CHAPTER 4

# Enhancing E-commerce Using an Advanced Search Engine and Recommendation System

A few decades ago, no one would have ever imagined that we could buy a 55-inch TV at midnight while sitting at home watching a 22-inch TV. Thanks to the Internet and e-commerce, we can buy any item at any time from anywhere, and it is delivered quickly. Flexibility has made e-commerce businesses expand exponentially. You don't have to visit the store, products have unlimited options, prices are lower, no standing in line to pay, and so forth.

Given the traction e-commerce is getting, many big names are taking part in it. Companies keep technology in check along with operations, supply chains, and marketing. To survive competition, the right use of digital marketing and social media presence is also required. Also, most importantly, businesses must leverage data and technology to personalize the customer experience.

## Problem Statement

One of the most talked-about problems of this era is recommendation systems. Personalization is the next big data science problem. It's almost everywhere—movies, music, e-commerce sites, and more.




A. Kulkarni et al., *Natural Language Processing Projects*, https://doi.org/10.1007/978-1-4842-7386-9_4

Since the applications are wide, let's pick an e-commerce product recommendation as one of the problem statements. There are multiple types of recommender systems. But the one that deals with text is content-based recommender systems. For example, if you see the diagram shown in Figure 4-1, similar products have been recommended based on the product description of the clicked product. Let's explore building this kind of recommender system in this chapter.

On the same lines, search engines in e-commerce websites also play a major role in user experience and increasing revenue. The search bar should give the relevant matches for a search query, and wrong results eventually result in the churn of the customers. This is another problem that we plan to solve.

To summarize, in this project, we aim to build a search and recommender system that can search and recommend products based on an e-commerce data set.

# Approach

Our main aim is to recommend the products or items based on users' historical interests. A recommendation engine uses different algorithms and recommends the most relevant items to the users. It initially captures the past behavior of the users. It recommends products based on that.

Let's discuss the various types of recommendation engines in brief before we move further. Figure 4-1 shows the types of recommendation engines.

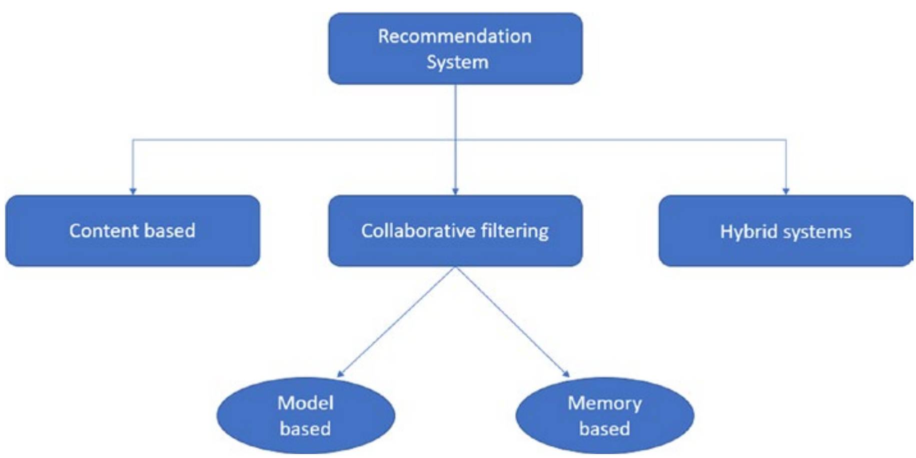


***Figure 4-1.** Types of recommendation engines*

The following are various types of recommendation engines.

- Market basket analysis (association rule mining)
- Content based
- Collaborative filtering
- Hybrid systems
- ML clustering based
- ML classification based
- Deep learning and NLP based

## Content-Based Filtering

Content Filtering algorithm suggests or predicts the items similar to the ones that a customer has liked or shown any form of interest. Figure 4-2 shows the example of content-based filtering.

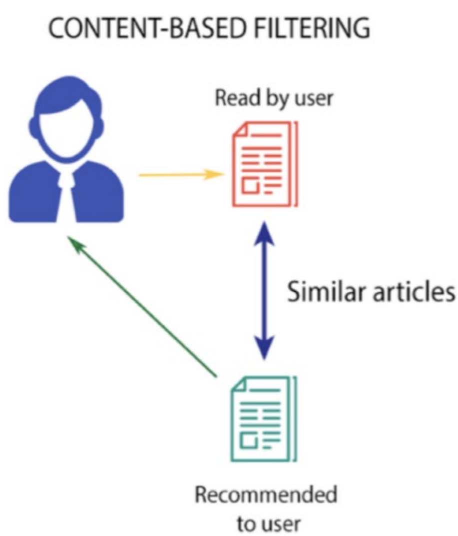


***Figure 4-2.*** *Content-based filtering*

The project aims to use deep learning techniques for information retrieval rather than the traditional word comparison approach to get better results. Also, it focuses on recommender systems that are everywhere and creates personalized recommendations to increase the user experience.

The methodology involves the following steps.

1. Data understanding
2. Preprocessing
3. Feature selection
4. Model building
5. Returning search queries
6. Recommending product

Figure 4-3 shows the Term Frequency–Inverse Document Frequency (TF-IDF) vectors-based approach to building a content-based recommendation engine that gives a matrix where every word is a column.

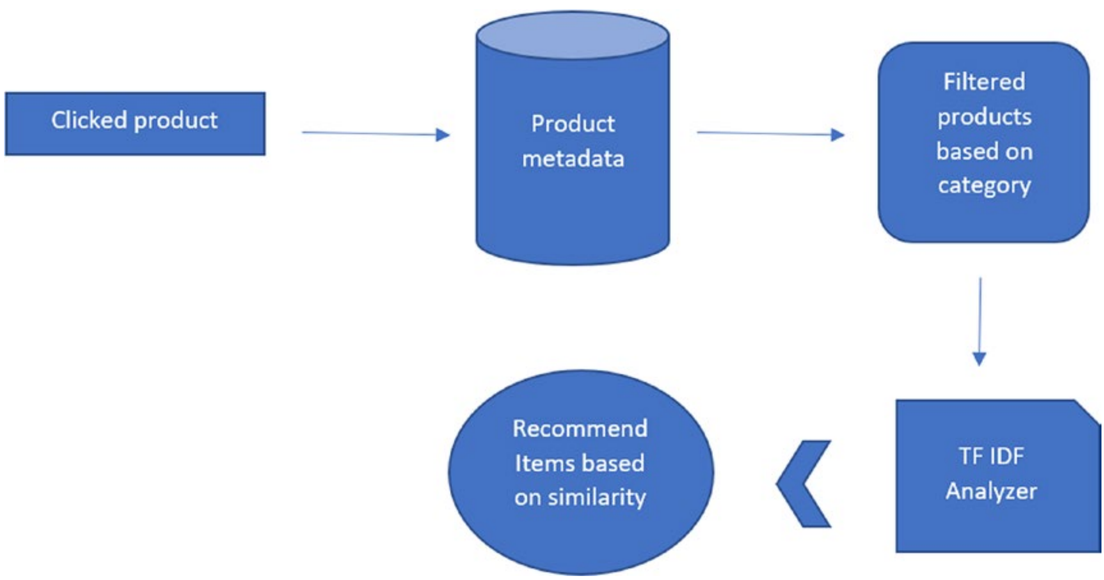


***Figure 4-3.*** *TF-IDF-based architecture*

Figure 4-4 shows the architecture for a product search bar. Here, word embeddings are used. Word embedding is a language modeling technique where it converts text into real numbers. These embeddings can be built using various methods, mainly neural networks.

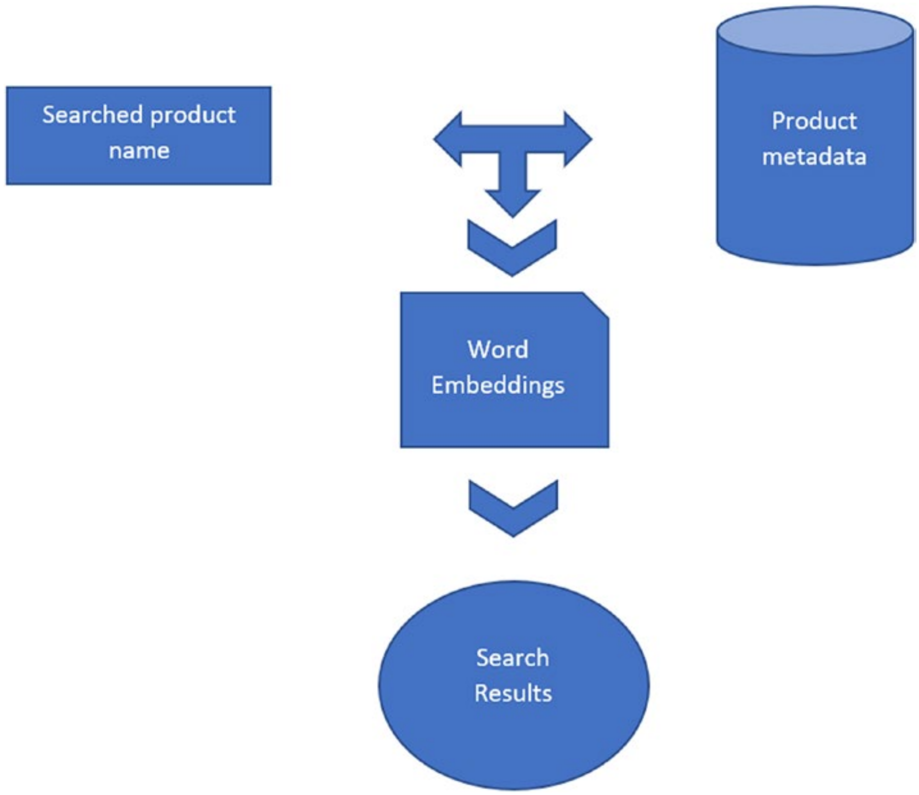


***Figure 4-4.*** *Word embeddings-based architecture*

## Environment Setup

Table 4-1 describes the environment setup that was used in this project. But, you could you Linux or macOS. To install Anaconda on Linux or macOS, please visit `www.anaconda.com`.

***Table 4-1.*** *Environment Setup*

| Set Up | Version |
|---|---|
| Processor | Intel(R) Core(TM) i5-4210U CPU @1.70GHz 2.40 GHz |
| Operating System | Windows 10- 64bit |
| Installed Memory (RAM) | 8.00 GB |
| Anaconda Distribution | 5.2.0 |
| Python | 3.6.5 |
| Notebook | 5.5.0 |
| NumPY | 1.14.3 |
| pandas | 0.23.0 |
| scikit-learn | 0.19.1 |
| Matplotlib | 2.2.2 |
| Seaborn | 0.8.1 |
| NLTK | 3.3.0 |
| Gensim | 3.4.0 |

# Understanding the Data

The e-commerce product recommendation data set has 20,000 observations and 15 attributes. The 15 features are listed in Table 4-2.

***Table 4-2.*** *Variables Present in the Data Set*

| Attribute Name | Data Type |
|---|---|
| uniq_id | object |
| crawl_timestamp | object |
| product_url | object |
| product_name | object |
| Pid | object |
| retail_price | float64 |
| discounted_price | float65 |
| image | object |
| is_FK_Advantage_product | bool |
| description | object |
| product_rating | object |
| overall_rating | object |
| brand | object |
| product_specifications | object |
| product_category_tree | object |

# Exploratory Data Analysis

The e-commerce data set has 15 attributes out of these labels, and we need the product name and description for this project.

Let's import all the libraries required.

```
#Data Manipulation
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#NLP for text pre-processing
import nltk
import scipy
import re
from scipy import spatial
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import PorterStemmer
tokenizer = ToktokTokenizer()

# other libraries
import gensim
from gensim.models import Word2Vec
import itertools
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA

# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# remove warnings
import warnings
warnings.filterwarnings(action = 'ignore')
```

Let's load the data set which you downloaded and saved in your local (see Figure 4-5).

| | uniq_id | crawl_timestamp | product_url | product_name | product_category_tree | pid | r |
|---|---|---|---|---|---|---|---|
| 0 | c2d766ca982eca8304150849735ffef9 | 2016-03-25 22:59:23 +0000 | http://www.flipkart.com/alisha-solid-women-s-c... | Alisha Solid Women's Cycling Shorts | ["Clothing >> Women's Clothing >> Lingerie, Sl... | SRTEH2FF9KEDEFGF | |
| 1 | 7f7036a6d550aaa89d34c77bd39a5e48 | 2016-03-25 22:59:23 +0000 | http://www.flipkart.com/fabhomedecor-fabric-do... | FabHomeDecor Fabric Double Sofa Bed | ["Furniture >> Living Room Furniture >> Sofa B... | SBEEH3QGU7MFYJFY | |
| 2 | f449ec65dcbc041b6ae5e6a32717d01b | 2016-03-25 22:59:23 +0000 | http://www.flipkart.com/aw-bellies/p/itmeh4grg... | AW Bellies | ["Footwear >> Women's Footwear >> Ballerinas >... | SHOEH4GRSUBJGZXE | |
| 3 | 0973b37acd0c664e3de26e97e5571454 | 2016-03-25 22:59:23 +0000 | http://www.flipkart.com/alisha-solid-women-s-c... | Alisha Solid Women's Cycling Shorts | ["Clothing >> Women's Clothing >> Lingerie, Sl... | SRTEH2F6HUZMQ6SJ | |
| 4 | bc940ea42ee6bef5ac7cea3fb5cfbee7 | 2016-03-25 22:59:23 +0000 | http://www.flipkart.com/sicons-all-purpose-arn... | Sicons All Purpose Arnica Dog Shampoo | ["Pet Supplies >> Grooming >> Skin & Coat Care... | PSOEH3ZYDMSYARJ5 | |

***Figure 4-5.*** *Sample data set*

```
data=pd.read_csv("flipkart_com-ecommerce_sample1.csv")
data.head()
```

```
data.shape
```

```
(20000, 15)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19998 entries, 0 to 19997
Data columns (total 15 columns):
uniq_id                    19998 non-null object
crawl_timestamp            19998 non-null object
product_url                19998 non-null object
product_name               19998 non-null object
product_category_tree      19998 non-null object
pid                        19998 non-null object
retail_price               19920 non-null float64
discounted_price           19920 non-null float64
image                      19995 non-null object
is_FK_Advantage_product    19998 non-null bool
description                19998 non-null object
product_rating             19998 non-null object
overall_rating             19998 non-null object
brand                      14135 non-null object
product_specifications     19984 non-null object
dtypes: bool(1), float64(2), object(12)
memory usage: 1.2+ MB
```

Here are the observations.

- The data set has a total of 15 columns and 20,000 observations.
- is_FK_Advantage_product is a boolean, the retail_price and discounted_price columns are numerical, and the remaining are categorical.

Let's add a new length column to give the total length of the 'description' input variable.

```
data['length']=data['description'].str.len()
```

Add a new column for the number of words in the description before text preprocessing.

```
data['no_of_words'] = data.description.apply(lambda x : len(x.split()))
```

The following is the word count distribution for 'description'.

```
bins=[0,50,75, np.inf]
data['bins']=pd.cut(data.no_of_words, bins=[0,100,300,500,800, np.inf],
labels=['0-100', '100-200', '200-500','500-800' ,'>800'])
words_distribution = data.groupby('bins').size().reset_index().
rename(columns={0:'word_counts'})
sns.barplot(x='bins', y='word_counts', data=words_distribution).
set_title("Word distribution per bin")
```

Figure 4-6 shows that most descriptions have less than 100 words and 20% have 100 to 200 words.

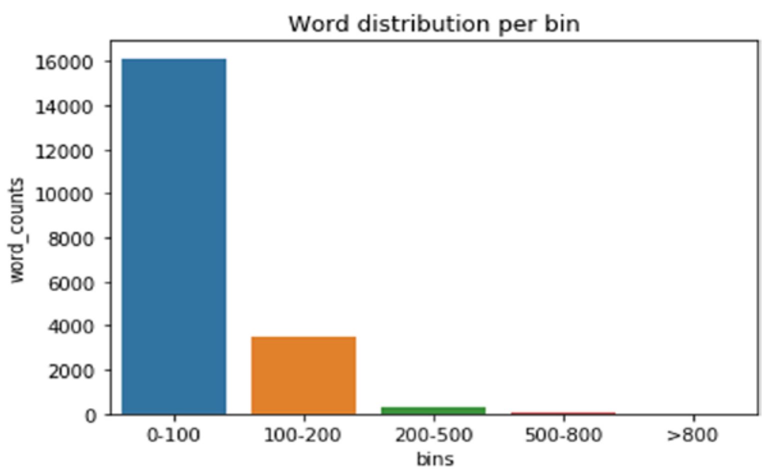


***Figure 4-6.** Word distribution of description column*

Now, let's do some data preprocessing.

# Data Preprocessing

Data preprocessing includes data cleaning, preparation, transformation, and dimensionality reduction, which convert the raw data into a form that is suitable for further processing.

```
# Number of missing values in each column
missing = pd.DataFrame(data.isnull().sum()).rename(columns = {0:
'missing'})

# Create a percentage of missing values
missing['percent'] = missing['missing'] / len(data)

# sorting the values in desending order to see highest count on the top
missing.sort_values('percent', ascending = False)
```

Figure 4-7 shows that nearly 30% of the brand variables have missing values. Other variables have a negligible number of missing values.

| | missing | percent |
|---|---|---|
| brand | 5863 | 0.293179 |
| retail_price | 78 | 0.003900 |
| discounted_price | 78 | 0.003900 |
| product_specifications | 14 | 0.000700 |
| image | 3 | 0.000150 |
| description | 0 | 0.000000 |
| no_of_words | 0 | 0.000000 |
| overall_rating | 0 | 0.000000 |
| product_rating | 0 | 0.000000 |
| uniq_id | 0 | 0.000000 |
| is_FK_Advantage_product | 0 | 0.000000 |
| crawl_timestamp | 0 | 0.000000 |
| pid | 0 | 0.000000 |
| product_category_tree | 0 | 0.000000 |
| product_name | 0 | 0.000000 |
| product_url | 0 | 0.000000 |
| bins | 0 | 0.000000 |

***Figure 4-7.*** *Missing value distribution*

Again, let's move into text preprocessing using multiple methods.

## Text Preprocessing

There is a lot of unwanted information present in the text data. Let's clean it up.

Text preprocessing tasks include

- Converting the text data to lowercase
- Removing/replacing the punctuations
- Removing/replacing the numbers
- Removing extra whitespaces
- Removing stop words
- Stemming and lemmatization

```
# Remove punctuation
data['description'] = data['description'].str.replace(r'[^\w\d\s]', ' ')

# Replace whitespace between terms with a single space
data['description'] = data['description'].str.replace(r'\s+', ' ')

# Remove leading and trailing whitespace
data['description'] = data['description'].str.replace(r'^\s+|\s+?$', '')

# converting to lower case
data['description'] = data['description'].str.lower()

data['description'].head()

0    key features of alisha solid women s cycling s...
1    fabhomedecor fabric double sofa bed finish col...
2    key features of aw bellies sandals wedges heel...
3    key features of alisha solid women s cycling s...
4    specifications of sicons all purpose arnica do...
Name: description, dtype: object

# Removing Stop words
stop = stopwords.words('english')
```

```
pattern = r'\b(?:{})\b'.format('|'.join(stop))
data['description'] = data['description'].str.replace(pattern, '')

# Removing single characters
data['description'] = data['description'].str.replace(r'\s+', ' ')
data['description'] = data['description'].apply(lambda x: " ".join(x for x
in x.split() if len(x)>1))

# Removing domain related stop words from description
specific_stop_words = [ "rs","flipkart","buy","com","free","day","cash","re
placement","guarantee","genuine","key","feature","delivery","products","pro
duct","shipping", "online","india","shop"]
data['description'] = data['description'].apply(lambda x: " ".join(x for x
in x.split() if x not in specific_stop_words))

data['description'].head()

0    features alisha solid women cycling shorts cot...
1    fabhomedecor fabric double sofa bed finish col...
2    features aw bellies sandals wedges heel casual...
3    features alisha solid women cycling shorts cot...
4    specifications sicons purpose arnica dog shamp...
Name: description, dtype: object
```

Let's also see what are the most occurred words in the corpus and understand the data better.

```
#Top frequent words after removing domain related stop words
a = data['description'].str.cat(sep=' ')
words = nltk.tokenize.word_tokenize(a)
word_dist = nltk.FreqDist(words)
word_dist.plot(10,cumulative=False)
print(word_dist.most_common(10))
```

Figure 4-8 shows that data has words like *women*, *price*, and *shirt* appeared commonly in the data because there are a lot of fashion-related items and most of it is for women.

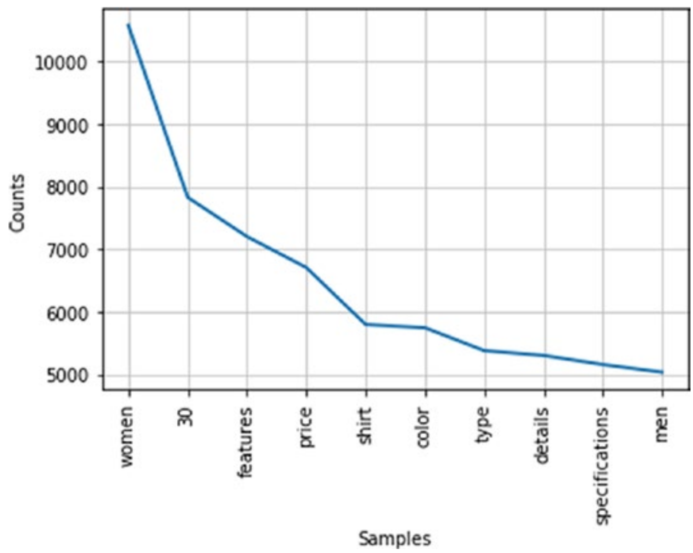


***Figure 4-8.*** *Top frequent words*

# Model Building

So far, we have tried to understand data to build better solutions. Now we need to solve problems using algorithms.

There are two models we want to build.

- A content-based recommendation system
- A product search engine

Let's use different NLP techniques, such as TF-IDF and word embeddings. TF-IDF and word embeddings can be used with both models. From the implementation point of view, both models are almost the same. But the problem each solves is different.

So, let's use the TF-IDF approach to solve with a content-based recommendation system and word embeddings for the search engine. But note that reserve can also be done.

Let's start with the recommendation system.

## Content-based Recommendation System

Now that you know about content-based recommender systems, let's start implementing one.

For content-based systems, let's use the TF-IDF approach.

```
#text cleaning
data['description'] = data['description'].fillna('')

#define the vectorizer
T_vec =  TfidfVectorizer(stop_words='english')

# get the vectors
T_vec_matrix = T_vec.fit_transform(data['description'])

#shape
T_vec_matrix.shape
```

```
(19998, 26162)
```

There are 26,000 unique words in the description.

Next, let's calculate similarity scores for each combination and generate matrix.

A cosine similarity is used in this project. We need to write a function that takes product descriptions as input and lists N most similar items/products.

We also need to do reverse mapping of product names to their indices.

```
# Reversing the map of indices and product

product_index = pd.Series(data.index, index=data['product_name']).drop_
duplicates()
product_index
```

Figure 4-9 shows the output.

```
product_name
Alisha Solid Women's Cycling Shorts                                          0
FabHomeDecor Fabric Double Sofa Bed                                          1
AW Bellies                                                                   2
Alisha Solid Women's Cycling Shorts                                          3
Sicons All Purpose Arnica Dog Shampoo                                        4
Eternal Gandhi Super Series Crystal Paper Weights  with Silver Finish        5
Alisha Solid Women's Cycling Shorts                                          6
FabHomeDecor Fabric Double Sofa Bed                                          7
dilli bazaaar Bellies, Corporate Casuals, Casuals                            8
Alisha Solid Women's Cycling Shorts                                          9
Ladela Bellies                                                              10
Carrel Printed Women's                                                      11
Sicons All Purpose Tea Tree Dog Shampoo                                     12
```

***Figure 4-9.*** *Product names with index*

In the following steps, everything is wrapped under a single function to make testing easier.

1. Obtain the index given the product.
2. Obtain cosine similarity scores.
3. Sort the scores.
4. Get the top N results from the list.
5. Output the product names.

```
# Function that takes in product title as input and outputs the most
similar product

def predict_products(text):

    # getting index
    index = product_index[text]

    # Obtaining the pairwsie similarity scores
    score_matrix = linear_kernel(T_vec_matrix[index], T_vec_matrix)
    matching_sc= list(enumerate(score_matrix[0]))

    # Sort the product based on the similarity scores
    matching_sc= sorted(matching_sc, key=lambda x: x[1], reverse=True)
```

```
    # Getting the scores of the 10 most similar product
    matching_sc= matching_sc[1:10]

    # Getting the product indices
    product_indices = [i[0] for i in matching_sc]

    # Show the similar products
    return data['product_name'].iloc[product_indices]

recommended_product = predict_products(input("Enter a product name: "))
if recommended_product is not None:
    print ("Similar products")
    print("\n")
    for product_name in recommended_product:
        print (product_name)
```

```
Enter a product name: Engage Urge and Urge Combo Set
```

```
Enter a product name: Engage Urge and Urge Combo Set
Similar products

Engage Rush and Urge Combo Set
Engage Urge-Mate Combo Set
Engage Jump and Urge Combo Set
Engage Fuzz and Urge Combo Set
Engage Mate+Urge Combo Set
Engage Urge+Tease Combo Set
Engage Combo Set
Engage Combo Set
Engage Combo Set
```

Let’s look at one more example.

```
Enter a product name: Lee Parke Running Shoes
```

```
Enter a product name: Lee Parke Running Shoes
Similar products
```

```
Lee Parke Walking Shoes
N Five Running Shoes
Knight Ace Kraasa Sports Running Shoes, Cycling Shoes, Walking Shoes
WorldWearFootwear Running Shoes, Walking Shoes
reenak Running Shoes
Chazer Running Shoes
Glacier Running Shoes
Sonaxo Men Running Shoes
ETHICS Running Shoes
```

Observe the results. If a customer clicks Lee Parke Running Shoes, they get recommendations based on any other brand running shoes or Lee Parke's any other products.

- Lee Parke Walking Shoes is there because of the Lee Parke brand.
- The rest of the recommendations are running shoes by a different brand.

You can also add price as a feature and get only products in the price range of the customer's selected product.

This is one version of the recommendation system using NLP. To get better results, you can do the following things.

- A better approach to the content-based recommender system can be applied by creating the user profile (currently not in the scope of the data set).
- Use word embeddings as features.
- Try different distance measures.

That's it. We explored how to build a recommendation system using natural language processing.

Now let's move on to another interesting use case which is a product search engine.

## Product Search Engine

The next problem statement is optimizing the search engine to get better search results. The biggest challenge is that most search engines are string matching and might not perform well in all circumstances.

For example, if the user searches "guy shirt", the search results should have all the results that have men, a boy, and so on. The search should not work based on string match, but the other similar words should also consider.

The best way to solve this problem is *word embeddings*.

Word embeddings are N-dimensional vectors for each word that captures the meaning of the word along with context.

word2vec is one of the methods to construct such an embedding. It uses a shallow neural network to build the embeddings. There are two ways the embeddings can be built: skip-gram and CBOW (common bag-of-words).

The CBOW method takes each word's context as the input and predicts the word corresponding to the context. The input to the network context and passed to the hidden layer with N neurons. Then at the end, the output layer predicts the word using the softmax layer. The hidden layer neuron's weight is considered the vector that captured the meaning and context.

The skip-gram model is the reverse of CBOW. The word is the input, and the network predicts context.

That's the brief theory about word embeddings and how it works. We can build the embeddings or use existing trained ones. It takes a lot of data and resources to build one, and for domains like healthcare, we need to build our own because generalized embeddings won't perform well.

### Implementation

Let's use the pretrained word2vec model on the news data set by Google. The trained model can be imported, and vectors can be obtained for each word. Then, any of the similarity measures can be leveraged to rank the results.

```
#Creating list containing description of each product as sublist
fin=[]
for i in range(len(data['description'])):
    temp=[]
    temp.append(data['description'][i])
```

```
    fin = fin + temp

data1 = data[['product_name','description']]

#import the word2vec

from gensim.models import KeyedVectors
filename = 'C:\\GoogleNews-vectors-negative300.bin'
model = KeyedVectors.load_word2vec_format(filename, binary=True,
limit=50000)

#Preprocessing

def remove_stopwords(text, is_lower_case=False):
    pattern = r'[^a-zA-z0-9\s]'
    text = re.sub(pattern, '', text[0])
    tokens = nltk.word_tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stop]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not
        in stop]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text

# Obtain the embeddings, lets use "300"

def get_embedding(word):
    if word in model.wv.vocab:
        return model[word]
    else:
        return np.zeros(300)
```

For every document, let's take the mean of all the words present in the document.

```
# Obtaining the average vector for all the documents

out_dict =  {}
for sen in fin:
    average_vector = (np.mean(np.array([get_embedding(x) for x in nltk.
    word_tokenize(remove_stopwords(sen))]), axis=0))
    dict = { sen : (average_vector) }
    out_dict.update(dict)

# Get the similarity between the query and documents

def get_sim(query_embedding, average_vector_doc):
    sim = [(1 - scipy.spatial.distance.cosine(query_embedding, average_
    vector_doc))]
    return sim

# Rank all the documents based on the similarity

def Ranked_documents(query):
    global rank
    query_words =  (np.mean(np.array([get_embedding(x) for x in nltk.word_
    tokenize(query.lower())],dtype=float), axis=0))
    rank = []
    for k,v in out_dict.items():
        rank.append((k, get_sim(query_words, v)))
    rank = sorted(rank,key=lambda t: t[1], reverse=True)
    dd =pd.DataFrame(rank,columns=['Desc','score'])
    rankfin = pd.merge(data1,dd,left_on='description',right_on='Desc')
    rankfin = rankfin[['product_name','description','score']]
    print('Ranked Documents :')
    return rankfin

# Call the IR function with a query
query=input("What would you like to search")
Ranked_documents(query)

# output
```

```
What would you like to searchbag
Ranked Documents :
```

| | product_name | description | score |
|---|---|---|---|
| 0 | Alisha Solid Women's Cycling Shorts | key features alisha solid women cycling shorts... | [1.0000000515865854] |
| 1 | FabHomeDecor Fabric Double Sofa Bed | fabhomedecor fabric double sofa bed finish col... | [1.0000000515865854] |
| 2 | AW Bellies | key features aw bellies sandals wedges heel ca... | [1.0000000515865854] |
| 3 | Alisha Solid Women's Cycling Shorts | key features alisha solid women cycling shorts... | [1.0000000515865854] |

***Figure 4-10.*** *Model output*

# Advanced Search Engine Using PyTerrier and Sentence-BERT

Let's few advanced deep learning-based solutions to solve this problem. Figure 4-11 shows the entire framework for this approach.

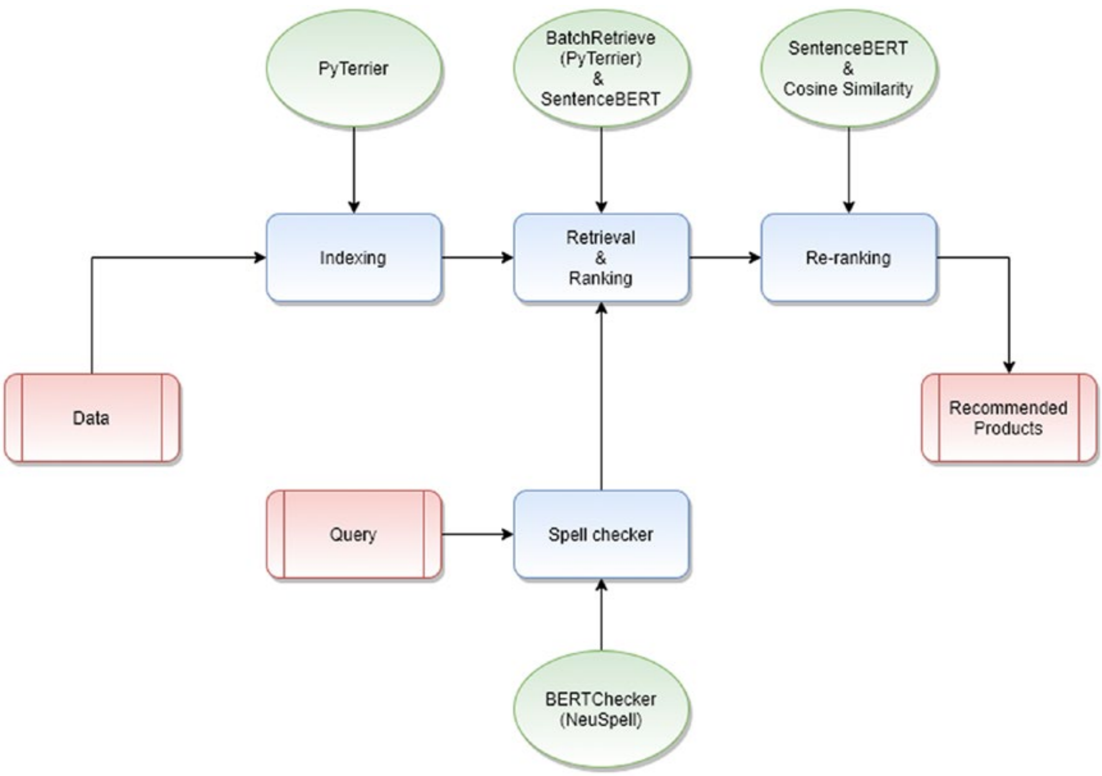


***Figure 4-11.*** *Pipeline of the implementation of the PyTerrier-based search engine*

Indexing is an important part of information retrieval (IR) systems. For indexing, we use DFIndexer. Indexing simplifies the retrieval process.

BatchRetrieve is one of the most widely used PyTerrier objects. It uses a pre-existing Terrier index data structure.

**NeuSpell** is an open source package for correcting spellings based on the context. This package has ten spell-checkers based on various neural models. To implement this model, import the BERTChecker package from NeuSpell.

BERTChecker works for multiple languages, including English, Arabic, Hindi, and Japanese.

Let's use the PyTerrier and Sentence-BERT libraries and proceed with implementation.

The following installs the required packages and libraries.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import string
import re
%matplotlib inline
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
exclude = set(string.punctuation)
import string
```

```
!pip install python-terrier
import pyterrier as pt
if not pt.started():
  pt.init()
```

```
!pip install -U sentence-transformers

!pip install neuspell
!pip install -e neuspell/
!git clone https://github.com/neuspell/neuspell; cd neuspell
import os
os.chdir("/content/neuspell")

!pip install -r /content/neuspell/extras-requirements.txt
!python -m spacy download en_core_web_sm

#Unzipping the multi-linguistic packages
!wget https://storage.googleapis.com/bert_models/2018_11_23/multi_cased_
L-12_H-768_A-12.zip
!unzip *.zip

#importing neuspell
from neuspell import BertChecker
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('sentence-transformers/bert-base-nli-mean-
tokens')
```

Load the data set.

```
df=pd.read_csv(flipkart_com-ecommerce_sample.csv)
df.head()
```

## Data Preprocessing

Let's do some more text preprocessing.

First, make the 'category_tree' column a simple list.

```
df['product_category_tree']=df['product_category_tree'].map(lambda x:x.
strip('[]'))
df['product_category_tree']=df['product_category_tree'].map(lambda x:x.
strip('"'))
df['product_category_tree']=df['product_category_tree'].map(lambda x:x.
split('>>'))
```

Next, drop unwanted columns.

```
df.drop(['crawl_timestamp','product_url','image',"retail_
price","discounted_price","is_FK_Advantage_product","product_
rating","overall_rating","product_specifications"],axis=1,inplace=True)
```

Then, drop duplicate products.

```
uniq_prod=df.copy()
uniq_prod.drop_duplicates(subset ="product_name", keep = "first", inplace =
True)
```

Remove stop words and punctuations and then perform tokenization and lemmatization.

```
def filter_keywords(doc):
    doc=doc.lower()
    stop_free = " ".join([i for i in doc.split() if i not in stop_words])
    punc_free = "".join(ch for ch in stop_free if ch not in exclude)
    word_tokens = word_tokenize(punc_free)
    filtered_sentence = [(lem.lemmatize(w, "v")) for w in word_tokens]
    return filtered_sentence
```

Apply the filter_keywords function to selected columns to obtain the keywords for each column.

```
uniq_prod['product'] = uniq_prod['product_name'].apply(filter_keywords)
uniq_prod['description'] = uniq_prod['description'].astype("str").
apply(filter_keywords)
uniq_prod['brand'] = uniq_prod['brand'].astype("str").apply(filter_
keywords)
```

Combine all the keywords for each product.

```
uniq_prod["keywords"]=uniq_prod['product']+uniq_prod['brand']+ df['product_
category_tree']+uniq_prod['description']
uniq_prod["keywords"] = uniq_prod["keywords"].apply(lambda x: ' '.join(x))
```

Creating a 'docno' column, which gives recommendations.

```
uniq_prod['docno']=uniq_prod['product_name']
```

Drop unwanted columns.

```
uniq_prod.drop(['product','brand','pid','product_
name'],axis=1,inplace=True)

uniq_prod.head()
```

Figure 4-12 shows the final data set that we are using going forward. We perform the query search in 'keywords' and obtain the corresponding products based on the most relevant keywords.

| | uniq_id | product_category_tree | description | keywords | docno |
|---|---|---|---|---|---|
| 0 | c2d766ca982eca8304150849735ffef9 | [Clothing , Women's Clothing , Lingerie, Sle... | [key, feature, alisha, solid, womens, cycle, s... | alisha solid womens cycle short alisha Clothin... | Alisha Solid Women's Cycling Shorts |
| 1 | 7f7036a6d550aaa89d34c77bd39a5e48 | [Furniture , Living Room Furniture , Sofa Be... | [fabhomedecor, fabric, double, sofa, bed, fini... | fabhomedecor fabric double sofa bed fabhomedec... | FabHomeDecor Fabric Double Sofa Bed |
| 2 | f449ec65dcbc041b6ae5e6a32717d01b | [Footwear , Women's Footwear , Ballerinas , ... | [key, feature, aw, belly, sandals, wedge, heel... | aw belly aw Footwear Women's Footwear Ball... | AW Bellies |
| 4 | bc940ea42ee6bef5ac7cea3fb5cfbee7 | [Pet Supplies , Grooming , Skin & Coat Care ... | [specifications, sicons, purpose, arnica, dog,... | sicons purpose arnica dog shampoo sicons Pet S... | Sicons All Purpose Arnica Dog Shampoo |
| 5 | c2a17313954882c1dba461863e98adf2 | [Eternal Gandhi Super Series Crystal Paper Wei... | [key, feature, eternal, gandhi, super, series,... | eternal gandhi super series crystal paper weig... | Eternal Gandhi Super Series Crystal Paper Weig... |

***Figure 4-12.*** *Data set snapshot*

## Building the Search Engine

Let's use the DFIndexer object to create the index for keywords.

```
!rm -rf /content/iter_index_porter
pd_indexer = pt.DFIndexer("/content/pd_index")
indexref = pd_indexer.index(uniq_prod["keywords"], uniq_prod["docno"])
```

Let's implementing the NeuSpell spell checker on the user query and save it to an object.

```
spellcheck = BertChecker()
spellcheck.from_pretrained(
    ckpt_path=f"/content/multi_cased_L-12_H-768_A-12"  # "<folder where the
    model is saved>"
)
```

```
X=input("Search Engine:")
query=spellcheck.correct(X)
print(query
```

```
Search Engine:womns clothing
women clothing
```

Perform ranking and retrieval using PyTerrier and Sentence-BERT.

```
prod_ret = pt.BatchRetrieve(indexref, wmodel='TF_IDF',
properties={'termpipelines': 'Stopwords'})
pr=prod_ret.compile()
output=pr.search(query)
docno=list(output['docno'])
transform=model.encode(docno)
```

Create embeddings and re-ranking using PyTerrier and cosine similarity.

```
embedding={}
for i,product in enumerate(docno):
  embedding[product]=transform[i]
```

```
q_embedding=model.encode(query).reshape(1,-1)
l=[]
for product in embedding.keys():
score=cosine_similarity(q_embedding,embedding[product].reshape(1,-1))[0][0]
l.append([product,score])
```

```
output2=pd.DataFrame(l,columns=['product_name','score'])
```

Let's look at the results.

```
output2.sort_values(by='score',ascending=False).head(10)
```

Figure 4-13 shows "women clothing" as the query. In the output, there is a list of products that are a part of women's clothing. The corresponding scores represent relevance. Note that the results are very relevant to the search query.

| | product_name | score |
|---|---|---|
| 95 | People Women's Dress | 0.862905 |
| 212 | Nasha Women's Gathered Dress | 0.763813 |
| 166 | Kasturi Women's Gathered Dress | 0.754854 |
| 207 | Sbuys Women's Gathered Dress | 0.751761 |
| 111 | Teemoods Women's Camisole | 0.748863 |
| 55 | Viyasha Women's, Girl's Shapewear | 0.747937 |
| 182 | Karishma Women's Gathered Dress | 0.742156 |
| 225 | Kwardrobe Women's Gathered Dress | 0.741498 |
| 200 | Modimania Women's Gathered Dress | 0.740214 |
| 722 | IDK Woman Women's Shift Dress | 0.736790 |

***Figure 4-13.*** *Model output*

# Multilingual Search Engine Using Deep Text Search

One of the challenging tasks in these search engines is the languages. These products are very regional, and English is not the only language spoken. To solve this, we can use Deep Text Search.

Deep Text Search is an AI-based multilingual text search engine with transformers. It supports 50+ languages. The following are some of its features.

- It has a faster search capability.
- It has very accurate recommendations.
- It works best for implementing Python-based applications.

Let's use the following data sets to understand this library.

- The English data set contains 30 rows and 13 columns.

https://data.world/login?next=%2Fpromptcloud%2Fwalmart-product-data-from-usa%2Fworkspace%2Ffile%3Ffilename%3Dwalmart_com-ecommerce_product_details__20190311_20191001_sample.csv

- The Arabic data set is a text file related to the Arabic newspaper corpus.

  https://www.kaggle.com/abedkhooli/arabic-bert-corpus

- The Hindi data set contains 900 movie reviews in three classes (positive, neutral, negative) collected from Hindi news websites.

  https://www.kaggle.com/disisbig/hindi-movie-reviews-dataset?select=train.csv

- The Japanese data set contains the Japanese prime minister's tweets.

  https://www.kaggle.com/team-ai/shinzo-abe-japanese-prime-minister-twitter-nlp

Let's start with the English data set.

Install the required packages and importing libraries.

```
!pip install neuspell
!pip install -e neuspell/
!git clone https://github.com/neuspell/neuspell; cd neuspell
!pip install DeepTextSearch
```

```
import os
os.chdir("/content/neuspell")
```

```
!pip install -r /content/neuspell/extras-requirements.txt
!python -m spacy download en_core_web_sm
```

```
#Unzipping the multi-linguistic packages
!wget https://storage.googleapis.com/bert_models/2018_11_23/multi_cased_L-12_H-768_A-12.zip
!unzip *.zip
```

```
# importing nltk
import nltk
```

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('wordnet')
```

```
#import DeepTextSearch
from DeepTextSearch import TextEmbedder,TextSearch,LoadData
from nltk.corpus import wordnet
import pandas as pd
from neuspell import BertChecker
```

Let's use BERTChecker for spell checking. It also supports multiple languages.

```
spellcheck = BertChecker()
spellcheck.from_pretrained(
    ckpt_path=f"/content/multi_cased_L-12_H-768_A-12")
# "<folder where the model is saved>"
```

Let's input the query and check how its works.

```
X=input("Enter Product Name:")
y=spellcheck.correct(X)
print(y)
```

```
Enter Product Name: shirts
shirts
```

Let's also use POS tagging to select relevant words from the given query.

```
#function to get the POS tag
def preprocess(sent):
    sent = nltk.word_tokenize(sent)
    sent = nltk.pos_tag(sent)
    return sent
```

```
sent = preprocess(y)
l=[]
for i in sent:
  if i[1]=='NNS' or i[1]=='NN':
```

```
    l.append(i[0])
print(l)
```

```
['shirts']
```

In the next step, let's use query expansion to get synonyms of words, so that we can get more relevant recommendations.

```
query=""
for i in l:
  query+=i
  synset = wordnet.synsets(i)
  query+=" "+synset[0].lemmas()[0].name()+" "

print(query)
```

```
shirts shirt
```

We created this dictionary to display the product names as per the recommendations given in the description.

```
#importing the data
df=pd.read_csv("walmart_com-ecommerce_product_details__20190311_20191001_
sample.csv")

df1=df.set_index("Description", inplace = False)

df2=df1.to_dict()
dict1=df2['Product Name']
```

Embed the data in a pickle file as the library requires it to be in that format.

```
data = LoadData().from_csv("walmart_com-ecommerce_product_
details__20190311_20191001_sample.csv")
TextEmbedder().embed(corpus_list=data)
corpus_embedding = TextEmbedder().load_embedding()
```

Search the ten most relevant products based on the query.

```
n=10
t=TextSearch().find_similar(query_text=query,top_n=n)
```

```
for i in range(n):
  t[i]['text']=dict1[t[i]['text']]
  print(t[i])
```

Figure 4-14 shows the results for the "shirts" search query.

```
{'index': 19, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 18, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 17, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 16, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 15, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 13, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 12, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 11, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 20, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
{'index': 10, 'text': "Men's Big & Tall Harbor Bay Space-Dye Piqué Polo Shirt", 'score': 0.34476066}
```

***Figure 4-14.*** *Model output*

Now let's see how the search works in Arabic.

```
X1=input("Search Engine:")
y1=spellcheck.correct(X1)
print(y1)
```

```
Search Engine:صاحب
صاحب
```

Let's import the Arabic data corpus to perform the search.

```
# import library and data
from DeepTextSearch import LoadData

data1 = LoadData().from_text("wiki_books_test_1.txt")
TextEmbedder().embed(corpus_list=data1)
corpus_embedding = TextEmbedder().load_embedding()
```

```
Embedding data Saved Successfully Again!
['corpus_embeddings_data.pickle', 'corpus_list_data.pickle']
Embedding data Loaded Successfully!
['corpus_embeddings_data.pickle', 'corpus_list_data.pickle']
```

Let's find the top 10 relevant documents using the textseach function. Figure 4-15 shows the output.

```
TextSearch().find_similar(query_text=y1,top_n=10)
```

```
[{'index': 383, 'score': 0.79338586, 'text': 'والثاني: تجب فيهما.'},
 {'index': 273, 'score': 0.74463975, 'text': 'والثاني: بالثاني.'},
 {'index': 433, 'score': 0.7429859, 'text': 'والثاني: على القولين.'},
 {'index': 32,
  'score': 0.7291739,
  'text': 'قلت: الثاني أصح، وصححه الاكثرون والله أعلم.'},
 {'index': 242, 'score': 0.7224536, 'text': 'وعلى الثاني: بعذر.'},
 {'index': 460, 'score': 0.71352804, 'text': 'والثاني: استحبابه.'},
 {'index': 446, 'score': 0.71352804, 'text': 'والثاني: استحبابه.'},
 {'index': 8885, 'score': 0.7128246, 'text': '. ومن وجه بالقسم الثاني'},
 {'index': 205, 'score': 0.68822443, 'text': 'والثاني: لا.'},
 {'index': 4641, 'score': 0.68177044, 'text': 'ولواحق خبر ثالث.'}]
```

***Figure 4-15.*** *Model output*

Now let's see how the search works in Hindi.

```
X_hindi=input("Search Engine:")
y_hindi=spellcheck.correct(X_hindi)
print(y_hindi)
```

```
Search Engine:निर्देशक
निर्देशक
```

```
#loading the Hindi data corpus
data_hindi = LoadData().from_csv("hindi.csv")
TextEmbedder().embed(corpus_list=data_hindi)
corpus_embedding = TextEmbedder().load_embedding()
```

Let's find the top 10 relevant results. Figure 4-16 show the output.

```
TextSearch().find_similar(query_text=y_hindi,top_n=10)
```

```
[{'index': 410,
  'score': 0.67329043,
  'text': 'बेनर :\nयूटीवी मोशन पिक्चर्स\n\nनिर्माता :\nरॉनी स्क्रूवाला\n\nनिर्देशन एवं संगीत :\nविशाल भारद्वाज\n\nगीत :\nगुलजार\n\nकलाकार :\nशाहिद कपूर, प्रियंका चोपड़ा, अम
 {'index': 651,
  'score': 0.6686705,
  'text': 'बेनर :\nहरी ओम एंटरटेनमेंट कं., थ्रीज़ कंपनी, यूटीवी मोशन पिक्चर्स\n\nनिर्माता :\nफराह खान, अक्षय कुमार, शिरीष कुंदर\n\nनिर्देशक :\nशिरीष कुंदर\n\nसंगीत :\nगौर
 {'index': 463,
  'score': 0.66178524,
  'text': '\n\nश्री-ड\nी\nबेनर :\nयूटीवी स्पॉट बॉय\n\nनिर्माता :\nसिद्धार्थ रॉय कपूर, रॉनी स्क्रूवाला\n\nनिर्देशक :\nरेमो डिसूजा\n\nसंगीत :\nसचिन जिगर\n\nकलाकार :\nप्रभुदेवा,
 {'index': 39,
  'score': 0.6527176,
  'text': 'बेनर :\nभंडारकर एंटरटेनमेंट, वाइड फ्रेम पिक्चर्स\n\nनिर्माता :\nकुमार मंगत पाठक, मधुर भंडारकर\n\nनिर्देशक :\nमधुर भंडारकर\n\nसंगीत :\nप्रीतम चक्रवर्ती\n\nकलाका
 {'index': 23,
  'score': 0.6512797,
  'text': 'बेनर :\nपीवीआर पिक्चर्स\n\nनिर्माता :\nअजय बिजली, दिबाकर बेनर्जी, प्रिया श्रीधरन, संजीव के. बिजली\n\nनिर्देशक :\nदिबाकर बेनर्जी\n\nसंगीत :\nविशाल-शेखर\n\nकल
 {'index': 359,
  'score': 0.6375189,
  'text': 'कुल मिलाकर कहा जा सकता है कि 'हरि पुत्तर' समय और पैसे की बर्बादी है।\n\nनिर्माता :\nलकी कोहली, मुनीष पुरी, ए पी पारिगी\n\nनिर्देशक :\nलकी कोहली, राजेश बज
 {'index': 84,
  'score': 0.63575315,
  'text': 'बेनर :\nसरोज एंटरटेनमेंट प्रा.लि.\n\nनिर्माता :\nरचना सुनील सिंह\n\nनिर्देशक :\nपार्थो घोष\n\nकलाकार :\nजैकी श्रॉफ, मनीषा कोइराला, निकिता आनंद, रोजा\n\n\n\n
 {'index': 151,
  'score': 0.62818766,
  'text': 'बेनर :\nवाय फिल्म्स\n\nनिर्माता :\nआशीष पाटिल\n\nनिर्देशक :\nबम्पी\n\nसंगीत :\nराम सम्पत\n\nकलाकार :\nश्रद्धा कपूर, ताहा शाह, शेनाज ट्रेजरीवाला, जन्नत जुबैर र
 {'index': 121,
  'score': 0.62714005,
  'text': 'बेनर :\nयशराज फिल्म्स\n\nनिर्माता :\nआदित्य चोपड़ा\n\nनिर्देशक :\nमनीष शर्मा\n\nसंगीत :\nसलीम मर्चेंट-सुलेमान मर्चेंट\n\nकलाकार :\nरणवीर सिंह, अनुष्का शर्मा, पा
 {'index': 10,
  'score': 0.62641734,
  'text': "IFM \r\t\t\t\t\t\t\t\t\t\tIFM \r\t\t\t\t\t\t\t\t\t\t \nIFM \r\t\t\t\t\t\t\t\t\t\tIFM \r\t\t\t\t\t\t\t\t\t\t \nनिर्माता :\nआदित्य चोपड़ा\nनिर्देशक :\n
```

***Figure 4-16.*** *Model output*

Let's try one more language, Japanese.

```
X_japanese=input("Search Engine:")
# y_japanese=spellcheck.correct(X_japanese)
print(X_japanese)
```

```
Search Engine:経済的
経済的
```

```
#loading the data
data_japanese = LoadData().from_text("Japanes_Shinzo Abe Tweet 20171024 - Tweet.csv")
TextEmbedder().embed(corpus_list=data_chinese)
```

```
corpus_embedding = TextEmbedder().load_embedding()
```

Find the top ten tweets based on the search query. Figure 4-17 shows the output.

```
TextSearch().find_similar(query_text=X_japanese,top_n=10)
```


***Figure 4-17.*** *Model output*

# Summary

We implemented a search engine and recommendation systems using various models in this chapter. We started with a simple recommender system using the TF-IDF approach to find the similarity score for all the products description. Based on the description, products are ranked and shown to the users. Later, we explored how to build a simple search engine using word embeddings and ranked the results.

Then, we jumped into advanced models like PyTerrier and Sentence-BERT, where pretrained models extract the vectors. Since these models are deep learning-based, results are a lot better when compared to traditional approaches. We also used Deep Text Search, another deep learning library that works for multilanguage text corpus.