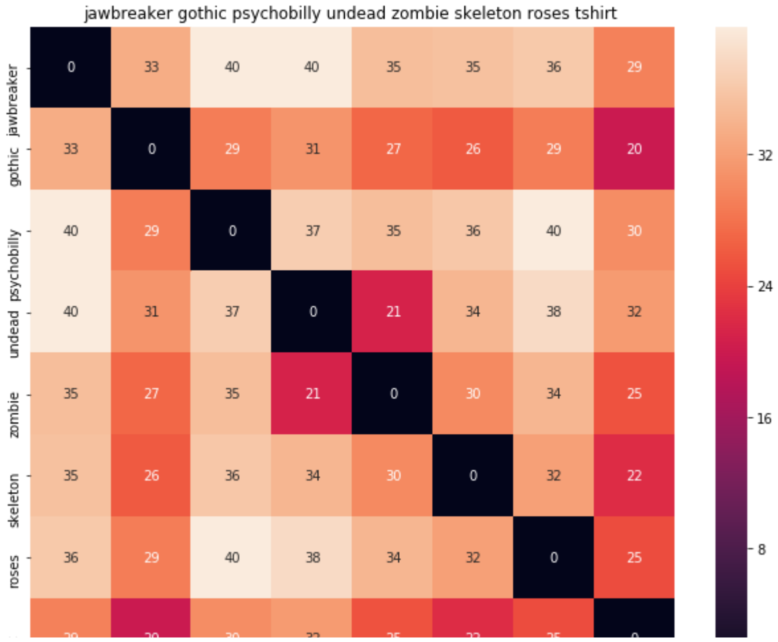
[9.4] IDF weighted Word2Vec for product similarity

In [48]:

```
doc_id = 0
w2v_title_weight = []
# for every title we build a weighted vector representation
for i in data['title']:
    w2v_title_weight.append(build_avg_vec(i, 300, doc_id,'weighted'))
    doc_id += 1
# w2v_title = np.array(# number of doc in courpus * 300), each row corresponds to a
w2v_title_weight = np.array(w2v_title_weight)
```

In [49]:

```
def weighted_w2v_model(doc_id, num_results):
    # doc_id: apparel's id in given corpus

    # pairwise_dist will store the distance from given input apparel to all remaini
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].r

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])

    for i in range(0, len(indices)):
        heat_map_w2v(data['title'].loc[df_indices[0]],data['title'].loc[df_indices[
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

weighted_w2v_model(12566, 20)
#931
#12566
# in the give heat map, each cell contains the euclidean distance between words i,
```



[9.6] Weighted similarity using brand and color.

In [50]:

```
# some of the brand values are empty.
# Need to replace Null with string "NULL"
data['brand'].fillna(value="Not given", inplace=True )

# replace spaces with hypen
brands = [x.replace(" ", "-") for x in data['brand'].values]
types = [x.replace(" ", "-") for x in data['product_type_name'].values]
colors = [x.replace(" ", "-") for x in data['color'].values]

brand_vectorizer = CountVectorizer()
brand_features = brand_vectorizer.fit_transform(brands)

type_vectorizer = CountVectorizer()
type_features = type_vectorizer.fit_transform(types)

color_vectorizer = CountVectorizer()
color_features = color_vectorizer.fit_transform(colors)

extra_features = hstack((brand_features, type_features, color_features)).tocsr()
```

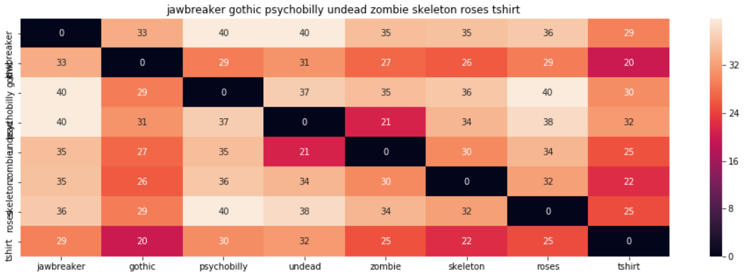
In [51]:

```
def heat_map_w2v_brand(sentance1, sentance2, url, doc_id1, doc_id2, df_id1, df_id2,

    # sentance1 : title1, input apparel
    # sentance2 : title2, recommended apparel
    # url: apparel image url
    # doc_id1: document id of input apparel
    # doc_id2: document id of recommended apparel
    # df_id1: index of document1 in the data frame
    # df_id2: index of document2 in the data frame
    # model: it can have two values, 1. avg 2. weighted

    #s1_vec = np.array(#number_of_words_title1 * 300), each row is a vector(weighte
    s1_vec = get_word_vec(sentance1, doc_id1, model)
    #s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighte
    s2_vec = get_word_vec(sentance2, doc_id2, model)

    # s1_s2_dist = np.array(#number of words in title1 * #number of words in title2
    # s1_s2_dist[i,j] = euclidean distance between words i, j
    s1_s2_dist = get_distance(s1_vec, s2_vec)

    data_matrix = [['Asin','Brand', 'Color', 'Product type'],
               [data['asin'].loc[df_id1],brands[doc_id1], colors[doc_id1], types[do
               [data['asin'].loc[df_id2],brands[doc_id2], colors[doc_id2], types[do

    colorscale = [[0, '#1d004d'],[.5, '#f2e5ff'],[1, '#f2e5d1']] # to color the hea

    # we create a table with the data_matrix
    table = ff.create_table(data_matrix, index=True, colorscale=colorscale)
    # plot it with plotly
    plotly.offline.iplot(table, filename='simple_table')

    # devide whole figure space into 25 * 1:10 grids
    gs = gridspec.GridSpec(25, 15)
    fig = plt.figure(figsize=(25,5))

    # in first 25*10 grids we plot heatmap
    ax1 = plt.subplot(gs[:, :-5])
    # ploting the heap map based on the pairwise distances
    ax1 = sns.heatmap(np.round(s1_s2_dist,6), annot=True)
    # set the x axis labels as recommended apparels title
    ax1.set_xticklabels(sentance2.split())
    # set the y axis labels as input apparels title
    ax1.set_yticklabels(sentance1.split())
    # set title as recommended apparels title
    ax1.set_title(sentance2)

    # in last 25 * 10:15 grids we display image
    ax2 = plt.subplot(gs[:, 10:16])
    # we dont display grid lins and axis labels to images
    ax2.grid(False)
    ax2.set_xticks([])
    ax2.set_yticks([])

    # pass the url it display it
    display_img(url, ax2, fig)

    plt.show()
```

In [52]:

```
def idf_w2v_brand(doc_id, w1, w2, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features

    # pairwise_dist will store the distance from given input apparel to all remaini
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].r
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist)/float(w1 + w2)

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])


    for i in range(0, len(indices)):
        heat_map_w2v_brand(data['title'].loc[df_indices[0]],data['title'].loc[df_in
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

#idf_w2v_brand(12566, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between words i,
```

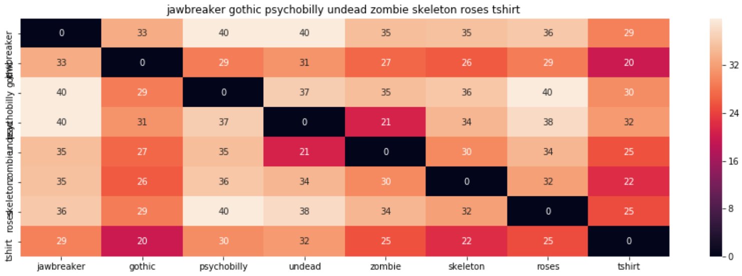
Asin	Brand	Color
B01D3R5VXC	Jawbreaker	Black



In [53]:

```
# brand and color weight =50
# title vector weight = 5

idf_w2v_brand(12566, 5, 50, 20)
```

Asin	Brand	Color
B01D3R5VXC	Jawbreaker	Black



ASSIGNMENT WORKSHOP

In [54]:

```
extra_features_new = hstack((brand_features, color_features)).tocsr()

bottleneck_features_train = np.load('16k_data_cnn_features.npy')
asins = np.load('16k_data_cnn_feature_asins.npy')
```

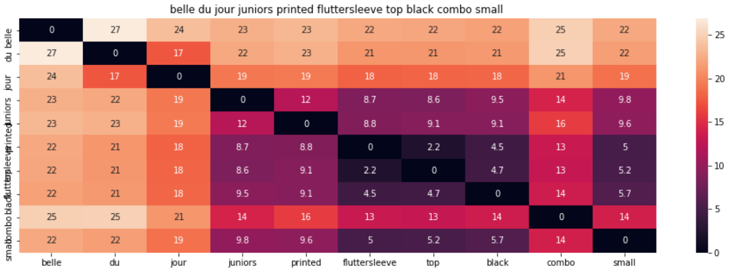
In [96]:

```
def idf_w2v_brand_n_cnn(doc_id, w1, w2, w3, num_results):
    # doc_id: apparel's id in given corpus
    # w1: weight for  w2v features
    # w2: weight for brand and color features

    # pairwise_dist will store the distance from given input apparel to all remaini
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity

    x=data[doc_id:doc_id+1]['asin'].values[0]
    new_doc_id = np.where(asins==x)[0][0]

    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].r
    ex_feat_dist = pairwise_distances(extra_features_new, extra_features_new[doc_id
    cnn_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_tr
    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist +w3*cnn_dist)/float(w

    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists  = np.sort(pairwise_dist.flatten())[0:num_results]

    #data frame indices of the 9 smallest distace's
    df_indices = list(data.index[indices])


    for i in range(0, len(indices)):
        heat_map_w2v_brand(data['title'].loc[df_indices[0]],data['title'].loc[df_in
        print('ASIN :',data['asin'].loc[df_indices[i]],x)
        print('Brand :',data['brand'].loc[df_indices[i]])
        print('euclidean distance from input :', pdists[i])
        print('='*125)

# in the give heat map, each cell contains the euclidean distance between words i,
```

In [97]:

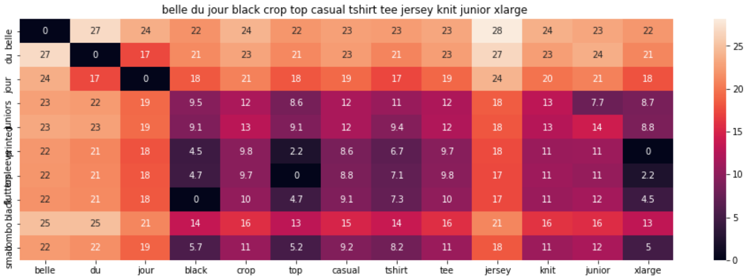
```
idf_w2v_brand_n_cnn(1250, 5, 10, 0, 10)
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



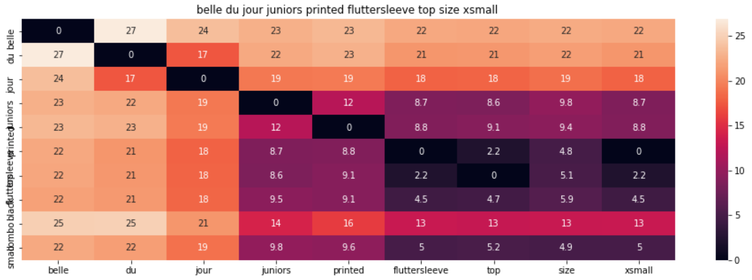
```
ASIN : B01K9CSVT0 B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 0.0006510416666666666
======================================================================
=======================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



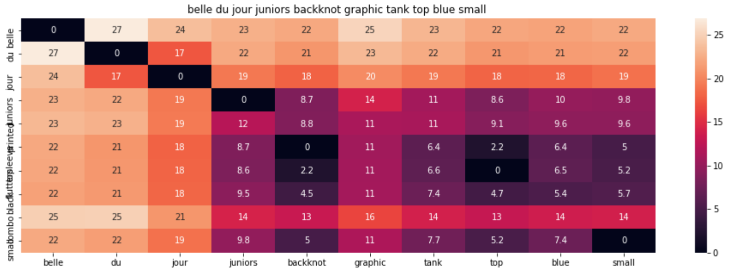
```
ASIN : B010QX9GLO B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.6115258534749348
======================================================================
=======================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



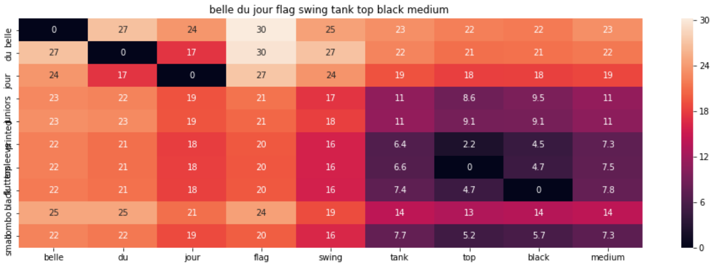
```
ASIN : B01CEZTXF6 B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.6485994387291862
==========================================================================
==========================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



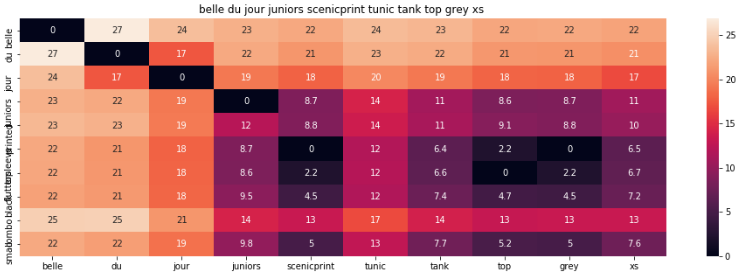
```
ASIN : B01KCY2IU8 B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.7440101671837762
==========================================================================
==========================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



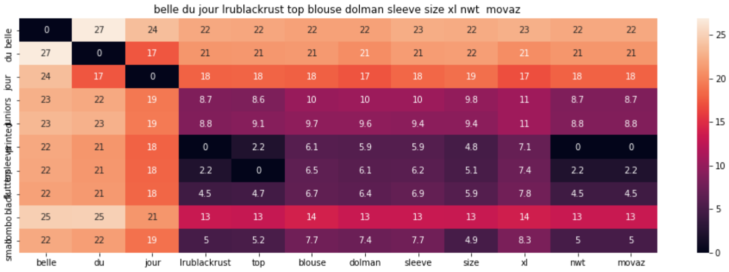
```
ASIN : B01EIEBG1A B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.8098664601643881
======================================================================
=======================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



```
ASIN : B01KCX44T2 B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.852085118355651
======================================================================
=======================================================
```

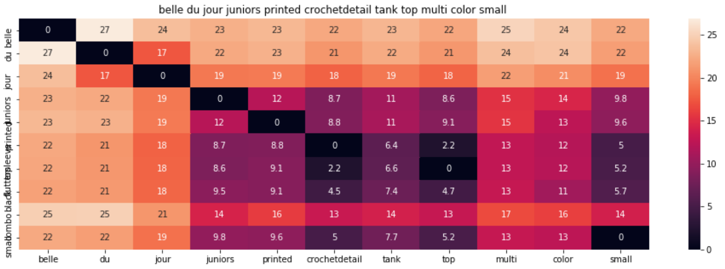
Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



```
ASIN : B0751GT3Q6 B01K9CSVT0
```

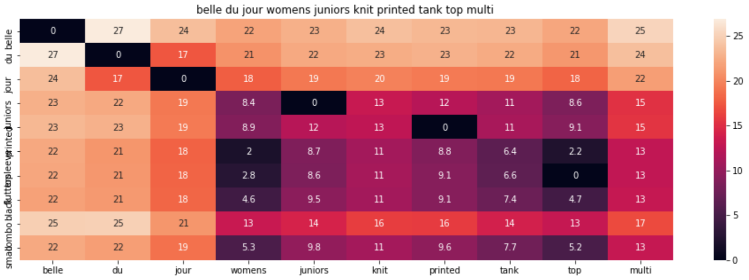
```
Brand : Belle du Jour
euclidean distance from input : 1.8854125389082228
============================================================================
============================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



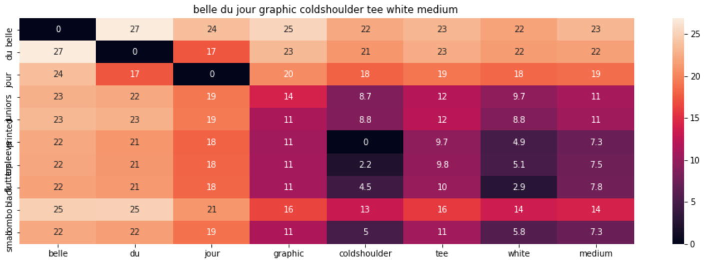
```
ASIN : B01KCY4YH8 B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.9750459671020508
============================================================================
============================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



```
ASIN : B00U4C9TSI B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 1.9937962849934896
============================================================================
============================================================
```

Asin	Brand	Color
B01K9CSVT0	Belle-du-Jour	Black-Combo



	belle	du	jour	graphic	coldshoulder	tee	white	medium
	0	27	24	25	22	23	22	23
	27	0	17	23	21	23	22	22
	24	17	0	20	18	19	18	19
	23	22	19	14	8.7	12	9.7	11
	23	23	19	11	8.8	12	8.8	11
	22	21	18	11	0	9.7	4.9	7.3
	22	21	18	11	2.2	9.8	5.1	7.5
	22	21	18	11	4.5	10	2.9	7.8
	25	25	21	16	13	16	14	14
	22	22	19	11	5	11	5.8	7.3

```
ASIN : B01K9BYYPQ B01K9CSVT0
Brand : Belle du Jour
euclidean distance from input : 2.0641607332848504
=======================================================================
=======================================================
```

In [94]:

```
arr=[[1],[2],[3],[4],[5]]
pairwise_distances(arr,[[3]])
```

Out[94]:

```
array([[2.],
       [1.],
       [0.],
       [1.],
       [2.]])
```

In [ ]: