الجامعة المصرية اليابانية للعلوم والتكنولوجيا
エ ジ プ ト 日 本 科 学 技 術 大 学
EGYPT-JAPAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CSE 434 Machine Learning

Food Recommendation System

Hager Tamer AbdAlfattah
120210092
Aya Elsheshtawy Mansoub Elbereky
120210140
Noura Moustafa Maklad
120210150

Submitted for
Prof.Walid Gomaa

1 Introduction

The increasing diversity of culinary options and growing interest in personalized food suggestions have fueled the development of recommendation systems within the food industry. These systems utilize algorithms to provide users with tailored recommendations based on individual preferences. By analyzing user data, they identify patterns and deliver results catering to specific needs and interests. In today's digital age, where users face information overload, recommendation systems play a crucial role in simplifying choices and facilitating informed decision-making.

Food recommender systems analyze user preferences, dietary needs, and other relevant factors to generate personalized recommendations. These systems enhance decision-making and improve user satisfaction by aligning suggestions with individual tastes and requirements. Combining data-driven methodologies and user-centric design, they have been successfully implemented across various domains, including e-commerce, streaming services, and food delivery applications. Within the food sector, these systems streamline selection from a vast array of options by considering dietary restrictions, nutritional needs, and user feedback.

2 Methodology

2.1 Data Preprocessing

The dataset comprises four files:

1. **pp_recipes:** Contains recipe information (ID, name tokens, ingredient tokens, etc.).
2. **pp_users:** Contains user information (techniques, items, ratings, etc.).
3. **raw_interactions:** Stores user-recipe interactions (user ID, recipe ID, date, rating, review).
4. **raw_recipes:** Contains detailed recipe information (name, ID, preparation time, etc.).

Preprocessing steps:

- **Irrelevant Column Removal:** Removed unnecessary columns from each DataFrame.

- **Duplicate Removal:** Removed duplicate rows.
- **Nutritional Information Splitting:** Split the `nutrition` column into individual components.
- **Data Cleaning:** Cleaned the columns to improve data quality
- **Active User Filtering:** Filtered users with fewer than 10 interactions.
- **Normalization:** Normalized the nutritional columns.
- **Outlier Removal:** Removed existing outliers to improve data quality.
- **ID Mapping:** Mapped user and recipe IDs to continuous indices.
- **Interaction Data Preparation:** Created arrays for the sparse interaction matrix.
- **Sparse Matrix Creation:** Created a sparse matrix to store ratings.

```
df_merged_sorted = pd.read_csv('/content/df_merged_sorted.csv')
# df_merged_sorted = df_merged_sorted.head(10000)
df_merged_sorted
```

	user_id	recipe_id	rating	name	calories	total fat	sugar	sodium	protein	saturated fat	carbohydrates (PDV)
0	1	39499	5	kerrieschotel meat and rice dish flavored wit...	0.001170	0.001397	0.000036	0.001636	0.007631	0.003848	0.000554
1	1	78391	5	cheesy charlies	0.000799	0.001164	0.000033	0.000920	0.005647	0.002213	0.000332
2	1	27789	4	ham and swiss in puff pastry	0.000516	0.001455	0.000006	0.000136	0.004731	0.004233	0.000000
3	1	63598	4	pepper steak fettuccine	0.001353	0.001630	0.000102	0.002113	0.011142	0.003848	0.000637
4	1	81473	5	broccoli with cheddar vinaigrette	0.000489	0.001455	0.000022	0.000307	0.003053	0.003271	0.000055
...	...	...	...	...	...	...	...	...	...	...	...
999995	206712	203755	0	chocolate lace cookies	0.002962	0.005820	0.001268	0.001534	0.005342	0.015488	0.001579
999996	206713	393600	5	pumpkin bread less sugar and less oil still...	0.000457	0.000582	0.000168	0.000443	0.000916	0.000481	0.000277
999997	206714	431399	4	stupid simple sugar cookies	0.000430	0.000698	0.000182	0.000136	0.000458	0.002405	0.000222
999998	206715	516662	5	black bean chocolate cake	0.000402	0.000815	0.000130	0.000273	0.001526	0.003175	0.000166
999999	206716	63786	5	steak or chicken fajitas	0.000316	0.000466	0.000022	0.000477	0.000763	0.000481	0.000166

1000000 rows × 11 columns

Table 1: Data Overview After Preprocessing

3 Recommendation System

3.1 Recommendation Techniques

This system uses different recommendation techniques for new and existing users:

3.1.1 New Users

- **Nutrient-Based Recommendations:** Recommends recipes based on general dietary preferences (e.g., high-protein, low-carb) by analyzing nutrient profiles and using cosine similarity to compare recipes based on their nutritional composition.

- **Top-Rated Recipes:** Recommends the most popular recipes based on aggregated user ratings, leveraging collaborative filtering.

3.1.2 Existing Users

- **Nutrient-Based Recommendations:** Similar to new users, but fine-tuned based on the user's past preferences and dietary goals using cosine similarity.

- **Top-Rated Recipes:** Recommends popular recipes based on overall user ratings using collaborative filtering.

- **Preference-Based Recommendations:** Personalized recommendations based on the user's history of liked/rated recipes, using collaborative filtering with K-Nearest Neighbors (KNN) to find recipes liked by similar users.

3.2 Algorithms and Metrics

3.2.1 Cosine Similarity

Cosine similarity measures the similarity between items or user profiles by calculating the cosine of the angle between two vectors representing their features (e.g., nutrient composition, user preferences):

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

where A and B are vectors representing items or user preferences.

3.2.2 K-Nearest Neighbors (KNN)

KNN identifies the nearest neighbors based on similarity metrics like cosine similarity, ensuring recommendations are derived from similar users or items.

3.3 Recommendation Logic

The system follows this logic:

1. **ID Mapping:** Maps IDs to continuous indices.
2. **Approximate Nearest Neighbors (KNN):** Finds similar users.
3. **Top-Rated Recommendations:** Recommends highest-rated recipes (Figure 1).

```python
def recommend_top_rated_recipes(num_recommendations=5):
    """
    Recommend a random selection of top-rated recipes with average ratings >= 4.
    """
    # Calculate the average rating for each recipe
    recipe_ratings = df_merged_sorted.groupby('recipe_id')['rating'].mean()
    # Filter recipes with an average rating of 4 or higher
    top_recipes = recipe_ratings[recipe_ratings >= 4].index
    # Filter the dataset to include only recipes with high average ratings
    top_rated_recipes = df_merged_sorted[df_merged_sorted['recipe_id'].isin(top_recipes)]
    # Remove duplicates to ensure each recipe is listed only once
    top_rated_recipes_unique = top_rated_recipes.drop_duplicates(subset=['recipe_id'])
    # Select only the required columns
    selected_columns = ['recipe_id', 'name', 'rating', 'calories', 'total fat', 'sugar', 'sodium', 'protein', 'saturated fat']
    top_rated_recipes_selected = top_rated_recipes_unique[selected_columns]
    # Calculate the average rating for each recipe and add it as a new column
    top_rated_recipes_selected['average_rating'] = top_rated_recipes_selected.groupby('recipe_id')['rating'].transform('mean')
    # Drop the individual rating column and keep the average rating
    top_rated_recipes_final = top_rated_recipes_selected.drop(columns=['rating'])
    # Select the top 15 rated recipes
    top_15_recipes = top_rated_recipes_final.head(15)
    # Randomly sample the specified number of recipes from the top 15
    recommendations = top_15_recipes.sample(n=num_recommendations, random_state=None)
    return recommendations
```

Figure 1: Code for top-rated recipes

4. **Nutrition-Based Recommendations:** Recommends recipes based on nutritional preferences (Figure 2).

```python
def recommend_recipes_based_on_nutrition(preferences, num_recommendations=5):
    if len(preferences) != len(nutrition_columns):
        raise ValueError(f"Expected {len(nutrition_columns)} preferences, got {len(preferences)}")

    # Calculate preference scores by computing the Euclidean distance
    preference_scores = df_merged_sorted[nutrition_columns].apply(
        lambda x: -np.linalg.norm(x - preferences), axis=1
    )

    # Get the top recommended indices based on preference scores
    recommendations = np.argsort(preference_scores)[-num_recommendations:][::-1]
    # Select the recommended recipes from the dataset
    recommended_recipes = df_merged_sorted.iloc[recommendations]
    # Select only the required columns
    selected_columns = ['recipe_id', 'name', 'rating', 'calories', 'total fat', 'sugar', 'sodium', 'protein', 'saturated fat']
    recommended_recipes_selected = recommended_recipes[selected_columns]
    # Calculate the average rating for each recipe and add it as a new column
    recommended_recipes_selected['average_rating'] = recommended_recipes_selected.groupby('recipe_id')['rating'].transform('mean')
    # Drop the individual rating column and keep the average rating
    recommended_recipes_final = recommended_recipes_selected.drop(columns=['rating'])
    # Remove duplicates based on recipe_id to ensure unique recommendations
    recommended_recipes_final_unique = recommended_recipes_final.drop_duplicates(subset=['recipe_id'])
    return recommended_recipes_final_unique
```

Figure 2: Code for Recommended recipes based on nutritions

5. **Similarity-Based Recommendations:** Recommends recipes based on similar user ratings (Figure 3).

```python
def recommend_recipes_based_on_similarity(user_id, num_recommendations=5, rating_threshold=4):
    if user_id not in user_id_map:
        raise ValueError("User ID not found.")
    user_idx = user_id_map[user_id]
    # Compute cosine similarity between the user and all other users
    user_interactions = interaction_matrix_sparse[user_idx]
    similarities = cosine_similarity(user_interactions, interaction_matrix_sparse).flatten()
    similarities[user_idx] = 0
    similar_users = np.argsort(-similarities)[:num_recommendations]
    similar_user_ratings = interaction_matrix_sparse[similar_users].toarray()
    # Compute the weighted average ratings for unrated recipes
    user_interactions_array = user_interactions.toarray().flatten()
    mask = (user_interactions_array == 0)  # Mask for unrated recipes
    weighted_ratings = np.zeros_like(user_interactions_array)
    for i, is_unrated in enumerate(mask):
        if is_unrated:
            # Get the ratings of similar users for this recipe
            weighted_ratings[i] = np.dot(similar_user_ratings[:, i], similarities[similar_users]) / (np.sum(similarities[similar_users]) + 1e-5)
    # Sort the unrated recipes based on the weighted ratings
    recommendations = np.argsort(-weighted_ratings)
    # Map the recommendations back to recipe IDs
    recommended_recipe_ids = [list(recipe_id_map.keys())[idx] for idx in recommendations]
    recommended_recipes_with_ratings = df_merged_sorted[df_merged_sorted['recipe_id'].isin(recommended_recipe_ids)]
    top_rated_recipes = recommended_recipes_with_ratings[recommended_recipes_with_ratings['rating'] >= rating_threshold]
    top_rated_recipes_unique = top_rated_recipes.drop_duplicates(subset=['recipe_id'])
    top_rated_recipes_unique = top_rated_recipes_unique.copy()
    top_rated_recipes_unique['average_rating'] = top_rated_recipes_unique.groupby('recipe_id')['rating'].transform('mean')
    selected_columns = ['recipe_id', 'name', 'average_rating', 'calories', 'total fat', 'sugar', 'sodium', 'protein', 'saturated fat']
    top_rated_recipes_final = top_rated_recipes_unique[selected_columns]
    # Select the top 20 recipes by average rating
    top_20_recipes = top_rated_recipes_final.sort_values(by='average_rating', ascending=False).head(20)
    # Randomly select 5 recipes from the top 20
    random_recommendations = top_20_recipes.sample(n=5, random_state=42)
    return random_recommendations
```

Figure 3: Code for Recommended recipes based on similar user ratings

4 Results

4.1 Detailed Evaluation Metrics

The following metrics were computed to evaluate the performance of the model:

- Mean Absolute Error (MAE): **0.9119**
- Mean Squared Error (MSE): **1.9410**
- Root Mean Squared Error (RMSE): **1.3932**
- Precision: **0.8827**
- Recall: **0.9876**
- F1-Score: **0.9322**

Visualization of Evaluation Metrics

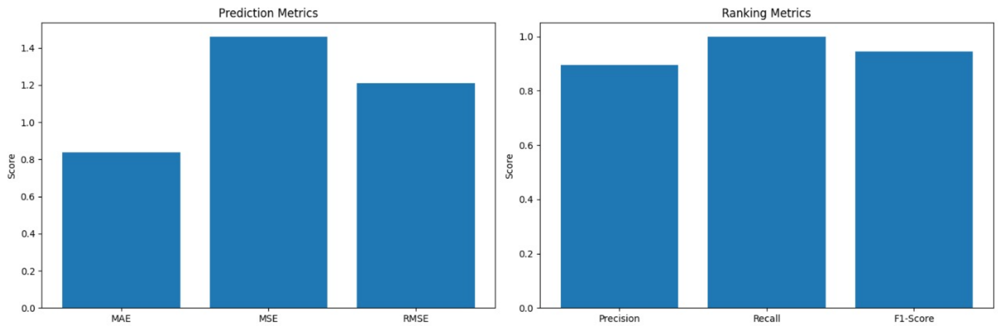

Figure 4: A visual representation of the evaluation metrics.

Prediction Statistics

The statistical summary of the predicted ratings is as follows:

Statistic	**Value**
Mean Predicted Rating	4.37
Standard Deviation	0.46
Minimum Predicted Rating	0.00
Maximum Predicted Rating	4.42

Table 2: Summary statistics of the predicted ratings.

4.2 figure of the output

4.2.1 New User

If the user is new and wants recommendations based on top-rated items (Figure 5):

Recommendations based on top-rated recipes:

	recipe_id	name	calories	total fat	sugar	sodium	protein	saturated fat	average_rating
17	17387	pizza breadsticks	0.000969	0.000873	0.000014	0.000920	0.003663	0.000770	5.0
6	52077	spicy banana fritters zitumbuwa	0.000275	0.000349	0.000050	0.000136	0.000916	0.000289	5.0
16	68986	apricot mousse	0.000544	0.000233	0.000433	0.000375	0.003816	0.000673	5.0
14	14807	californian apple crunch	0.000664	0.001862	0.000229	0.000170	0.000305	0.006349	5.0
2	27789	ham and swiss in puff pastry	0.000516	0.001455	0.000006	0.000136	0.004731	0.004233	4.0

Figure 5: Snippet answer based on top-rated items

If the user is new and wants recommendations based on nutrition (Figure 6):

Recommendations based on nutritional preferences:

	recipe_id	name	calories	total fat	sugar	sodium	protein	saturated fat	average_rating
455710	264904	easy caramel frosting	0.005281	0.001571	0.005795	0.000750	0.001526	0.005195	4.666667
80513	913	ritz mock apple pie iii	0.003735	0.002037	0.003374	0.001091	0.001832	0.003271	5.000000
194823	81289	chocolate lemonade	0.004060	0.001280	0.003937	0.000784	0.002747	0.002982	4.000000

Figure 6: Snippet answer based on nutrition

4.2.2 Existing User

If the user has an ID, one more choice will be added, allowing them to select whether they want recommendations based on top-rated recipes, nutrition, or ratings from other similar users.

If the user is existing and wants recommendations based on ratings from other users (Figure 7):

```
Are you a new user or an existing user? (new/existing): existing
Enter your user ID: 10
Would you like recommendations based on:
1. Nutritional preferences
2. Top-rated recipes
3. Similarity between ratings
Enter your choice (1/2/3): 3
Recommendations based on similarity between ratings:
        recipe_id                                              name  \
0           39499  kerrieschotel  meat and rice dish flavored wit...
321011     365585                               lemon dill broiled fish
321370     203973                                 crunchy chicken salad
321164     148569                                easy decadent truffles
321241      85381                            awesome creamy coconut dip

        average_rating  calories  total fat     sugar    sodium   protein  \
0                  5.0  0.001170   0.005541  0.000036  0.003273  0.007631
321011             5.0  0.000425   0.003002  0.000000  0.000614  0.007631
321370             5.0  0.000990   0.010159  0.000039  0.001568  0.009005
321164             5.0  0.000178   0.001154  0.000113  0.000000  0.000153
321241             5.0  0.008605   0.093050  0.003647  0.002796  0.007326

        saturated fat
0            0.005818
321011       0.001164
321370       0.003636
321164       0.001600
321241       0.151564
```

Figure 7: Snippet answer based on ratings from similar users

5 Conclusion

This report details a food recommendation system using content-based and collaborative filtering. The system uses data preprocessing, KNN, and cosine similarity for personalized recommendations. Future work includes more sophisticated filtering, UI improvements, and real-time updates.