**Amazon Product Recommendation**

* dataset
* Project Analysis Report
* Recommendation system

**Data Overview**

The dataset used in this project contains information about Amazon products, including product titles, descriptions, and categories. The dataset was preprocessed by applying stemming to the text data and then vectorizing it using TF-IDF to calculate the similarity between products. The dataset contains 10,000 records.

***Report:***

Introduction:

The Amazon Product Recommendation project aims to recommend products to users based on their search queries. The project uses natural language processing techniques to preprocess and tokenize text data, followed by cosine similarity to calculate the similarity between user queries and product descriptions. The output is a list of recommended products sorted by similarity.

Data:

The dataset used in this project is a CSV file containing product information, such as title, category, description, and price. The dataset has a total of 9,434 records and 4 features.

Data Cleaning and Preprocessing:

The following steps were taken to clean and preprocess the data:

Removed the 'id' column as it was not necessary for the project

Tokenized and stemmed the text data using NLTK's SnowballStemmer

Calculated the TF-IDF matrix and cosine similarity between user queries and product descriptions

Product Recommendation:

The recommendation engine uses cosine similarity to calculate the similarity between user queries and product descriptions. The top 10 most similar products are recommended to the user.

Conclusion:

The Amazon Product Recommendation project is a simple yet effective way to recommend products to users based on their search queries. The project uses natural language processing techniques to preprocess and tokenize text data, followed by cosine similarity to calculate the similarity between user queries and product descriptions. The output is a list of recommended products sorted by similarity. With further optimization and tuning, this recommendation engine could be deployed to a live environment to help users find products they are interested in.

**Recommendation system:**

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. In the context of recommendation systems, it is commonly used to find the similarity between two items, and therefore recommend items to users based on their similarity scores.

In the case of our product recommendation system, we are using cosine similarity to find the similarity between a user's query and all the products in our dataset. To do this, we first preprocess the text data by tokenizing and stemming the words, and then we use the TF-IDF vectorization technique to convert the text data into numerical vectors. We then calculate the cosine similarity between the user's query and each product in our dataset using the cosine\_similarity function from the scikit-learn library.

The cosine similarity score ranges from -1 to 1, where -1 means that the two items are completely dissimilar, 0 means that they are orthogonal or have no similarity, and 1 means that they are identical or have maximum similarity. In our case, we sort the products based on their similarity scores in descending order and recommend the top 10 products with the highest scores to the user.

This approach is effective for recommendation systems because it takes into account the overall content of the items being compared and not just certain features. Therefore, it can recommend items that are similar in terms of overall content, but may not have identical features.

**Note Book Code:**

**import** numpy **as** np  
**import** pandas **as** pd  
**import** nltk

**load data**

[38]

amazon = pd.read\_csv("amazon\_product.csv")

[39]

amazon.head()

**necessary Steps**

[40]

amazon.drop('id',axis=1,inplace=True)

[41]

amazon

[42]

amazon.shape

(668, 3)

[43]

amazon.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 668 entries, 0 to 667  
Data columns (total 3 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Title 668 non-null object  
 1 Description 668 non-null object  
 2 Category 668 non-null object  
dtypes: object(3)  
memory usage: 15.8+ KB

**Cleaning Dataset**

[44]

**from** nltk.stem.snowball **import** SnowballStemmer  
stemer = SnowballStemmer('english')

[45]

**def** tokenize\_stem(text):  
    tokens = nltk.word\_tokenize(text.lower())  
    stemming = [stemer.stem(w) **for** w **in** tokens]  
      
    **return** " ".join(stemming)  
amazon['stemmed\_tokens'] = amazon.apply(**lambda** row: tokenize\_stem(row['Title'] + ' ' + row['Description']), axis=1)

**cosine similarity between two documents (in this case, product titles and descriptions)**

[46]

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer  
**from** sklearn.metrics.pairwise **import** cosine\_similarity  
  
tfvector = TfidfVectorizer(tokenizer=tokenize\_stem)  
  
**def** cos\_sim(txt1,txt2):  
    tfidmatrix = tfvector.fit\_transform([txt1,txt2])  
    similar\_vectors = cosine\_similarity(tfidmatrix)[0][1]  
    **return** similar\_vectors

**Recommend Products**

[58]

**def** recommend\_product(query):  
    tokenized\_query = tokenize\_stem(query)  
    amazon['similarity'] = amazon['stemmed\_tokens'].apply(**lambda** x: cos\_sim(tokenized\_query, x))  
    final\_df = amazon.sort\_values(by=['similarity'], ascending=False).head(10)[['Title', 'Description', 'Category']]  
    **return** final\_df

[59]

final\_df = recommend\_product(' PURELL ES8 Professional HEALTHY SOAP Foam Refill, Fresh Scent Fragrance, 1200 mL Soap Refill for PURELL ES8 Touch-Free Dispenser (Pack of 2) - 7777-02 ')

[61]

final\_df

[60]

**import** pickle   
pickle.dump(final\_df,open("final\_df.pkl",'wb'))

**Drive Code**

**import** numpy **as** np  
**import** pandas **as** pd  
**import** nltk  
  
amazon = pd.read\_csv("amazon\_product.csv")  
  
amazon.drop('id',axis=1,inplace=True)  
  
**from** nltk.stem.snowball **import** SnowballStemmer  
stemer = SnowballStemmer('english')  
  
**def** tokenize\_stem(text):  
    tokens = nltk.word\_tokenize(text.lower())  
    stemming = [stemer.stem(w) **for** w **in** tokens]  
      
    **return** " ".join(stemming)  
amazon['stemmed\_tokens'] = amazon.apply(**lambda** row: tokenize\_stem(row['Title'] + ' ' + row['Description']), axis=1)  
  
  
  
**from** sklearn.feature\_extraction.text **import** TfidfVectorizer  
**from** sklearn.metrics.pairwise **import** cosine\_similarity  
  
tfvector = TfidfVectorizer(tokenizer=tokenize\_stem)  
  
**def** cos\_sim(txt1,txt2):  
    tfidmatrix = tfvector.fit\_transform([txt1,txt2])  
    similar\_vectors = cosine\_similarity(tfidmatrix)[0][1]  
    **return** similar\_vectors  
  
  
**def** recommend\_product(query):  
    tokenized\_query = tokenize\_stem(query)  
    amazon['similarity'] = amazon['stemmed\_tokens'].apply(**lambda** x: cos\_sim(tokenized\_query, x))  
    final\_df = amazon.sort\_values(by=['similarity'], ascending=False).head(10)[['Title', 'Description', 'Category']]  
    **return** final\_df  
  
final\_df = recommend\_product(' PURELL ES8 Professional HEALTHY SOAP Foam Refill, Fresh Scent Fragrance, 1200 mL Soap Refill for PURELL ES8

**Flask Code:**

from flask import Flask**,** render\_template**,** request  
import pandas as pd  
import nltk  
import pickle  
  
app = Flask(\_\_name\_\_)  
  
# Load dataset  
amazon = pd.read\_csv("amazon\_product.csv")  
  
amazon.drop('id'**,** axis=**1,** inplace=True)  
  
from nltk.stem.snowball import SnowballStemmer  
  
stemer = SnowballStemmer('english')  
  
  
def tokenize\_stem(text):  
 tokens = nltk.word\_tokenize(text.lower())  
 stemming = [stemer.stem(w) for w in tokens]  
  
 return " ".join(stemming)  
  
  
amazon['stemmed\_tokens'] = amazon.apply(lambda row: tokenize\_stem(row['Title'] + ' ' + row['Description'])**,** axis=**1**)  
  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import cosine\_similarity  
  
tfvector = TfidfVectorizer(tokenizer=tokenize\_stem)  
  
  
# functinos=========================  
def tokenize\_stem(text):  
 tokens = nltk.word\_tokenize(text.lower())  
 stemming = [stemer.stem(w) for w in tokens]  
  
 return " ".join(stemming)  
  
  
amazon['stemmed\_tokens'] = amazon.apply(lambda row: tokenize\_stem(row['Title'] + ' ' + row['Description'])**,** axis=**1**)  
  
def cos\_sim(txt1**,** txt2):  
 tfidmatrix = tfvector.fit\_transform([txt1**,** txt2])  
 similar\_vectors = cosine\_similarity(tfidmatrix)[**0**][**1**]  
 return similar\_vectors  
  
  
def recommend\_product(query):  
 tokenized\_query = tokenize\_stem(query)  
 amazon['similarity'] = amazon['stemmed\_tokens'].apply(lambda x: cos\_sim(tokenized\_query**,** x))  
 final\_df = amazon.sort\_values(by=['similarity']**,** ascending=False).head(**10**)[['Title'**,** 'Description'**,** 'Category']]  
 return final\_df  
  
  
  
# create app  
@app.route("/")  
def index():  
 return render\_template('index.html')  
  
@app.route('/predict'**,**methods=['POST'])  
def predict():  
 query = request.form['query']  
  
 df = recommend\_product(query)  
 print(df.to\_string())  
 return render\_template('index.html'**,**mydf=df)  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

**Front End Code:**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Product Recommendation</title>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <meta http-equiv="X-UA-Compatible" content="ie=edge">  
 <title>Abalone Age Prediction</title>  
  
 <!-- Bootstrap CSS -->  
 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"  
 integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"  
 crossorigin="anonymous">  
</head>  
<body style="background:black;color:white">  
  
<!--===========================================-->  
<div class="container my-3 mt-3">  
 <h1 class="text-center my-4">Amazon Products Recommendation</h1>  
 <form action="/predict" method="post">  
 <div class="form-group">  
 <label for="text">Enter Product name</label>  
 <input type="text" class="form-control" id="query" name="query">  
 </div>  
 <button type="submit" class="btn btn-primary">get recommendation</button>  
 </form>  
  
  
<div class="row my-3 mt-3">  
 {% for product in mydf.itertuples() %}  
 <div class="col-md-4">  
 <div class="card mb-4">  
 <div class="card-body" style="color:black">  
 <img src="{{ url\_for('static',filename='img.jpg') }}" class="card-img-top" alt="{{ product['Title'] }}">  
 <h5 class="card-title">Title: {{ product['Title'] }}</h5>  
 <p class="card-text">Category: {{ product['Category'] }}</p>  
 <p class="card-text">Description: {{ product['Description'] }}</p>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
</div>  
  
  
  
  
</div>  
</body>  
</html>