School of Computing & Communications | Lancaster University

MSc Data Science

A Simple Character Based n-gram Language Identification Approach

SCC413 — Applied Data Mining

Kieran Molloy

Lancaster University

13th December 2023

SIGNED DECLARATION

I certify that the material contained in this coursework is my own work and does not contain unreferenced or unacknowledged material. Regarding the electronically submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my coursework being placed in the public domain, with my name explicitly included as the author of the work.

Name : Kieran Molloy
Date : 13th December 2023

0.1 Introduction

A major challenge in computational linguistics is building models that enable natural language detection and therefore understanding. The precursor to understanding the meaning of text is to determine the language of the text. There are several previous attempts to classify text via word ngrams [1]. All analysis is python because of the authors previous understanding, as well as available libraries such as sklearn.

0.2 Algorithm

This paper outlines 3 strategies for language detection. (std) bigrams (rek1) rekishikon 3 language (rek2) rekishikon 55 language. Both methods allow for easy expansion to other languages, but rekishikon is able to compute a larger set of languages. Preprocessing removes unwanted characters, such as whitespace and punctuation. The dataset is split into train, and test sets in the ratio [9,1] (as per section 0.2).

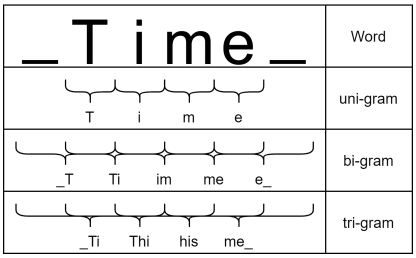
Character n-gram Profiles

The n-gram profiles are generated by iterating over each sentence in a document, and counting the character sequences (see Figure 1). Method (1) uses the algorithm shown in Listing 1, however rekishikon uses the total frequency instead of the log-probability as this was found to perform stronger when there are more than 3 language profiles. This concept can be used to generate any size n grams, but the balance between too specific and too general must be assessed on a per problem basis. Each language has specific language structure that is specific to itself, languages that are very different such as English and Igbo use different alphabets - so the distinction between them is relatively simple. Comparing English to Dutch, is far more complicated as the same alphabet is used, however the structure of words, and hence the language differs. This is represented in the number of appearances of specific letters, or sequences of letters.

Listing 1: log-probability n-gram profile generation algorithm

```
def generate_grams():
    preprocess()
    for i in words:
        if i is a gram:
            gram[i] += 1
    f = gram.total()
    for gram:
        if "_" : r = 2, else: r = 1
    key = log(key / r * f)
    return key
```

Figure 1: n-gram word breakdown into unigram, bigram, trigram



Language Detection

A language can be detected by calculating the distance between the test sample and any ngram profile. This is done using frequency observations, and determining how far out of place an n-gram is from its place in a different profile. Selecting the language with the lowest difference is the expected language. The similarity metric utilised in method (1) calculates a weighted distance between the language profile and sentence profile - it was found that the negative weighting of terms not present could be effective in specific scenarios of misidentification [2].

For language detection to be effective, a large array of languages must be supported, and it must fast. rekishikon is a python port of a java project titled language-detection [3]. It has been expanded to support further languages, as well as the addition of various utility functions.

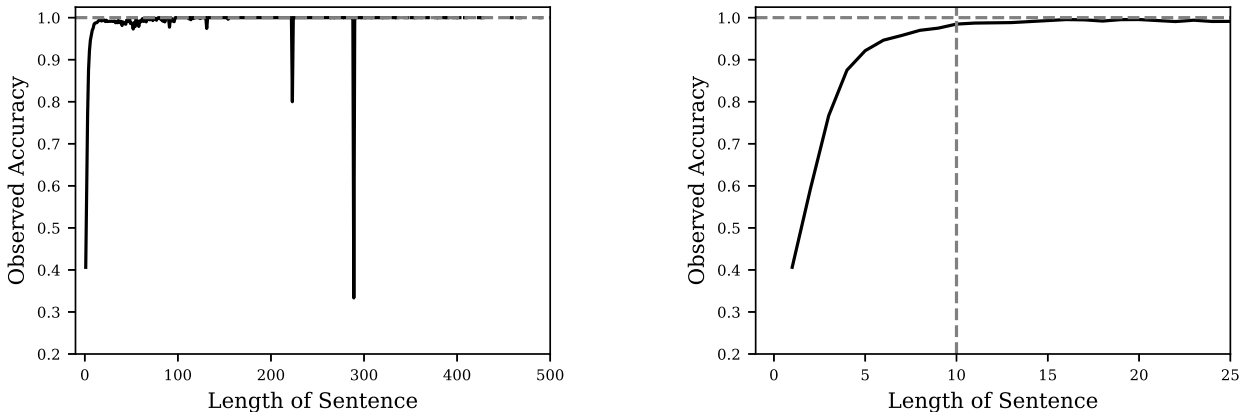
rekishikon uses naive bayes to classify languages, it does this by updating the posterior probabilities of categories by feature probabilities in each category. Formally, if X is the document, indexed by i, and C_k is the language, the classifier can be predicted using

$$P(C_k|X)^{(m+1)} = P(C_k|X)^m \times P(X_i|C_k) \tag{1}$$

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. It is trained on language profiles from [4].

Methods of Validation

The experiments use repeated stratified k-fold which returns approximately the same percentage of samples of each target class, this is to ensure the train and test sets are not imbalanced - a problem in early testing, especially with igbo. This is repeated 16 times with different randomisation seeds to ensure the whole set is tested.



(a) All n (b) Sentence size $0-25$

Figure 2: Accuracy of language prediction for size n

Sentence Exclusion

Defining two thresholds of 8 characters and 15 characters to remove sentences below this length - sentences below this length often do not contain the required information to establish a language or are not a traditional natural sentence.

0.3 Results and Evaluation

The first experiment tests overall accuracy of each method using the cross-validator, using a dataset with 103998 sentences, 08-rm removes 800 sentences, 15-rm removes 2000 sentences. The rough test-train split is [69332,34666], and each experiment is run 15 times (due to the cross-validator). Table 1 shows a clear trend of increasing accuracy and lower deviations as more low-quality sentences are removed. rek2 also sees lower accuracy and recall for most experiments, it however has the highest precision values with lowest deviation. Figure 2 demonstrates the correlation

		Accuracy (sd)	Precision (std)	Recall (std)	F1-Score (std)
	std	0.96914 (0.0008)	0.97251 (0.0007)	0.96914 (0.0008)	0.96981 (0.0008)
no-rm	rek1	0.97591 (0.0007)	0.97777 (0.0006)	0.97591 (0.0007)	0.97624 (0.0007)
	rek2	0.93972 (0.0009)	0.99085 (0.0003)	0.93972 (0.0009)	0.96420 (0.0006)
	std	0.97500 (0.0009)	0.97726 (0.0007)	0.97500 (0.0009)	0.97545 (0.0008)
08-rm	rek1	0.97817 (0.0006)	0.97973 (0.0005)	0.97817 (0.0006)	0.97844 (0.0005)
	rek2	0.94398 (0.0008)	0.99126 (0.0005)	0.94398 (0.0008)	0.96663 (0.0005)
	std	0.98307 (0.0009)	0.98422 (0.0008)	0.98307 (0.0009)	0.9833 (0.0009)
15-rm	rek1	0.98222 (0.0004)	0.98329 (0.0004)	0.98222 (0.0004)	0.98237 (0.0004)
	rek2	0.95328 (0.0011)	0.99252 (0.0005)	0.95328 (0.0011)	0.97207 (0.0008)

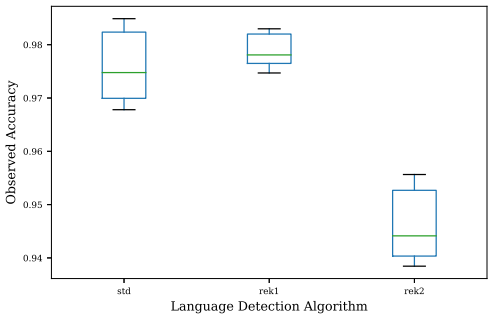
Table 1: Accuracy and weighted classification metrics

of sentence length and prediction accuracy, a good guess can be made with 5 with an accurate prediction after 10 characters. However when using languages of similar descent, such as German and English, this extends to around 50 as more sentence structure is required. There is a dip at around $n=230$ and 290, these are perhaps a single sentence, or series of non-sentences that cause problems. The boxplots in 3 demonstrate the increase of accuracy and reduction of variance by using rekishikon. Additionally this figure shows the increased accuracy by removing low quality sentences, with a clear upward trend from no-rm to 08-rm to 15-rm, it can be seen that variance decreases.

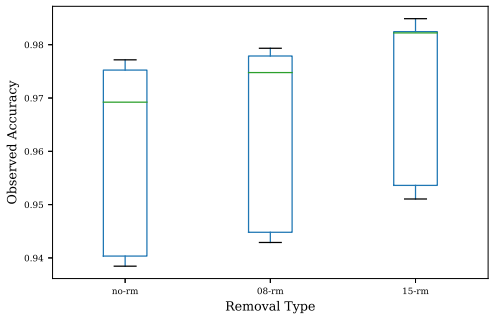
The second experiment looks at the required training data to confidently create a model, analysing the test-train split using Method 1 using split ratios from 0.05 to 0.95. The results are shown in Table 2 and the clear downward trend can be seen in Figure 4, it appears to be a linear relationship with 0.99 test size attaining an accuracy of 0.60, noticeably higher than pure random guessing (0.3). It is clear that more training creates a better model, it is clear that more training creates a better ngram profile.

0.4 Conclusions

The basic model gives strong results for the 3 languages, with misclassifications coming from non-sentences (such as numbers, or speech in other languages). This method is slow, and whilst it does provide room for extension to other



(a) Accuracy for each Method, showing sd of all runs

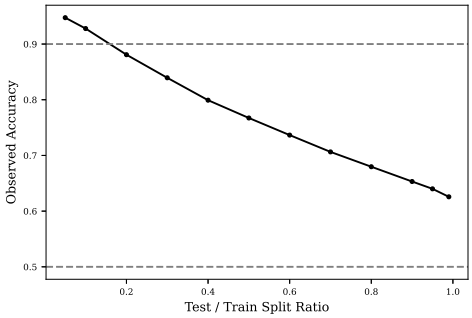


(b) Accuracy for each removal size, showing sd of all runs

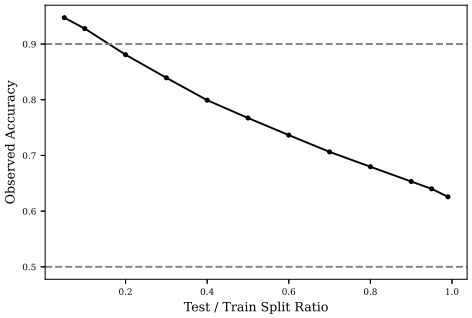
Figure 3: Accuracy Boxplots for varying method and removal type

Split Size	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95	0.99
Accuracy	0.94311	0.92307	0.87528	0.83270	0.79184	0.75966	0.72876	0.69824	0.67128	0.64341	0.62923	0.60852
Testing Time	00:11	00:24	00:52	01:27	01:54	02:35	03:07	03:46	04:37	05:24	05:36	05:30

Table 2: Summary of test-train split size



(a) Accuracy



(b) Precision

Figure 4: Summary of test-train split size graphically

languages, the computation time would make this infeasible. rekishikon alleviates this problem for 3 language model, but when more are introduced it makes some mistakes. The framework is language independent, thus it can work with any language (currently via ISO639-1). An extension to ISO639-3 would allow for experimentation with macro-languages and attempting to identify regionality within languages. Removing non-sentences can improve accuracy as they are unpredictable to train or test on. The second experiment demonstrates the need for large quantities of good training data to create an accurate language model, having less than 60k+ sentences to train a language on allows problematic sentences to have a larger impact, whilst using a test split larger than 0.2 leads to precision problems. An interesting extension would be to implement some form of graph theory to predict languages based on ngrams to both minimise overhead and increase prediction speed, this could alleviate the previous point on test size.

References

[1] X. Yang and W. Liang, 'An n-gram-and-wikipedia joint approach to natural language identification,' in *2010 4th International Universal Communication Symposium*, 2010, pp. 332–339.

[2] A. Pauls and D. Klein, 'Faster and smaller n-gram language models,' in *Proceedings of the 49th annual meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011, pp. 258–267.

[3] S. Nakatani, *Language detection library for java*, 2010.

[4] Wikimedia, *Wikimedia*, 2020.