# Exiger (NYC) – Detecting Language of Name Origin

**Author**: Stefani Hermanto
**Teammates**: Jolie Huang, Anna Yang, Ariana Beckford, Wildline Lincifort
**GitHub**: https://github.com/shermanto24/name-language-of-origin
**Presentation**: Copy of Exiger: Fall 2023 AI Studio | Final Project Presentation

## Business Context

Our team of five worked with Exiger to develop a model that can identify the language of origin of a given name. The purpose was to enable DDIQ, Exiger's AI-Based due diligence solution, to more accurately identify parts of names. This would improve name recognition and lessen false positives and negatives, helping Exiger better assess risks for clients.

## Data Preparation and Validation

### DATASET DESCRIPTION

We were given the following files to work with: company_person_name_dataset.csv, 10 xlsx files in the exigerData folder, and 18 txt files in the txt_dataset folder. The company_person_name_dataset.csv was available on Kaggle, and contains names of both companies and people. Aside from the csv file, the other files each correspond to a single language.

### EDA (EXPLORATORY DATA ANALYSIS)

We first decided to not work with the txt_dataset folder because each file contained only single names (only first or only last name) and the sample sizes for each language were small. Then, we split up the remaining datasets among ourselves. Since the company_person_name_dataset.csv file needed the most cleaning, we had

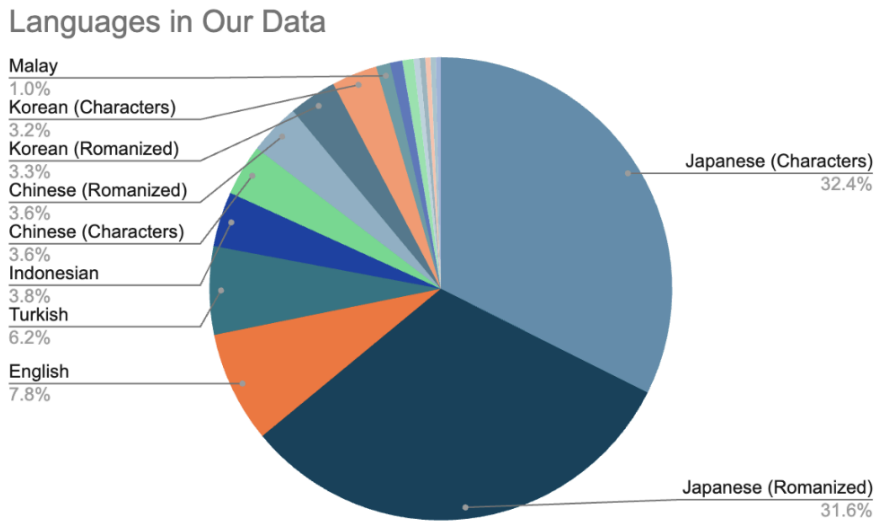
two teammates work on it. The rest of us each worked on at least two files from the exigerData folder. I worked on the files with Indonesian and Malay names.

We did some preprocessing to remove unwanted columns, including blank or unnamed columns, comment columns containing notes for some names, and the first and last name columns. My teammates who worked on the company_person_name_dataset.csv file dropped the names of companies, which were labeled with a class of 0. Afterwards, for each dataset we were left with a single column containing full names.

Some steps in our data cleaning process included removing names with special characters, removing null and duplicate entries, etc. At this stage, most of the issues we ran into were specific to the languages we worked on. For example, we received two datasets of Indonesian names, but after combining them I had to drop around ¾ of the data due to the names being duplicates.

Our data initially contained names from 103 languages, and most of them were in the company_person_name_dataset.csv file. However, we decided to drop languages that had less than a thousand samples because the languages with small samples performed poorly. In the end, we worked with 17 languages, totaling to about 293,000 names.

I handled most of the data consolidation, and after I finished this was the breakdown of our data. Japanese was the language with the most samples, taking up around two thirds of the data, followed by English having the second most samples.




Total for cleaned data:
292,900 names,
17 languages

Smaller languages not labeled:

- Spanish
- Vietnamese
- Italian
- French
- Portuguese
- German
- Arabic (Romanized)

FEATURE ENGINEERING

The following are the features we engineered:

| | |
|---|---|
| alphabet | a list with the alphabet of each character in the name using the name function from the unicodedata Python library |
| unigrams | a list of every character in the name |
| bigrams | a list of every adjacent 2 characters in the name |
| trigrams | a list of every adjacent 3 characters in the name |
| word_ngrams | a list of every part of the name<br><br>- We did not use this since this is more suited for applications like sentiment analysis. Instead we used character unigrams, bigrams, and trigrams. |
| name_length | length of full name |
| avg_token_length | average token length, where each part of the name is a token |
| num_tokens | number of tokens in the name |
| num_alphabets | the number of unique alphabets in each name according to the alphabet feature |
| edit_distance | minimum number of operations (insertion, deletion, or substitution) to get from the original name to its transliteration |
| accent_count | the number of accents in the name using the normalize function from the unicodedata Python library |

| | |
|---|---|
| transliteration | transliteration of the name using the unidecode Python library |
| period_freq, dash_freq, apostrophe_freq, space_freq | the number of times the corresponding special character appears in the name |
| lang_unigrams_cosine_sim | the cosine similarity score of this name's unigrams distribution compared to every language's unigrams distribution<br><br>- Explained more below as it made up the majority of our features. |

Here is an example of what some of these features look like with names:

| | fullname | original_fullname | alphabet | unigrams | bigrams | trigrams | char_ngrams | word_ngrams | name_length |
|---|---|---|---|---|---|---|---|---|---|
| **0** | annuar rapaee | Annuar Rapaee | [LATIN, LATIN, LATIN, LATIN, LATIN, LATIN, SPA... | [a, n, n, u, a, r, , r, a, p, a, e, e] | [(a, n), (n, n), (n, u), (u, a), (a, r), (r, ... | [(a, n, n), (n, n, u), (n, u, a), (u, a, r), (... | [a, n, n, u, a, r, , r, a, p, a, e, e, (a, n)... | [annuar, rapaee] | 13 |
| **1** | tash yong | Tash Yong | [LATIN, LATIN, LATIN, LATIN, SPACE, LATIN, LAT... | [t, a, s, h, , y, o, n, g] | [(t, a), (a, s), (s, h), (h, ), ( , y), (y, o... | [(t, a, s), (a, s, h), (s, h, ), (h, , y), (... | [t, a, s, h, , y, o, n, g, (t, a), (a, s), (s... | [tash, yong] | 9 |

The features that we fed into the model were ultimately all numerical, namely, the following:

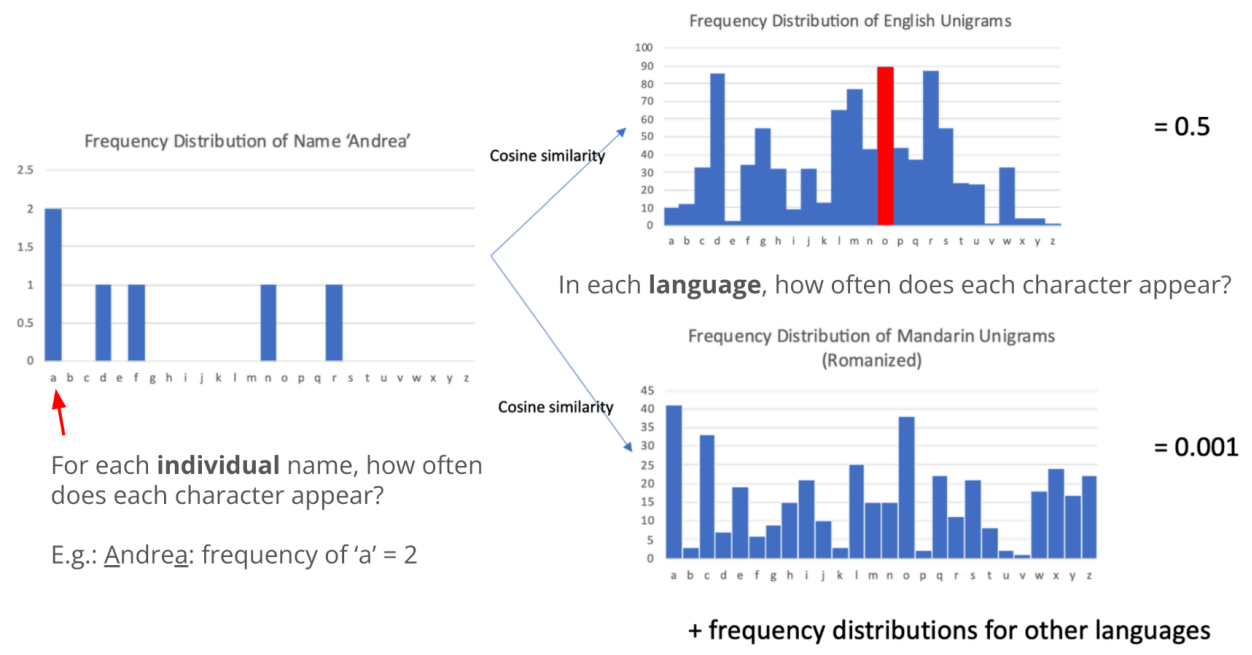
```
In [95]: merged_df.columns

Out[95]: Index(['name_length', 'avg_token_length', 'num_tokens', 'period_freq',
                'dash_freq', 'apostrophe_freq', 'space_freq', 'bigrams_cosine_sim',
                'accent_count', 'num_alphabets', 'edit_distance',
                'indo_unigrams_cosine_sim', 'malay_unigrams_cosine_sim',
                'viet_unigrams_cosine_sim', 'chinese_unigrams_cosine_sim',
                'turk_unigrams_cosine_sim', 'korean_unigrams_cosine_sim',
                'japan_unigrams_cosine_sim', 'arab_rom_unigrams_cosine_sim',
                'eng_unigrams_cosine_sim', 'french_unigrams_cosine_sim',
                'german_unigrams_cosine_sim', 'ital_unigrams_cosine_sim',
                'portug_unigrams_cosine_sim', 'span_unigrams_cosine_sim',
                'parentheses_freq', 'quotation_freq', 'language'],
               dtype='object')
```

**N-gram Distributions**

Character n-grams (unigrams, bigrams, and trigrams) were essential to our project because they are able to capture patterns in names. N-grams are commonly used in natural language processing (NLP), and for our purposes, character n-grams provide more insight than word n-grams since the latter is more useful in applications like sentiment analysis. In order to feed n-grams into the model, we had to represent them numerically, which we did in three steps:

1. **Language n-gram distributions**: created character n-gram distributions for each language
    a. It would have been ideal to find these distributions from other sources so that they would be sourced outside our own data. However, they were difficult to find for every language and would have been derived from a wide variety of words, not just names, which could lead to lower similarity scores. Creating the distributions was more feasible, but could be a potential source of bias.
2. **Individual n-gram distributions**: created character n-gram distributions for each name
3. **Similarity scores**: used cosine similarity to compare the distributions against each other. We compared every individual distribution against the distribution of every language.

Here is a visual example where the unigrams distribution of the name ‘Andrea’ is compared to the English and Chinese distributions.




The end goal was to feed the similarity scores into the model. The similarity scores for the name ‘Andrea’ could look like this:

| Name | Name Length | Alphabet | Cosine Similarity with English Unigram | Cosine Similarity with Chinese Unigram | Cosine Similarity with French Unigram | Cosine Similarity with .... |
|---|---|---|---|---|---|---|
| Andrea | 6 | latin | 0.5 | 0.001 | 0.2 | 0.43 |

This was the most time-consuming part of our project as it took about 3-4 weeks. I spearheaded the coding process for all three steps and wrote reusable commented code for my teammates to use on their own datasets to ensure that it would work as expected. By having everyone test the code individually, we were able to troubleshoot issues before we ran the code on the consolidated data. For example, my teammate Jolie was unable to create individual bigrams and trigrams distributions for the Chinese datasets because there were too many unique characters and the kernel died. I had faced the same issue with Indonesian names, but only with trigrams. As a result, we realized that we lacked the computational

resources needed to create *both* bigrams and trigrams distributions across the consolidated data, since it would contain even more unique characters. We communicated this to our Challenge Advisor Anna and she permitted us to use only unigrams since our resources were limited.

## Approach

### SELECTED MODELS

We selected five classifier models to categorize names into the right languages:

1. **Random forest** – we chose this model because it works well for NLP tasks, is relatively fast and simple to implement compared to other models, and is less prone to overfitting since it trains on many different decision trees.
2. **K-nearest neighbors** – we chose this model because it easily handles multinomial classification, is fast and simple to implement, and can capture non-linear patterns in data.
3. **Multinomial Naive Bayes** – we chose this model because it has a fast runtime and usually works well for NLP tasks.
4. **Logistic regression** – we chose this model because it is easy to implement and works well for classification problems.
5. **Gradient boosting** – we chose this model because it can capture non-linear patterns in data.

As students with limited time and computational resources, we generally opted for models that had a fast runtime. I did an initial run for each model with RandomizedSearchCV to provide some boilerplate code that my teammates could use to further tune the models. Then, each of us worked on one model, and I worked on tuning the random forest model.

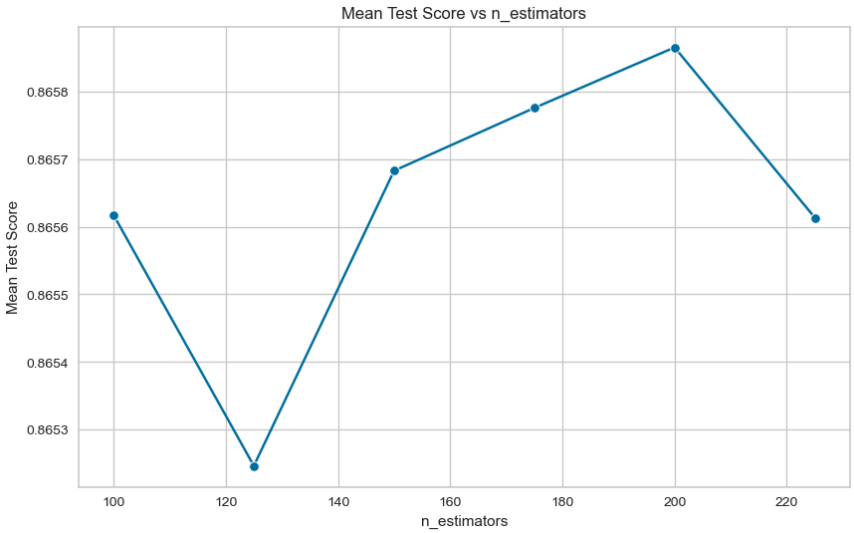
### EVALUATION METRICS

We chose **precision** as our primary method of evaluation because it is centered around reducing false positives, which is essential to our business context. For us, a false positive is when a name is predicted to be a target language, but it is actually a different language. If a language has a high precision score, it means that our model is predicting that language correctly most of the time. In other words, we prioritized confidence in identifying the correct language of origin. Furthermore, precision works well for data with imbalanced classes, which is applicable to our data since we

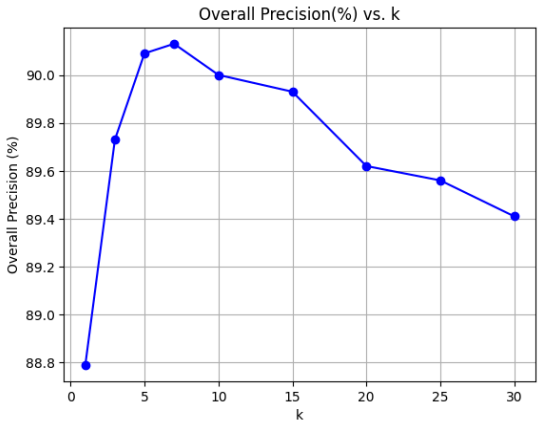
had over 150,000 Japanese names while other languages had as low as 1,000 samples. We specifically used macro-average precision to assess performance across all languages and weighted average precision to assess performance of the languages with the most samples. We also used accuracy, recall, and F1 score to provide other insights into our model performance.

## HYPERPARAMETER TUNING

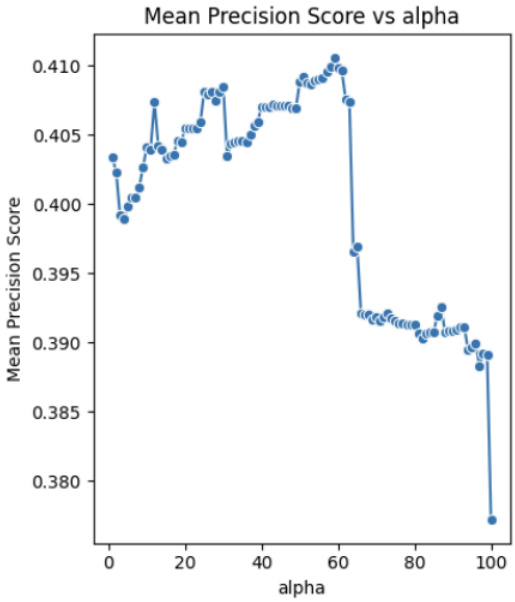
1. **Random forest**
   Best number of decision trees (n_estimators): 200
   Test set weighted precision: 87.94%

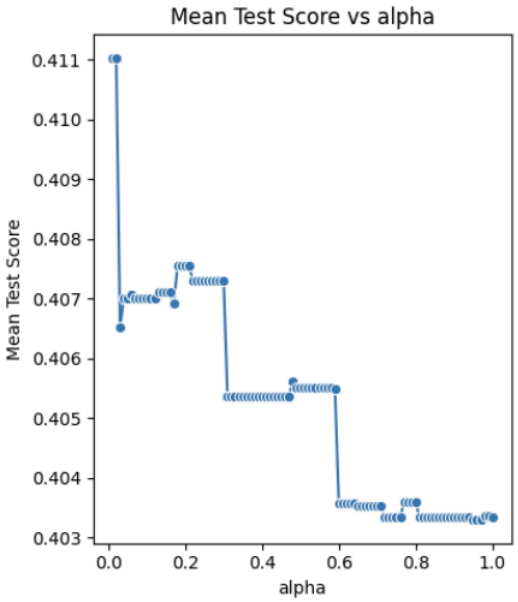



2. **K-nearest neighbors**
   Best number of neighbors (k): 7
   Test set precision: 90.13%



3. **Multinomial Naive Bayes**
   Best alpha value (1–100): 1
   Best alpha value (0.01–1): 0.03
   Test set precision: 45%




4. **Logistic regression**
   Maximum number of iterations (max_iter): 10000
   Test set precision: 74%

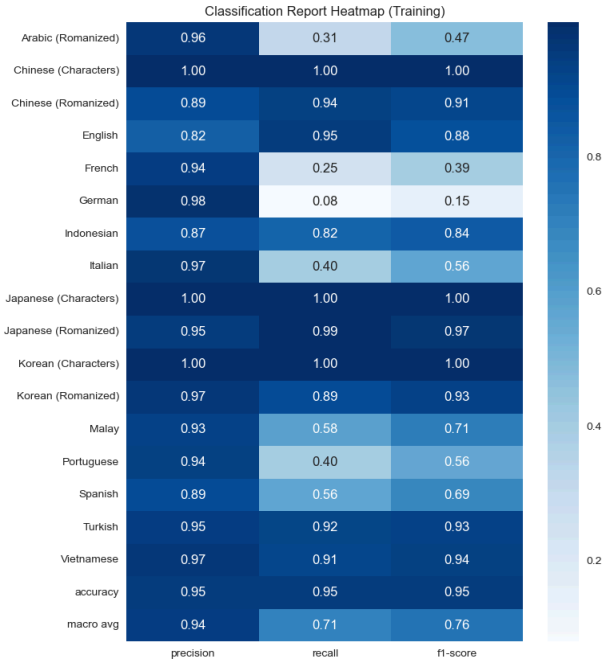
5. **Gradient boosting**
   We were unable to extract results; hyperparameter tuning with GridSearchCV had not finished running after 3 days.

PRECISION SCORES AND CLASSIFICATION HEATMAPS

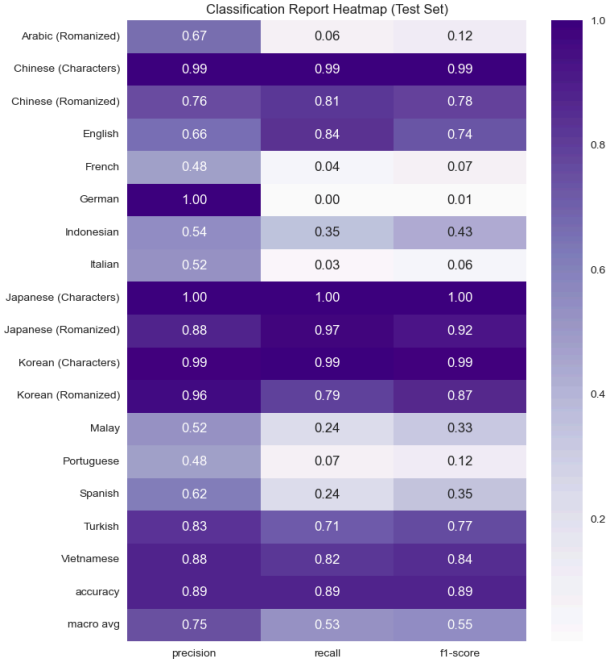
We created the following classification heatmaps in order to better visualize our results. The columns from left to right are precision, recall, and F1 score. Darker cells indicate higher scores for those languages.

1. **Random forest**

Macro training set precision: 94%
Weighted training set precision: 96%

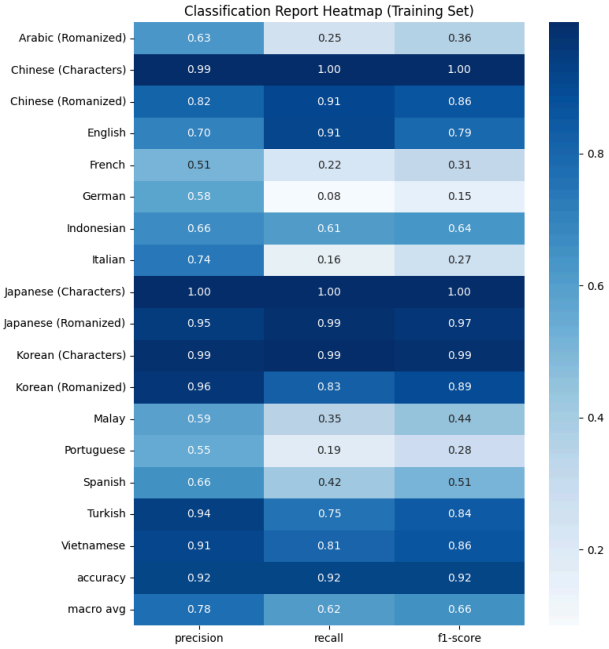



Macro test set precision: 75%
Weighted test set precision: 88%

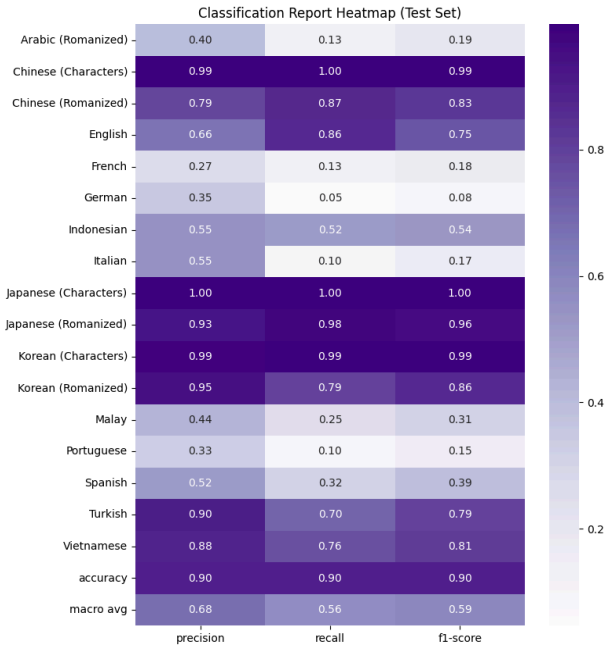


2. **K-nearest neighbors**

Macro training set precision: 78%
Weighted training set precision: 92%

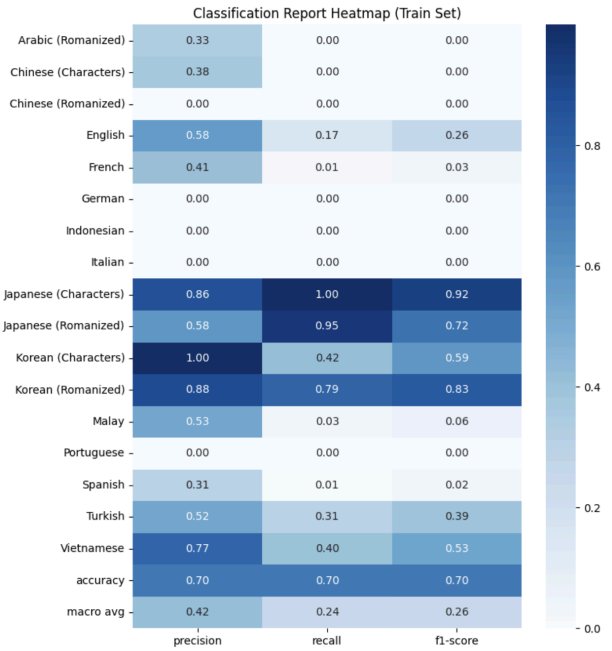



Macro training set precision: 68%
Weighted test set precision: 90%

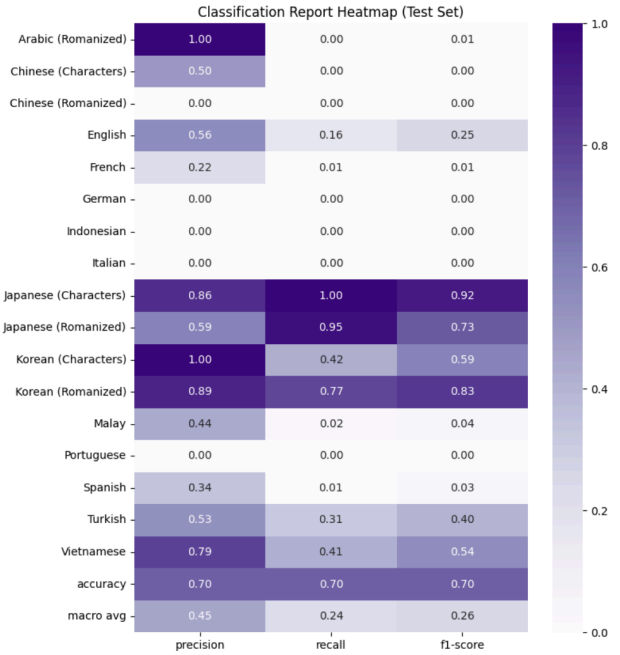



3. **Multinomial Naive Bayes**

Macro training set precision: 42%
Weighted training set precision: 63%

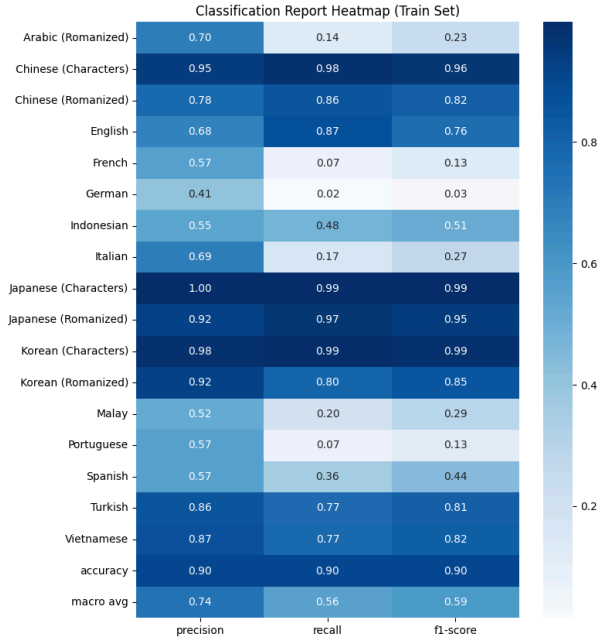



Macro test set precision: 45%
Weighted test set precision: 64%

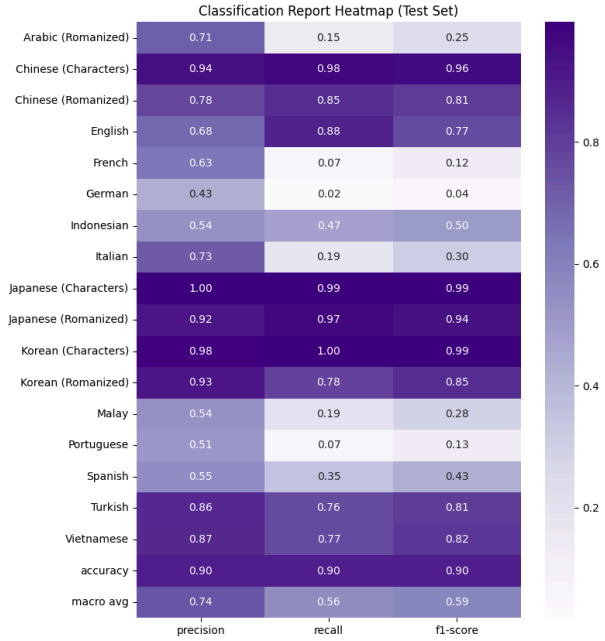


4. **Logistic regression**

Macro training set precision: 74%
Weighted training set precision: 89%




Macro test set precision: 74%
Weighted test set precision: 89%




## RESULTS AND ANALYSIS

### Language Performance and Impact of Sample Size

The languages with the highest precision were consistently **Chinese**, **Korean**, and **Japanese** (CJK), followed by **English**, **Turkish**, and **Vietnamese**. In general, sample size had the most impact on performance, but there were exceptions. Since Japanese names made up around 64% of our samples, it is no surprise that it performed well throughout all four models. While English had the second most samples—taking up around 7.8% of our data—its precision was always worse than that of Chinese and Korean, which each made up around 6–7% of our data. Turkish, which took up 6.2% of our data, performed a little better than English. What was most interesting was that Vietnamese, making up less than 1% of our data—equivalent to less than 3000 samples—had similar scores to Turkish.

**Random Forest: Feature Importances**

To further investigate, I looked into my random forest model's feature importances.

```
edit_distance                  0.282322
japan_unigrams_cosine_sim      0.107250
avg_token_length               0.078073
name_length                    0.054848
chinese_unigrams_cosine_sim    0.041380
num_alphabets                  0.040522
```

```
viet_unigrams_cosine_sim       0.034360
eng_unigrams_cosine_sim        0.015606
accent_count                   0.015479
dash_freq                      0.013133
num_tokens                     0.009694
space_freq                     0.008860
```

The features with the most impact on the random forest model were **edit distance**, **Japanese unigrams cosine similarity**, and **average token length**.

As mentioned earlier, edit distance is the minimum number of operations needed to get from a name to its transliteration. For example, the edit distance between tử hoàng thông and its transliteration, tu hoang thong, is 3. Edit distance is one reason why Chinese, Korean, Turkish, and particularly Vietnamese performed well despite having less samples, since names of those languages tend to have edit distances greater than 0. On the other hand, the edit distance is usually 0 not only for English names, but also for other languages like Indonesian, Malay, and Spanish, which had poor performance.

Japanese cosine similarity being the second most important feature makes sense given that Japanese names made up the majority of our data. Average token length likely contributed to Chinese, Korean, and Vietnamese names' high performance since they tend to have shorter token lengths.

**Comparing Model Performance**

In terms of **macro average** test set precision, **random forest** and **logistic regression** had the highest scores of **75%** and **74%** respectively, followed by k-nearest neighbors with a score of 68% and multinomial Naive Bayes with a score of 45%. Since macro average precision gives equal weight to all languages, this highlights room for improvement in performance for the languages with fewer samples—namely, Arabic (Romanized), English, French, German, Indonesian, Italian, Malay, Portuguese, and Spanish.

In terms of **weighted average** test set precision, **k-nearest neighbors** was the best performing model with a score of **90%**. **Logistic regression** and **random forest** followed closely with scores of **89%** and **88%**, and multinomial Naive Bayes had the worst score of 64%. Since weighted average precision assigns more weight to classes with more samples, this means that the models performed well for languages with large sample sizes, which reinforces the findings in this section.

For both random forest and k-nearest neighbors, training set precision was higher than test set precision, which could be an indication of overfitting. With more time, we would have further investigated this and performed more hyperparameter tuning.

Multinomial Naive Bayes consistently performed worse than the other three models even though it typically works well for NLP problems. We believe this may be because we only kept numerical features and Naive Bayes works with categorical features.

Logistic regression had the same precision scores for both the training and test sets, which suggests more reliability for performance on unseen data.

## Insights

Our team learned a lot over the course of this project. Along with gaining experience in NLP, the following are some other technical skills I learned:

- Setting up a Python environment with conda
- Working with larger datasets needing substantial cleaning
- Working with text data
- Leveraging unidecode for transliteration
- Implementing n-grams frequency distributions

Regarding data science, here are some of our general takeaways from this project:

1. **The more data, the better.** Our dataset was very imbalanced, which was reflected in the performance of the languages with less samples. With more data, our models would have certainly improved.
2. **Data preparation takes a long time.** This was especially true for my teammates who worked on the company_person_name_dataset.csv file. Moreover, even after we thought we were done with this stage, occasionally we would have to go back and clean some part of the data.
3. **Lack of computational resources causes limitations.** Prior to this project, we had only worked with small datasets throughout our summer course. Having to drop the bigrams and trigrams distributions, encountering issues exporting the Japanese dataset due to its size, and being unable to run the gradient boosting model to completion made it clear that computational resources are essential for larger scale projects like this one.
4. **Data science is a learning process.** As long as you can justify your methods, there is rarely a "right" or "wrong", but there is always room for improvement.

ACKNOWLEDGEMENTS

Working on this project has been an incredible experience. I've learned so much over the past few months and I am eager to apply my newly gained skills in my future career endeavors. I'm immensely grateful to my teammates Jolie, Anna, Ariana, and Wildline for always being communicative and for putting their all into seeing this through. To our Challenge Advisor, Anna, thank you so much for your kind encouragement, detailed answers to our many questions, and for bringing over two hundred people from Exiger to our final presentation! Last but not least, thank you to Break Through Tech for providing us with this opportunity! The program has been amazing and I look forward to Spring AI Studio.