Make an e-commerce item recommender system with content-based filtering

```
                                                                written by: zohreh karimi
```

We'll learn how to make an e-commerce item recommender system with a technique called **content-based filtering.**

A content-based recommender works with data that the user provides,either explicitly(rating) or implicitly(clicking on a link).Based on that data, a user profile is generated,which is then used to make suggestions to the user. As the user provides more inputs or takes actions on those recommendations, the engine becomes more and more accurate.

A recommendation system has to decide between two methods for information delivery when providing the user with recommendations:

Explotation: The system chooses documents similar to those for which the user already expressed a preference.

Exploration: The system chooses documents where the user profile does not provide evidence to predict the user's reaction.

Let's work through an implementation of a content-beased filtering system.

dataset:use this link for a dataset of 500 entries of different items like shoes,shirt,etc,with an item-id and a textual description of the item. *https://github.com/nikitaa30*

```
 #import librariries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

data=pd.read_csv('sample-data.csv')
print(data.head())
```

Creating a TF-IDF Vectorizer

The **TF-IDF** algorithm is used to weight a keyword in any document and assign the importance to the keyword based on the number of times it appeares in the document. The higher the TF-IDF score(weight) the rarer and more important the term.

Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called **the TF-IDF weight of the term.**

TF(term frequency) of a word is the number of times it appears in a document.

TF(t)=(Number of times term t appears in a document)/(Total number of terms in the documnet)

IDF(inverse document frequency)of a word is the measure of how significant in the whole corpus.

IDF(t)=log_e(Total number of documents)/Number of documents with term t in it.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

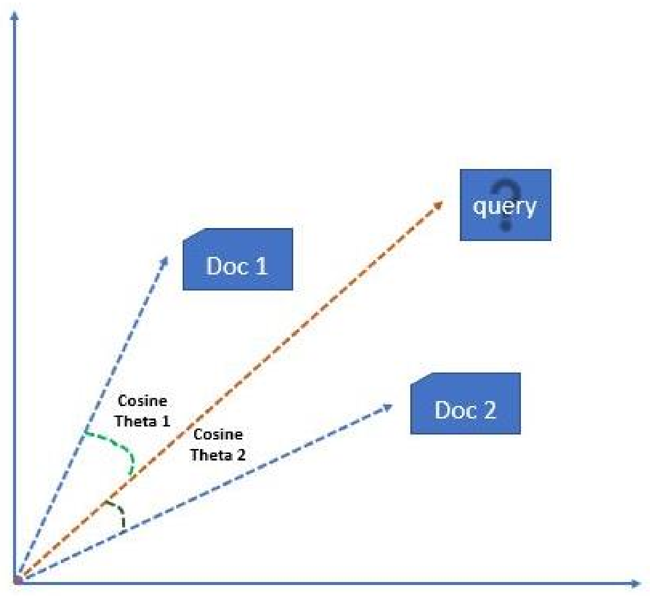
TF-IDF calculation

```
 #create Tfidf
tf=TfidfVectorizer(analyzer='word',ngram_range=(1,3),min_df=0,stop_words='english')
tfidf_matrix=tf.fit_transform(data['description'])
```

the *tfidf_matrix* is the matrix containing each word and its TF-IDF score with regard to each document , or item in this case. stop words are simply words that add no significant value to our system, like 'an','is','the', and hence are ignored by the system. Now we have a representation of every item in terms of its description. Next, we need to calculate the relevance or similarity of one document to another.

Vector Space Model

In this model, each item is stored as a vector of its attributes(which are also vectors)in an n-dimensional space, and the angle between the vectors are calculated to determine the similarity between the vectors.



Documents represented as vectors

The method of calculating the user's like/dislike/measures is calculated by taking the cosine of the angle between user profile vector (Ui) and the document vector. [here we use the angle between two document vectors] So, we'll calculate the cosine similarity of each item with every other item in the dataset , and then arrange them according to their similarity with item *i*, and store the values in *results*.

```
cosine_similarities=linear_kernel(tfidf_matrix,tfidf_matrix)
result={}

for item,row in data.iterrows():
    similar_indices=cosine_similarities[item].argsort()
    similar_items=[(cosine_similarities[item][i],data['id'][i])for i in similar_indices]
    result[row['id']]=similar_items[1:]
```

```
# Making a recommendation
def item(id):
    return data.loc[data['id']==id]['description'].tolist()[0].split('-')[0]

def recommend(item_id,num):
    print('Recommending ' + str(num)+' products similar to '+ item(item_id)+ '...')
    print('-------------------------------------------------------------------------')
    recs=result[item_id][:num]
    for rec in recs:
        print('recommended: '+item(rec[1])+"(score: "+ str(rec[0])+ ")")
```

```
recommend(item_id=20,num=4)
```

```
Recommending 4 products similar to Cap 1 graphic t...
-----------------------------------------------------------------------
recommended: Flyfishing the athabasca poster (score: 0.01706551235263075)
recommended: Symmetry w16 poster (score: 0.017105744408204927)
recommended: Lead an examined life poster (score: 0.017374631822288104)
recommended: Sticks 'n stones morocco poster (score: 0.018157605349684747)
```

```
 **Code**

#import librariries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

#load dataset
data=pd.read_csv('sample-data.csv')



#create Tfidf
tf=TfidfVectorizer(analyzer='word',ngram_range=(1,3),min_df=0,stop_words='english')
tfidf_matrix=tf.fit_transform(data['description'])

#cosine similarity
cosine_similarities=linear_kernel(tfidf_matrix,tfidf_matrix)
result={}
for item,row in data.iterrows():
    similar_indices=cosine_similarities[item].argsort()
    similar_items=[(cosine_similarities[item][i],data['id'][i])for i in similar_indices]
    result[row['id']]=similar_items[1:]

# Making a recommendation
def item(id):
    return data.loc[data['id']==id]['description'].tolist()[0].split('-')[0]

def recommend(item_id,num):
    print('Recommending ' + str(num)+' products similar to '+ item(item_id)+ '...')
    print('---------------------------------------------------------------------')
    recs=result[item_id][:num]
    for rec in recs:
        print('recommended: '+item(rec[1])+"(score: "+ str(rec[0])+ ")")

recommend(item_id=20,num=4)
```

reference:

```
 - heartbeat.fritz.ai
- https://github.com/nikitaa30
```