My Perfect Meal

by Daniel Darms (Danield), Christian Lejko (Enderkiki123), Clément Guyot de La Pommeraye (clementlapomme), Alexandra Popescu (APCoder), Arthur Mazaudier (ArthurMazaudier), Alessio Simonetta (simale01), and Jessie Castella (jcastella)

"My Perfect Meal" is an interactive, personalized recommendation system for choosing a restaurant. It combines Machine Learning with location, weather and emotional data to offer users individually tailored meal suggestions. A short setup captures preferences such as taste, nutrition, price preferences and emotional mood. This information, together with previous user interactions, flows into a learning random forest model that continuously improves recommendations. Real-time data is integrated via APIs such as Google Maps, Openweather and IPinfo. The results are clearly displayed on a map. The system thus provides targeted support with the question: What am i eating today - and where does it suit me best?

The Files

The **LAUNCHER.py** has been split into two "execution" parts: main() and setup()

- Main() covers the manual input, recommendation code and visual output.
- Setup() covers the registration and login – thus the input into the database with username, Mood, "Food-tinder" and the biases. It also skips the Tinder for existing users.

The **data_ml.py** acts as:

1. A function repository (hence why 80% of the code is def…(….):)
2. The main communicator between the Database, the pickle file and thus the ML
3. The Debugger for ML and vectors

The **recommendation.db** houses the database of our code. It consists of three relevant tables:

- Users: Name & unique userID, their biases based on the sliders
- Searches: Every manual meal search is linked to a userID, their emotion, preferences, diet restrictions, monetary constraints, and cuisine.
- Interactions: every "Tinder-swipe" gets registered and linked to a userID with dislike (0) and like (1)

The **random_forest_model.pkl** is a Python "pickle-file" and works in binary, thus it is not possible to edit. The file itself houses the "memory" of the Machine Learning which is trained either manually by data_ml or automatically by LAUNCHER after completing main(). A pickle-file deconstructs data and "dumps" the data into a much smaller, efficient file system, like a beamer in Science Fiction, and is rebeamed back to data_ml when needed for ML reuse.

Instructions

Step 1: Save all the files (**LAUNCHER.py**, **data_ml.py**, **random_forest_model.pkl**, **recommendation.db**) in one folder on your device. Make sure that you have the file path for **LAUNCHER.py** ready to paste later.

- On **MacOS**, do this by **right-clicking** the file and selecting **Get Info**. Under **Where**, copy the file path.
- On **Windows**, **right-click** the file and select **Properties**. Copy the path from the **Location** field.
- On **Linux**, **right-click** the file and select **Properties**. Under the **Basic** tab, copy the path next to **Parent Folder**.

Step 2: Open the terminal on your device.

- On **MacOS**, press **Cmd + Space**, type Terminal, and hit **Enter**.
- On **Windows**, press **Windows + R**, type cmd, and hit **Enter**.
- On **Linux**, press **Ctrl + Alt + T** or search for **Terminal** in your applications menu.

Step 3: Install the required libraries/modules by entering the following commands into your terminal:

```
pip install ipinfo
pip install streamlit
pip install folium
pip install googlemaps
```

Make sure you have Python and pip installed on your system. If not, install Python first from https://python.org.

Step 4: Run the application by entering the following command:

```
streamlit run "your_file_path"
```

→ Replace "your_file_path" with the actual path to **LAUNCHER.py** (e.g., "~/Documents/project/LAUNCHER.py").
→ Don't forget the quotation marks around the path.

Step 5: Fill in your username and preferences as prompted.
Please note: due to **Streamlit** limitations, you may need to enter certain inputs twice.

The ML Process

1. First the data is collected through user inputs both in setup () and main (), this means also recent searches.
2. Then we explicitly determined features (all the input now quantified), like bias, mood, diets, min price, max price, cuisine, diet. These features are now translate to mostly binary vectors like [1,0,0,0,0,0,0,0] -> user only likes Italian food according to the cuisine vector.
3. Now all vectors will be put together to create the “feature vector”, e.g. someone who only likes Italian, loves local food by 0.8, prefers warmer food by 0.7, feels happy and has a price range of 1-3 has [0.7, 0.8, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 3]
4. The feature vector is then brought to the pretrained Random-forest-Model, which predicts which restaurants he or she might like by expressing 1 or 0 (liked or disliked), the 1 gets then again recrafted to a word query in Google Places API
5. If disliked, the query is generalized, and random restaurants appear. In this case the user must more often use manual search until the ML is trained.

Random Forest Model

Training the Model:

- A collection of decision trees is trained using historical user data. In our case I have injected a test.pkl with a third python code, that contained randomized tests to check, if the pkl and data_ml were communicating with each other.

- Each decision tree learns to classify inputs (feature vectors) into binary outcomes (Liked = 1, Disliked = 0).

How Trees Work: A decision tree splits the data repeatedly based on feature thresholds. Here is an example:

- First split: Is Temperature Bias > 0.5?
  Second split: Is Cuisine = Italian?
  Third split: Is Emotion = Happy?
  The tree reaches a final decision (1 or 0) at its leaf nodes.

The Random Forest consists of many decision trees, each trained on slightly different subsets of data. Predictions from all trees are aggregated (e.g., majority voting) to make the final prediction. We use Random Forest for three reasons:

1. **Robustness:** Combines the strengths of multiple trees, reducing overfitting.
2. **Feature Importance:** Identifies which features (e.g., "Cuisine" or "Emotion") contribute most to the prediction.
3. **Accuracy:** Handles diverse inputs (like your feature vectors) effectively.