0.Build_HAPI_Labels

December 5, 2025

1 HAPI Label Generation Script (Target Definition)

1.1 Description

This script constructs the "Ground Truth" target labels for the machine learning model. It identifies patients who developed a Hospital-Acquired Pressure Injury (HAPI) by combining structured billing data with unstructured clinical notes.

1.2 Clinical Justification for Target Definition

Defining the target variable accurately is critical for model validity: 1. **Structured Data (ICD-9/10):** Billing codes are highly specific but often have low sensitivity (under-coding). 2. **Unstructured Data (NLP):** Scanning discharge notes for keywords (e.g., "decubitus", "stage 2") captures clinical documentation that may not have made it to the final bill. 3. **The 48-Hour Rule:** Per CMS (Centers for Medicare & Medicaid Services) guidelines, a condition is considered "Hospital-Acquired" only if evidence appears **> 48 hours** after admission. Injuries documented within the first 48 hours are classified as "Present on Admission" (POA) and are excluded from the positive class.

1.3 Inputs & Outputs

- **Inputs:** `admissions.csv`, `discharge.csv`, `diagnoses_icd.csv`
- **Output:** `hospitalwide_hapi_labels.csv`
- **Target Variable:** `HAPI_FINAL` (1 = Acquired HAPI, 0 = No HAPI)

[1]:
```python
import re
import pandas as pd
import os
```

[2]:
```python
# Configuration
BASE_DIR = r"D:\School\5141"

ADMISSIONS_PATH = os.path.join(BASE_DIR, "admissions.csv", "admissions.csv")
DISCHARGE_PATH  = os.path.join(BASE_DIR, "discharge.csv", "discharge.csv")
DIAGNOSES_PATH  = os.path.join(BASE_DIR, "diagnoses_icd.csv", "diagnoses_icd.
↪csv")
OUTPUT_PATH     = os.path.join(BASE_DIR, "hospitalwide_hapi_labels.csv")

# Processing Constants
```

```
NOTES_CHUNKSIZE = 100_000

# CLINICAL THRESHOLD: The "48-Hour Rule"
# Conditions appearing < 48 hours from admit are "Present on Admission" (Not␣
↪HAPI).
# Conditions appearing > 48 hours are "Hospital Acquired" (HAPI).
HAPI_THRESHOLD_HOURS = 48.0

pd.set_option("display.max_rows", 30)
```

[3]:
```
# Dignoses Codes for Pressure Ulcers

# STRUCTURED CODES (ICD-9 & ICD-10)
# These represent billing diagnoses for pressure ulcers of various stages for␣
↪ICD 9/10.
icd10_pu_codes = [
    "L89000","L89001","L89002","L89003","L89004","L89009",
    "L89010","L89011","L89012","L89013","L89014","L89019",
    "L89020","L89021","L89022","L89023","L89024","L89029",
    "L89100","L89101","L89102","L89103","L89104","L89109",
    "L89110","L89111","L89112","L89113","L89114","L89119",
    "L89120","L89121","L89122","L89123","L89124","L89129",
    "L89130","L89131","L89132","L89133","L89134","L89139",
    "L89140","L89141","L89142","L89143","L89144","L89149",
    "L89150","L89151","L89152","L89153","L89154","L89159",
    "L89200","L89201","L89202","L89203","L89204","L89209",
    "L89210","L89211","L89212","L89213","L89214","L89219",
    "L89220","L89221","L89222","L89223","L89224","L89229",
    "L89300","L89301","L89302","L89303","L89304","L89309",
    "L89310","L89311","L89312","L89313","L89314","L89319",
    "L89320","L89321","L89322","L89323","L89324","L89329",
    "L8940","L8941","L8942","L8943","L8944","L8945",
    "L89500","L89501","L89502","L89503","L89504","L89509",
    "L89510","L89511","L89512","L89513","L89514","L89519",
    "L89520","L89521","L89522","L89523","L89524","L89529",
    "L89600","L89601","L89602","L89603","L89604","L89609",
    "L89610","L89611","L89612","L89613","L89614","L89619",
    "L89620","L89621","L89622","L89623","L89624","L89629",
    "L89810","L89811","L89812","L89813","L89814","L89819",
    "L89890","L89891","L89892","L89893","L89894","L89899",
    "L8990","L8991","L8992","L8993","L8994","L8995",
]

icd9_pu_codes = [
    "7070","70700","70701","70702","70703","70704",
    "70705","70706","70707","70709",
    "70720","70721","70722","70723","70724","70725",
]

PRESSURE_ULCER_CODES = set(
    c.strip().upper() for c in (icd10_pu_codes + icd9_pu_codes)
)

# Unstructured Keywords (NLP)
# Search for clinical synonyms of pressure injuries in the text notes.
# This captures stages (1-4), Deep Tissue Injury (DTI), and location-specific␣
  ↪ulcers.
WOUND_KEYWORDS = [
    "ulcer", "pressure ulcer", "pressure injury",
    "decubitus", "bedsore", "skin breakdown",
    "non-blanch", "nonblanch", "non blanch",
    "erythema", "unstageable", "deep tissue", "dti", "sdtpi",
    "stage 1", "stage i", "stage 2", "stage ii", "stage 3", "stage iii",
    "stage 4", "stage iv",
    "sacral wound", "sacral ulcer", "coccyx ulcer",
    "heel ulcer", "ischial ulcer",
]

WOUND_PATTERN = re.compile("|".join(re.escape(k.lower()) for k in␣
  ↪WOUND_KEYWORDS))
```

[4]:

```
#Helpter Functions

def load_admission_info(path: str):
    """
    Load admission timestamps to establish 'Time Zero'.

    This is critical for the 48-hour rule: we calculate all wound timings
    relative to 'admittime'.
    """

    df = pd.read_csv(path, usecols=["hadm_id", "admittime"], low_memory=False)
    df["hadm_id"] = df["hadm_id"].astype("Int64")
    df["admittime"] = pd.to_datetime(df["admittime"])
    print("Admissions loaded:", len(df))
    return df


def find_wound_notes(discharge_path: str):
    """
    NLP Processing: Scan discharge summaries for wound keywords.

    Structured billing codes often under-report early-stag to improve
    sensitivity (recall) of our target labels.
    """
    wound_notes = []

    # Process in chunks to handle memory limits
    reader = pd.read_csv(
        discharge_path,
        usecols=["hadm_id", "charttime", "text"],
        chunksize=NOTES_CHUNKSIZE,
        low_memory=False,
    )

    for i, chunk in enumerate(reader, start=1):
        chunk = chunk.dropna(subset=["hadm_id"])
        chunk["hadm_id"] = chunk["hadm_id"].astype("Int64")

        # Lowercase for case-insensitive matching
        chunk["text"] = chunk["text"].astype(str).str.lower()

        # Apply Regex Pattern defined in WOUND_PATTERN
        chunk["is_wound"] = chunk["text"].apply(
            lambda t: bool(WOUND_PATTERN.search(t))
        )

        # Keep only positive matches to save memory
        wound_chunk = chunk[chunk["is_wound"]].copy()

        if not wound_chunk.empty:
            wound_chunk["charttime"] = pd.to_datetime(
                wound_chunk["charttime"], errors="coerce"
            )
            wound_notes.append(wound_chunk)

    if wound_notes:
        all_wound = pd.concat(wound_notes, ignore_index=True)
    else:
        all_wound = pd.DataFrame(columns=["hadm_id", "charttime", "text",␣
 ↪"is_wound"])

    print("Total wound notes found:", len(all_wound))
    return all_wound


def earliest_wound_note(all_wound: pd.DataFrame):
    """
    Identify the timestamp of the first mention of a pressure injury.

    This timestamp ('first_wound_note_time') is compared against 'admittime'
    to determine if the injury was present on admission or acquired later.
    """
    if all_wound.empty:
        return pd.DataFrame(columns=["hadm_id", "first_wound_note_time"])

    df = (
        all_wound
        .dropna(subset=["charttime"])
        .sort_values(["hadm_id", "charttime"])
        .groupby("hadm_id", as_index=False)
        .first()[["hadm_id", "charttime"]]
        .rename(columns={"charttime": "first_wound_note_time"})
    )
    print("Admissions with  1 wound note:", len(df))
    return df


def compute_unstructured_labels(adm: pd.DataFrame,
                                first_wound: pd.DataFrame):
    """
    Apply the Clinical 48-Hour Rule to NLP findings.

    Logic:
      1. Calculate 'hours_after_admit' = (First Note Time - Admit Time).
      2. If time > 48 hours: Labeled 'HAPI_UNSTRUCTURED' = 1 (Acquired).
      3. If time < 48 hours: Labeled 'HAPI_UNSTRUCTURED' = 0 (Present on␣
 ↪Admission).
    """
    labels = adm.merge(first_wound, on="hadm_id", how="left")

    # Calculate time change in hours
    labels["hours_after_admit"] = (
        labels["first_wound_note_time"] - labels["admittime"]
    ).dt.total_seconds() / 3600.0

    # Binary flag: Does note mention presure ulcer
    labels["notes_pu"] = labels["first_wound_note_time"].notna().astype(int)

    # Initialize Target as 0
    labels["HAPI_UNSTRUCTURED"] = 0
    mask = labels["hours_after_admit"].notna()

    # Only count as HAPI if documented AFTER the threshold (48h)
    labels.loc[
        (labels["hours_after_admit"] > HAPI_THRESHOLD_HOURS) & mask,
        "HAPI_UNSTRUCTURED"
    ] = 1
    # Mark Present on Admission (POA) as 0
    labels.loc[
        (labels["hours_after_admit"] <= HAPI_THRESHOLD_HOURS) & mask,
        "HAPI_UNSTRUCTURED"
    ] = 0

    print("Unstructured HAPI distribution (Based on 48h Rule):")
    print(labels["HAPI_UNSTRUCTURED"].value_counts(dropna=False))
    return labels


def add_structured_labels(labels: pd.DataFrame,
                          dx_path: str):
    """
    Merge in Structured Billing Codes (ICD-9/10).

    While billing codes are less granular regarding timing, they are␣
  ↪high-specificity
    indicators. We treat any billing code as a potential HAPI indicator and␣
  ↪combine
    it with our NLP findings for maximum recall.
    """
    dx = pd.read_csv(dx_path, low_memory=False)
    dx["icd_code"] = dx["icd_code"].astype(str).str.upper().str.strip()
    dx["hadm_id"] = dx["hadm_id"].astype("Int64")

    # Filter for Pressure Ulcer codes defined above
    pu_dx = dx[dx["icd_code"].isin(PRESSURE_ULCER_CODES)]

    pu_by_hadm = (
        pu_dx.groupby("hadm_id")
        .size()
        .reset_index(name="pu_count")
    )
    pu_by_hadm["HAPI_STRUCTURED"] = 1
    pu_by_hadm = pu_by_hadm[["hadm_id", "HAPI_STRUCTURED"]]

    # Merge into main label table
    labels = labels.merge(pu_by_hadm, on="hadm_id", how="left")
    labels["HAPI_STRUCTURED"] = labels["HAPI_STRUCTURED"].fillna(0).astype(int)

    print("HAPI_STRUCTURED distribution:")
    print(labels["HAPI_STRUCTURED"].value_counts(dropna=False))
    return labels
```

```
def combine_labels_and_save(labels: pd.DataFrame,
                            output_path: str):
    """
    Final Union of Target Variables.

    HAPI_FINAL = (Structured Code Present) OR (Unstructured Note > 48h).
    This creates the final binary classification target for the model.
    """
    labels["has_HAPI"] = (
        (labels["HAPI_STRUCTURED"] == 1) |
        (labels["HAPI_UNSTRUCTURED"] == 1)
    ).astype(int)

    labels["HAPI_FINAL"] = labels["has_HAPI"]

    print("HAPI_FINAL distribution (Class Balance):")
    print(labels["HAPI_FINAL"].value_counts(dropna=False))

    # Keep only analytic columns
    final = labels[[
        "hadm_id",
        "admittime",
        "first_wound_note_time",
        "hours_after_admit",
        "notes_pu",
        "HAPI_UNSTRUCTURED",
        "HAPI_STRUCTURED",
        "has_HAPI",
        "HAPI_FINAL",
    ]]

    final.to_csv(output_path, index=False)
    print("Saved label table to:", output_path)
    return final
```

[5]:
```
# Execution

def main():
    adm = load_admission_info(ADMISSIONS_PATH)
    all_wound = find_wound_notes(DISCHARGE_PATH)
    first_wound = earliest_wound_note(all_wound)

    # Compute labels based on logic
    labels = compute_unstructured_labels(adm, first_wound)
    labels = add_structured_labels(labels, DIAGNOSES_PATH)

    combine_labels_and_save(labels, OUTPUT_PATH)
```

```python
if __name__ == "__main__":
    main()
```

```
Admissions loaded: 546028
Total wound notes found: 195795
Admissions with  1 wound note: 195795
Unstructured HAPI distribution (Based on 48h Rule):
HAPI_UNSTRUCTURED
0    412105
1    133923
Name: count, dtype: int64
HAPI_STRUCTURED distribution:
HAPI_STRUCTURED
0    537574
1      8454
Name: count, dtype: int64
HAPI_FINAL distribution (Class Balance):
HAPI_FINAL
0    408955
1    137073
Name: count, dtype: int64
Saved label table to: D:\School\5141\hospitalwide_hapi_labels.csv
```