# dERIVABLE 1

iNTRODUCTION

In this document, the whole process of detecting a language from a text is presented. First, a preprocessing of the dataset containing the corpus is preprocessed. To continue, the dataset is split in training and testing, the training part was used to construct the classifier used to detect the language. All these processes and its results, as well as the variations that were considered to increase the performance of the classification task are exposed through the following sections. The programming language used was [Python](https://www.python.org/) due to its simplicity and available libraries such as the leading actor [NLTK](https://www.nltk.org/) (Natural Language Toolkit) or [Pandas](https://pandas.pydata.org/) lor data management.

All results and explanations are also available in our [GitHub](https://github.com/norhther/MUD-project) repository.

Preproces

Although it is possible to apply brute force to the process and use the corpus as it is first presented. A basic Bayes classifier used for preliminary testing, provided better results when preprocessing tasks were applied in advance. This seems to be the most reasonable approach for many reasons.

First, some of the data entries of the original corpus, contained more than one phrase of the same language. This could result in a bad detection of the context of each phrase. Identification of some languages like Chinese, can be significantly influenced by different sentence structure. It is because all of this that the [*sent\_tokenize*](https://www.nltk.org/api/nltk.tokenize.sent_tokenize.html)method was used in first place, and labels of languages were properly cloned.

Secondly, a finer tokenization of words was performed with the [*word\_tokenize*](https://www.nltk.org/api/nltk.tokenize.html) function was used. Splitting each sentence in atomic units is essential when performing any other analysis, since it provides a basic unit to work with to the classifier. Since a bi-gram approach is firstly used, this step becomes fundamental.

Lastly, the tokenized corpus goes through a stemming process with the [*stem*](https://www.nltk.org/howto/stem.html) object of NLTK. This process, reduces a word to its root, formally known as *lemma.* By removing suffixes and prefixes as well as other lexicographical modifications, this makes the use of vocabulary (most frequently used words), more effective, since many words are mapped to a single root.

Classification

For the classification task, 4 kinds of classifiers were considered: NaiveBayes, SVC, RandomForest and Knn. Since we wanted to extend the classification results of these models as much as possible, we extended the experiments by variating the n-grams and the size of the vocabulary. This classification task was performed over chars and tokenized words, providing different results for each of the approaches

Table 1.1: Naive Bayes Classifier



Table 1.2: Knn



Table 1.3: Random Forest



Table 1.4: SVC



Classification by Words

In our first approach, the classification of the language was performed by using tokenized words. The tables presented above, contain all results generated ordered first by weighted F1 and second by coverage. The F1 statistic, measures the precision by taking into account the sensitivity (true positive rate). The first classifier used was one of Naive Bayes kind, however, we considered that there was room for improvement and decided to try the other 3 types.

As it becomes visible at first sight, the second most relevant factor apart from the model structure is the vocabulary size, it becomes natural, that increasing the number of words in the pull will result in a direct increasement of the classifying success. The coverage also increases, since it is directly bonded to the number of words in the vocabulary.

This does not happen when the ngrams vary, since we can’t see any difference in the until the fourth decimal in F1(weighted). However, the ngrams does seem to have impact over the coverage. There is a direct correlation with the number of words that we cover. This is coherent with the fact that, the higher the aggregation in the analysis, the more we travel ovber the vocabulary.

When all these results are taken into account, we can claim that the best results are achieved with the Random Forest classifier with a vocabulary size of 5000 and the use of unigrams, resulting in a F1 score of 0.943 and a coverage of 0.397. The results will be analyzed later in the corresponding section.

Classification by Chars

This second approach is based in the use of chars as an atomic unit when training the classifier. Results are presented in the below tables, again separated by vocabulary size and ngram type.

The notable increasement in the F1 score and coverage immediately draws our attention. Changing the vocabulary size results in F1 increasement as well as varying the ngrams result in coverage improvement. Although we can observe improvement over all the models, the Random Forest with 5000 of vocabulary size, using unigrams, gives the best performance with 0.982 of F1 weighted score and 0.94 of coverage, which highly surpasses the previous performance.

Table 2.1: Naive Bayes Classifier



Table 2.2: Knn



Table 2.3: Random Forest



Table 2.4: SVC



Results & Interpretation

Confusion matrix

Table 3: Best and Worst



For the different models and hyperparameters tested, this is the best and worst results by F1 (weighted) score. We are going to give the best 2 and worst 2 classifiers that we found:

For the language identification task with characters, the best classifier was the random forest (rf) with ngrams (1,3) and a vocabulary size of 2000. This classifier achieved a high F1 (weighted) score of 0.982775. Although its coverage was lower than the worst classifier, the confusion matrix showed that the rf model performed well in identifying the languages with no remarkable errors. Conversely, the worst classifier was the naive bayes (nb) with a character vocabulary size of 5000 and a coverage of 0.999286. Despite its high coverage, the nb model had the lowest F1 (weighted) score among all the models and hyperparameters tested. The confusion matrix showed that the nb model had difficulties distinguishing between Dutch, Estonian, and Swedish. Latin was also missclassified as Estonian. However, the nb model performed well in identifying other languages.

For the language identification task with words, the best classifier was the random forest with a word vocabulary size of 5000 and ngrams (1,1). This classifier achieved an F1 (weighted) score of 0.973735. The confusion matrix showed that the rf model performed well in identifying the languages, with no remarkable errors. Conversely, the worst classifier was the k-nearest neighbor (knn) with a word vocabulary size of 5000 and ngrams (1,1). This classifier achieved an F1 (weighted) score of 0.815479. The confusion matrix showed that the knn model had difficulties distinguishing between Chinese and Japanese. Other languages such as Hindi, Korean, Indonesian, and Thai were also mistaken for Chinese. This could be due to the small dataset size for some languages or tokenization issues.

In conclusion, the results show that the random forest model is the best classifier for this language identification task among the other models we tried, regardless of whether characters or words are used. The choice of hyperparameters such as ngram and vocabulary size can significantly impact the performance of the models. Also, it seems like the task is easier to perform using only characters, as with a small vocabulary and the appropiate model, we can achieve the best results.

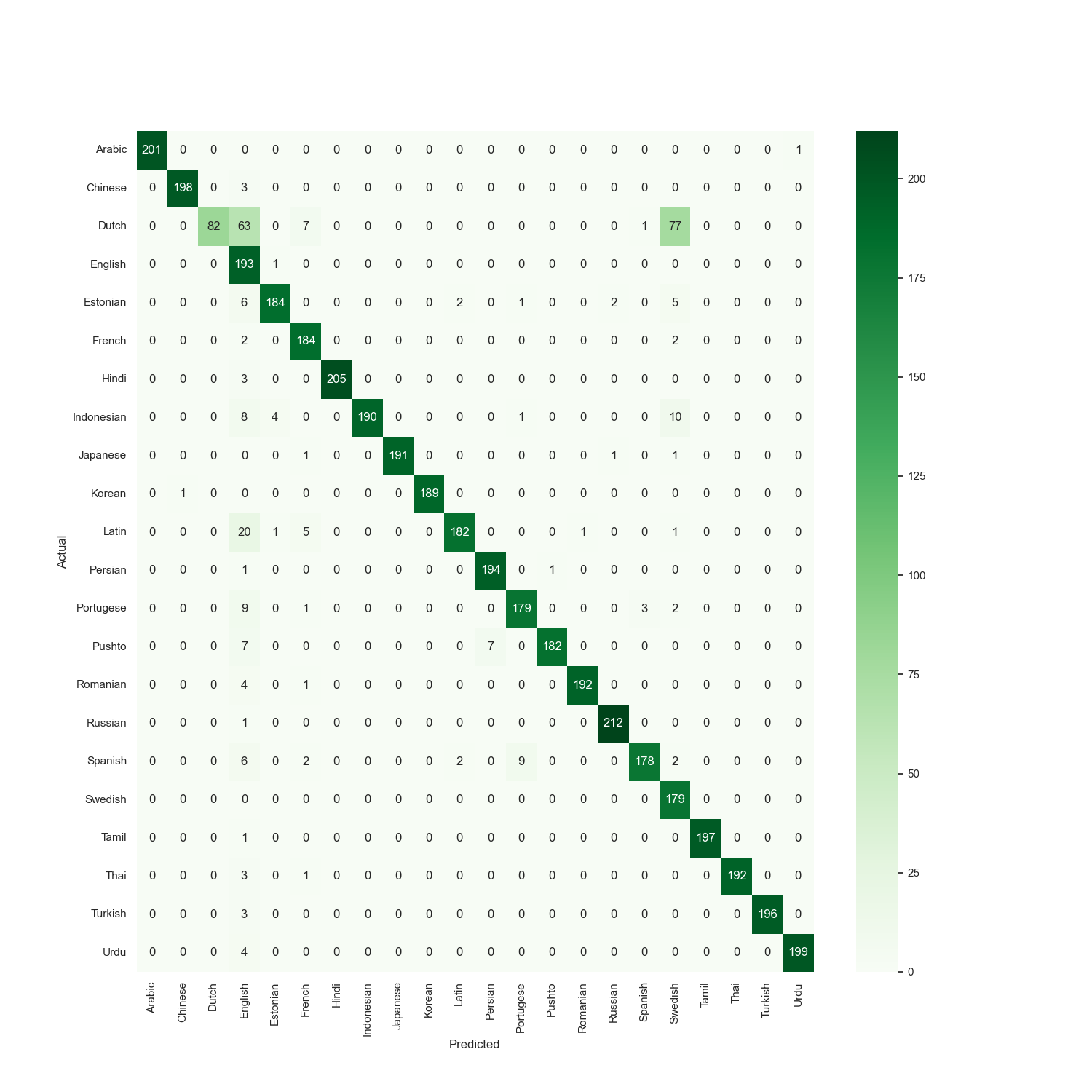
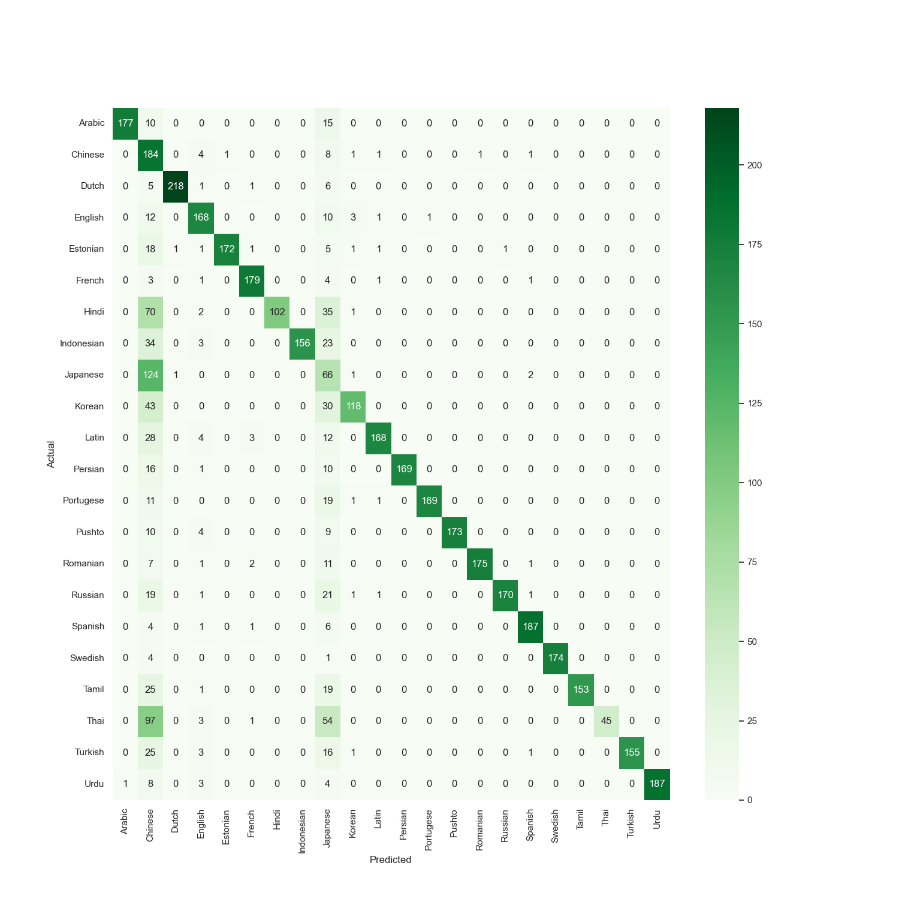
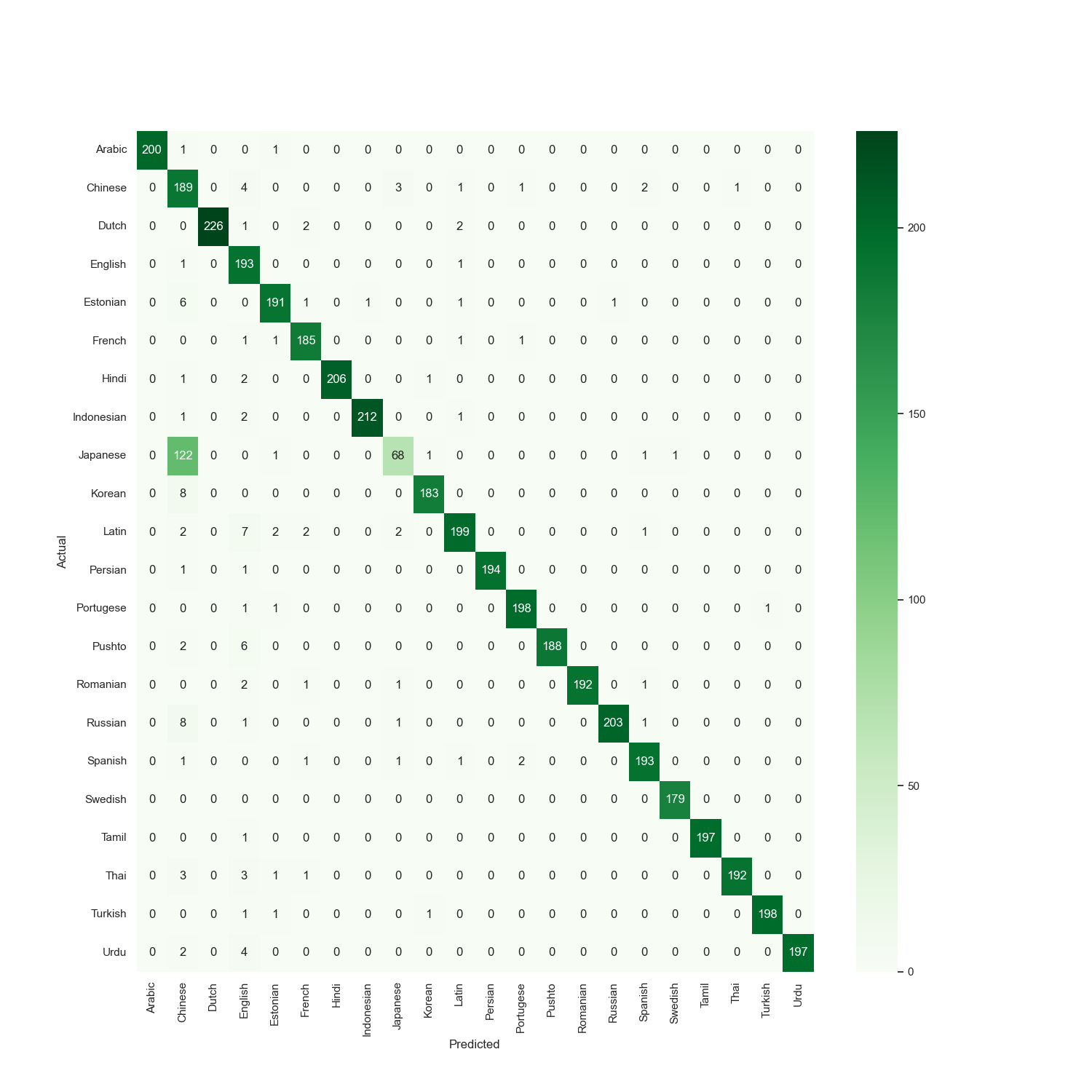
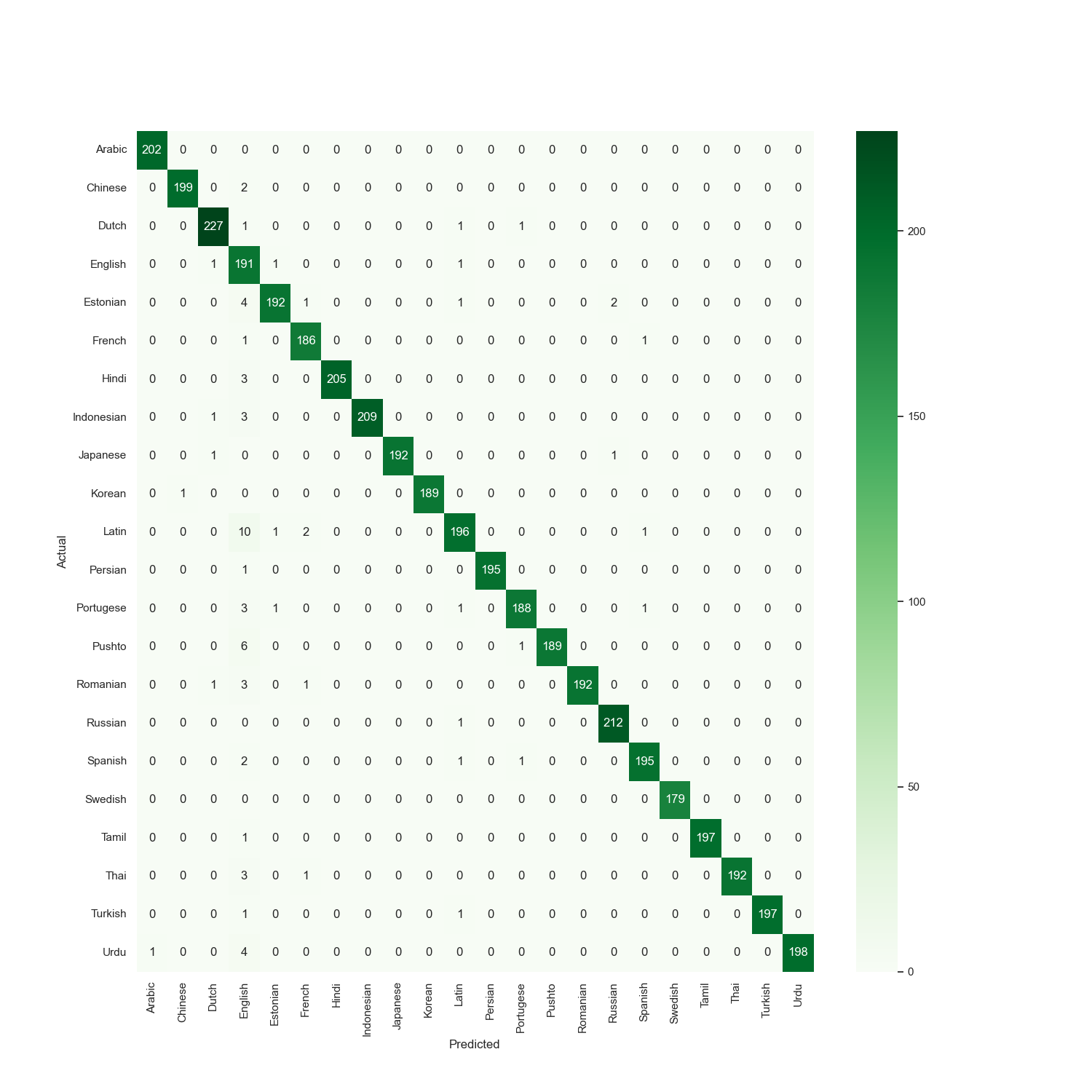


Figure 2: Naive Bayes Char

Figure 1: Knn Word

Figure 4: Random Forest Char

Figure 3: Random Forest Word



Results: Pca

When analyzing the principal component analysis (PCA) of our language identification task, we encounter the challenge of interpreting the axes since we are comparing languages. While it may be tempting to speculate that certain axes are correlated with specific language families, we cannot make definitive conclusions in this regard.

Instead, we can observe the PCA plots for the two best and two worst performing classifiers. We found that the naive Bayes model with a 5000 vocabulary size for character-level features had a nearly identical shape to the random forest model with a 2000 vocabulary size and ngrams of size 3. Notably, clusters of languages such as Chinese, Japanese, and Thai can be observed, as well as a cluster of Latin languages such as Spanish and German. This cluster appears particularly dense.

Similarly, the K-nearest neighbors and random forest models for word-level features also exhibited nearly identical shapes in the PCA plot. In this case, we observe clusters of languages, but with more dispersion along the x and y axes compared to the character-level models.

Although we cannot definitively interpret the axes of the PCA plot, we can observe the clustering behavior of our best and worst performing models and gain insights into the performance of our language identification task.

Figure 5: PCA Knn Word

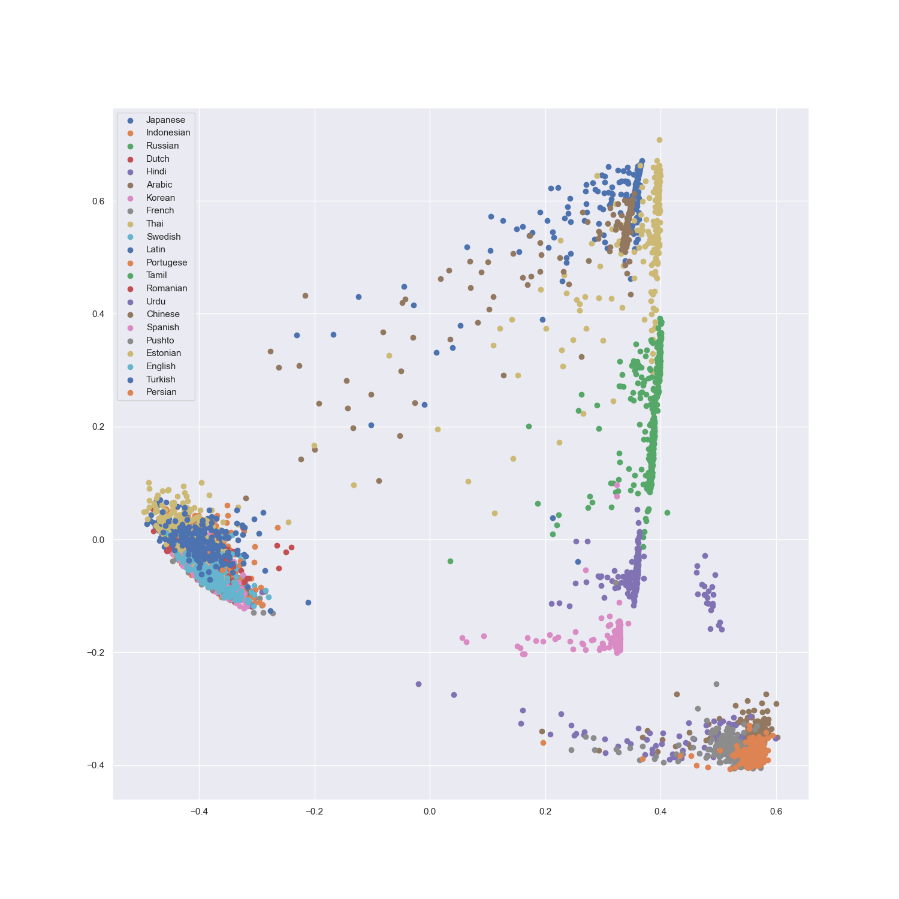
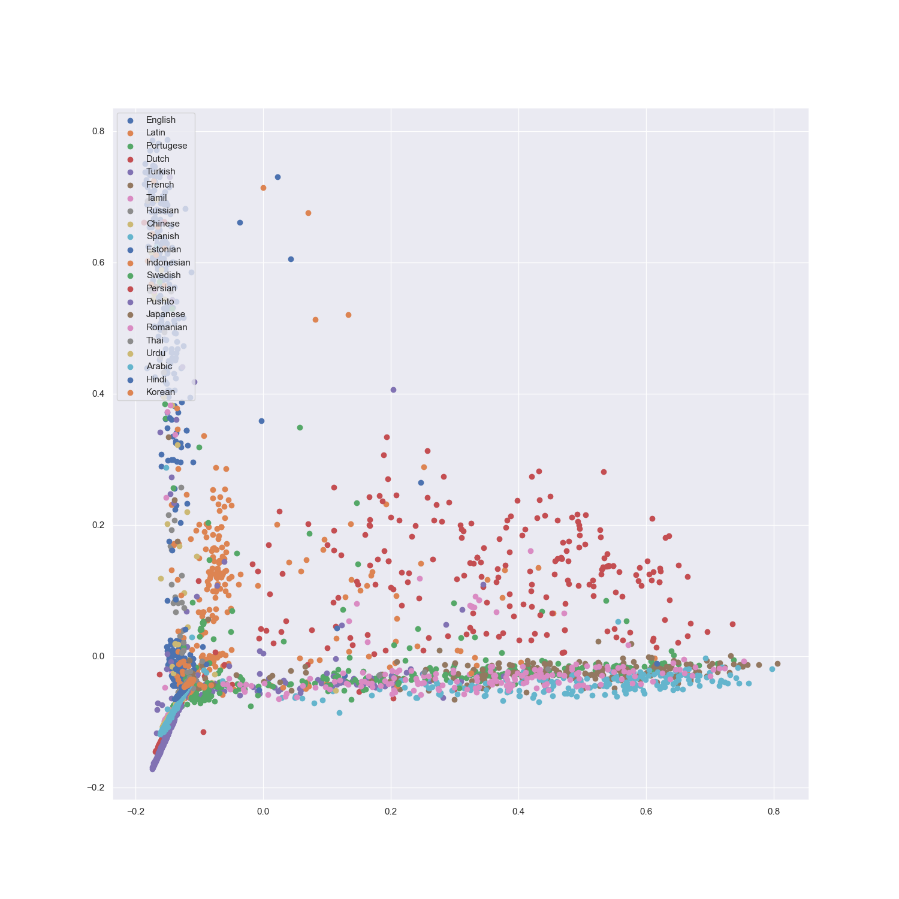


Figure 6: PCA Naive Bayes Char



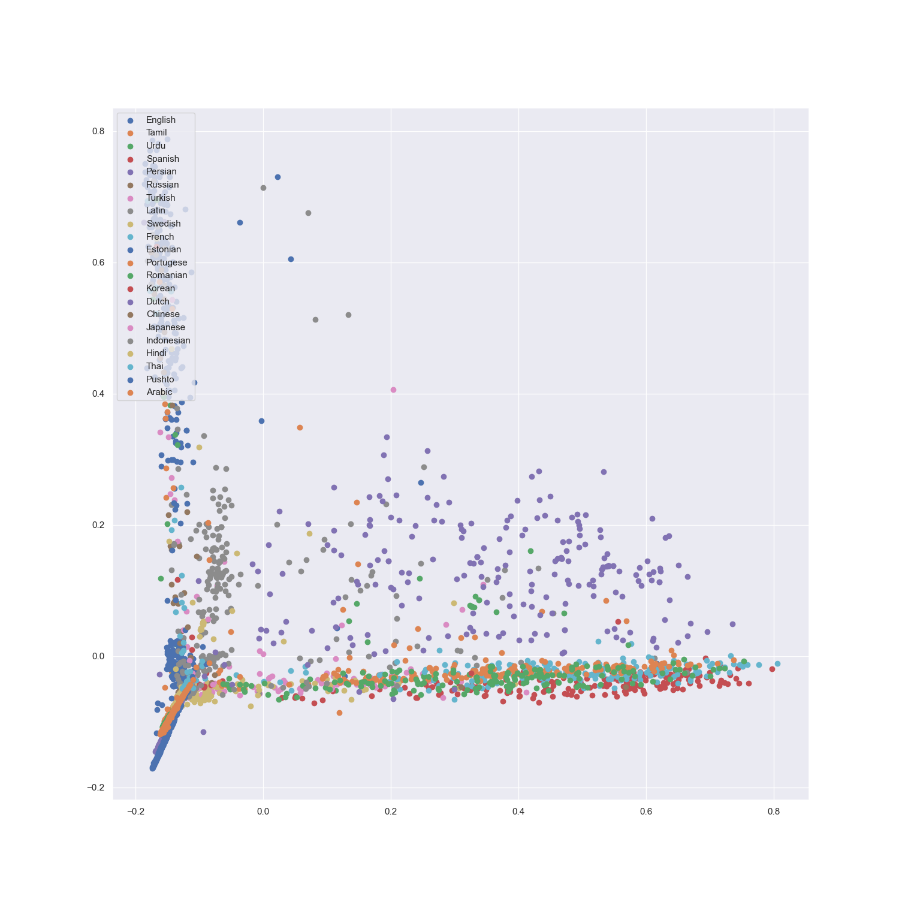
Code

Figure 6: PCA Random Forest Word

Figure 7: PCA Random Forest Char

Since we decided to present the required results together with the extension over models and variables, the document size may exceed the limit of pages. However, all the code (including the modification of the original script) is available in the [GitHub](https://github.com/norhther/MUD-project) repository, together with the results obtained through the experiments.