**Person name matching**

**ENTITY MATCHING**

**Author: Hridaya Saboo**

Contents

[Problem Statement 3](#_Toc182015411)

[Datasets 3](#_Toc182015412)

[1. Matching names dataset 3](#_Toc182015413)

[2. Non-matching (mismatch) names dataset 4](#_Toc182015414)

[3. Baseline data 4](#_Toc182015415)

[Train-Val-Test Split 4](#_Toc182015416)

[Preprocessing 5](#_Toc182015417)

[Feature Generation 5](#_Toc182015418)

[Feature Selection 6](#_Toc182015419)

[1. Univariate Analysis 6](#_Toc182015420)

[2. Bivariate Analysis 7](#_Toc182015421)

[Modelling 7](#_Toc182015422)

[Threshold tuning 9](#_Toc182015423)

[Results Analysis 10](#_Toc182015424)

[1. Validation data 10](#_Toc182015425)

[2. Baseline (Test) data 11](#_Toc182015426)

[Control Mechanisms 11](#_Toc182015427)

[Model Usage 12](#_Toc182015428)

[Conclusion 12](#_Toc182015429)

[References 12](#_Toc182015430)

# Problem Statement

The objective of Entity Matching is to provide an automated solution to match different variations of name of an individual or a company. This has various use cases for the bank. For eg. In regulatory compliance banks are prohibited to perform transactions for clients that are sanctioned by various international organizations like OFAC, EU etc for their involvement in illegal activities like drug trafficking, money laundering, terrorism financing etc. To identify these clients banks perform referential screening on its clients to identify if their names are part of sanctioned lists issued by these international organizations. To achieve this task banks use conventional, rule-based name matching softwares like Fircosoft API used by Socgen. The problem with these softwares is that these are highly inefficient and generate a lot of false name matches with very low transformation rate.

As these softwares have licenses and regulatory approvals we cannot entirely replace these systems. An alternative approach towards making this system more efficient is by adopting a solution on the top of these systems which can automatically filter out false positive matches generated by these systems.

In this study, we design a name matching solution that can be applied on the output of these conventional systems by evaluating these name matches and filtering out highly false positive cases with zero risk tolerance for missing true matches.

Emphasizing once again, the primary objective of this solution is not to exactly identify if two names are same, but it is to minimize the false positives with zero risk appetite for missing true matches.

# Datasets

## Matching names dataset

Before creating different variations of the same name, we first brainstormed on what could we the sources of variations. The variations can happen because of several reasons that can be attributed to human errors like typing errors, pronunciation error, phonetic errors etc. and errors due to limitations of the machines in terms of the size of the input fields, name parsing, optical recognition errors due to different styles of handwriting or the quality of images etc. These errors (more like variations) could also be generated due to different ways of writing same name in different languages i.e. during transliterations as well as in the same language i.e. nicknames or abbreviations. Also sometimes honorific titles can also add to these name variations.

After identifying all these variations, we either manually generated these variations or sourced datasets from different places which provides us those variations.

Following is the file that summarizes all the variations,



We generated a very conservative names dataset with multiples levels of variations Level1 to Level 5, with increasing complexity of errors respectively. This will help us build a more robust algorithm to identify true matches.

To generate spelling errors, we use the following error length distribution. To prevent true matches strings from being completely dissimilar we also rectify the error length to 30% of the length of a name. Also, as we expect in any real use case, we generated very few names with very large spelling errors.

|  |  |
| --- | --- |
| **ERROR LENGTH** | **PROBABILITY** |
| 1 | 20% |
| 2 | 12% |
| 3 | 6% |
| 4 | 15% |
| 5 | 15% |
| 6 | 6% |
| 7 | 6% |
| 8 | 6% |
| 9 | 6% |
| 10 | 6% |
| 11 | 1% |
| 12 | 1% |

To generate the keyboard typing error we used the following keyboard layout and used Euclidean distance between the positions of the letters as weight to replace with adjacent characters.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | q | w | e | r | t | y | u | i | o | p |
| 1 |  | a | s | d | f | g | h | j | k | l |
| 2 |  |  | z | x | c | v | b | n | m |  |

## Non-matching (mismatch) names dataset

To generate these mismatches we randomly shuffle and equate the names or parts of names against each other. Like matching names dataset, we created non-matching names dataset with multiple levels of similarities from Level1 to Level 3 with increasing level of similarity respectively.

Following is the file that summarizes all the mismatches that have been generated,



## Baseline data (test data)

We use this dataset for testing our model.

# Train-Val-Test Split

In the above section, before generating the matches and mismatches datasets from each data source we split unique names to create distinct train and validation datasets. We then separately generate name matches and mismatches for both the datasets as explained above. We use the baseline data to create test dataset.

# Preprocessing

Before generating the features for name matching, we perform same preprocessing steps to the name pairs to simplify the name matching process. Following are the steps that we applied,

* Lower case names
* Strip the names from either sides
* Remove multiple spaces within the names
* Transliterate the names (remove accents) to convert to roman characters (UTF-8 to ASCII)
* Remove all characters except alphabetical characters
* Remove honorific titles like dr. phd. etc.

We took honorific titles in multiple languages. Following are the list of all the titles that we considered,

|  |  |
| --- | --- |
| Region | Title |
| Western | **Mr, Mrs, Miss, Ms, Dr, Prof, Rev** (Reverend for clergy), **Sir** (Honorifics for knights and dames in british honors system) **Lord/Lady** (Nobility titles) |
| Arabic | **Sheikh** (Respect for elders, community leaders), **Sayyid/Sayyida**: For descendents of prophet Muhammed **Abu/Um, Al** |
| South Asia (Ind, Pak, Ban) | **Sri, Smt, Pandit, Swami, Baba** |
| East Asia | **Sensei** (Japan Teacher), **San**, **Sama, Kun, Chan** (Japanese Suffixes Formality and Familiarity Tanaka San, Yuki Chan), **Shi:** Chinese suffix for respect (Mr/Ms), **Sifu:** Martial Art instructors |
| Africa | **Chief, Baba** (Father or elder)**, Imam** (Religious Titles) |
| Latin America | **Don, Doctor, Padre, Lic** (In Mexico people with degree) |
| French | **Monsieur, Madame, Mademoiselle** |
| Germanic countries | **Herr, Frau** |

# Feature Generation

Before feature generation, to simplify the name matching process we use Smith Waterman Algorithm to get the best possible alignment between the name pairs. To do that we obtain different permutations of parts of names and select the best aligned name pairs.

Based on the types of errors there are different similarity/distance based algorithms that can be used to identify matching between two texts. To cater to these different sources of errors that could be present in our dataset we generate our features based on these algorithms. We modify these algorithms to make them independent of length to compare across strings of different lengths and for the simplification of analysis we convert all the distances into similarity measures. This [paper](https://www.researchgate.net/publication/215992032_A_Comparison_of_Personal_Name_Matching_Techniques_and_Practical_Issues) introduces several of these algorithms for personal name matching application grouped into 3 types,

* Phonetic algorithms: These algorithms cater to the spelling errors due to same pronunciations of two different names. The algorithms tend to convert a name string into a phonetic code based on how the name is pronounced. Most of these techniques have been developed with English in mind.
* Pattern matching algorithms: These algorithms are commonly used for approximate string matching which can be used to cater to several types of spelling errors. These are mainly edit distance based (no. of insertions, deletions and substitutions) and qgram based algorithms.
* Combined Algorithms: These algorithms combine phonetic enocoding with pattern matching to improve the matching quality.

Following are the list of features that we have generated and packages that we have used,

|  |  |
| --- | --- |
| **Algorithm** | **Pip Package** |
| soundex | Pyphonetics |
| double\_metaphone | Phonetics |
| jaccard\_2 (2-grams) | Textdistance |
| jaccard\_3 (3-grams) | Textdistance |
| sorenson\_2 (2-grams) | Textdistance |
| sorenson\_3 (3-grams) | Textdistance |
| overlap\_2 (2-grams) | Textdistance |
| overlap\_3 (3-grams) | Textdistance |
| bag | Textdistance |
| levenshtein | Textdistance |
| dlevenshtein | Textdistance |
| jaro\_winkler | Textdistance |
| smith\_waterman | Textdistance |
| editex | Textdistance |
| lcsseq | Textdistance |
| lcsstr | Textdistance |
| bz2ncd | Textdistance |
| zlibncd | Textdistance |

# Feature Selection

We perform both univariate and bivariate analysis to determine the most relevant features for our dataset.

## Univariate Analysis

In this analysis we evaluate the name similarity features on different name variations using average precision score on the train dataset.

We observe most of the features work for simpler errors like space error, truncated string errors, word jumble errors and hyphenated errors, whereas for short strings the overlap algorithms gives the best performance. For complex name variations only bz2ncd (normalized compression distance) algorithm gives the best performance.

The algorithms double metaphone, lcsstr, and zlibncd doesn’t provide any significant performance boost in any of the variations so we decided to eliminate this feature.



## Bivariate Analysis

In this analysis we used pearson correlation to identify and remove correlated features. Following file contains the correlation between the features,



In the correlation matrix, we observe that in the n-gram based methods the 2-gram and 3-gram of the same similarity measures are highly correlated with each other. Based on the performance in the univariate analysis, the 2-grams based similarity outperforms in our dataset. We also observe that the Sorenson similarity is also highly correlated with Jaccard similarity which outperforms the former.

We also observe that the Longest common subsequence similarity is correlated with the editex similarity where the former outperforms the latter. But we also know that the editex algorithm can capture phonetic variations which might not be captured by the Longest common subsequence algorithm.

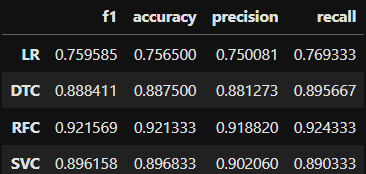
Based on the above analysis, following are the final set of features we have selected,

|  |
| --- |
| soundex |
| jaccard\_2 (2-grams) |
| overlap\_2 (2-grams) |
| bag |
| levenshtein |
| dlevenshtein |
| jaro\_winkler |
| smith\_waterman |
| editex |
| lcsseq |
| bz2ncd |

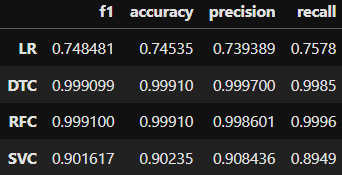
# Modelling

First, we compared the following 4 models (LR = Logistic Regression, DTC = Decision Tree Classifier, RFC = Random Forest Classifier, SVC = Support Vector Classifier) with default hyperparameter settings,

We observed the following performance on validation data using threshold=0.5,



We also observed the following performance on train data,



Here we can observe that the Random Forest Classifier gave the best performance, however with default settings the model tends to overfit. Hence, we further performed hyperparameter tuning to find the best hyperparameters to regularize the model.

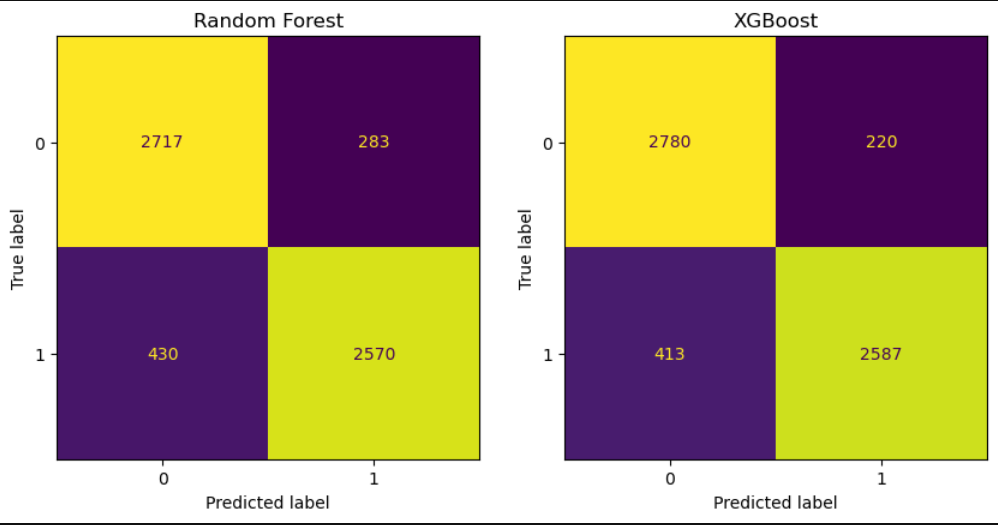
We performed hyperparameter tuning using Randomized search method and compared XGBoost Classifier with Random Forest Classifier. For each set of hyperparameters we recorded the results on both train and validation data.

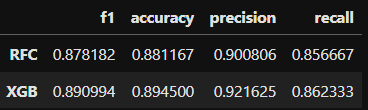
Following file contains the results of hyperparameter search for both the models,



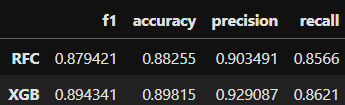
For both family of models we filtered the best performing models based on F1 score on validation data. Out of these best performing models, we selected the least overfitting models as the models with the best set of hyperparameters. We obtain the results for these models below.

Following, we have observed the results of the model on validation data using threshold = 0.5,





We also observed the performance of the model on training data,

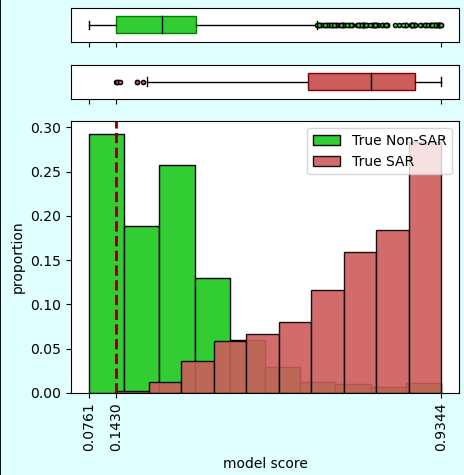
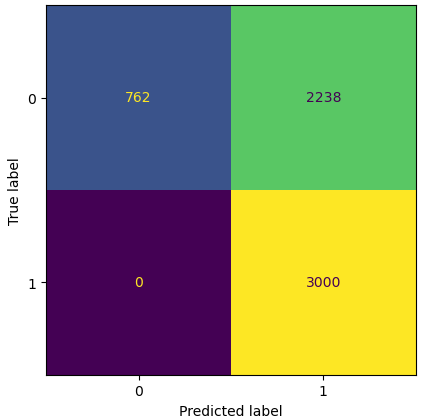


Based on the above results, we can clearly observe that the XGBoost Classifier outperforms Random Forest Classifier in all the metrics. Also, the performance on the training and validation data converges, we finalize XGBoost Classifier as the performing model for name matching application.

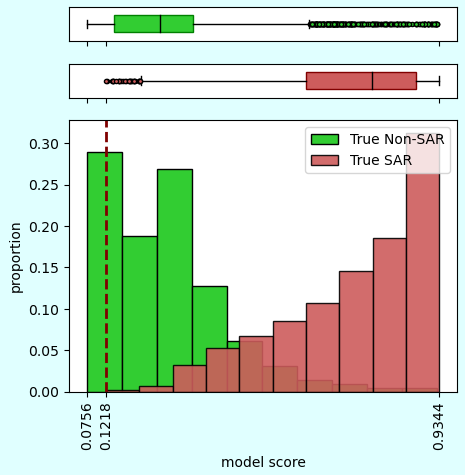
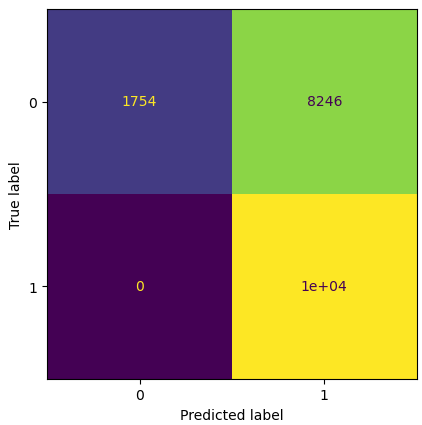
# Threshold tuning

To tune the threshold, we calibrate the finalized XGBoost Classifier for 100% recall on train and validation data,

We obtain threshold = 0.1430 after calibration on validation data for 100% recall.

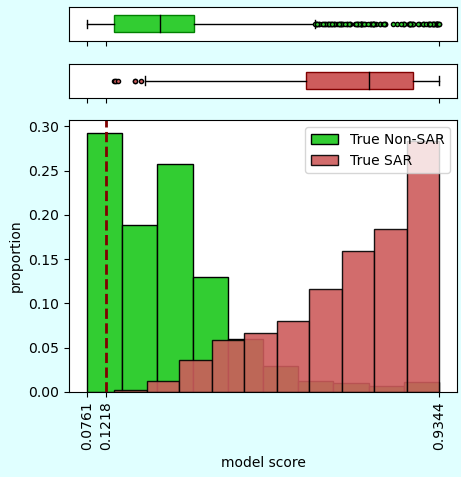
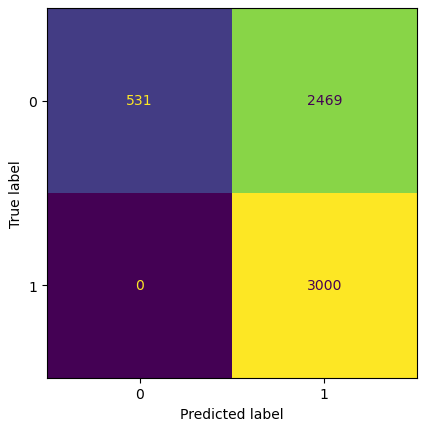
 

We obtain threshold = 0.1218 after calibration on train data for 100% recall.

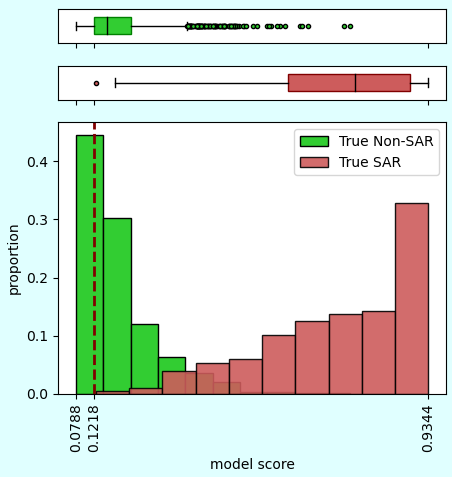
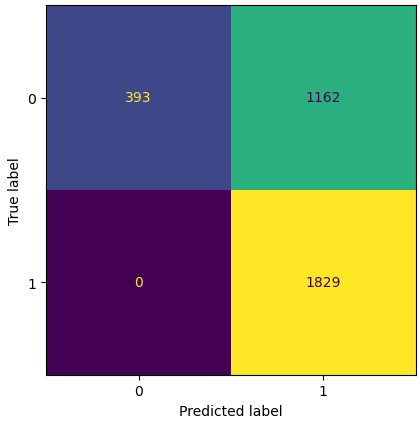
 

We chose the threshold as minimum of the two thresholds = 0.1218. The name pairs with scores below 0.1218 will be identified as false positives.

Following are the new results on validation data using 0.1218 as threshold,

We further evaluated the model on baseline data. Following are the results obtained,

f1 score = 0.7589211618257262,

accuracy score = 0.6566193853427896,

precision score = 0.6115011701771983,

recall score = 1.0

The above results show that our model is disqualifying 25% false positives with 100% recall on the test data.

# Results Analysis

## Validation data

1. Model Score vs Variations

We analyzed the score vs variations for true name matches. We expect higher scores for true matches. From the analysis we observed that as the complexity of variation increases the model score drops which means that the model is finding difficult to predict complex name matches.



1. Model Score vs Length of target name

We also analyzed the score vs length for true name matches. There is no significant insight that we could draw but we observed that there are very few longer names with lower scores which shows that the model could provide higher scores for longer strings as well.

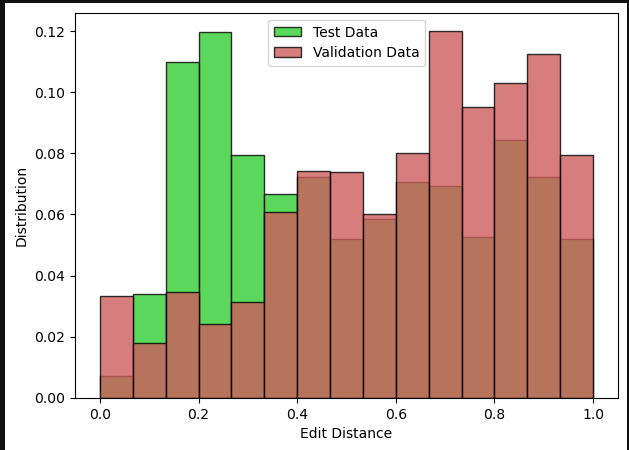


## Baseline (Test) data

1. Data Drift (Test v/s Validation data)

In the following visualization, we have compared the Edit distance (Levenshtein distance) between the name pairs in Test and Validation data for true name matches.

We can observe that the true name pairs in validation data that we have generated have higher edit distances as compared to the test data which shows the conservativeness of our data. We have used this data to calibrate the threshold for our model for zero risk which makes it even more conservative when applied to other datasets.



1. Model Score vs Length of target name

We analyzed the model score vs length of target name in the test data for true name matches. There is no significant trend that can be observed from this data.



# Control Mechanisms

So far, we have tried to keep our solution approach highly conservative to keep up with zero risk tolerance. We have trained and calibrated our model on a very conservative data with complex name pattern. The 100% recall on the test data also proves the conservativeness in our approach. However, to prevent any obvious mismatches we can further complement our solution with control mechanisms. Following are the control mechanisms, that we have added to our solution,

* In the name pair is one name abbreviation of another name?
* Is one name a subset of another name?

If any of the above questions are valid then the name pair will not be disqualified and will be raised as true match.

# Model Usage

For model usage refer to the README file in Github repository

# Conclusion

In this study, we have tried to solve the personal name matching problem by providing a solution that can be implemented on the top of the existing conventional systems to improve its efficiency while respecting the zero-risk tolerance. First, we generated a very conservative name pairs dataset having variations with multiple levels of complexity using data extracted from multiple sources simulating different types of errors. Using this data, we explored several existing text similarity algorithms and then we combined these algorithms into a unified model utilizing these as features. This enabled us to disqualify 25% of the false name pairs on a completely exclusive baseline dataset with 100% recall. Throughout our approach we have tried to be both parsimonious and conservative for the scalability, ease of implementation and to respect the zero risk tolerance as a crucial requirement in any financial institution. As we were in the limited time of the scope of the hackathon, we have restricted ourselves to classical machine learning approaches but in future this solution can be improvised by exploring state of the art deep learning algorithms.

# References

[1] [A Comparison of Personal Name Matching: Techniques and Practical Issues](https://www.researchgate.net/publication/215992032_A_Comparison_of_Personal_Name_Matching_Techniques_and_Practical_Issues)

[2] [XGBoost Documentation — xgboost 2.1.2 documentation](https://xgboost.readthedocs.io/en/latest/index.html)

[3] [GitHub - sigpwned/popular-names-by-country-dataset: A dataset of popular forenames and surnames by country](https://github.com/sigpwned/popular-names-by-country-dataset)

[4] [Handwriting Recognition](https://www.kaggle.com/datasets/landlord/handwriting-recognition)

[5] [nicknames/names.csv at master · carltonnorthern/nicknames · GitHub](https://github.com/carltonnorthern/nicknames/blob/master/names.csv)