Multilingual Room Matching with (Fuzzy Logic) Sentence Transformer and XGBoost

Ryoji – Room Match ML API Project

April 2025

Contents

1 Introduction

This project builds a multilingual machine learning API for matching hotel room listings, which is similar to Cupid's Room Match API. The system accepts POST requests with structured room data from both suppliers and a reference catalog, and returns probabilistic room match predictions. It supports mixed-language inputs (e.g., English, Arabic, Korean, etc.) and uses fuzzy logic, language detection, and machine learning classification.

2 Project Structure and API Setup

- `Room_Match/` (project root)
  - `README.md` *This file*
  - `requirements.txt` *All dependencies*
  - `app.py` *Flask API server*
  - `matcher.py` *Core logic for matching*
  - `models/` *Pretrained models*
    - `model.pkl` *Generated in `room_match_dev.ipynb` (XGBoost model)*
    - `lid.176.bin` *Downloaded fastText language model*
  - `sample_request.json` *Example POST request payload*
  - `test_post.py` *Simple script to send a test POST request*
  - `run_server_and_test.sh` *Shell script to run server and test*
  - `scripts/` *Utility or development scripts*
    - `__init__.py` *Placeholder*
  - `tests/` *Unit tests for matcher and API*
    - `__init__.py`
    - `conftest.py`
    - `test_room_match.py`
    - `test_post.py`
  - `notebooks/` *EDA, model training and evaluation*
    - `room_match_dev.ipynb`
  - `__pycache__/` *Python bytecode cache*

Running the API:

1. `pip install -r requirements.txt`
2. `FLASK_APP=app.py flask run --host=0.0.0.0 --port=5050`
3. Send a test request:

```
curl -X POST http://127.0.0.1:5050/room_match \
  -H 'Content-Type: application/json' \
  -d @sample_request.json
```

4. Or run `python test_post.py`

```
{
  "inputCatalog": [
    {
      "supplierId": "nuitee",
      "supplierRoomInfo": [
        {"supplierRoomId": "2", "supplierRoomName": "Classic Room - Olympic Queen Bed - ROOM ONLY"}
      ]
    }
  ],
  "referenceCatalog": [
    {
      "propertyId": "5122906",
      "propertyName": "Pestana Park Avenue",
      "referenceRoomInfo": [
        {"roomId": "512290602", "roomName": "Classic Room"},
        {"roomId": "512290608", "roomName": "Classic Room - Disability Access"}
      ]
    }
  ]
}
```


3.2 Room Matching Strategy

To develop the backend ML model, I first loaded and explored the datasets:

```
df_rooms: updated_core_rooms.csv
df_ref: reference_rooms-1737378184366.csv
```

Exploratory Data Analysis (EDA) included inspecting schema with `df.info()`, removing records where `room_name` is `NaN`, and understanding key identifier relationships like `lp_id`, `hotel_id`, `room_id`, and `core_room_id`. The `room_id` typically acts as a foreign key while `core_room_id` reflects internal indexing within the database. The `hotel_id` uniquely identifies the hotel property, and `lp_id` corresponds to the landing page ID used primarily for tracking and marketing purposes.

Importantly, even if `lp_id` values differ, matching `hotel_id` and `room_id` across listings—particularly when referenced from the `core_room` database—generally signifies the same room. This indicates that for data integrity and mapping, `hotel_id` and `room_id` should be treated as the authoritative identifiers for identifying and matching room entities.

However, based on exploratory data analysis (EDA), we observed the following:

```
{
    'lp_id match': 113,718,
    'lp_id + hotel_id match': 130,
    'lp_id + hotel_id + room_id match': 0
}
```

These figures highlight critical ID inconsistencies in the dataset:

- While `lp_id` overlaps are relatively common, their reliability in isolation is questionable.
- Matching both `lp_id` and `hotel_id` is rare (130 entries), suggesting high variability or inconsistent supplier mappings.
- There are no entries where `lp_id`, `hotel_id`, and `room_id` all match together—indicating that `room_id` cannot be safely used to match rooms across listings without also confirming `hotel_id`.

This underscores a key limitation: relying solely on `room_id` (even with high room name similarity) can lead to incorrect matches across different hotels. Thus, for safe and accurate mapping:

- **`hotel_id` + room name similarity (e.g., cosine similarity** ≥ 0.85**)** should be used as the primary matching criteria.
- `room_id` can serve as a secondary signal, but *only within the same* `hotel_id` context (which is *not* the case in this dataset).

Figures Figure 1 summarizes the ID, hotel_id matching counts of `lp_id`, `hotel_id`, `room_id`, and `core_room_id`.

Language detection was performed using `fastText` to annotate room names for multilingual handling.

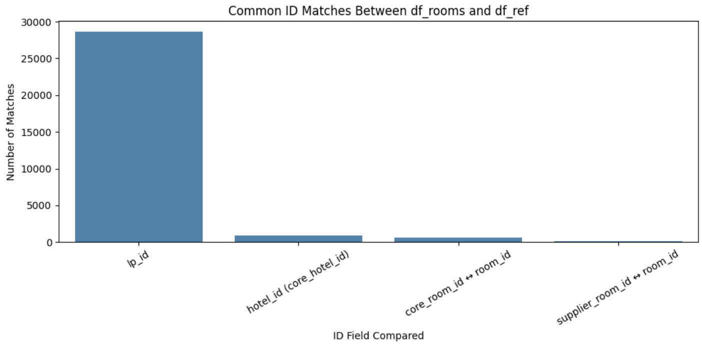

Figure 1: Common ID Matches Between core rooms and reference rooms

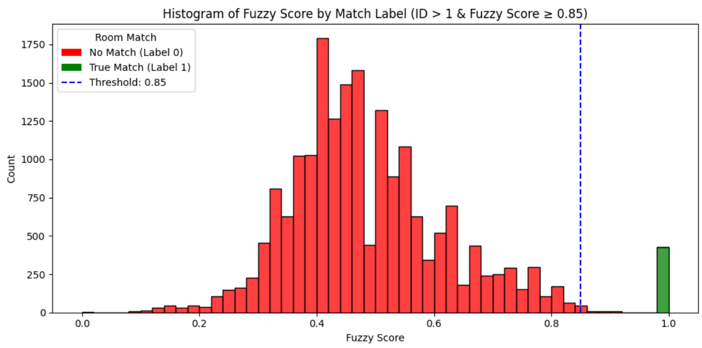

Figure 2: Histogram of Fuzzy Score by Match Label (hotel_id) > 1 & Cosine Sim ≥ 0.85

Figure 2 displays the ID matching and cosine_sim ($\geq$ 0.85[1] and the rest of the counts. It is worth noting that the fuzzy_score distribution is heavily dominated by values to 1.0, and the resulting 0/1 label distribution is approximately 1:40.

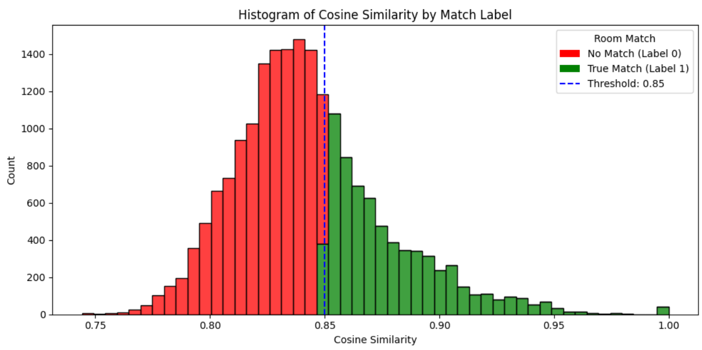

Figure 3: Cosine Similarity Histogram using Sentence-Transformer (`intfloat/multilingual-e5-base`) Embeddings

Figure 3 shows the cosine similarity distribution using multilingual E5 embeddings. Unlike the fuzzy string score, this model better separates true matches from false ones across languages and scripts. As a result, we obtained more balanced label distribution with richer semantic embeddings, especially for rooms in mulit-langauges.

For supervised model training:

- Matching candidates were created when `hotel_id` matched more than once.
- Similarity scores were computed using fastText embedding similarity.
- The dataset was labeled and split accordingly.
- A tuned `XGBoost` classifier was trained on features including ID match booleans and text-based similarity.

Evaluation metrics:

- **Confusion Matrix** to identify true/false positives and negatives
- **F1-Score** to balance precision and recall
- **ROC Curve** for threshold-independent classification performance

Figures below show the ROC curve and the confusion matrix.

3.3 Model Training

- Label = 1 if cosine_sim ≥ 0.85 and `hotel_id` match
- Model: XGBoost classifier
- Hyperparameter Tuning: Optuna
- Metrics: F1-Score, ROC-curve, Confusion Matrix

3.4 Multilingual Handling

- `fastText` supports 100+ languages.
- Can detect Arabic, Korean, Japanese, etc. — but only the dominant language.
- Mixed-language strings may produce partial results.
  - Example: `Deluxe Room (デラックスルーム)` may be detected as either Japanese or English depending on the structure and dominant script.

[1] I tested thresholds at 0.75, 0.85, and 0.95. The label distribution ratios 0/1 were approximately 14, 40, and 44 respectively. Despite increasing imbalance, the XGBoost classifier achieved high performance across all thresholds, with F1-scores 1.0 across all fuzzy score ranges.

- In contrast, the `e5-base` multilingual transformer model provides a significantly more robust representation of multilingual text by embedding the full semantic meaning—regardless of language or script. It enables accurate cross-language matching (e.g., Japanese-English or Arabic-French) even when strings are partially translated or use a mix of alphabets.

Limitation: `fastText` cannot detect or translate multiple languages in one string. It returns only the dominant language. It cannot measure similarity or equivalence between different languages.

Recommendation: If resources allow, consider even larger LLMs (e.g., `bge-large-en`, `all-mpnet-base-v2`, or multilingual variants like `LaBSE`) for more robust semantic matching, especially in diverse language pairs or low-resource scenarios.

4 Results

- **F1-score:** ~ 100 %.
- **ROC-AUC:** High, shown in Fig. 4.
- **Confusion Matrix:** Small false negatives, shown in Fig. 5.

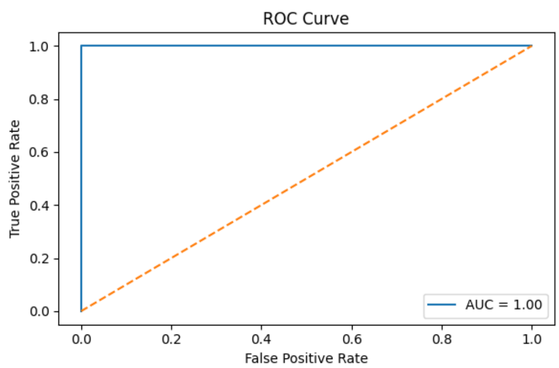

Figure 4: ROC AUC Curve

5 Sample Output

```
[fontsize=\small]
{
  "matches": [
    {
      "supplierRoomId": "2",
      "supplierRoomName": "Classic Room
       - Olympic Queen Bed - ROOM ONLY",
      "refRoomId": "512290602",
```

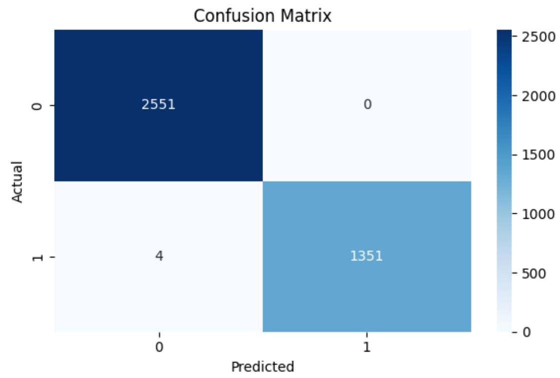

Figure 5: Confusion Matrix

```
      "refRoomName": "Classic Room",
      "cosine_sim": 0.8887,
      "match_score": 0.9945
    },
    {
      "supplierRoomId": "2",
      "supplierRoomName": "Classic Room
       - Olympic Queen Bed - ROOM ONLY",
      "refRoomId": "512290608",
      "refRoomName": "Classic Room
       - Disability Access",
      "cosine_sim": 0.8666,
      "match_score": 0.9945
    }
  ]
}
```

6 Limitations and Future Work

- `fastText` cannot measure similarity or equivalence between different languages. Therefore, it may miss correct `room_matching` cases where room names are written in different languages but share the same meaning.
- Only one supplier — extension to multiple suppliers for real-world evidence (RWE).
- Current model uses only name-based features.
- The current implementation leverages the `intfloat/multilingual-e5-base` model, which offers strong multilingual semantic matching. While this represents a significant improvement over traditional fuzzy matching, more powerful models (e.g., `e5-large` or cross-encoder architectures) could be explored to further enhance accuracy—provided sufficient computational resources are available.
- Future versions should incorporate richer room attributes:

  - Room view, floor, and amenities
  - Descriptions and full metadata (e.g., configurations, images)

6.1 Deployment Notes

- Docker for reproducibility
- CI/CD with Jenkins or GitHub Actions
- Hosting via FastAPI or TorchServe

6.2 LLM Potential

- Use of `SentenceTransformer("intfloat/multilingual-e5-base")` improves multilingual and cross-script similarity.
- Future: Fine-tuning with LoRA for domain-specific vocabulary.
- Possible integration with RAG and retrieval-based matching on room description embeddings.