ML FInal Report

Recommender System for Laptop Selection

Nurlykhan Dastan
ID:220103027

INTRODUCTION

Recommender systems play a crucial role in various industries by providing personalized suggestions to users, enhancing their experience and engagement. This project focuses on building a recommendation system to suggest laptops to students and teachers based on their specific needs and preferences. The system is designed to assist users in selecting the best laptops for work and study, addressing challenges faced during the decision-making process.

This project was inspired by the personal experience of encountering difficulties when applying to university and selecting a suitable laptop for academic purposes. By leveraging both content-based and collaborative filtering techniques, we aim to provide accurate and reliable recommendations. The system integrates a hybrid approach to combine the strengths of these methodologies and evaluates performance using key metrics such as RMSE, MAE, Precision, Recall, and F1-score.

The dataset for this project was scraped from the Technodom.kz website, a leading e-commerce platform in Kazakhstan. The dataset includes product specifications, prices, and user reviews, ensuring relevance and practicality for the target audience.

WHY I CHOSE A RECOMMENDER SYSTEM

The motivation to build a recommender system stems from its potential to address real-world challenges in decision-making. When I applied to university, I faced significant difficulties selecting a laptop that met my academic needs. A well-designed recommendation system could have simplified this process by analyzing available products and aligning them with my preferences.

Recommender systems are invaluable in scenarios with an overwhelming number of choices, such as the laptop market. By tailoring suggestions to individual needs, these systems save time and reduce cognitive overload for users. This project aims to fill this gap for students and teachers, ensuring they can make informed decisions based on accurate and personalized recommendations.

CONTENT

The dataset includes :

- **Product Attributes:** RAM, SSD size, price, purpose, and operating system.
- **User Ratings:** Simulated user ratings for products to support collaborative filtering.

The dataset was preprocessed to handle missing values, assign unique product IDs, and normalize features for content-based filtering. Simulated ratings were generated to mimic real-world user behavior.

DATA PREPROCESSING

Steps Performed:

1. **Handling Missing Values:**

    - Missing values in RAM, SSD, and price columns were filled with default or mean values.
2. **Feature Normalization:**
    - RAM, SSD, and price features were normalized to ensure uniformity in content-based filtering.
3. **Simulated User Ratings:**
    - Randomized ratings between 0 and 5 were assigned to each user-product pair to facilitate collaborative filtering.
4. **Data Splitting:**
    - The dataset was split into training and testing sets to evaluate model performance.

MODEL ARCHITECTURE

Techniques Explored:

1. **Content-Based Filtering:**
    - Recommendations based on cosine similarity between user preferences and product attributes (RAM, SSD, and price).
2. **Collaborative Filtering:**
    - User-item collaborative filtering using cosine similarity to identify similar users and recommend products based on their preferences.
3. **Hybrid Model:**
    - Combines content-based and collaborative filtering to provide more comprehensive recommendations.

TRAINING PROCESS

Key Components:

- **Loss Function:** Not directly applicable as the system relies on similarity-based rankings.
- **Evaluation Metrics:**
    - RMSE, MAE for numerical accuracy.

    - Precision, Recall, and F1-score for binary relevance predictions.

Hyperparameters:

- Number of Users: 2
- Number of Epochs (for simulation): 30
- Rating Range: 0 to 5

EVALUATION METRICS

Results:

- **Root Mean Squared Error (RMSE):** Measures the average deviation of predicted ratings from actual ratings.
- **Mean Absolute Error (MAE):** Reflects the average magnitude of prediction errors.

Example Output:

- RMSE:
- MAE:

DISCUSSIONS

Strengths:

- Effective combination of content-based and collaborative filtering.
- High precision and recall indicate a balance between recommending relevant products and minimizing irrelevant suggestions.
- Robust evaluation metrics provide insights into system performance.

Challenges:

- Simulated ratings may not fully represent real-world user behavior.

- Limited dataset size can affect generalization.

EXPLANATION OF MODEL

IMPORTING

```
import requests  # To send HTTP requests
from bs4 import BeautifulSoup  # To parse HTML content
import pandas as pd  # To work with data structures like DataFrame
import os  # To interact with the operating system (checking if a file exists)
import re
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
```

requests: To fetch HTML content from the Technodom.kz website.
BeautifulSoup: To parse and extract data from the HTML fetched using requests.
pandas: To organize and manipulate the scraped data into a structured format, typically a DataFrame.
os: To check for the existence of files or manage file paths during data storage.
re: To perform advanced pattern matching and string manipulation, likely useful for extracting specific data like product specifications or cleaning text.
matplotlib.pyplot and seaborn: To create insightful and aesthetic visualizations of the data, aiding in exploratory data analysis (EDA).
numpy: For numerical operations, such as matrix manipulation or computing distances.
cosine_similarity from sklearn.metrics.pairwise: To calculate the similarity between laptop specifications or user profiles, which is central to recommendation systems.

DATA SCRAPING

The goal of the data scraping process is to collect detailed product information from the Technodom.kz website. The scraped data includes product names, ratings, number of reviews, prices, bonuses, and credit terms, which are essential for building a laptop recommendation system tailored to students and teachers.To scrape the data, a web scraping script was implemented using Python with the `requests` and `BeautifulSoup` libraries. The script efficiently navigates through multiple pages of the website, extracting key attributes of laptops.

1.Base URL Initialization:

The base URL of the Technodom.kz laptops category was defined to ensure the script fetches data from the correct webpage.

2. Pagination Handling:

The script loops through pages 1 to 40, constructing page-specific URLs dynamically and sending GET requests to fetch the corresponding HTML content.

3.Data Extraction:

For each product on the page:

- Product Name: Extracted using the specific CSS class associated with the product title.
- Product Rating: Extracted if available; otherwise, marked as 'N/A'.
- Number of Reviews: Parsed similarly, defaulting to 'N/A' if absent.
- Price: Scraped from the price section of the product card.
- Bonuses: Information about promotional bonuses was included if available.
- Credit Terms: Data about installment or credit plans for the product was also captured.

4.Data Storage:

The extracted data was stored in Python lists and subsequently organized into a `pandas.DataFrame` for further processing. The DataFrame was saved to a CSV file (`ml_final_dataset.csv`) in append mode if the file already existed, ensuring no data loss during multiple runs.

Results

The scraping process successfully retrieved the following attributes for each product:

- Product Name
- Product Rating
- Number of Reviews
- Price
- Bonus
- Credit Terms

Function: process_product

```python
# Update the return statement to include 'operating_system'
def process_product(product_name):
    """
    Process the product name to classify its purpose, extract RAM, SSD, and clean the name.
    """
    # Clean up the product name
    product_name = product_name.replace("\xa0", " ")  # Replace non-breaking spaces
    product_name = re.sub(r"\s+", " ", product_name.strip())  # Normalize spaces

    # Identify purpose
    if "Игровой ноутбук" in product_name:
        purpose = "For game"
    elif "Ноутбук" in product_name:
        purpose = "For study"
    else:
        purpose = "Both"

    # Extract SSD size explicitly
    ssd_match = re.search(r"(\d+)\s*[Гг]?[Бб]?\s*SSD", product_name, re.IGNORECASE)
    ssd = int(ssd_match.group(1)) if ssd_match else np.nan

    # Extract RAM size explicitly
    ram_match = re.search(r"(\d+)[Гг][Бб]\s*(?!SSD)", product_name, re.IGNORECASE)
    ram = int(ram_match.group(1)) if ram_match else np.nan

    # Extract operating system
    os_match = re.search(r"(DOS|Win\d+)", product_name, re.IGNORECASE)
    operating_system = os_match.group(1) if os_match else "MacOS"

    # Clean product name
    name = re.sub(r"(Игровой ноутбук|Ноутбук)", "", product_name, flags=re.IGNORECASE).strip()

```

This function processes product names to extract meaningful information like purpose, RAM size, SSD size, and the operating system.

Steps:
Clean Product Name:
Replace non-breaking spaces (\xa0) with regular spaces.Normalize multiple spaces into a single space using regex.
Classify Purpose:
Check if the product name contains keywords:
"Игровой ноутбук" indicates a gaming laptop ("For game").
"Ноутбук" indicates a study laptop ("For study").
Otherwise, classify as "Both" (general-purpose).
Extract SSD Size:
Use regex to find SSD sizes, e.g., "512 SSD".
Convert the extracted value to an integer; set it to NaN if no match is found.
Extract RAM Size:
Use regex to find RAM sizes, e.g., "16ГБ".
Exclude any matches that may overlap with SSD ("(?!SSD)").
Convert the extracted value to an integer; set it to NaN if no match is found.
Extract Operating System:

Use regex to extract the operating system (DOS, WinXX, etc.).
Default to "MacOS" if no operating system is specified.
Clean Product Name:
Remove specific keywords like "Игровой ноутбук" or "Ноутбук" from the product name for clarity.
Return Values:
name: Cleaned product name.
purpose: Identified purpose ("For game", "For study", "Both").
ram: Extracted RAM size in GB.
ssd: Extracted SSD size in GB.
operating_system: Extracted or default operating system.
Processing and DataFrame Creation
Apply the process_product Function:
Process each product name in the data['Product Name'] column.
Extract Product Name, Purpose, RAM, SSD, and Operating System.
Create DataFrame:
Convert the processed data into a new DataFrame processed_df with the extracted columns.
Add Additional Columns:
If the original dataset includes a Price column, copy it to the new DataFrame. Otherwise, fill it with NaN.
Similarly, handle the Product Rating column.
Save the Processed DataFrame:
Save the updated DataFrame as a CSV file (2.csv) for further analysis or use.
Code Purpose:
This code is designed to preprocess a dataset of product names, extracting key attributes for further analysis and use in recommendation systems. The final DataFrame includes clean and structured data that can be used for content-based filtering or hybrid models in a recommender system.

```python
def simulate_user_data(data, num_users=5, random_state=42):
    """
    Assign different User IDs and Ratings for collaborative filtering simulation.
    """
    np.random.seed(random_state)  # Set random state for reproducibility
    simulated_data = []
    for _, row in data.iterrows():
        for user_id in range(1, num_users + 1):
            rating = np.random.uniform(0, 5)  # Random ratings between 1 and 5
            simulated_data.append({
                'User ID': user_id,
                'Product ID': row['Product ID'],
                'Product Name': row['Product Name'],
                'Product Rating': rating
            })
    return pd.DataFrame(simulated_data)
```

The function simulate_user_data generates a simulated dataset of user ratings for collaborative filtering. Here's an explanation of its components:

Function Purpose

This function creates a dataset where each product is rated by multiple simulated users. This is essential for collaborative filtering, which relies on user-item interactions to make recommendations.

Parameters:

data: The original product dataset, which must include columns like Product ID and Product Name.

num_users: The number of users to simulate. Default is 1.

random_state: Ensures reproducibility by setting a seed for random number generation.

Steps:

Set Random State:

Ensures consistent results across runs by fixing the seed (np.random.seed(random_state)).

Iterate Over Products:

For each row (product) in the dataset:

Simulate ratings for num_users users.

Ratings are uniformly distributed between 0 and 5.

Generate Simulated Data:

For every user-product pair, create a dictionary with:

User ID: The ID of the simulated user (1 to num_users).

Product ID: The product's unique identifier.

Product Name: The product's name.

Product Rating: A randomly generated rating between 0 and 5.

Return DataFrame:

Convert the list of dictionaries (simulated_data) into a Pandas DataFrame for further analysis or use.

```python
def user_preferences(purpose):
    print(f"Enter preferences for {purpose}:")
    try:
        ram = int(input("Preferred RAM (GB): "))
        ssd = int(input("Preferred SSD size (GB): "))
        min_price = float(input("Minimum price (KZT): "))
        max_price = float(input("Maximum price (KZT): "))
        if min_price > max_price:
            print("Minimum price cannot be greater than maximum price. Restart and try again.")
            exit()
    except ValueError:
        print("Invalid input. Please enter numerical values.")
        exit()

    return {'RAM': ram, 'SSD': ssd, 'Min Price': min_price, 'Max Price': max_price}
```

The function `user_preferences` collects user preferences for filtering and recommending laptops. Here's a breakdown of its components:
Purpose:To capture user-specific requirements for laptop attributes such as RAM, SSD size, and price range. These preferences are used as input for recommendation algorithms (e.g., content-based filtering).
Steps
1.Prompt for User Preferences:
- Asks the user for their desired specifications based on the provided `purpose` (e.g., "Gaming," "Study").
- Accepts values for:
  - RAM(in GB): Integer input for required memory size.
  - SSD(in GB): Integer input for required storage size.
  - Min Price(in KZT): Floating-point input for minimum budget.
  -Max Price(in KZT): Floating-point input for maximum budget.

2.Validation:
- Ensures `Min Price` does not exceed `Max Price`. If invalid, the program exits with a message.If non-numerical values are entered, the program catches the `ValueError` and exits gracefully with an error message.
3.Return User Preferences:
- Constructs and returns a dictionary with user preferences:

```
{
    'RAM': ram,
    'SSD': ssd,
    'Min Price': min_price,
    'Max Price': max_price
}
```

This function is useful for filtering a dataset of laptops based on user-defined criteria and generating personalized recommendations. Let me know if you want additional functionality or integration examples

```python
def collaborative_recommendations(user_id):
    """
    Recommend laptops based on user-item collaborative filtering using cosine similarity.
    """
    # Create the user-item matrix
    user_item_matrix = simulated_user_data.pivot_table(
        index='User ID',
        columns='Product ID',
        values='Product Rating'
    ).fillna(0)

    # Check if user_id exists
    if user_id not in user_item_matrix.index:
        return pd.DataFrame()  # Return empty DataFrame if user_id not found

    # Compute cosine similarity
    cosine_sim = cosine_similarity(user_item_matrix)

    # Get the user index for the provided user_id
    user_idx = user_item_matrix.index.get_loc(user_id)

    # Get similarity scores for the target user
    sim_scores = list(enumerate(cosine_sim[user_idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Skip the target user (first one in the sorted list) and get the next 5 most similar users
    top_users = sim_scores[1:6]
    similar_user_indices = [idx for idx, score in top_users]

    # Get product recommendations based on similar users' ratings
    recommended_products = user_item_matrix.iloc[similar_user_indices].mean(axis=0)
    recommended_products = recommended_products.sort_values(ascending=False).head(5)

    # Map product IDs back to product names
    recommended_product_ids = recommended_products.index
    return processed_df[processed_df['Product ID'].isin(recommended_product_ids)][['Product ID', 'Product Name', 'Price', 'RAM', 'SSD']]
```

This function generates laptop recommendations for a specific user using user-item collaborative filtering and cosine similarity.
Steps:
1.Create the User-Item Matrix:
- A pivot table is created with:
  - Rows: `User ID` (each row represents a user).
  - Columns: `Product ID` (each column represents a product).
  - Values: `Product Rating` (ratings provided by users).
- Missing values (`NaN`) are replaced with `0` to ensure proper similarity computation.
2. Check if the `User ID` Exists:
- If the provided `user_id` is not in the dataset, an empty DataFrame is returned.
3.Compute Cosine Similarity:
- The `cosine_similarity` function calculates similarity scores between all users based on their ratings.
- The resulting matrix has dimensions `(num_users, num_users)`.
4.Identify Similar Users:
- The similarity scores for the target user are retrieved using their index.
- The scores are sorted in descending order.
- The target user is skipped (highest similarity with themselves), and the next 5 most similar users are selected.
5. Aggregate Ratings from Similar Users:
- Ratings for products are averaged across the selected similar users.

- The products are sorted by the average ratings in descending order, and the **top 5 products** are selected.
6. Map Product IDs Back to Names:
- The recommended product IDs are mapped back to the original dataset (`processed_df`) to retrieve their names and attributes (e.g., `Price`, `RAM`, `SSD`).
This function is ideal for generating personalized recommendations based on user interactions, particularly when no explicit product features are available. Let me know if you'd like additional enhancements or explanations.

```python
def content_based_filtering(data, user_preferences):
    """
    Recommend laptops based on cosine similarity.
    """
    # Fill NaN values with 0 (or other suitable default values)
    features = data[['RAM', 'SSD', 'Price']].fillna(0)

    # Normalize feature values
    features_normalized = (features - features.min()) / (features.max() - features.min())
    features_normalized = features_normalized.fillna(0)  # Ensure no NaN after normalization

    # Prepare user vector and normalize it
    user_vector = np.array([
        user_preferences['RAM'],
        user_preferences['SSD'],
        (user_preferences['Min Price'] + user_preferences['Max Price']) / 2
    ])
    user_vector_normalized = (user_vector - features.min().values) / (features.max().values - features.min().values)
    user_vector_normalized = np.nan_to_num(user_vector_normalized)  # Handle any residual NaN values

    # Compute cosine similarity
    similarity_scores = cosine_similarity([user_vector_normalized], features_normalized.values)[0]

    # Append similarity scores to the data and sort
    data = data.copy()
    data['Similarity'] = similarity_scores
    return data.sort_values(by='Similarity', ascending=False)
```

This function implements content-based filtering to recommend laptops to users based on their specified preferences. It uses cosine similarity to measure how closely each product matches the user's preferences.
Steps
1.Fill Missing Values
- Any `NaN` values in the `RAM`, `SSD`, or `Price` columns are replaced with `0`.
- Ensures no interruptions during normalization or similarity computations.
2.Normalize Feature Values
- Normalize the features (`RAM`, `SSD`, `Price`) using the formula:

$$\text{Normalized Value} = \frac{\text{Value} - \text{Min}}{\text{Max} - \text{Min}}$$

- This scales all features to a range between `0` and `1`.
- Ensures attributes with larger numerical ranges (e.g., `Price`) do not dominate similarity calculations.
3.Prepare User Vector
- The user's preferences (`RAM`, `SSD`, and the midpoint of the `Min Price` and `Max Price`) are used to create a vector.

- Normalize the user vector using the same formula as the product features.

4.Compute Cosine Similarity

- Cosine similarity is calculated between the user's vector and all normalized product feature vectors: $$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$
  - Measures how similar the user's preferences are to each product in the dataset.

5.Append Similarity Scores

- A new column, `Similarity`, is added to the dataset.
- Products are sorted in descending order of similarity scores.

Parameters:

1.`data`: A DataFrame containing product attributes (`RAM`, `SSD`, `Price`) and any other columns.

2.`user_preferences`: A dictionary specifying the user's preferences:

```
{
    'RAM': int,
    'SSD': int,
    'Min Price': float,
    'Max Price': float
}
```

Use Case

This function is ideal for scenarios where detailed product specifications are available, and recommendations need to align closely with user-defined criteria.

Let me know if you'd like additional functionality or clarifications

```python
def hybrid_filtering(data, user_preferences, content_top_n=5, collaborative_user_id=4):
    """
    Combine content-based and collaborative filtering for recommendations.
    """
    content_recs = content_based_filtering(data, user_preferences).head(content_top_n)

    # Get collaborative recommendations for the specified user
    collaborative_recs = collaborative_recommendations(collaborative_user_id)

    # Ensure collaborative_recs has the necessary columns
    if not collaborative_recs.empty:
        if 'Similarity' not in collaborative_recs.columns:
            collaborative_recs['Similarity'] = np.nan
        collaborative_recs = collaborative_recs[['Product ID', 'Product Name', 'Price', 'RAM', 'SSD', 'Similarity']]

    # Combine content-based and collaborative recommendations
    if not collaborative_recs.empty:
        hybrid_recs = pd.concat([
            content_recs[['Product ID', 'Product Name', 'Price', 'RAM', 'SSD', 'Similarity']],
            collaborative_recs
        ], ignore_index=True).drop_duplicates(subset='Product ID')
    else:
        hybrid_recs = content_recs[['Product ID', 'Product Name', 'Price', 'RAM', 'SSD', 'Similarity']]

    return hybrid_recs.sort_values(by='Similarity', ascending=False)
```

The `hybrid_filtering` function combines content-based filtering and collaborative filtering to generate a more comprehensive set of recommendations. By merging the strengths of both methods, the hybrid approach provides personalized and balanced suggestions.

Steps

1.Generate Content-Based Recommendations

- The function uses the `content_based_filtering` method to generate recommendations based on the user's preferences.
- Returns the top `content_top_n` recommendations sorted by similarity scores.

2.Generate Collaborative Recommendations

- The function uses the `collaborative_recommendations` method to generate suggestions based on the preferences of users similar to `collaborative_user_id`.

3.Check for Collaborative Recommendations

- If `collaborative_recs` is not empty, it ensures that the `Similarity` column is present. If not, it assigns a default value of `NaN`.

4.Combine Recommendations

- If both content-based and collaborative recommendations are available:
  - Combines the two sets into a single DataFrame.
  - Removes duplicate products based on `Product ID` to avoid redundancy.
- If collaborative recommendations are empty:
  - Returns only the content-based recommendations.

5.Sort Combined Recommendations

- The combined recommendations are sorted by the `Similarity` column in descending order, prioritizing the most relevant items.

Parameters

1.`data`: A DataFrame containing product details (e.g., `RAM`, `SSD`, `Price`).
2.`user_preferences`: A dictionary specifying the user's preferences for content-based filtering.
3.`content_top_n`:Number of top content-based recommendations to include (default is 5).

Use Case

This hybrid function is ideal for situations where both product attributes and user interaction data are available, providing a more comprehensive recommendation system.

```python
print("Select the purpose of the laptop:")
print("1) Gaming")
print("2) Study")
print("3) Both")
try:
    purpose_choice = int(input("Enter your choice (1, 2, or 3): "))
    if purpose_choice == 1:
        purpose = "For game"
    elif purpose_choice == 2:
        purpose = "For study"
    elif purpose_choice == 3:
        purpose = "Both"
    else:
        print("Invalid choice. Restart and try again.")
        exit()
except ValueError:
    print("Invalid input. Restart and enter a number.")
    exit()

# Filter laptops based on purpose
filtered_df = processed_df[processed_df['Purpose'] == purpose]
filtered_df = assign_ids(filtered_df)

# Ask about OS preference
os_choice = input("Do you want a specific operating system? (yes/no): ").lower()
if os_choice == 'yes':
    filtered_df = filtered_df[filtered_df['Operating System'].isin(['Win11', 'MacOS'])]
elif os_choice == 'no':
    filtered_df = filtered_df[filtered_df['Operating System'] == 'DOS']

if filtered_df.empty:
    print("No laptops found matching your criteria. Try different preferences.")
    exit()

# Get user preferences
user_preferences = user_preferences(purpose)
```

This code allows a user to filter and select laptops based on their specific preferences and provides recommendations using content-based filtering. Below is a detailed breakdown of its components:

1.User Input for Laptop Purpose

- Prompts the user to choose the purpose of the laptop:
  - 1:Gaming
  - 2:Study
  - 3:Both
- The purpose is assigned to the variable `purpose` based on the user's choice.

Error Handling:

- If the input is not a valid number or outside the range `1-3`, the program exits with an error message.

2.Filter Laptops Based on Purpose

- Filters the `processed_df` DataFrame to only include laptops matching the selected purpose.
- Uses the helper function `assign_ids` to ensure each laptop has a unique product ID.

3.Operating System Preference

- Asks the user if they want a specific operating system:

  - `yes`:Filters laptops with operating systems `Win11` or `MacOS`.
  - `no`: Filters laptops with the `DOS` operating system.
- If no laptops match the criteria, the program exits with an appropriate message.

4.User Preferences

- Collects additional preferences such as:
  -RAM (GB)
  - SSD Size (GB)
  - Price Range (Min and Max in KZT)
- The function `user_preferences` validates these inputs and returns them as a dictionary.

5.Filter Laptops Within Price Range

- Filters the DataFrame further to only include laptops within the user-defined price range:

- If no laptops match, the program exits with a message.

6.Content-Based Recommendations

- Passes the filtered dataset and user preferences to the `content_based_filtering` function.
- Generates a ranked list of laptops based on cosine similarity between product attributes and user preferences.

7.Display Recommendations

- Displays the top 5 content-based recommendations with the following attributes:
  - `Product Name`
  - `Purpose`
  - `Operating System`
  - `Price`
  - `RAM`
  - `SSD`
  - `Similarity`

CONCLUSION

This project demonstrates the development of a hybrid recommendation system using content-based and collaborative filtering techniques. By evaluating performance with key metrics, the system shows promise in delivering accurate and personalized recommendations. Future work will focus on incorporating additional user attributes, expanding the dataset, and optimizing algorithms for better generalization.