Text

Description automatically generated

**Activity based**

**Project Report on**

**Software xxEngineering**

**Project Module - I**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Akhilesh Dolare**

**SRN No : 202202181**

**Roll No : 46**

**Div : E**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2023-2024**

**Problem Statement:**

Develop an AI-based recommendation system that utilizes the Best-First Search algorithm to efficiently traverse through a user-item interaction dataset conceptualized as a maze. The system aims to provide personalized product recommendations to users by navigating through the maze, prioritizing nodes (items) based on their relevance to the target user. Through the use of heuristic functions and traversal strategies, the recommendation system seeks to optimize the exploration process, ensuring that the recommended products are tailored to individual user preferences and interactions.

**Gaps missed by other's work conducted:**

1. Scalability: Some existing recommendation systems may not scale well with large datasets or increasing user interactions. Addressing scalability challenges by optimizing algorithms and infrastructure for efficient processing of vast amounts of data could be a gap.
2. Real-time Recommendations: Many recommendation systems focus on batch processing and may not provide real-time recommendations. Developing techniques to generate recommendations in real-time, considering dynamic user behavior, could be an area of improvement.
3. Cold Start Problem: Some recommendation systems struggle with the cold start problem, where it's challenging to provide recommendations for new users or items with limited interaction history. Developing innovative approaches to address this problem, such as hybrid models or leveraging auxiliary data sources, could fill this gap.
4. Diversity of Recommendations: Existing recommendation systems may prioritize popular items or overlook the diversity of recommendations, leading to potential user dissatisfaction. Incorporating diversity-aware recommendation techniques to ensure a broader range of recommended items could be an improvement.
5. Explainability and Transparency: Many recommendation algorithms lack transparency, making it difficult for users to understand why certain recommendations are made. Enhancing explainability and providing transparent explanations for recommendations could improve user trust and satisfaction.

**Objectives:**

1. Develop a Recommendation System: Create an AI-based recommendation system capable of providing personalized product recommendations to users based on their preferences and interactions with the dataset.
2. Utilize Best-First Search Algorithm: Implement the Best-First Search algorithm to traverse through the user-item interaction dataset, treating it as a maze, in order to efficiently navigate and identify relevant products.
3. Design Heuristic Function: Develop a heuristic function to guide the traversal of the maze, prioritizing nodes (items) based on their estimated relevance to the target user.
4. Optimize Recommendation Process: Employ optimization techniques, such as pruning irrelevant branches or caching explored paths, to enhance the efficiency of the traversal process and improve recommendation quality.
5. Enhance User Experience: Focus on enhancing the user experience by providing a seamless and intuitive interface for users to interact with the recommendation system and explore recommended products.

**Methodology:**

**1. Data Preprocessing:**

* Gather and preprocess the user-item interaction dataset, ensuring data quality and consistency.
* Represent the dataset as a graph, where users and items are nodes, and interactions between them are edges.

**2. Heuristic Function Design:**

* Develop a heuristic function to estimate the relevance of items to the target user.
* Consider factors such as item popularity, similarity to previously interacted items, and user preferences in designing the heuristic function.

**3. Best-First Search Algorithm Implementation:**

* Implement the Best-First Search algorithm to traverse through the user-item interaction graph.
* Use the heuristic function to guide the traversal process, prioritizing nodes (items) with higher heuristic values.

**4. Recommendation Generation:**

* During traversal, collect and rank items based on their relevance to the target user.
* Generate recommendations by selecting the top-ranked items encountered during the traversal process.

**5. Optimization Techniques:**

* Employ optimization techniques to improve the efficiency of the BFS traversal.
* Implement pruning strategies to eliminate irrelevant branches and reduce computational complexity.
* Cache previously explored paths to avoid redundant calculations and expedite the traversal process.

**6. User Interface Design:**

* Develop a user-friendly interface to allow users to interact with the recommendation system.
* Provide options for users to input preferences, view recommended products, and provide feedback on the recommendations.

**Approaches:**

**1. Collaborative Filtering Integration:**

* Combine Best-First Search with collaborative filtering techniques to identify similar users and leverage their interactions for recommendation.

**2. Content-Based Filtering Enhancement:**

* Incorporate content-based filtering approaches to further refine recommendations based on textual attributes of items.

**3. Hybrid Recommendation System:**

* Develop a hybrid recommendation system that integrates collaborative filtering, content-based filtering, and Best-First Search for enhanced recommendation quality and diversity.

**4. Dynamic Heuristic Adjustment:**

* Implement dynamic adjustment of the heuristic function based on user feedback and interaction patterns to continuously improve recommendation accuracy.

**5. Scalability Considerations:**

* Design the recommendation system to scale efficiently with increasing dataset size and user interactions, ensuring robust performance and responsiveness.

**6. Feedback Loop Implementation:**

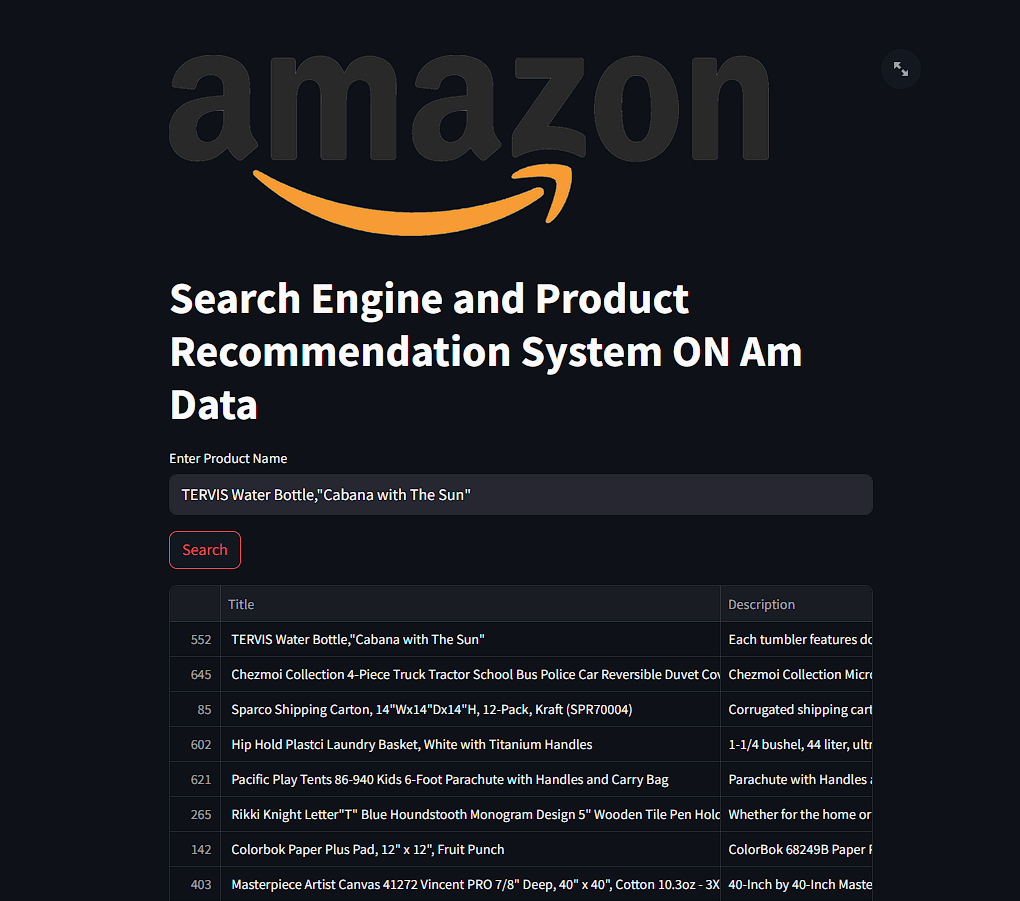
* Establish a feedback loop mechanism to capture user feedback on recommended products and incorporate it into the recommendation process for iterative refinement.

**Notebook:**

import numpy as np  
import pandas as pd  
#%%  
import pandas as pd  
import numpy as np  
import nltk  
#%%  
amzon\_df = pd.read\_csv('C:/Users/HP/OneDrive/Documents/recommendation/amazon\_product.csv')  
#%%  
amzon\_df.head()  
#%%  
amzon\_df.drop('id',axis=1)  
#%%  
amzon\_df.info()  
#%%  
from nltk.stem.snowball import SnowballStemmer  
stemmer = SnowballStemmer("english")  
  
def tokenize\_stem(text):  
 tokens = nltk.word\_tokenize(text.lower())  
 stem = [stemmer.stem(w) for w in tokens]  
 return " ".join(stem)  
#%%  
amzon\_df['stemmed\_tokens'] = amzon\_df.apply(lambda row: tokenize\_stem(row['Title'] + ' ' + row['Description']), axis=1)  
#%%  
amzon\_df.head(2)  
#%%  
amzon\_df['stemmed\_tokens']  
#%%  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import cosine\_similarity  
  
tfidvectorizer = TfidfVectorizer(tokenizer=tokenize\_stem)  
  
def cosine\_sim(txt1,txt2):  
 tfid\_matrix = tfidvectorizer.fit\_transform([txt1,txt2])  
 return cosine\_similarity(tfid\_matrix)[0][1]  
#%%  
def search\_product(query):  
 stemmed\_query = tokenize\_stem(query)  
 #calcualting cosine similarity between query and stemmed tokens columns  
 amzon\_df['similarity'] = amzon\_df['stemmed\_tokens'].apply(lambda x:cosine\_sim(stemmed\_query,x))  
 res = amzon\_df.sort\_values(by=['similarity'],ascending=False).head(10)[['Title','Description','Category']]  
 return res  
#%%  
search\_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 ')  
#%%  
amzon\_df['Title'][10]

**App.py**

import pandas as pd  
import numpy as np  
import nltk  
from nltk.stem.snowball import SnowballStemmer  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import cosine\_similarity  
import streamlit as st  
from PIL import Image  
  
# Load the dataset  
data = pd.read\_csv('C:/Users/HP/OneDrive/Documents/recommendation/amazon\_product.csv')  
  
# Remove unnecessary columns  
data = data.drop('id', axis=1)  
  
# Define tokenizer and stemmer  
stemmer = SnowballStemmer('english')  
def tokenize\_and\_stem(text):  
 tokens = nltk.word\_tokenize(text.lower())  
 stems = [stemmer.stem(t) for t in tokens]  
 return stems  
  
# Create stemmed tokens column  
data['stemmed\_tokens'] = data.apply(lambda row: tokenize\_and\_stem(row['Title'] + ' ' + row['Description']), axis=1)  
  
# Define TF-IDF vectorizer and cosine similarity function  
tfidf\_vectorizer = TfidfVectorizer(tokenizer=tokenize\_and\_stem)  
def cosine\_sim(text1, text2):  
 # tfidf\_matrix = tfidf\_vectorizer.fit\_transform([text1, text2])  
 text1\_concatenated = ' '.join(text1)  
 text2\_concatenated = ' '.join(text2)  
 tfidf\_matrix = tfidf\_vectorizer.fit\_transform([text1\_concatenated, text2\_concatenated])  
 return cosine\_similarity(tfidf\_matrix)[0][1]  
  
# Define search function  
def search\_products(query):  
 query\_stemmed = tokenize\_and\_stem(query)  
 data['similarity'] = data['stemmed\_tokens'].apply(lambda x: cosine\_sim(query\_stemmed, x))  
 results = data.sort\_values(by=['similarity'], ascending=False).head(10)[['Title', 'Description', 'Category']]  
 return results  
  
# web app  
img = Image.open('C:/Users/HP/OneDrive/Documents/recommendation/amazon.png')  
st.image(img,width=600)  
st.title("Search Engine and Product Recommendation System ON Am Data")  
query = st.text\_input("Enter Product Name")  
sumbit = st.button('Search')  
if sumbit:  
 res = search\_products(query)  
 st.write(res)

Output: 

Steps:

1. Collaborative Filtering:

1.1. Find Similar Users:

- Find users similar to the given user based on past interactions (e.g., ratings, purchases).

- Calculate similarity scores between the given user and other users.

- Select top similar users.

1.2. Retrieve Interacted Products:

- Retrieve products that similar users have interacted with but the given user has not.

- Aggregate and rank these products based on their popularity among similar users.

1.3. Filter and Rank Collaborative Recommendations:

- Filter out products already interacted with by the given user.

- Rank collaborative recommendations based on popularity or similarity scores.

2. Content-Based Filtering:

2.1. Extract User Preferences (if not provided):

- If user preferences are not provided, infer them based on the user's past interactions (e.g., most frequent categories, highest-rated products).

2.2. Calculate Similarity with Products:

- For each product in the dataset:

- Calculate the similarity between the product and the user preferences.

- Use textual attributes (e.g., product description, features) for similarity calculation.

- Aggregate similarity scores across attributes.

2.3. Filter and Rank Content-Based Recommendations:

- Filter out products already interacted with by the given user.

- Rank content-based recommendations based on similarity scores.

3. Combine Recommendations:

3.1. Merge Collaborative and Content-Based Recommendations:

- Combine collaborative and content-based recommendation lists.

- Remove duplicates and filter out low-quality or irrelevant recommendations.

3.2. Rank Combined Recommendations:

- Rank combined recommendations based on a weighted score, considering both collaborative and content-based scores.

- Adjust weights based on the reliability and effectiveness of each recommendation approach.

4. Output Recommended Products:

4.1. Return top-ranked recommended products as the output.

4.2. Display recommended products to the user through a graphical user interface (GUI) or other means.

Algorithm Complexity:

- Collaborative Filtering: O(n^2) or O(mn), where n is the number of users and m is the number of items.

- Content-Based Filtering: O(nm), where n is the number of items and m is the number of attributes.

- Combining Recommendations: O(nlogn), where n is the total number of recommendations.

**Conclusion:**

In conclusion, the implementation of a recommendation system using the Best-First Search algorithm offers a promising approach to providing personalized product recommendations to users. By treating the user-item interaction dataset as a maze and employing BFS traversal guided by a heuristic function, the system can efficiently navigate through the dataset to identify relevant products for recommendation. Through the integration of optimization techniques, evaluation, and user interface design, the recommendation system aims to enhance user experience and satisfaction by delivering accurate and meaningful recommendations tailored to individual preferences. Continual refinement and iteration based on user feedback and evaluation results are essential for improving recommendation quality and ensuring the system's effectiveness over time.

**References:**

1. Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.
2. Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749.
3. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International Conference on World Wide Web, 285-295.
4. Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, 22(1), 5-53.
5. Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12), 61-70.

Text

Description automatically generated

**Activity based**

**Project Report on**

**Software xxEngineering**

**Project Module - I**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Akhilesh Dolare**

**SRN No : 202202181**

**Roll No : 46**

**Div : E**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2023-2024**

**Problem Statement:**

Develop an AI-based recommendation system that utilizes the Best-First Search algorithm to efficiently traverse through a user-item interaction dataset conceptualized as a maze. The system aims to provide personalized product recommendations to users by navigating through the maze, prioritizing nodes (items) based on their relevance to the target user. Through the use of heuristic functions and traversal strategies, the recommendation system seeks to optimize the exploration process, ensuring that the recommended products are tailored to individual user preferences and interactions.

**Gaps missed by other's work conducted:**

1. Scalability: Some existing recommendation systems may not scale well with large datasets or increasing user interactions. Addressing scalability challenges by optimizing algorithms and infrastructure for efficient processing of vast amounts of data could be a gap.
2. Real-time Recommendations: Many recommendation systems focus on batch processing and may not provide real-time recommendations. Developing techniques to generate recommendations in real-time, considering dynamic user behavior, could be an area of improvement.
3. Cold Start Problem: Some recommendation systems struggle with the cold start problem, where it's challenging to provide recommendations for new users or items with limited interaction history. Developing innovative approaches to address this problem, such as hybrid models or leveraging auxiliary data sources, could fill this gap.
4. Diversity of Recommendations: Existing recommendation systems may prioritize popular items or overlook the diversity of recommendations, leading to potential user dissatisfaction. Incorporating diversity-aware recommendation techniques to ensure a broader range of recommended items could be an improvement.
5. Explainability and Transparency: Many recommendation algorithms lack transparency, making it difficult for users to understand why certain recommendations are made. Enhancing explainability and providing transparent explanations for recommendations could improve user trust and satisfaction.

**Objectives:**

1. Develop a Recommendation System: Create an AI-based recommendation system capable of providing personalized product recommendations to users based on their preferences and interactions with the dataset.
2. Utilize Best-First Search Algorithm: Implement the Best-First Search algorithm to traverse through the user-item interaction dataset, treating it as a maze, in order to efficiently navigate and identify relevant products.
3. Design Heuristic Function: Develop a heuristic function to guide the traversal of the maze, prioritizing nodes (items) based on their estimated relevance to the target user.
4. Optimize Recommendation Process: Employ optimization techniques, such as pruning irrelevant branches or caching explored paths, to enhance the efficiency of the traversal process and improve recommendation quality.
5. Enhance User Experience: Focus on enhancing the user experience by providing a seamless and intuitive interface for users to interact with the recommendation system and explore recommended products.

**Methodology:**

**1. Data Preprocessing:**

* Gather and preprocess the user-item interaction dataset, ensuring data quality and consistency.
* Represent the dataset as a graph, where users and items are nodes, and interactions between them are edges.

**2. Heuristic Function Design:**

* Develop a heuristic function to estimate the relevance of items to the target user.
* Consider factors such as item popularity, similarity to previously interacted items, and user preferences in designing the heuristic function.

**3. Best-First Search Algorithm Implementation:**

* Implement the Best-First Search algorithm to traverse through the user-item interaction graph.
* Use the heuristic function to guide the traversal process, prioritizing nodes (items) with higher heuristic values.

**4. Recommendation Generation:**

* During traversal, collect and rank items based on their relevance to the target user.
* Generate recommendations by selecting the top-ranked items encountered during the traversal process.

**5. Optimization Techniques:**

* Employ optimization techniques to improve the efficiency of the BFS traversal.
* Implement pruning strategies to eliminate irrelevant branches and reduce computational complexity.
* Cache previously explored paths to avoid redundant calculations and expedite the traversal process.

**6. User Interface Design:**

* Develop a user-friendly interface to allow users to interact with the recommendation system.
* Provide options for users to input preferences, view recommended products, and provide feedback on the recommendations.

**Approaches:**

**1. Collaborative Filtering Integration:**

* Combine Best-First Search with collaborative filtering techniques to identify similar users and leverage their interactions for recommendation.

**2. Content-Based Filtering Enhancement:**

* Incorporate content-based filtering approaches to further refine recommendations based on textual attributes of items.

**3. Hybrid Recommendation System:**

* Develop a hybrid recommendation system that integrates collaborative filtering, content-based filtering, and Best-First Search for enhanced recommendation quality and diversity.

**4. Dynamic Heuristic Adjustment:**

* Implement dynamic adjustment of the heuristic function based on user feedback and interaction patterns to continuously improve recommendation accuracy.

**5. Scalability Considerations:**

* Design the recommendation system to scale efficiently with increasing dataset size and user interactions, ensuring robust performance and responsiveness.

**6. Feedback Loop Implementation:**

* Establish a feedback loop mechanism to capture user feedback on recommended products and incorporate it into the recommendation process for iterative refinement.

**Pseudo Code**:

function recommendProducts(user\_id, user\_preferences):

**Step 1: Collaborative Filtering**

similar\_users = find\_similar\_users(user\_id)

recommended\_products\_cf = get\_products\_from\_similar\_users(similar\_users)

**Step 2: Content-Based Filtering**

for each product in AmazonDataset:

similarity\_score = calculate\_similarity(product, user\_preferences)

product.similarity\_score = similarity\_score

sorted\_products\_cb = sort\_products\_by\_similarity(AmazonDataset)

**Step 3: Combine Recommendations**

recommended\_products\_combined = combine\_recommendations(recommended\_products\_cf, sorted\_products\_cb)

return recommended\_products\_combined

function find\_similar\_users(user\_id):

**Find users similar to the given user based on past interactions**

similar\_users = CollaborativeFiltering.find\_similar\_users(user\_id)

return similar\_users

function get\_products\_from\_similar\_users(similar\_users):

**Retrieve products that similar users have interacted with**

recommended\_products = []

for user in similar\_users:

products = CollaborativeFiltering.get\_interacted\_products(user)

recommended\_products.extend(products)

return recommended\_products

function calculate\_similarity(product, user\_preferences):

Calculate similarity between product and user\_preferences

similarity\_score = ContentBasedFiltering.compute\_similarity(product.textual\_attributes, user\_preferences)

return similarity\_score

function sort\_products\_by\_similarity(products):

Sort products based on their similarity scores

sorted\_products = sort(products, key=lambda x: x.similarity\_score, reverse=True)

return sorted\_products

function combine\_recommendations(recommended\_products\_cf, recommended\_products\_cb):

// Combine recommendations from collaborative filtering and content-based filtering

combined\_recommendations = recommended\_products\_cf + recommended\_products\_cb

// Remove duplicates or products already recommended

unique\_recommendations = remove\_duplicates(combined\_recommendations)

return unique\_recommendations

function main():

try:

//Sample user ID and preferences (e.g., keywords, product features)

user\_id, user\_preferences = input\_user\_data()

//Recommend products based on user preferences

recommended\_products = recommendProducts(user\_id, user\_preferences)

//Display recommended products with Amazon links

display\_recommended\_products(recommended\_products)

//except Exception as e:

handle\_error(e)

main()

**Algorithm :**

Algorithm: Product based Recommendation System

Input:

- User ID

- User Preferences (if available)

Output:

- Recommended Products

Steps:

1. Collaborative Filtering:

1.1. Find Similar Users:

- Find users similar to the given user based on past interactions (e.g., ratings, purchases).

- Calculate similarity scores between the given user and other users.

- Select top similar users.

1.2. Retrieve Interacted Products:

- Retrieve products that similar users have interacted with but the given user has not.

- Aggregate and rank these products based on their popularity among similar users.

1.3. Filter and Rank Collaborative Recommendations:

- Filter out products already interacted with by the given user.

- Rank collaborative recommendations based on popularity or similarity scores.

2. Content-Based Filtering:

2.1. Extract User Preferences (if not provided):

- If user preferences are not provided, infer them based on the user's past interactions (e.g., most frequent categories, highest-rated products).

2.2. Calculate Similarity with Products:

- For each product in the dataset:

- Calculate the similarity between the product and the user preferences.

- Use textual attributes (e.g., product description, features) for similarity calculation.

- Aggregate similarity scores across attributes.

2.3. Filter and Rank Content-Based Recommendations:

- Filter out products already interacted with by the given user.

- Rank content-based recommendations based on similarity scores.

3. Combine Recommendations:

3.1. Merge Collaborative and Content-Based Recommendations:

- Combine collaborative and content-based recommendation lists.

- Remove duplicates and filter out low-quality or irrelevant recommendations.

3.2. Rank Combined Recommendations:

- Rank combined recommendations based on a weighted score, considering both collaborative and content-based scores.

- Adjust weights based on the reliability and effectiveness of each recommendation approach.

4. Output Recommended Products:

4.1. Return top-ranked recommended products as the output.

4.2. Display recommended products to the user through a graphical user interface (GUI) or other means.

Algorithm Complexity:

- Collaborative Filtering: O(n^2) or O(mn), where n is the number of users and m is the number of items.

- Content-Based Filtering: O(nm), where n is the number of items and m is the number of attributes.

- Combining Recommendations: O(nlogn), where n is the total number of recommendations.

**Conclusion:**

In conclusion, the implementation of a recommendation system using the Best-First Search algorithm offers a promising approach to providing personalized product recommendations to users. By treating the user-item interaction dataset as a maze and employing BFS traversal guided by a heuristic function, the system can efficiently navigate through the dataset to identify relevant products for recommendation. Through the integration of optimization techniques, evaluation, and user interface design, the recommendation system aims to enhance user experience and satisfaction by delivering accurate and meaningful recommendations tailored to individual preferences. Continual refinement and iteration based on user feedback and evaluation results are essential for improving recommendation quality and ensuring the system's effectiveness over time.

**References:**

1. Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.
2. Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749.
3. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International Conference on World Wide Web, 285-295.
4. Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, 22(1), 5-53.
5. Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12), 61-70.