# Entity Matching Pipeline for Bibliographic Data

Data Integration and Large-Scale Analysis, WS 2025/26

Enis Jusic, Tarik Barucija, Namik Barucija

January 10, 2026

## 1 Entity Matching Pipeline

### 1.1 Data Preparation

#### Cleaning and Normalisation

For each dataset the following preprocessing steps are applied:

- **Title normalisation**: titles are converted to lowercase, Unicode-normalised (NFKD), and stripped of punctuation and extra whitespace.
- **Title tokenisation**: the normalised title is split into a set of whitespace-delimited tokens.
- **Author last names**: the raw `authors` string is split on commas and the word "and". Each part is normalised and the last token is interpreted as the author's last name. The result is a set of last names per record.
- **Year parsing**: the `year` column is converted to integers where possible, otherwise set to `None`.

#### Deduplication Strategy

Within each source we deduplicate potential duplicate records referring to the same publication. The procedure is:

1. Compute a *deduplication block key* as the lexicographically smallest three title tokens ("title prefix").
2. For each block, compare all pairs of records within that block.
3. For a pair $(r_1, r_2)$, compute:
   - Jaccard similarity of title tokens.
   - Whether there is at least one overlapping author last name.
   - Whether the years (if present) differ by at most 1.
4. Two records are considered duplicates if they satisfy either:
   - **Strict rule**: title Jaccard $\geq 0.95$ and at least one shared author, or
   - **Relaxed rule**: title Jaccard $\geq 0.85$, author overlap, and year difference $\leq 1$.
5. Duplicate groups are formed by a simple union operation over these pairwise links; each group is assigned a canonical identifier and only one representative is kept.

Table 1: Deduplication statistics (Task 1.1).

| Dataset | Input records | Duplicate groups | Final unique records |
|---|---:|---:|---:|
| DBLP | 2616 | 28 | 2563 |
| Scholar | 64263 | 761 | 63382 |

Table 1 summarises the deduplication results.

DBLP contains very few duplicates, consistent with its curated nature, whereas the Scholar data has more redundancy. In both cases the number of unique records remains close to the original size, indicating that the deduplication strategy is conservative rather than overly aggressive.

## 1.2 1.2 Blocking

After deduplication we must avoid the $O(n^2)$ cross-product over all pairs of DBLP and Scholar records. We therefore apply two blocking rules and take the union of the resulting candidate pairs.

**Blocking Rule 1: Title Prefix.** We reuse the three-token title prefix and pair any DBLP record with any Scholar record sharing the same prefix.

**Blocking Rule 2: Author Last-Name Overlap.** We build an inverted index from author last names to canonical Scholar records. A DBLP record is paired with a Scholar record if they share at least one last name.

The union of both candidate sets yields the final candidate pool. Table 2 shows the blocking results.

Table 2: Blocking statistics and gold retention (Task 1.2).

| Quantity | Value |
|---|---:|
| Candidates (title prefix) | 8041 |
| Candidates (author overlap) | 374247 |
| Candidates (union) | 379552 |
| Gold pairs total | 4805 |
| Gold pairs retained by blocking | 4626 |
| Gold retention | 96.27% |

Blocking thus reduces the search space from $\approx 1.62 \times 10^8$ to $3.80 \times 10^5$ pairs (about 0.23% of the full cross product) while retaining over 96% of all gold pairs. This is an excellent trade-off and satisfies the requirement that blocking must not lose too many true matches.

## 1.3 Similarity Scoring

For each blocked candidate pair we compute a similarity score based on cosine similarity in a TF–IDF vector space built over all titles.

- Titles are normalised as in Section 1.1 and transformed into TF–IDF vectors using `sklearn`'s `TfidfVectorizer`.
- Separate matrices are built for DBLP and Scholar titles; cosine values are computed via row-wise dot products of the L2-normalised vectors.

A total of 379 552 candidate pairs are scored. The cosine distribution is summarised in Table 3.

Table 3: Cosine similarity statistics over candidate pairs (Task 1.3).

| Quantity | Value |
|---|---|
| Candidate pairs scored | 379552 |
| Pairs with cosine $\geq 0.95$ | 2870 |
| Cosine minimum | 0.000 |
| Cosine mean | 0.022 |
| Cosine maximum | 1.000 |

Most candidate pairs have extremely low cosine similarity, as expected for unrelated publications, while a relatively small subset has cosine above 0.95, which is close to the number of true matches.

In addition to cosine, later stages of the pipeline derive field-specific similarity features (Jaccard, Dice, character-level overlaps) which are used for machine learning in Task 02.

### 1.4 Matching and Evaluation

We convert cosine scores into binary match decisions by applying several thresholds and evaluate the resulting predictions against the gold mapping. Any gold pair not present in the blocked candidate set is counted as a false negative (FN), which makes the evaluation "honest" with respect to blocking.

Table 4 reports precision (P), recall (R), and F1-score for four cosine thresholds, as well as the counts of true positives (TP), false positives (FP), and false negatives (FN).

Table 4: Threshold-based matching results (Task 1.4).

| Threshold | P | R | F1 | TP | FP | FN |
|---|---|---|---|---|---|---|
| 0.70 | 0.8658 | 0.7680 | 0.8139 | 3690 | 572 | 1115 |
| 0.80 | 0.8605 | 0.6483 | 0.7395 | 3115 | 505 | 1690 |
| 0.90 | 0.8478 | 0.5369 | 0.6575 | 2580 | 463 | 2225 |
| 0.95 | 0.8408 | 0.5022 | 0.6288 | 2413 | 457 | 2392 |

The results demonstrate the expected precision–recall trade-off. At high thresholds (e.g. 0.95) precision is slightly higher, but recall suffers significantly because many true matches with noisy or slightly different titles are missed. The best F1-score ($\approx 0.81$) is achieved around threshold 0.70. This motivates the use of a supervised model in Task 02, where multiple signals can be combined instead of relying on a single cosine threshold.

## 2 Feature Vector and ML Model

### 2.1 Create a Training Dataset

We start from the scored candidate pairs in `pairs_scored.csv`, which contains for each blocked pair: `dblp_canon_id`, `scholar_canon_id`, and the cosine similarity.

Each pair is labelled using the ground-truth mapping in canonical ID space:

- **label** = 1 if the pair appears in the gold mapping,
- **label** = 0 otherwise.

The resulting dataset `training_features_labeled.csv` contains 379 552 pairs with 4 626 positive and 374 926 negative examples. Duplicate pairs are removed. This is a strongly imbalanced classification problem with only about 1.2% positive examples.

## 2.2 Feature Engineering

For each candidate pair we compute a feature vector combining title, author, year, and venue information. The most important features are:

- **Title-based features**
  - *title_cos*: cosine similarity of TF–IDF title vectors.
  - *title_jacc*: Jaccard similarity of title token sets.
  - *title_len_diff*: absolute difference in the number of distinct title tokens.
  - *title_shared_tokens*: number of shared title tokens.
  - *title_char_jacc*: Jaccard similarity of character 3-gram sets extracted from normalised titles.
- **Author-based features**
  - *auth_olap*: binary indicator of at least one shared author last name.
  - *auth_jacc*: Jaccard similarity of author last-name sets.
  - *auth_dice*: Dice coefficient over author last-name sets.
- **Year-based features**
  - *year_diff*: absolute difference in publication years (with a sentinel value $-1$ for missing years).
  - *year_ok*: binary indicator that both years are present and differ by at most 1.
  - *year_missing*: binary indicator used later to flag missing year information.
- **Venue-based features**
  - *venue_exact*: binary indicator that normalised venue strings are equal and non-empty.
  - *venue_jacc*: Jaccard similarity of tokenised venue names.
  - *venue_dice*: Dice coefficient over venue tokens.

Together these features capture both high-level signals (exact or near-exact matches) and softer similarities (character-level, token overlaps), which are important for handling noisy real-world data.

## 2.3 Preprocessing Training Data

### Missing Values

Missing years are encoded with the sentinel value $-1$ and an additional binary indicator *year_missing*. During preprocessing, numeric features (including *year_diff*) are median-imputed to avoid losing instances.

### Feature Scaling

Features are separated into binary and continuous subsets. A `ColumnTransformer` applies a `StandardScaler` to the continuous features while passing binary features through unchanged. This is particularly important for SVMs, which are sensitive to feature scales.

**Imbalance Handling**

Several strategies are explored:

- **Class weights**: Logistic Regression and Random Forests are trained with class weights (`class_weight="balanced"` or `"balanced_subsample"`) so that minority-class errors are penalised more heavily.
- **SMOTE**: an additional Logistic Regression model is trained within an `ImbPipeline` that includes SMOTE oversampling of the minority class.
- **Undersampling**: an intermediate variant undersamples negatives to achieve an approximate 1:5 ratio (4 626 positives, 23 130 negatives), mainly for exploratory analysis.

The final reported models (Random Forest and SVM) use class weights and the full dataset; their evaluation is based on stratified three-fold cross validation.

## 2.4 Model Training

Two final classifiers are trained and evaluated:

**Random Forest (RF)** A `RandomForestClassifier` with 300 trees, class-weighted subsampling, and the standard Gini impurity criterion. The RF can naturally handle non-linear feature interactions and is relatively robust to feature scaling, although scaling does not hurt performance.

**SVM (RBF)** An SVM with RBF kernel, $C = 1.0$, and `gamma="scale"`, using class weights to address imbalance. The SVM benefits strongly from the scaling of continuous features.

Both models are wrapped in a `Pipeline` with the shared preprocessing step described above. Evaluation uses stratified three-fold cross validation with precision, recall and F1 as metrics.

## 2.5 Evaluation

Table 5 summarises the cross-validated performance of both models. Each value is the mean over three folds, with standard deviation in parentheses.

Table 5: 3-fold cross-validated performance of ML models (Task 2.4/2.5).

| Model | Precision | Recall | F1-score |
|---|---|---|---|
| Random Forest | $0.9131 \pm 0.0084$ | $0.9436 \pm 0.0037$ | $0.9280 \pm 0.0038$ |
| SVM (RBF) | $0.7764 \pm 0.0129$ | $0.9957 \pm 0.0008$ | $0.8724 \pm 0.0081$ |

The assignment specifies baselines of F1 $\approx 0.76$ for the SVM and $\approx 0.79$ for the Random Forest. Both of our models exceed these numbers by a large margin. The Random Forest achieves an F1-score of approximately 0.93, while the SVM achieves about 0.87.

The two models exhibit complementary behaviour:

- **Random Forest**: Balances precision and recall very well. With precision above 0.91 and recall around 0.94, it yields the best overall F1-score. This suggests that the tree ensemble can effectively exploit non-linear combinations of title, author, year and venue features.
- **SVM**: Achieves extremely high recall ($> 0.99$), indicating that it almost never misses a true match, but its precision is lower ($\approx 0.78$) due to more false positives. This model may be preferable in scenarios where missing a match is much more costly than reviewing extra candidate pairs.

**Feature Importance (Qualitative).** While Random Forest feature importances are not printed in the notebook, we can qualitatively reason about the most influential features:

- High-quality title similarities (*token Jaccard, character-level Jaccard*) are crucial for distinguishing matches from non-matches.
- Author overlap features (*auth_olap, auth_jacc, auth_dice*) help disambiguate records with similar titles.
- Year and venue features (*year_ok, year_diff, venue_exact, venue_jacc*) capture complementary evidence and help reduce false positives where titles and authors alone are ambiguous.

A small implementation issue is that the feature `title_cos` is currently read from a column named `cosine_tfidf`, whereas the scored pairs file uses the column name `cosine`. As a result, `title_cos` is effectively zero for all examples. Fixing this would likely further improve model performance by explicitly incorporating the TF–IDF similarity.

## 3 Reporting and Reproducibility

The implementation is delivered as a Jupyter notebook `final1.ipynb` together with:

- a `requirements.txt` file specifying exact package versions (including `numpy`, `pandas`, `scikit-learn`, `imbalanced-learn`, and `ipykernel`),
- all generated code for data loading, preprocessing, blocking, similarity computation and ML training,
- a `README.md` explaining how to recreate the Conda environment and run the pipeline end-to-end.

The pipeline is fully reproducible: starting from the raw CSV files and the perfect mapping, it outputs the same blocking statistics, similarity scores, labelled training data, and cross-validated evaluation metrics reported in this document.

## 4 Conclusion

We implemented and evaluated a complete entity matching pipeline for bibliographic data. The rule-based part of the pipeline achieves competitive performance using TF–IDF cosine similarity alone, while the machine-learning models substantially improve F1-score by leveraging a richer set of similarity features and appropriate handling of class imbalance.

The Random Forest model in particular achieves high precision and recall, comfortably surpassing the target baselines. The SVM model offers an alternative operating point with almost perfect recall. The overall design illustrates a practical and scalable approach to entity matching in large bibliographic datasets.