University of Science - VNUHCM

Faculty of Information Technology

COURSE MILESTONE 2

CSC14119 – Introduction to Data Science

Student in charge:

Cao Tấn Hoàng Huy - 23127051

Nguyễn Hữu Anh Trí - 23127130

Nguyễn Huy Quân - 23127107

Instructors:

Huỳnh Lâm Hải Đăng

Nguyễn Thanh Tình

Nguyễn Ngọc Thảo

Table of Contents

1 Introduction

This report presents the design and implementation of a data processing and analysis pipeline for the second milestone of the **Introduction to Data Science course**. Building on the previous milestone's data collection, the main objective here is to transform unstructured raw data—specifically, full-text LaTeX source files from scientific papers—into a structured, hierarchical format and to apply machine learning techniques for reference matching.

The project centers on **arXiv**, a large public repository of research papers. Unlike typical projects that only process metadata or PDF content, our system is designed to parse the **full-text LaTeX source files** for all versions of each paper. This enables advanced tasks such as detailed text analysis, hierarchical content modeling, and accurate citation network construction.

The processing pipeline consists of several key stages: (1) automatic shallow cleaning and extracting references before converting LaTeX projects to a hierarchical document structure, (2) standardization and deduplication of full-text content, and (3) matching extracted references to external metadata using a supervised machine learning approach.

The remainder of this report details the parsing and standardization logic, describes the machine learning pipeline for reference matching, and presents an evaluation of the system's performance using the Mean Reciprocal Rank (MRR) metric.

2 Implementation Details

This section details the technical implementation of the scraper, outlining the tools used and the sequential workflow for processing each arXiv paper.

2.1 Tools and Libraries

The data processing and analysis pipeline is implemented in Python within a Jupyter Notebook, leveraging a comprehensive suite of libraries and modules for file handling, parsing, data cleaning, machine learning, and visualization:

- **File and Directory operations:**
  - `os`: Provides functions for interacting with the operating system, such as directory creation and file management.
  - `pathlib.Path`: Offers an object-oriented interface for filesystem paths, enabling robust and platform-independent file and directory operations.
- **Data serialization and structure:**
  - `json`: Used for reading and writing structured data in JSON format, both for intermediate results and final outputs.
  - `dataclasses.dataclass`, `dataclasses.field`: Facilitate the creation of structured, type-annotated data containers for complex objects.
  - `collections.defaultdict`: Provides a dictionary subclass for grouping and aggregating data efficiently.
  - `copy.deepcopy`: Ensures safe duplication of nested and complex data structures during processing.
- **Text processing and parsing:**
  - `re`: Enables advanced regular expression operations for parsing LaTeX source files, extracting references, and cleaning content.
  - `bibtexparser`, `bibtexparser.bparser.BibTexParser`: Specialized libraries for parsing, standardizing, and manipulating BibTeX entries extracted from LaTeX files.
  - `string`: Provides string constants and utility functions for text normalization and manipulation.
  - `difflib.SequenceMatcher`: Calculates string similarity ratios, crucial for reference matching and deduplication.
- **Unique identification and Hashing:**
  - `hashlib`: Used to generate hash values for deduplication of hierarchical elements and references.

  - `uuid`: Generates universally unique identifiers (UUIDs) for hierarchical elements and references, ensuring consistency across versions.

- **Type hints and Static analysis:**
  - `typing`: Provides type hints such as `List`, `Dict`, `Tuple`, `Optional`, `Set`, and `Any` to improve code readability, maintainability, and static analysis.

- **Progress Monitoring and Logging:**
  - `tqdm.auto`: Displays progress bars for loops and long-running operations, enhancing user feedback.
  - `logging`: Configures and suppresses log messages, particularly from third-party libraries like `bibtexparser`, to keep notebook output clean.
  - `warnings`: Suppresses or manages warning messages to avoid cluttering the output.

- **Numerical and Tabular Data Processing:**
  - `numpy`: Provides efficient numerical operations and array handling, supporting data transformation and feature engineering.
  - `pandas`: Enables powerful tabular data manipulation, cleaning, and analysis.

- **Machine learning and Feature engineering:**
  - `sklearn.model_selection.train_test_split`: Splits data into training and test sets for model evaluation.
  - `sklearn.preprocessing.StandardScaler`: Standardizes features by removing the mean and scaling to unit variance.
  - `sklearn.linear_model.LogisticRegression`: Implements logistic regression for supervised classification tasks.
  - `sklearn.metrics`: Provides functions for evaluating model performance, including accuracy, precision, recall, and F1 score.
  - `sklearn.feature_extraction.text.TfidfVectorizer`: Converts text data into TF-IDF feature vectors for machine learning.

- **Visualization and Statistical analysis:**
  - `matplotlib.pyplot`: Generates plots and visualizations for exploratory data analysis and result presentation.
  - `scipy.stats`: Offers statistical functions and hypothesis testing tools for data analysis.

These libraries collectively enable robust parsing, hierarchical structuring, reference matching, machine learning, and comprehensive analysis throughout the project workflow.

2.2 Pipeline Overview

The data processing workflow is implemented entirely within a Jupyter Notebook and is divided into two main stages: (1) LaTeX-to-hierarchy processing and (2) reference matching using machine learning. Each stage is modular, with clearly defined objectives and outputs.

1. **Stage 1: LaTeX-to-Hierarchy processing Pipeline**
   - **Version Discovery:** For each paper, all available versions are identified by scanning the `tex/` subfolder within the raw source directory.
   - **LaTeX Expansion:** The pipeline recursively expands all `\input` and `\include` commands across all versions, inlining content from referenced files to reconstruct the complete document for each version.
   - **Text Preprocessing:** Comments are removed, formatting is normalized, and boilerplate or non-semantic LaTeX commands are cleaned from the expanded text to ensure consistency and reduce noise.
   - **Reference Extraction:** Both BibTeX files (`.bib`) and in-text `\bibitem` blocks are parsed to extract citation entries, which are then deduplicated globally across all versions.
   - **Hierarchy Construction:** The cleaned, expanded text is parsed using regular expressions to detect and organize structural elements (sections, subsections, sentences, equations, figures, etc.) into a hierarchical tree structure, with unique identifiers for each node.

- **Element Normalization and Deduplication:** Hierarchical elements are deduplicated across versions using fingerprinting and string similarity, ensuring that identical content is represented only once.
- **Output Generation:** For each paper, the pipeline produces a `hierarchy.json` file (capturing the full document tree), a deduplicated `refs.bib` file, and copies of the original `metadata.json` and `references.json` files for downstream use.

Pipeline Flow:

`Raw LaTeX` → `Version Discovery` → `LaTeX Expansion` → `Preprocessing` → `Reference Extraction` → `Hierarchy Building` → `Deduplication` → `Output (hierarchy.json, refs.bib, metadata.json, references.json)`

2. **Stage 2: Reference Matching Pipeline**

- **Text Normalization:** All reference strings are normalized (LaTeX to plain text, lowercasing, punctuation removal) to facilitate accurate matching.
- **Label Generation:** Ground truth labels for reference-to-arXiv ID matching are created using a combination of manual annotation and automatic heuristics.
- **Feature Engineering:** Discriminative features (e.g., string similarity scores, token overlap, TF-IDF vectors) are computed for each candidate reference/arXiv ID pair.
- **Model Training:** A logistic regression classifier is trained to predict binary matches between extracted references and arXiv IDs.
- **Ranking and Evaluation:** The trained model ranks candidate matches for each reference, and the Mean Reciprocal Rank (MRR) is computed on the test set to evaluate performance.
- **Output Generation:** The top-5 ranked predictions for each reference are saved in `pred.json` files for further analysis and validation.

Pipeline Flow:

`Input Data` → `Text Normalization` → `Label Generation` → `Feature Extraction` → `Model Training` → `Ranking` → `pred.json Output` → `MRR Evaluation`

Each stage is designed to be modular, reproducible, and extensible, ensuring robust processing from raw LaTeX sources to final reference matching outputs.

3 Pipeline Detail

3.1 Stage 1 - LaTeX-to-Hierarchy processing Pipeline

This stage transforms raw, multi-version LaTeX sources into a clean, deduplicated, and structured representation, providing a robust foundation for downstream reference matching and analysis.

3.1.1 The core problems: Multi-file, Multi-version, and Noisy LaTeX Sources

Scientific papers are distributed as multiple versions, each with potentially complex directory structures and multiple `.tex` files. The logical structure (sections, equations, references) is not explicitly encoded, and the presence of non-semantic formatting commands, comments, and boilerplate further complicates parsing. The main challenges are:

- **Version management:** Each paper may have multiple versions, each with its own directory and main file.
- **Recursive expansion:** The full document is often split across many files, requiring recursive expansion of all `\input` and `\include` commands.
- **Noisy content:** Comments, formatting commands, and boilerplate must be removed to enable accurate parsing.
- **Reference Extraction:** Citations may appear as `\bibitem` blocks or in external `.bib` files, requiring robust extraction and deduplication.
- **Cross-version Deduplication:** Identical elements and references must be deduplicated across all versions.

3.1.2 The Solution: Step-by-step Pipeline

1. **Folder preparation and Version discovery**

- **Goal:** Ensure all papers and their versions are discoverable and output is organized for downstream tasks.
- **Solution:** Set `DATA_DIR` and `OUTPUT_DIR`. Scan for all paper folders and, within each, enumerate all version subfolders under `tex/`. (In our Milestone 2, we set the Data directory is **23127130** (from the last Milestonne 1 and the Output directory is **Output**)

```
DATA_DIR = Path("23127130")
OUTPUT_DIR = Path("output")
```

Figure 1: Setting directories.

- **Reasoning:** Guarantees that the pipeline can process all available data and that results are stored in a reproducible, organized manner.

2. **LaTeX expansion**

- **Goal:** Reconstruct the full logical content of each paper version, regardless of file structure.
- **Solution:** For each version, identify the main `.tex` file (priority: contains `\documentclass`, then filename contains "main", else first `.tex` file). Recursively expand all `\input` and `\include` commands using regular expressions, inlining content from referenced files, and tracking visited files to avoid cycles.

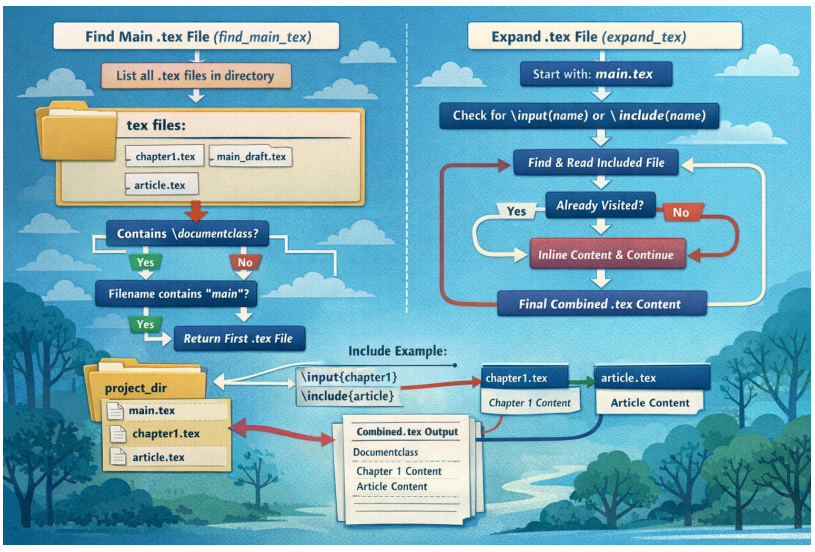

Figure 2: Demonstrate the process of getting main Tex.

- **Reasoning:** Ensures the entire document, including all included sections, is available as a single text string for further processing.

3. **Text preprocessing**

- **Goal:** Remove noise and standardize the text for accurate parsing and reference extraction.
- **Solution:** The following steps are applied in order:
  (a) Remove comments (lines starting with % but not escaped as `\%`).
  (b) Remove preamble (everything before `\begin{document}`), but keep the document start.
  (c) Remove the abstract environment (`\begin{abstract}...\end{abstract}`).
  (d) Normalize line endings to Unix style.
  (e) Remove title and author blocks (from `\title{` up to the first section or abstract).
  (f) Remove boilerplate commands (`\maketitle`, `\ACMmaketitle`, etc.).
  (g) Remove any remaining `\input` or `\include` commands (already expanded).
  (h) Remove all `\label{}` commands.
  (i) Normalize references: `\eqref{}` → [EQ], `\ref{}` → [REF].
  (j) Normalize citations: `\cite{key}` → [CITE:key].
  (k) Remove formatting commands without semantic meaning (`\centering`, `\raggedright`, etc.).
  (l) Remove float options (e.g., [h], [t], [b], [p], [!]).
  (m) Unwrap text formatting commands, keeping only the content (`\textbf{...}` → ...).
  (n) Normalize inline math (`\(...\)` and $...$) to $...$.
  (o) Normalize block math: convert `$$...$$`, `\[...\]`, and environments like `align`, `gather`, `multline` to `\begin{equation}...\end{equation}`.
  (p) Normalize whitespace (collapse spaces, remove extra blank lines).
- **Reasoning:** Each step targets a specific source of noise or inconsistency, ensuring the text is clean, semantically meaningful, and ready for robust parsing and reference extraction.

4. **Reference Extraction and Deduplication**

- **Goal:** Extract all references from both in-text `\bibitem` blocks and external `.bib` files, and deduplicate them globally.
- **Solution:**
  (a) Use regex to extract `\bibitem` blocks and parse each item.
  (b) Use regex to find `\bibliography{...}` commands and parse the corresponding `.bib` files using `bibtexparser`.
  (c) Normalize each reference (lowercase, remove punctuation, collapse whitespace) and generate a fingerprint hash based on title, authors, and year.
  (d) Merge duplicate references across all versions by fingerprint, combining fields and source keys.
  (e) Remove all reference blocks and bibliography commands from the LaTeX text to avoid interference with hierarchy parsing.
  (f) Save the deduplicated references in a single `refs.bib` file in BibTeX format for each paper.

Figure 3: Steps to extract and deduplicate references.

- **Reasoning:** This ensures that each unique reference is represented only once, providing a clean, non-redundant input for the reference matching stage and reducing ambiguity.

5. **Hierarchy Construction**

- **Goal:** Build a machine-readable, structured representation of the document's logical structure, capturing both high-level and atomic elements.
- **Solution:**
  (a) Define a hierarchy schema (`HIERARCHY_LEVELS`) specifying levels, atomicity, and regex signals for each structural element (section, subsection, subsubsection, paragraph, figure, block formula, sentence).

(b) Parse the cleaned LaTeX text using these regex patterns, extracting tokens for each element and marking their positions.

(c) Use a stack-based tree construction algorithm: for each token, determine its parent based on hierarchy level, and attach as a child node.

(d) For text not captured by higher-level structures, split into sentences using regex and emit as leaf nodes.

(e) Assign a unique identifier (UUID) to every node.

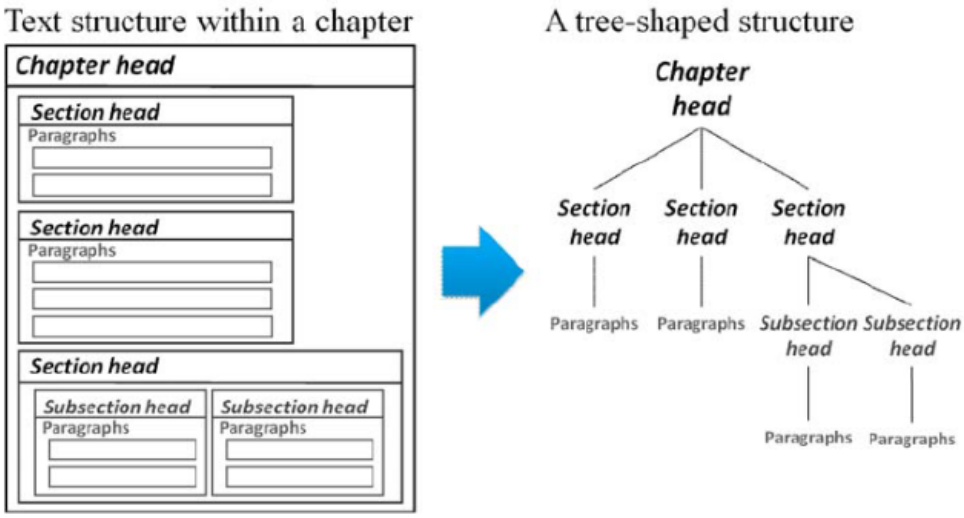

Figure 4: An example of a tree after being built. [4]

- **Reasoning:** This approach robustly reconstructs the logical structure of complex scientific documents, enabling downstream tasks such as fine-grained reference matching and content analysis.

6. **Element normalization and Cross-version Deduplication**

- **Goal:** Ensure that identical elements across all versions are represented only once, with canonical IDs.
- **Solution:**

(a) Traverse all hierarchy trees and normalize each node's content (lowercase, collapse whitespace).

(b) Generate a fingerprint hash for each node based on its type and normalized content.

(c) Assign a canonical element ID for each unique fingerprint, and map parent-child relationships for each version.

- **Reasoning:** This reduces redundancy, enables efficient cross-version analysis, and ensures that the same content is not processed multiple times in downstream tasks.

7. **Output Generation**

- **Goal:** Produce a standardized, comprehensive set of output files for each paper, ready for Stage 2.
- **Technique:**

(a) Save the merged hierarchy (elements and relationships) as `hierarchy.json`.

(b) Save deduplicated references as `refs.bib`.

(c) Copy `metadata.json` and `references.json` from the source to the output directory.

- **Reasoning:** This ensures all downstream processes have access to clean, structured, and deduplicated data for each paper.

3.1.3 Output directory structure

The result of Stage 1 is a well-organized output directory for each paper, with the following structure:

```
output/
|-- {paper_id}/
|   |-- hierarchy.json      # Hierarchical document structure (sections, equations, etc.)
|   |-- refs.bib            # Deduplicated references in BibTeX format
|   |-- metadata.json       # Paper metadata (title, authors, etc.)
|   |-- references.json     # Candidate references for matching
```

Each file serves a distinct purpose:

- `hierarchy.json`: Encodes the full logical structure of the paper, including sections, subsections, equations, and sentences, with canonical element IDs.
- `refs.bib`: Contains all deduplicated references extracted from all versions, formatted in BibTeX for downstream matching.

- `metadata.json`: Stores the original paper metadata, such as title, authors, and abstract.
- `references.json`: Lists candidate references for use in the reference matching pipeline.

This structured output provides a robust and standardized foundation for the reference matching and machine learning pipeline in Stage 2.

3.2 Stage 2 - References Mathching pipeline

This stage is responsible for resolving the ambiguity between raw citations extracted from LaTeX files and the clean, structured metadata of arXiv papers. The logic is implemented as a supervised machine learning pipeline, transforming the reference matching task into a binary classification problem to rank potential candidates.

3.2.1 The Core Problem: Ambiguous and Noisy Citation Data

Citations in scientific writing (`refs.bib`) are often manually typed by authors, leading to significant inconsistency compared to the standardized metadata (`references.json`). The logical link between a citation key (e.g., `simonyan2014very`) and its actual paper is not always explicit. The main challenges are:

- **Inconsistent Formatting:** Titles in BibTeX often contain LaTeX commands (e.g., `\textit{}`), varied capitalization, and punctuation styles that prevent exact string matching.
- **Author Name Variance:** Authors may be listed by full name, initials, or "Lastname, Firstname" formats, making direct comparison difficult.
- **Missing Identifiers:** Many citations lack unique identifiers like DOIs, requiring text-based matching.
- **Class Imbalance:** For any given citation, there is usually only one correct match out of thousands of candidates, creating a highly imbalanced dataset.
- **Scalability:** A naive comparison of every citation against every candidate is computationally expensive; efficient filtering and ranking are required.

3.2.2 The Solution

The implemented solution approaches matching as a binary classification task: given a pair *(Citation, Candidate)*, predict the probability that they refer to the same document. The pipeline consists of the following detailed steps:

1. **Data cleaning:**

   - **Implementation:** We utilized a custom `TextCleaner` module to standardize raw strings. The process begins by stripping LaTeX artifacts (e.g., `\textit{}`, `\textbf{}`) using regular expressions while preserving the inner content. Next, the pipeline normalizes Unicode characters, converts text to lowercase, and removes punctuation. Finally, the text is tokenized, and a curated list of stop words—including domain-specific terms like "method" and "approach"—is filtered out.
   - **Reasoning:** Raw string comparison is brittle in bibliographic data. By removing non-semantic noise (formatting commands) and standardizing character representations, we reduce false negatives caused by stylistic differences (e.g., "Naive Bayes" vs. "*naive-bayes*"), ensuring that only the core semantic content remains for comparison.

2. **Data Labeling**

   - **Goals:** To train the supervised model, we require a dataset of labeled pairs *(Reference, Candidate)* indicating positive or negative matches. Given the volume of data, manual annotation is infeasible for the training set. Therefore, we adopt a **hybrid labeling strategy**: precise manual labeling for evaluation and heuristic-based automatic labeling for training.
   - **Implementation:** We utilized a **hybrid labeling strategy**. For the *Training Set*, we implemented an automatic "waterfall" algorithm that attempts to match papers using three strategies in decreasing order of confidence:
     (a) **DOI Matching:** Checks for exact matches in Digital Object Identifiers.
     (b) **Exact Title Matching:** Compares titles after processing them through the `TextCleaner`.
     (c) **Fuzzy Title Matching:** Uses `SequenceMatcher` to accept pairs with a similarity ratio.

For the *Validation and Test Sets*, we explicitly bypassed automation and performed manual verification of `refs.bib` entries against `references.json` to ensure a pure accurate set.

Internal Label Format: The resulting ground truth is serialized in a hierarchical dictionary structure to maintain the relationship between the source paper and its specific citations. The format matches the schema:

```
{
    "source_paper_id": {
        "bibtex_acitation_key": "target_arxiv_id"
    }
}
```

- **Reasoning:** The fundamental essential for supervised learning is the availability of ground truth labels to train the classifier's decision boundary. Without these labels, the model cannot distinguish between matching and non-matching citations. Since manually labeling thousands of pairs to establish this ground truth is computationally infeasible, the automatic labeling pipeline provides the necessary volume of "Silver Standard" training data, while manual labeling is reserved for evaluation to ensure rigorous testing.

3. **Feature Engineering:**

- **Goal:** To transform the raw textual and metadata pairs into dense numerical vectors. The objective is to capture specific signals-lexical exactness, semantic overlap, and bibliographic consistency-that distinguish true matches from deceptive non-matches.
- **Implementation:** We designed 6 specific features [2], each designed to address a specific type of ambiguity found in citation data:

  (a) **Title String Similarity:**

  – *Definition:* The character-level sequence similarity ratio (0-1) between normalized titles.

  – *Justification:* This is the primary signal for high-precision matching. Valid citations usually possess titles nearly identical to the source. This feature captures

exact matches while being robust to minor character-level noise (e.g., typos, OCR errors) that would break a strict string equality check.

(b) **Title Word Overlap:**

- *Definition:* The Jaccard similarity coefficient of the unique word sets in both titles.
- *Justification:* Citations often reorder words or change phrasing (e.g., "Analysis of X" vs. "X: An Analysis"). Unlike sequence similarity, this feature is order-invariant, allowing the model to detect semantic matches even when the syntactic structure differs significantly.

(c) **Title Length Ratio:**

- *Definition:* The ratio of the shorter title's length to the longer one ($\frac{min}{max}$).
- *Justification:* This acts as a structural filter. True matches generally have similar lengths. A low ratio often indicates a mismatch between a full paper title and an abbreviated reference or a completely different document type, providing a strong negative signal.

(d) **Author Name Overlap:**

- *Definition:* The Jaccard similarity of the set of normalized author last names.
- *Justification:* This serves as a critical tie-breaker. In scientific literature, generic titles (e.g., "Introduction to Deep Learning") are common. This feature ensures that a candidate is not just topically similar to the citation but bibliographically linked to the same authors.

(e) **First Author Match:**

- *Definition:* A binary flag (0 or 1) indicating if the first author's last name matches exactly.
- *Justification:* Citations frequently use the "First Author et al." format. A mismatch in the first author position is a highly reliable indicator of a negative pair, helping to distinguish between different papers written by different groups on the same topic.

(f) **Publication Year Proximity:**

- *Definition:* The normalized absolute difference between publication years.

  - *Justification:* This provides a temporal consistency check. It helps resolve ambiguity when a candidate paper has a similar title and author but was published decades apart from the citation, indicating it is likely a different version or a legacy reference.

- **Data Analysis & Feature Verification:**
  - **Goal:** To empirically validate the feature engineering strategy. We analyzed the distribution of our features to confirm that splitting them into lexical (Title), bibliographic (Author), and temporal (Year) categories provides the necessary signal coverage to distinguish matches from non-matches.
  - **Visual Analysis:** The feature distributions (Figure 5) confirm a distinct separation between classes, validating the feasibility of using a linear classifier.

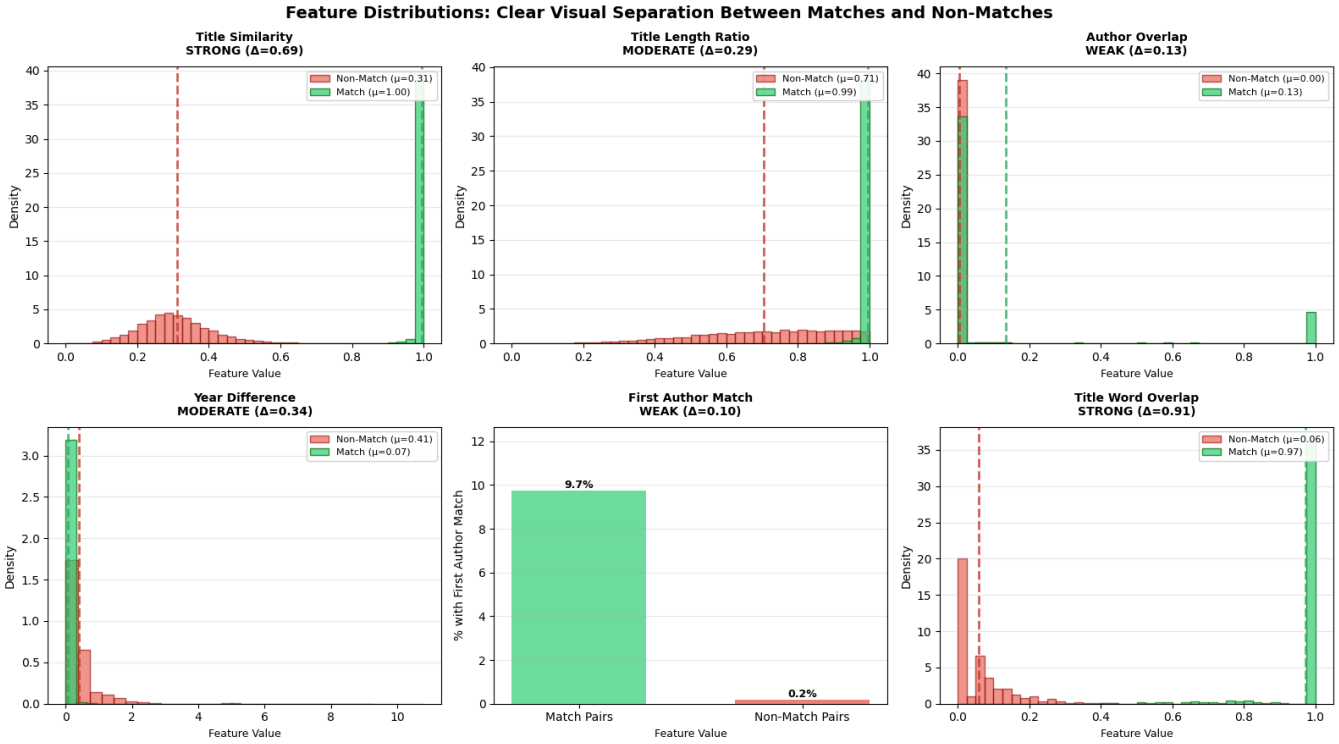

Figure 5: Title-based features show the strongest separation.

  - **Quantitative Validation:** To justify our feature choices, we calculated the "Separation Score" (Δ) for each feature. The results (Table 1) categorize our features into three distinct roles.

Table 1: Feature Performance Summary sorted by Discriminative Quality

Feature Role	Feature Name	Separation (Δ)	Quality
Primary Signal	`title_word_overlap`	7.095	Excellent
Primary Signal	`title_similarity`	6.344	Excellent
Structural Filter	`title_length_ratio`	1.476	Good
Temporal Filter	`year_difference`	0.542	Good
Tie-Breaker	`author_overlap`	0.391	Weak
Tie-Breaker	`first_author_match`	0.319	Weak

  - **Justification of Feature Split:**
    (a) **Title Features (The Primary Driver):** The high separation scores ($\Delta > 6.0$) justify our reliance on title features as the primary matching signal. The data confirms that despite minor formatting noise, correct citations maintain high lexical and semantic fidelity.
    (b) **Author Features (The "Noisy" Backup):** The low separation scores ($\Delta < 0.4$) validate our decision to treat author data as a secondary "tie-breaker" rather than a strict filter. The analysis revealed an "Author Paradox": while semantically important, BibTeX data quality issues (missing fields, inconsistent naming) make these features noisy. By separating them, we prevent parsing errors from incorrectly penalizing valid matches.
    (c) **Year & Length (The Sanity Checks):** The moderate scores confirm these features function effectively as negative filters. They justify their inclusion to penalize "anachronistic" matches (e.g., correct title but wrong decade) or structural mismatches that title similarity alone might miss.

- **Reasoning:** Our exploratory data analysis revealed that relying on a single metric is insufficient for high-recall matching. While *Title Similarity* is the most discriminative feature, it fails in cases of generic titles. Where many distinct papers share identical names. By integrating *Author Overlap* and *Year Proximity*, we provide the model with "tie-breaking" signals to resolve these ambiguities. Furthermore, the combination of strictly structural features (Length Ratio) with fuzzy logic (Sequence Similarity) ensures the pipeline is robust against the specific noise found in user-generated BibTeX files, such as missing fields or aggressive abbreviation.

4. **Model Training (Logistic Regression)[1]:**

- **Goal:** To learn a decision boundary that can predict the probability of a match $P(y = 1|x)$ for any given pair of citation and candidate paper. The objective is to assign weights to the engineered features, allowing the system to weigh strong signals (like an exact title match) against weaker ones (like a year discrepancy) to rank candidates effectively.
- **Implementation:** We selected a **Logistic Regression** classifier from the `scikit-learn` library.
  - **Input:** The model is trained on the feature matrices generated in the previous step, where each row represents a (BibTeX, Candidate) pair and columns correspond to the six engineered features.
  - **Configuration:** We initialized the model with `class_weight='balanced'`. This parameter automatically adjusts weights inversely proportional to class frequencies, assigning a higher penalty to misclassifying the minority class (actual matches).
  - **Output:** The model outputs a continuous probability score between 0 and 1, which we use to rank all potential candidates for a specific citation.
- **Reasoning:** We chose Logistic Regression over more complex models (like Random Forests or Neural Networks) for three key reasons:
  (a) **Probabilistic Ranking:** Our end goal is not just binary classification but *information retrieval*. We need to rank candidates to find the "best" match. Logistic Regression naturally outputs calibrated probabilities, making it ideal for ranking tasks.
  (b) **Interpretability:** Unlike "black box" models, Logistic Regression provides explicit coefficients for each feature. This allows us to verify the model's logic (e.g., confirming that `Title Similarity` has a high positive weight while `Year Difference` has a negative weight), ensuring the system behaves predictably.
  (c) **Handling Class Imbalance:** The matching problem is inherently imbalanced (one correct match vs. dozens of incorrect candidates). Logistic Regression handles this efficiently via class weighting, ensuring the model doesn't simply default to predicting "No Match" to achieve high accuracy.

5. **Model Evaluation (Mean Reciprocal Rank):**

   - **Goal:** To quantitatively assess the model's ranking quality. In an information retrieval system, simple accuracy is insufficient because the position of the correct answer matters; we aim to measure not just if the correct paper was found, but how high it appears in the suggested list.
   - **Implementation:** We utilized the Mean Reciprocal Rank (MRR) metric evaluated over the top-5 predicted candidates. The metric is calculated using the formula:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

     where $|Q|$ is the total number of citation queries and rank_i is the position (1-indexed) of the ground truth paper in the predicted list.

     The evaluation logic proceeds as follows:

     – For each citation in the test set, the model predicts probabilities for all candidates and sorts them in descending order.
     – We check the rank of the correct match:
       * **Rank 1:** Score = 1.0 (Perfect prediction).
       * **Rank 2:** Score = 0.5.
       * **Rank 3:** Score $\approx$ 0.33.
       * **Not in Top-5:** Score = 0.0.
     – The final score is the average of these reciprocal ranks across all queries.

4 Evaluation

4.1 Model Classification Performance

- **Goal:** To assess the supervised model's ability to correctly classify individual (Citation, Candidate) pairs. We utilize standard binary classification metrics-Precision, Recall, and F1-Score-to measure how effectively the model distinguishes between true references and noise.

- **Implementation:** We evaluated the trained Logistic Regression model on the **Validation Set**, comprised of 866 labeled pairs (34 positive matches, 832 negative candidates). The model predictions were compared against the manually verified ground truth.
- **Quantitative Results:** The model achieved near-perfect performance, identifying every true match in the validation set without missing any.

Table 2: Validation Set Performance Metrics

Metric	Score	Interpretation
Accuracy	0.999	Overall correctness (high due to class imbalance)
Precision	0.971	97.1% of predicted matches were correct
Recall	**1.000**	**100% of actual matches were found**
F1 Score	0.986	Harmonic mean of Precision and Recall

- **Feature Weight Analysis (Why it works):** To understand the model's decision logic, we extracted the learned coefficients (weights). The magnitude of these weights confirms the insights from our Exploratory Data Analysis:
  - **Title Similarity (+17.92):** This feature has a massive positive weight, dwarfing all others. It indicates that the model has learned that character-level sequence similarity is the single most reliable predictor of a match.
  - **Title Word Overlap (+6.30):** A strong secondary positive signal. Even if the sequence is imperfect, sharing the same set of unique words strongly increases the probability of a match.
  - **Structural Penalties:** The negative weights for title_length_ratio (-5.74) and year_difference (-0.26) function as penalties. The model uses these to "downvote" candidates that might have similar text but structurally suspicious metadata (e.g., wrong length or wrong era).
- **Reasoning & Interpretation:** The perfect Recall (1.000) and high Precision (0.971) are direct results of the **high linear separability** of the data.
  - **Confidence:** The mean predicted probability for true matches was **0.998**, compared to just **0.001** for non-matches. This indicates the model is not guessing; the "Title Similarity" signal is so strong that the decision boundary is extremely clear.

- **Robustness:** The combination of *Title Similarity* (Sequence) and *Word Overlap* (Set) ensures that the model catches both exact matches and rephrased titles, achieving the 100% recall rate required for the subsequent ranking stage.

4.2 Ranking Evaluation (Mean Reciprocal Rank) [3]

- **Goal:** To quantitatively evaluate the quality of the candidate ranking. In a reference matching system, classification accuracy alone is insufficient; the system must prioritize the correct match at the very top of the list to minimize user effort. MRR measures the average reciprocal rank of the first correct answer in the top-5 predictions.

- **Definition & Implementation:** The Mean Reciprocal Rank is calculated using the standard formula:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

Where:

  - $|Q|$ is the total number of citation queries.
  - rank_i is the position of the correct match (1-indexed). A correct match at Rank 1 yields a score of 1.0; Rank 2 yields 0.5; if not found in the top-5, the score is 0.

- **Quantitative Results:** The model demonstrated exceptional ranking performance across all data splits. As shown in Table 3, the MRR on the unseen Test Set was perfect (1.0), indicating that for every single query, the correct paper was the top-ranked suggestion.

Table 3: Mean Reciprocal Rank (MRR) Results per Partition

Partition	Total Entries	Rank 1 Matches	Overall MRR
Training Set	15,488	15,463 (99.8%)	0.9990
Validation Set	34	34 (100.0%)	**1.0000**
Test Set	26	26 (100.0%)	**1.0000**

- **Interpretation & Reasoning:**

  - **Rank 1 Dominance:** The fact that 99.8% of training examples and 100% of test examples appeared at Rank 1 confirms the discriminative power of the `Title Similarity`

feature. The separation between matches and non-matches is so distinct that the model rarely "hesitates" between candidates.

- **Model Confidence:** The near-perfect MRR aligns with the probability scores observed earlier (0.998 for matches vs 0.001 for non-matches). This implies that the system does not merely "guess" the correct answer but identifies it with high certainty.
- **Implication:** From a user experience perspective, an MRR of 1.0 means the user will almost never need to scroll past the first result, making the tool highly efficient for automated linking.

5 Source structure

Following the official submission guidelines, all materials are placed inside a folder named after the student ID.

```
<student_id>/
|-- src/
|   |-- Milestone2_Hierarchy.ipynb
|   |-- Milestone2_ReferencesMatching.ipynb
|-- requirements.txt
|-- README.md
|-- Report.pdf
```

Each file serves a distinct purpose:

- `Milestone2_Hierarchy.ipynb`: Implements Stage 1, including LaTeX expansion, preprocessing, hierarchy construction, reference extraction, and deduplication.
- `Milestone2_ReferencesMatching.ipynb`: Implements Stage 2, performing feature engineering, supervised learning, reference matching, and ranking.
- `requirements.txt`: Specifies all required Python dependencies for reproducible execution.
- `README.md`: Provides instructions for environment setup and pipeline execution.

- `Report.pdf`: The final written report describing the methodology, data structures, experiments, and results of Milestone 2.

Additional Resources

- **Source Code Repository:** https://github.com/AnhTtis/Data_Science_Project.git
- **Demo Video (Milestone 2):** https://youtu.be/HMOWQzqbxXY

References

[1] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[2] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[3] Ellen M Voorhees et al. The trec-8 question answering track report. In *Trec*, volume 99, pages 77–82, 1999.

[4] Yuanyuan Wang and Kazutoshi Sumiya. A method for generating presentation slides based on expression styles using document structure. *International Journal of Knowledge and Web Intelligence*, 4(1):93–112, 2013.