Documentation for Matchmaking Algorithm

Algorithm Overview

Explanation of Matching Logic

The matchmaking algorithm is designed to calculate compatibility scores between users in a dating application. It evaluates user profiles based on shared characteristics and assigns a score to determine potential matches. The algorithm follows these steps:

- ➔ Extract user attributes like hobbies, interests, education.
- ➔ Compare the attributes of the user with other candidates.
- ➔ Assign weights to each factor based on importance.
- ➔ Aggregate the weighted scores to compute the final compatibility score.

Factors Considered and Their Weights

- ➔ **Shared Hobbies (40%)**: High weight as shared hobbies often drive compatibility.
- ➔ **Common Interests (30%)**: Moderate weight to reflect alignment in life perspectives.
- ➔ **Education and Occupation Match (20%)**: Lower weight but crucial for long-term compatibility.
- ➔ **Personality Traits (10%)**: Minor weight, used as a differentiator in close matches.

Mathematical Formulas

The compatibility score is calculated using the following formula:

Where:

- Weight assigned to factor.
- Match score for factor.

Technical Implementation

Code Structure

The project is structured as follows:

MatchingMakingProject/

- ➔ Apps ->
  main.py (API endpoints)
  models.py (Data models)
- ➔ Matching ->
  __init__.py

algorithm.py (Matchmaking logic)

➔ Mock_data ->

users.json (Mock user data)

➔ Requirements.txt (dependencies)

Libraries Used and Why

- **FastAPI**: To build and manage the API endpoints.
- **Uvicorn**: ASGI server for running FastAPI applications.
- **JSON**: For handling mock user data.
- **Typing**: To ensure type safety in Python code.

Performance Considerations

- Current implementation operates on in-memory data (JSON), making it suitable for small datasets.
- To scale, a database (e.g., PostgreSQL) should replace JSON for optimized queries and persistent storage.
- Algorithm performance can be improved using parallel processing or caching.

Future Improvements

What Would You Do Differently With More Time?

1. **Enhance the Algorithm**: Introduce advanced factors like cultural preferences, time zone proximity, and communication styles.
2. **Data Quality**: Improve data cleaning and preprocessing to handle edge cases like missing attributes.
3. **Integration**: Incorporate machine learning models for dynamic weight adjustment based on user feedback.

Scalability Considerations

- Use a relational database with indexing to handle large datasets efficiently.
- Implement API rate limiting and caching to manage high traffic.
- Leverage cloud services like AWS Lambda for auto-scaling.

Additional Features

- Real-time chat functionality for matched users.

- User feedback loop to refine match accuracy.
- Integration with external APIs like social media for richer profiles.
- Location-based matching for nearby candidates.