Project Report: Amazon Product Recommender

1. Executive Summary

This document presents a product recommendation system built using content-based filtering techniques. It helps users find similar Amazon products based on their features like name, description, and category. The system is implemented as an interactive web application using Streamlit.

2. Introduction

Recommendation systems are essential for enhancing user experience in e-commerce. This project utilizes text-based data and TF-IDF vectorization to identify and recommend similar items.

3. System Architecture

The application consists of the following components:

- - Data preprocessing
- - Feature engineering with TF-IDF
- - Similarity computation using cosine similarity
- - Streamlit-based user interface

4. Dataset Details

The dataset contains Amazon product information with the following fields:
- product_name
- about_product
- category
- product_link
Missing or null values are handled by replacing them with empty strings during preprocessing.

5. Methodology

The recommender uses a content-based filtering method:
1. Combine relevant textual fields.
2. Apply TF-IDF vectorization to create numerical features.

3. Compute cosine similarity between product vectors.
4. Retrieve top N most similar products based on similarity scores.

6. Key Features

- - Real-time search and selection of products.
- - Displays similarity score for transparency.
- - Provides clickable links to product pages.

7. Installation & Execution

To run the application:
1. Install dependencies using pip:
   pip install streamlit pandas scikit-learn
2. Launch the app:
   streamlit run recommender_app.py
3. Ensure the 'amazon.csv' file is in the project directory.

8. Performance Optimization

Streamlit caching is used to avoid recomputation:
- Data loading with @st.cache_data
- Model and similarity matrix with @st.cache_resource

9. Future Work

- - Integrate product images and reviews.
- - Add advanced filtering (e.g., by price, brand).
- - Incorporate collaborative filtering for hybrid recommendations.

10. Code and comments

aiproject.py > ...

```python
import streamlit as st
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from collections import defaultdict

# Load dataset
@st.cache_data
def load_data():
    df = pd.read_csv("amazon.csv")
    df['combined_features'] = (
        df['product_name'].fillna('') + ' ' +
        df['about_product'].fillna('') + ' ' +
        df['category'].fillna('')
    )
    return df

# Compute TF-IDF and similarity matrix
@st.cache_resource
def compute_similarity(df):
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(df['combined_features'])
    cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

    product_name_to_indices = defaultdict(list)
    for idx, name in enumerate(df['product_name']):
        product_name_to_indices[name].append(idx)

    return cosine_sim, product_name_to_indices

# Recommendation function
def get_recommendations(product_name, df, cosine_sim, product_name_to_indices, top_n=5):
    indices = product_name_to_indices.get(product_name)
    if not indices:
```

```python
    indices = product_name_to_indices.get(product_name)
    if not indices:
        return []

    idx = indices[0]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = [score for score in sim_scores if score[0] != idx]
    sim_scores = sim_scores[:top_n]

    recommended_indices = [i[0] for i in sim_scores]
    similarities = [score[1] for score in sim_scores]
    avg_similarity = sum(similarities) / len(similarities)

    return df[['product_name', 'product_link']].iloc[recommended_indices], avg_similarity

# Streamlit UI
st.set_page_config(page_title="E-Commerce Product Recommender", layout="wide")
st.title("🛍 Amazon Product Recommender")

df = load_data()
cosine_sim, product_name_to_indices = compute_similarity(df)

# Search bar + dropdown
product_list = sorted(df['product_name'].unique())
search_query = st.text_input("🔍 Search for a product:")
filtered_products = [p for p in product_list if search_query.lower() in p.lower()] if search_query else product_list

if search_query and not filtered_products:
    st.warning("No products found matching your search.")

selected_product = st.selectbox("Choose a product to get recommendations:", filtered_products)
```

```python
if st.button("Get Recommendations"):
    recommendations, avg_similarity = get_recommendations(selected_product, df, cosine_sim, product_name_to_indices)

    if len(recommendations) == 0:
        st.warning("No recommendations found for this product.")
    else:
        st.subheader("🔎 Recommended Products:")
        st.markdown(f"*Similarity Score:* {avg_similarity * 100:.2f}%")
        for _, row in recommendations.iterrows():
            st.markdown(f"- [{row['product_name']}]({row['product_link']})")
```

11. Conclusion

The Amazon Product Recommender system offers a simple yet powerful way to discover related products. Its modular and efficient design ensures usability and extensibility for future enhancements.