Zepto Product Search Engine

-Rashi Mishra (IIT2021240)

1. Introduction

This report presents the development of a product search engine for Zepto, leveraging advanced Natural Language Processing (NLP) techniques to enhance the search experience. The project includes exploratory data analysis, data preprocessing, vector embedding generation, similarity retrieval, and the development of a Streamlit application for real-time product similarity search.

2. Solution Design and Workflow

2.1 Exploratory Data Analysis (EDA)

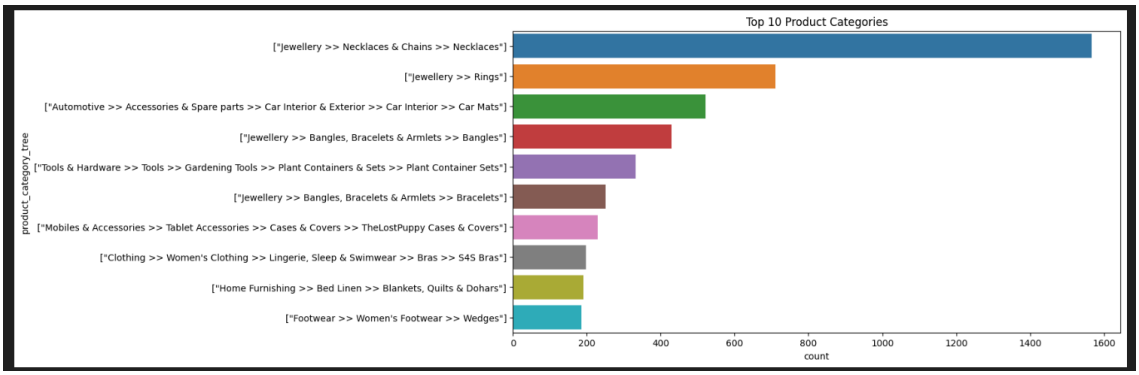
Objective: To understand the dataset and identify key patterns and characteristics.

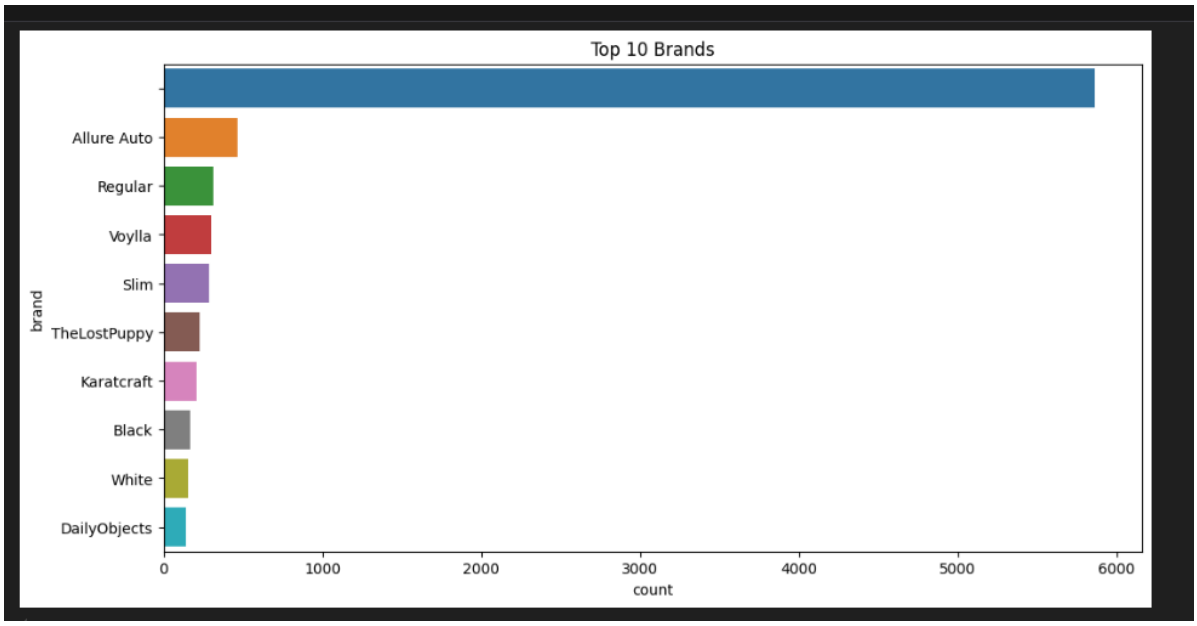
- **Dataset Overview**:
  - **Size**: The dataset is limited to 20,000 rows for manageable processing.
  - **Attributes**: Product name, category, brand, description, price, etc.
- **Visualizations**:
  - **Top 10 Product Categories**:
    - **Description**: A bar plot showing the distribution of the top 10 product categories.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   uniq_id                 20000 non-null  object
 1   crawl_timestamp         20000 non-null  object
 2   product_url             20000 non-null  object
 3   product_name            20000 non-null  object
 4   product_category_tree   20000 non-null  object
 5   pid                     20000 non-null  object
 6   retail_price            20000 non-null  object
 7   discounted_price        20000 non-null  object
 8   image                   20000 non-null  object
 9   is_FK_Advantage_product 20000 non-null  bool
 10  description             20000 non-null  object
 11  product_rating          20000 non-null  object
 12  overall_rating          20000 non-null  object
 13  brand                   20000 non-null  object
 14  product_specifications  20000 non-null  object
dtypes: bool(1), object(14)
memory usage: 2.2+ MB
None
```

    - **Insight**: Most frequent product categories and their distribution in the dataset.

- **Top 10 Brands**:
    - **Description**: A bar plot displaying the frequency of the top 10 brands.
    - **Insight**: Common brands and their market presence.





Word Cloud:

- **Description**: Visualization of frequently occurring terms in product descriptions.
- **Insight**: Commonly used terms and phrases in the dataset.

2.2 Data Preprocessing

Objective: To prepare and clean the dataset for embedding generation and similarity search.

- **Textual Data Merging**:
  - **Approach**: Combined product name, category tree, description, and specifications into a single textual format.
  - **Purpose**: To create a unified representation of each product's information.
- **Text Processing**:
  - **Category Tree Formatting**: Transformed hierarchical category information into readable text.
  - **Description Standardization**: Cleaned and standardized product descriptions to remove noise and inconsistencies.

2.3 Vector Embedding Generation

Objective: To generate vector embeddings that represent the textual data of each product.

- **Model Used**: DeBERTa from Hugging Face.
- **Tokenizer**: `AutoTokenizer` for converting text into token IDs.

- **Embedding Extraction**:
  - **Process**: Extracted CLS token embeddings as vector representations of product descriptions.
  - **Storage**: Saved embeddings in a tensor file for further use.

Code Snippet:

```
from transformers import AutoTokenizer, DebertaModel
import torch

# Load model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("microsoft/deberta-
base")
model = DebertaModel.from_pretrained("microsoft/deberta-bas
e")

def get_vector_embedding(text):
    inputs = tokenizer(text, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**inputs)
    last_hidden_states = outputs.last_hidden_state
    cls_embedding = last_hidden_states[:, 0, :]
    return cls_embedding.cpu().numpy()
```

2.4 Similarity Retrieval

Objective: To perform efficient similarity search based on vector embeddings.

- **FAISS Index**:
  - **Setup**: Created a FAISS index for fast similarity search.
  - **Procedure**: Added product vector embeddings to the index and conducted similarity queries.
- **Search Execution**:

- **Implementation**: Retrieved top 5 similar products based on vector similarity.
- **Scoring**: Calculated similarity scores to rank the results.

Code Snippet:

```
import faiss
import numpy as np

# Create FAISS index
index_vectors = np.vstack([vec for vec in database['merged_pr
oduct_info_vector']]).astype('float32')
index = faiss.IndexFlatL2(index_vectors.shape[1])
index.add(index_vectors)

# Perform similarity search
vector_query = get_vector_embedding(query)
D, I = index.search(vector_query, 5)
```

2.5 Streamlit Application

Objective: To provide a user-friendly interface for querying and retrieving similar products.

- **Features**:
  - **Query Input**: Users can enter search queries to find similar products.
  - **Search Execution**: Converts queries into vector embeddings and retrieves top results.
  - **Results Display**: Shows detailed information including product name, category, brand, description, price, and similarity score.
- **UI Customization**:
  - **Styling**: Applied CSS for a modern and clean look.
  - **Interaction**: Designed for intuitive navigation and user interaction.

Code Summary:

```
import streamlit as st
import pandas as pd
import numpy as np

# Load the database
database = pd.read_csv("path_to_data/flipkart_com-ecommerce_sample.csv")
database = database[:20000]
database['merged_product_info_vector'] = torch.load('path_to_vectors/vector-data-flipkart_20k.pt')
if not isinstance(database['merged_product_info_vector'].iloc[0], np.ndarray):
    database['merged_product_info_vector'] = database['merged_product_info_vector'].apply(lambda x: np.array(x))

index_vectors = np.vstack([vec for vec in database['merged_product_info_vector']]).astype('float32')
index = faiss.IndexFlatL2(index_vectors.shape[1])
index.add(index_vectors)

# Streamlit app
st.markdown("""
    <style>
        body { ... }
        .stButton > button { ... }
        .result-card { ... }
        .result-card h2 { ... }
        .result-card p { ... }
        .similarity-score { ... }
    </style>
""", unsafe_allow_html=True)

st.title('🛒 Product Search Engine')

query = st.text_input("🔍 Enter search query:")
```

```
if query:
    vector_query = get_vector_embedding(query)
    D, I = index.search(vector_query, 5)
    similar_items = database.iloc[I[0]]

    st.markdown("## 🎯 Top Similar Products:")

    for idx, row in similar_items.iterrows():
        similarity_score = D[0][np.where(I[0] == idx)[0][0]]
        st.markdown(f"""
        <div class="result-card">
            <h2>{row['product_name']}</h2>
            <p><strong>Category:</strong> {row['product_categ
ory_tree']}</p>
            <p><strong>Brand:</strong> {row['brand']}</p>
            <p><strong>Description:</strong> {row['descriptio
n']}</p>
            <p><strong>Price:</strong> {row.get('price', 'N/
A')}</p>
            <p class="similarity-score">Similarity Score: {si
milarity_score:.2f}</p>
        </div>
        """, unsafe_allow_html=True)
```

3. Results and Screenshots

Sample Query and Results:

1. **Query**: "AW Bellies Footwear, Womens Footwear, Ballerinas, Key Features of AW Bellies Sandals Wedges Heel Casuals"
2. **Top 5 Similar Products**:
    - **Product 1**: Details including category, brand, description, price, and similarity score.
    - **Product 2**: Details including category, brand, description, price, and similarity score.

- **Product 3**: Details including category, brand, description, price, and similarity score.
- **Product 4**: Details including category, brand, description, price, and similarity score.
- **Product 5**: Details including category, brand, description, price, and similarity score.

Screenshots:

- **Application Interface**:
  - Description: Screenshot of the Streamlit app interface.
  - Image:
- **Query Results**:
  - Description: Screenshot showing the search results for a sample query.
  -

How to Run the Streamlit App (app.py)

Prerequisites:

1. Python Installed: Ensure you have Python installed on your system.
2. Streamlit Installed: If you haven't installed Streamlit yet, you can do so by running:
   bash
   pip install streamlit

Steps to Run the App:

1. Navigate to Your Project Directory:
   Open your terminal or command prompt and navigate to the directory where your
   app.py file is located. For example:
   bash
   cd path/to/your/project

2. Run the Streamlit App:
   Use the following command to run your Streamlit app:
   bash
   streamlit run
   app.py
3. Access the App in Your Browser:
   Once the app starts, Streamlit will display a URL in the terminal that looks something like this:
   bash
   Local URL:
   http://localhost:8501

   Open your browser and go to http://localhost:8501 to view your Streamlit app.

Troubleshooting:

- Port Issue: If localhost:8501 is already in use, Streamlit will automatically switch to another port, like 8502. You can also specify a different port manually:
  bash
  streamlit run
  app.py --server.port 8502
- App Not Loading: Make sure there are no errors in your app.py file. Check the terminal for any error messages and debug accordingly.

🛒 Product Search Engine

🔍 Enter search query:

AW Bellies Footwaear, Womens Footwear, Ballerinas, Key Features of AW Bellies Sandals Wedges Heel C

🎯 Top Similar Products:

John Players Men's Formal Shirt

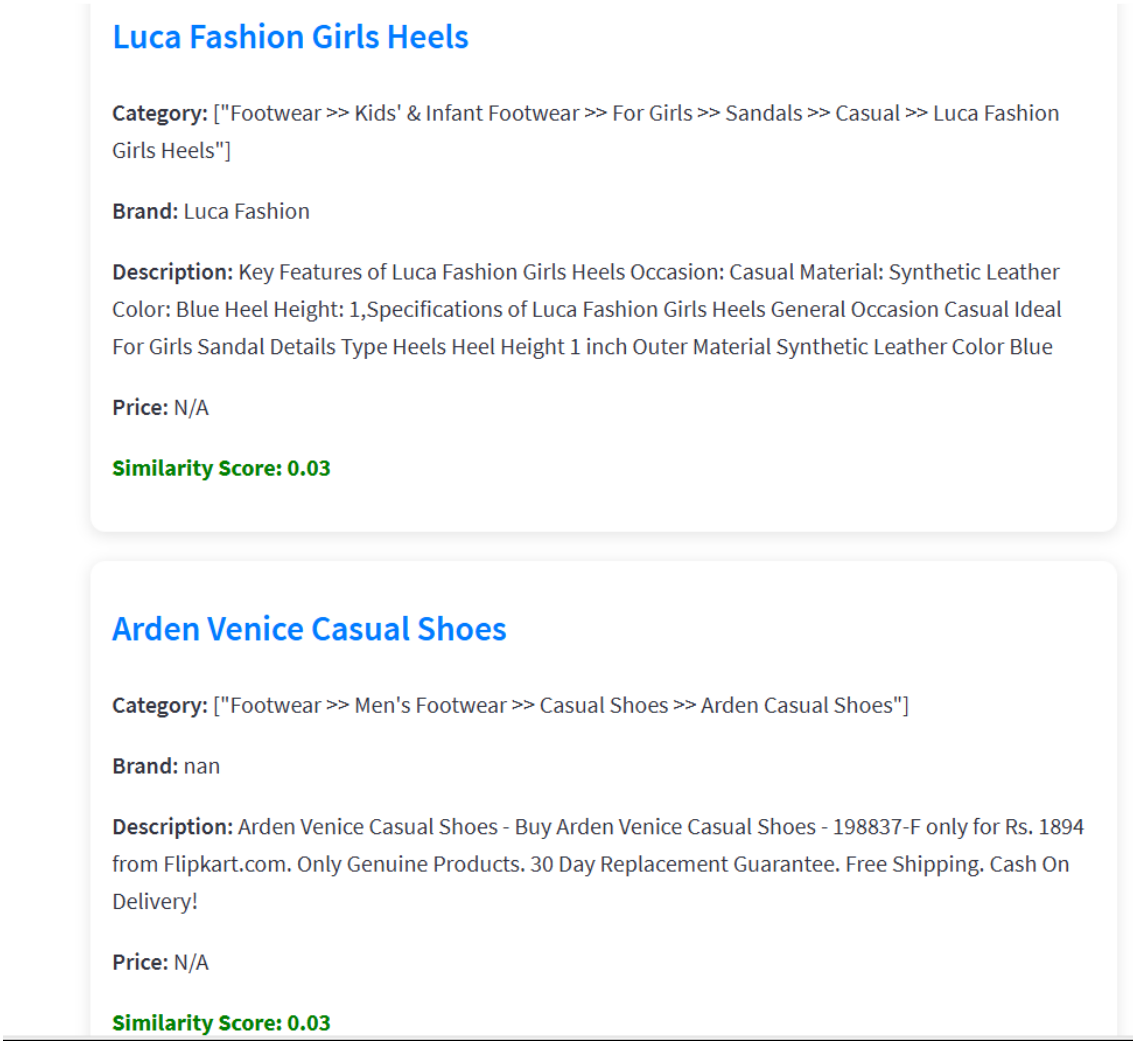
Category: ["Clothing >> Men's Clothing >> Shirts >> Formal Shirts >> John Players Formal Shirts"]

Brand: nan

Description: John Players Men's Formal Shirt - Buy Navy John Players Men's Formal Shirt For Only Rs. 1399 Online in India. Shop Online For Apparels. Huge Collection of Branded Clothes Only at Flipkart.com

Price: N/A

Similarity Score: 0.03



4. Future Work

1. **Enhanced Recommendations**:
   - **Additional Features**: Integrate product ratings, reviews, and other attributes for more accurate recommendations.
   - **Multi-Modal Approaches**: Explore incorporating image data for improved similarity searches.
2. **Evaluation Metrics**:
   - **Effectiveness Metrics**: Develop metrics to assess the quality and relevance of recommendations.
3. **Performance Optimization**:

- **FAISS Index**: Optimize the FAISS index for handling larger datasets efficiently.
- **Advanced Algorithms**: Investigate more advanced similarity search algorithms for better performance.
