MedConnector – Matching Module (matching.py) Documentation

This document explains the key components and algorithms implemented in `matching.py`, the core matching engine that powers MedConnector. The goal of the module is to generate a numeric "match index" (0–1) that quantifies how well a given medical student fits a specific pre‑medical student's preferences across multiple dimensions.

# 1  Overview & Architectural Design

The file centers around a single class – `MatchingService` – which exposes:  
• Data‑loading helpers (`load\_data`).  
• Dimension‑specific matching functions (e.g.  
  `match\_by\_year`, `match\_by\_undergrad\_degree`).  
• Utility methods for text cleaning and safe extraction.  
• `overall\_match\_index`, which aggregates component scores into one global index via a weighted average.

A diagrammatic breakdown of the workflow:  
1. Load the medical‑student and (optionally) pre‑med spreadsheets.  
2. For each dimension (year, gap years, degree, clinical interests, research, motivation essay, student orgs) compute a similarity score ∈ [0, 1].  
3. Combine the scores using static weights (configurable through an environment variable) to obtain the final match index.

# 2  Static Configuration & Weights

At the top of the file a dict called `DEFAULT\_WEIGHTS` defines how much each dimension contributes to the overall score. The helper `\_active\_weights()` allows DevOps/CI pipelines to override any weight at runtime via `MATCH\_WEIGHTS='<json>'` – a convenient, zero‑code‑change way to A/B‑test weighting schemes.

|  |  |
| --- | --- |
| Component | Default Weight |
| year | 0.2 |
| gap | 0.1 |
| degree | 0.15 |
| clinical | 0.25 |
| research | 0.15 |
| motivation | 0.1 |
| orgs | 0.05 |

The function `\_safe\_weighted\_sum` performs the actual aggregation while gracefully handling missing fields.

# 3  Initialisation of `MatchingService`

When instantiated, the service:  
• Creates a `MinMaxScaler` (reserved for future numeric features).  
• Sets up two `SentenceTransformer` pipelines:  
  – \*\*`all‑mpnet‑base‑v2`\*\* – a strong, general‑purpose sentence model used for  
    motivation essays.  
  – \*\*`BiomedBERT (uncased‑abstract)`\*\* – explicitly wrapped with a mean‑token  
    pooling layer for biomedical semantics, powering clinical, research, and  
    degree comparisons.  
Embedding models are cached internally by `sentence‑transformers`, so reuse is inexpensive.

# 4  Data Loading Helpers

`load\_data()` accepts paths to the medical‑student roster and (optionally) a single pre‑med file. Columns are stripped of whitespace to prevent mismatches introduced by Excel.

# 5  Field‑Specific Matching Logic

## 5.1  Year in School

`\_convert\_year` maps literal strings (M1–M4) → integers (1–4). `calculate\_year\_compatibility` applies a crisp rule‑based score:  
   • Exact → 1.0  
   • Adjacent (±1) → 0.5  
   • Otherwise → 0.0

## 5.2  Gap Years

`\_convert\_gap\_year` normalises textual answers into a 0‑3 scale. `gap\_year\_match` mirrors the year logic (exact / ±1 / other).

## 5.3  Undergraduate Degree

Arguably the most sophisticated matcher in the file. It blends:  
• \*\*Fuzzy token‑set ratio\*\* (RapidFuzz).  
• \*\*BioMedBERT cosine similarity\*\*.  
• Direct substring bonuses.  
• Domain heuristics (life‑sci, social‑sci, physical‑sci, engineering) to reward cross‑lingual equivalence such as \*human biology\* ↔ \*physiology\*. Penalty logic prevents false positives like \*biology\* vs \*sociology\*. All intermediate values can be examined by toggling the built‑in debug flag.

## 5.4  Clinical Interests

Inputs are normalised through `\_normalize\_to\_set`, which removes placeholder tokens such as \*other…\* and unifies comma/newline delimiters. Scores combine:  
• Jaccard overlap on the cleaned sets.  
• Fuzzy token‑set ratio on flattened text.  
• BioMedBERT embeddings.

## 5.5  Research Interests

A two‑factor hybrid:  
• 0.6 × BioMedBERT similarity.  
• 0.4 × Jaccard on extracted biomedical keywords.  
A +0.3 boost is applied when the pre‑med string appears verbatim inside the med‑student string.

## 5.6  Student Organisations

A simple exact‑set matcher – any common organisation triggers a score of 1.0. Otherwise 0.0. The rationale: shared extracurriculars are a strong proxy for cultural fit.

## 5.7  Motivation Essay

Three signals are ensembled:  
• General Sentence Transformer similarity (all‑mpnet).  
• Jaccard on a hand‑curated lexicon of motivation keywords.  
• TF‑IDF cosine similarity (top‑50 terms across the corpus). Weights: 0.5 / 0.3 / 0.2.

# 6  Utility Methods

• `\_safe\_get` – bullet‑proof accessor that handles Series, NaN, and missing columns.  
• `\_extract\_keywords` – lightweight keyword extractor that strips punctuation and stopwords for biomedical text.  
• `\_normalize\_clinical\_fields` – prepares each row with convenient lowercase lists for fast set operations.

# 7  Global Match Index

`overall\_match\_index` iterates through the medical‑student DataFrame, collects component scores via `\_component\_scores`, then feeds them into `\_safe\_weighted\_sum` using `\_active\_weights()`. The result is rounded to four decimals for stable ranking.

# 8  Extensibility & Customisation

✅ \*\*Add a new dimension\*\* – implement `<match\_by\_your\_feature>` plus the underlying comparator, then register it in `\_component\_scores` and `DEFAULT\_WEIGHTS`.  
✅ \*\*Tweak scoring formulas\*\* – each matcher is self‑contained; feel free to swap the blend weights or models.  
✅ \*\*Runtime weight overrides\*\* – export `MATCH\_WEIGHTS` in your deployment environment to tune without code edits.

# 9  Error Handling & Debugging

• Defensive type checking prevents `NoneType` / `Series` truth‑value errors.  
• Key operations are wrapped in `try/except` blocks to fall back to safer models.  
• Setting environment variable `DEBUG\_DEGREE\_MATCH=1` (example) surfaces granular logs for the degree matcher.

# 10  Minimal Usage Example

```python  
svc = MatchingService()  
svc.load\_data('medstudents.xlsx') # roster  
premed = pd.read\_excel('single\_premed.xlsx') # one‑row file  
results = svc.match\_single\_premed('single\_premed.xlsx')  
top\_global = results['global\_matches'][:5]  
```