**DOCUMENT STANDARDIZATION**

The task is to build a simple solution that recognizes headers in documents using ML.

**1. Solution Overview**

The goal is to distinguish header rows in tabular data by utilizing distinct text and structure patterns. The solution includes the following key steps:

* **Data Loading**: Processes JSON data in chunks to efficiently handle large datasets.
* **Labeling**: Labels rows as headers (1) or non-headers (0) for training.
* **Feature Engineering**: Extracts row-level features like text patterns to distinguish headers.
* **Class Imbalance Handling**: Uses SMOTE to balance the dataset between headers and non-headers.
* **Model Training**: Trains a RandomForestClassifier pipeline combined with SMOTE for interpretability and robust performance.
* **Evaluation**: Measures performance with metrics like accuracy, precision, recall, and F1-score.

**2. Data Analysis and Feature Engineering**

**Data Analysis**

Exploratory analysis revealed structural and content differences in header rows. For instance, headers tend to have more uppercase letters, fewer digits, and unique characters, which form the basis of feature selection.

**Feature Engineering**

* **Chunk Loading**: Processes data in manageable chunks to optimize memory use.
* **Generated Features**:
  + **num\_columns**: Number of non-empty columns in each row.
  + **text\_length**: Total character count across cells in a row.
  + **digit\_count**: Digit frequency within the row’s text, as headers typically have fewer numbers.
  + **capitalization**: Count of uppercase characters, often more common in headers.
  + **special\_chars**: Counts special characters, which are also more typical in header rows.

This approach captures structural patterns and content differences to enhance model performance.

**3. Alternative Approaches Considered**

Several models were considered before selecting RandomForestClassifier:

* **Logistic Regression**: Discarded due to limitations in capturing non-linear patterns.
* **Naive Bayes**: Found unsuitable as it assumes feature independence, which doesn’t align with header patterns.
* **Random Forest**: Chosen for its ability to capture non-linear relationships and interpret feature importance effectively.

### 4. Evaluation and Results

* **Accuracy**: At 92%, the model is effectively distinguishing between headers and non-headers, especially considering the large dataset.
* **Precision and Recall**:
* **Precision (0 class - Non-Header)**: 0.93
* **Recall (0 class - Non-Header)**: 0.92
* **Precision (1 class - Header)**: 0.92
* **Recall (1 class - Header)**: 0.93

This balance in precision and recall for both classes (0 and 1) suggests the model is accurately identifying header rows without heavily favoring one class over the other.

**5. Future Improvements**

**Feature Expansion**: NLP-based features, like word embeddings, could add value by capturing header-specific patterns.

**6. Repository and Usage**

Storing code, models, and datasets in a versioned repository (e.g., GitHub) allows easy access and reproducibility.

<https://github.com/Susma261/Header-Standardization>

**7. Prerequisites**

**Python Version**: 3.7 or later  
**Libraries**: Make sure these libraries are installed in your Jupyter Notebook environment.

Run the following code in a Jupyter Notebook cell to install the required libraries if they aren’t already installed:

Install required libraries in the notebook environment

!pip install pandas numpy scikit-learn imbalanced-learn joblib

**System Requirements**

* **Memory**: Ensure your system has at least 4 GB of RAM to handle larger datasets.
* **Storage**: Ensure enough storage space for dataset files and model artifacts.

**Step-by-Step Execution in Jupyter Notebook**

1. **Data Preparation**: Place your training data in an accessible location (e.g., data/training\_data.txt or data/training\_data.json).
2. **Load Data, Train the Model, and Display Evaluation Metrics**: Run each code block as separate cells in the notebook.

### 8. Testing with New Data

Once test data is available, we’ll load it, preprocess it, and use the trained model to predict which rows are headers. This will follow a similar process to training, except we’ll skip model training and directly load the model.

**9. Accessing Model and Results**

After execution, results are printed in the console, including accuracy and classification metrics. The trained model (header\_recognition\_model.pkl) is saved and can be used to classify headers in new data files with:

model = joblib.load('header\_recognition\_model.pkl')

This solution provides a robust approach for header row classification, backed by well-structured features and a tested model.