Amzn_apparel_recommendation_Excercise

February 14, 2019

0.1 Excercise: Build a weighted Nearest Neighbor model using visual,brand,text and color?

```
In [1]: from PIL import Image #support for opening,manipulating,saving any different file form
        import requests #deals with url's
        from io import BytesIO #data can be kept as in-memory buffer which is faster
        import matplotlib.pyplot as plt #for graphs and visualization
        import numpy as np #for numerical computation
        import pandas as pd #for data manipulation
        import warnings #to avoid errors
        from bs4 import BeautifulSoup #for webscrapping
        from nltk.corpus import stopwords #to avoid stopwords
        from nltk.tokenize import word_tokenize #splits text into words
        import nltk #natural language toolkit library
        import math #for math operations
        import time #for printing excution time
        import re #delas with regular expression
        import os #for reading and writing to system
        import seaborn as sns #for effective visualisation
        from collections import Counter #counter
        from sklearn.feature_extraction.text import CountVectorizer #extract text features and
        from sklearn.feature_extraction.text import TfidfVectorizer #tf-idf transformer
        from sklearn.metrics.pairwise import cosine_similarity #computes similarity with dot p
        from sklearn.metrics import pairwise_distances #computes distance matrice
        from matplotlib import gridspec #specifies geometry of gird that a subplot will be pla
        from scipy.sparse import hstack #horizontal stack sparse matrix horizontally
        import plotly #for 3d visualization
        import plotly.figure_factory as ff #wrapper function contains unique chart types
        from plotly.graph_objs import Scatter, Layout #defines size and layout

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

```
In [2]: #after removing duplicates and stop words we got 16k_apparel_data_preprocessed pickle
        data = pd.read_pickle('pickels/16k_apperal_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 [ ]:

In [8]:
```
idf_title_vectorizer = CountVectorizer()
idf_title_features = idf_title_vectorizer.fit_transform(data['title'])
```

In [10]:
```
def n_containing(word):
    # return the number of documents which had the given word
    return sum(1 for blob in data['title'] if word in blob.split())

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

In [11]:
```
# 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 id
    # idf_title_features[:, idf_title_vectorizer.vocabulary_[i]].nonzero()[0] will re
    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_courpus] = idf value of word
        idf_title_features[j,idf_title_vectorizer.vocabulary_[i]] = idf_val
```

In [ ]:

In [16]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
with open('word2vec_model', 'rb') as handle:
    model = pickle.load(handle)
```

In [21]:

```
# 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):
    # sentace: its title of the apparel
    # num_features: the lenght of word2vec vector, its values = 300
    # m_name: model information it will take two values
        # if  m_name == 'avg', we will append the model[i], w2v representation of wor
        # if m_name == 'weighted', we will multiply each w2v[word] with the idf(word)

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

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

In [ ]:

In [24]:

```
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 [29]:

```
import numpy as np #numerical computation
from keras.preprocessing.image import ImageDataGenerator#processing image
from keras.models import Sequential #creating sequential model
```

```
         from keras.layers import Dropout, Flatten, Dense #sequential model with droput,flatte
         from keras import applications #to use pretrained models on imagenet
         from sklearn.metrics import pairwise_distances #for distance
         import matplotlib.pyplot as plt #for visualization
         import requests #for url dealing
         from PIL import Image #for image manipulating, saving diff formats
         import pandas as pd # for data processing
         import pickle #for faster loading

In [31]: #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

In [32]: # some of the brand values are empty.
         # Need to replace Null with string "NULL"
         data['brand'].fillna(value="Not given", inplace=True )

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

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

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

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

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

In [67]: def predicter(doc_id,document_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) =
             # http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity
```

```
    idf_w2v_dist  = pairwise_distances(w2v_title_weight, w2v_title_weight[doc_id].resh

    ex_feat_dist = pairwise_distances(extra_features, extra_features[doc_id])

    document_id = asins.index(df_asins[document_id])
    imge_dist = pairwise_distances(bottleneck_features_train, bottleneck_features_trai


    pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist + w3 * imge_dist)/float

    # 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(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]])
            print('='*125)

predicter(12566,4,5, 5, 5, 20)
# in the give heat map, each cell contains the euclidean distance between words i, j
```

```
    ---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-67-a12899a2b3e4> in <module>
     35             print('='*125)
     36
---> 37 predicter(12566,4,5, 5, 5, 20)
     38 # in the give heat map, each cell contains the euclidean distance between words i,


    <ipython-input-67-a12899a2b3e4> in predicter(doc_id, document_id, w1, w2, w3, num_resul
     15
     16
---> 17     pairwise_dist   = (w1 * idf_w2v_dist +  w2 * ex_feat_dist + w3 * imge_dist)/flo
     18
     19     # np.argsort will return indices of 9 smallest distances

    ValueError: operands could not be broadcast together with shapes (16435,1) (16042,1)
```

```
In [ ]:
```