

AppliedAIWorkshop

July 23, 2019

Amazon Apparel Recommendations

0.0.1 [4.2] Data and Code:

<https://drive.google.com/open?id=0BwNkduBnePt2VWhCYXhMV3p4dTg>

0.0.2 [4.3] Overview of the data

In [77]: *#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 [4]: 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

0.0.3 Terminology:

What is a dataset? Rows and columns Data-point Feature/variable

```

In [5]: # each product/item has 19 features in the raw dataset.
        data.columns # prints column-names or feature-names.

```

```

Out[5]: Index(['asin', 'author', 'availability', 'availability_type', 'brand', 'color',
               'editorial_review', 'editorial_review', 'formatted_price',
               'large_image_url', 'manufacturer', 'medium_image_url', 'model',
               'product_type_name', 'publisher', 'reviews', 'sku', 'small_image_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 apparel, 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 [6]: data = data[['asin', 'brand', 'color', 'medium_image_url', 'product_type_name', 'title']

```

```

In [7]: 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[7]:      asin      brand      color \
0  B016I2TS4W      FNC7C      None
1  B01N49AI08  FIG Clothing      None
2  B01JDPCOHO  FIG Clothing      None
3  B01N19U5H5      Focal18      None
4  B004GSI2OS  FeatherLite  Onyx Black/ Stone

```

```

medium_image_url product_type_name \
0 https://images-na.ssl-images-amazon.com/images... SHIRT
1 https://images-na.ssl-images-amazon.com/images... SHIRT
2 https://images-na.ssl-images-amazon.com/images... SHIRT
3 https://images-na.ssl-images-amazon.com/images... SHIRT
4 https://images-na.ssl-images-amazon.com/images... SHIRT

title formatted_price
0 Minions Como Superheroes Ironman Long Sleeve R... None
1 FIG Clothing Womens Izo Tunic None
2 FIG Clothing Womens Won Top None
3 Focal18 Sailor Collar Bubble Sleeve Blouse Shi... None
4 Featherlite Ladies' Long Sleeve Stain Resistan... $26.26

```

0.0.4 [5.1] Missing data for various features.

Basic stats for the feature: product_type_name

```
In [8]: # 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 [9]: # 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' 'UNDERWEAR'
 'SKIRT' 'OUTERWEAR' 'BRA' 'ACCESSORY' 'ART_SUPPLIES' 'SLEEPWEAR'
 'ORCA_SHIRT' 'HANDBAG' 'PET_SUPPLIES' 'SHOES' 'KITCHEN' 'ADULT_COSTUME'
 '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' 'SHIRTS'
 'ABIS_APPAREL' 'AUTO_ACCESSORY' 'NONAPPARELMISC' 'TOOLS' 'BABY_PRODUCT'
 'SOCKSHOSIERY' 'POWERSPORTS RIDING_SHIRT' 'EYEWEAR' 'SUIT'
 'OUTDOOR_LIVING' 'POWERSPORTS RIDING_JACKET' 'HARDWARE' 'SAFETY_SUPPLY'
 '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_PART'
```

```
'PROFESSIONAL_HEALTHCARE' 'SEEDS_AND_PLANTS' 'WIRELESS_ACCESSORY']
```

```
In [10]: # find the 10 most frequent product_type_names.  
product_type_count = Counter(list(data['product_type_name']))  
product_type_count.most_common(10)  
  
Out[10]: [('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 [11]: # 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 [12]: brand_count = Counter(list(data['brand']))  
brand_count.most_common(10)  
  
Out[12]: [('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 [13]: 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 [14]: color_count = Counter(list(data['color']))
color_count.most_common(10)

Out[14]: [(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 [15]: 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 [16]: price_count = Counter(list(data['formatted_price']))
price_count.most_common(10)

Out[16]: [(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 [17]: 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 [21]: # data.to_pickle('pickels/180k_apparel_data')  
  
if os.path.isfile("pickels/180k_apparel_data"):  
    print("File already present :)")  
else:  
    data.to_pickle('pickels/180k_apparel_data')  
  
File already present :)
```

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 [24]: data['formatted_price'].isnull().sum()
```

```
Out[24]: 154743
```

```
In [25]: # 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
```

So above we remove the rows for which value of 'formated_price' is Null and now we are only left with 28k number of rows to proceed with.

```
In [28]: # consider products which have color information  
# data['color'].isnull() => gives the information about the dataframe row's which have  
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
```

Similarly we remove the rows for which value of 'color' column is Null.

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 [29]: # data.to_pickle('pickels/28k_apparel_data')  
  
if os.path.isfile("pickels/28k_apparel_data") :  
    print("File already exists :)")  
  
else :  
    data.to_pickle('pickels/28k_apparel_data')  
  
File already exits :)
```

```
In [0]: # 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 images.iterrows():  
    url = row['large_image_url']  
    response = requests.get(url)  
    img = Image.open(BytesIO(response.content))  
    img.save('images/28k_images/' + row['asin'] + '.jpeg')  
  
'''
```

```
Out[0]: "\nfrom PIL import Image\nimport requests\nfrom io import BytesIO\n\nfor index, row in
```

0.0.5 [5.2] Remove near duplicate items

[5.2.1] Understand about duplicates.

```
In [39]: # 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 [40]: # read data from pickle file from previous stage
data = pd.read_pickle('pickels/28k_apparel_data')
```

In [41]: data.head()

```
Out[41]:      asin          brand      color \
4    B004GSI20S  FeatherLite  Onyx Black/ Stone
6    B012YX2ZPI  HX-Kingdom Fashion T-shirts      White
11   B001LOUGE4        Fitness Etc.      Black
15   B003BSRPB0  FeatherLite      White
21   B014ICEDNA        FNC7C      Purple

                                         medium_image_url product_type_name \
4  https://images-na.ssl-images-amazon.com/images...            SHIRT
6  https://images-na.ssl-images-amazon.com/images...            SHIRT
11 https://images-na.ssl-images-amazon.com/images...            SHIRT
15 https://images-na.ssl-images-amazon.com/images...            SHIRT
21 https://images-na.ssl-images-amazon.com/images...            SHIRT
```

		title	formatted_price
4	Featherlite Ladies' Long Sleeve Stain Resistan...	\$26.26	
6	Women's Unique 100% Cotton T - Special Olympic...	\$9.99	
11	Ladies Cotton Tank 2x1 Ribbed Tank Top	\$11.99	
15	FeatherLite Ladies' Moisture Free Mesh Sport S...	\$20.54	
21	Supernatural Chibis Sam Dean And Castiel Short...	\$7.50	

```
In [50]: # 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 [51]: # Sort the whole data based on title (alphabetical order of title)
data_sorted.sort_values('title', inplace=True, ascending=False)
data_sorted.head()
```

```
Out[51]:      asin    brand    color \
61973  B06Y1KZ2WB  Éclair  Black/Pink
133820  B010RV33VE  xiaoming  Pink
81461   B01DDSDLNS  xiaoming  White
75995   B00X5LY09Y  xiaoming  Red Anchors
151570  B00WPJG35K  xiaoming  White
```

		medium_image_url	product_type_name	
61973	https://images-na.ssl-images-amazon.com/images...	SHIRT		
133820	https://images-na.ssl-images-amazon.com/images...	SHIRT		
81461	https://images-na.ssl-images-amazon.com/images...	SHIRT		
75995	https://images-na.ssl-images-amazon.com/images...	SHIRT		
151570	https://images-na.ssl-images-amazon.com/images...	SHIRT		

		title	formatted_price
61973	Éclair Women's Printed Thin Strap Blouse Black...	\$24.99	
133820	xiaoming Womens Sleeveless Loose Long T-shirts...	\$18.19	
81461	xiaoming Women's White Long Sleeve Single Brea...	\$21.58	
75995	xiaoming Stripes Tank Patch/Bear Sleeve Anchor...	\$15.91	
151570	xiaoming Sleeve Sheer Loose Tassel Kimono Woma...	\$14.32	

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

```
In [55]: indices = []
for i, row in data_sorted.iterrows():
    indices.append(i)
```

```
In [59]: import itertools
stage1_deduplicate_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', 'Queen']
    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', 'Qu
        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 strings
        count = 0

        # itertools.zip_longest(a,b): will map the corresponding words in both strings
        # example: a = ['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a', 'a'), ('b', 'b'), ('c', 'd'), ('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 consid
        # if the number of words in which both strings differ are < 2 , we are consid
        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 stri
            stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[i]])

        # if the comprision between is between num_data_points, num_data_points-
        if j == num_data_points-1: stage1_dedupe_asins.append(data_sorted['asin'])

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

```

In [60]: data = data.loc[data['asin'].isin(stage1_dedupe_asins)]

We removed the duplicates which differ only at the end.

```
In [61]: print('Number of data points : ', data.shape[0])
```

```
Number of data points : 17593
```

```
In [62]: if os.path.isfile('pickels/17k_apperal_data') :  
    print("File already exists :")
```

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

```
File already exists :)
```

[5.2.3] Remove duplicates : Part 2

```
In [70]: data = pd.read_pickle('pickels/17k_apperal_data')
```

```
In [71]: # 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 apparel's title  
    a = data['title'].loc[i].split()  
    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Queen'  
    for j in indices:  
  
        b = data['title'].loc[j].split()  
        # store the list of words of jth string in b, ex: b = ['tokidoki', 'The', 'Qu  
length = max(len(a), len(b))  
  
    # count is used to store the number of words that are matched in both strings  
    count = 0  
  
    # itertools.zip_longest(a,b): will map the corresponding words in both strings  
    # example: a = ['a', 'b', 'c', 'd']  
    # b = ['a', 'b', 'd']  
    # itertools.zip_longest(a,b): will give [('a', 'a'), ('b', 'b'), ('c', 'd'), ('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 considering
if (length - count) < 3:
    indices.remove(j)

-----
KeyboardInterrupt                               Traceback (most recent call last)

<ipython-input-71-7cebe2e33e39> in <module>()
  16     for j in indices:
  17
-> 18         b = data['title'].loc[j].split()
  19             # store the list of words of jth string in b, ex: b = ['tokidoki', 'The',
  20

          ~\Anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
 1371
 1372             maybe_callable = com._apply_if_callable(key, self.obj)
-> 1373             return self._getitem_axis(maybe_callable, axis=axis)
 1374
 1375     def _is_scalar_access(self, key):

          ~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
 1624
 1625             # fall thru to straight lookup
-> 1626             self._has_valid_type(key, axis)
 1627             return self._get_label(key, axis=axis)
 1628

          ~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _has_valid_type(self, key, axis)
 1442
 1443     def _has_valid_type(self, key, axis):
-> 1444         ax = self.obj._get_axis(axis)
 1445
 1446         # valid for a label where all labels are in the index

KeyboardInterrupt:

```

In [0]: # from whole previous products we will consider only

```

# the products that are found in previous cell
data = data.loc[data['asin'].isin(stage2_deduplicate_asins)]
```

In [0]: 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: 16042

In [73]: 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.

1 6. Text pre-processing

In [72]: 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 [75]: # 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: {'my', 'so', 'his', 'where', 'nor', 'no', 'or', 'she', 'doing', 'here', 'ha

In [76]: 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")

5.8583171 seconds

In [77]: data.head()

Out[77]:
      asin          brand      color \
4   B004GSI20S  FeatherLite  Onyx Black/ Stone
6   B012YX2ZPI  HX-Kingdom Fashion T-Shirts     White
15  B003BSRPB0  FeatherLite     White
27  B014ICEJ1Q        FNC7C     Purple
46  B01NACPBG2      Fifth Degree     Black

                           medium_image_url product_type_name \
4   https://images-na.ssl-images-amazon.com/images...           SHIRT
6   https://images-na.ssl-images-amazon.com/images...           SHIRT
15  https://images-na.ssl-images-amazon.com/images...           SHIRT
27  https://images-na.ssl-images-amazon.com/images...           SHIRT
46  https://images-na.ssl-images-amazon.com/images...           SHIRT

                           title formatted_price
4   featherlite ladies long sleeve stain resistant...       $26.26
6   womens unique 100 cotton special olympics wor...       $9.99
15  featherlite ladies moisture free mesh sport sh...     $20.54
27  supernatural chibis sam dean castiel neck tshi...     $7.39
46  fifth degree womens gold foil graphic tees jun...     $6.95

```

```
In [78]: data.to_pickle('pickels/16k_apperial_data_preprocessed')
```

1.1 Stemming

```

In [79]: 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

```

2 [8] Text based product similarity

```
In [2]: data = pd.read_pickle('pickels/16k_apperial_data_preprocessed')
data.head()
```

```

Out[2]:      asin           brand       color \
4   B004GSI2OS      FeatherLite  Onyx Black/ Stone
6   B012YX2ZPI  HX-Kingdom Fashion T-shirts      White
15  B003BSRPB0      FeatherLite      White
27  B014ICEJ1Q          FNC7C      Purple
46  B01NACPBG2      Fifth Degree      Black

                           medium_image_url product_type_name \
4   https://images-na.ssl-images-amazon.com/images...      SHIRT
6   https://images-na.ssl-images-amazon.com/images...      SHIRT
15  https://images-na.ssl-images-amazon.com/images...      SHIRT
27  https://images-na.ssl-images-amazon.com/images...      SHIRT
46  https://images-na.ssl-images-amazon.com/images...      SHIRT

                           title formatted_price
4   featherlite ladies long sleeve stain resistant...     $26.26
6   womens unique 100 cotton special olympics wor...     $9.99
15  featherlite ladies moisture free mesh sport sh...    $20.54
27  supernatural chibis sam dean castiel neck tshi...    $7.39
46  fifth degree womens gold foil graphic tees jun...    $6.95

```

In [40]: # 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 occurrence of the word
    # labels: len(labels) == len(keys), the values of labels depends on the model
        # 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))
    # 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, plotting heat map that represents the count of commonly occurred words in
    ax = plt.subplot(gs[0])

```

```

# it displays a cell in white color if the word is intersection(lis of words)
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 to
    intersection = set(vec1.keys()) & set(vec2.keys())

    # we set the values of non intersecting words to zero, this is just to show the difference
    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 title2)
    values = [vec2[x] for x in vec2.keys()]

    # labels: len(labels) == len(keys), the values of labels depends on the model we choose
    # 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 corpus
        # tfidf_title_features[doc_id, index_of_word_in_corpus] will give the tfidf value
        if x in tfidf_title_vectorizer.vocabulary_:
            labels.append(tfidf_title_features[doc_id, tfidf_title_vectorizer.vocabulary_.index(x)])
        else:
            labels.append(0)
elif model == 'idf':
    labels = []
    for x in vec2.keys():
        # idf_title_vectorizer.vocabulary_ it contains all the words in the corpus
        # idf_title_features[doc_id, index_of_word_in_corpus] will give the idf value
        if x in idf_title_vectorizer.vocabulary_:
            labels.append(idf_title_features[doc_id, idf_title_vectorizer.vocabulary_.index(x)])
        else:
            labels.append(0)

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

# this function gets a list of words 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.split(" ")'
    return Counter(words) # Counter counts the occurrence of each word in list, it returns a dict

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)

    # vector2 = dict{word21:#count, word22:#count, etc.}
    vector2 = text_to_vector(text2)

    plot_heatmap_image(doc_id, vector1, vector2, url, text2, model)

```

2.1 [8.2] Bag of Words (BoW) on product titles.

```
In [4]: 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 occured in doc_id
```

Out[4]: (16042, 12609)

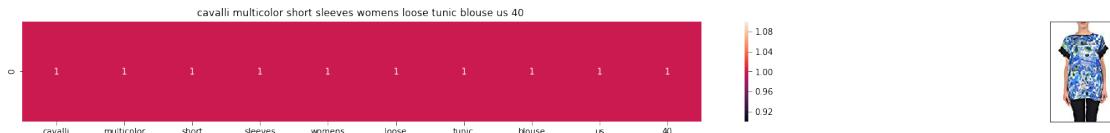
```
In [5]: def bag_of_words_model(doc_id, num_results):
        # doc_id: apparel's id in given corpus

        # pairwise_dist will store the distance from given input apparel to all remaining
        # the metric we used here is cosine, the coside distance is mesured as  $K(X, Y) = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}$ 
        # 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 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[df_indices[i]])
            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)
bag_of_words_model(15000, 20)
# in the output heat map each value represents the tfidf values of the label word, the
```

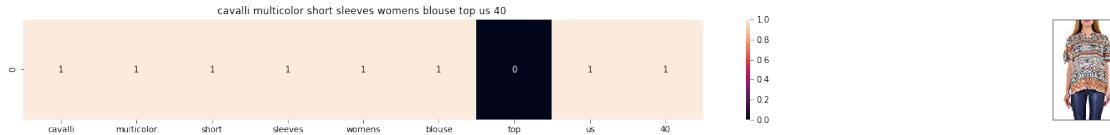


ASIN : B01151KH8M

BRAND : Just Cavalli

Euclidean distance from the given image : 0.0

=====

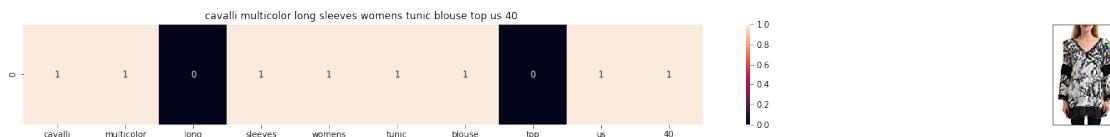


ASIN : B0115CTTIO

BRAND : Just Cavalli

Euclidean distance from the given image : 1.7320508075688772

=====

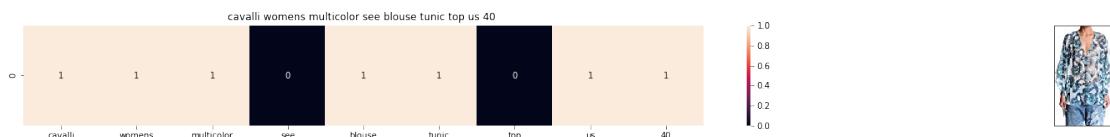


ASIN : B01HWH2F42

BRAND : Just Cavalli

Euclidean distance from the given image : 2.0

=====

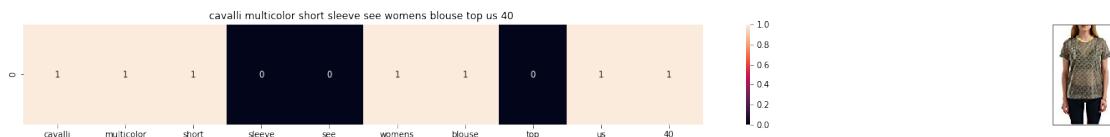


ASIN : B01FZTUK8M

BRAND : Just Cavalli

Euclidean distance from the given image : 2.23606797749979

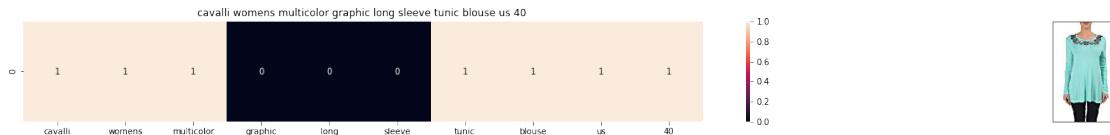
=====



ASIN : B01HSKDJXY

BRAND : Just Cavalli

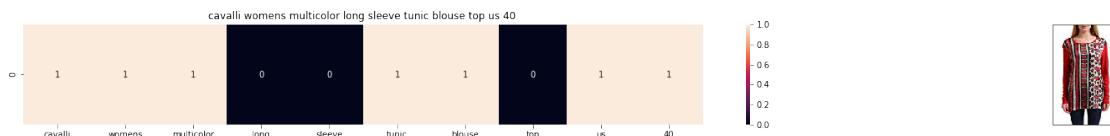
Euclidean distance from the given image : 2.449489742783178



ASIN : B017ADYVZO

BRAND : Just Cavalli

Euclidean distance from the given image : 2.449489742783178



ASIN : B01CASX59M

BRAND : Just Cavalli

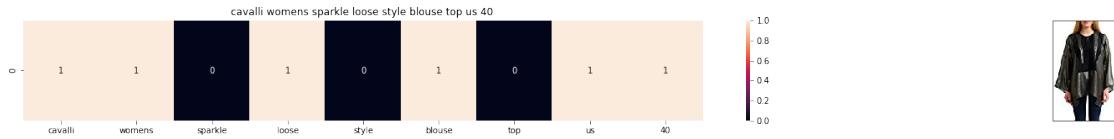
Euclidean distance from the given image : 2.449489742783178



ASIN : B0180UM25M

BRAND : Just Cavalli

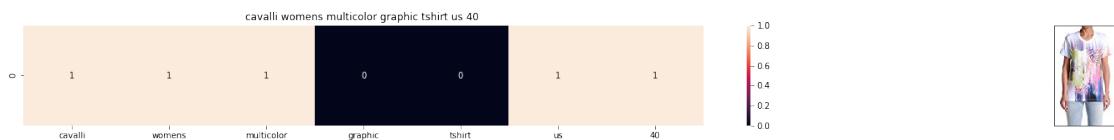
Euclidean distance from the given image : 2.449489742783178



ASIN : B01CAIM9OE

BRAND : Just Cavalli

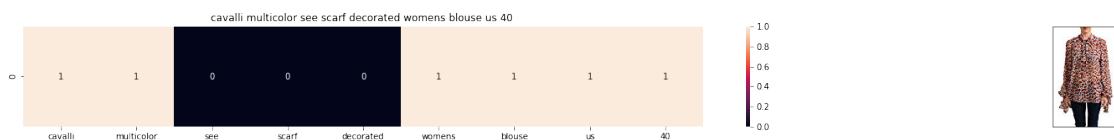
Eucliden distance from the given image : 2.6457513110645907



ASIN : B01FZUNBVO

BRAND : Just Cavalli

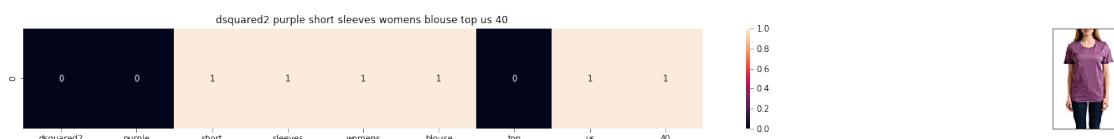
Eucliden distance from the given image : 2.6457513110645907



ASIN : B01INS02IU

BRAND : Just Cavalli

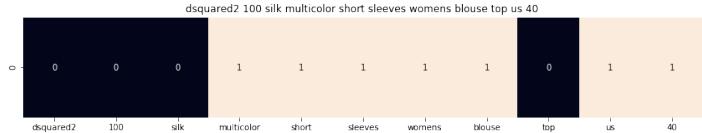
Eucliden distance from the given image : 2.6457513110645907



ASIN : B01D3VEI0G

BRAND : DSQUARED2

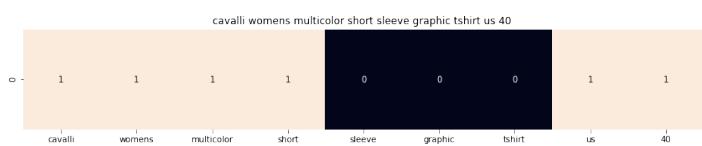
Eucliden distance from the given image : 2.6457513110645907



ASIN : B01D6F5CDK

BRAND : DSQUARED2

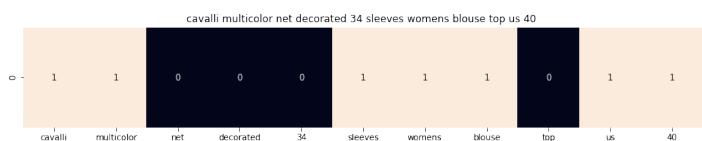
Eucliden distance from the given image : 2.6457513110645907



ASIN : B01FYHEEVY

BRAND : Just Cavalli

Eucliden distance from the given image : 2.6457513110645907



ASIN : B01HWETL10

BRAND : Just Cavalli

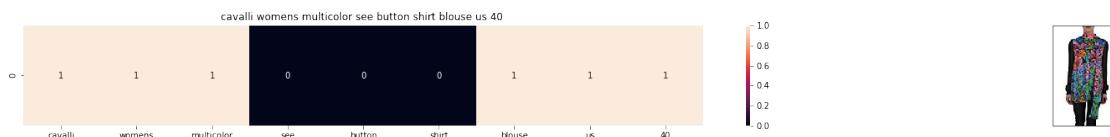
Eucliden distance from the given image : 2.6457513110645907



ASIN : B0175AE1TI

BRAND : Just Cavalli

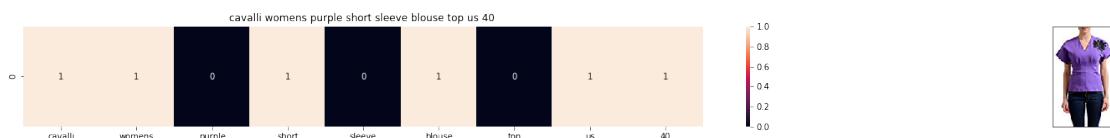
Euclidean distance from the given image : 2.6457513110645907



ASIN : B00TQ2LM1E

BRAND : Just Cavalli

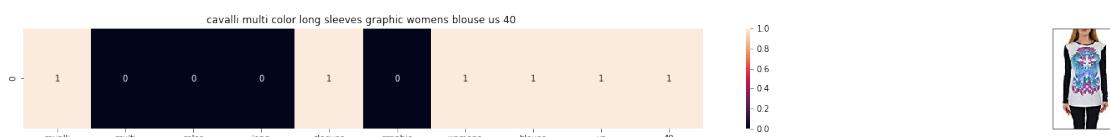
Euclidean distance from the given image : 2.6457513110645907



ASIN : B01CAU74LK

BRAND : Just Cavalli

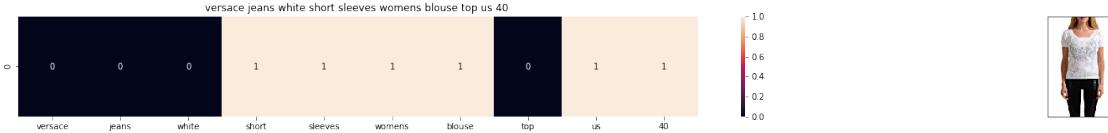
Euclidean distance from the given image : 2.6457513110645907



ASIN : B0177ZROQI

BRAND : Just Cavalli

Euclidean distance from the given image : 2.8284271247461903



ASIN : B01LI5ED76

BRAND : Versace Jeans

Euclidean distance from the given image : 2.8284271247461903

2.2 [8.5] TF-IDF based product similarity

```
In [6]: tfidf_title_vectorizer = TfidfVectorizer(min_df = 0)
        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 sparse matrix of dimensions :
        # tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the word in j
```

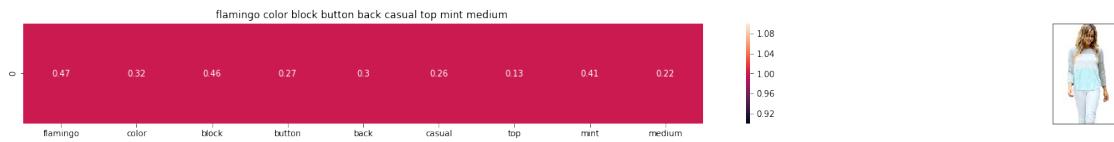
```
In [7]: def tfidf_model(doc_id, num_results):
        # doc_id: apparel's id in given corpus

        # pairwise_dist will store the distance from given input apparel to all remaining
        # the metric we used here is cosine, the coside distance is measured as  $K(X, Y) = \frac{X \cdot Y}{\|X\| \|Y\|}$ 
        # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
        pairwise_dist = pairwise_distances(tfidf_title_features,tfidf_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 distance's
        df_indices = list(data.index[indices])

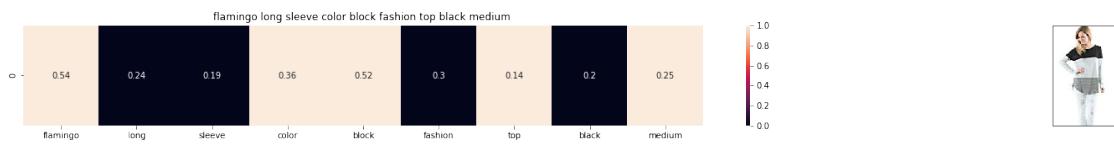
        for i in range(0,len(indices)):
            # we will pass 1. doc_id, 2. title1, 3. title2, url, model
            get_result(indices[i], data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]])
            print('ASIN : ',data['asin'].loc[df_indices[i]])
            print('BRAND : ',data['brand'].loc[df_indices[i]])
            print ('Euclidean distance from the given image : ', pdists[i])
            print('='*125)
    tfidf_model(2112, 10)
    # in the output heat map each value represents the tfidf values of the label word, the
```



ASIN : B017MPGX1K

BRAND : FLAMINGO

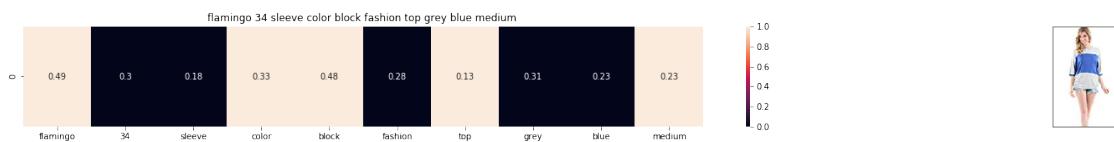
Eucliden distance from the given image : 0.0



ASIN : B017WSP16K

BRAND : FLAMINGO

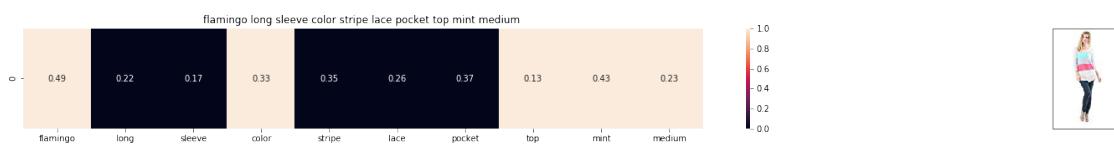
Eucliden distance from the given image : 0.7989014601682404



ASIN : B0182TDI8Q

BRAND : FLAMINGO

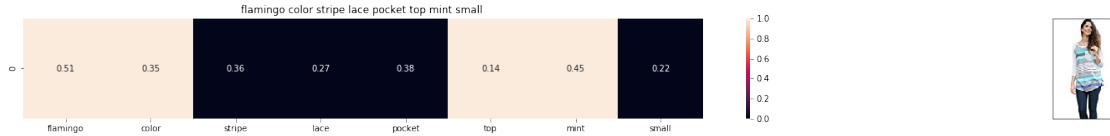
Eucliden distance from the given image : 0.8651206757737551



ASIN : B017OK3ECE

BRAND : FLAMINGO

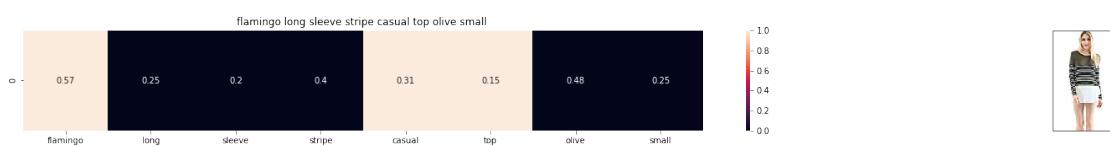
Eucliden distance from the given image : 0.9192373199332413



ASIN : B017OK38LQ

BRAND : FLAMINGO

Eucliden distance from the given image : 0.9472426598341093



ASIN : B017WSOVJS

BRAND : FLAMINGO

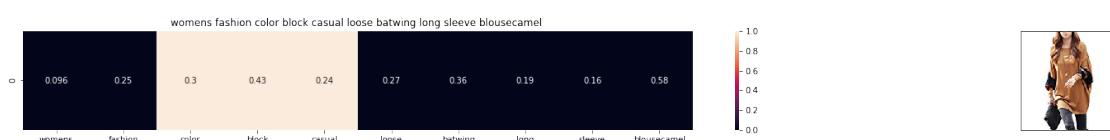
Eucliden distance from the given image : 1.126782832955081



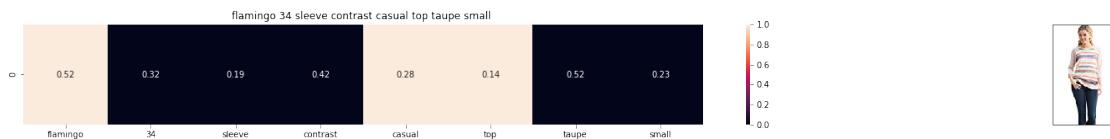
ASIN : B06WW3LPR5

BRAND : Sankuwen

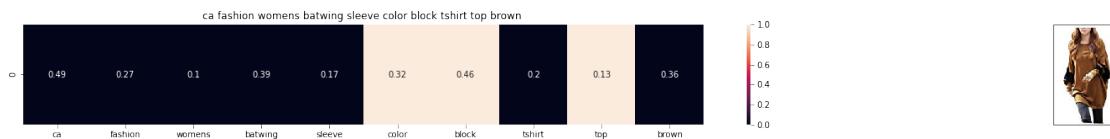
Eucliden distance from the given image : 1.1280136453455298



ASIN : B01MF78AC1
 BRAND : CHICFOR
 Eucliden distance from the given image : 1.135938039795153



ASIN : B017I2JWLO
 BRAND : FLAMINGO
 Eucliden distance from the given image : 1.1541564341832478



ASIN : B00GX7LR70
 BRAND : CA Fashion
 Eucliden distance from the given image : 1.156749328362347

2.3 [8.5] IDF based product similarity

```
In [8]: 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 dimensions :
        # idf_title_features[doc_id, index_of_word_in_corpus] = number of times the word occurs
```

```
In [9]: def nContaining(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] / (nContaining(word)))

```

In [10]: # 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
 # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will return
 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 word
 # idf_title_features[doc_id, index_of_word_in_corpus] = idf value of word
 idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val

In [11]: 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 remaining
 # the metric we used here is cosine, the coside distance is measured as $K(X, Y) = \frac{X \cdot Y}{\|X\| \|Y\|}$
 # <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[df_indices[i]])
 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(12566,10)
# in the output heat map each value represents the idf values of the label word, the

```



ASIN : B00JXQB5FQ

Brand : Si Row

euclidean distance from the given image : 0.0



ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from the given image : 12.20507131122177



ASIN : B00JXQCWT0

Brand : Si Row

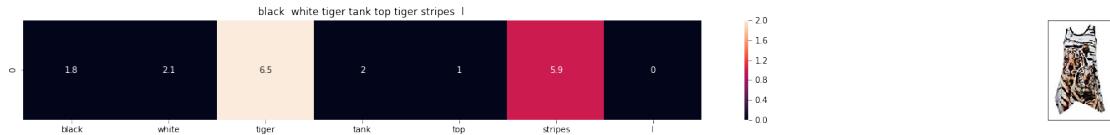
euclidean distance from the given image : 14.468362685603465



ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from the given image : 14.486832924778964



ASIN : B00JXQA094

Brand : Si Row

euclidean distance from the given image : 14.833392966672909



ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from the given image : 14.898744516719225



ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from the given image : 15.224458287343769



ASIN : B074T8ZYGX

Brand : MKP Crop Top

euclidean distance from the given image : 17.080812955631995



ASIN : B0OKF2N5PU

Brand : Vietsbay

euclidean distance from the given image : 17.090168125645416



ASIN : B00JPOZ9GM

Brand : Sofra

euclidean distance from the given image : 17.153215337562703

3 [9] Text Semantics based product similarity

```
In [0]: # 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)

...

```

In [12]:

```

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 which contains a dict ,
# and it contains all our corpus 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/0B7XkCwpI5KDYNlNUTlSS21pQmM/edit
# it's 1.9GB in size.

...
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
...


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

```

In [41]: # 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 word
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)
    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.vocabulary_])
            elif m_name == 'avg':

```

```

        vec.append(model[i])
    else:
        # if the word in our corpus 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) in
    return np.array(vec)

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

    final_dist = []
    # for each vector in vec1 we calculate the distance(euclidean) to all vectors in vec2
    for i in vec1:
        dist = []
        for j in vec2:
            # np.linalg.norm(i-j) will result the euclidean distance between vectors i, j
            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):
    # sentence1 : title1, input apparel
    # sentence2 : 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(weighted/avg)
    s1_vec = get_word_vec(sentence1, doc_id1, model)
    # s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighted/avg)
    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 image
    gs = gridspec.GridSpec(2, 2, width_ratios=[4,1], height_ratios=[2,1])
    fig = plt.figure(figsize=(15,15))

```

```

ax = plt.subplot(gs[0])
# plotting 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 [42]: # 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 given
def build_avg_vec(sentence, num_features, doc_id, m_name):
    # sentence: its title of the apparel
    # num_features: the length 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 word
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)

featureVec = np.zeros((num_features,), dtype="float32")
# we will initialize a vector of size 300 with all zeros
# we add each word2vec(wordi) to this featureVec
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_title_
        elif m_name == 'avg':
            featureVec = np.add(featureVec, model[word])
if(nwords>0):
    featureVec = np.divide(featureVec, nwords)
# returns the avg vector of given sentence, its of shape (1, 300)
return featureVec

```

3.0.1 [9.2] Average Word2Vec product similarity.

```
In [43]: 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 d
w2v_title = np.array(w2v_title)

In [44]: def avg_w2v_model(doc_id, num_results):
        # doc_id: apparel's id in given corpus

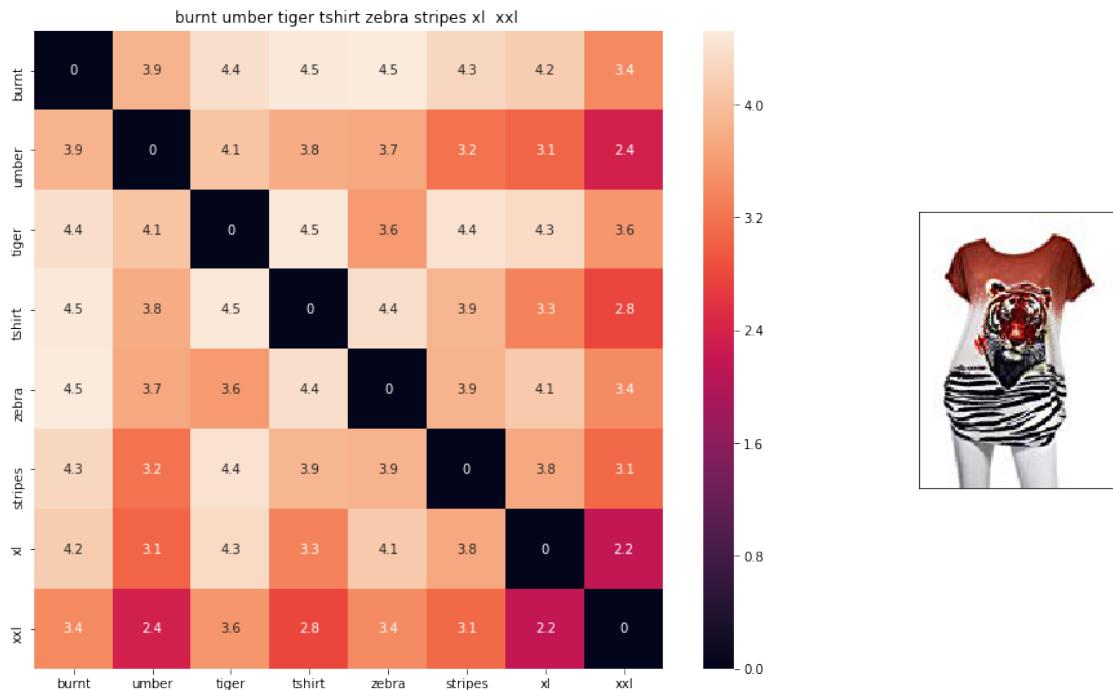
        # dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
        pairwise_dist = pairwise_distances(w2v_title, w2v_title[doc_id].reshape(1,-1))

        # 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[i]])
            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)

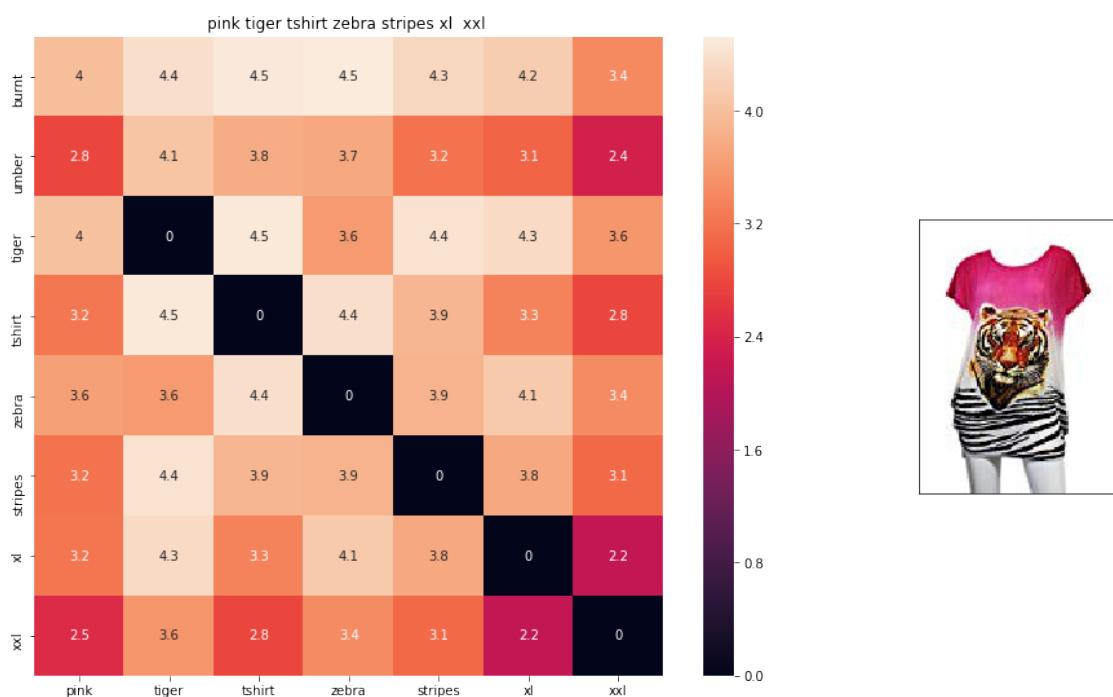
avg_w2v_model(12566, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```



ASIN : B00JXQB5FQ

BRAND : Si Row

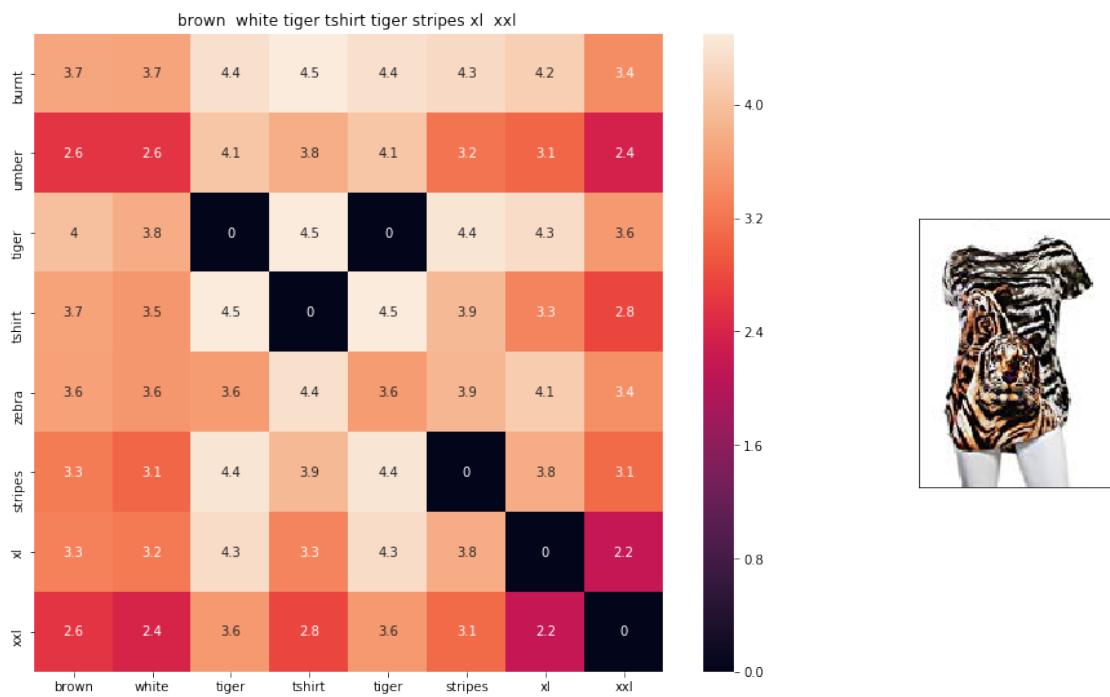
euclidean distance from given input image : 0.0



ASIN : BO0JXQASS6

BRAND : Si Row

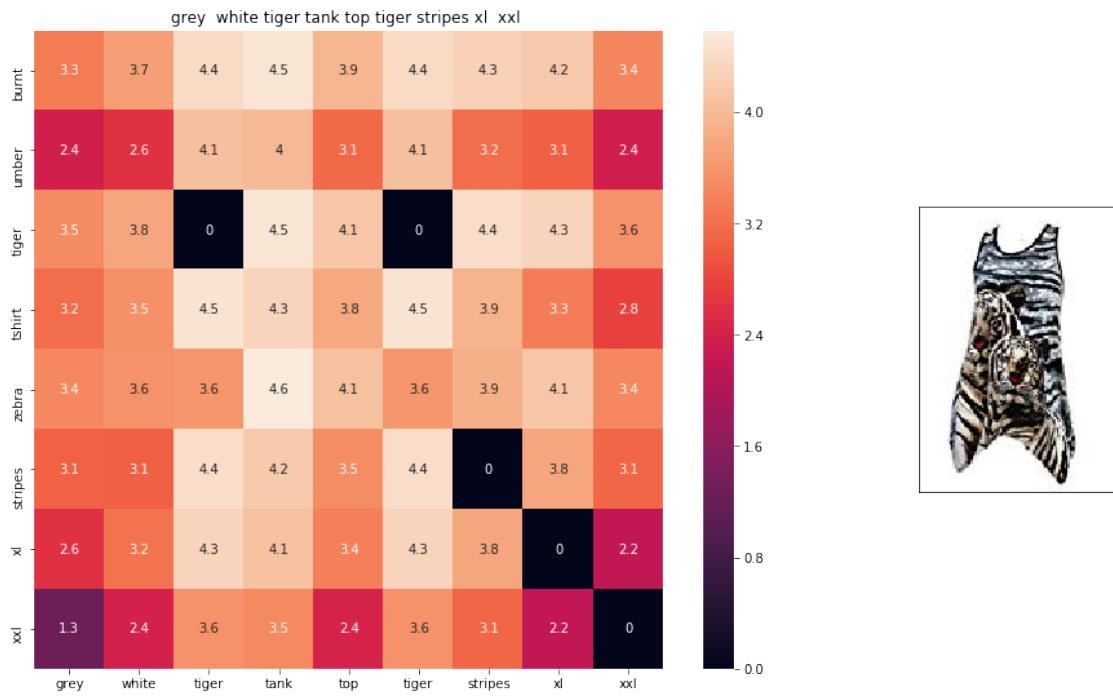
euclidean distance from given input image : 0.5891926



ASIN : BO0JXQCWT0

BRAND : Si Row

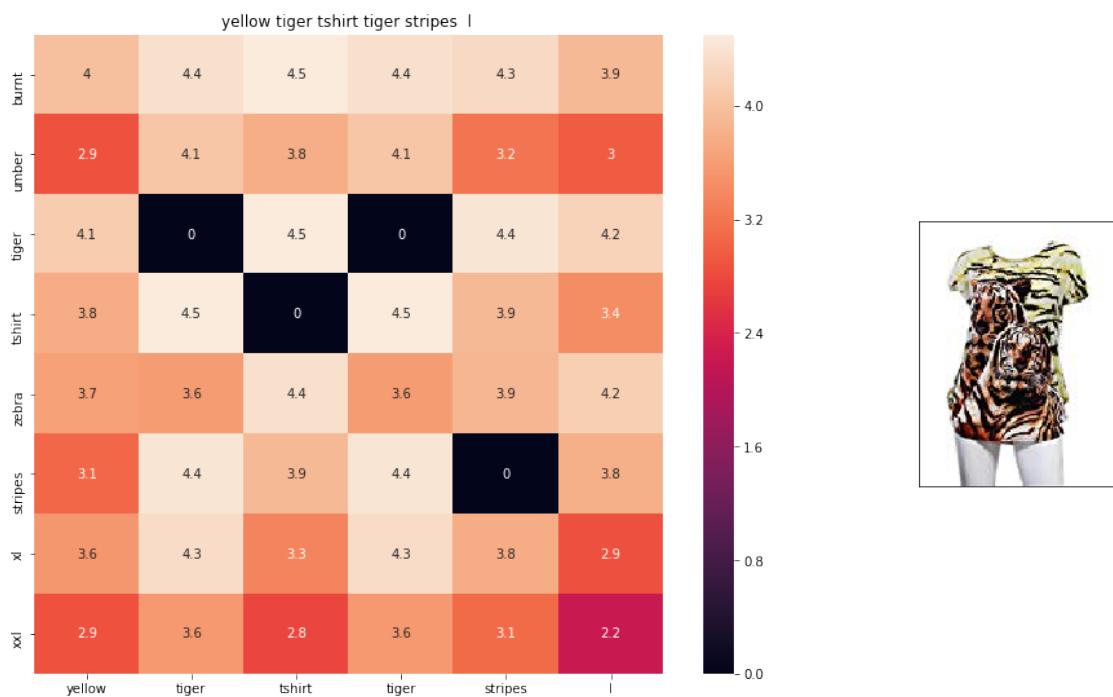
euclidean distance from given input image : 0.7003438



ASIN : B00JXQAFZ2

BRAND : Si Row

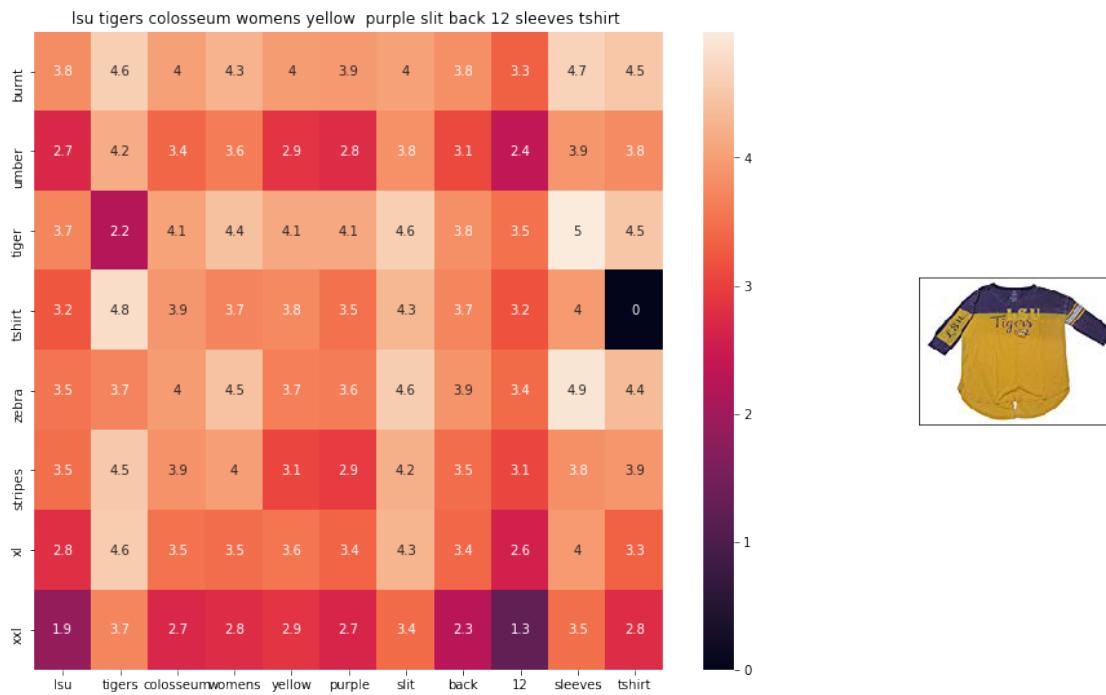
euclidean distance from given input image : 0.89283955



ASIN : B00JXQCUIC

BRAND : Si Row

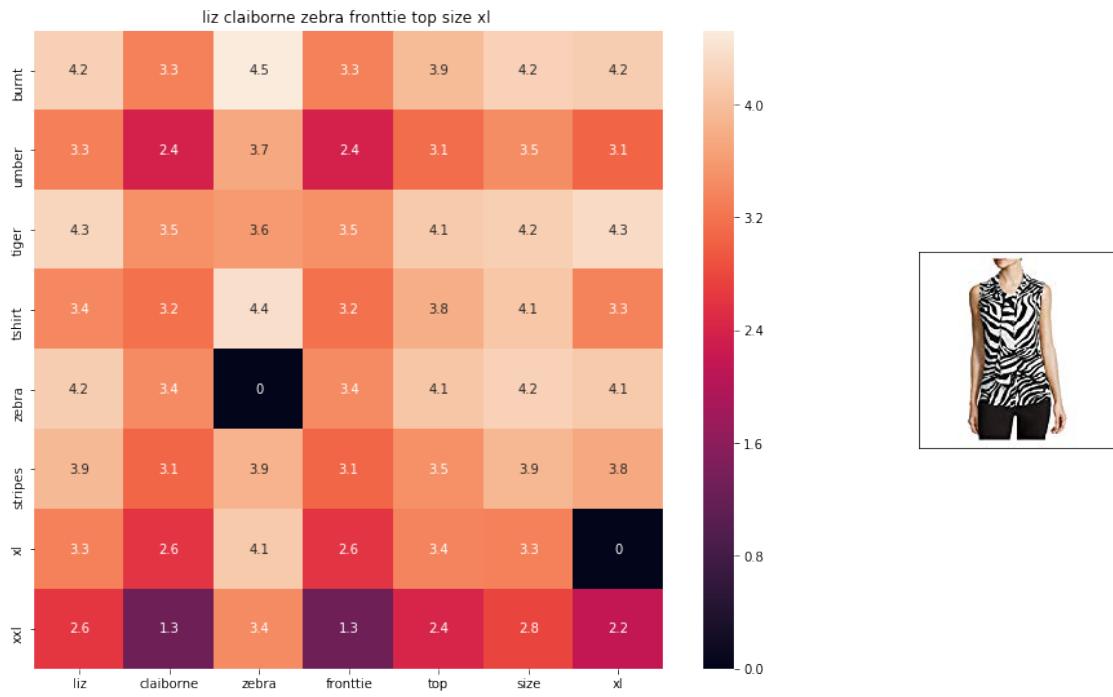
euclidean distance from given input image : 0.95601255



ASIN : B073R5Q8HD

BRAND : Colosseum

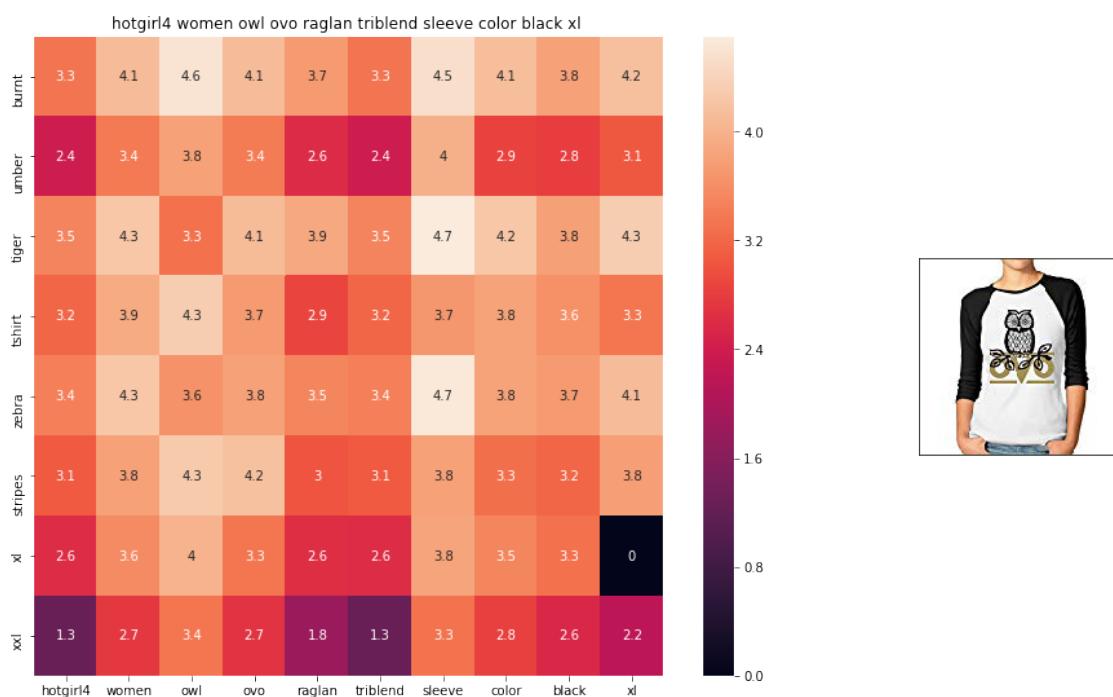
euclidean distance from given input image : 1.022969



ASIN : B06XBY5QXL

BRAND : Liz Claiborne

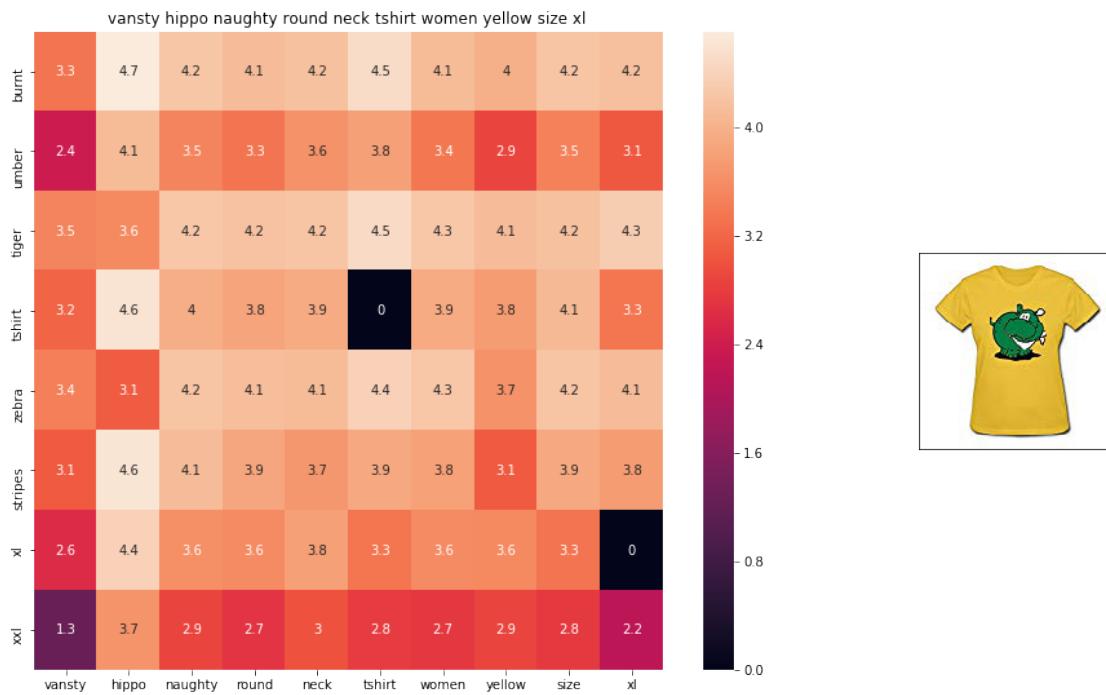
euclidean distance from given input image : 1.0669324



ASIN : B01L8L73M2

BRAND : Hotgirl4 Raglan Design

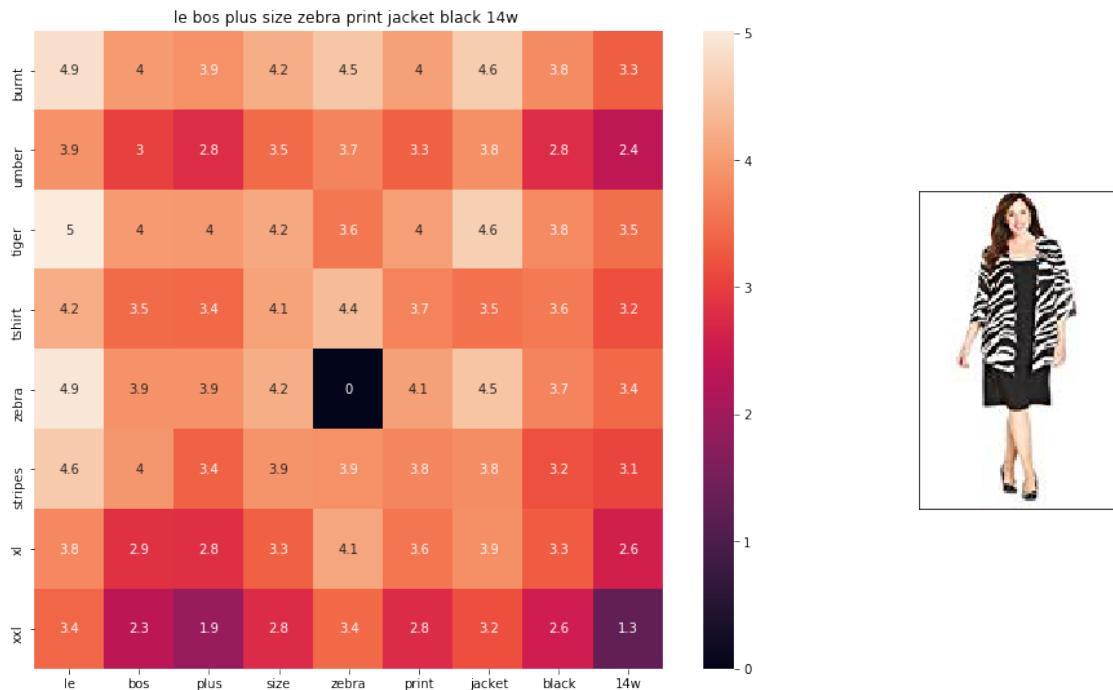
euclidean distance from given input image : 1.0731405



ASIN : B01EJS5H06

BRAND : Vansty

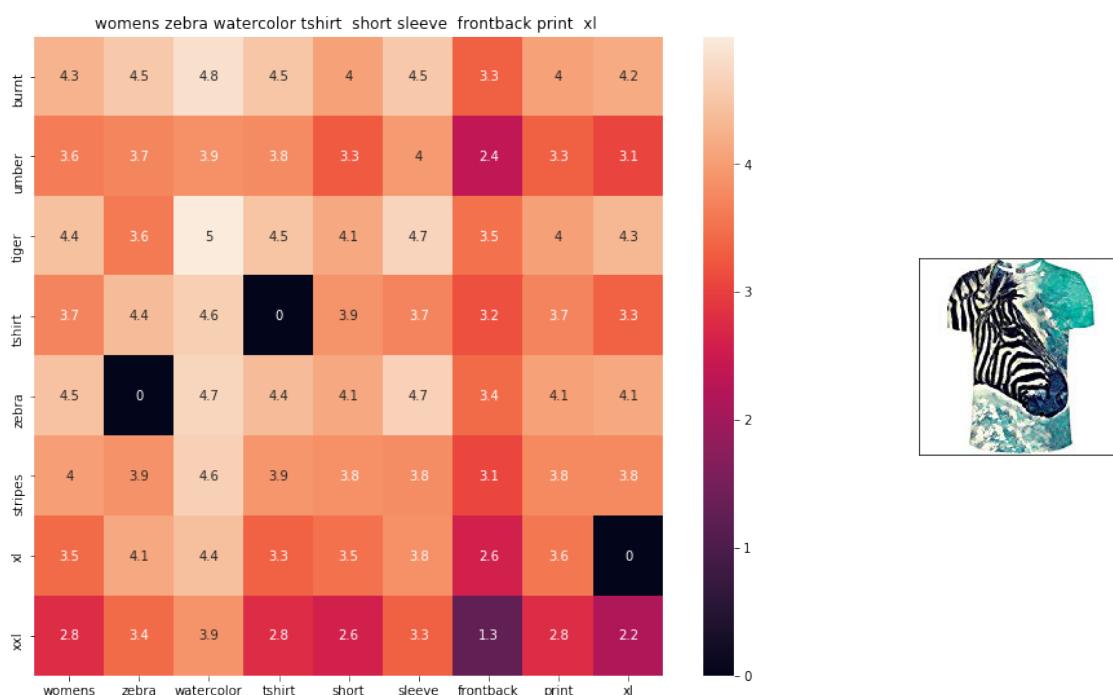
euclidean distance from given input image : 1.075719



ASIN : B01B01XRK8

BRAND : Le Bos

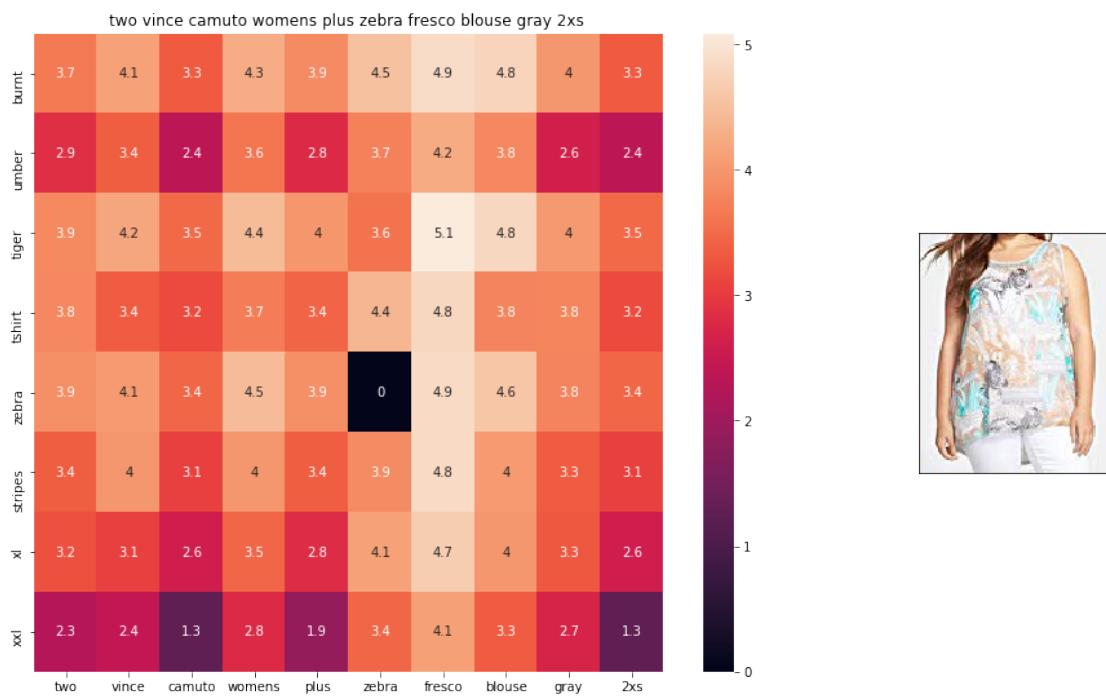
euclidean distance from given input image : 1.0839964



ASIN : B072R2JXKW

BRAND : WHAT ON EARTH

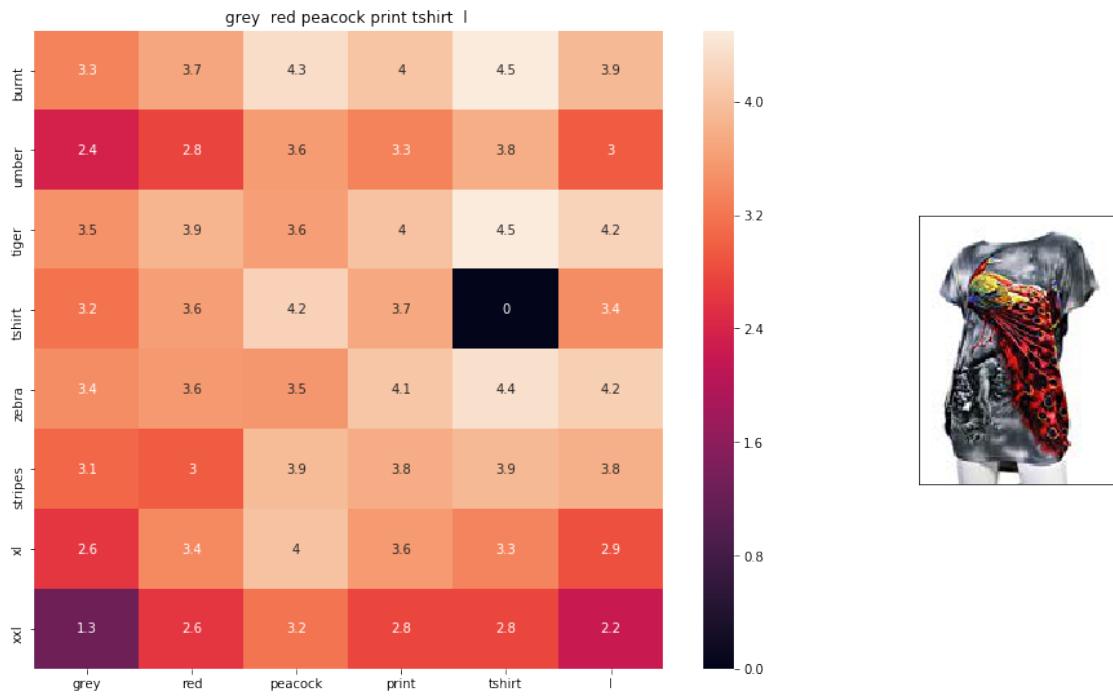
euclidean distance from given input image : 1.0842218



ASIN : B074MJRGW6

BRAND : Two by Vince Camuto

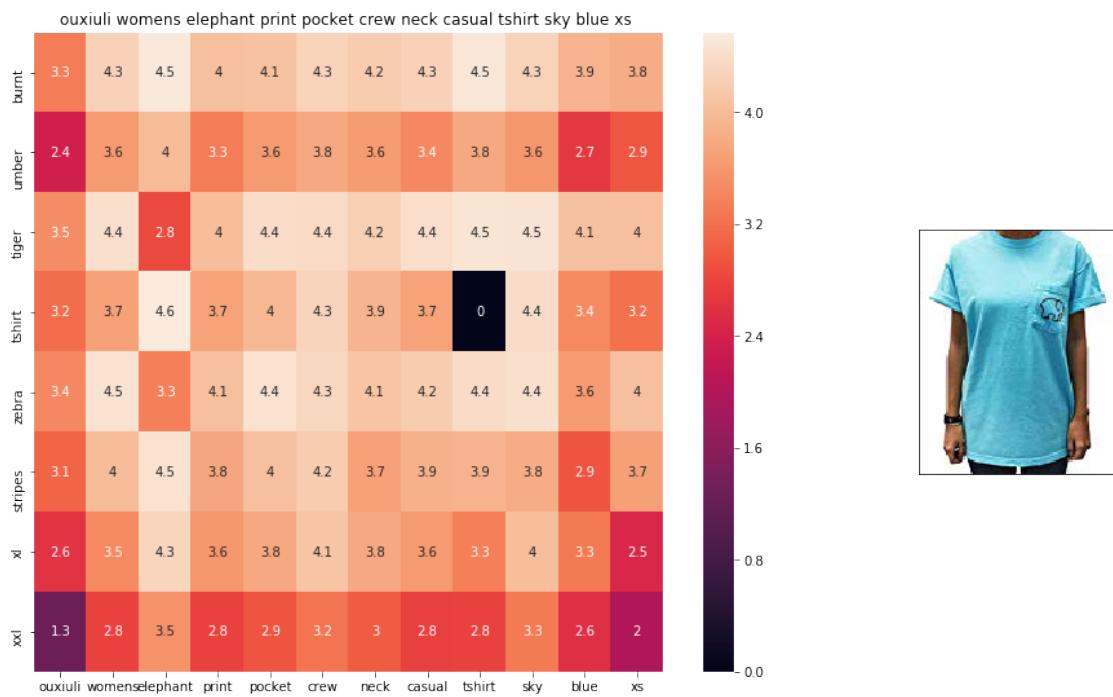
euclidean distance from given input image : 1.0895038



ASIN : BO0JXQCFRS

BRAND : Si Row

euclidean distance from given input image : 1.0900588

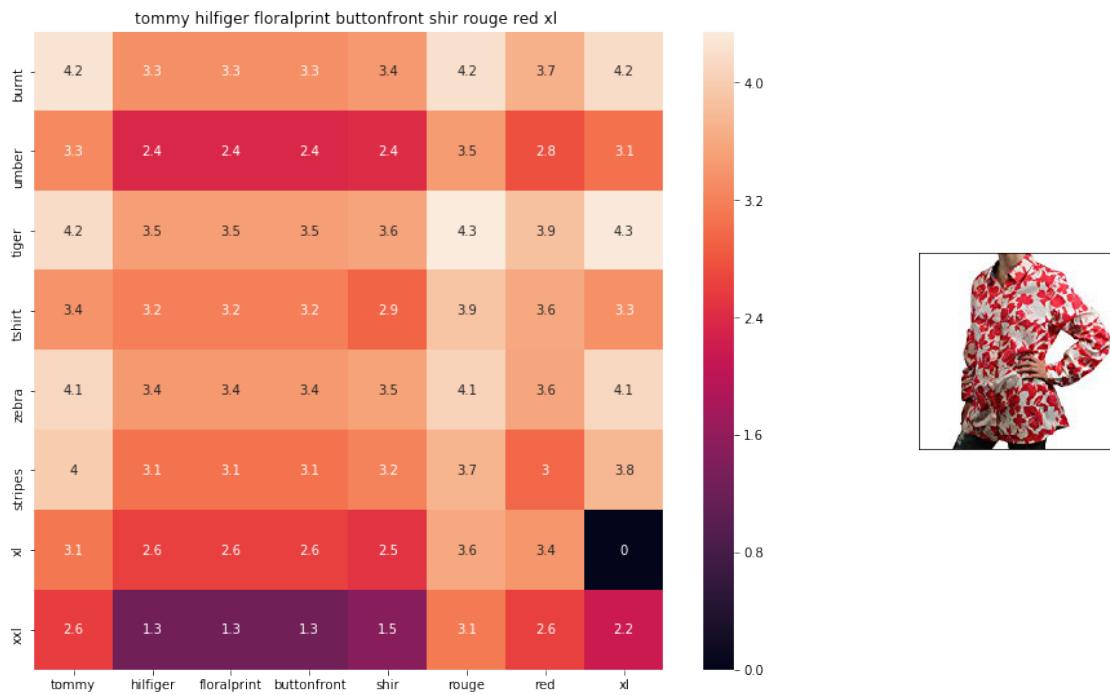


ASIN : B01I53HU6K

BRAND : ouxiuli

euclidean distance from given input image : 1.0920111

=====

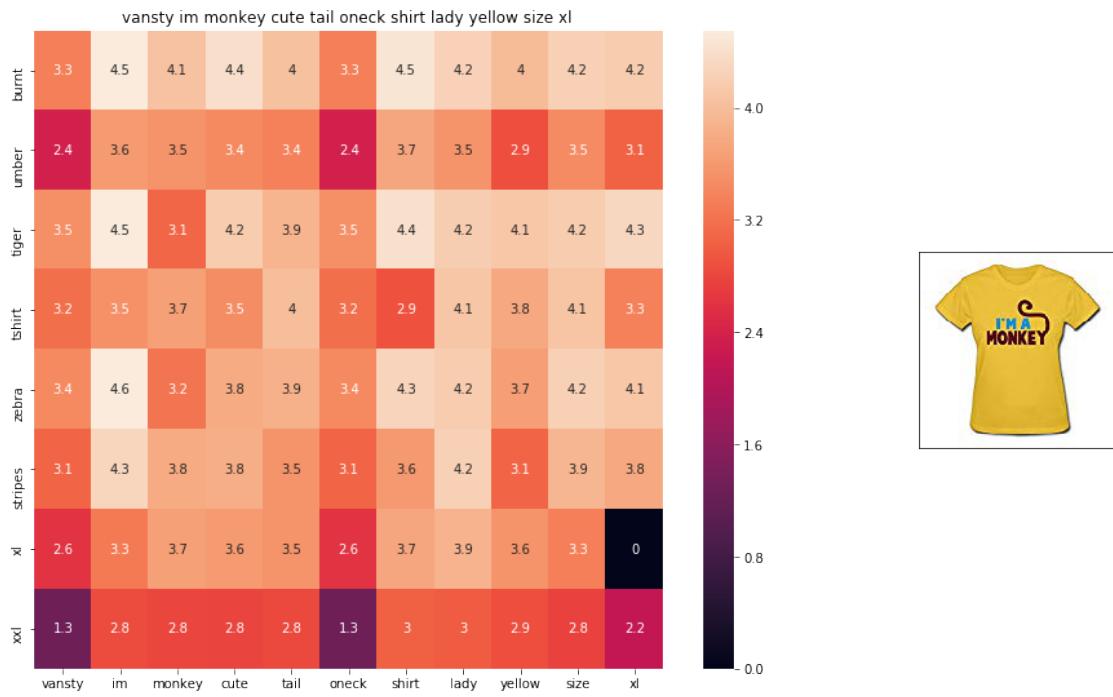


ASIN : B0711NGTQM

BRAND : THILFIGER RTW

euclidean distance from given input image : 1.0923415

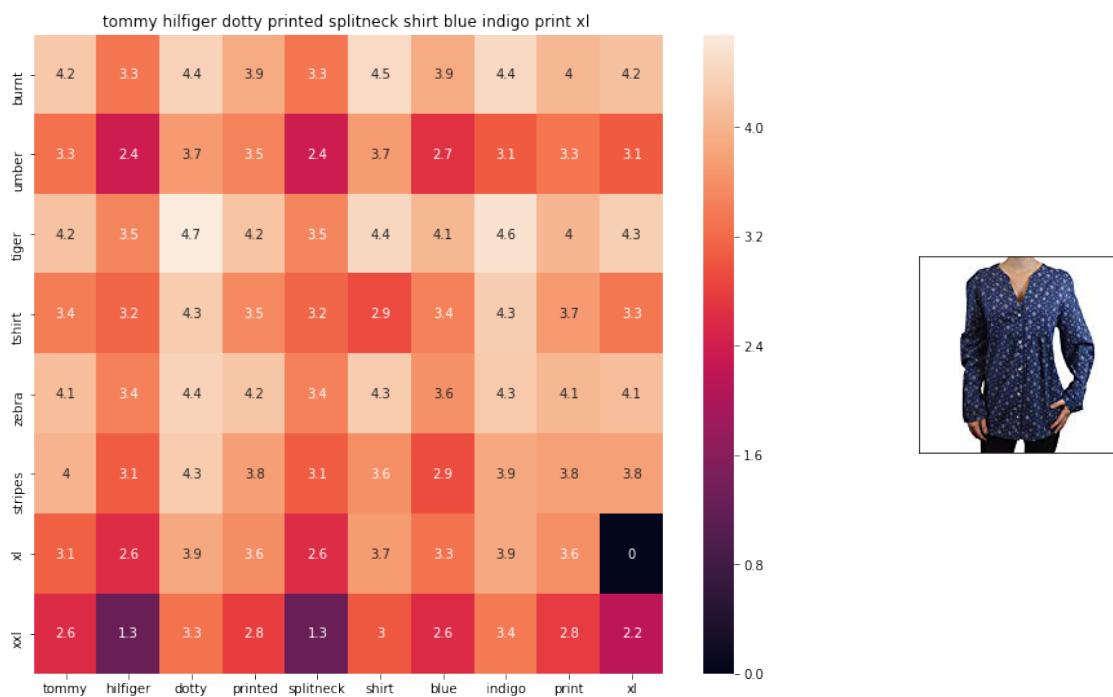
=====



ASIN : B01EFSL08Y

BRAND : Vansty

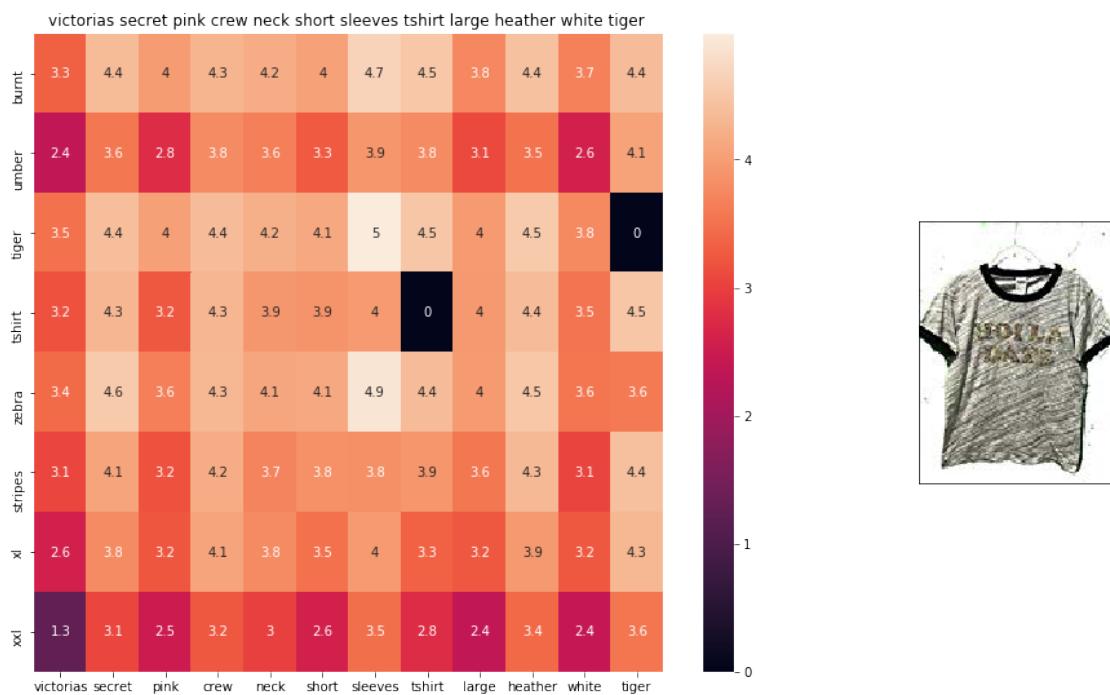
euclidean distance from given input image : 1.0934004



ASIN : B0716TVWQ4

BRAND : THILFIGER RTW

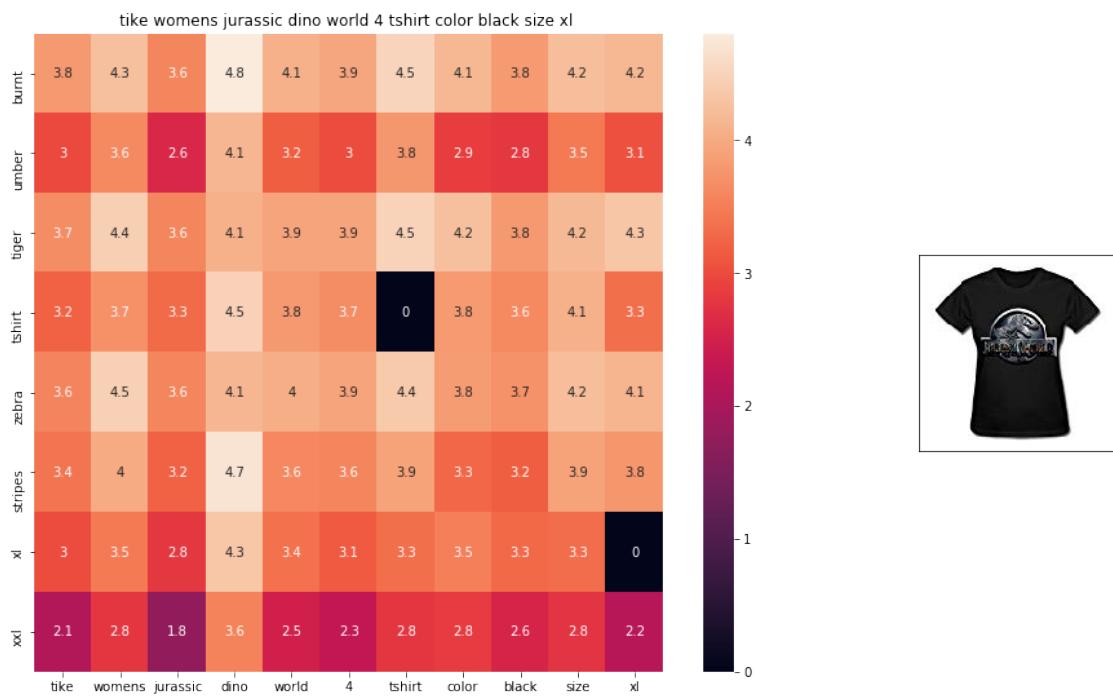
euclidean distance from given input image : 1.0942024



ASIN : B0716MVPGV

BRAND : V.Secret

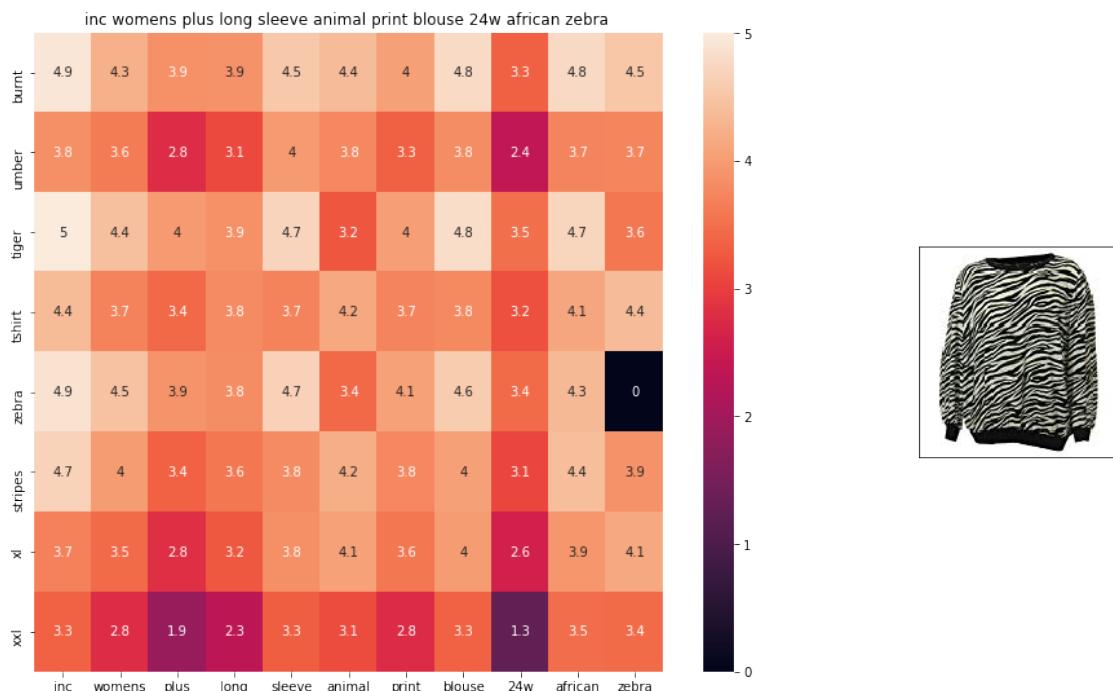
euclidean distance from given input image : 1.0948304



ASIN : B0160PN40I

BRAND : TIKE Fashions

euclidean distance from given input image : 1.0951275



```

ASIN : B018WDJCUA
BRAND : INC - International Concepts Woman
euclidean distance from given input image : 1.0966892
=====

```

3.0.2 [9.4] IDF weighted Word2Vec for product similarity

```

In [17]: 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 d
        w2v_title_weight = np.array(w2v_title_weight)

In [18]: 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 remaining
        # 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].res

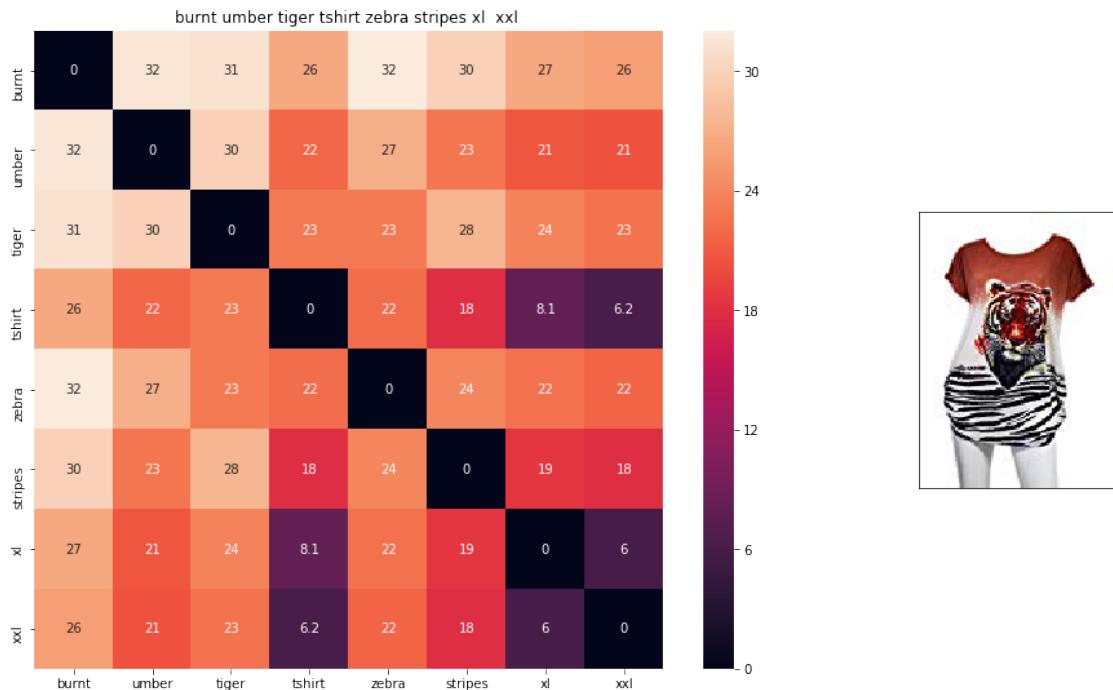
        # np.argsort will return indices of 9 smallest distances
        indices = np.argsort(pairwise_dist.flatten())[0:num_results]
        #pdists will store the 9 smallest distances
        pdists = np.sort(pairwise_dist.flatten())[0:num_results]

        #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[i]])
            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, 10)
        #931
        #12566
        # in the give heat map, each cell contains the euclidean distance between words i, j

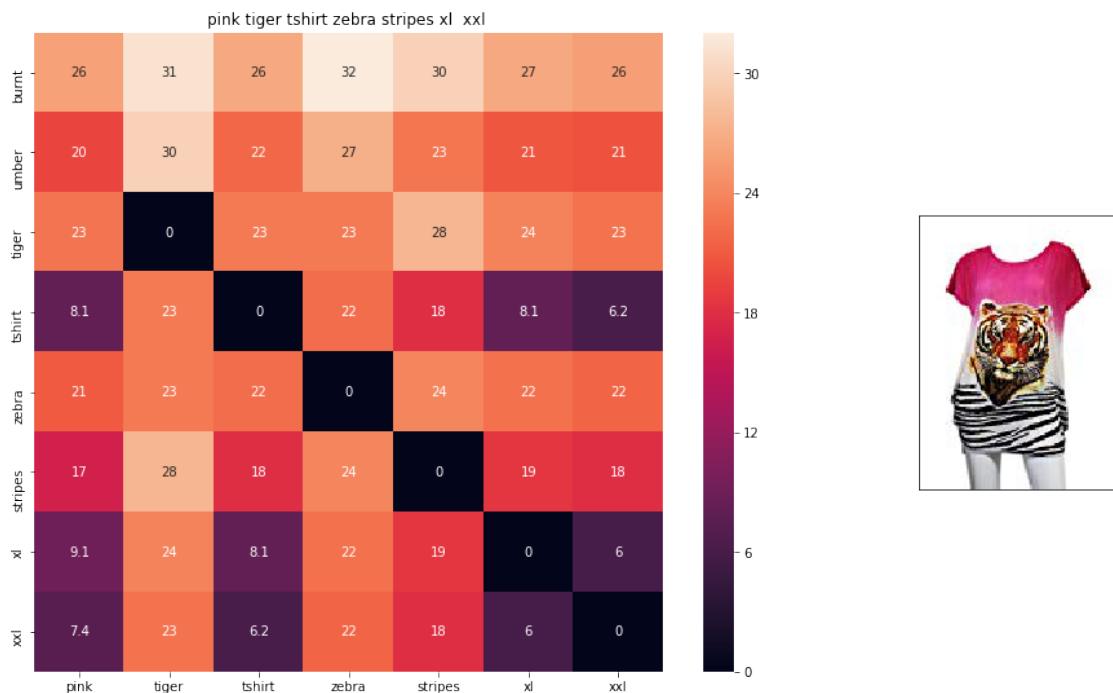
```



ASIN : B00JXQB5FQ

Brand : Si Row

euclidean distance from input : 0.0

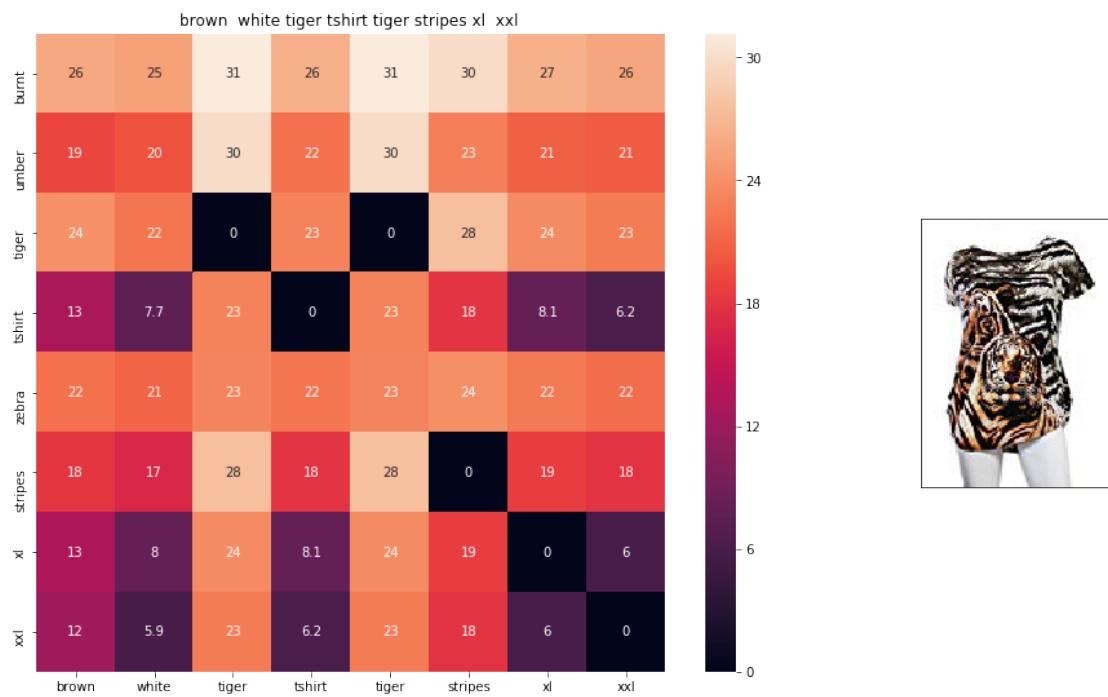


ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 4.0638866

=====

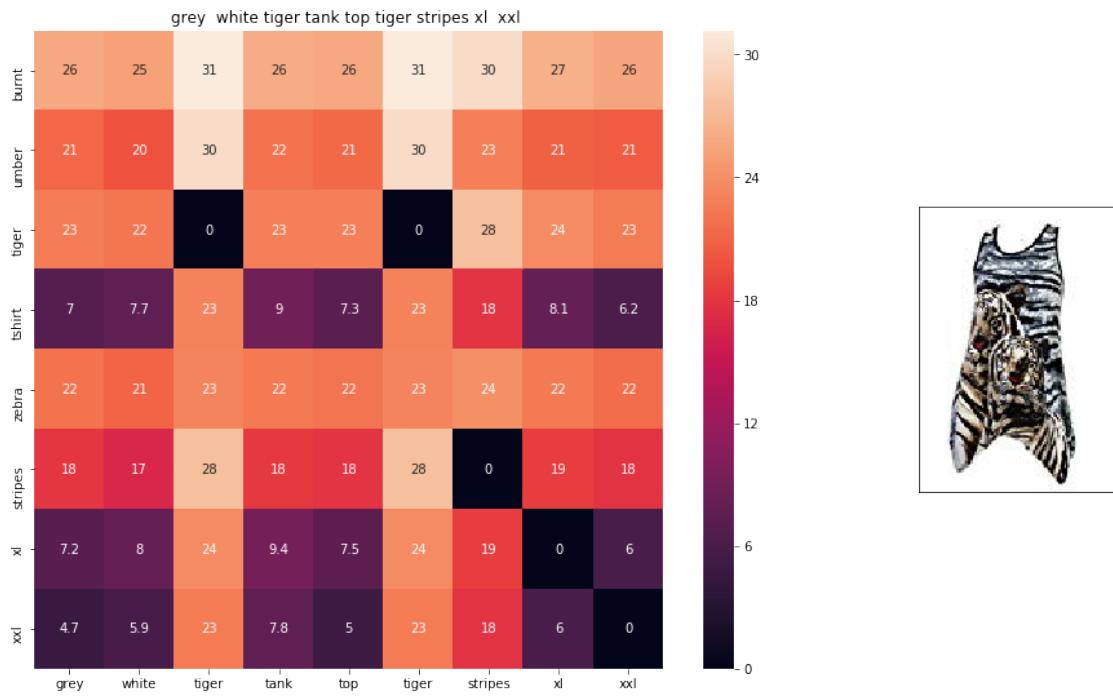


ASIN : B00JXQCWT0

Brand : Si Row

euclidean distance from input : 4.7709413

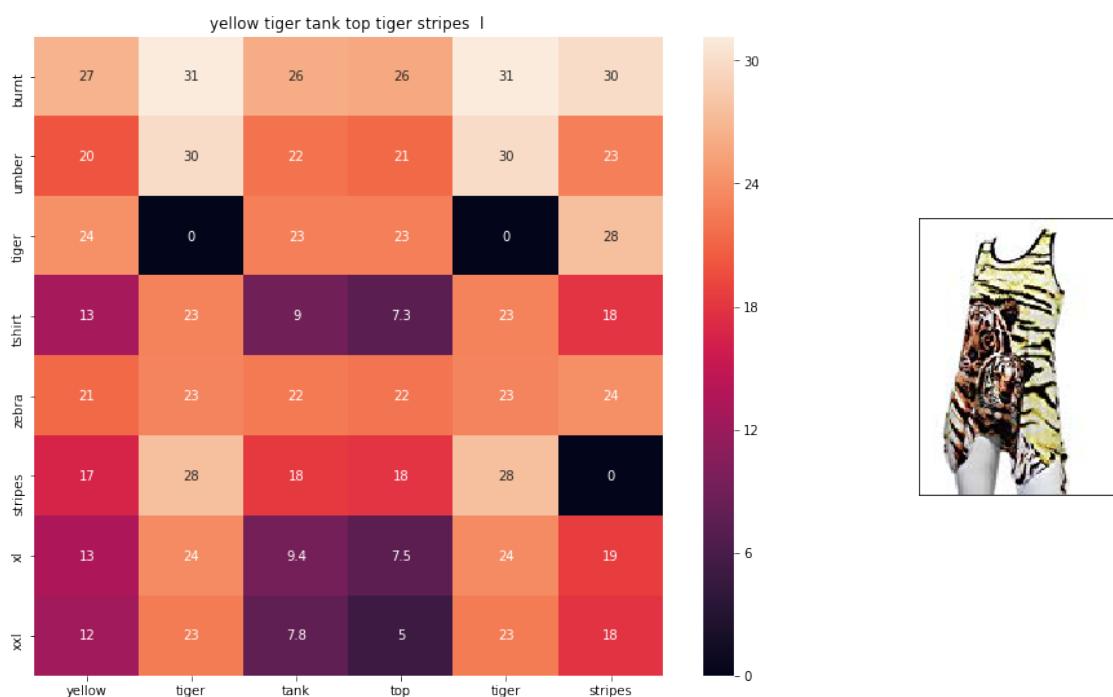
=====



ASIN : B00JXQAFZ2

Brand : Si Row

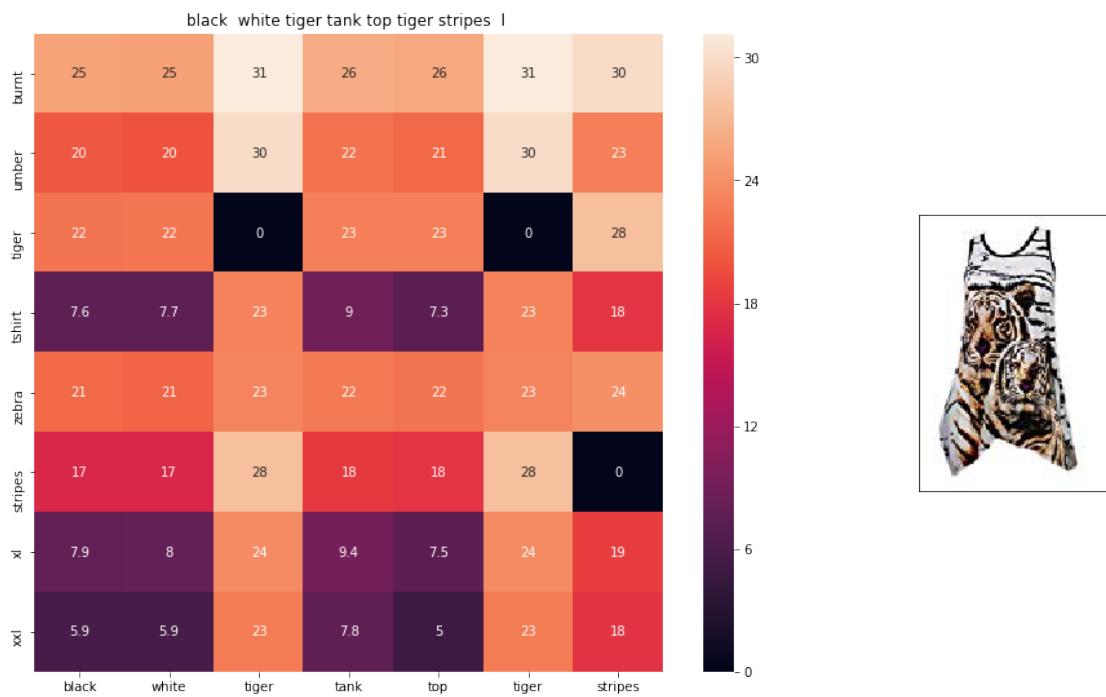
euclidean distance from input : 5.3601604



ASIN : BOOJXQAUWA

Brand : Si Row

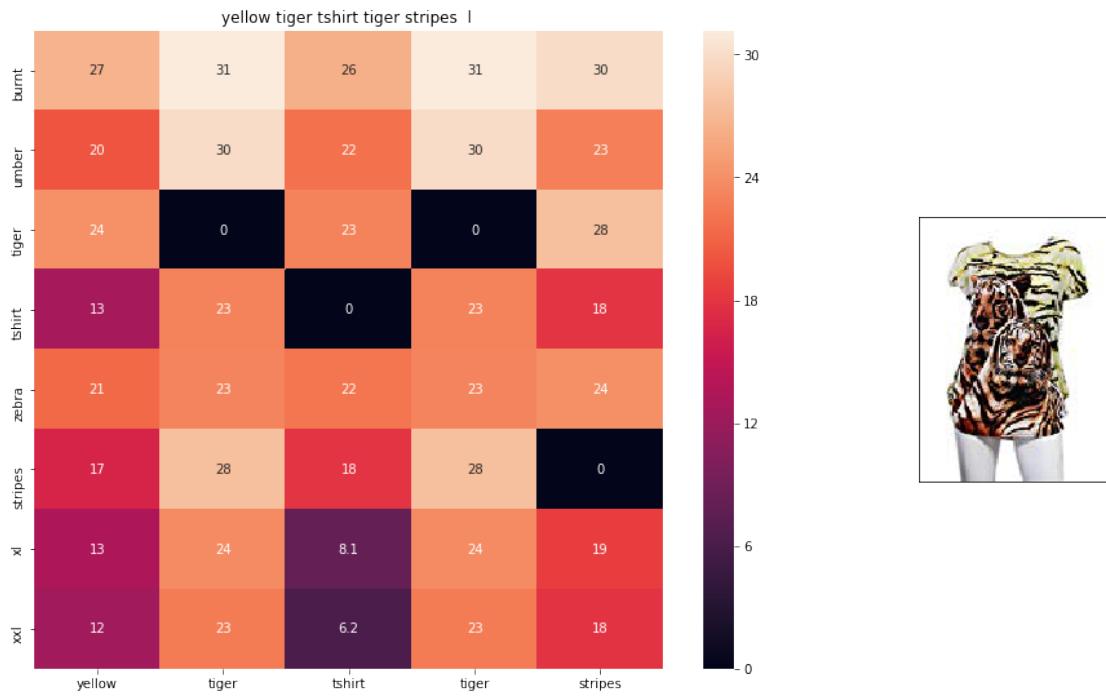
euclidean distance from input : 5.6895227



ASIN : BOOJXQA094

Brand : Si Row

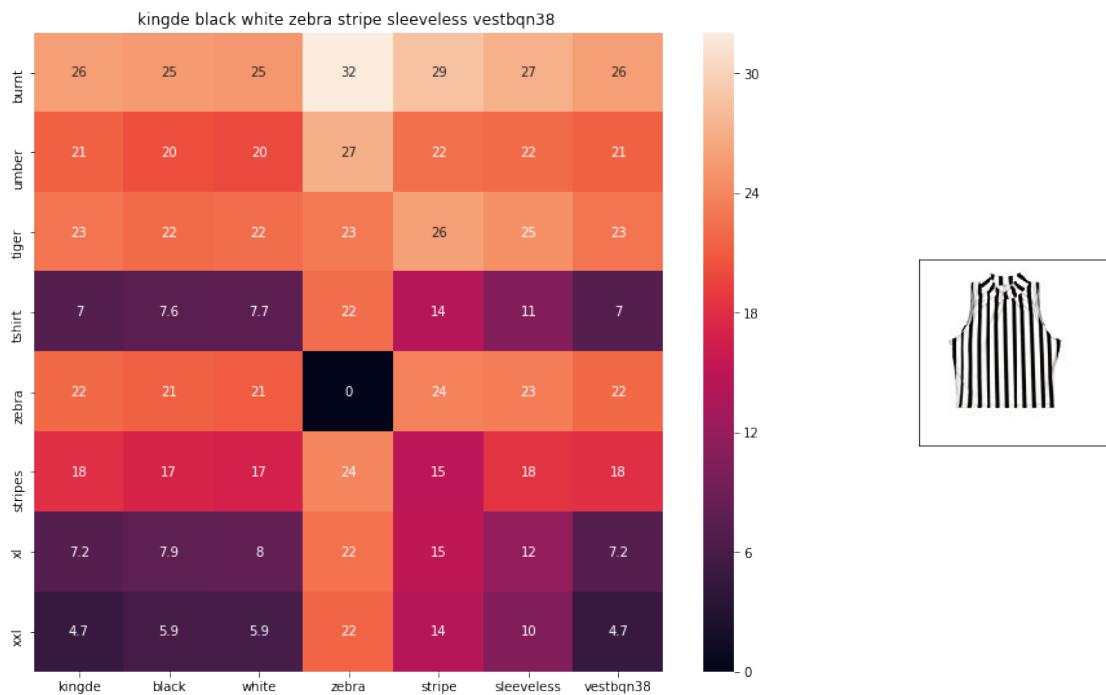
euclidean distance from input : 5.693021



ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 5.893442

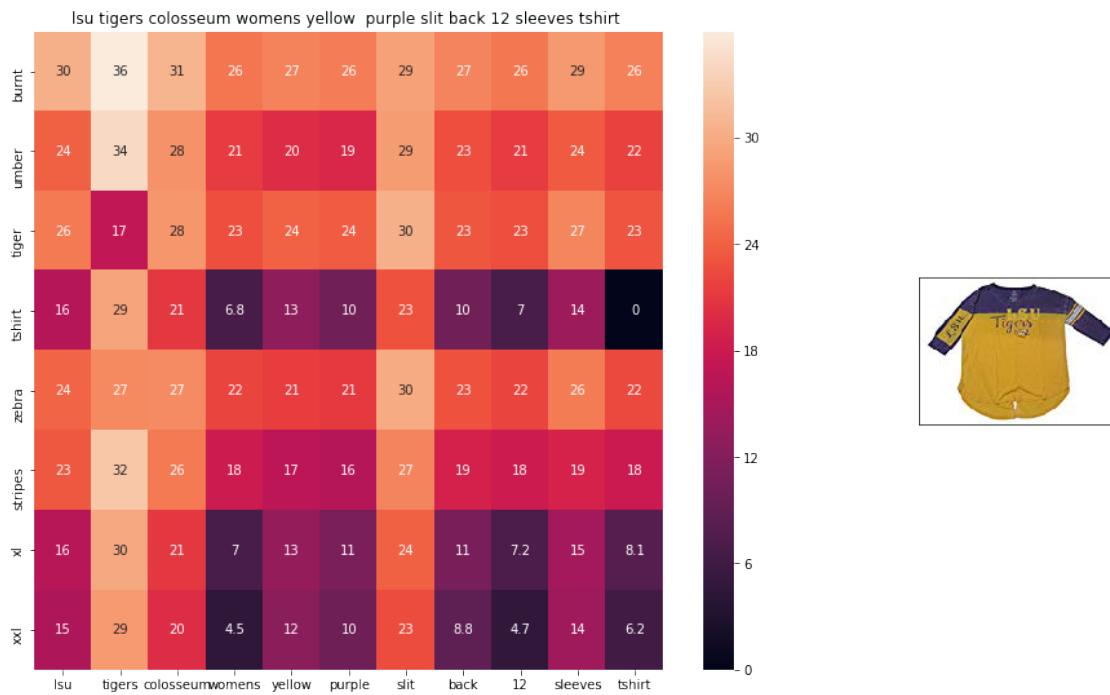


ASIN : B015H41F6G

Brand : KINGDE

euclidean distance from input : 6.1329894

=====

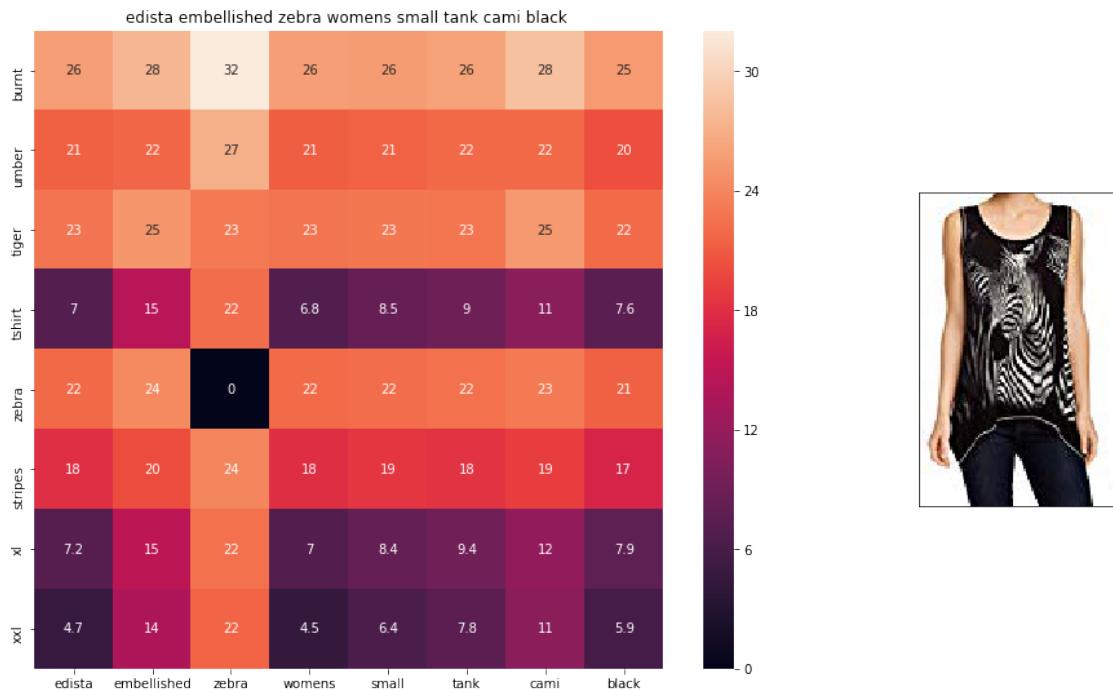


ASIN : B073R5Q8HD

Brand : Colosseum

euclidean distance from input : 6.2567053

=====



ASIN : B074P8MD22

Brand : Edista

euclidean distance from input : 6.3922033

3.0.3 [9.6] Weighted similarity using brand and color.

```
In [19]: # 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 hyphen
      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 [45]: def heat_map_w2v_brand(sentance1, sentance2, url, doc_id1, doc_id2, df_id1, df_id2, m

    # 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(weighted/
s1_vec = get_word_vec(sentance1, doc_id1, model)
#s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighted/
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[doc_id1]],
               [data['asin'].loc[df_id2], brands[doc_id2], colors[doc_id2], types[doc_id2]]]

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

# 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])
# plotting 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 lines 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 [46]: 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 remaining
    # the metric we used here is cosine, the coside distance is measured as  $K(X, Y) = \frac{X \cdot Y}{\|X\| \|Y\|}$ 
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].reshape(1, -1))
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    pairwise_dist = (w1 * idf_w2v_dist + w2 * ex_feat_dist)/float(w1 + w2)

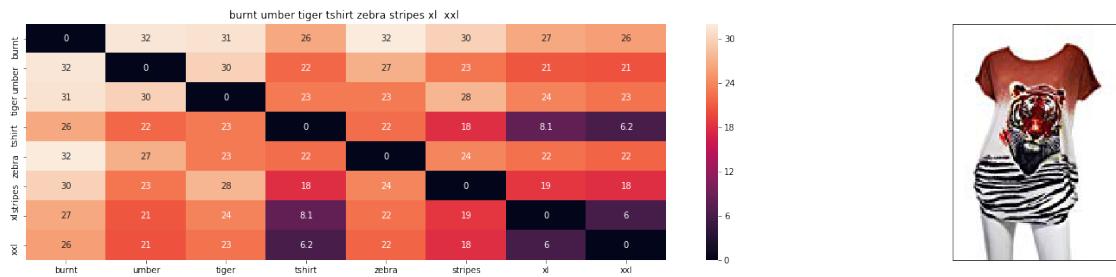
    # np.argsort will return indices of 9 smallest distances
    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    #pdists will store the 9 smallest distances
    pdists = np.sort(pairwise_dist.flatten())[0:num_results]

    #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_indices[i]])
        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, j

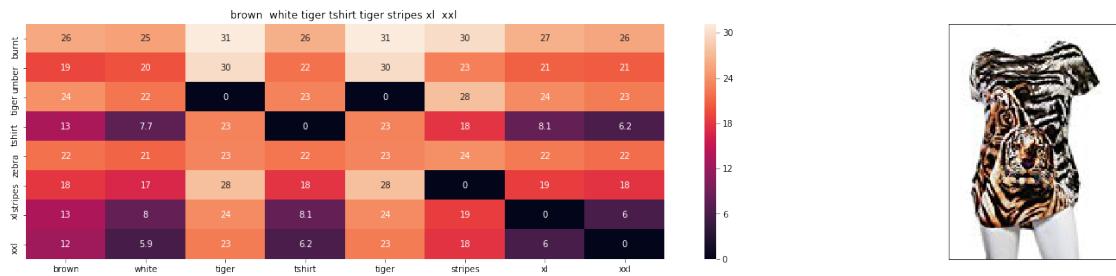
```



ASIN : B00JXQB5FQ

Brand : Si Row

euclidean distance from input : 0.0



ASIN : B00JXQCWT0

Brand : Si Row

euclidean distance from input : 2.3854705810546877



ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 2.739050102414575

=====

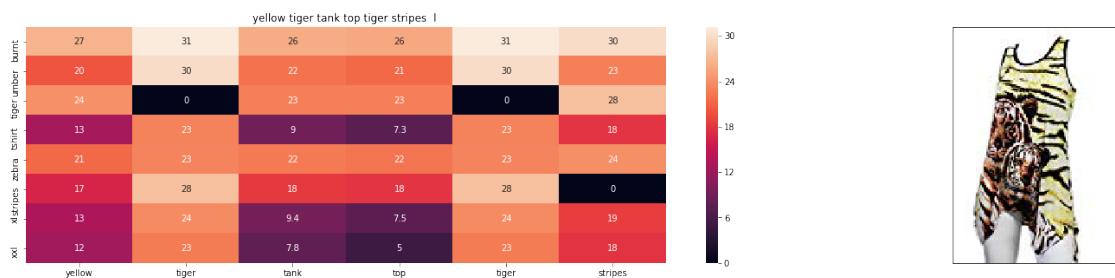


ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from input : 3.387187004270044

=====

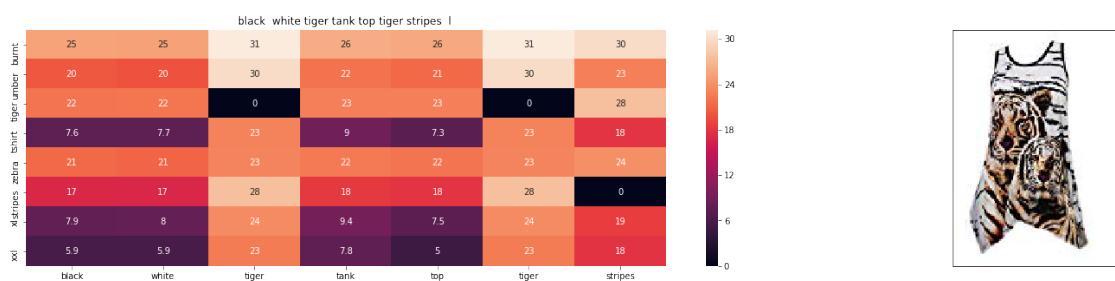


ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from input : 3.5518680574316646

=====

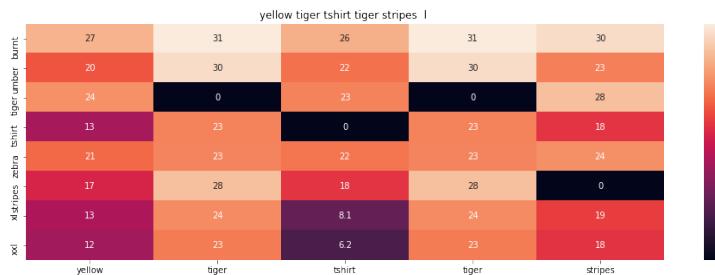


ASIN : B00JXQA094

Brand : Si Row

euclidean distance from input : 3.5536170961279536

=====

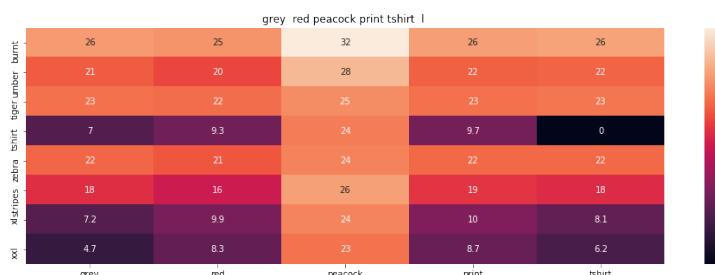


ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 3.6538278581518795

=====

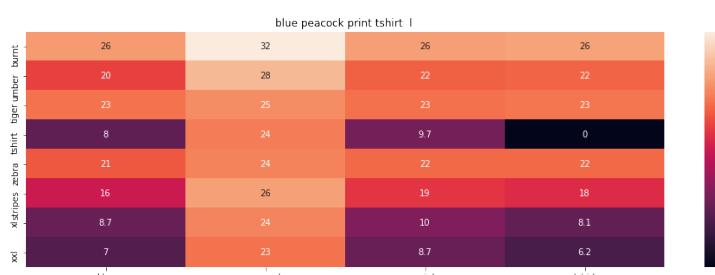


ASIN : B00JXQCFRS

Brand : Si Row

euclidean distance from input : 4.128811264218774

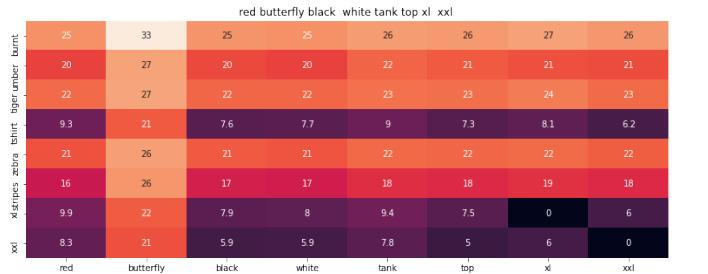
=====



ASIN : B00JXQC8L6

Brand : Si Row

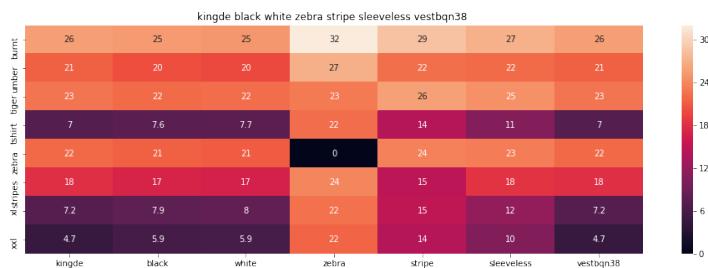
euclidean distance from input : 4.203900146665063



ASIN : B00JV63CW2

Brand : Si Row

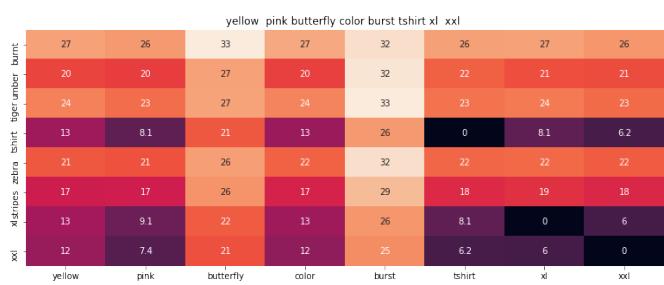
euclidean distance from input : 4.286586380185571



ASIN : B015H41F6G

Brand : KINGDE

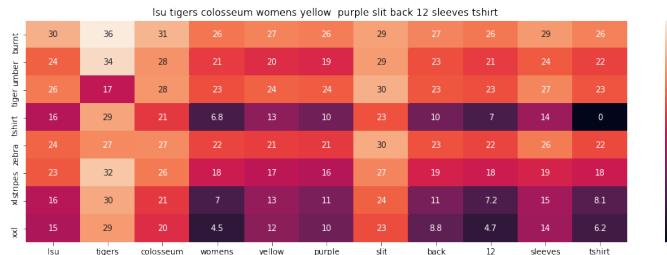
euclidean distance from input : 4.389370406508858



ASIN : B00JXQBBMI

Brand : Si Row

euclidean distance from input : 4.397909927548852



ASIN : B073R5Q8HD

Brand : Colosseum

euclidean distance from input : 4.451228392959053



ASIN : B074P8MD22

Brand : Edista

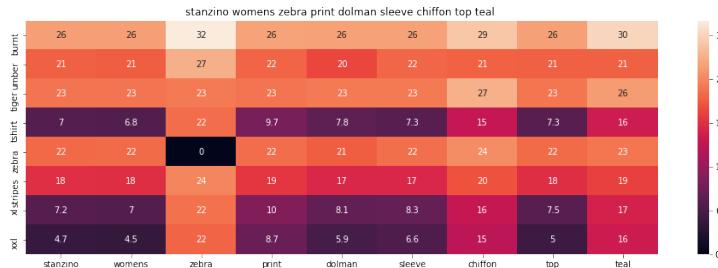
euclidean distance from input : 4.518977416396553



ASIN : B00JV63QQE

Brand : Si Row

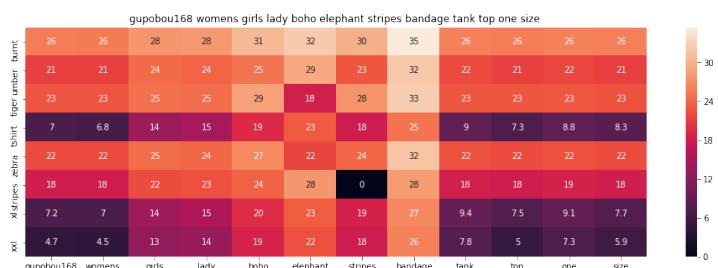
euclidean distance from input : 4.529374695004907



ASIN : B00C0I3U3E

Brand : Stanzino

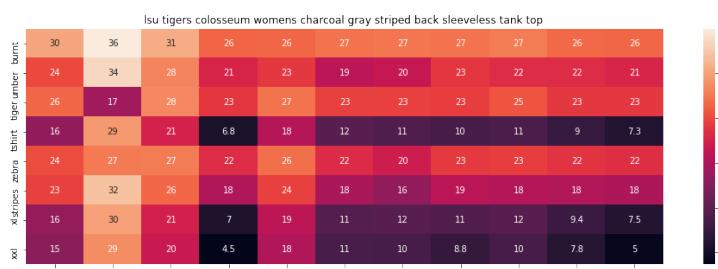
euclidean distance from input : 4.530325759292061



ASIN : B01ER18406

Brand : GuPoBoU168

euclidean distance from input : 4.546816642558488

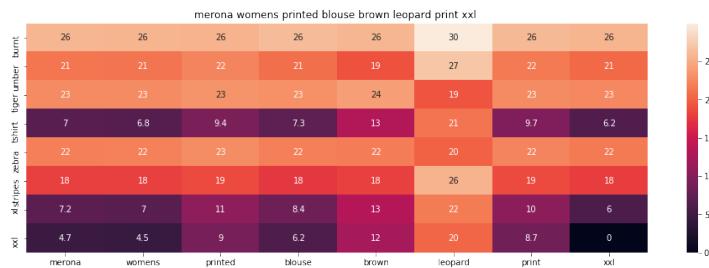


ASIN : B073R4ZM7Y

Brand : Colosseum

euclidean distance from input : 4.548355162978584

=====



ASIN : B071YF3WDD

Brand : Merona

euclidean distance from input : 4.610626662612374

=====



ASIN : B01C60RLDQ

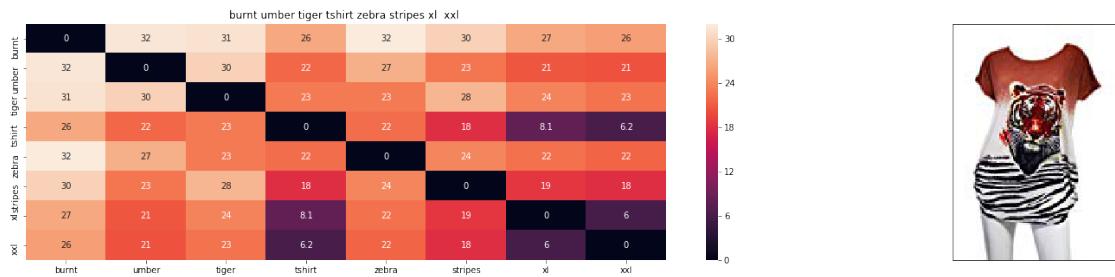
Brand : 1 Mad Fit

euclidean distance from input : 4.645917892817431

=====

In [121]: # brand and color weight =50
title vector weight = 5

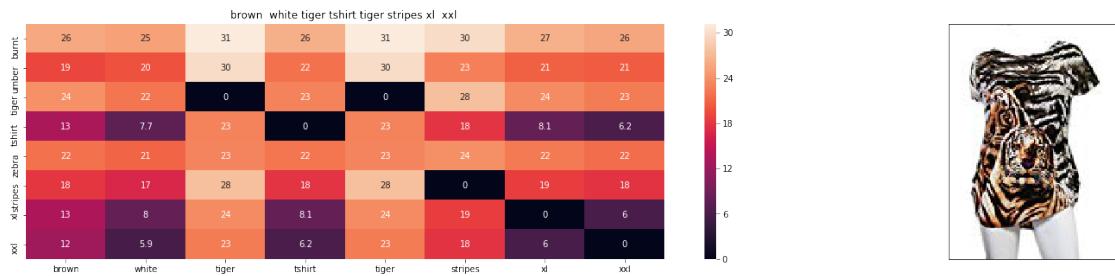
idf_w2v_brand(12566, 5, 50, 20)



ASIN : B00JXQB5FQ

Brand : Si Row

euclidean distance from input : 0.0



ASIN : B00JXQCWT0

Brand : Si Row

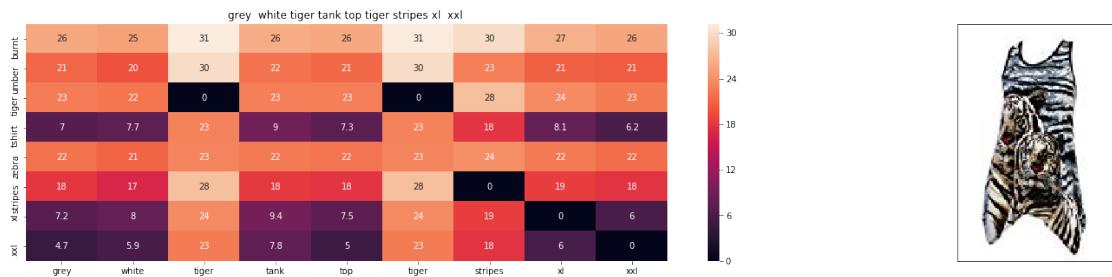
euclidean distance from input : 0.433721923828125



ASIN : B00JXQASS6

Brand : Si Row

euclidean distance from input : 1.6550929332897277



ASIN : B00JXQAFZ2

Brand : Si Row

euclidean distance from input : 1.7729360063543584



ASIN : B00JXQAUWA

Brand : Si Row

euclidean distance from input : 1.8028780160201077

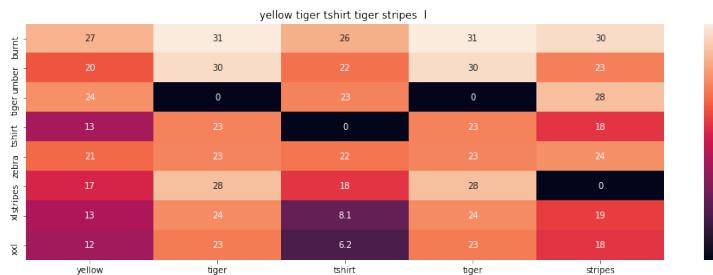


ASIN : B00JXQA094

Brand : Si Row

euclidean distance from input : 1.8031960230557966

=====

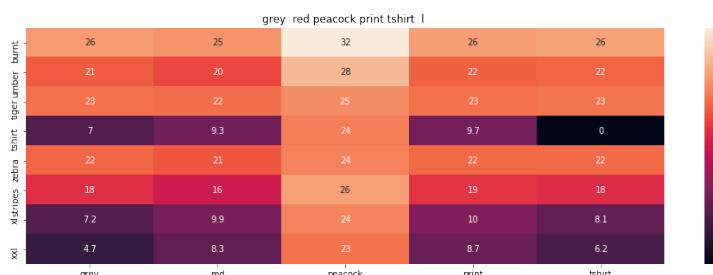


ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 1.8214161616056013

=====

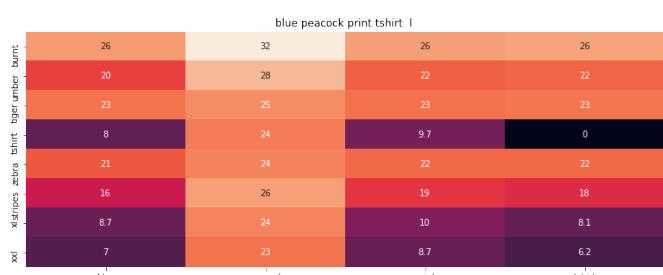


ASIN : B00JXQCFRS

Brand : Si Row

euclidean distance from input : 1.9077767808904913

=====

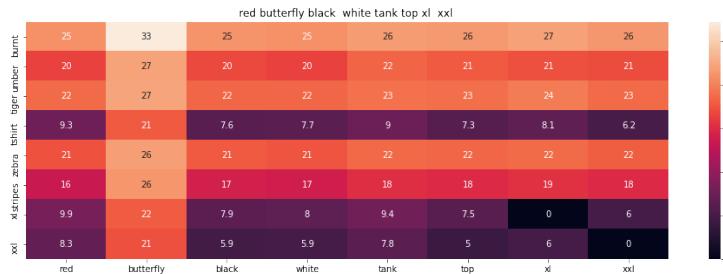


ASIN : B00JXQC8L6

Brand : Si Row

euclidean distance from input : 1.9214293049716347

=====

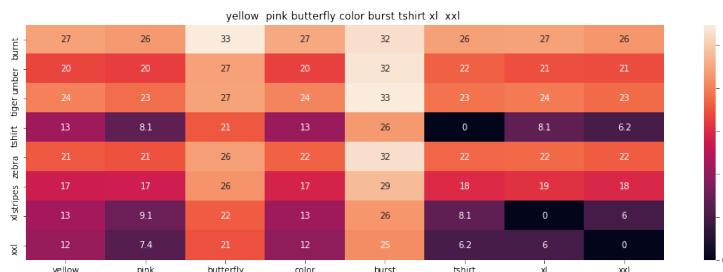


ASIN : B00JV63CW2

Brand : Si Row

euclidean distance from input : 1.936463165611727

=====

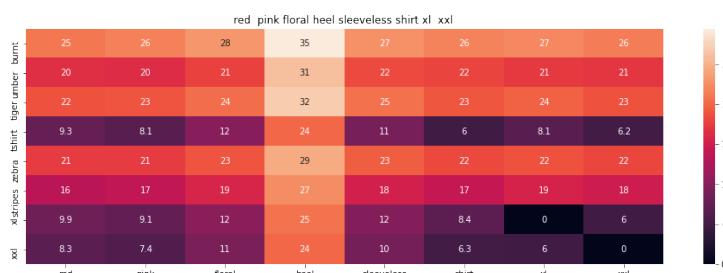


ASIN : B00JXQBBMI

Brand : Si Row

euclidean distance from input : 1.956703810586869

=====



ASIN : B00JV63QQE

Brand : Si Row

euclidean distance from input : 1.9806064955788791

=====

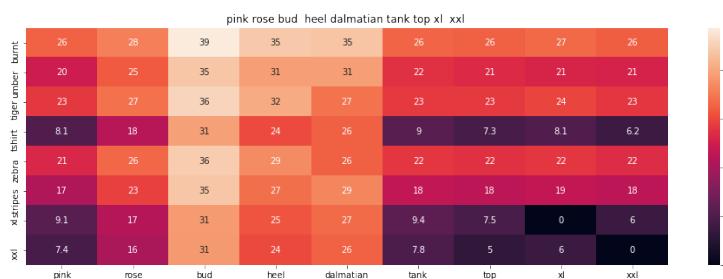


ASIN : B00JV63VC8

Brand : Si Row

euclidean distance from input : 2.012185530557572

=====

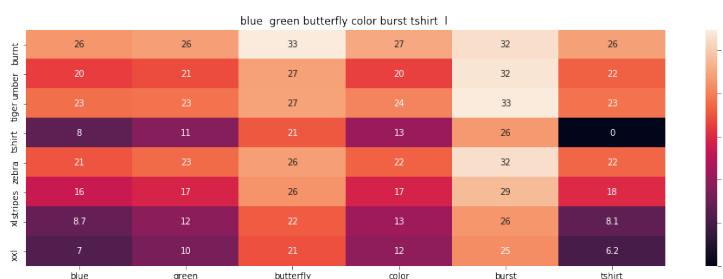


ASIN : B00JXQAX2C

Brand : Si Row

euclidean distance from input : 2.01335171819074

=====

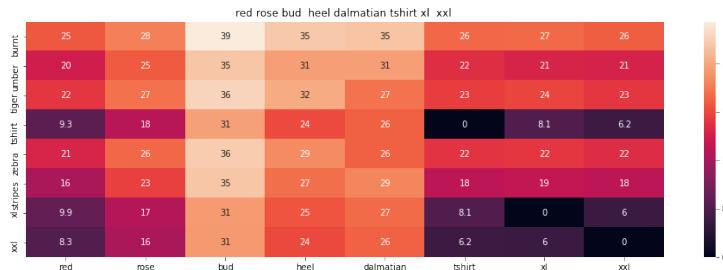


ASIN : B00JXQCOC8

Brand : Si Row

euclidean distance from input : 2.0138832789151717

=====

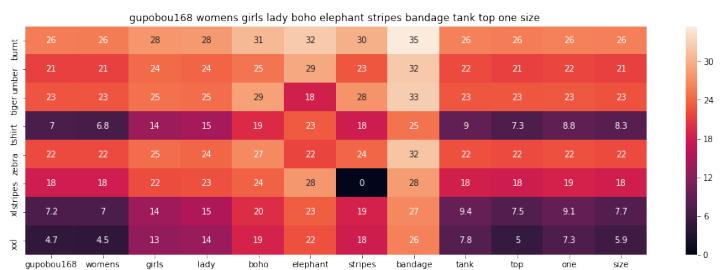


ASIN : B00JXQABBO

Brand : Si Row

euclidean distance from input : 2.0367257554998663

=====

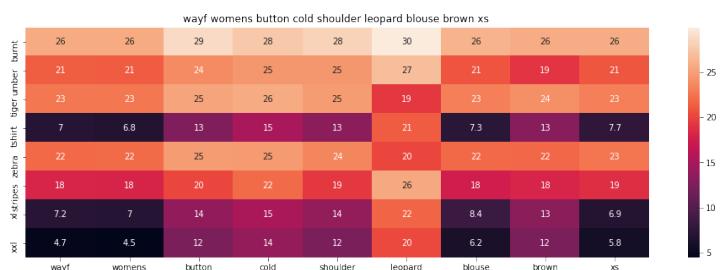


ASIN : B01ER18406

Brand : GuPoBoU168

euclidean distance from input : 2.656204098419553

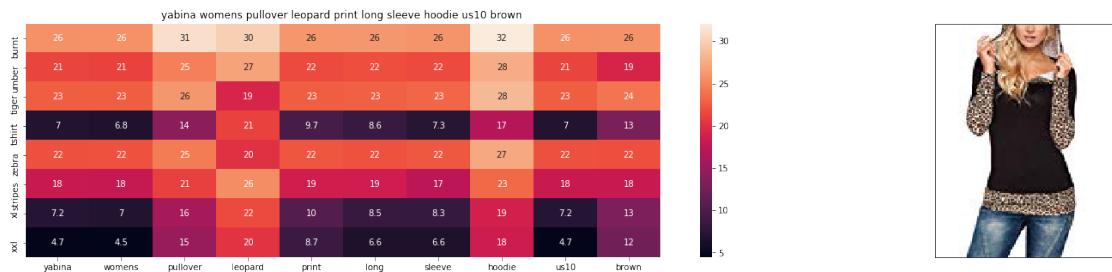
=====



ASIN : B01LZ7BQ4H

Brand : WAYF

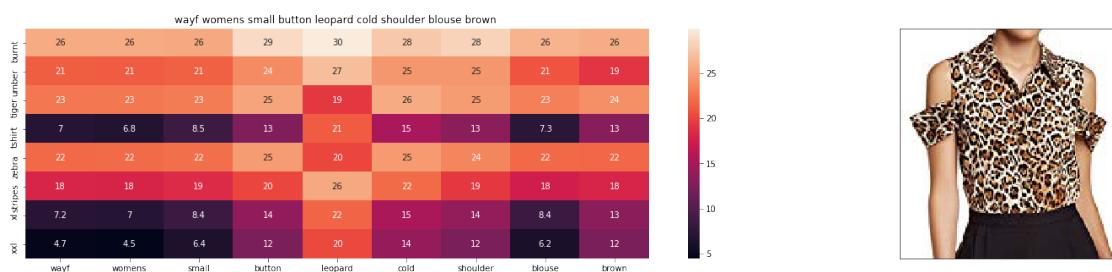
euclidean distance from input : 2.6849067129437008



ASIN : B01KJUM6JI

Brand : YABINA

euclidean distance from input : 2.6858381232997024



ASIN : B01M06V4X1

Brand : WAYF

euclidean distance from input : 2.694761948650377

4 [10.2] Keras and Tensorflow to extract features

```
In [22]: import numpy as np
        from keras.preprocessing.image import ImageDataGenerator
```

```

from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras import applications
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt
import requests
from PIL import Image
import pandas as pd
import pickle

```

Using TensorFlow backend.

```

In [0]: # https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069
# Code reference: https://blog.keras.io/building-powerful-image-classification-models--.html

# This code takes 40 minutes to run on a modern GPU (graphics card)
# like Nvidia 1050.
# GPU (Nvidia 1050): 0.175 seconds per image

# This code takes 160 minutes to run on a high end i7 CPU
# CPU (i7): 0.615 seconds per image.

#Do NOT run this code unless you want to wait a few hours for it to generate output

# each image is converted into 25088 length dense-vector

'''

# dimensions of our images.
img_width, img_height = 224, 224

top_model_weights_path = 'bottleneck_fc_model.h5'
train_data_dir = 'images2/'
nb_train_samples = 16042
epochs = 50
batch_size = 1

def save_bottlebeck_features():

    #Function to compute VGG-16 CNN for image feature extraction.

    asins = []
    datagen = ImageDataGenerator(rescale=1. / 255)

```

```

# build the VGG16 network
model = applications.VGG16(include_top=False, weights='imagenet')
generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

for i in generator.filenames:
    asins.append(i[2:-5])

bottleneck_features_train = model.predict_generator(generator, nb_train_samples // 10)
bottleneck_features_train = bottleneck_features_train.reshape((16042, 25088))

np.save(open('16k_data_cnn_features.npy', 'wb'), bottleneck_features_train)
np.save(open('16k_data_cnn_feature_asins.npy', 'wb'), np.array(asins))

save_bottlebeck_features()

...

```

5 [10.3] Visual features based product similarity.

```

In [23]: #load the features and corresponding ASINS info.
bottleneck_features_train = np.load('16k_data_cnn_features_2.npy')
asins = np.load('16k_data_cnn_feature_asins.npy')
asins = list(asins)

# load the original 16K dataset
data = pd.read_pickle('pickels/16k_apperial_data_preprocessed')
df_asins = list(data['asin'])

from IPython.display import display, Image, SVG, Math, YouTubeVideo

#get similar products using CNN features (VGG-16)
def get_similar_products_cnn(doc_id, num_results):
    doc_id = asins.index(df_asins[doc_id])
    pairwise_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_train)

    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    pdists = np.sort(pairwise_dist.flatten())[0:num_results]

    for i in range(len(indices)):

```

```

rows = data[['medium_image_url','title']].loc[data['asin']==asins[indices[i]]]
for idx, row in rows.iterrows():
    display(Image(url=row['medium_image_url'], embed=True))
    print('Product Title: ', row['title'])
    print('Euclidean Distance from input image:', pdists[i])
    print('Amazon Url: www.amazon.com/dp/' + asins[indices[i]])

get_similar_products_cnn(12565, 10)

```



Product Title: mamilisari always dobby crop top time harry potter shirt tshirt tee
 Euclidean Distance from input image: 5.2234896e-06
 Amazon Url: www.amazon.com/dp/B01M014EA0



Product Title: mamilisari always draco malfoy crop top parody harry potter shirt tshirt tee
 Euclidean Distance from input image: 22.855217
 Amazon Url: www.amazon.com/dp/B01LXQZYA5



Product Title: mamilisari time crop top always severus snape shirt tshirt tee white

Euclidean Distance from input image: 24.955975

Amazon Url: www.amazon.com/dp/B01M1BXA2V



Product Title: mamilisari drink know things jon snow crop top shirt tshirt tee

Euclidean Distance from input image: 25.36372

Amazon Url: www.amazon.com/dp/B01LWRXIEH



Product Title: mamilisari studio ghibli crop top cute totoro neighbor shirt tshirt tee

Euclidean Distance from input image: 27.247107

Amazon Url: www.amazon.com/dp/B01LXR02UP



Product Title: mamilisari always severus snape crop top hogwarts time shirt tshirt tee

Euclidean Distance from input image: 27.96179

Amazon Url: www.amazon.com/dp/B01LZPC561



Product Title: mamilisari alice wonderland tattoos crop top cool gothic punk shirt tshirt tee w

Euclidean Distance from input image: 28.558498

Amazon Url: www.amazon.com/dp/B01LYEF2CN



Product Title: mamilisari stop stressing meowt crop top cats lover kitten shirt tshirt tee

Euclidean Distance from input image: 28.727636

Amazon Url: www.amazon.com/dp/B01LZPCD7D



Product Title: mamilari nap queen crop top tee lazy sleep tired shirt tshirt
Euclidean Distance from input image: 29.282055
Amazon Url: www.amazon.com/dp/B01M0ZXUN1

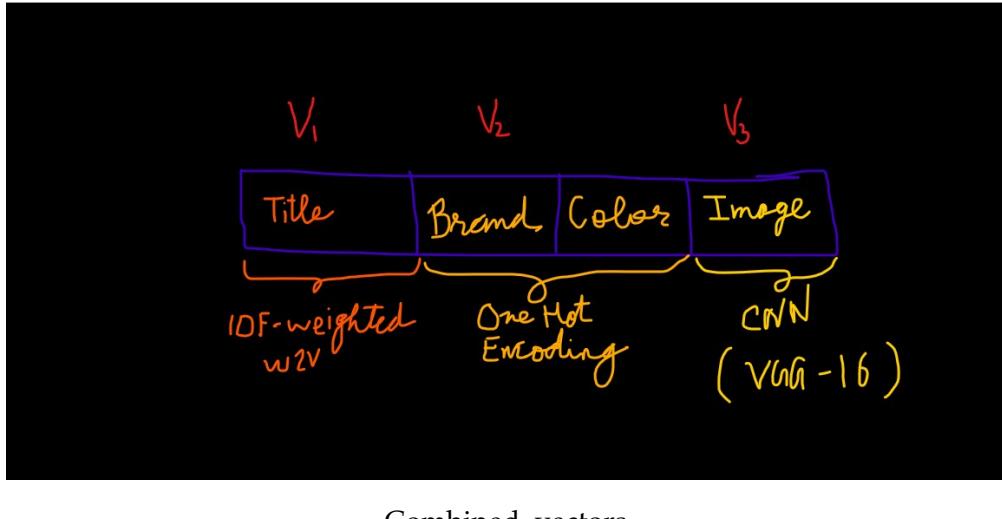


Product Title: solemnly swear good crop top dylan obrien shirt tshirt tee
Euclidean Distance from input image: 29.913097
Amazon Url: www.amazon.com/dp/B01M6ZFB56

5.1 Exercise :

5.1.1 Weighted Euclidean distance based similarity in which we'll be using 4 Vectors.

- Here for finding the similar products, instead of simply just using "title" or "brand" or "color" or "Image" itself individually.
- we'll use a combined vector which includes title vector formed usnig Idf weighted w2v.
- Brand and color vectors formed using onehot encoding and
- We also use image vector formed using CNN(vgg-16 here)



Combined_vectors

5.2 Generating Idf weighted w2v vector

```
In [31]: 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 d
        w2v_title_weight = np.array(w2v_title_weight)

In [32]: 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 remaining
        # 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].resl

        # 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])

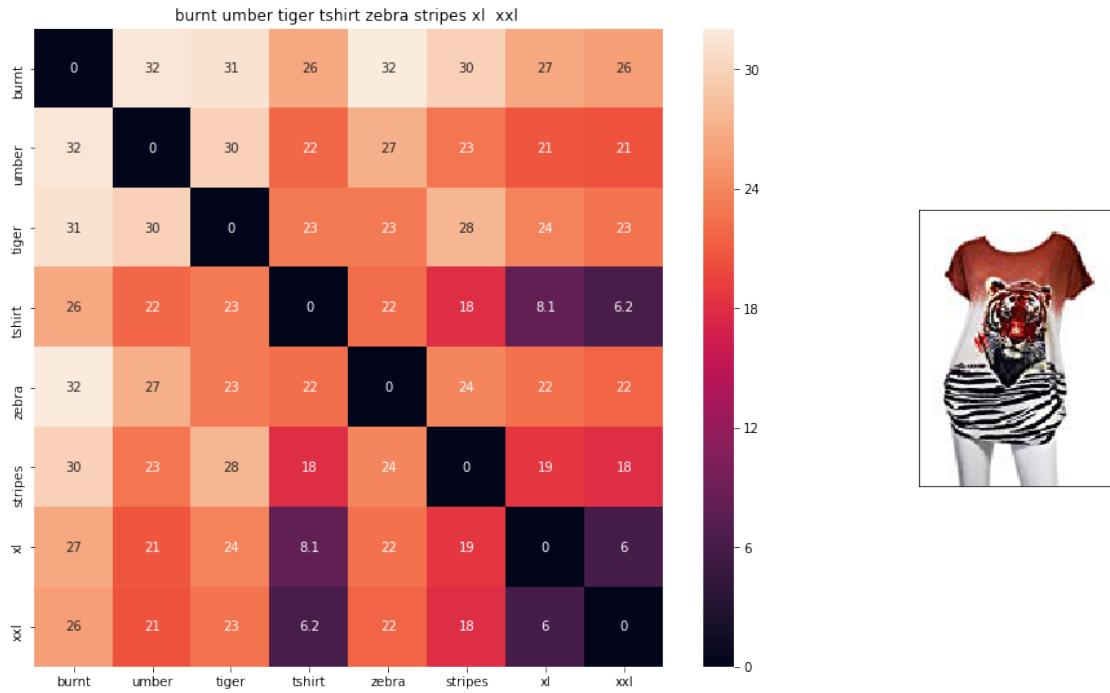
        for i in range(0, len(indices)):
            heat_map_w2v(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]])
            print('ASIN : ', data['asin'].loc[df_indices[i]])
            print('Brand : ', data['brand'].loc[df_indices[i]])
            print('euclidean distance from input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

```

print('=*125)

weighted_w2v_model(12566, 10)
#931
#12566
# in the give heat map, each cell contains the euclidean distance between words i, j

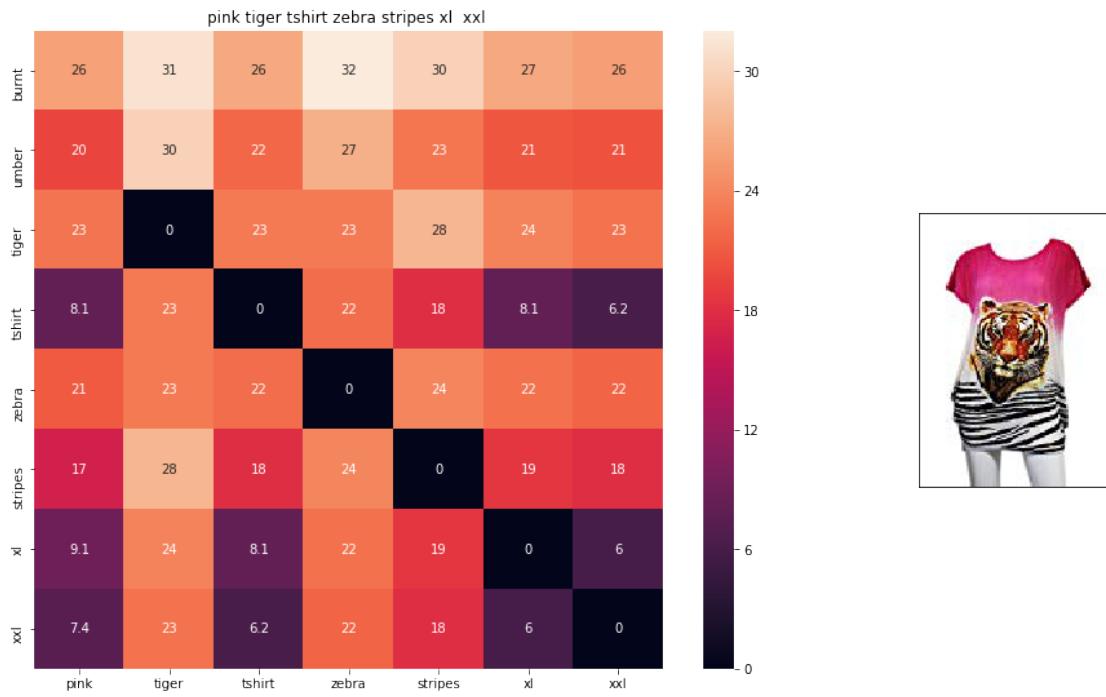
```



ASIN : B00JXQB5FQ

Brand : Si Row

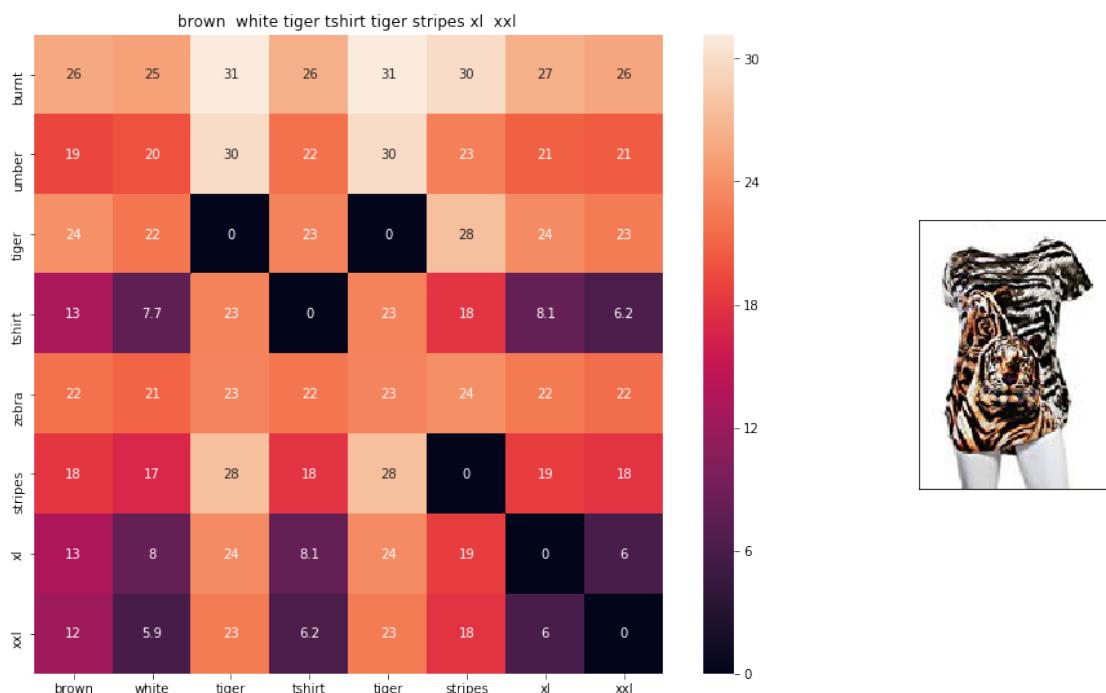
euclidean distance from input : 0.0



ASIN : BOOJXQASS6

Brand : Si Row

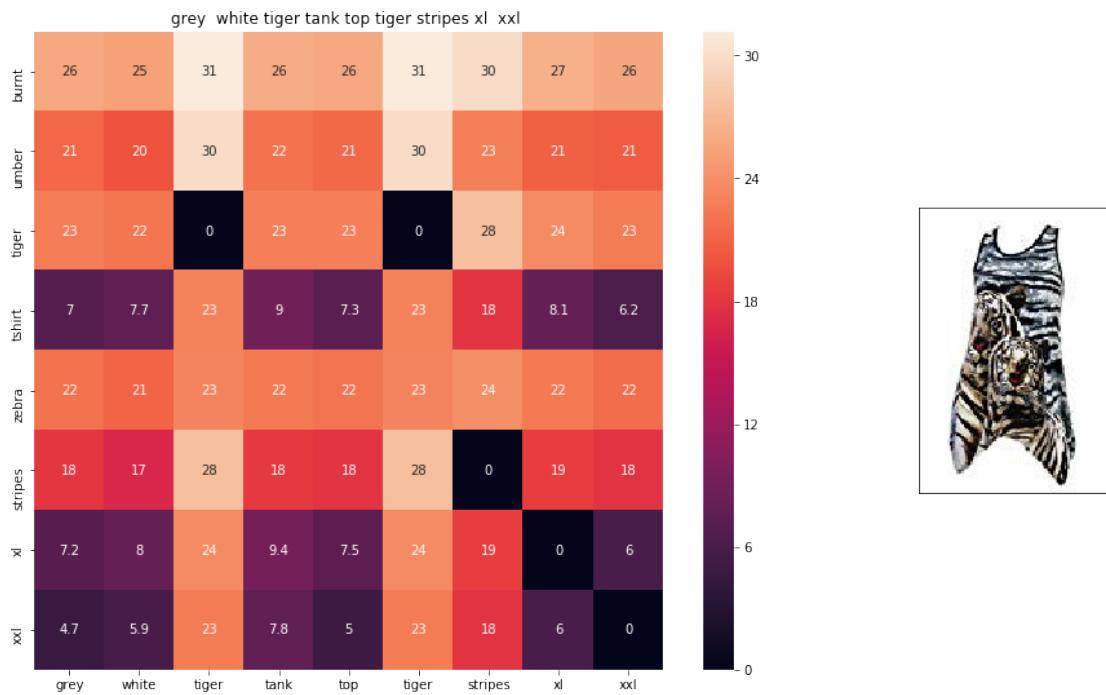
euclidean distance from input : 4.0638866



ASIN : B00JXQCWT0

Brand : Si Row

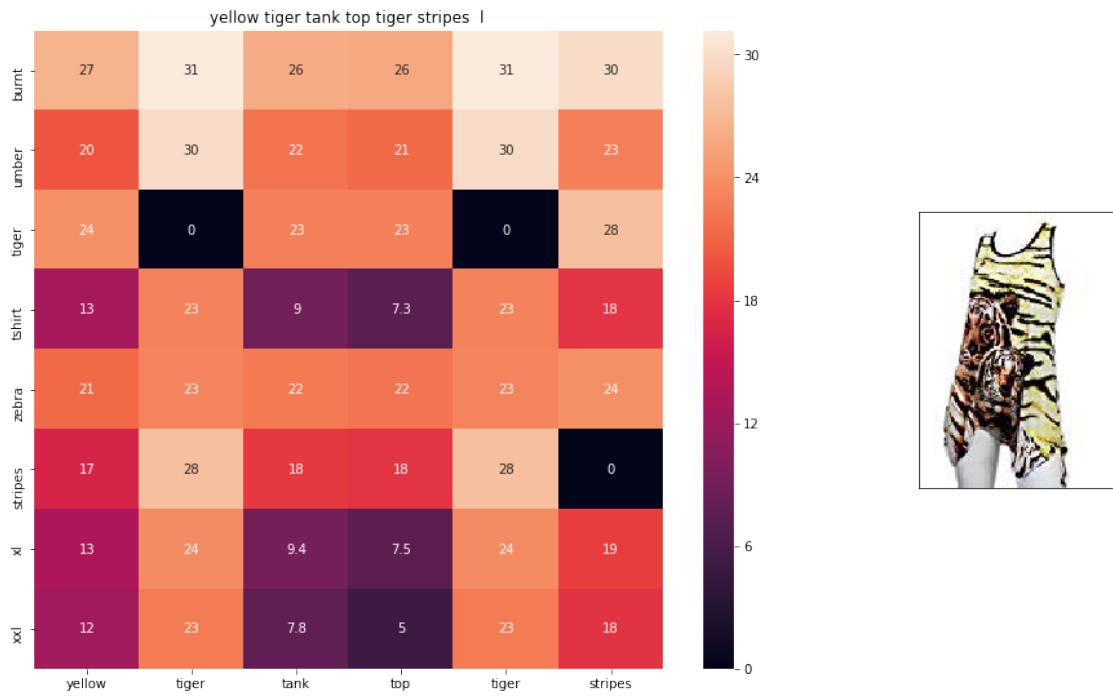
euclidean distance from input : 4.7709413



ASIN : B00JXQAFZ2

Brand : Si Row

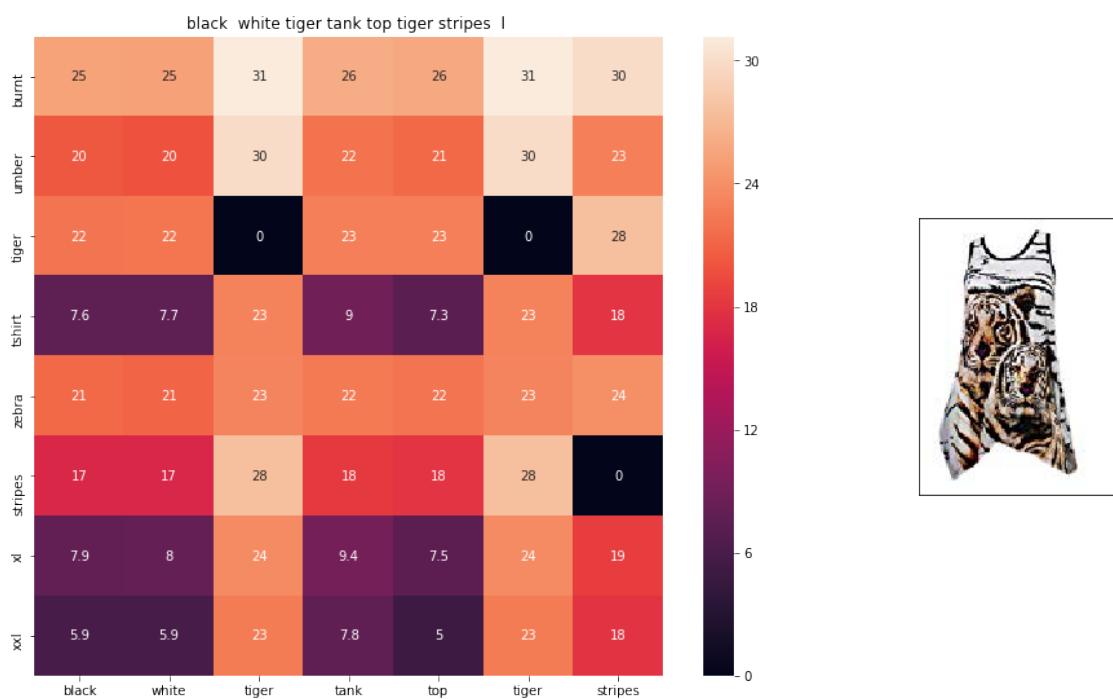
euclidean distance from input : 5.3601604



ASIN : BOOJXQAUWA

Brand : Si Row

euclidean distance from input : 5.6895227

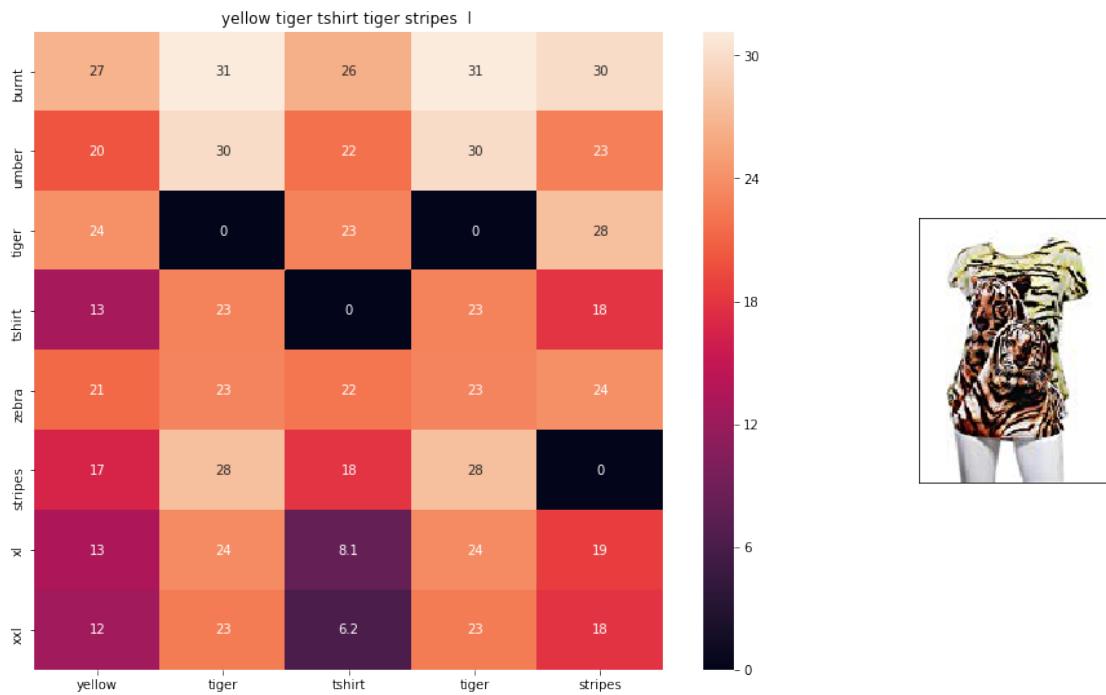


ASIN : B00JXQA094

Brand : Si Row

euclidean distance from input : 5.693021

=====

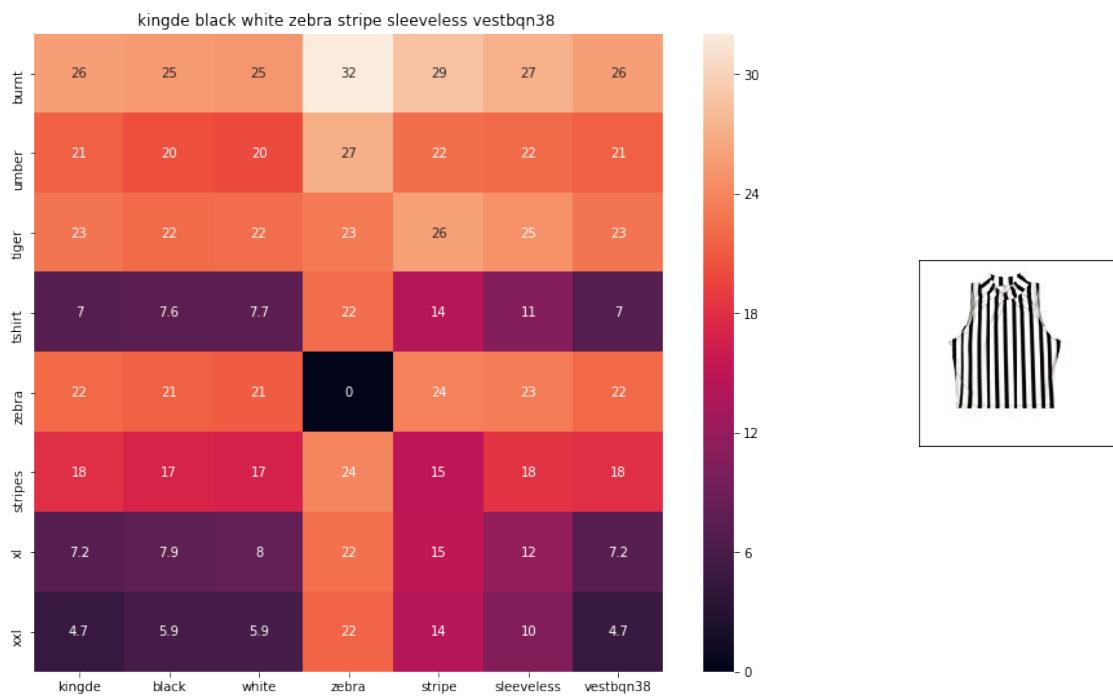


ASIN : B00JXQCUIC

Brand : Si Row

euclidean distance from input : 5.893442

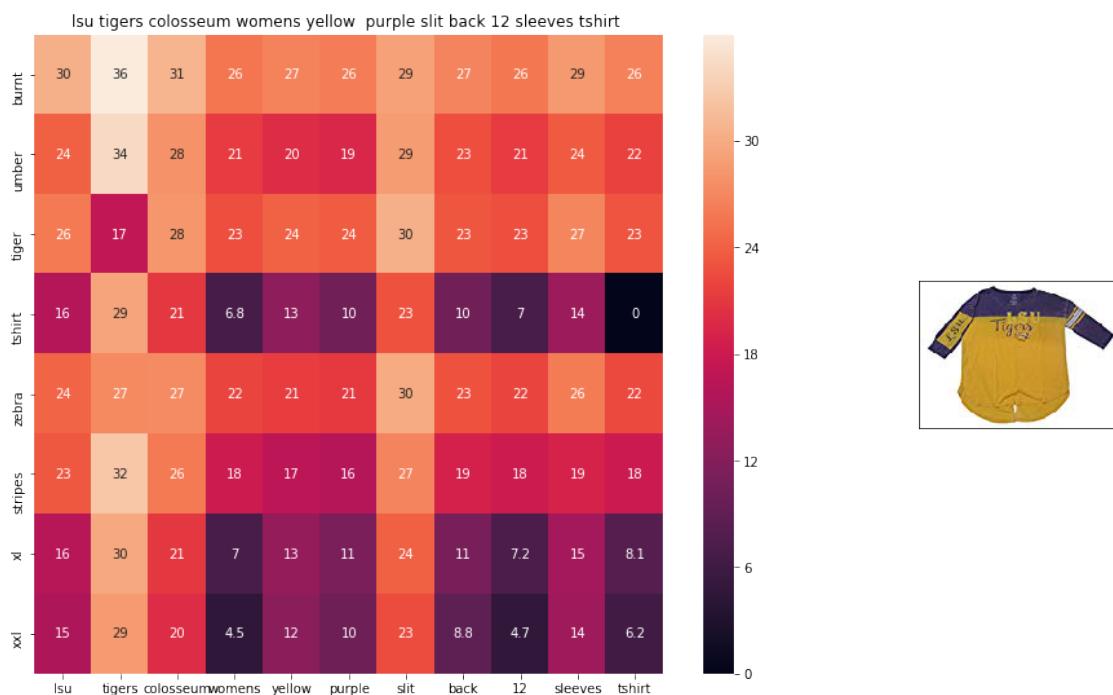
=====



ASIN : B015H41F6G

Brand : KINGDE

euclidean distance from input : 6.1329894

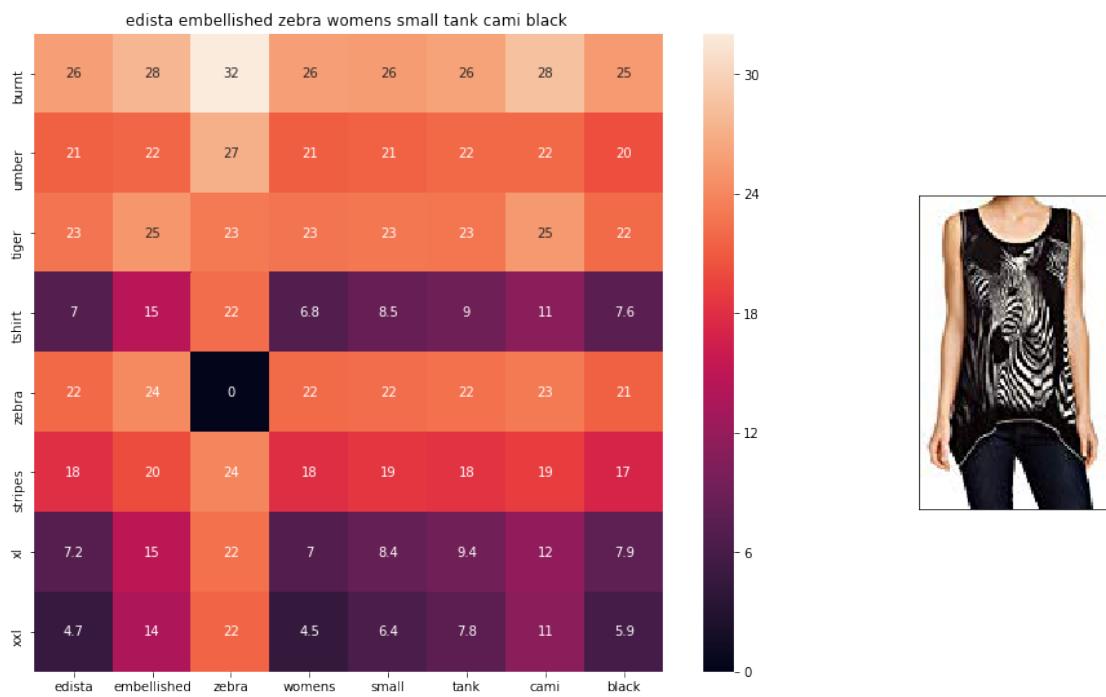


ASIN : B073R5Q8HD

Brand : Colosseum

euclidean distance from input : 6.2567053

=====

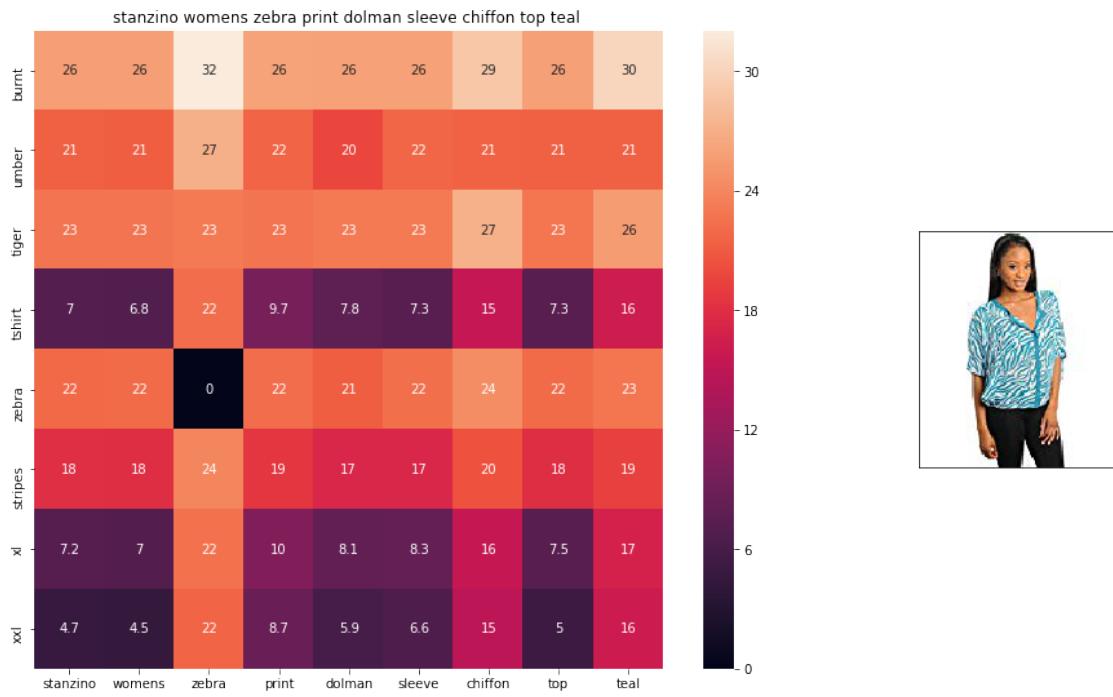


ASIN : B074P8MD22

Brand : Edista

euclidean distance from input : 6.3922033

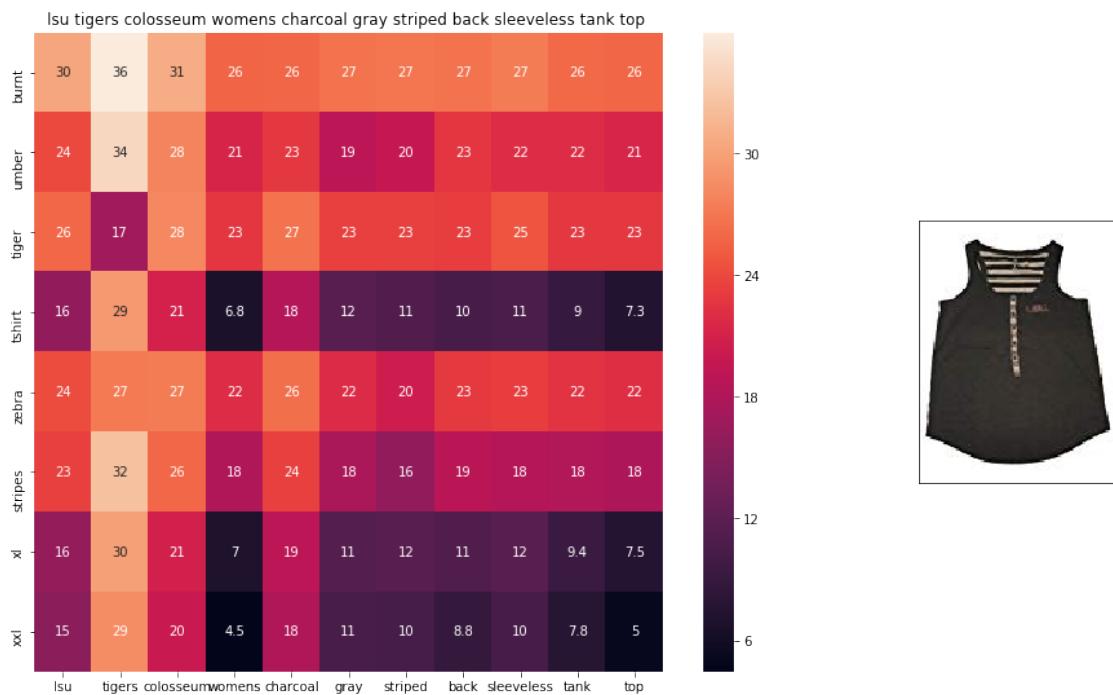
=====



ASIN : B00C0I3U3E

Brand : Stanzino

euclidean distance from input : 6.4149003

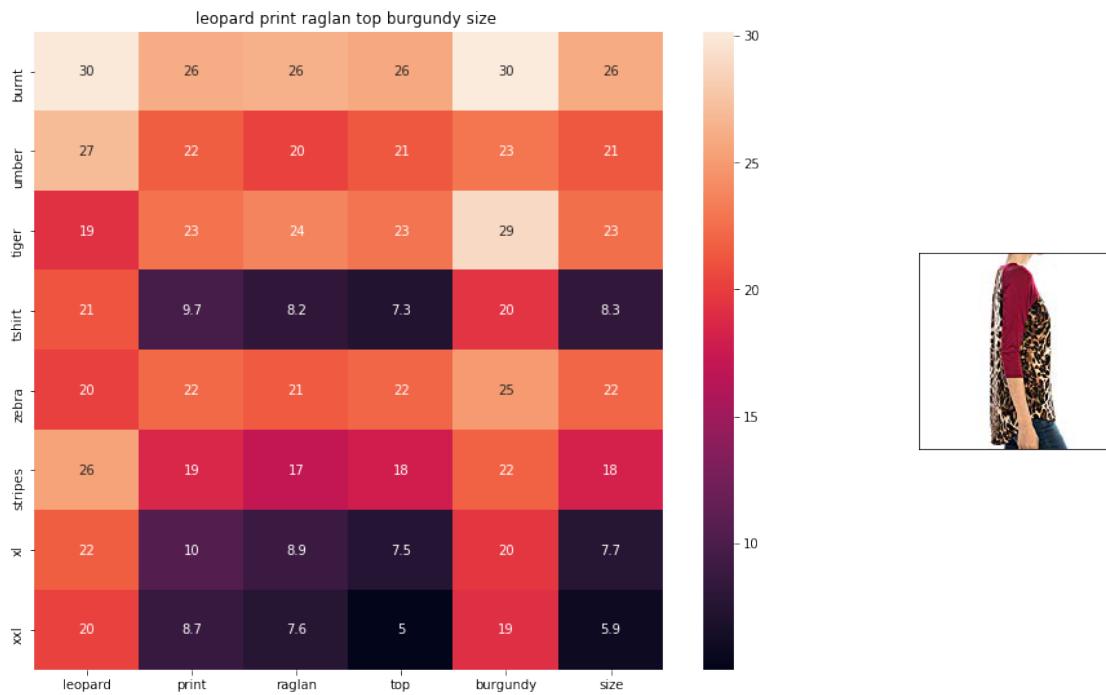


ASIN : B073R4ZM7Y

Brand : Colosseum

euclidean distance from input : 6.450959

=====

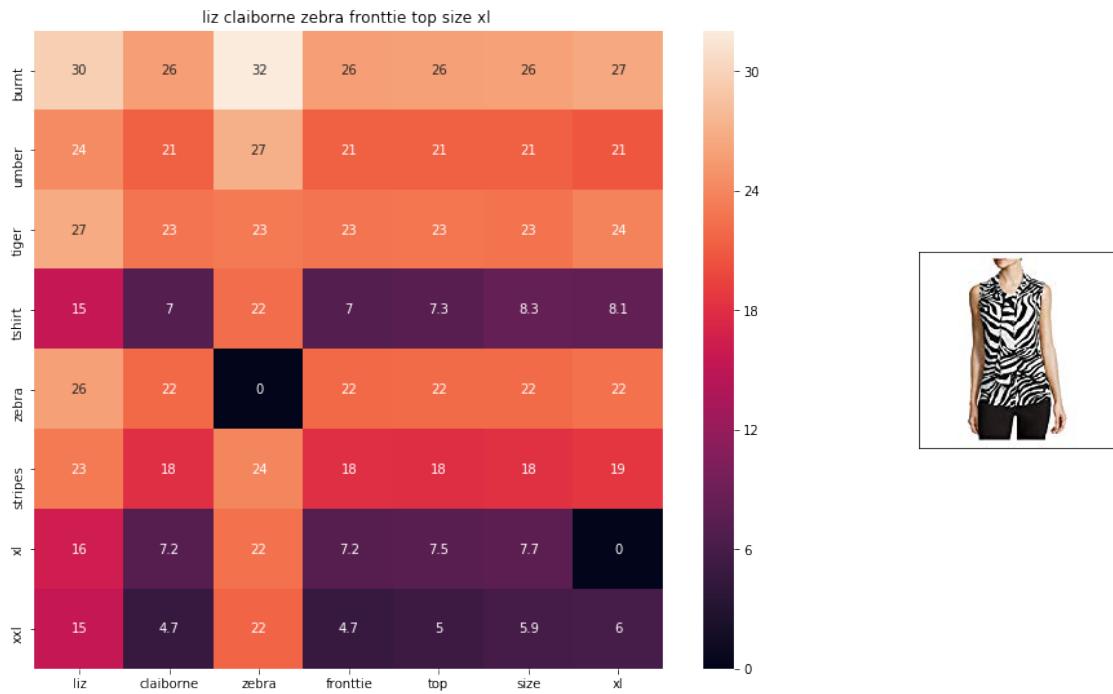


ASIN : B01C60RLDQ

Brand : 1 Mad Fit

euclidean distance from input : 6.463409

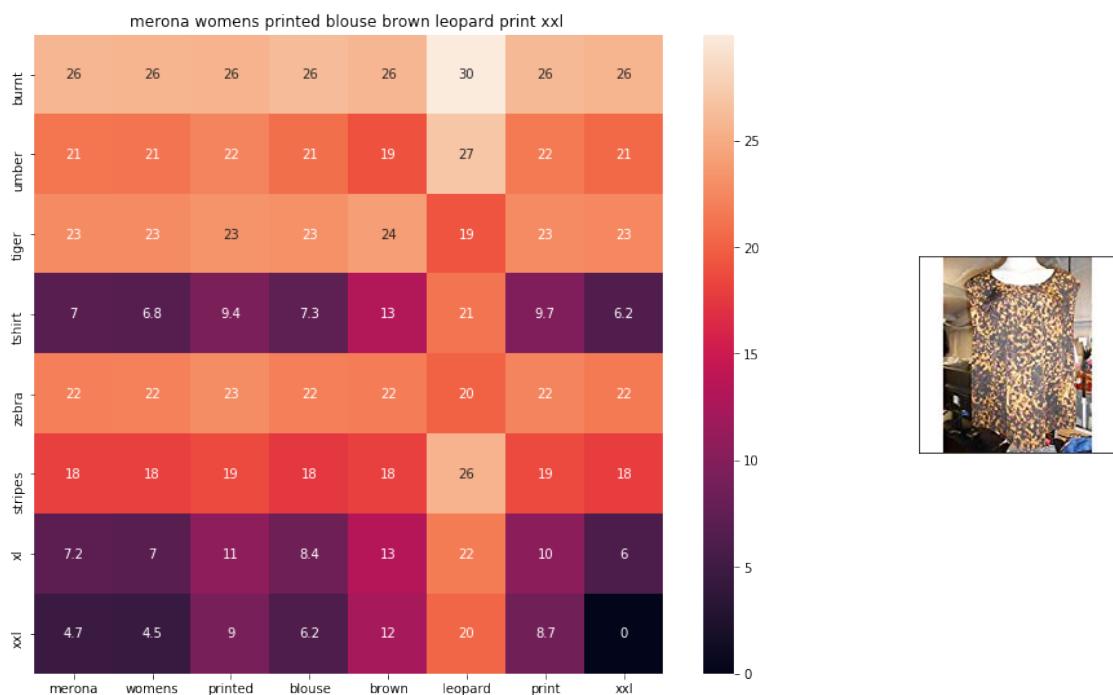
=====



ASIN : B06XBY5QXL

Brand : Liz Claiborne

euclidean distance from input : 6.5392227

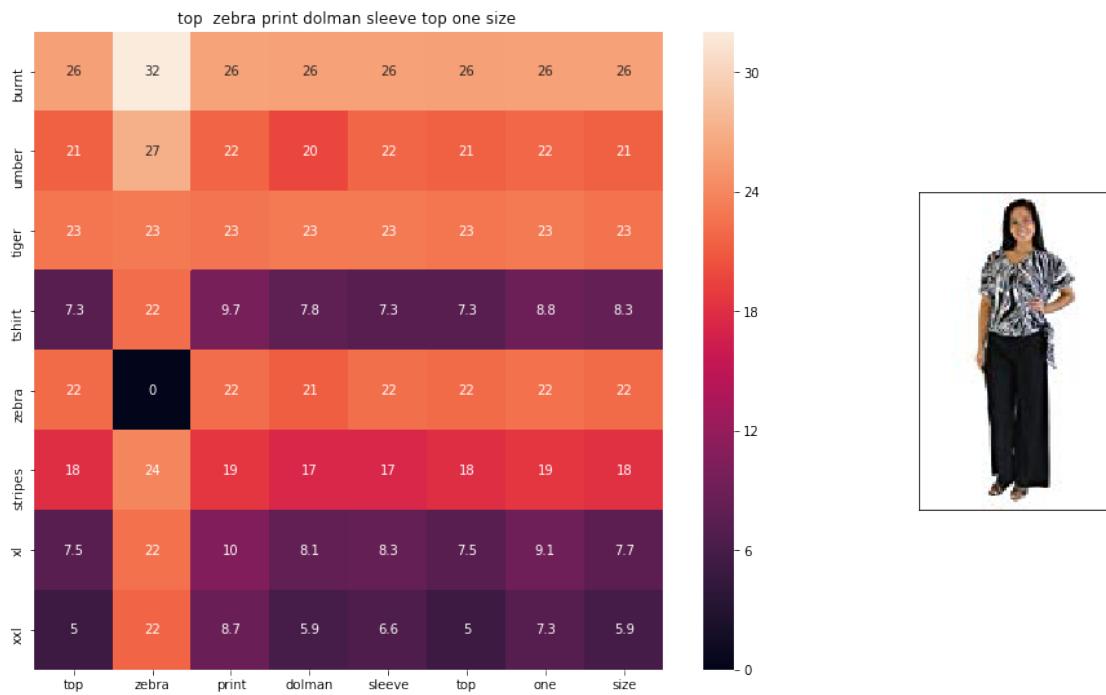


ASIN : B071YF3WDD

Brand : Merona

euclidean distance from input : 6.5755024

=====

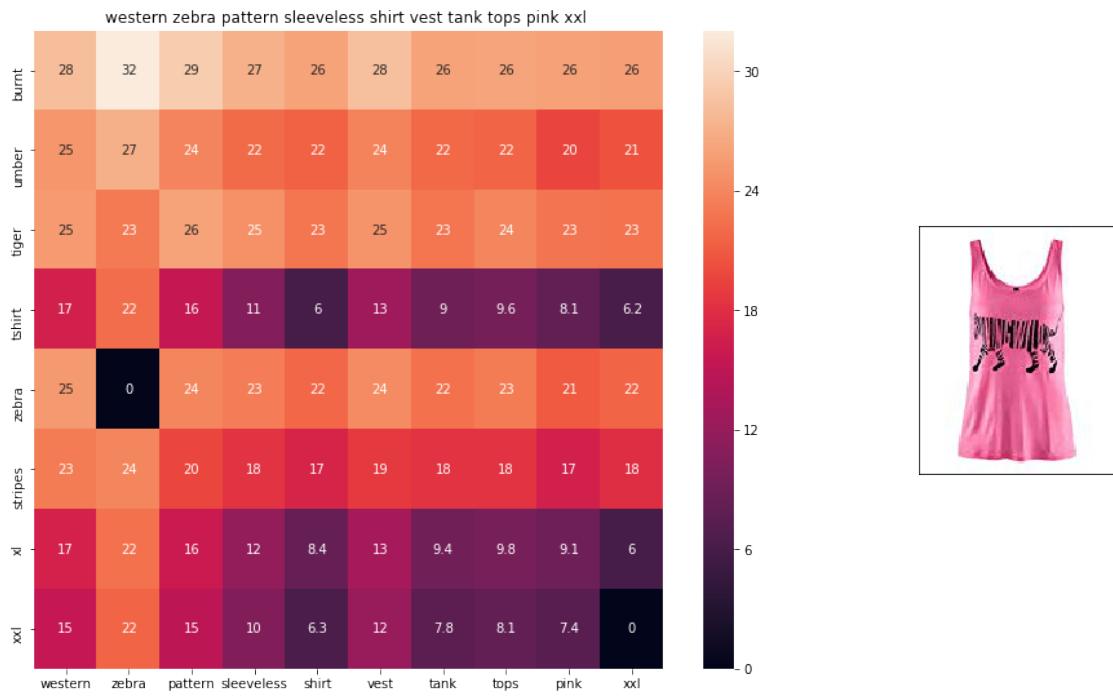


ASIN : B00H8A6ZLI

Brand : Vivian's Fashions

euclidean distance from input : 6.6382146

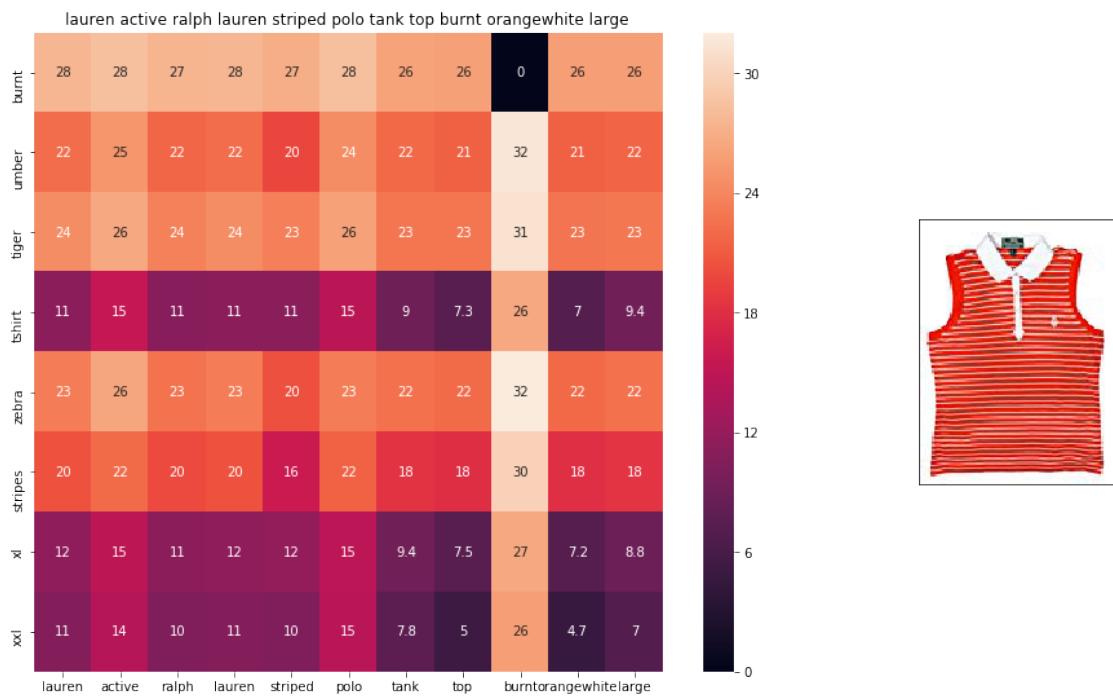
=====



ASIN : B00Z6HEXWI

Brand : Black Temptation

euclidean distance from input : 6.6607366

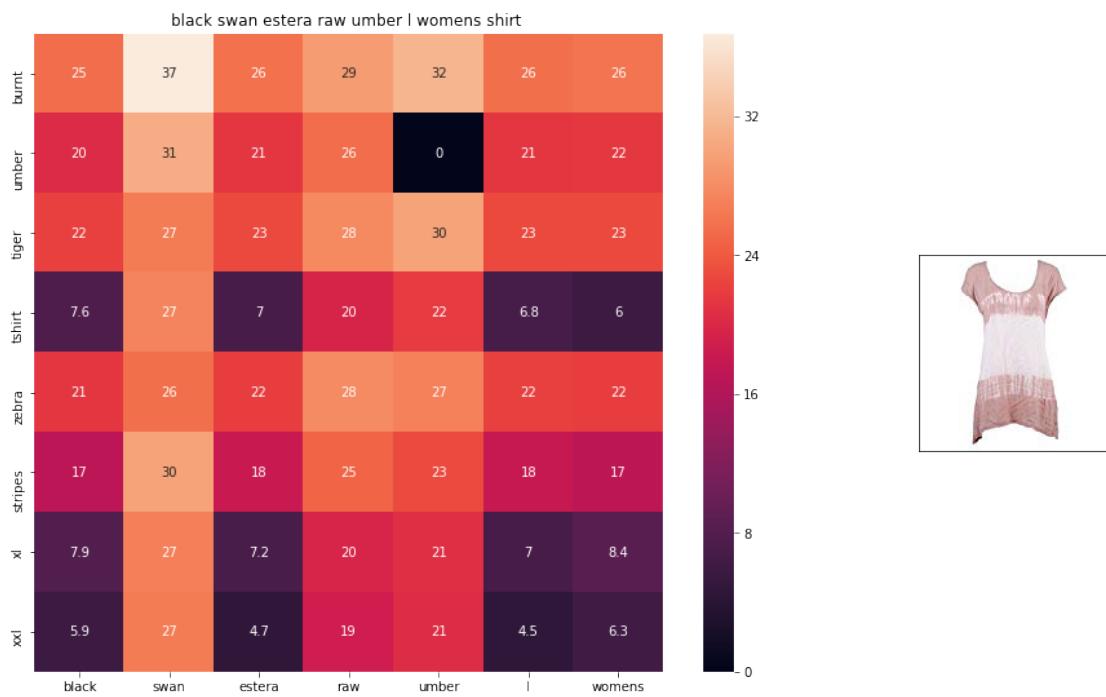


ASIN : B00IILGH50Y

Brand : Ralph Lauren Active

euclidean distance from input : 6.6839046

=====

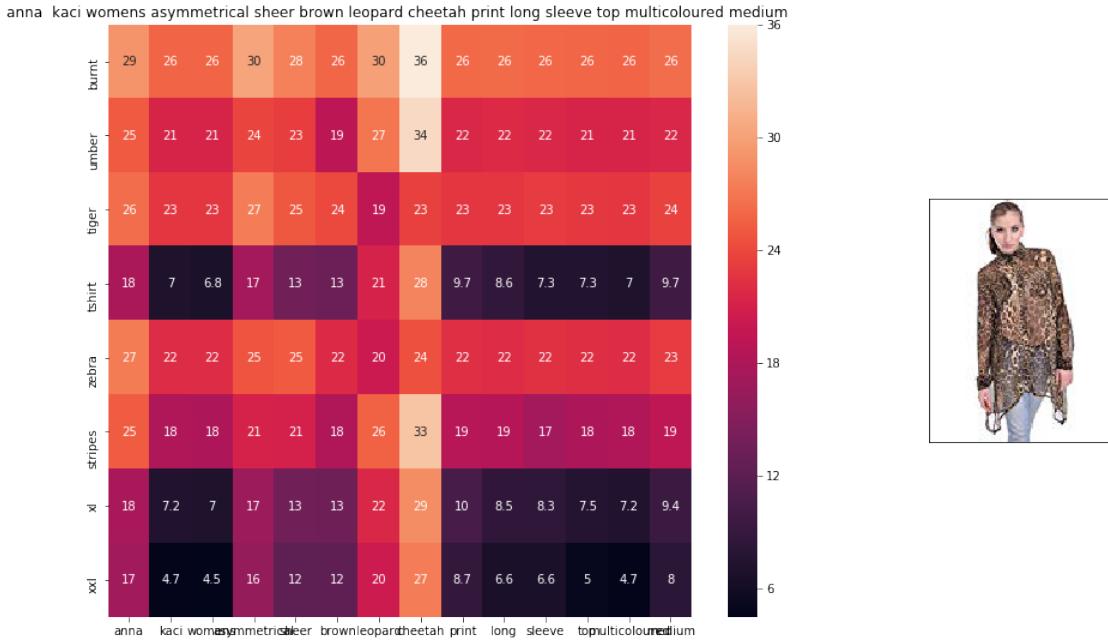


ASIN : B06Y1VN8WQ

Brand : Black Swan

euclidean distance from input : 6.705763

=====



ASIN : BOOKSNTY7Y

Brand : Anna-Kaci

euclidean distance from input : 6.7061243

In [42]: `data['brand'].isnull().any()`

Out[42]: True

5.3 Generating vectors for brand and color

```
In [59]: # 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 hyphen
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()

```

- After we generated vectors using one-hot encoding for "brand", "color" and "product_type" we combined them into one vector using "hstack" into a single vector named "extra_features".

5.4 Generating vectors from Image data using CNN

```

In [ ]: #load the features and corresponding ASINS info.
bottleneck_features_train = np.load('16k_data_cnn_features_2.npy')
asins = np.load('16k_data_cnn_feature_asins.npy')
asins = list(asins)

# load the original 16K dataset
data = pd.read_pickle('pickels/16k_apperal_data_preprocessed')
df_asins = list(data['asin'])

from IPython.display import display, Image, SVG, Math, YouTubeVideo

#get similar products using CNN features (VGG-16)
def get_similar_products_cnn(doc_id, num_results):
    doc_id = asins.index(df_asins[doc_id])
    pairwise_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_train)

    indices = np.argsort(pairwise_dist.flatten())[0:num_results]
    pdists = np.sort(pairwise_dist.flatten())[0:num_results]

    for i in range(len(indices)):
        rows = data[['medium_image_url', 'title']].loc[data['asin']==asins[indices[i]]]
        for indx, row in rows.iterrows():
            display(Image(url=row['medium_image_url'], embed=True))
            print('Product Title: ', row['title'])
            print('Euclidean Distance from input image:', pdists[i])
            print('Amazon Url: www.amazon.com/dp/' + asins[indices[i]])
get_similar_products_cnn(12565, 10)

```

5.5 Computing our solution using the combined vectors

```

In [79]: def heat_map_w2v_brand_color_image(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(weighted/
s1_vec = get_word_vec(sentance1, doc_id1, model)
#s2_vec = np.array(#number_of_words_title2 * 300), each row is a vector(weighted/
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[doc_id1]],
               [data['asin'].loc[df_id2], brands[doc_id2], colors[doc_id2], types[doc_id2]]]

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

# 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])
# plotting 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 lines 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 [84]: def idf_w2v_brand_color_image(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 remaining
    # the metric we used here is cosine, the coside distance is mesured as  $K(X, Y) = \frac{X^T Y}{\|X\| \|Y\|}$ 
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    idf_w2v_dist = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].reshape(1, -1))
    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])
    image_dist = pairwise_distances(bottleneck_features_train, bottleneck_features[doc_id])
    pairwise_dist = (w1 * idf_w2v_dist + w2 * ex_feat_dist + w3 * image_dist) / float(w1 + w2 + w3)

    # 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_color_image(data['title'].loc[df_indices[0]], data['title'].loc[df_indices[i]])
        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_color_image(5956, 50, 40, 10, 20)
    # in the give heat map, each cell contains the euclidean distance between words i, j

```

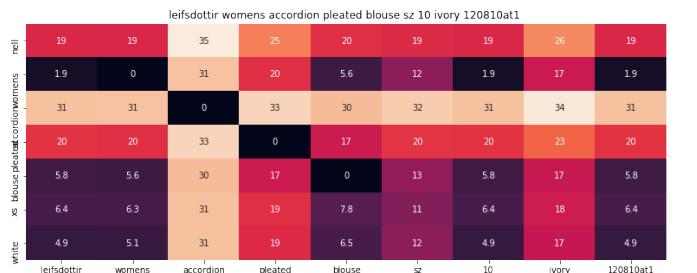


ASIN : B071KFZQ8R

Brand : Nell

euclidean distance from input : 6.468134233728051e-07

=====

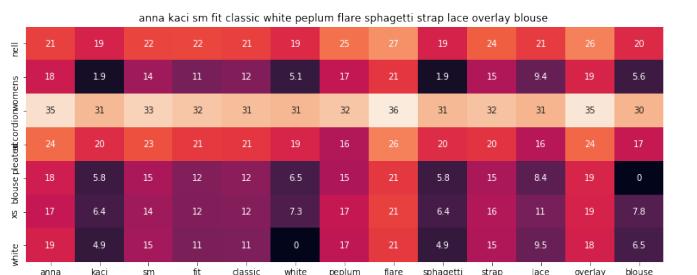


ASIN : B01B6E0DX2

Brand : Leifsdottir

euclidean distance from input : 8.361108093261718

=====

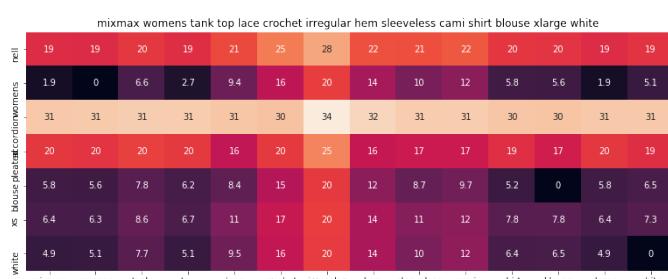


ASIN : B017I32MOI

Brand : Anna-Kaci

euclidean distance from input : 8.407499278105675

=====



ASIN : B01FFA6LNY

Brand : Mixmax

euclidean distance from input : 8.467916259910176

=====



ASIN : B01N9VLKDR

Brand : DKNY

euclidean distance from input : 8.496068420410156

=====



ASIN : B071XQWGSP

Brand : Greylin

euclidean distance from input : 8.542618103027344

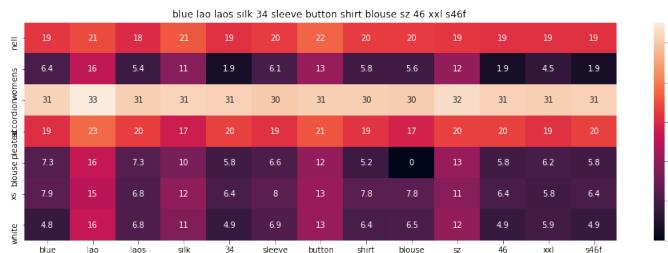
=====



ASIN : B016CW52N4

Brand : HelloPretty

euclidean distance from input : 8.553618774558613



ASIN : B071HBD7LY

Brand : Nanon

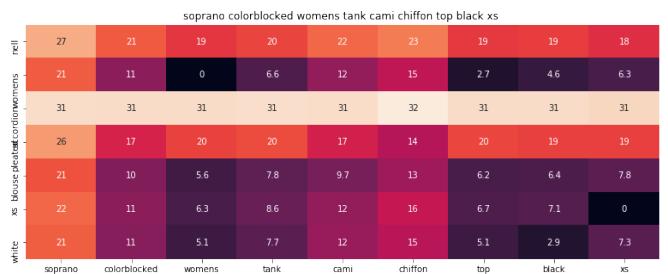
euclidean distance from input : 8.581898803710937



ASIN : B074LVZ7H8

Brand : Gracia

euclidean distance from input : 8.588372497703144



ASIN : B074WF471Z

Brand : Soprano

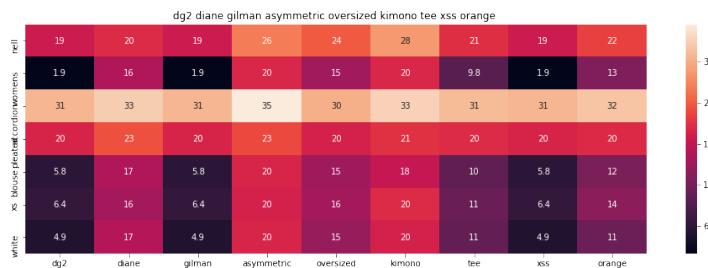
euclidean distance from input : 8.593223571777344



ASIN : B073XF6KSD

Brand : Mukdahan Craft

euclidean distance from input : 8.644105535726478



ASIN : B01M9EETB3

Brand : Diane Gilman

euclidean distance from input : 8.656272283773353

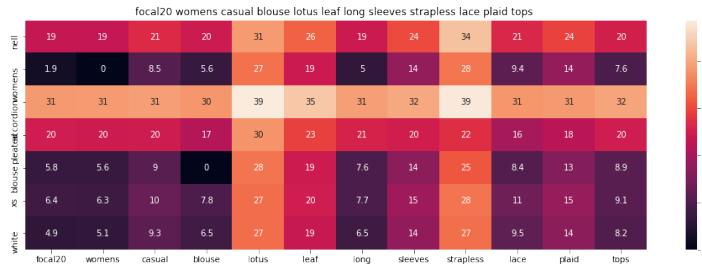


ASIN : B01BZH5ZDI

Brand : Sofra

euclidean distance from input : 8.665142212058614

=====



ASIN : B06XX46TJ3

Brand : Focal20

euclidean distance from input : 8.668409118652344

=====

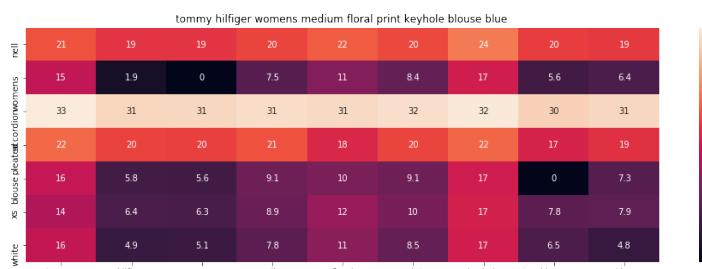


ASIN : B074KFCFQ7

Brand : ClothingLoves

euclidean distance from input : 8.675027771140645

=====



ASIN : B0713TH39L

Brand : Tommy Hilfiger

euclidean distance from input : 8.68239716170304

=====

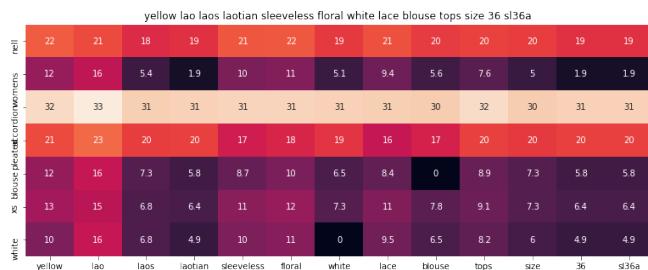


ASIN : B0753N2FWQ

Brand : susana monaco

euclidean distance from input : 8.691125494222572

=====



ASIN : B074J781N9

Brand : Nanon

euclidean distance from input : 8.72220672607422

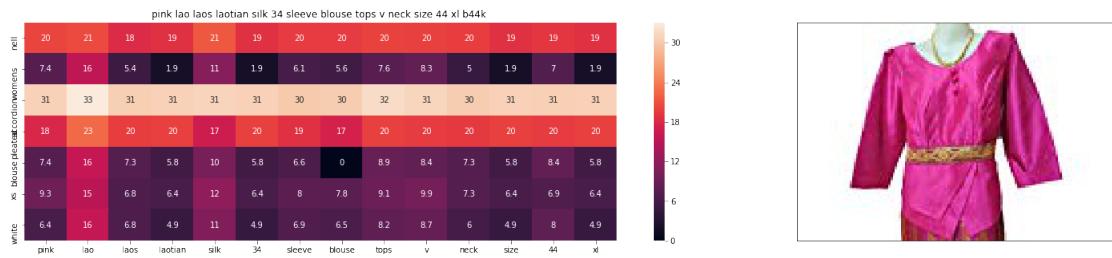
=====



ASIN : B071RPQKSS

Brand : David Lerner

euclidean distance from input : 8.724149363554893



ASIN : B072DXG86D

Brand : Nanon

euclidean distance from input : 8.729853515625

5.6 Conclusion

1. So we find the similar products using our combined vector which include 'title_vector', 'brand_vector', 'color_vector' and 'image_vector'.
 2. For generating title_vector we used idf-weighted w2v.
 3. For generating brand_vector and color_vector we used one-hot encoding.
 4. For generating image_vector we used CNN(VGG-16).
 5. We used weighted euclidean distances to compute the results which means we can assign a weight value corresponding to each vector to get desired results.
 6. Which means if we give more weight to brand then in our results, products with similar brand will be shown more often.
- And for finding the best solution to our business problem we could experiment by giving different weight values to the all the vectors according our business constraints & then use A/B testing.



best_sol