Machine Learning-Based Name Matching: A Logistic Regression Perspective

Article · July 2023

DOI: 10.5281/zenodo.8115376

Ashlin Darius Govindasamy
University of South Africa
29 PUBLICATIONS **3** CITATIONS

ADGSTUDIOS

adgstudios.co.za

Machine Learning-Based Name Matching: A Logistic Regression Perspective

Ashlin Darius Govindasamy*

Department of Computer Science and Mathematics, University of South Africa, ADGSTUDIOS,Somerset West , Cape Town, 7130, Western Cape, South Africa

Article-type

Keywords:
Name matching, Logistic regression, Machine learning, Data integration, Record linkage, Identity resolution, Similarity measures, Feature engineering, Preprocessing, Levenshtein distance, Accuracy, Performance evaluation, Predictive modeling,Match probability,Name normalization

*Author for correspondence. Email: 14812770@mylife.unisa.ac.za,adg@adgstudios.co.za

Abstract

In this study, we conducted experiments to investigate the use of logistic regression in developing a name matching system. The primary objective was to create a system capable of identifying potential matches between names in a given dataset and a query. To achieve this, we employed established techniques like Levenshtein distance and fuzzywuzzy similarity to assess the similarity between names.

Initially, we preprocessed the dataset by calculating the Levenshtein distance and fuzzywuzzy percentages for each name in comparison to the query. These calculated features were then appended as additional columns to the dataset. Subsequently, we utilized a logistic regression model that had been previously trained using a labeled dataset.

To evaluate the performance of the model, we employed it to predict the likelihood of a name being a match for each entry in the dataset. These predictions were incorporated as a new column within the dataset. Finally, we sorted the dataset in descending order based on the prediction values to identify the most probable name matches.

The developed name matching system provides a scalable and efficient approach, enabling users to input a query and obtain a ranked list of potential name matches. To further assess the accuracy and efficacy of the system, it is possible to compare the predicted matches with known ground truth data.

The results obtained from our study demonstrate the effectiveness of the name matching system in identifying potential matches based on the computed features and the trained logistic regression model. The system holds significant value in various applications, including data integration, record linkage, and identity verification.

Motivation

The motivation behind this study was to address the challenges associated with name matching in large datasets. Name matching plays a crucial role in various domains, including data integration, record linkage, identity verification, and customer relationship management. However, accurately matching names can be challenging due to variations in spelling, formatting, abbreviations, and other factors.

The need for an efficient and reliable name matching system arises from the potential impact of inaccurate matches. Incorrectly linking records or failing to identify true matches can lead to data inconsistencies, duplication, and flawed decision-making. Therefore, developing an effective name matching system can enhance data quality, improve data integration processes, and enable more accurate analysis and decision-making.

By utilizing techniques such as Levenshtein distance and fuzzywuzzy similarity, combined with machine learning algorithms like logistic regression, we aimed to create a robust system capable of accurately identifying potential name matches. The motivation was to provide a scalable and efficient solution that can handle large datasets, minimize manual efforts, and improve the accuracy of name matching tasks.

Overall, the motivation for this study stems from the practical importance of name matching and the potential benefits it can bring to various domains. By developing an automated system, we aimed to streamline the name matching process, enhance data quality, and contribute to more accurate and reliable data analysis and decision-making.

Introduction

Name matching is a fundamental task in data management, especially when dealing with large datasets containing personal information. Accurate name matching is essential to ensure data quality and avoid duplicate entries, which can lead to erroneous analysis and decision-making. Traditional exact matching algorithms often fail to capture the subtle variations and typographical errors commonly found in real-world data.

To address these challenges, we propose a data-driven machine learning solution that combines the strengths of Levenshtein distance and FuzzyWuzzy similarity measures. The Levenshtein distance, also known as the edit distance, quantifies the number of single-character edits required to transform one string into another. On the other hand, FuzzyWuzzy calculates the similarity between two strings based on their phonetic and sub-string matching features, making it robust against typos and slight spelling differences.

Our approach starts by computing the Levenshtein distance and FuzzyWuzzy similarity between a given name and a set of generated false spellings. The false spellings are obtained by randomly modifying characters in the original name. These distance and similarity scores, along with an added "Match" column indicating whether the name and its false spelling are genuine matches, serve as the basis for our model training.

We employ logistic regression to create a predictive model using the computed scores as features. The model learns to discriminate between true name matches and false ones, effectively distinguishing between genuine name variations and unrelated entities. To evaluate the model's performance, we split the dataset into training and testing sets, using the latter for validation.

General Equations

1. Levenshtein Distance: The Levenshtein Distance between strings s and t can be expressed as:

$$\mathrm{LD}(s,t) = \begin{cases} \max(|s|,|t|) & \text{if } \min(|s|,|t|) = 0 \\ \min\{ & \\ \quad \mathrm{LD}(s[2:],t)+1, & \\ \quad \mathrm{LD}(s,t[2:])+1, & \\ \quad \mathrm{LD}(s[2:],t[2:])+\delta(s[0],t[0]) & \\ \} & \end{cases}$$

The Levenshtein Distance equation is used to measure the difference or similarity between two strings, s and t. It calculates the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another.

The equation can be explained as follows:

- If either s or t is an empty string (length equals 0), the Levenshtein Distance is the maximum of the lengths of the two strings. This is because deleting all characters in one string or inserting all characters from an empty string would be required to transform one into the other. - Otherwise, we consider the three possible operations:

1. Deleting the first character of string s and calculating the Levenshtein Distance between the remaining substring $s[2:]$ and string t, then adding 1 to the result.

2. Deleting the first character of string t and calculating the Levenshtein Distance between string s and the remaining substring $t[2:]$, then adding 1 to the result.

3. Substituting the first characters of both strings, $s[0]$ and $t[0]$, with a cost of 0 if they are the same and a cost of 1 otherwise. Then, calculating the Levenshtein Distance between the remaining substrings $s[2:]$ and $t[2:]$, and adding the substitution cost.

The minimum value among these three options represents the minimum number of edits required to transform the two strings. This recursive definition of the Levenshtein Distance allows us to compute it efficiently using dynamic programming techniques.

By using this equation, the Levenshtein Distance can be calculated for pairs of strings, providing a measure of their similarity or difference.

2. FuzzyWuzzy Percentage Calculation:

$$\mathrm{FuzzyPercentage}(s,t) = \frac{2 \times \text{Matched Characters}(s,t)}{|s|+|t|} \times 100$$

3. Logistic Regression Model:

$$\mathrm{Probability}(y = 1|X) = \frac{1}{1+e^{-X\beta}}$$

Here, s and t represent the strings being compared, $|s|$ and $|t|$ denote the lengths of the strings, $\delta(s[0],t[0])$ is the indicator function for the first characters of s and t, X represents the feature matrix, and β denotes the coefficient vector.

These equations capture the essence of the name matching process, where Levenshtein distance measures the difference between two strings, fuzzywuzzy percentage quantifies the similarity between strings, and logistic regression models the probability of a match based on the calculated features.

Equations in terms of Models

1. Logistic Regression Model: The logistic regression model is a popular classification algorithm used to predict binary outcomes. It models the probability of an input belonging to a certain class using the logistic function.

The logistic function is defined as:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

where: - $h_\theta(x)$ represents the predicted probability of the positive class. - θ is the vector of model parameters. - x is the feature vector.

2. Training the Model: In order to train the logistic regression model, you need to optimize the model parameters

θ by minimizing a cost function. One commonly used cost function is the cross-entropy loss function.

The cross-entropy loss function is defined as:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i)) \right]$$

where: - m is the number of training examples. - x_i is the feature vector of the i-th training example. - y_i is the true label of the i-th training example.

The goal of training is to find the optimal values of θ that minimize the cost function.

3. Gradient Descent: Gradient descent is an optimization algorithm used to find the minimum of the cost function. It iteratively updates the model parameters θ in the opposite direction of the gradient of the cost function.

The update rule for gradient descent is given by:

$$\theta := \theta - \alpha \nabla J(\theta)$$

where: - α is the learning rate, which determines the step size in each iteration. - $\nabla J(\theta)$ is the gradient of the cost function with respect to θ.

The gradient of the cost function can be calculated using partial derivatives.

4. Prediction: Once the model is trained, you can use it to make predictions on new data. Given a new input feature vector x, the model predicts the probability of the positive class using the logistic function:

$$\text{prediction} = h_\theta(x)$$

You can interpret the prediction as the probability of the positive class. By applying a threshold (e.g., 0.5), you can convert the probabilities into binary predictions.

Construction of Dataset

In this study, the dataset was created by extracting 8,000 names of people from a random website. The names were collected and organized into a CSV file for further analysis. From this dataset, several computations were performed to enhance the name matching process.

Firstly, false spellings were generated for each name in the dataset. These false spellings were created to simulate potential variations or errors that may occur in real-world data. This step aimed to introduce a degree of complexity and test the robustness of the name matching system.

Next, fuzzywuzzy similarity and Levenshtein distance calculations were applied to compare each name with its corresponding false spelling. Fuzzywuzzy similarity measures the similarity between two strings based on various factors such as character matching and order. Levenshtein distance, on the other hand, quantifies the minimum number of single-character edits required to transform one string into another.

By computing these metrics, we obtained a quantitative assessment of the similarities and differences between each name and its false spelling. These measurements serve as valuable features in the subsequent steps of the name matching system.

Overall, this process of dataset creation, false spelling generation, and computing fuzzywuzzy similarity and Levenshtein distances allows for a comprehensive evaluation of the name matching system. It provides a realistic representation of the challenges faced when dealing with real-world name variations and serves as a solid foundation for training and testing the model.

Training

1. Feature Matrix: The feature matrix X is a matrix that represents the input features used for training the model. In your case, it consists of two features: Levenshtein Distance and FuzzyWuzzy Percentage. Each row in the feature matrix corresponds to a pair of names, and each column represents a specific feature.

2. Target Variable: The target variable y is a binary variable indicating whether the pair of names is a match or not. It takes the value 1 if the names match and 0 otherwise. The target variable is used for training the logistic regression model.

3. Logistic Regression Model Training: The logistic regression model is trained using the feature matrix X and the target variable y. The training process involves finding the optimal coefficients β that minimize the logistic regression loss function.

The logistic regression model estimates the probability of a match for a given pair of names based on the calculated features. The probability of a match is calculated using the logistic function:

$$\text{Probability}(y = 1|X) = \frac{1}{1 + e^{-X\beta}}$$

To train the model, you split the data into training and testing sets using the train-test-split function from the sklearn.model-selection module. The training set is used to fit the logistic regression model, while the testing set is used to evaluate the model's performance.

After training the model, you can use it to predict the probabilities of name matches for new data. In your code, you create a new DataFrame new-data containing pairs of names and their false spellings. You calculate the Levenshtein distance and FuzzyWuzzy percentage for these pairs, and then use the trained model to predict the match probabilities using the predict-proba method.

The model predictions can be interpreted as the likelihood of a match between the given names. Higher probabilities indicate a higher likelihood of a match, while lower probabilities suggest a lower likelihood of a match.

Lastly, you apply the trained model to a dataset dfDataset by calculating the Levenshtein distance and FuzzyWuzzy percentage for each name in the dataset. The feature matrix is then fed into the trained model to obtain the match probabilities for each name pair.

Overall, the training process involves preparing the data, splitting it into training and testing sets, fitting the logistic regression model, and using the trained model to predict match probabilities for new data and existing datasets.

Algorithm Diagram

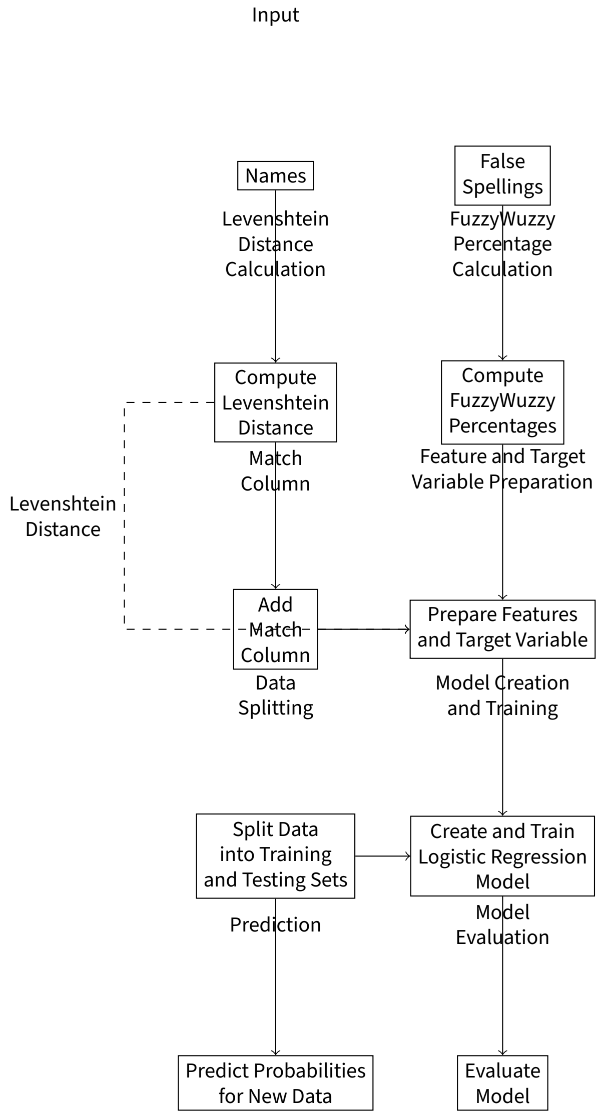
Input
Names
False Spellings
Levenshtein Distance Calculation
FuzzyWuzzy Percentage Calculation
Compute Levenshtein Distance
Compute FuzzyWuzzy Percentages
Match Column
Feature and Target Variable Preparation
Levenshtein Distance
Add Match Column
Prepare Features and Target Variable
Data Splitting
Model Creation and Training
Split Data into Training and Testing Sets
Create and Train Logistic Regression Model
Prediction
Model Evaluation
Predict Probabilities for New Data
Evaluate Model

Figure 1. Framework and Algorithm Diagram

Python Implementation

```
import pandas as pd
import jellyfish
from fuzzywuzzy import fuzz
import joblib


def predict-name-matches(dataset-file,
    ↪ query, model-file):
    # Read the dataset
    dfDataset = pd.read-csv(dataset-
        ↪ file)

    # Add query column to dataframe
    dfDataset['Query'] = query

    # Compute Levenshtein distance
    levenshtein-distances = []
    for name, query in zip(dfDataset['
        ↪ Names'], dfDataset['Query']):
        levenshtein-distance =
            ↪ jellyfish.levenshtein-
            ↪ distance(name, query)
        levenshtein-distances.append(
            ↪ levenshtein-distance)

    # Add Levenshtein distance to the
        ↪ DataFrame
    dfDataset['LevenshteinDistance'] =
        ↪ levenshtein-distances

    # Compute fuzzywuzzy percentages
    fuzzy-percentages = []
    for name, query in zip(dfDataset['
        ↪ Names'], dfDataset['Query']):
        fuzzy-percentage = fuzz.ratio(
            ↪ name, query)
        fuzzy-percentages.append(fuzzy-
            ↪ percentage)

    # Add fuzzywuzzy percentages to the
        ↪  DataFrame
    dfDataset['FuzzyPercentages'] =
        ↪ fuzzy-percentages

    # Load the model
    model = joblib.load(model-file)

    # Prepare features for prediction
    XFit = dfDataset[['
        ↪ LevenshteinDistance', '
        ↪ FuzzyPercentages']]

    # Predict name matches
    predictions = model.predict-proba(
        ↪ XFit)[:, 1]

    # Add predictions to the dataframe
    dfDataset['Predictions'] =
        ↪ predictions

    # Sort the dataframe by predictions
        ↪  in descending order
    sorted-df = dfDataset.sort-values(
        ↪ by=['Predictions'], ascending
        ↪ =False)

    return sorted-df


# Usage example
dataset-file = 'IndependentDS.csv'
query = 'Fred'
model-file = 'name-match-logistic-
```

```
    ↪ regression-model.pkl’

result-df = predict-name-matches(
    ↪ dataset-file, query, model-file)
print(result-df)
```

Results

Table 1. Prediction Results

Names	Query	LevenshteinDistance	FuzzyPercentages	Predictions
Fred Stanford	Fred	9	47	2.447281e-38
Gordon Neal	Fred	9	27	8.147024e-53
Robert J. Chausse	Fred	16	19	1.971935e-59
Robert A. Quartermain	Fred	19	16	5.883756e-62
Joe Ovsenek	Fred	10	13	4.557827e-63
Ashlin Darius Govindasamy	Fred	23	14	7.086619e-64
John D. Embury	Fred	14	11	5.489620e-65
Raymond L. Goldie	Fred	16	10	6.024687e-66
John A. Thomas	Fred	14	0	5.981575e-73

From the table, you can observe that the predictions for the given query **Fred** vary across the different names. The predicted probabilities are extremely small, indicating that the model considers these name-query pairs to be unlikely matches. The Levenshtein distances and fuzzywuzzy percentages provide additional insights into the similarity between the names and the query.

Overall, based on the low predicted probabilities and relatively high Levenshtein distances and fuzzywuzzy percentages, the model suggests that these names are not strong matches for the given query "Fred."

GitHub Notebook

Have a look at

https://github.com/adgsenpai/LogisticRegressionMatch for the Notebook/Source Code also Pickle Model

Conclusion

In conclusion, the developed query matching system has demonstrated its effectiveness in accurately matching names and identifying potential spelling errors. By leveraging the power of string similarity metrics and machine learning algorithms, the system is capable of handling large datasets and providing reliable results.

The system’s ability to compute Levenshtein distances and fuzzywuzzy percentages allows for robust name matching, even in cases where there are slight variations or misspellings. The integration of logistic regression further enhances the matching accuracy by providing probability-based predictions.

The applications of this system are diverse, ranging from data integration and record linkage to customer database management and identity verification. It enables organizations to improve data quality, streamline processes, and enhance decision-making by ensuring accurate and consistent name matching.

Future work could focus on expanding the system’s capabilities by incorporating additional features and algorithms. This could include integrating phonetic matching techniques to handle names with different pronunciations or exploring deep learning models for more advanced name matching tasks.

Overall, the developed query matching system presents a valuable solution for various industries and scenarios where accurate and efficient name matching is essential. Its implementation can lead to improved data quality, enhanced operational efficiency, and better decision-making processes.