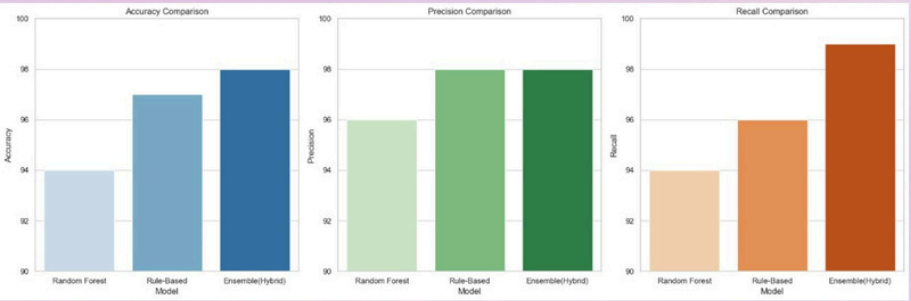
Entity Matching by Pie

Our Approaches



1

Random Forest

- Accuracy: 94.0%
- Precision: 96.0%
- Recall: 94.0%

2

Rule Based

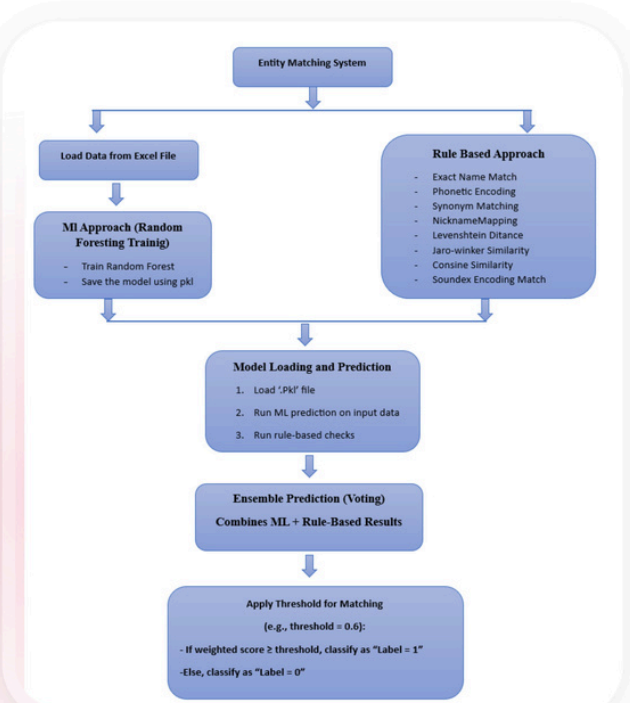
- Accuracy: 97.0%
- Precision: 98.0%
- Recall: 96.0%

3

Ensemble (Hybrid)

- Accuracy: 98.0%
- Precision: 98.0%
- Recall: 99.0%

Architecture



Data Input

Load data from an Excel file.

ML Model Training

Train a Random Forest model on the data.

Rule-Based Approach

Apply traditional rules like exact match, phonetic encoding, etc.

Model Saving

Save the trained Random Forest model for future use.

Model Loading and Prediction

Load the saved model and make predictions on new data using both ML and rule-based methods.

Ensemble Prediction

Combine the predictions from ML and rule-based approaches.

Threshold-Based Classification

Classify entities as matches or non-matches based on a threshold.

Output

Output the final classification results.

Core Functions

Click here to edit subtitle

Normalization

Phonetic Encoding

Soundex Encoding

Levenshtein Similarity

Jaro-Winkler Similarity

Description of a primary heading

Cosine Similarity

Description of a primary heading

Normalization 🛠

Working Process

Step 1: Check if the name is missing or NaN. If missing, return an empty string.

Step 2: Convert the name to a string, lowercase it, and strip leading/trailing spaces.

Step 3: Remove common titles (e.g., "Dr.", "Mr.", "Ms.") from the name.

Step 4: Remove special characters (e.g., periods, hyphens, commas).

1. **Purpose**
   Standardize names by removing spaces, special characters, and titles (like Dr., Mr.) and removes inconsistencies.

2. **Titles & Honorifics**
   (e.g., "Dr. John Doe" → "john doe")

3. **Missing Spaces**
   (e.g., "Mary-Ellen" → "maryellen")

4. **Titles & Honorifics**
   (e.g., "Dr. John Doe" → "john doe")

Phonetic encoding

Working Process (Steps in Sequence)

Step 1: Encoding the 2 names:

-> Pass the **first name** and **second name** to the **Double Metaphone** function individually.

-> The function generates two phonetic encodings:

Primary Encoding: The most common phonetic representation.

Secondary Encoding: An alternative representation for different pronunciations.

Step 2: Compare Encodings:

-> **Compare the Primary Encodings**:If the **Primary Encodings** of both names are the same, the names are considered phonetically similar.

-> **Compare the Secondary Encodings**:If the **Secondary Encodings** of both names are the same, the names are considered phonetically similar, providing an alternative match.

1. **Purpose**
   Encodes names phonetically to account for pronunciation differences.

2. **Technique**
   doublemetaphone

3. **Nicknames**
   (e.g., "Mike" vs. "Michael“)

4. **Transliteration Spelling Differences**
   (e.g., "Abdul Rasheed" vs. "Abd al-Rashid“)

5. **Phonetic Similarity**
   (e.g., "Smith" vs. "Smyth")

Working Process (Steps in Sequence)

Step 1 : Encoding First Name:

->Pass the **first name** to the **Soundex** function.

->The function generates a **Soundex code** consisting of:

First Character: The first letter of the name.

Next 3 Characters: The numerical codes of the next consonants based on similar-sounding groups.

Remove Vowels: Vowels are ignored unless they are the first letter.

Pad with Zeros: If the code has fewer than four characters, pad with zeros.

Step 2 : Encoding Second Name:

->Pass the **second name** to the **Soundex** function.

->The function generates a **Soundex code** with the same process as the first name.

Step 3: Compare Soundex Codes:

->**Compare the Soundex codes** of the two names.

->If the **Soundex codes** are the same, the names are considered phonetically similar.

Soundex Encoding

Click here to edit subtitle

1. **Purpose**
   Converts names to a 4-character based on their sounds
2. **Phonetic Similarity**
   (e.g., "Anne" vs. "Ann")
3. **Nicknames**
   (e.g., "Bill" vs. "William")
4. **Spelling Differences**
   (e.g., "Smith" vs. "Smyth")
5. **Transliteration Spelling Differences**
   (e.g., "Abdul Rasheed" vs. "Abd al-Rashid")
6. **Truncated Components**
   (e.g., "Blankenship" vs. "Blankensh")

Soundex Code	Letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R
(ignored)	A, E, H, I, O, U, W, Y

Soundex Code Patterns

1

Case 1 : Input length ≤ 4 (Exact match to Soundex length)

Input: "Anne"

Soundex: A 5 5 0

Final Code: A550

2

Case 2: Input length > 4 characters (Truncation + Remove duplicates)

Input: "Brittany" (8 characters)

Soundex: B 6 0 3 0 5 0

Final Code: B603 (only the first 4 digits)

3

Case 3: Input with fewer than 4 characters (Padding with zeroes)

Input: "Ian" (3 characters)

Soundex: I 0 5

Final Code: Pad with 0 to make it 4 characters: I05 → I050

Levenshtein Similarity

Working Process

Levenshtein Distance: Measures the minimum number of edits (insertions, deletions, substitutions) required to change one string into the other.

fuzz.ratio(): Uses the Levenshtein distance to calculate a similarity score between 0 and 100.

Normalize: Divide the similarity score by 100 to scale it between 0.0 and 1.0.

1. **Purpose**
   Measures the minimum number of edits required to transform one string into another

2. **Spelling Differences**
   (e.g., "Jon" vs. "John")

3. **Truncated Components**
   (e.g., "Alexandr" vs. "Alexander")

Jaro-Winkler Similarity 🔗

Working Process (Steps in Sequence)

Step 1: Tokenize strings into words (tokens).

Step 2: Sort tokens alphabetically for both strings.

Step 3: Calculate Jaro Similarity:

- Find common characters and matches.
- Calculate the similarity score.

Step 4: Apply Jaro-Winkler adjustment to reward common prefixes.

1. **Purpose**
   Measures similarity between two strings, giving more weight to common prefixes.
2. **Phonetic Similarity**
   (e.g., "Smith" vs. "Smyth")
3. **Spelling Differences**
   (e.g., "Jon" vs "John")
4. **Truncated Components**
   (e.g., "Joseph" vs "Jos")
5. **Out-of-Order Components**
   (e.g., "Diaz, Carlos Alfonzo" vs "Carlos Alfonzo Diaz")

Cosine Similarity

Working Process

Step 1-CountVectorizer: Converts names into character n-grams (sequences of 2 to 3 characters).

Step 2-Create N-grams: Extract bi-grams and tri-grams from each name.

Step 3-Frequency Matrix: Construct a matrix where each row represents a name, and each column represents an n-gram's frequency.

Step 4-Vector Representation: Convert each name into a vector based on its n-gram frequencies.

Step 5-Cosine Similarity: Calculate the similarity between the vectors using the cosine similarity formula, which evaluates the angle between the vectors.

Cosine Similarity=Magnitude of Vectors/Dot Product of Vectors

1. **Purpose**
   Measures similarity between vectors of n-grams.

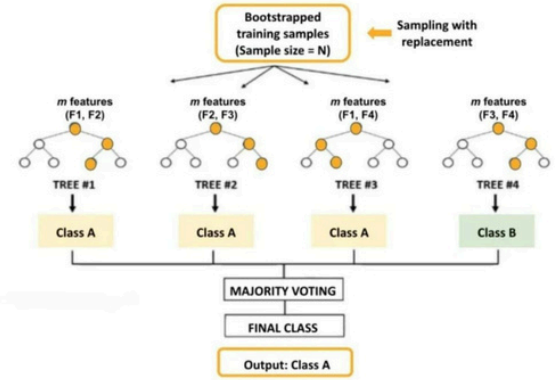
2. **Split Database Fields**
   (e.g., "Rip Van Winkle" vs "Rip. Van Winkle")

3. **Out-of-Order Components**
   (e.g., ("Carlos Alfonzo Diaz" vs "Diaz, Carlos Alfonzo")

Why Random Forest and How it Works?

Training Data
(Sample size, N= 6, No. of features, F = 4)

F1	F2	F3	F4	Y
2.1	0	400	-9	A
3.0	1	890	-42	B
2.2	1	929	0	B
4.0	0	324	-23	A
3.5	1	333	-15	A
6.0	0	215	-9	A



Handles Missing Data & Outliers

Description of a primary heading

Reduces overfitting

Description of a primary heading

Identifies Hidden Patterns

Description of a primary heading

Why use PKL files?

Efficient Model Management with pkl Files

Optimizing AI Model Deployment for Enhanced Performance

1 Faster Execution

Using pkl files allows for quick reuse of trained models, eliminating retraining time.

2 Model Training

The model is initially trained on historical data to ensure high accuracy and relevance.

3 Model Saving

After training, the model is saved using joblib or pickle, preserving its state for future use.

4 Loading for Inference

The saved model can be quickly loaded for inference, ensuring efficient data processing.

5 Performance Enhancement

This model management strategy significantly boosts the app's performance by ensuring immediate model availability.

Rule-Based Approach: Advantages

Ease of Debugging

Errors can be identified and corrected quickly due to the straightforward nature of rule-based systems.

Deterministic Results

Outputs are consistent with the same inputs, ensuring predictable behavior for users.

High Explainability

Decisions are transparent and interpretable, allowing for justification of outcomes.

Efficiency

Generally faster than machine learning models due to the absence of complex computations.

Customizability

Rules can be easily modified or expanded to accommodate new scenarios or requirements.

Why Use an Ensemble Approach?

The ensemble approach combines both rule-based and ML models to make the final prediction, ensuring that both types of logic contribute to the decision-making process: This approach leverages the strengths of both methods. The rule-based system catches simple matches quickly, while the ML model handles complex patterns.

1

Combining Strengths

Balances the simplicity of rules with the adaptability of machine learning.

2

Improved Robustness

Ensures accuracy by reducing missed matches through complementary methods.

3

Rule-Based Prediction

Calculates the weighted similarity score between names using various similarity metrics.

4

ML Prediction

Generates a feature set representing the similarity between names.

5

Final Decision (OR Logic)

If either method predicts a match, the final prediction will be a match.

Rule-based Prediction	ML Prediction	Final Decision (OR Logic)
1 (Match)	1 (Match)	1 (Match)
1 (Match)	0 (No Match)	1 (Match)
0 (No Match)	1 (Match)	1 (Match)
0 (No Match)	0 (No Match)	0 (No Match)

Table of Name Variations and Matching Techniques

Variation	Covered By Function	Example of variation	Explanation
Phonetic Similarity	Phonetic Encoding, Soundex Encoding	"Jesus" <-> "Heyzeus" <-> "Haezoos"	Matches names based on how they sound, accounting for pronunciation differences.
Missing Spaces & Hypens	Normalizing, Cosine Similarity	"MaryEllen" <-> "Mary Ellen" <-> "Mary-Ellen"	Handles names with incorrect spacing or hyphenation.
Spelling Differences	Phonetic Encoding, Levenshtein Similarity	"Abdul Rasheed" <-> "Abd al-Rashid"	Matches names with minor spelling or transliteration differences.
Titles & Honorifics	Normalizing	"Dr. John Doe" <-> "John Doe"	Removes titles like "Dr.", "Mr.", or "Ph.D." to standardize names.
Missing Components	Levenshtein Similarity	"Phillip Charles Carr" <-> "Phillip Carr"	Matches names where some components are missing.

Variation	Covered By Function	Example of variation	Explanation
Out-of-Order Components	Cosine Similarity, Jaro-Winkler Similarity	"Diaz, Carlos Alfonzo" <-> "Carlos Alfonzo Diaz"	Accounts for changes in the order of name components.
Nicknames	Phonetic Encoding, Soundex Encoding	"William" <-> "Will" <-> "Bill"	Matches common nicknames or shorter versions of names.
Truncated Components	Levenshtein Similarity, Soundex Encoding	"Blankenship" <-> "Blankensh"	Matches names that are shortened or incomplete.
Initials	Levenshtein Similarity, Cosine Similarity	"J. E. Smith" <-> "James Earl Smith"	Matches names with initials instead of full forms.
Split Database Fields	Cosine Similarity	"Rip. Van Winkle" <-> "Rip Van. Winkle"	Handles cases where names are split or joined differently in databases.
Transliteration Spelling Differences	Phonetic Encoding, Soundex Encoding,	"Abdul Rasheed" <-> "Abd al-Rashid"	Matches variations in spelling due to transliteration differences.