Final Report

Team : *61* ***ALPHA bet***
Team Members: *Gopendra Singh - 2022101003, Yash Nitin Dusane - 2022102078*

Introduction

System that takes a recipe text as input and analyzes it to generate a detailed **Health Chart**. The goal is to automatically extract and predict key information from the recipe using text processing and predictive modeling techniques. The system performs four main tasks:

1. **Cuisine Classification** – Identifying the regional or cultural origin of the recipe (e.g., Indian, Italian).

2. **Dietary Category Detection** – Classifying the recipe based on dietary restrictions or preferences, such as vegan, vegetarian, or gluten-free.

3. **Recipe Difficulty Prediction** – Estimating how easy or hard the recipe is to prepare.

4. **Nutritional Value Prediction** – Predicting approximate nutritional content, including calories, fat, protein, and carbohydrates.

Ingredient Identification for Recipe Analysis

Ingredient identification is crucial for tasks like cuisine classification, dietary categorization, and nutritional prediction. We use **Named Entity Recognition (NER)** to extract ingredients from recipe texts, employing two approaches:

1. **Rule-Based NER**: This method uses predefined rules and pattern matching to identify ingredients based on common phrases and measurements (e.g., "1 cup of sugar"). It is accurate in controlled environments but less flexible when dealing with new or complex ingredient variations.

2. **Machine Learning-Based NER (spaCy)**: A machine learning model trained on annotated recipe data learns to recognize ingredients based on context and linguistic patterns. It is more adaptable, can handle variations and new ingredients, but requires a large dataset and computational resources for training.

Rule Based NER:

This is a **rule-based ingredient extractor** that processes recipe text and extracts structured ingredient phrases.
It tries to find, for each ingredient mentioned:

- **Quantity** (e.g., "2")
- **Unit** (e.g., "tablespoons")
- **Modifiers** (e.g., "chopped", "fresh")
- **Food name** (e.g., "onion", "paneer butter", "red chili powder")

It outputs results as a list of strings like "2 tablespoons chopped onion".

PROCEDURE:

1. **Preprocessing the Text**
   - The preprocess_text function uses spaCy to tokenize the text.
   - It removes stopwords (except conjunctions like "and", "or") and keeps only alphabetic tokens or numbers.
   - The result is a **cleaned list of tokens** that are more likely to be relevant to ingredients.

INTERESTING FACT: tokens like "of" or "each" between unit and food name are mandatory to be removed as stopwords as they are the part of ingredient segment generally. Eg. 250 grams of paneer.

2. **Main Extraction Loop**
   - The main function processes the cleaned text using spaCy again to get tokens.
   - It loops through each token in the document, trying to identify the start of an ingredient phrase.

3. **Extracting Quantity and Unit**
   - If the current token is a quantity (like a number or a fraction), it is stored as quantity.

- If the next token is a unit (like "grams", "cups"), it is stored as unit.

4. **Extracting Modifiers**
   - After quantity and unit, the code checks for any modifiers (like "chopped", "fresh") and adds them to the mods list.
   - The index advances for each modifier found.
5. **Extracting the Food Name**
   - The code then tries to chain together as many tokens as possible that are present in the given food lexicon.
   - This allows for **multi-word food names** (like "paneer butter").
   - The tokens are joined to form the food item.
   -
6. **Building the Ingredient Phrase**
   - If a food name is found, the code combines the quantity, unit, modifiers, and food name into a single string (skipping any that are empty).
   - This phrase is added to the ingredients list.

Strengths of This Approach

- **Handles multi-word food names** if they are in the lexicon.
- **Attaches quantity, unit, and modifiers** when present.
- **Ignores irrelevant words** due to preprocessing.

Limitations

- **Strict adjacency**: Quantity and unit must be immediately before the food name.
- **Lexicon dependence**: Only finds food names present in lexicon.
- **No fallback for foods not in lexicon**: If a food name is missing from the lexicon, it won't be extracted.
- **No handling for optional/parenthetical ingredients** or more complex grammatical structures.
- **Repetition of ingredients**

EXAMPLE:

[1] To make Paneer Butter Masala, start by cutting 250 grams of paneer into cubes. In a pan, heat 2 tablespoons of butter and 1 tablespoon of coconut oil. Add a finely chopped onion, 1 tbsp ginger-garlic paste, and 2 pureed tomatoes. Then, stir in 1/2 tsp turmeric powder, red chili powder, coriander powder, and salt. Add 1/4 cup cashew paste and cook for 2-3 minutes. Stir in cream, garam masala, and sugar. Add paneer cubes and simmer.

→ Extracted Ingredients:
['paneer butter masala', '250 grams paneer', '2 tablespoons butter', '1 tablespoon coconut oil', 'chopped onion', '1 tbsp ginger garlic', 'stir', '1/2 tsp turmeric', 'chili', 'coriander', 'salt', '1/4 cup cashew', 'stir cream', 'masala', 'sugar', 'paneer']

FURTHER ADVANCEMENT:

We will be looking at the 1st limitation of the code, which is **strict adjacency** and **Repitition**.

Longest Multi-word Ingredient Matching (FoodIE-style)

- **Initial:** Only chained tokens as long as each token was in the lexicon (all_ingredients), which could miss multi-word ingredients if not all tokens were present in the lexicon as single words.
- **New:**
  - For every position, it tries **all possible spans** (from current token to the end) and matches the **longest possible phrase** (either by text or lemma) in the lexicon.
  - This means it can match "red chili powder" or "paneer butter" as a whole, even if "red", "chili", and "powder" are not all in the lexicon as single words.
  - **Benefit:** More robust and accurate for multi-word and lemmatized ingredient names.

2. **Lemmatized Matching**
- **Initial:** Matched only the exact token text to the lexicon.
- **New:**
  - Matches both the **original token text** and the **lemmatized form** (e.g., "tomatoes" → "tomato").
  - **Benefit:** Handles plural/singular variations and improves recall for ingredients like "tomatoes" vs "tomato".

3. **Preference for Most Complete Phrase**
- **Initial:** Added every ingredient phrase found, which could result in duplicates or less-informative entries (e.g., both "paneer" and "250 grams paneer").
- **New:**
  - Uses a dictionary (results) to **keep only the most complete phrase** for each food item (prefers the longer phrase if multiple is found).
  - **Benefit:** Avoids redundant or partial ingredient phrases, making the output cleaner.

➔ Here the main part is the FoodIE implementation (simplified version)

Source: LINK

- **Chaining:** It groups together adjacent adjectives and nouns, similar to FoodIE's chaining rules.
- **POS-based:** Uses POS tags to decide which tokens to include in a food entity.

Updated Example Output:

--- Recipe 1 ---
Extracted Ingredients:
- Paneer Butter
- 250 grams paneer
- 2 tablespoons butter
- 1 tablespoon coconut oil
- chopped onion
- 1 tbsp ginger garlic paste
- 2 pureed tomatoes
- 1/2 tsp turmeric powder

- red chili powder
- coriander powder
- salt
- 1/4 cup cashew paste
- cream
- garam masala
- sugar

Learning based NER:

The code follows a **standard pipeline** for training a **custom Named Entity Recognition (NER)** model using **spaCy** to detect INGREDIENT entities in text.

1. **Data Loading and Preprocessing**:
    - **Load Data**: Training data is read from a CSV file containing text and entities. The entities column contains annotations (start, end, label) indicating where in the text an entity appears.
    - **Remove Overlapping Entities**: Overlapping entities are filtered out to ensure that only non-conflicting entities are used in training, preventing model confusion. This step sorts entities by their start positions and keeps the first non-overlapping entity at each position.

2. **Model Setup**:
    - **Pre-trained Model**: The pre-trained **spaCy model** (en_core_web_sm) is loaded, which includes basic NLP components (tokenizer, POS tagger, dependency parser).
    - **Customize NER**: A custom label (INGREDIENT) is added to the NER pipeline so the model can recognize and classify ingredients in text.

3. **Training Process**:
    - **Disable Unnecessary Components**: Only the **NER** component is trained by disabling all other components (e.g., parser, tagger) to focus resources on entity recognition.
    - **Optimizer and Minibatches**: The model is fine-tuned using an **optimizer**. Training is done using **minibatches** (grouped examples of text), where the batch size increases gradually to balance performance and memory. The compounding method adjusts the batch size dynamically, starting from 4 and growing to 32.
    - **Dropout Regularization**: Dropout (set to 0.3) is applied during training to prevent overfitting by randomly disabling a fraction of model connections in each update.
    - **Training Iterations**: The training loop runs for **10 iterations** where the model continuously improves by processing batches of text, learning to predict INGREDIENT entities and adjusting the model's weights based on the computed **loss**.

4. **Model Evaluation**:
    - **Loss Tracking**: The loss is calculated after each iteration and printed. This helps monitor the model's progress and whether it is learning effectively. Lower loss indicates better predictions

5. **Model Saving**:
    - After training, the model is saved to disk in a directory (ingredient_ner_model). This allows the trained model to be reused later for inference on new text or for further fine-tuning.

Recipe Classification

Classification using BERT Model:

Classification based on cuisines :

We have implemented classifying recipes into different cuisines using a **BERT (Bidirectional Encoder Representations from Transformers)** model. The focus is on leveraging NLP techniques to process recipe instructions and predict their respective cuisine labels.

1. Data Preprocessing

- The dataset contains recipe instructions and their corresponding cuisine labels. Missing values are removed, and the cuisine labels are encoded into numeric values using **LabelEncoder** to make them suitable for the model.
- The dataset is split into training (80%) and validation (20%) sets to evaluate the model's performance.

2. Text Tokenization

- **BERT Tokenizer**: The BERT tokenizer converts the recipe instructions into a format that BERT can understand. The instructions are tokenized into word pieces and padded to a fixed length (128 tokens).
- A custom Dataset class is used to handle tokenization and batching, ensuring the input data is properly processed for the BERT model.

3. Model Architecture

- The model uses **BERT** (bert-base-uncased) for text representation. The BERT model is pre-trained on a large corpus and is fine-tuned for the classification task.
- The model adds a **fully connected layer** on top of BERT's output to predict the cuisine labels. A **dropout layer** is included to reduce overfitting.

4. Training

- The model is trained using the **AdamW optimizer** with a low learning rate (2e-5) and **cross-entropy loss**, a common choice for multi-class classification tasks.
- During each epoch, the model processes batches of tokenized instructions, computes the loss, and updates its parameters.

5. Evaluation

- The model's performance is evaluated using the **classification report** from scikit-learn, which calculates metrics like precision, recall, and F1-score for each cuisine class.

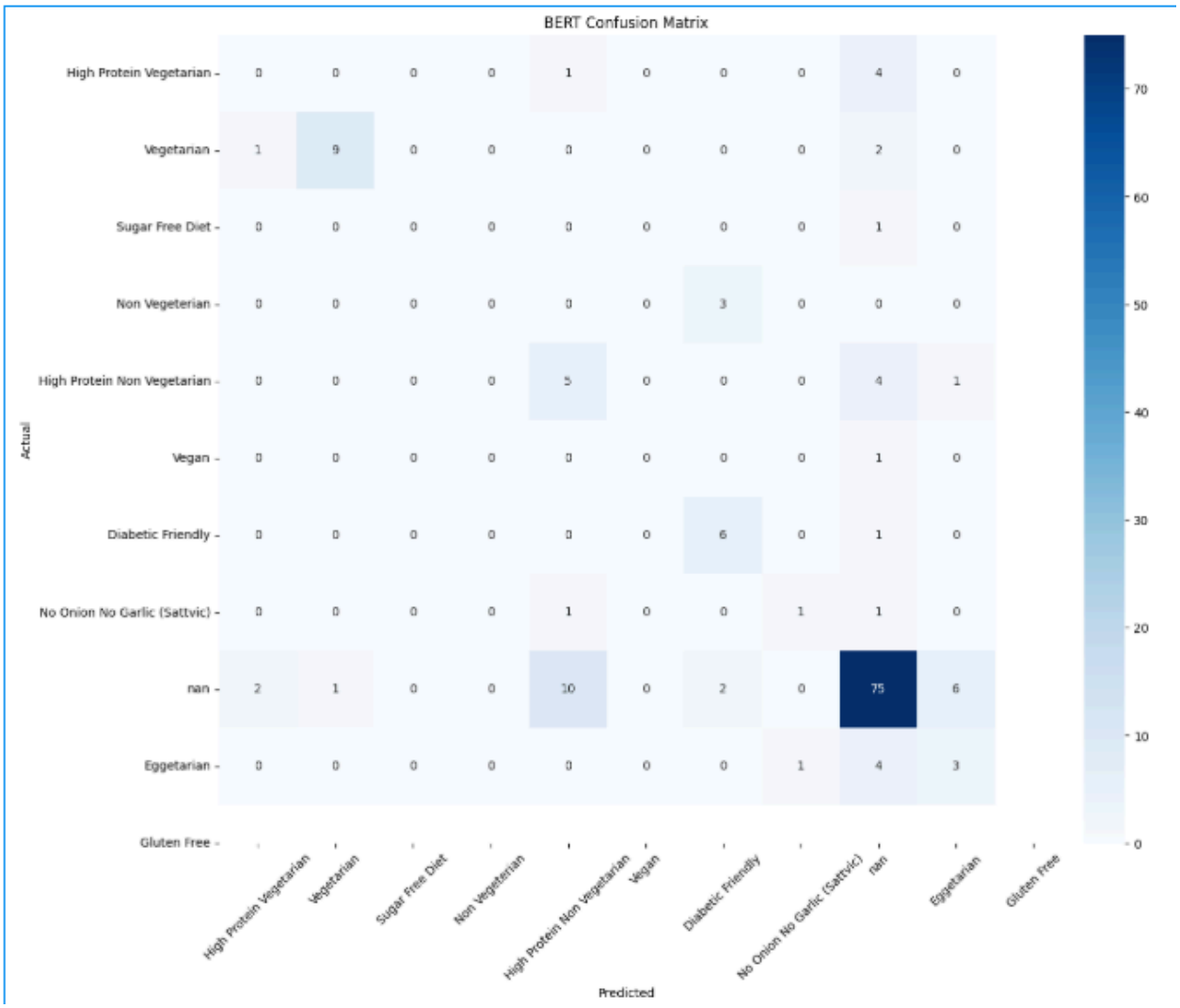
Similar models are implemented for Dietary classification and recipe difficulty classifier. Following are the results of the model outputs:

Cuisine Classifier Output:

```
Training Loss: 0.0594
100%|██████████| 75/75 [00:12<00:00,  6.05it/s]
```

	precision	recall	f1-score	support
Afghan	0.00	0.00	0.00	0
African	0.50	1.00	0.67	3
American	0.00	0.00	0.00	0
Andhra	0.89	0.80	0.84	20
Appetizer	0.00	0.00	0.00	1
Arab	0.00	0.00	0.00	3
Asian	0.14	0.17	0.15	12
Assamese	1.00	0.50	0.67	2
Awadhi	0.58	0.70	0.64	10
Bengali Recipes	0.85	0.74	0.79	39
Bihari	0.67	0.67	0.67	3
British	0.00	0.00	0.00	3
Brunch	0.00	0.00	0.00	0
Burmese	0.00	0.00	0.00	0
Cantonese	0.00	0.00	0.00	1
Caribbean	0.00	0.00	0.00	0
Chettinad	0.91	0.67	0.77	15
Chinese	0.40	0.46	0.43	13
Coastal Karnataka	0.00	0.00	0.00	2
Continental	0.79	0.80	0.79	176
Coorg	0.33	1.00	0.50	1
Dessert	0.00	0.00	0.00	0
Dinner	0.00	0.00	0.00	0
...				
accuracy			0.61	1188
macro avg	0.39	0.37	0.37	1188
weighted avg	0.61	0.61	0.61	1188

Classification based on Dietary category



BERT Model Performance:

Accuracy: 0.6781

Classification Report:

	precision	recall	f1-score	support
Diabetic Friendly	0.00	0.00	0.00	5
Eggetarian	0.90	0.75	0.82	12
...				
accuracy			0.68	146
macro avg	0.33	0.36	0.34	146
weighted avg	0.68	0.68	0.67	146

Classification based on TF IDF with Regression:

Classification based on dietary category :

Model Performance Metrics:

Accuracy: 0.6507

Classification Report:

	precision	recall	f1-score	support
Diabetic Friendly	0.33	0.20	0.25	5
Eggetarian	0.63	1.00	0.77	12
Gluten Free	0.00	0.00	0.00	1
High Protein Non Vegetarian	1.00	0.33	0.50	3
High Protein Vegetarian	0.32	0.80	0.46	10
No Onion No Garlic (Sattvic)	0.00	0.00	0.00	1
Non Vegeterian	0.86	0.86	0.86	7
Sugar Free Diet	0.00	0.00	0.00	0
Vegan	1.00	0.33	0.50	3
Vegetarian	0.86	0.65	0.74	96
nan	0.24	0.50	0.32	8
accuracy			0.65	146
macro avg	0.48	0.42	0.40	146
weighted avg	0.75	0.65	0.67	146

Classification based on difficulty level:

Accuracy: 0.60

Classification Report:

	precision	recall	f1-score	support
easy	0.75	0.25	0.38	36
hard	0.50	0.13	0.21	30
medium	0.60	0.95	0.73	78
accuracy			0.60	144
macro avg	0.62	0.44	0.44	144
weighted avg	0.61	0.60	0.53	144

Thus, we have observed different classifications using different embedding methods.

Moving on further to Nutritional Analysis of Recipe.

Nutritional Analysis of Recipe

Method 1:

Data Preparation:

- Reads a CSV of recipes (recipes.csv).
- Splits the ingredients string into a list of lowercase ingredient names.
- Parses nutrition strings into a dictionary of nutrient-value pairs.

Cleaning:

- Filters out rows with missing nutrition data or empty ingredient lists.

Feature Encoding:

- Uses MultiLabelBinarizer to **one-hot encode** ingredients:
  - Each recipe becomes a binary vector (1 = ingredient present, 0 = not).
  - Example: If ['salt', 'sugar'] are the only ingredients in a recipe, it becomes [1, 1, 0, ...].

Targets:

- Extracts nutrition columns like Total Fat, Protein, Cholesterol, etc.
- Standardizes the target values using StandardScaler.

Model Training:

- Splits data into training and test sets (80/20).
- Trains a **Random Forest Regressor** to predict nutrition values from binary ingredient vectors.

Prediction & Evaluation:

- Predicts on the test set.
- Inverses the target scaling to get actual nutrient values.
- Calculates:
  - MSE (Mean Squared Error)
  - RMSE (Root Mean Squared Error)
  - MAE (Mean Absolute Error)
  - Accuracy within ±10 units per nutrient

OUTPUT:

MSE: 26268.74
RMSE: 162.08
MAE: 34.58

Accuracy within ±10:
Overall: 64.57%
Per nutrient:
  Total Fat: 77.48%
  Saturated Fat: 96.69%
  Cholesterol: 37.75%
  Sodium: 11.92%
  Total Carbohydrate: 43.71%
  Dietary Fiber: 100.00%
  Total Sugars: 60.26%
  Protein: 88.74%

For Recipe 1:
Predicted Nutrition:
Total Fat: 6.39
Saturated Fat: 3.34
Cholesterol: 8.60
Sodium: 105.18
Total Carbohydrate: 21.51
Dietary Fiber: 1.48
Total Sugars: 13.21
Protein: 2.64

Method 2:

BERT Embeddings: It uses bert-base-uncased to convert ingredient names (e.g., "flour") into dense vector embeddings.

Quantity-Aware Embeddings: Each ingredient's embedding is concatenated with its quantity (e.g., "2 cups flour" → "flour", 2.0, "cups").

Recipe Representation: A recipe is represented as the **mean** of all its ingredient embeddings.

Further part is same as in part 1 where you use a Random Forest Regressor.

OUTPUT :

MSE: 22507.98
RMSE: 150.03

MAE: 36.73

Accuracy within ±10:
Overall: 59.93%
Per nutrient:
  Total Fat: 71.52%
  Saturated Fat: 97.35%
  Cholesterol: 27.81%
  Sodium: 8.61%
  Total Carbohydrate: 39.07%
  Dietary Fiber: 100.00%
  Total Sugars: 50.33%
  Protein: 84.77%

Example Prediction with Units:
Total Fat: 16.09 g
Saturated Fat: 6.02 g
Cholesterol: 55.30 mg
Sodium: 506.64 mg
Total Carbohydrate: 41.58 g
Dietary Fiber: 2.71 g
Total Sugars: 25.13 g
Protein: 14.04 g

Method 3:

Neural Network Architecture

- A custom PyTorch neural network NutritionNN is defined:
    - **Input**: BERT-based vector representation of a recipe (i.e., embedding).
    - **Layers**:
        - Fully connected layers: 512 → 256 → 128 → output.
        - Activation: ReLU.
        - Dropout: Helps prevent overfitting.
    - **Output**: A vector of predicted nutrition values (e.g., fat, sugar, protein).

2. Ingredient Embedding with BERT

- get_bert_embedding:

    - Uses bert-base-uncased to encode **ingredient names** into dense vector representations.
    - For each ingredient, it takes the **mean of token embeddings** from BERT's last hidden layer.

3. Data Preprocessing

- Loads and processes a recipes DataFrame:
    - Splits the ingredients string into a list.
    - Parses nutrition text fields into structured dictionaries.
    - Filters out incomplete or invalid data.
    - Extracts and standardizes target columns: Total Fat, Protein, etc.

4. Recipe Representation

- get_ingredient_embeddings:
    - Converts all ingredients in a recipe into BERT vectors.
    - Computes the **mean embedding** across ingredients for the recipe.
    - This produces a **fixed-size input vector** regardless of recipe length.

5. Output

- X: Input matrix — one row per recipe, each row is a BERT-based recipe vector.
- y: Target matrix — actual nutrition values.

MSE: 21796.51
RMSE: 147.64
MAE: 38.85

Accuracy within ±10:
Overall: 56.71%
Per nutrient:
  Total Fat: 66.89%
  Saturated Fat: 97.35%
  Cholesterol: 20.53%
  Sodium: 3.31%
  Total Carbohydrate: 33.11%
  Dietary Fiber: 100.00%
  Total Sugars: 45.70%
  Protein: 86.75%

Example Prediction with Units:
Total Fat: 20.26 g
Saturated Fat: 7.61 g
Cholesterol: 84.49 mg
Sodium: 613.21 mg

Total Carbohydrate: 31.92 g
Dietary Fiber: 3.00 g
Total Sugars: 14.38 g
Protein: 21.53 g

OBSERVATION :
In each case, we could get a soft accuracy of around 60% where we are under range of 10 units difference. As if now we couldn't get a good accuracy rise by different methods.

End-to-End Pipeline Architecture

Here's how the system flows:

1. **Input:** Raw recipe text
2. → **Ingredient Extraction:** NER (rule-based + ML-based)
3. → **Negation Filtering:** Filters out unused ingredients
4. → **Text & Feature Embedding:** BERT + ingredient vectors
5. → **Classification Models:** Cuisine, dietary, difficulty
6. → **Nutrition Prediction:** Estimate macro nutrition
7. → **Output:** A health chart summarizing all predictions

LINK TO ALL TRAINED MODEL FILES:
https://drive.google.com/drive/folders/1m8ZnHL8bJDzYKE1OsKpmHVREV0xBaoyY?usp=sharing