Chapter 6

6 Conclusion and Future Scope

6.1 Introduction

The final chapter of the thesis is dedicated to consolidating the research findings and outlining the application of a Hybrid Model within a Decision Support System (DSS) specifically designed for dietary recommendations aimed at cardiac patients. This chapter not only serves as a finishing touch by integrating the methodologies and analyses presented throughout the thesis but also emphasizes practical implementations and the potential impacts on public health.

The introduction begins by summarizing the motivations behind developing the Hybrid Model, which combines collaborative filtering, popularity-based recommendations, and new-user recommendations to offer a robust, versatile system. This model is particularly crucial in addressing the diverse dietary needs and preferences of cardiac patients, ensuring that the recommendations are both scientifically sound and personalized.

The Hybrid Model leverages a comprehensive dataset, inclusive of user demographics, dietary preferences, and health objectives, to dietary suggestions. Such customization is crucial for cardiac patients who require specific nutritional care to manage their health conditions effectively. The model's development is rooted in the need to enhance the accuracy and relevance of food recommendations, thereby aiding patients in making informed decisions about their diets.

The chapter details the technical execution of the model, utilizing Flask as a web framework to deploy the recommendation system. This setup allows for real-time interaction with users, providing a user-friendly interface where patients can receive immediate dietary suggestions based on their inputs regarding age, weight, height, activity level, and specific health goals.

This comprehensive approach not only highlights the practical application of theoretical knowledge but also sets the stage for discussing the broader implications of the research, the system's performance, and potential areas for future development. Through detailed implementation descriptions and performance evaluations, this chapter aims to demonstrate

the effectiveness of the Hybrid Model and its capacity to contribute significantly to dietary management and health optimization for individuals with cardiac concerns.

6.2 Implementation of Hybrid Model

6.2.1 Implementation-Based on Collaborative Filtering

To understand how the hybrid model based on collaborative filtering is implemented in a Flask application, let's break down the relevant parts of the code provided. This implementation focuses on integrating a collaborative filtering model into a web application that can provide personalized food recommendations.

1. Import Necessary Libraries:

Flask: The framework to build and run the web server.

pickle: For loading the pre-trained machine learning model.

pandas: To handle data manipulation tasks.

```python
from flask import Flask, render_template, request, jsonify
import pickle
import pandas as pd
```

2. Setup Flask Application:

Initializes the Flask application to handle requests and responses.

```python
app = Flask(__name__)
```

3. Load Data:

Loads datasets required for the application, including food identifiers, user information, and popular food items.

```python
food_ids = pd.read_csv("data/food_ids.csv")
userinfo = pd.read_csv("data/userinfo1.csv")
popular_food = pd.read_csv("data/popular_food.csv")
```

4. Model Loading Function:

A function to load the pre-trained collaborative filtering model from a file. This model is essential for making predictions based on user preferences.

```python
def load_model(file_path):
    with open(file_path, 'rb') as f:
        model = pickle.load(f)
    return model
```

5. Get User-Specific Recommendations:

Retrieves personalized food recommendations for a user based on their username by querying the collaborative filtering model.

```python
def get_users_recommendations_by_name(user_name, n, model, food_ids_df):
    user_id = userinfo[userinfo['Username'] == user_name]['id'].values[0]
    recommended_items = pd.DataFrame(model.loc[user_id])
    recommended_items.columns = ["predicted_rating"]
    recommended_items = recommended_items.sort_values('predicted_rating', asce
    recommended_items = recommended_items.head(n)
    recommended_food_ids = recommended_items.index.tolist()
    recommended_food_names = [food_ids_df[food_ids_df['food_id'] == food_id]['
    return recommended_food_names
```

6. Flask Routing for User Recommendations:

This route handles the web page for user-based recommendations. It checks if the user exists in the database, fetches recommendations using the loaded model, and displays them on the webpage.

```python
@app.route('/user_base', methods=['POST', 'GET'])
def user_reco():
    if request.method == 'POST':
        user_name = request.form['user_name']
        num_recommendations = int(request.form['num_recommendations'])
        model = load_model('model/KNNWithZScore_cf_model.pkl')
        if user_name in list(userinfo['Username']):
            recommendations = get_users_recommendations_by_name(user_name, num_recommendat
        else:
            recommendations = get_new_users_recommendations(num_recommendations , food_ids
        return render_template("user_base.html", recommendations=recommendations)
    return render_template("user_base.html")
```

Explanation of Code Functionality:

This Flask application serves as a backend to handle user requests for food recommendations. Users can enter their details, and the application utilizes a collaborative filtering model to predict and display personalized food recommendations. This integration represents the collaborative filtering component of the hybrid model, allowing for dynamic user interaction and personalized experiences based on historical data and user preferences.

This structure facilitates understanding the practical implementation of machine learning models within web applications, specifically for personalized recommendation systems.

6.2.2 Implementation-Based on Popularity-Based

The popularity-based implementation within the hybrid model leverages the most liked or frequently chosen items to recommend food choices to new or existing users without personal historical data. This method is effective for providing generic recommendations based on broader user preferences. Let's break down the relevant sections of the Flask application that handle this popularity-based recommendation.

Code Implementation for Popularity-Based Recommendations:

1. Flask Route for Popularity-Based Recommendations:

• Route: This part of the code sets up an endpoint '/pop_food' that listens for POST and GET requests.

• Functionality: When a POST request is made, it captures the number of recommendations requested by the user. It then fetches this number of popular food items and displays them using the pop_food.html template. If a GET request is made, it simply returns the HTML page without any dynamic content.

```python
@app.route('/pop_food', methods=['POST', 'GET'])
def pop_food():
    if request.method == 'POST':
        num_recommendations = int(request.form['num_recommendations'])
        recommendations = get_new_users_recommendations(num_recommendations, fo
        return render_template("pop_food.html", recommendations=recommendations
    return render_template("pop_food.html")
```

2. Fetching Popular Food Recommendations:

Data Selection: This function selects a random sample of popular foods. It initially fetches a list twice the size of the required recommendations and randomly samples the desired number from this list. This approach adds a level of randomness to the popular choices.

Mapping IDs to Names: It maps the food IDs obtained to their corresponding names using the food_ids.csv dataframe for display.

```python
def get_new_users_recommendations(num_recommendations, food_ids_df):
    recommended_items = list(popular_food.head(num_recommendations * 2).sample(num_recomme
    recommended_food_names = [food_ids_df[food_ids_df['food_id'] == food_id]['Food_Name'].
    return recommended_food_names
```

Explanation:

The popularity-based recommendation part of the hybrid model is straightforward and does not depend on user-specific data, making it ideal for new users or when sufficient user data is not available. It relies on general popularity metrics, possibly derived from sales data, user ratings, or other forms of engagement metrics that indicate popularity. This method is implemented in a Flask route that handles both the display and backend logic for fetching and displaying these recommendations.

By setting this route within the Flask application, the hybrid model can cater to users by providing generalized recommendations based on what is most popular among a broader audience, thereby enhancing user experience when personalized data is not predominant. This implementation is critical for maintaining user engagement, especially for new platform users who have yet to generate enough personal data for more tailored recommendations.

6.2.3 Implementation-Based on New-User Recommendation

For the section on the hybrid model framework, the implementation focuses on accommodating users who do not have previous interactions or preferences logged within the system. This segment of the code ensures that new users receive recommendations based on popular items or a predefined logic suitable for first-time interactions. Here's a breakdown of how this is implemented in the Flask web application:

Relevant Code Implementation for New-User Recommendations:

1. Flask Route for Handling Recommendations for New Users:

Endpoint Setup: The /user_base route listens for both GET and POST requests. The POST request handles form submissions where user details and preferences are submitted.

Form Processing: When a form is submitted, the function checks if the username exists in the user info database. If the user is new (i.e., not found in the database), the system provides general recommendations instead of personalized ones.

Recommendation Logic: For new users, the get_new_users_recommendations function is invoked, which typically might involve fetching popular items or using a different algorithm suited for users without history.

python Copy code

```python
@app.route('/user_base', methods=['POST', 'GET'])
def user_reco():
    if request.method == 'POST':
        user_name = request.form['user_name']
        num_recommendations = int(request.form['num_recommendations'])

        # Check if the user exists and load recommendations accordingly
        if user_name in list(userinfo['Username']):
            # Existing user: fetch personalized recommendations
            recommendations = get_users_recommendations_by_name(user_name, num_recommendat
        else:
            # New user: fetch general recommendations
            recommendations = get_new_users_recommendations(num_recommendations, food_ids)

        return render_template("user_base.html", recommendations=recommendations)

    return render_template("user_base.html")
```

2. Function for Fetching General Recommendations for New Users:

Recommendation Selection: This function is designed to provide recommendations without prior user data. It selects several items from the top of a list sorted by popularity or other criteria (e.g., the popular food dataset), ensuring that new users receive items that are generally well-accepted or trending.

Dynamic Sampling: To maintain a level of freshness in recommendations, it randomly samples from the top-n popular items, making each recommendation unique.

```python
def get_new_users_recommendations(num_recommendations, food_ids_df):
    recommended_items = list(popular_food.head(num_recommendations * 2).sample(num_recomme
    recommended_food_names = [food_ids_df[food_ids_df['food_id'] == food_id]['Food_Name'].
    return recommended_food_names
```

Explanation:

This implementation ensures that new users, who are yet to establish a preference profile within the system, are still catered for effectively. It makes the hybrid model robust, enabling it to serve both users with rich interaction histories and those just starting out. The approach helps in improving user satisfaction and retention from the very first interaction by leveraging the insights gained from aggregate data on popular choices. This method is crucial for systems where building immediate engagement is necessary to convert new users into regular users.

6.3 Conclusion

The conclusion of this thesis brings together the essential findings and implications of implementing a hybrid model for food recommendation systems. Throughout this work, we've meticulously developed and assessed various methodologies that provide not only to existing users but also effectively engage new users. This dual approach enhances the user experience across different interaction levels, fostering both satisfaction and sustained engagement.

The collaborative filtering method, as detailed in the thesis, illustrates a refined strategy for personalized recommendations. By tapping into user-specific data, this method tailors suggestions that resonate more profoundly with individual tastes and preferences. Meanwhile,

the popularity-based recommendation system leverages the collective preferences of the user base to guide new or less active users towards universally appreciated choices, ensuring a reliable recommendation quality when personalized data is sparse.

Moreover, the implementation tailored for new users highlights our system's adaptability, providing meaningful engagement from the outset. This approach is crucial for maintaining user interest and encouraging deeper exploration of the platform, which is particularly important in a competitive digital landscape where first impressions are vital.

The integration of these methods into a single, cohesive system using Flask demonstrates a practical application of theoretical concepts to real-world scenarios. It showcases the flexibility and scalability of our recommendations, making the system robust against various user behaviors and preferences. This adaptability is pivotal for the evolving nature of user interactions and preferences in digital platforms.

Furthermore, this thesis has laid the groundwork for future enhancements. The modular nature of the implementation invites further refinement and integration of more sophisticated algorithms, such as machine learning models that predict user preferences with even greater accuracy and dynamism. Future work could also explore deeper personalization aspects, considering more diverse user attributes and behavioral patterns to enhance recommendation relevance and user satisfaction.

In conclusion, the work conducted provides a strong foundation for developing advanced recommendation systems that are not only reactive but also proactive in understanding and catering to user needs. This research contributes to the broader field of data analytics and system design, offering insights that can be applied across various domains where recommendation systems play a crucial role.

6.4 Future Scope

The future scope of this thesis on food recommendation systems extends into several promising areas, reflecting the dynamic and evolving nature of technology and user engagement strategies. As the field of recommendation systems continues to grow, the potential for incorporating more advanced techniques and technologies to enhance user experience and system performance becomes increasingly feasible and necessary.

1. **Incorporation of Advanced Machine Learning Models:** The next steps could involve integrating more sophisticated machine learning algorithms such as deep

learning, which could further improve the accuracy and personalization of the recommendations. These models can utilize a wider array of user data, including real-time interactions and multi-dimensional user profiles, to offer even more tailored suggestions.

2. **Expanding Data Sources:** Expanding the data sources to include more diverse aspects such as user social media activity, geographical data, and contextual information can provide a more holistic view of user preferences. This expanded data set could be used to refine user profiles and enhance the predictive power of the recommendation algorithms.
3. **Real-time Recommendations:** Developing capabilities for real-time recommendations based on live user data and immediate feedback could significantly enhance user engagement. This approach would allow the system to adapt quickly to changes in user preferences and external factors, such as seasonal food trends or new dietary guidelines.
4. **Interactive User Interfaces:** Future work could also focus on enhancing the user interface to make it more interactive. Features like user feedback loops, where users can indicate their satisfaction with the recommendations, could be used to continuously refine and improve the recommendation process.
5. **Cross-platform Integration:** Extending the recommendation system to work seamlessly across different platforms and devices would increase its accessibility and utility. This could include integrating with smart home devices, mobile apps, and even virtual assistants, providing users with recommendations in various contexts and formats.
6. **Ethical and Privacy Considerations:** As data-driven technologies continue to advance, addressing ethical and privacy concerns will be paramount. Future developments should include robust mechanisms to protect user data and ensure transparency in how data is used within the recommendation algorithms.

By addressing these areas, the recommendation system can continue to evolve, providing more value to users and maintaining relevance in a rapidly changing technological landscape. These enhancements will not only improve the system's performance but also ensure it remains a beneficial and trusted tool for users navigating their dietary choices and preferences.