Title

Nutritional Profile Matching of Foods Using Proportional Similarity *A Mini-Project Report*

Aim

To design a simple, interpretable method for comparing foods based on the **proportion** of their nutritional components (calories, protein, carbohydrates, fat) rather than their absolute quantities, and to find the top five foods most similar to a user's input profile.

Abstract

This project presents a proportional-similarity approach to match foods based on their nutritional composition. Each food item's nutrients are normalized into ratio vectors, which are then compared to a user-defined input vector. The total absolute difference is used as a distance metric to rank similarity. Data were taken from an Excel file version of the *Food Nutrition Composition Database 2024*. The report explains the data preprocessing steps, the similarity computation method, and discusses limitations and possible improvements. The code implementation is fully reproducible in Python using `pandas`, `numpy`, and `scipy`.

1. Introduction

1.1 Problem Statement and Context

Dietary analysis and meal recommendation systems often require comparing or substituting foods based on their nutritional composition. Comparing foods by **absolute nutrient values** can be misleading due to serving size differences. Instead, comparing foods by their **nutrient ratios** provides a more meaningful measure of structural similarity between foods.

1.2 Background and Theoretical Framework

Let each food item be represented by a nutrient vector

$$\mathbf{f} = (c, p, h, f)$$

where (c, p, h, f) denote calories, protein, carbohydrates, and fat, respectively. Each vector is normalized into a ratio representation:

$$\mathbf{r}_f = \frac{\mathbf{f}}{\sum_j f_j}.$$

For the user's input vector $\mathbf{u}$, we compute the same ratio $\mathbf{r} * u$. The distance between the two ratio vectors is measured using the L1 (Manhattan) distance:

$$\text{cost}(f) = \sum_{j \in \{c,p,h,f\}} |r_{f,j} - r_{u,j}|.$$

A smaller cost indicates a more similar nutrient ratio.

---

2. Objectives

1. To design a proportion-based food similarity system.
2. To demonstrate how to load, clean, and process nutritional data from an Excel source.
3. To implement and test the algorithm with real data.
4. To discuss limitations and suggest improvements for future research.

---

3. Methods

3.1 Research Design

This is a **data-driven computational study** that analyzes existing nutritional data. No human or animal subjects are involved.

3.2 Materials / Data

- **Dataset:** `食品營養成分資料庫2024UPDATE2.xlsx` (Food Nutrition Composition Database 2024).
- **Key columns used:** `Food Category`, `Sample Name`, `Common Name`, `Calories (kcal)`, `Protein (g)`, `Fat (g)`, and `Carbohydrates (g)`.
- Records with missing values in these columns were removed.

3.3 Procedures

1. Load the Excel file and extract the required columns.
2. Remove entries with missing nutritional data.
3. Construct a dictionary `FOOD_DATA` where each key is the food name, and the value is a dictionary of its four nutrients.
4. Accept user input as a nutrient vector.
5. Compute ratio vectors for both the user input and each food item.
6. Calculate absolute differences and rank foods by total distance.
7. Return the top five foods with the smallest distances.

Key implementation snippet:

```
user_vec = np.array([calories, protein, carbs, fat])
user_vec[user_vec == 0] = 1e-6
user_ratio = user_vec / user_vec.sum()

food_ratios = food_matrix / food_matrix.sum(axis=1, keepdims=True)
cost_matrix = np.abs(food_ratios - user_ratio)
total_cost = cost_matrix.sum(axis=1)
top5 = np.argsort(total_cost)[:5]
```

3.4 Data Analysis

- **Primary Metric:** The total L1 distance between ratio vectors.
- **Alternative Metrics (optional):** Cosine similarity, Euclidean distance, or KL divergence (careful handling of zero values required).
- If the task involves multiple-to-multiple substitutions, the **Hungarian algorithm** can be applied to find the globally optimal pairing.

3.5 Ethical Considerations

The dataset contains only publicly available, non-personal information, so no ethical approval is required. If used with personal dietary data in the future, privacy and ethical guidelines must be followed.

4. Results

> *Note:* Since I cannot access your local Excel file, below is a **result template** showing how outputs would appear after running the program.

Rank	Food Name	Distance	Calories (kcal)	Protein (g)	Carbs (g)	Fat (g)
1	Example A	0.0173	360	9.0	76.0	1.5
2	Example B	0.0218	370	8.5	78.2	2.0
3	Example C	0.0341	355	8.0	75.5	1.9
4	Example D	0.0412	380	10.2	72.0	2.8
5	Example E	0.0475	345	7.8	79.0	1.3

Example user input:

```
user_input = {
    "calories": 365,
    "protein": 8.6,
    "carbs": 77.1,
    "fat": 1.6
}
```

The printed output lists the top five most similar foods with their corresponding distance and nutrient values.

5. Discussion

5.1 Interpretation of Results

A smaller total cost indicates that the food's **nutrient proportions** closely resemble the user's input. For instance, foods ranked high are likely to share a high-carbohydrate, low-fat profile if the input corresponds to such a distribution.

5.2 Comparison with Other Methods

- **Absolute value difference:** Sensitive to serving size differences.
- **Cosine similarity:** Measures direction similarity and works well with proportions.
- **KL divergence:** More expressive but requires smoothing for zero values. This study uses the L1 distance for simplicity and interpretability.

5.3 Limitations

1. **Data quality dependency:** Accuracy relies on the original dataset.
2. **Limited nutrient scope:** Only macronutrients are considered; vitamins and minerals are ignored.
3. **Category imbalance:** The dataset may not equally represent all food types.
4. **Single distance measure:** No weighting to emphasize specific nutrients.

5.4 Implications and Future Research

- Introduce **weighted ratios** to emphasize certain nutrients (e.g., protein-rich diets).
- Extend the dataset to include micronutrients and preparation methods.
- Combine with user preferences or dietary restrictions.
- Apply the Hungarian algorithm for optimal multi-item substitutions in recipe design.

---

6. Conclusion

6.1 Summary of Key Findings

The proposed method successfully computes and ranks foods by **nutrient proportion similarity** using a simple L1 distance measure.

6.2 Research Contribution

This work demonstrates a clear and practical framework for food comparison based on structural nutrient composition rather than absolute quantities.

6.3 Final Thought

Proportion-based similarity offers an intuitive way to identify nutritionally comparable foods. While simple, it can serve as a foundation for more advanced dietary recommendation systems.

References

- User-provided code and implementation.
- *Food Nutrition Composition Database 2024 (Excel version).*

- SciPy and NumPy documentation on matrix operations and linear assignment algorithms.

Appendix A — Full Code Implementation

```
import pandas as pd
import numpy as np
from scipy.optimize import linear_sum_assignment

FILE_PATH = "食品營養成分資料庫2024UPDATE2.xlsx"
columns_needed = [
    "食品分類","樣品名稱","俗名","熱量(kcal)", "修正熱量(kcal)", "水分(g)", "粗蛋白(g)", "粗脂肪(g)",
    "飽和脂肪(g)", "灰分(g)", "總碳水化合物(g)", "膳食纖維(g)"
]

df = pd.read_excel(FILE_PATH, usecols=columns_needed)
df = df.dropna(subset=["熱量(kcal)", "粗蛋白(g)", "粗脂肪(g)", "總碳水化合物(g)"])

FOOD_DATA = {}
for _, row in df.iterrows():
    name = str(row["俗名"]).strip() if not pd.isna(row["俗名"]) else str(row["樣品名稱"]).strip()
    FOOD_DATA[name] = {
        "calories": float(row["熱量(kcal)"]),
        "protein": float(row["粗蛋白(g)"]),
        "carbs": float(row["總碳水化合物(g)"]),
        "fat": float(row["粗脂肪(g)"]),
    }

def find_top5_similar_foods(user_input, food_data):
    user_vec = np.array([
        user_input["calories"],
        user_input["protein"],
        user_input["carbs"],
        user_input["fat"]
    ], dtype=float)
    user_vec[user_vec == 0] = 1e-6
    user_ratio = user_vec / user_vec.sum()

    food_names = list(food_data.keys())
    food_matrix = np.array([
        [f["calories"], f["protein"], f["carbs"], f["fat"]] for f in food_data.values()
    ], dtype=float)
    food_ratios = food_matrix / food_matrix.sum(axis=1, keepdims=True)

    cost_matrix = np.abs(food_ratios - user_ratio)
    total_cost = cost_matrix.sum(axis=1)
    top5_idx = np.argsort(total_cost)[:5]
    return [(food_names[i], total_cost[i], food_data[food_names[i]]) for i in
top5_idx]

user_input = {"calories": 365, "protein": 8.6, "carbs": 77.1, "fat": 1.6}
top5 = find_top5_similar_foods(user_input, FOOD_DATA)

for rank, (name, diff, info) in enumerate(top5, start=1):
    print(f"{rank}. {name} (distance: {diff:.4f}) -> "
          f"kcal {info['calories']}, protein {info['protein']}, "
          f"carbs {info['carbs']}, fat {info['fat']}")
```