Semantic Column Classification & Parsing

Prathap Rane
ME22B056

1. Problem, Data & System Overview

Problem Setting

- Input: arbitrary CSV files with weak / missing / misleading column names.
- Goal (Part A): infer the **semantic type** of a column as one of:
  - `PhoneNumber`, `CompanyName`, `Country`, `Date`, `Other`.
- Goal (Part B): once types are known, **parse & normalize** selected columns to a clean schema.

Training Data (`data/`)

- `phoneNumber.csv` – example phone numbers in various formats.
- `Company.csv` – company names.
- `Dates.csv` – date strings in multiple formats (e.g. `2024-11-27`, `01/01/2000`).
- `Countries.txt` – list of country names.
- `legal.txt` – legal suffixes (e.g. `pvt ltd`, `gmbh co kg`, `ltd`, `llc`).

High-level Architecture

- **Training pipeline**
  - Build a labeled dataset: each training value is tagged as `PHONE` / `COMPANY` / `COUNTRY` / `DATE` / `OTHER`.
  - Extract features:
    - Character-level TF–IDF (n-grams of length 2–4).
    - Numeric text statistics (length, digit / letter ratios, punctuation ratio, date-like score, etc.).
  - Train a multinomial Logistic Regression classifier on these features.
  - Persist the trained pipeline as `models/semantic_classifier.joblib`.
- **Serving components**
  - `predict.py`: Part A – classify a single column in a CSV and print only the semantic type.
  - `parser.py`: Part B – detect best Phone / Company columns and produce normalized output.
  - `mcp_server.py` (optional): expose the same functionality as MCP tools for external clients.

2. Model Design, Column Logic & Parsing

Feature Engineering & Model

- **Character-level TF–IDF**
  - Computed on raw cell text with n-grams of length 2–4.
  - Captures patterns such as `+91`, `044`, `pvt`, `/20`, etc.
- **Text statistics** (from `text_features.py`):
  - Total length of the string.
  - Fraction of digits / letters / punctuation.
  - Longest run of consecutive digits.
  - Presence of special characters like "+" or brackets.
  - Date-likeness score; approximate match to country names from `Countries.txt`.
- **Model:** features are combined via `FeatureUnion` and fed to:
  - Multinomial Logistic Regression (fast to train, robust with sparse features, good probability calibration).

Column-level Semantic Classification (Part A)

- For a given column:
  - Sample up to 200 non-empty cells.
  - For each cell, predict the probability over the 5 classes.
  - Aggregate per-cell probabilities into a column-level score using:
    - Confidence-weighted averaging of class probabilities.
    - Majority vote over the most likely per-cell labels.
  - Apply minimum probability and margin thresholds:
    - Decide between `PhoneNumber`, `CompanyName`, `Country`, `Date`, `Other`.
    - If no class is sufficiently strong, fall back safely to `Other`.
- `predict.py` wraps this logic and prints the final semantic type as a single line (as required by the assignment).

Parsing & Normalization Workflow (Part B)

- **Phone columns**
  - Use the column classifier to select the best phone column in the file.
  - For each row (`parse_phone_value`):
    - Strip common extensions (e.g. `ext`, `x`, `#`).
    - Detect and skip obvious date-like strings (DOBs placed in the phone column by mistake).
    - If the value starts with "+", infer `Country` from the dialing code (e.g. `+91` → India).
    - Normalize the main `Number` to digits-only and apply simple length checks.
- **Company columns**
  - Use the same column classifier logic to select the best company column.
  - Normalize text and split into:
    - `Name` (base company name),
    - `Legal` (legal suffix) using longest-suffix match from `legal.txt`.
- **Final output schema** produced by `parser.py`:
  - Columns: `PhoneNumber, Country, Number, CompanyName, Name, Legal`.
  - If a file has only phones or only companies, irrelevant columns are left blank.
  - Original values are preserved in `PhoneNumber` and `CompanyName` for traceability.