## Amazon Fashion Discovery Engine (Content-Based recommendation)

The main objective of this case study is to recommend similar items to similar users.

There are generally 2 types of recommendation algorithms used in E-commerce. They are

a) Content-Based Recommendation
b) Collaborative filtering recommendation

In content-based recommendations, the products are recommended to the users, based on the product names, brand names, images, colors, etc, whereas in collaborative filtering recommendations, the items are recommended to the users based on their preferences with similar preferences of the other users.

In this case study, we are going to only use the Content-Based Recommendation. The task here is if a product is given by a user, then we have to recommend many other products that belong to the same category, by taking the text and the images of the query product into consideration.

## Overview of the data and Terminology

The data is given in a JSON format. We need to load the 'tops_fashion.json' file into a pandas data frame.

Number of Data points = 183138

Number of Features = 19

```
# 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')
```

```
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
```

**Dataset**

The given data is in the form of a matrix. The entire combination of the rows and columns in the given data matrix is considered as the dataset.

**Row**

Here each data point in the given dataset is called a Row.

**Column**

Each feature in the given dataset is called a Column.

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)
3) Color (Color information of the apparel. It also can contain many colors as a single value. Ex: Red and Black Stripes)
4) Product Type Name (Type/Category the product belongs to, Ex: Shirt/T-Shirt/Trouser, etc)
5) Medium Image URL (Represents the URL of the image)
6) Title (Product Title)
7) Formatted Price (Price of the product)

## Data Cleaning and Understanding

It is an important stage in Machine Learning. It is often overlooked and also might take a few hours to a few days. The more we understand the data, the better we can build the models. We can compute the basic statistics for each feature in the dataset like the number of different categories, the category with the maximum number of points(ie., the top most category), etc. Let us look at the examples

**Basic stats for the feature: product_type_name**

In [38]:
```
# 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
```

```
# 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' 'UNDERWEA
R'
 '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' 'SHIRT
S'
 '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']
```

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

After computing the basic statistics of the ‘product_category’, we can compute the basic statistics of the ‘Brand’ column.

**Basic stats for the feature: brand**

```
# 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
```

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

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

We can clearly see that there are 151 data points with the values missing in the ‘Brand’ column. There are 10577 brands and the topmost brand is ‘Zago’ with 223 pieces.

**Basic stats for the feature: color**

```
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
```

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

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

We shall now also explore the ‘color’ column now. We came to know that there are 7380 different colors available in the inventory, and the most common color is Black.

Now we shall explore the ‘formatted_price’ column and there are 3135 unique price values, and the highest value is 19.99$.

**Basic stats for the feature: formatted_price**

```
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
```

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

```
[(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**

```
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
```

Now we shall remove all those points that have the null values in the ‘formatted_price’ and the ‘color’ columns. After removal, we are left with 28385 points.

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

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

```
# 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
```

```
# consider products which have color information
# data['color'].isnull() => gives the information about the dataframe row's which have null values price == None|Null
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
```

Due to the time and memory constraints, we are using only the leftover 28385 points. We are storing these 28385 points into a pickle file, and let's name it as 'pickels/28k_apparel_data'.

## Removing the Duplicate Items

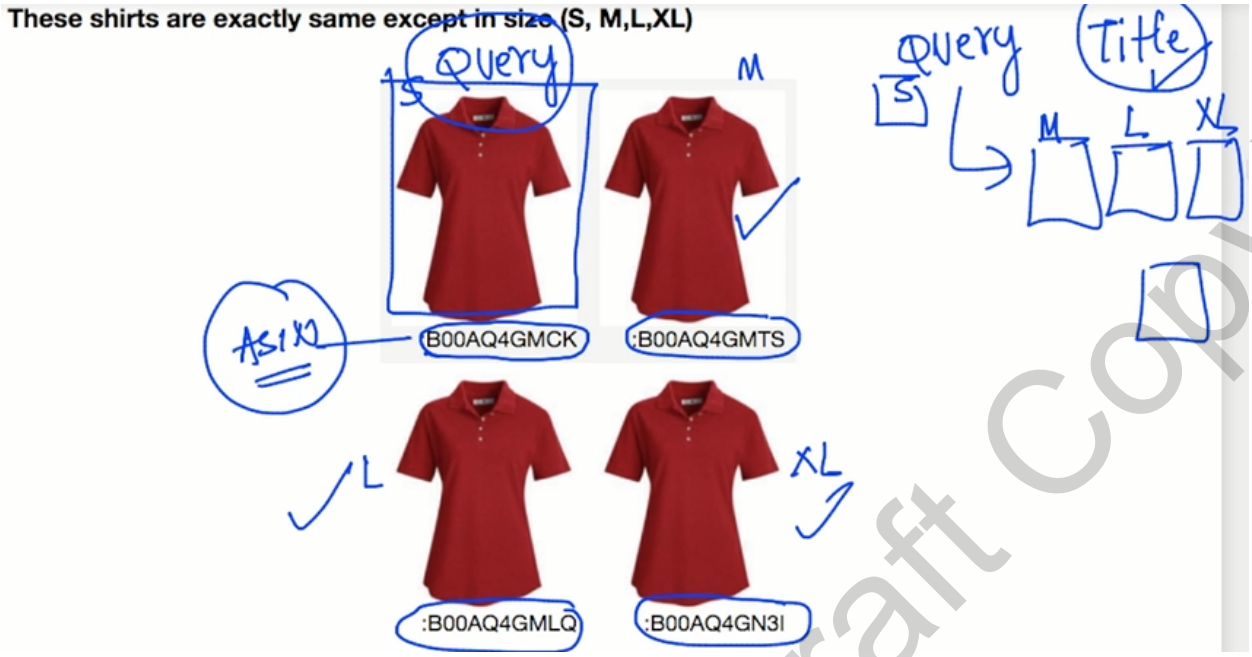
**[5.2.1] Understand about duplicates.**

```
# 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
```

We are loading the 28K data points that were saved in a pickle file, into a pandas data frame, and then counting the number of rows/data points with duplicate values in the 'title' column. We got 2325 data points with duplicate values present in the 'title column.

One of the best examples of duplicate records is the points representing the same item but of different sizes.




The other example of duplicate items is the data points representing the same item of the same size, but of different colors, as shown below.

Here we are removing the rows with less than or equal to 4 words in the text column.

```
# 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
```

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

After the removal of such points, we are left with 27949 points. The reason for choosing a length of '4' comes from the experience. After removing the points, we are sorting the points.

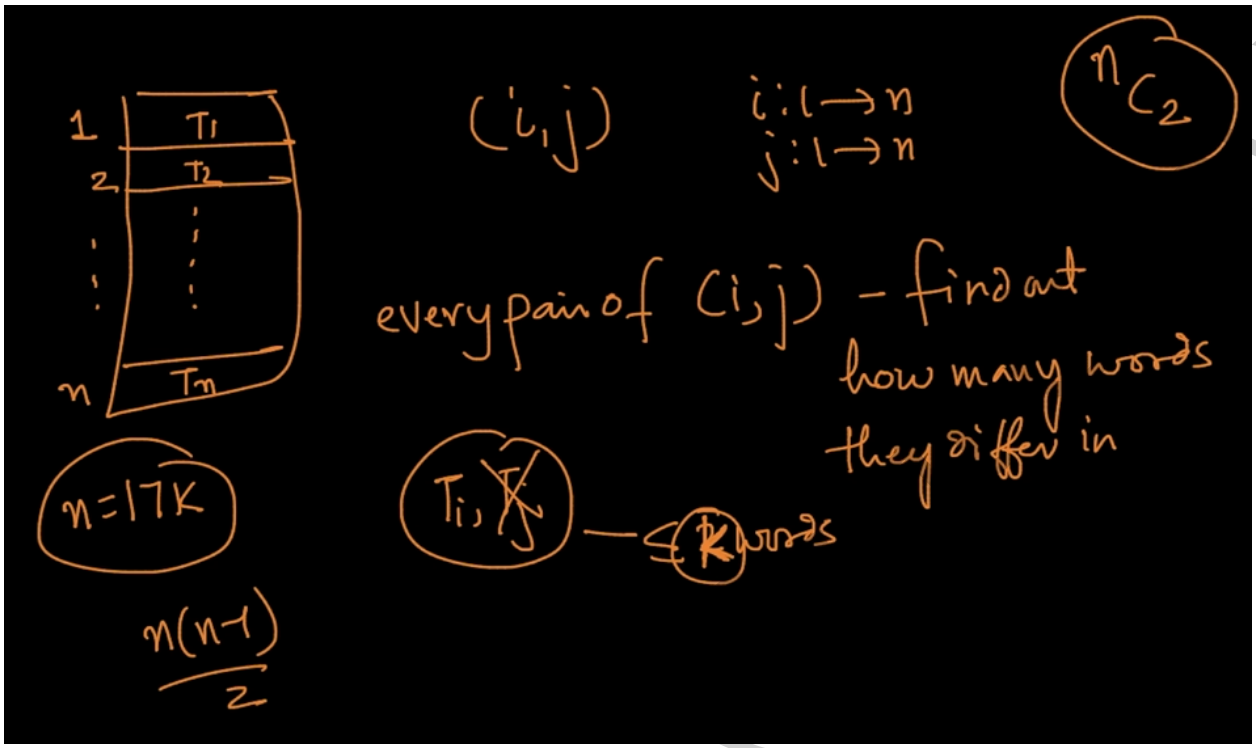
We also do come across a situation where the data points differ by one or two words at the last in the text. Such data points are also considered as duplicates, and we have to remove them. You can see a few examples of such type.

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

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

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

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

All those data points that differ in the last words are also considered duplicates. After removing them we are left with 17593 points. But it is not mandatory for the different words to be only at the end. There are also chances for the different words to occur in the middle. For that, we need to take $17593C_2$ pair combinations and compare the pairs. If any pair is having a number of different words less than 'K' (say), then those two data

points are also considered duplicates, and after removing them, we are left with 16K points.




Below is the code to perform this process.

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

    previous_i = i

    # store the list of words of ith string in a, ex: a = ['tokidoki', 'The', 'Queen', 'of', 'Diamonds', 'Women's', 'Shirt', 'X-
    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', 'Queen', 'of', 'Diamonds', 'Women's', 'Shirt',
        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, it will appened None in case of unequal
        # example: a =['a', 'b', 'c', 'd']
        # b = ['a', 'b', 'd']
        # itertools.zip_longest(a,b): will give [('a','a'), ('b','b'), ('c','d'), ('d', None)]
        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 considering it as those two apperals are differen
        # if the number of words in which both strings differ are < 2 , we are considering it as those two apperals are same, he
        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 string index
            stage1_dedupe_asins.append(data_sorted['asin'].loc[indices[i]])


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

So far the number of points were reduced from 183K(in the beginning) to 28K (in the middle), and now we are left with 17K points.

**Text Preprocessing (Tokenization and Stopword Removal)**

So far we have worked on the removal of the duplicate points. Now it's time for us to remove the stop words. The main reason for the stop word removal is that they do not add any value in the model building or decision making but are just used for sentence completion purposes. So first of all we have to tokenize the text values into a list of substrings, and then remove the stop words from them.

```
Examples:

Titles-1
86261.  UltraClub Women's Classic Wrinkle-Free Long Sleeve Oxford S
hirt, Pink, XX-Large
115042. UltraClub Ladies Classic Wrinkle-Free Long-Sleeve Oxford Li
ght Blue XXL

TItles-2
75004.  EVALY Women's Cool University Of UTAH 3/4 Sleeve Raglan Tee
109225. EVALY Women's Unique University Of UTAH 3/4 Sleeve Raglan T
ees
120832. EVALY Women's New University Of UTAH 3/4-Sleeve Raglan Tshi
rt
```

So the below code demonstrates the removal of non alpha-numeric characters, conversion to lowercase, and removal of the stopwords from the given text.

```
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()
```

```
# 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: {'such', 'and', 'hers', 'up', 'she', 'd', 'further', 'all', 'than', 'under', 'is', 'off', 'both', 'most',
'few', 'should', 're', 'very', 'just', 'then', 'didn', 'myself', 'in', 'too', 's', 'shouldn', 'herself', 'because', 'how', 'its
elf', 'what', 'shan', 'weren', 'doing', 'them', 'couldn', 'their', 'so', 'ain', 'haven', 'yourself', 'now', 'll', 'isn', 'abou
t', 'over', 'into', 'before', 'during', 'on', 'as', 'aren', 'against', 'above', 'down', 'they', 'below', 'me', 'again', 'for',
'why', 'been', 'yourselves', 'more', 'her', 'that', 'can', 'am', 'was', 'themselves', 'mightn', 'does', 'those', 'only', 'has
n', 'any', 'ma', 'are', 'nor', 'out', 'you', 'ourselves', 'the', 'an', 'has', 'where', 'i', 'while', 'ours', 'its', 'your', 'ha
d', 'were', 'being', 'no', 'or', 'needn', 've', 'y', 'a', 'each', 'have', 'through', 'when', 'mustn', 'by', 'won', 'from', 'ow
n', 'will', 'there', 't', 'him', 'these', 'doesn', 'theirs', 'my', 'did', 'of', 'who', 'until', 'wouldn', 'we', 'do', 'having',
'yours', 'other', 'wasn', 'it', 'with', 'once', 'here', 'don', 'o', 'whom', 'this', 'if', 'but', 'hadn', 'our', 'some', 'm', 'n
ot', 'between', 'himself', 'same', 'at', 'be', 'he', 'after', 'which', 'to', 'his'}
```

```
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")
```

```
3.5727220000000006 seconds
```

```
data.head()
```

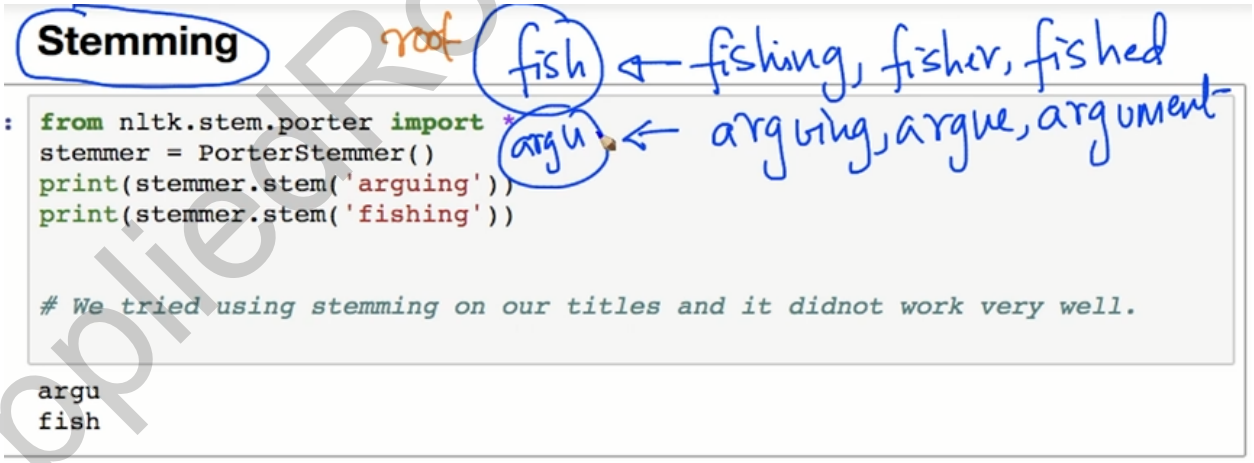
| | asin | brand | color | medium_image_url | product_type_name | title | formatted_price |
|---|---|---|---|---|---|---|---|
| 4 | B004GSI2OS | FeatherLite | Onyx Black/ Stone | https://images-na.ssl-images-amazon.com/images... | SHIRT | featherlite ladies long sleeve stain resistant... | $26.26 |
| 6 | B012YX2ZPI | HX-Kingdom Fashion T-shirts | White | https://images-na.ssl-images-amazon.com/images... | SHIRT | womens unique 100 cotton special olympics wor... | $9.99 |
| 15 | B003BSRPB0 | FeatherLite | White | https://images-na.ssl-images-amazon.com/images... | SHIRT | featherlite ladies moisture free mesh sport sh... | $20.54 |
| 27 | B014ICEJ1Q | FNC7C | Purple | https://images-na.ssl-images-amazon.com/images... | SHIRT | supernatural chibis sam dean castiel neck tshi... | $7.39 |
| 46 | B01NACPBG2 | Fifth Degree | Black | https://images-na.ssl-images-amazon.com/images... | SHIRT | fifth degree womens gold foil graphic tees jun... | $6.95 |

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

## Stemming

We apply stemming in order to remove the tense(like present, past, and future) forms, and reduce such words to a common root word so that all these words will be considered as the same feature. If not, the different forms of the same word in the text, are considered as different dimensions/features, during the vectorization.

But here, after working on it practically, we have observed that, in this context, applying stemming on the text data doesn't yield better results. So we are ignoring the stemming operation for the time being.



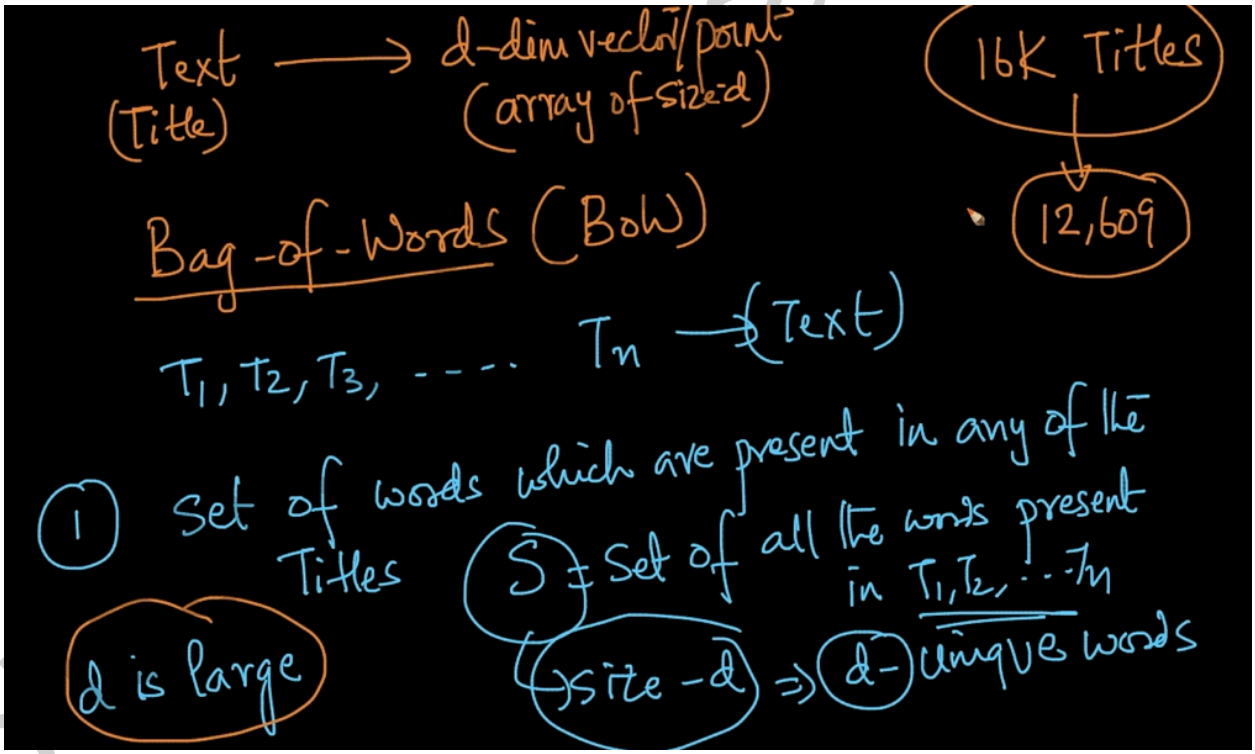
**Text-Based Product Similarity: Converting text to n-D vector: BOW**

In this case study, we shall give top priority to the text of the products to build a recommendation engine. We have to convert the text column values of the points into vector form using a vectorization technique like Bag of Words, and then compute the similarity on the basis of the euclidean distance between them.

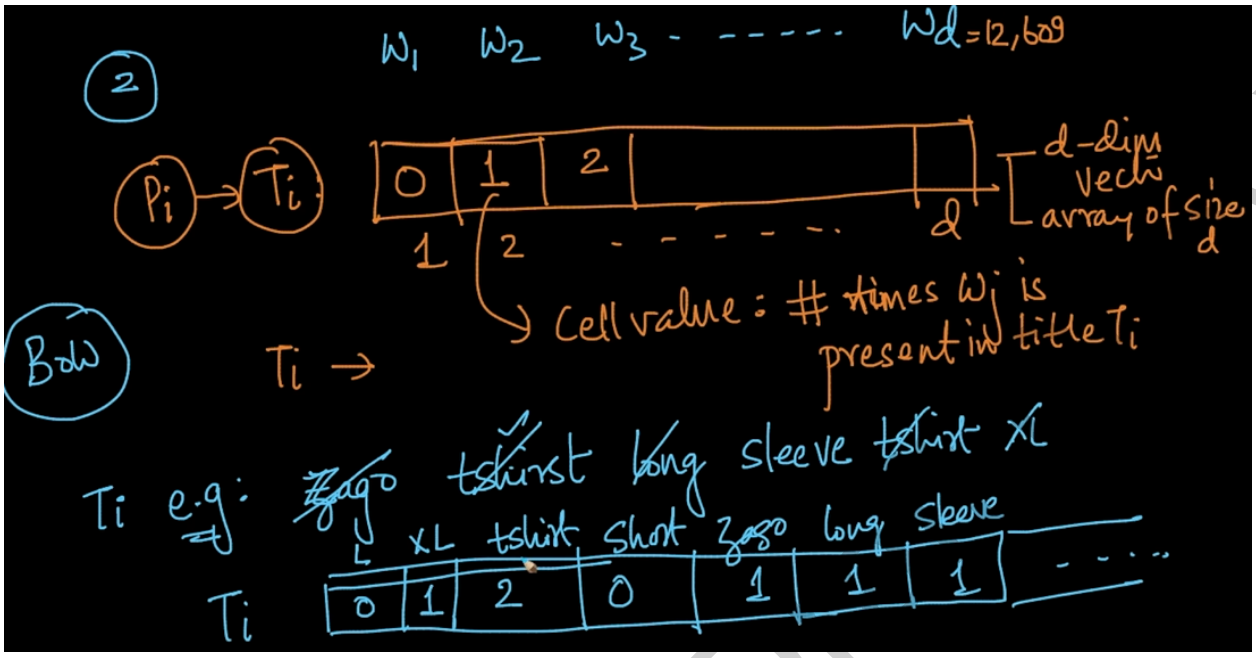
**Bag of Words (BOW)**

Let us assume the text column values of all the points in the given dataset are $T_1$, $T_2$, $T_3$, …, $T_n$.

a) We have to create a set of words present across all the 'text' values $T_1$, $T_2$, $T_3$, …, $T_n$. Let the size of this set be denoted as 'd', which means there are 'd' unique number of words across the 'text' column.




b) We have to build the vectors for each of these 'text' column values with these 'd' words as the dimensions/features, and fill the corresponding cells of each feature, with the number of times that feature occurs in the 'text' value.



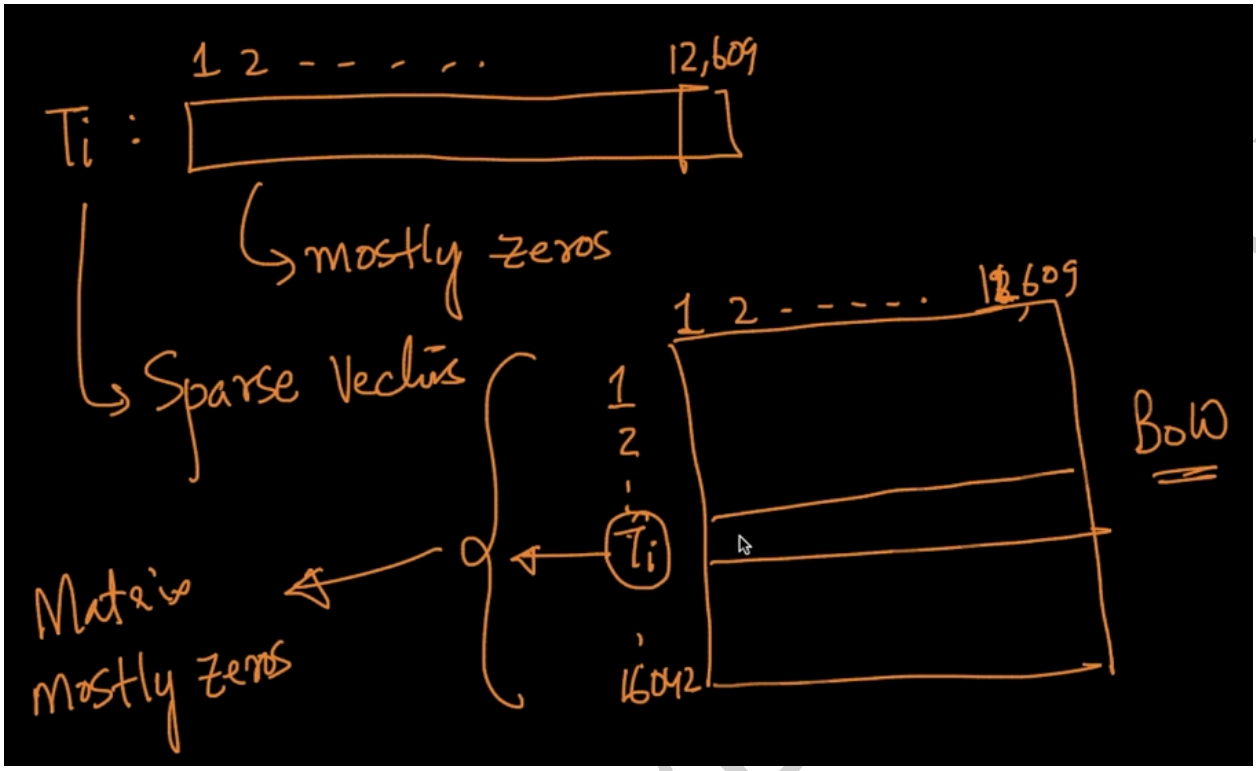

Though Bag of Words is a powerful approach to vectorize the text data, it discards the sequence information. The code to apply BOW vectorization is given below

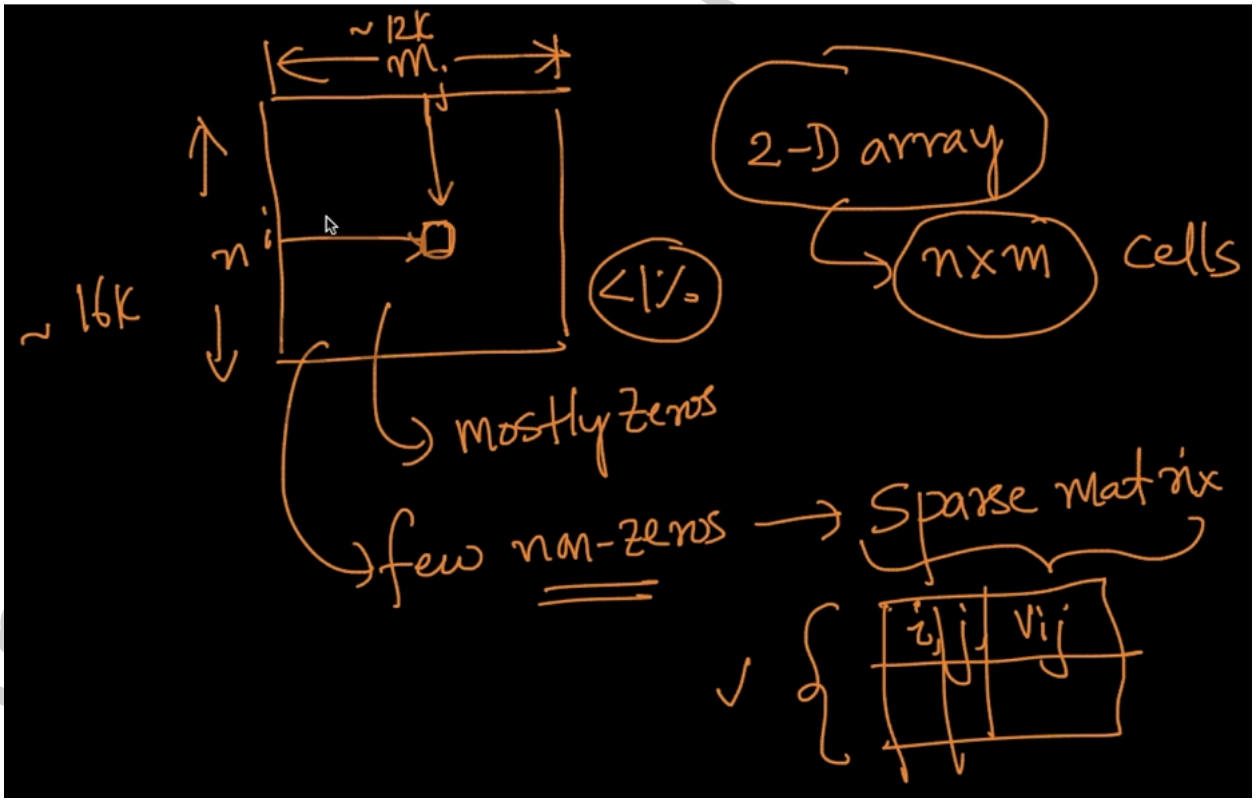
**[8.2] Bag of Words (BoW) on product titles.**

```
In [17]: 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 matr
         # 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 wor

Out[17]: (16042, 12609)
```



The result of BOW vectorization is a sparse matrix. A sparse matrix is an efficient way to store the values when the majority of the values are 0. If we store all the elements (including the zeros), it occupies a huge memory space. So in this case, if the majority of the values are 0, then we can use sparse matrix representation, to save the memory space.


1 2 - - - - - 12,609
Ti :
mostly zeros
Sparse Vectors
1 2 - - - - - 12,609
1
2
Ti
16042
BoW
Matrix
mostly zeros

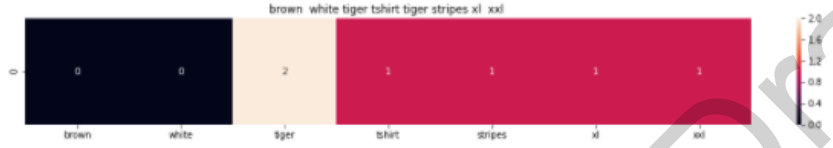

~ 12K
m
n
i
j
~ 16K
<1%
2-D array
nxm
cells
mostly zeros
few non-zeros
Sparse matrix
i j Vij

After creating the vector form for the ‘text’ value of each data point, we have to compute the pairwise Euclidean distances and recommend the top 20 products/items(the number of products to be recommended depends on the problem you are solving. Here in this problem, we took it as 20) that are nearer to a query point from the euclidean distance point. Below is the code for this task.

```
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 apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, Y> / (||X||*||Y||)
    # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
    pairwise_dist = pairwise_distances(title_features,title_features[doc_id])

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

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

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

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


ASIN : B00JXQB5FQ
Brand: Si Row
Title: burnt umber tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 0.0
==========


ASIN : B00JXQASS6
Brand: Si Row
Title: pink tiger tshirt zebra stripes xl xxl
Euclidean similarity with the query image : 1.73205080757
==========

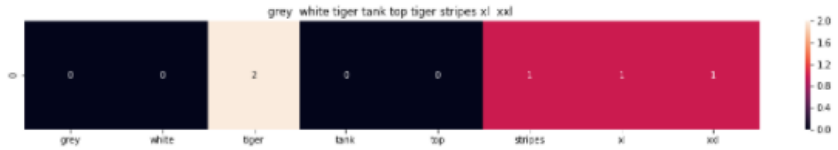



ASIN : B00JXQCWTO
Brand: Si Row
Title: brown white tiger tshirt tiger stripes xl xxl
Euclidean similarity with the query image : 2.44948974278
==========

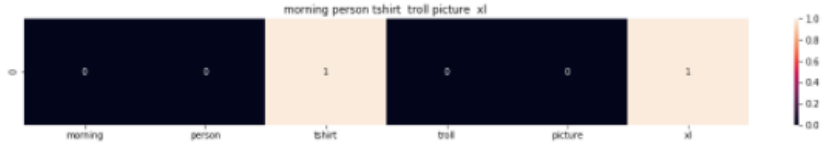

ASIN : B00JXQCUIC
Brand: Si Row
Title: yellow tiger tshirt tiger stripes l
Euclidean similarity with the query image : 2.64575131106
==========


ASIN : B07568NZX4
Brand: Rustic Grace
Title: believed could tshirt
Euclidean similarity with the query image : 3.0
==========

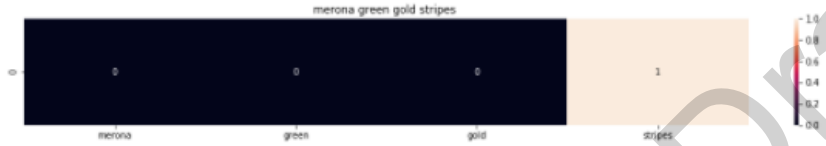

ASIN : B01NB0NKRO
Brand: Ideology
Title: ideology graphic tshirt xl white
Euclidean similarity with the query image : 3.0
==========



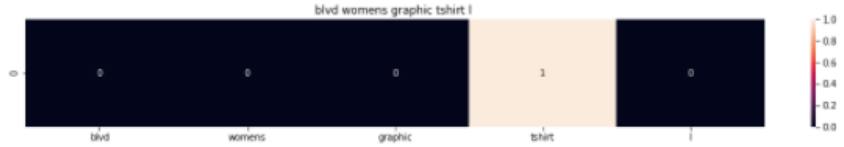

ASIN : B00JXQAFZ2
Brand: Si Row
Title: grey  white tiger tank top tiger stripes xl  xxl
Euclidean similarity with the query image : 3.0
==========



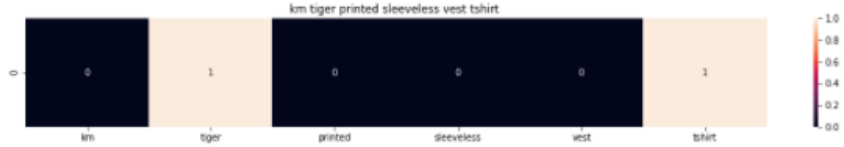

ASIN : B01CLS8LMW
Brand: Awake
Title: morning person tshirt  troll picture  xl
Euclidean similarity with the query image : 3.16227766017
==========



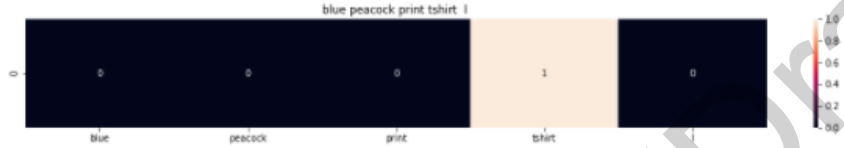

ASIN : B01KVZUB6G
Brand: Merona
Title: merona green gold stripes
Euclidean similarity with the query image : 3.16227766017
==========




ASIN : B0733R2CJK
Brand: BLVD
Title: blvd womens graphic tshirt l
Euclidean similarity with the query image : 3.16227766017
==========



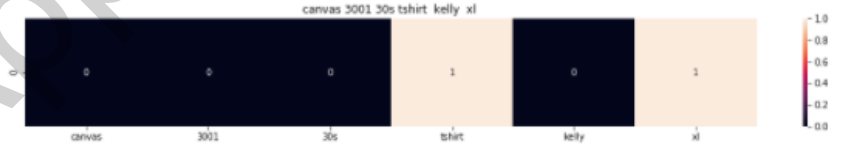

ASIN : B012VQLT6Y
Brand: KM T-shirt
Title: km tiger printed sleeveless vest tshirt
Euclidean similarity with the query image : 3.16227766017
==========



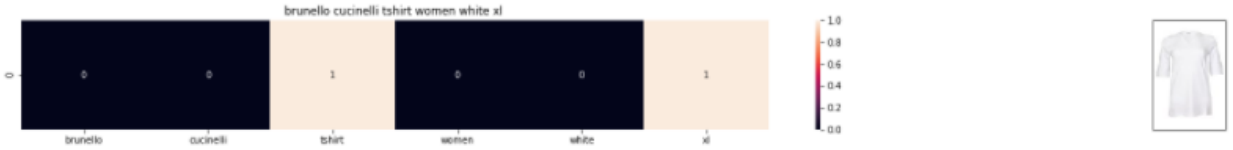

ASIN : B00JXQC8L6
Brand: Si Row
Title: blue peacock print tshirt l
Euclidean similarity with the query image : 3.16227766017
==========


ASIN : B06XC3CZF6
Brand: Fjallraven
Title: fjallraven womens ovik tshirt plum xxl
Euclidean similarity with the query image : 3.16227766017
==========

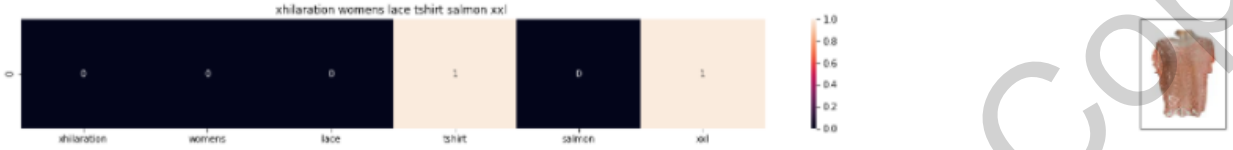

ASIN : B005IT8OBA
Brand: Hetalia
Title: hetalia us girl tshirt
Euclidean similarity with the query image : 3.16227766017
==========



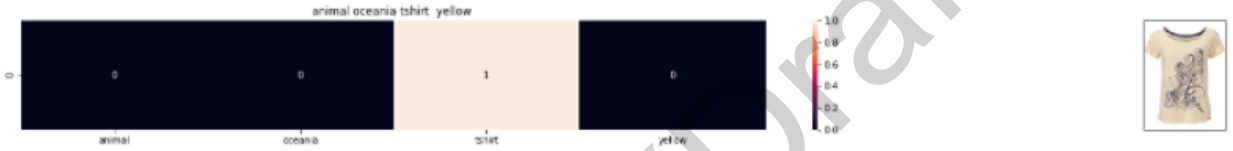

ASIN : B0088PN0LA
Brand: Red House
Title: canvas 3001 30s tshirt kelly xl
Euclidean similarity with the query image : 3.16227766017
==========




ASIN : B06X99V6WC
Brand: Brunello Cucinelli
Title: brunello cucinelli tshirt women white xl
Euclidean similarity with the query image : 3.16227766017
============================================================

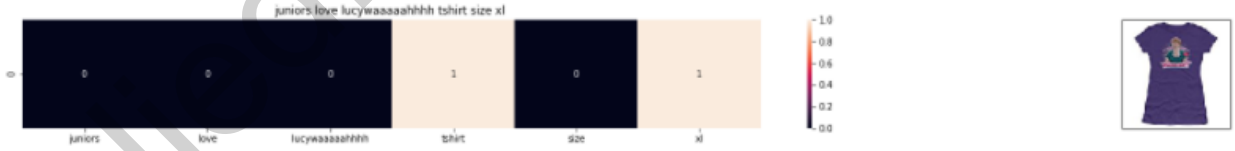



ASIN : B06Y1JPW1Q
Brand: Xhilaration
Title: xhilaration womens lace tshirt salmon xxl
Euclidean similarity with the query image : 3.16227766017
============================================================




ASIN : B06X6GX6WG
Brand: Animal
Title: animal oceania tshirt  yellow
Euclidean similarity with the query image : 3.16227766017
============================================================


ASIN : B017X8PW9U
Brand: Diesel
Title: diesel tserraf tshirt black
Euclidean similarity with the query image : 3.16227766017
============================================================

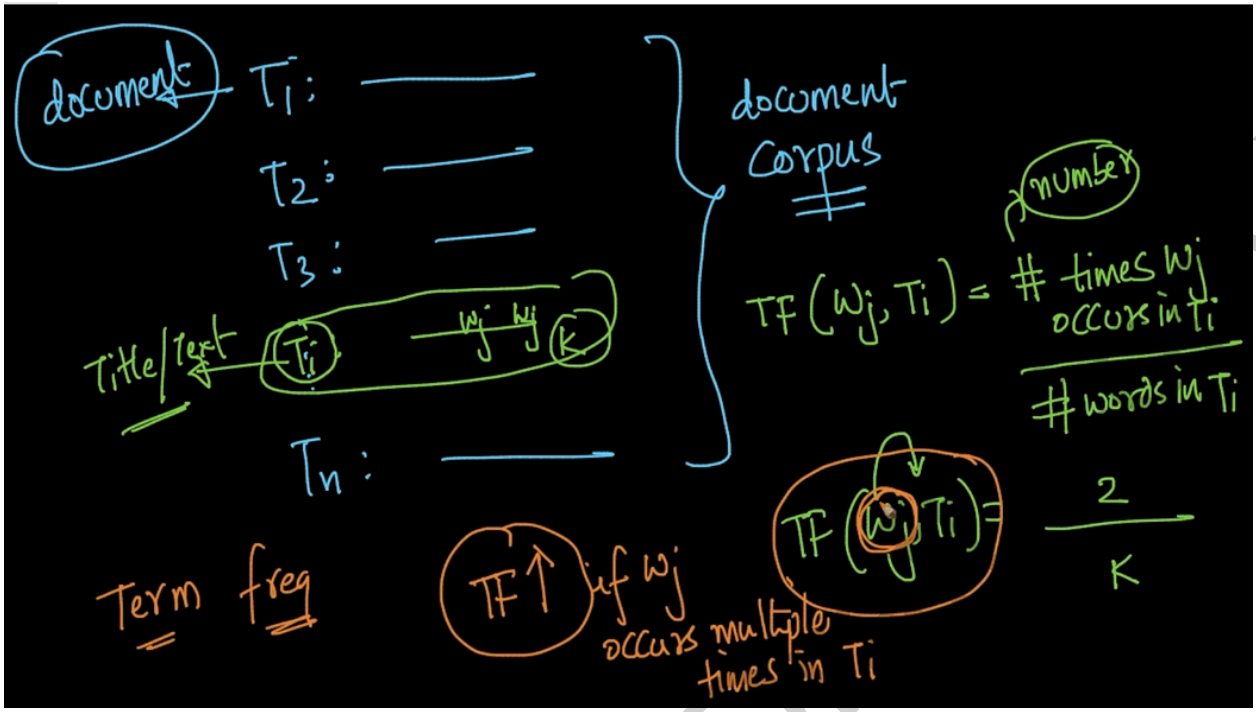



ASIN : B00IAA4JIQ
Brand: I Love Lucy
Title: juniors love lucywaaaaahhhh tshirt size xl
Euclidean similarity with the query image : 3.16227766017
============================================================



## TF-IDF based Text Vectorization

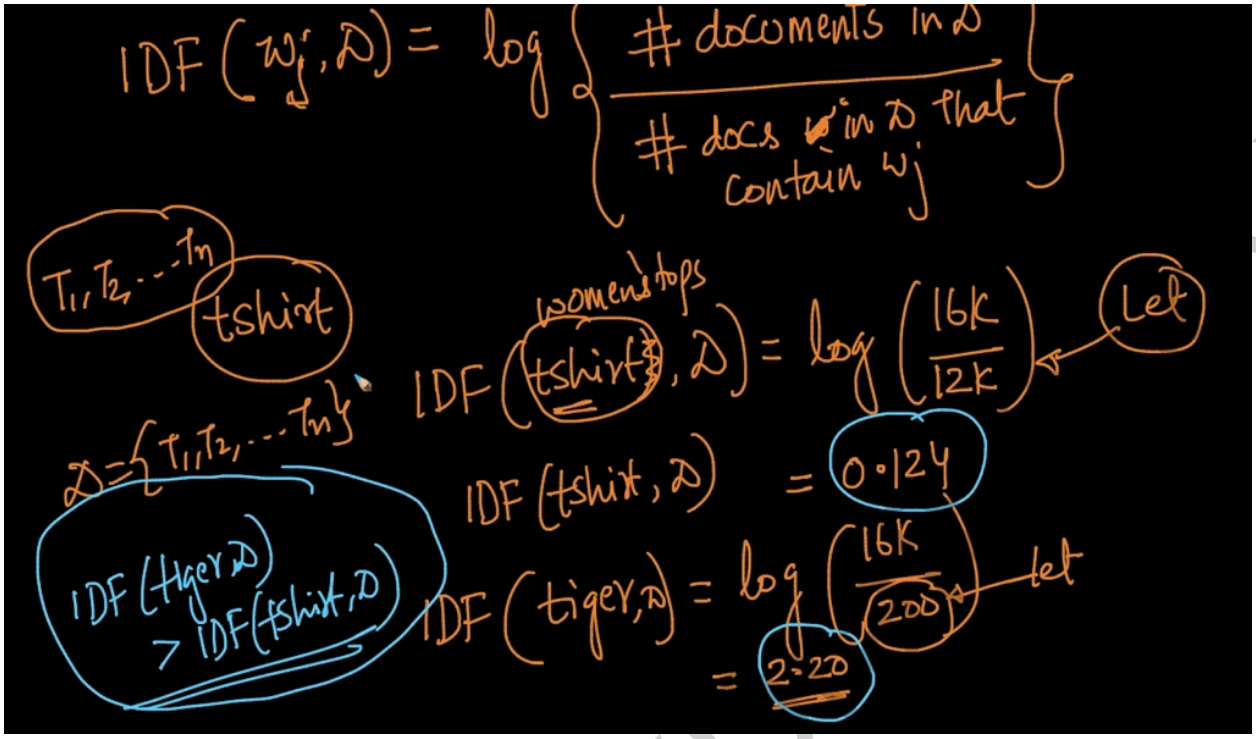
So far we have used the Bag of Words approach to convert the given text into numerical vector form.




In this approach, we replace the counts with the product of Term Frequency and the Inverse Document Frequencies.

*Term Frequency ($w_j$, $T_j$) = Number of times the word '$w_j$' has occurred in the text '$T_j$'/(Total number of words in the text '$T_j$')*

*Inverse Document Frequency ($w_j$, D) = $\log_e$(Total number of documents in the corpus/number of documents that contain the word '$w_j$')*




So now while vectorizing a text '$T_i$', if it contains the word '$w_j$', then the value/component associated with this word '$w_j$' is filled with **TF($w_j$, $T_i$)*IDF($w_j$, D)**. If the word '$w_j$' is not present in the text '$T_i$', then the corresponding component becomes 0.

The number of dimensions produced in BOW and TF-IDF is the same. The only difference lies in the strategy used to compute the vector components.

The approach of computing the similarities is again the same from here. We have to compute the pair-wise distances between the TF-IDF vectors, and those vectors that are nearer according to the euclidean distance, are considered to be similar products.

The code to perform the above task is given below

```
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 sparase matrix of dimensions #data_points * #words_in_corpus
# tfidf_title_features[doc_id, index_of_word_in_corpus] = tfidf values of the word in given doc
```

```
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 apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, 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 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]], data['medium_image_url'].loc[
        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(12566, 20)
# in the output heat map each value represents the tfidf values of the label word, the color represents the intersection with inp
```

The top 20 recommended products for the given query point with index 12566 are given below




```
ASIN : B00JXQB5FQ
BRAND : Si Row
Eucliden distance from the given image : 0.0
=============================================================================================================================
```




```
ASIN : B00JXQASS6
BRAND : Si Row
Eucliden distance from the given image : 0.753633191245
=============================================================================================================================
```




```
ASIN : B00JXQCWTO
BRAND : Si Row
Eucliden distance from the given image : 0.935764394377
=============================================================================================================================
```

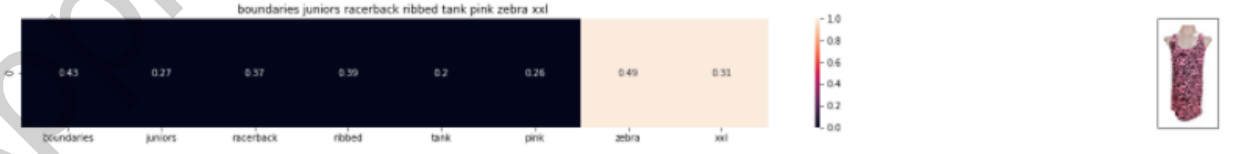

ASIN : B00JXQAFZ2
BRAND : Si Row
Eucliden distance from the given image : 0.95861535242
==========


ASIN : B00JXQCUIC
BRAND : Si Row
Eucliden distance from the given image : 1.00007496145
==========


ASIN : B00JXQAO94
BRAND : Si Row
Eucliden distance from the given image : 1.02321555246
==========


ASIN : B00JXQAUWA
BRAND : Si Row
Eucliden distance from the given image : 1.0319918463
==========

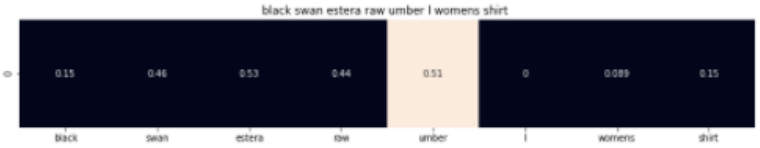

ASIN : B06XSCVFT5
BRAND : Studio M
Eucliden distance from the given image : 1.21068436704
==========




ASIN : B06Y2GTYPM
BRAND : No Boundaries
Eucliden distance from the given image : 1.21216838107
==========

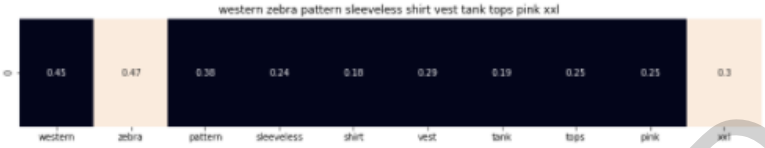

ASIN : B012VQLT6Y
BRAND : KM T-shirt
Eucliden distance from the given image : 1.21979064028

---



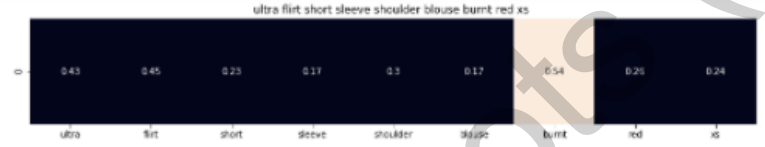

ASIN : B06Y1VN8WQ
BRAND : Black Swan
Eucliden distance from the given image : 1.220684966

---



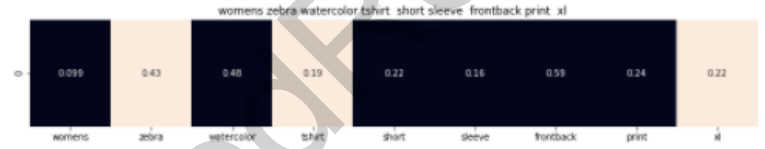

ASIN : B00Z6HEXWI
BRAND : Black Temptation
Eucliden distance from the given image : 1.22128139212

---



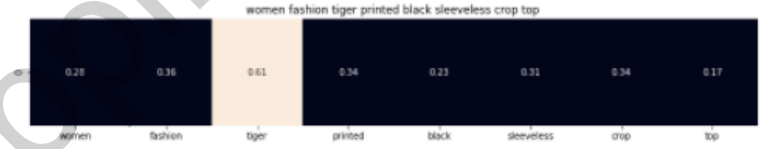

ASIN : B074TR12BH
BRAND : Ultra Flirt
Eucliden distance from the given image : 1.23133640946

---



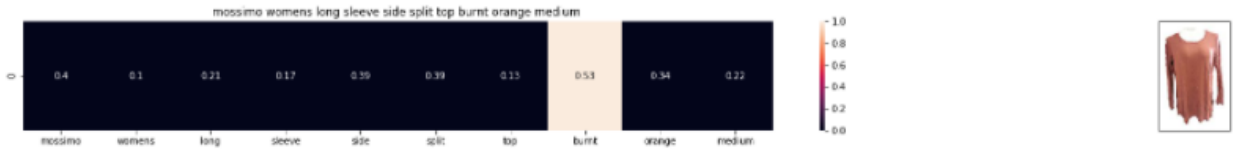

ASIN : B072R2JXKW
BRAND : WHAT ON EARTH
Eucliden distance from the given image : 1.23184519726

---

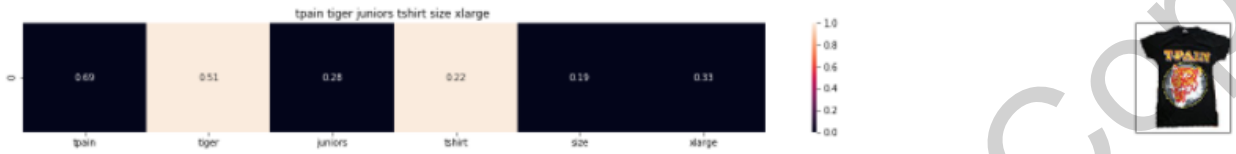



ASIN : B074T8ZYGX
BRAND : MKP Crop Top
Eucliden distance from the given image : 1.23406074574

---



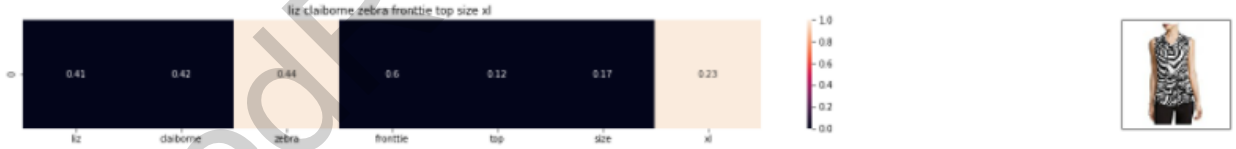

ASIN : B071ZDF6T2
BRAND : Mossimo
Eucliden distance from the given image : 1.23527855777




ASIN : B01K0H02OG
BRAND : Tultex
Eucliden distance from the given image : 1.23645729881


ASIN : B00H8A6ZLI
BRAND : Vivian's Fashions
Eucliden distance from the given image : 1.24996155053

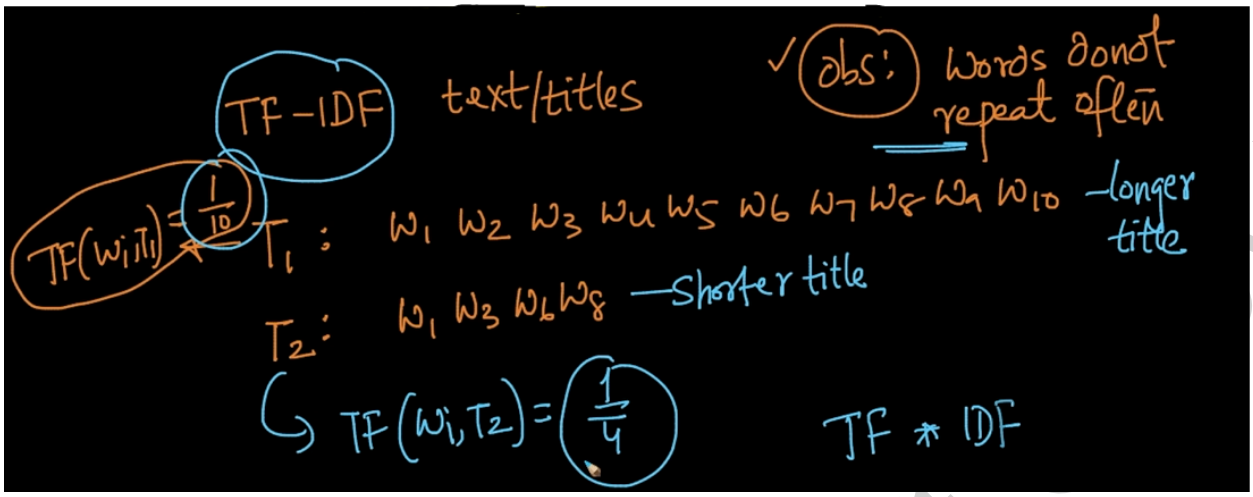

ASIN : B010NN9RXO
BRAND : YICHUN
Eucliden distance from the given image : 1.25354614209




ASIN : B06XBY5QXL
BRAND : Liz Claiborne
Eucliden distance from the given image : 1.25388329384

## IDF Based Product Similarity

So far we have worked with TF-IDF vectorization. But here we have a problem. Let us see the example given below.




In this example, we have two sentences '$T_1$' and '$T_2$'. In both these statements, all the words occur only once. So

a) In '$T_1$', for any word '$w_i$', TF($w_i$, $T_1$) = 1/10
b) In '$T_2$', for any word '$w_i$', TF($w_i$, $T_2$) = ¼

Here even though all the words occur only once in both these sentences, the lengths of these two sentences are different. So such values could not yield appropriate results in terms of recommendations.

The TF-IDF approach was designed to work with search engines and large documents. So in order to get rid of the above problem, we drop the TF score and use only the IDF scores to compute the similarities between the vectors.

```
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 #data_points * #words_in_corpus
# idf_title_features[doc_id, index_of_word_in_corpus] = number of times the word occured in that doc
```

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

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

```
# 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 values of the word
    # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will return all documents in which the word i prese
    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 i
        # idf_title_features[doc_id, index_of_word_in_courpus] = idf value of word
        idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val
```

```
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 apparels
    # the metric we used here is cosine, the coside distance is mesured as K(X, Y) = <X, 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]], data['medium_image_url'].loc[d
        print('ASIN :',data['asin'].loc[df_indices[i]])
        print('Brand :',data['brand'].loc[df_indices[i]])
        print ('euclidean distance from the given image :', pdists[i])
        print('='*125)


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

When we compare the results, we can conclude that the IDF-based similarity doesn't yield much better results in this case, when compared to the TF-IDF-based similarity.

So the order of vectorizations on the basis of yielding appropriate results is **TF-IDF > IDF > BOW**.

**Semantic Similarity and Word2Vec**

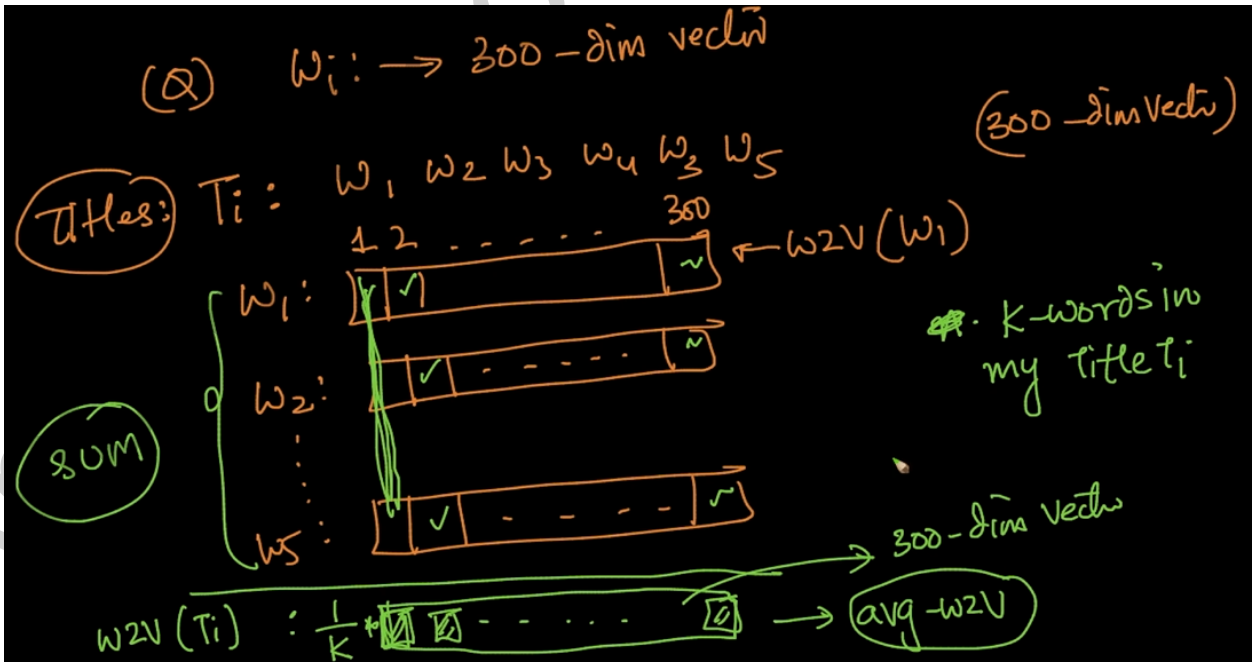
So far we have seen BOW and TF-IDF vectorization in featurizing the text data, but these techniques do not account for the semantic meaning, as they are frequency-based techniques. One technique we use to vectorize the text data, and also account for the semantic meaning is **Word2Vec**.

**Word2Vec**

In this technique, if we give a word '$w_i$, then it gives a d-dimensional array. (For this workshop, we shall assume d=300). Word2Vec gives the result in the form of a dense matrix, whereas techniques like BOW/TF-IDF give the result in the form of a sparse matrix.

So far in BOW and TF-IDF techniques, the vectors are created for text statements, whereas in Word2Vec, the vectors are created for each word. But Word2Vec requires a huge data corpus for training. Word2Vec preserves the relationships like male-female, country-capital, country-currency, words with their tenses, etc. All the vectors with similar relationships are almost parallel to each other.

**Average Word2Vec**



One of the variants of the Word2Vec model is the **Average Word2Vec**. In this variant, we perform the average of the word vectors of all the words present in the given text.

**TF-IDF Weighted Average Word2Vec**

So far we have seen the Average Word2Vec technique where we have performed a simple average of the word vectors of all the words present in the given statement/text. Here the weightage of all the words in the given text is the same.

But here in this context, we multiply the component of each word in a vector with its TF-IDF score (which is a product of TF and IDF scores).

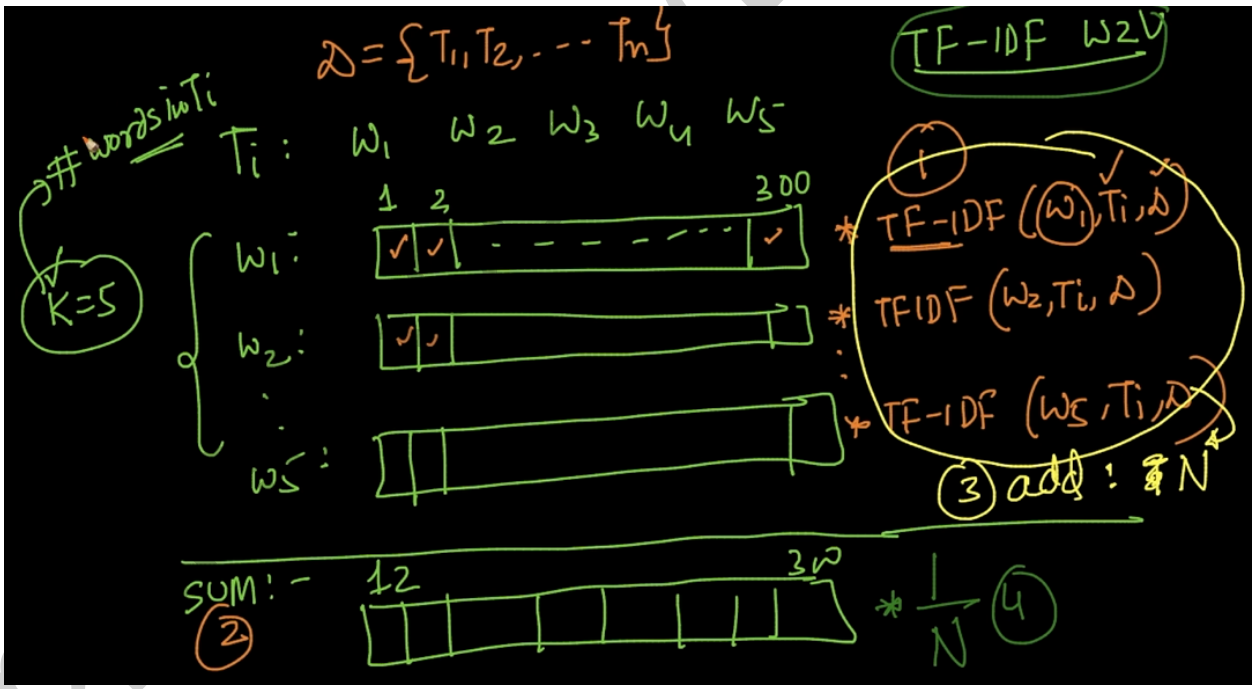
$TF(w_i, T_j)$ → Number of times the word '$w_i$' occurs in the text '$T_j$'.

$IDF(w_i, D) = \log_e(N/n_i)$

Where N → total number of texts/documents in the given corpus

$n_i$ → Number of documents/texts that contain the word '$w_i$'.

D → Data Corpus




**Step 1**

Here for every word '$w_i$', we have to create their word vectors. For a query text '$T_j$', we have to check all the words present in '$T_j$', and multiply the word vectors of every word '$w_i$' with the TF-IDF($w_i$, $T_j$, D) score.

**$TF\text{-}IDF(w_i, T_j, D) = TF(w_i, T_j) * IDF(w_i, D)$**

**Step 2**

Compute the sum of all the TF-IDF weighted word vectors.

**Step 3**

Compute the total number of texts present in the data corpus. Let us denote it as ‘N’.

**Step 4**

Compute the average by dividing the whole sum vector by ‘N’. The result obtained is the TF-IDF Weighted Average Word2vec vector.

The major thing that is present in TF-IDF Weighted Average word2vec is that it takes the word importances and the semantic similarity into consideration while building a vector for the given text. This concept of taking the word importance and semantic similarity into consideration is not seen in Average Word2Vec.

We also have another approach which is called **IDF weighted Wor2Vec**, in which instead of multiplying the word vectors with the TF-IDF scores, we only multiply with the IDF scores. This approach has to be used only when the Term Frequencies (TF scores) are very small. In such cases, the IDF scores dominate the TF scores, and hence the presence of the TF scores will have a negligible impact.

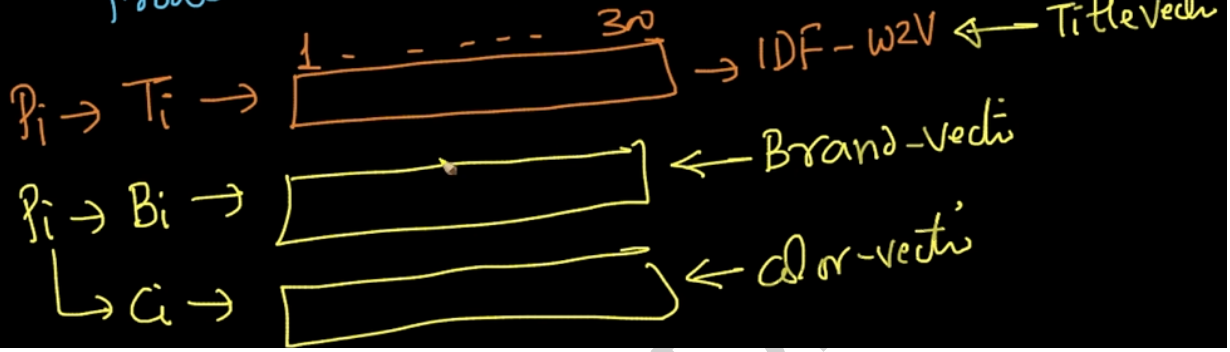
**Weighted Similarity using Brand and Color**

So far we have applied different vectorization techniques on the text data. Now we have to vectorize the ‘Brand’ and the ‘Color’ values using one-hot encoding, to create the brand and the color vectors respectively.

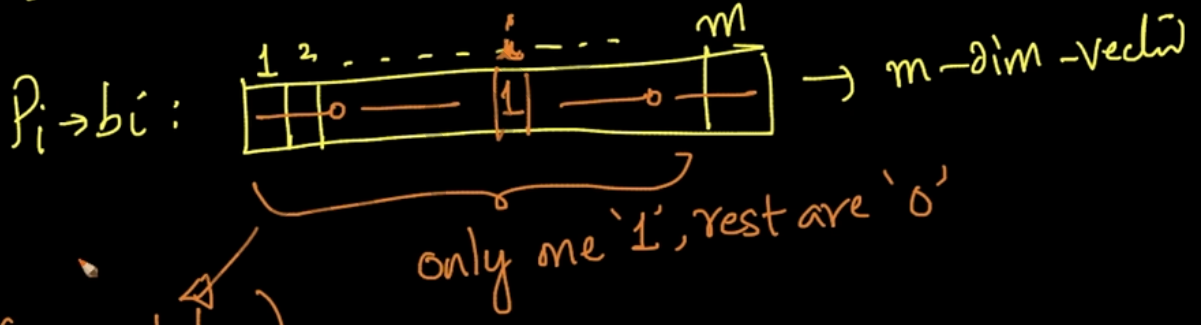
# Weighted similarity using Brand & color

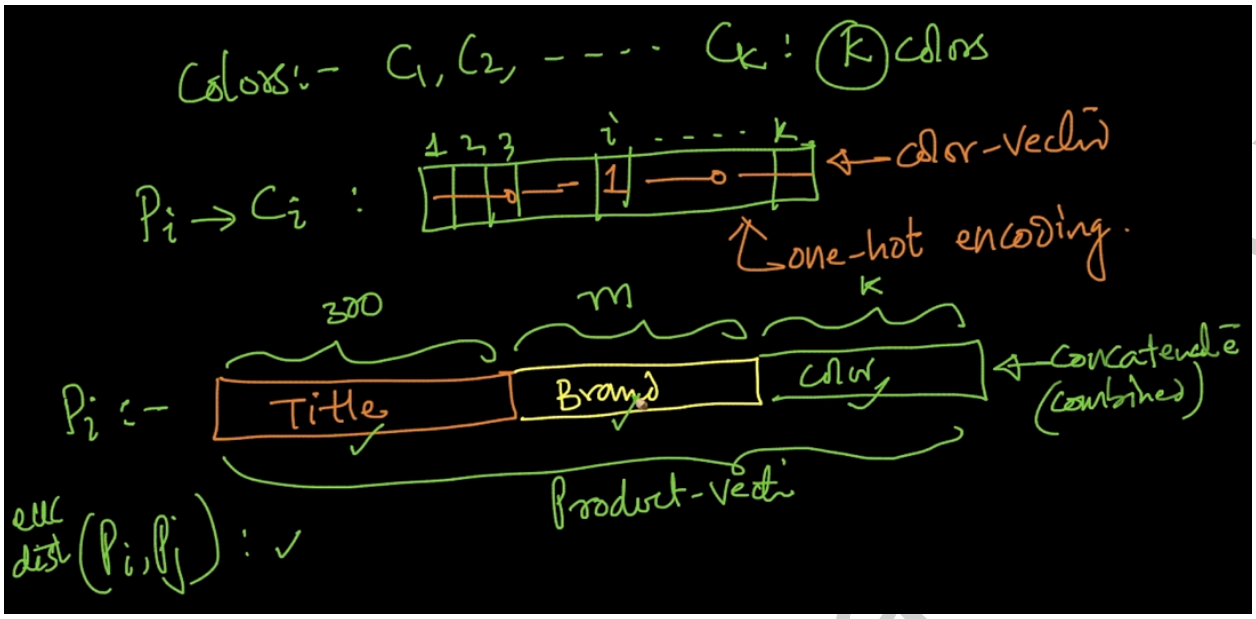
Titles → product similarity

Product → Title, Brand, color




brands: $b_1, b_2, b_3, \ldots b_m$ : m-brands





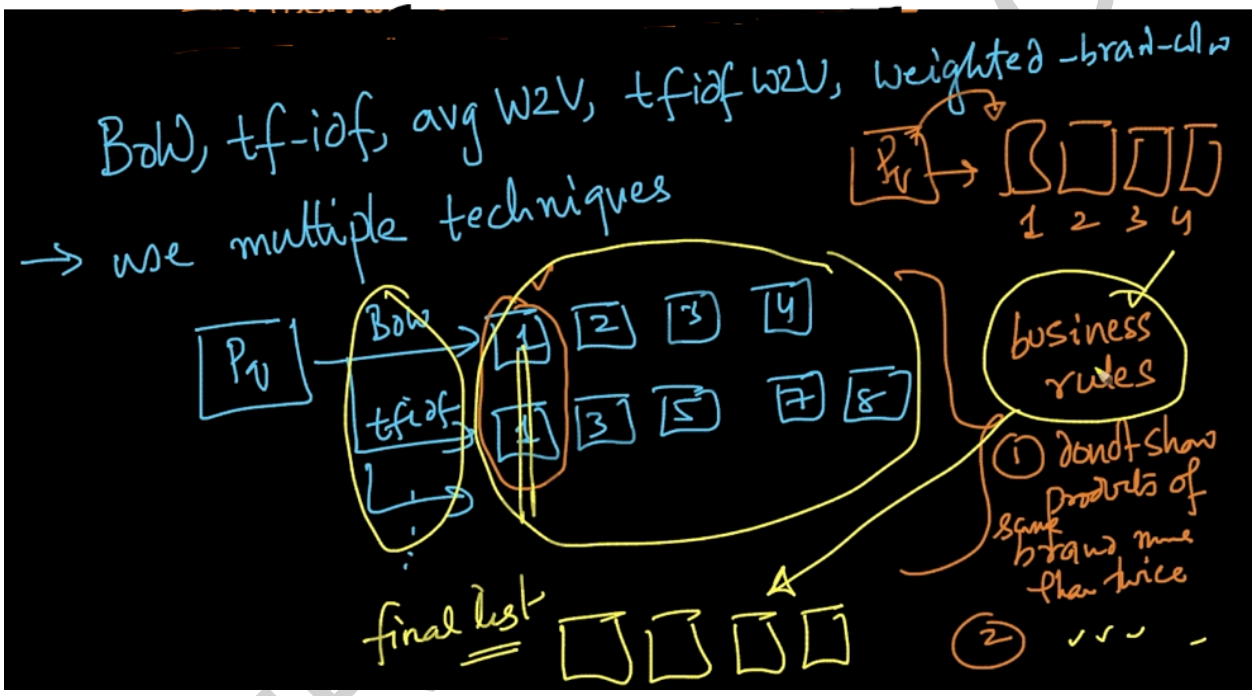

In this way, after vectorizing the text data, 'brand' feature and the 'color' feature values, we have to concatenate them, and then use these concatenated vectors to find the similarities between the products in terms of the euclidean distances.

In case, if we want to recommend the products of a particular brand or a particular color, then in order to get such results,we have to slightly modify the euclidean distance. Instead of the simple euclidean distance, we have to compute the weighted euclidean distances.

In order to compute the weighted euclidean distances, we have to first come up with the weights '$w_T$', '$w_B$' and '$w_C$'. For every product vector '$p_i$', we have to multiply the components obtained by the text vectorization with the value '$w_T$', the components obtained by the 'Brand' value vectorization with the value '$w_B$', and the components obtained by the 'Color' value vectorization with the value '$w_C$'. All these vectors obtained after multiplying the product vectors with the weights are called the **weighted product vectors**. We now have to compute the euclidean distances between these weighted product vectors, and recommend the products accordingly. Such euclidean distance obtained between the weighted product vectors is called **weighted euclidean distance**.

**Building a real-world solution**

So far we have applied techniques like BOW, TF-IDF, Average Word2Vec, TF-IDF Weighted Word2Vec, Weighted brand and color, etc to give better and appropriate recommendations to the query users. In real-time we take all the recommendations obtained by using the above mentioned techniques, combine them and apply business rules to get a final list of the recommendations to the users. This solution is not just about recommending the right items/products, but also should have a huge impact on the business.



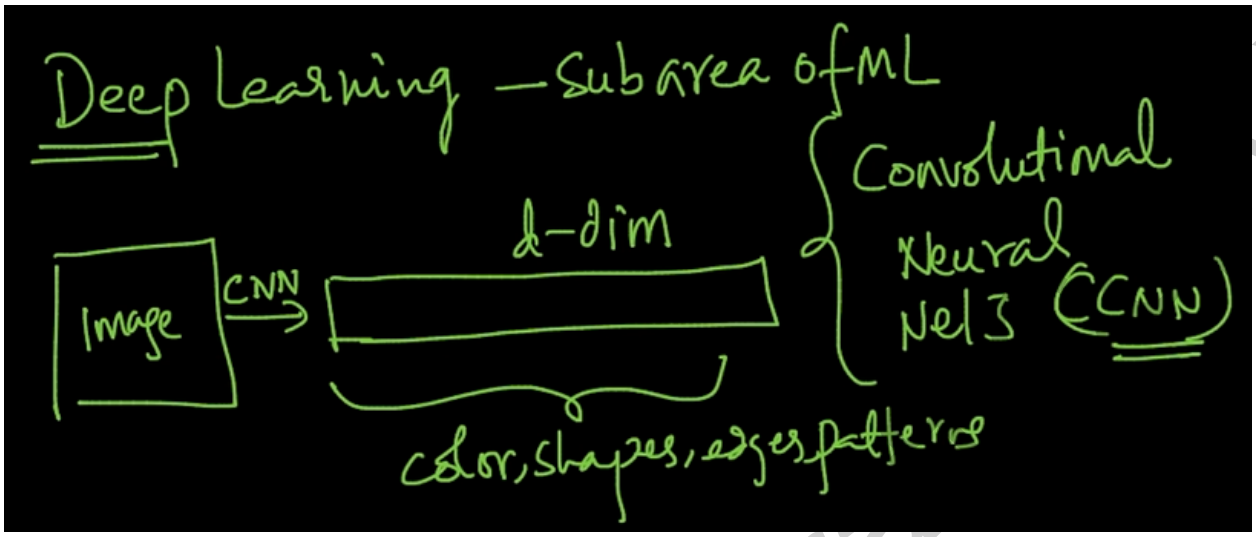

In order to measure the goodness of our solution, we have to go for A/B testing.

**Deep Learning based Visual Similarity**

So far we have used the text features, brand name, and color, and applied techniques like BOW, TF-IDF, Word2Vec, One-Hot-Encoding, etc to vectorize the data and obtain similar items for recommendations.

But there is one thing that we have missed. For every product, there is a URL of the image. We can download the image, and featurize/convert the image into a vector form. There could be edges, shapes, colors, patterns, etc in the image. Given an image, we have to construct a vector

with the values associated with these properties, using the Convolutional Neural Networks.



**Using Keras+Tensorflow to extract the features**

The below code snippet generates the features out of the given images.

```
# 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 // batch_size)
    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()
```

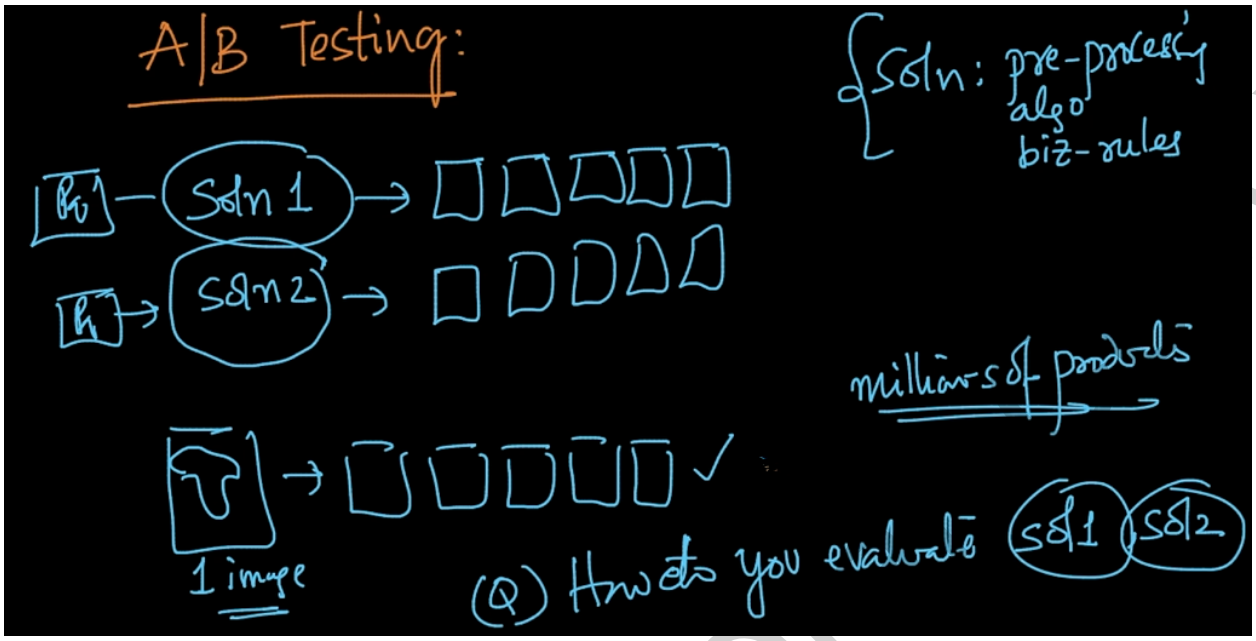
## Visual Similarity Based Product Similarity

```
#load the features and corresponding ASINS info.
bottleneck_features_train = np.load('16k_data_cnn_features.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[doc_id].reshape(1,-1))

    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.amzon.com/dp/'+ asins[indices[i]])

get_similar_products_cnn(12566, 20)
```

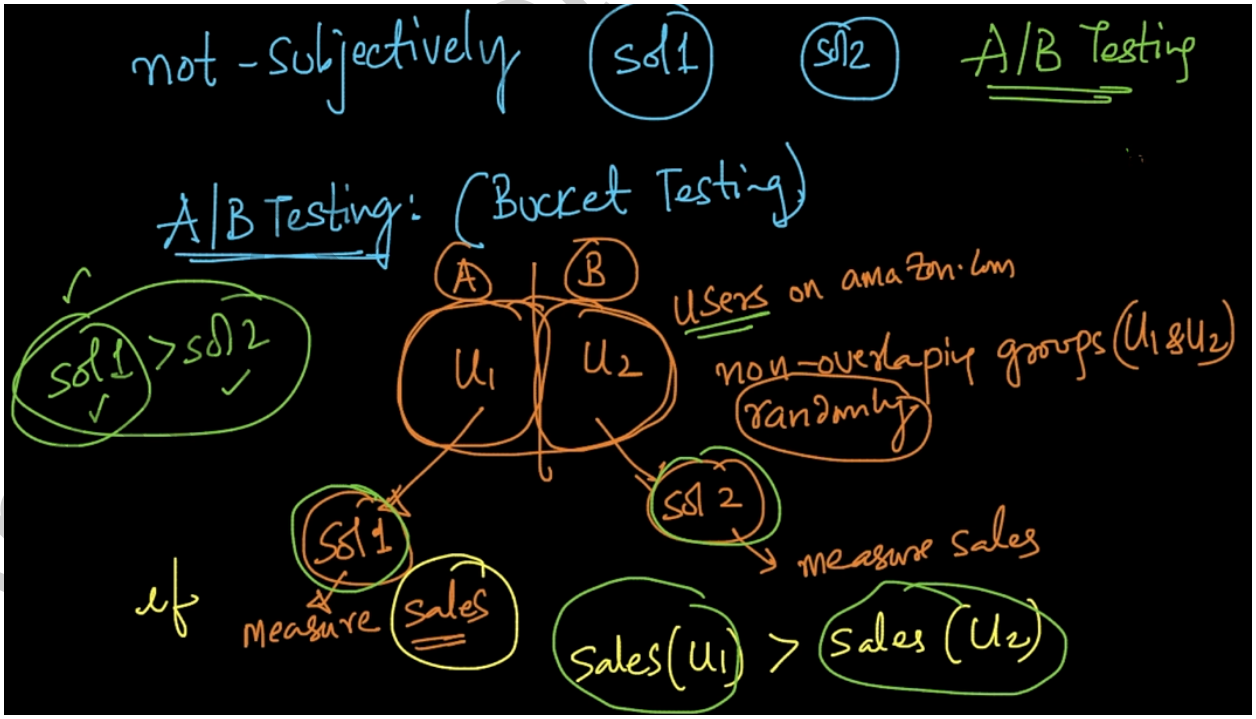
In the above code, we are loading the CNN-generated features. Again we compute the pairwise distances between the points and recommend similar items.

In this recommendation, the title, brand name, and color aren't taken into consideration for recommending. It is only the CNN-generated features of each image that are taken into consideration, for computing the pairwise distances and recommending similar products to the users.

## A/B Testing




Let us assume, we have built two solutions for the given problem. **Solution 1** recommends a few products, and **Solution 2** also recommends a few products to a query user. Now the task here is to find out which solution among the two is better. We have a statistical technique called **A/B Testing** that is used to find out which solution is better among the two.



In our problem of recommending the items to the users, we divide the users randomly into two non-overlapping groups. Let us denote these two groups as '$\mathbf{u_1}$' and '$\mathbf{u_2}$'. Let us recommend the products/items that resulted from solution 1 to the users in '$\mathbf{u_1}$' and the products/items that resulted from solution 2 to the users '$\mathbf{u_2}$'. Now we have to compare the sales from the groups '$\mathbf{u_1}$' and '$\mathbf{u_2}$'. If the number of sales obtained from the group '$\mathbf{u_1}$' is greater than the number of sales obtained from the group '$\mathbf{u_2}$', then it means the solution 1 is liked/preferred by the users in '$\mathbf{u_1}$', then the solution 2 by the users in '$\mathbf{u_2}$'. So we go ahead with deploying solution 1 in production to increase the sales.