Standardizing Clinical Symptoms of Rare Disease with Human Phenotype Ontology (HPO) in Python

Background

In literature reviews and evidence synthesis for rare diseases, clinical symptoms are often reported in non-standardized ways, making it difficult to compare or merge them computationally.

This real-world challenge motivated us to develop a data solution for standardizing free-text symptom reports. Using the open-source Human Phenotype Ontology (HPO) and its API, we can map reported symptoms to controlled ontology terms, with the process automated in Python.

I streamlined the workflow into several steps: retrieving candidate HPO terms and synonyms, linking them to IDs and definitions, and applying fuzzy matching to identify similarities between reported symptoms and retrieved HPO terms. This notebook demonstrates the pipeline with minimal documentation as a reference for the community.

The workflow has been tested in a real-world project on congenital myasthenic syndromes (CMS). While effective, there remain opportunities to refine and expand the approach.

Alogrithm explained

Goal Standardize free-text clinical symptoms by mapping them to HPO (Human Phenotype Ontology) terms, then verify and contextualize each match.

Inputs: symptom (str): A reported, free-text symptom (e.g., “ptosis”, “muscle weakness”).

External resources & libs

- Search API: https://ontology.jax.org/api/hp/search/?q= (top result taken)
- Ontology file: http://purl.obolibrary.org/obo/hp.obo (definitions, synonyms, hierarchy)
- Python libs: requests, fuzzywuzzy.process.extractOne, obonet, functools.lru_cache, pandas (optional)

High-level flow

1. Search HPO: Query the JAX HPO API with the input symptom → get top candidate (name, id) or no result.
2. Fuzzy validation: Compute a fuzzy score between the input symptom and the returned HPO term name.
3. Context retrieval: From hp.obo, pull definition and synonyms for the candidate HPO ID.

4. Lineage extraction: From the same ontology graph, compute depth and path to root HP:0000001 (using first parent if multiple).
5. Accept/Reject decision - Accept if fuzzy_score ≥ 80 OR HPO name appears in its synonyms (case-insensitive check). - Reject otherwise (or if API/ontology lookup fails).

Decision rule (acceptance)

- Threshold: fuzzy_score ≥ 80
- Synonym override: Accept if HPO term name is present among its synonyms (case-insensitive)

Rationale: Puts speed/recall first (top hit) with a sanity check on similarity; adds semantic cushion via synonyms.

Outputs (as implemented)

The pipeline returns 8 fields in fixed order:

1. reported_symptom (str)
2. hpo_term (str | None)
3. hpo_id (str | None)
4. definition (str | None)
5. rank (int | None)
6. path (list[str] | [])
7. fuzzy_score (int | 0)
8. status ("matched" | "not matched")

Load necessary modules

In [29]:
```
# General modules
import requests
import pandas as pd
import time
from typing import List, Tuple, Dict, Iterable, Optional

# Specific modules
import rapidfuzz
from fuzzywuzzy import process
import obonet
from functools import lru_cache
```

Brief information for the specific modules

- `fuzzywuzzy.process`: Provides fuzzy string matching, useful for comparing free-text symptoms to HPO terms and synonyms.
- `obonet`: Loads and parses OBO-formatted ontology files, such as the Human Phenotype Ontology, into network structures.
- `functools.lru_cache`: Decorator for caching function results, improving performance when repeatedly querying or processing the same data.

Implementation

Step 2: Estimate the fuzzy score between the input term and the HPO term

In [16]:
```python
# --- Fuzzy matching shim: prefer RapidFuzz; fallback to FuzzyWuzzy ---
try:
    from rapidfuzz import fuzz as _fuzz
    from rapidfuzz import process as _process
    _FUZZ_LIB = "rapidfuzz"
except Exception:
    # Fallback (ensure python-Levenshtein is installed for speed)
    from fuzzywuzzy import fuzz as _fuzz
    from fuzzywuzzy import process as _process
    _FUZZ_LIB = "fuzzywuzzy"

def fuzzy_extract_one(query, choices, scorer=None):
    """
    Wrapper around extractOne with a unified return shape:
    returns (match_str, score).
    """
    if _FUZZ_LIB == "rapidfuzz":
        # RapidFuzz returns (match, score, index). Default scorer ~ ratio
        # You can set scorer=_fuzz.WRatio for WRatio behavior, or _fuzz.ratio.
        match, score, _ = _process.extractOne(query, choices, scorer=scorer or _f
        return match, score
    else:
        # FuzzyWuzzy returns (match, score)
        match, score = _process.extractOne(query, choices, scorer=scorer or _fuzz
        return match, score
```

Notes

- RapidFuzz: ratio, partial_ratio, token_sort_ratio, WRatio are available; default above uses ratio.
- FuzzyWuzzy: WRatio is a solid general-purpose scorer, hence the fallback default.

In [17]:
```python
def estimate_fuzzy_score(input_term: str, hpo_term: str) -> float:
    """
    Return similarity score in [0, 100], using RapidFuzz if available,
    otherwise FuzzyWuzzy (preferably with python-Levenshtein installed).
    """
    if not isinstance(input_term, str):
        raise ValueError("reported_term must be a string.")
    if not isinstance(hpo_term, str):
        return 0.0
    _, score = fuzzy_extract_one(input_term, [hpo_term])
    # RapidFuzz and FuzzyWuzzy both output 0-100 scale
    return float(score)
```

Step 3: Look up HPO synonyms and definition

In [20]:
```python
# --- Shared ontology loader (already in your notebook) ---
import obonet
from functools import lru_cache

HPO_URL = "http://purl.obolibrary.org/obo/hp.obo"

@lru_cache(maxsize=1)
def load_graph():
    """Load and cache the HPO graph once per session."""
    return obonet.read_obo(HPO_URL)


# --- Improved: fast, cached meta lookup ---
@lru_cache(maxsize=8192)  # cache per-HPO-ID lookups, too
def get_hpo_definitions_and_synonyms(hpo_id: str):
    """
    Return (synonyms, definition) for an HPO ID using the cached graph.
    - Avoids repeated obo downloads/parsing.
    - Cleans OBO-quoted strings for readability.
    """
    graph = load_graph()  # <-- reuse cached graph
    node = graph.nodes.get(hpo_id)
    if not node:
        return [], None

    # Synonyms: in OBO it's usually 'synonym' (singular); keep a fallback.
    raw_syn = node.get("synonym", []) or node.get("synonyms", [])

    def _clean_obo_text(s: str) -> str:
        # OBO annotation format often looks like:  "\"text\" EXACT [XREF:...]"
        if isinstance(s, str) and '"' in s:
            try:
                return s.split('"', 2)[1]
            except Exception:
                return s
        return s

    synonyms = [_clean_obo_text(s) for s in raw_syn]

    # Definition may be a single string like "\"text\" [PMID:...]"
    raw_def = node.get("def")
    definition = _clean_obo_text(raw_def) if isinstance(raw_def, str) else None

    return synonyms, definition
```

Step 4: Get full lineage for a given HPO ID

In [22]:
```python
import obonet
from functools import lru_cache

# HPO_URL = "http://purl.obolibrary.org/obo/hp.obo"

# # Cache the graph so it loads only once
# @lru_cache(maxsize=1)
# def load_graph():
#     return obonet.read_obo(HPO_URL)
```

```python
def get_rank_and_path(hpo_id):
    """
    Return rank and path from root to this term (shortest path).
    """
    graph = load_graph()
    if hpo_id not in graph:
        return None, []

    path = [hpo_id]
    depth = 0
    current = hpo_id
    while True:
        parents = graph.nodes[current].get("is_a", [])
        if not parents:
            break
        current = parents[0]  # take first parent if multiple
        path.append(current)
        depth += 1
        if current == "HP:0000001":
            break
    return depth, list(reversed(path))
```

Normalizatoin and synonym mathcing

In [24]:

```python
from typing import Iterable, Tuple, Optional
try:
    # Prefer RapidFuzz (faster, no GPL issues)
    from rapidfuzz import fuzz, process as rf_process
    _USE_RF = True
except Exception:
    # Fall back to fuzzywuzzy if RapidFuzz isn't available
    from fuzzywuzzy import fuzz, process as fw_process
    _USE_RF = False

def _norm(s: Optional[str]) -> str:
    return (s or "").strip().lower()

def synonym_matches_input(
    input_symptom: str,
    synonyms: Iterable[str],
    exact: bool = True,
    fuzzy_threshold: int = 90
) -> bool:
    """
    Return True if the input symptom matches any synonym (exact case-insensitive
    or fuzzy >= threshold).
    """
    inp = _norm(input_symptom)
    syns = [_norm(s) for s in (synonyms or []) if s]

    # Exact (case-insensitive)
    if exact and inp in syns:
        return True

    # Fuzzy fallback if desired
    if fuzzy_threshold is not None and len(syns) > 0:
        if _USE_RF:
            # RapidFuzz: compute max similarity quickly
```

```
            # (rf_process.extractOne returns (match, score, idx))
            _, score, _ = rf_process.extractOne(inp, syns, scorer=fuzz.ratio)
            return score >= fuzzy_threshold
        else:
            # FuzzyWuzzy fallback
            best, score = fw_process.extractOne(inp, syns)
            return score >= fuzzy_threshold

    return False
```

Step 1: Map reported symptoms to HPO terms using the HPA API

In [ ]:

```
import time
import requests
from typing import Iterable, Optional, Tuple, List, Dict



def map_symptoms_to_hpo(
    symptom: str,
    timeout: int = 10,
    retries: int = 2,
    backoff: float = 0.7,
    top_k: int = 5
) -> List[Dict[str, str]]:
    """
    Query the JAX HPO search API and return up to top_k candidates:
    [{ "name": <term_name>, "id": <hpo_id> }, ...]
    Returns [] on failure or no results.
    """
    url = "https://ontology.jax.org/api/hp/search/"
    params = {"q": symptom}

    for attempt in range(retries + 1):
        try:
            resp = requests.get(url, params=params, timeout=timeout)
            resp.raise_for_status()
            data = resp.json()
            results = data.get("terms", []) or []
            # Trim to top_k if requested
            out = []
            for r in results[:max(1, top_k)]:
                name = r.get("name")
                hpo_id = r.get("id")
                if name and hpo_id:
                    out.append({"name": name, "id": hpo_id})
            return out
        except requests.exceptions.RequestException:
            if attempt < retries:
                time.sleep(backoff * (attempt + 1))
                continue
            return []
        except ValueError:
            return []


def _score_candidate(
    symptom: str,
```

```
    cand_name: str,
    cand_id: str,
    score_scorer=None,
    synonym_exact: bool = True,
    synonym_fuzzy_threshold: int = 90
) -> Dict[str, object]:
    """
    Compute metrics for a candidate:
    - fuzzy score (symptom vs candidate label)
    - synonym match (exact/fuzzy)
    - definition, rank, path for the candidate
    """
    # Fuzzy similarity to the label
    fuzzy_score = estimate_fuzzy_score(symptom, cand_name)

    # Synonyms/definition
    synonyms, definition = get_hpo_definitions_and_synonyms(cand_id)
    syn_ok = synonym_matches_input(
        input_symptom=symptom,
        synonyms=synonyms,
        exact=synonym_exact,
        fuzzy_threshold=synonym_fuzzy_threshold
    )

    # Lineage
    rank, path = get_rank_and_path(cand_id)

    return {
        "name": cand_name,
        "id": cand_id,
        "fuzzy_score": float(fuzzy_score),
        "syn_match": bool(syn_ok),
        "definition": definition,
        "rank": rank,
        "path": path or [],
        "synonyms": synonyms,  # useful for debugging
    }


def _choose_best_candidate(
    scored: List[Dict[str, object]],
    score_threshold: int = 80
) -> Optional[Dict[str, object]]:
    """
    Pick the best candidate with the following priority:
    1) Any candidate with syn_match=True and fuzzy_score >= score_threshold
    2) Any candidate with syn_match=True (highest fuzzy_score wins)
    3) Highest fuzzy_score candidate overall
    """
    if not scored:
        return None

    # 1) syn match + score above threshold
    tier1 = [c for c in scored if c["syn_match"] and c["fuzzy_score"] >= score_th
    if tier1:
        return max(tier1, key=lambda c: c["fuzzy_score"])

    # 2) syn match (best fuzzy among them)
    tier2 = [c for c in scored if c["syn_match"]]
```

```
    if tier2:
        return max(tier2, key=lambda c: c["fuzzy_score"])

    # 3) best fuzzy overall
    return max(scored, key=lambda c: c["fuzzy_score"])
```

Pipeline function to chain step 1-4

In [31]:

```
def map_symptoms_to_hpo_pipeline(
    symptom: str,
    score_threshold: int = 80,
    synonym_fuzzy_threshold: int = 90,
    synonym_exact: bool = True,
    top_k: int = 5,
    return_debug: bool = False,
):
    """
    Evaluate top-K candidates from API and choose the best per fuzzy/synonym logi

    Returns (always 8 fields):
    1) reported_symptom : str
    2) hpo_term         : str | None
    3) hpo_id           : str | None
    4) definition       : str | None
    5) rank             : int | None
    6) path             : list[str]
    7) fuzzy_score      : float
    8) status           : 'matched' | 'not matched'

    If return_debug=True, also returns a 9th field:
    9) debug_candidates : list[dict] with per-candidate scores & flags
    """
    # Step 1: fetch candidates
    candidates = map_symptoms_to_hpo(symptom, top_k=top_k)

    if not candidates:
        base = (symptom, None, None, None, None, [], 0.0, "not matched")
        return (base + ([],)) if return_debug else base

    # Step 2: score each candidate
    scored = [
        _score_candidate(
            symptom=symptom,
            cand_name=c["name"],
            cand_id=c["id"],
            synonym_exact=synonym_exact,
            synonym_fuzzy_threshold=synonym_fuzzy_threshold,
        )
        for c in candidates
    ]

    # Step 3: choose best
    best = _choose_best_candidate(scored, score_threshold=score_threshold)

    if not best:
        base = (symptom, None, None, None, None, [], 0.0, "not matched")
        return (base + (scored,)) if return_debug else base
```

```
    # Step 4: accept/reject
    accept = (best["fuzzy_score"] >= score_threshold) or best["syn_match"]
    status = "matched" if accept else "not matched"

    result = (
        symptom,
        best["name"],
        best["id"],
        best["definition"],
        best["rank"],
        best["path"],
        float(best["fuzzy_score"]),
        status,
    )
    return (result + (scored,)) if return_debug else result
```

How to use (quick demo)

In [32]:
```
symptoms = ["ptosis", "weak suck", "exercise intolerance"]
rows = [map_symptoms_to_hpo_pipeline(s, top_k=8, return_debug=True) for s in symp

# Unpack for viewing
import pandas as pd
cols = ["reported_symptom","hpo_term","hpo_id","definition","rank","path","fuzzy_
pd.DataFrame(rows, columns=cols)
```

Out[32]:

	reported_symptom	hpo_term	hpo_id	definition	rank	path	fuzzy_score
0	ptosis	Ptosis	HP:0000508	The upper eyelid margin is positioned 3 mm or ...	4	[HP:0000001, HP:0000118, HP:0000478, HP:001237...	83.333333
1	weak suck	Weak cry	HP:0001612	None	4	[HP:0000001, HP:0000118, HP:0001608, HP:002542...	58.823529
2	exercise intolerance	Exercise intolerance	HP:0003546	A functional motor deficit where individuals w...	3	[HP:0000001, HP:0000118, HP:0025142, HP:0003546]	95.000000