

Language recognition implemented into machine learning algorithms*

Żaneta Pawelec^{1,*†}, Grzegorz Grochowski^{1,†} and Aleksandra Starowicz^{1,†}

¹Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND

Abstract

Language recognition algorithms play a pivotal role in various domains, offering applications ranging from automatically detecting the language of textual data to powering multilingual customer support systems. As the foundation of modern technologies like Artificial Intelligence, these algorithms enable content localization, facilitate language translation services, and drive personalized marketing strategies by analyzing linguistic patterns in customer feedback and social media interactions. This project compares five machine learning algorithms for language recognition, focusing on Bayesian classifiers and K-Nearest Neighbors (KNN). Through experimentation with different variations of these algorithms, including custom implementations, the project evaluates their effectiveness in recognizing 17 foreign languages. Methodologically, the project explores the nuances of each algorithm, discussing their underlying principles and implementation details. Experimental results reveal insights into the performance of each algorithm, providing valuable considerations for practical applications. Additionally, the project discusses the significance of precision, recall, F1-score, and accuracy metrics in assessing algorithm performance. Overall, this study contributes to advancing language recognition technology, offering valuable insights into algorithmic approaches and their real-world implications.

Keywords

language recognition, knn, clustering, artificial intelligence, Bayesian classifier, K-Nearest Neighbors

1. Introduction

Language recognition algorithms offer numerous applications across various domains [1, 2, 3, 4]. From automatically detecting the language of textual data to increasing performance of spam filtering and powering multilingual customer support systems. The importance of these algorithms enhances every day and becomes the crucial foundation for developing modern technologies such as Artificial Intelligence. Furthermore, they enable content localization, facilitate language translation services, and drive personalized marketing strategies [5] by analyzing linguistic patterns in customer feedback and social media interactions. We can easily spot them in our daily lives, using social media, web browsers and so on, that is why their accuracy and efficiency need to be constantly improved in order to make things easier. Moreover, language recognition algorithms underpin voice assistants and speech recognition systems, contributing to seamless user experiences. With their ability to discern linguistic nuances and patterns, language recognition algorithms continue to fuel innovation and efficiency across a wide array of real-life problems.


*IVUS2024: Information Society and University Studies 2024, May 17, Kaunas, Lithuania

^{1,*} Corresponding author

[†] These authors contributed equally.

✉ zp307912@polsl.pl (Z. Pawelec); gg307869@student.polsl.pl (G. Grochowski); @as307323.polsl.pl

 0009-0003-9824-7609 (Z. Pawelec); 0009-0009-8977-0442 (G. Grochowski); 0009-0003-0939-8748 (A. Starowicz)

 © 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Our program aims to compare five machine learning algorithms. All algorithms calculate the effectiveness of recognizing 17 foreign languages using different variations of the Bayesian classifier [6] and K-Nearest Neighbours classifier [7, 8]. The calculations are based on a longer or shorter sentence retrieved from a database.

To get a closer look into the applied classifiers, the following paragraphs will briefly describe them to illustrate how different these calculation methods are from each other.

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem, which calculates the probability of a certain class given a set of features. It assumes that the features are conditionally independent, hence "naive." It's widely used for classification tasks, especially in text classification and spam filtering.

K-Nearest Neighbors (KNN) is a non-parametric supervised learning algorithm used for classification and regression tasks. In KNN, the class of a new data point is determined by the majority class among its k nearest neighbors in the feature space. It's simple to implement and understand but can be computationally expensive for large datasets (like the one we are using), as it requires storing all training data and computing distances for each prediction.

Both algorithms have varying time consumption, with KNN being more computationally expensive due to its need to calculate distances for each prediction. Now, let's delve into a brief explanation of each of the applied algorithms and the underlying thought process behind their selection. The first classifier is the Bayesian classifier from the library, which provides the most effective results and thus serves as the main benchmark that we tried to achieve in the other algorithms. Next, we independently create a second Bayesian classifier aiming to mimic the version from the library. The third classifier is also a modified Bayesian classifier, determining the language by the probability of neighboring letters. In executing this algorithm, we assumed that each language has recurring sequences of letters that can enable assigning a given sentence to the language in which this sequence most commonly occurs. We derived an appropriate formula that allowed us to implement our idea into the program. The fourth classifier is a K-Nearest Neighbours from the library, but with implemented different distance calculation methods which we adjusted to our specific database. The fifth classifier is also the K-Nearest Neighbours algorithm but in this instance written by us. It was created following open-access models with an intent to achieve as high accuracy as the one from the imported KNN classifier. In order to achieve a satisfying outcome it required us to apply many adjustments in the distance calculating method. After performing the calculations, each algorithm displays a table with the results of the effectiveness of defining each language.

2. Methodology

Data from the set is divided into subsets X, containing texts in various languages, and Y containing the language classes of the texts from set X. The initial two Bayes classifiers and both KNN algorithms operate on a dataset converted into a matrix of token counts using the CountVectorizer class from the sklearn library. This is a one-dimensional matrix of the length of the dictionary containing all the words from the dataset. Each text sequence from set X is represented by such a matrix, where the words occurring in this sequence are represented by

the number of their occurrences in the appropriate matrix position and the rest are filled with zeros.

First, we used the MultinomialNB class contained in the sklearn library. For calculations, it uses the formula:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \quad (1)$$

where: θ_{yi} is the probability $P(x_i | y)$ of feature x_i appearing in a sample belonging to class y .

$N_{yi} = \sum_{x \in T} x_i$ is the count of occurrences of parameter i in class y in the training set, while

$N_y = \sum_{i=1}^n N_{yi}$ is the number of all parameters in set y . α is the smoothing prior, which in this case is Laplace smoothing - $\alpha = 1$. n is the number of classes in set Y .

Next, we attempted to replicate the function contained in the library, aiming to obtain similar results. However, in our version of the algorithm, we did not consider the smoothing parameter.

Algorithm 1: Method 'OwnMNB.fit' training the algorithm

Data: sets x_train and y_train

Result: None

```

1 classes := set of values of y_train,
2 tokens := empty dictionary;
3 foreach  $c \in \textit{classes}$  do
4    $x_c := x\_train \in c$ ,
5    $V\_Sum :=$  sum of vectors in  $x_c$ ;
6    $\textit{tokens}[c] := VSum /$  length of  $x_c$ 

```

Algorithm 2: Method 'OwnMNB.predict' performing calculations

Data: x_test

Result: list y_pred

```

1 y_pred := empty list;
2 foreach  $x\_row \in x\_test$  do
3   prob := empty dictionary;
4   foreach  $c \in \textit{classes}$  do
5      $\textit{prawl} :=$  vector sum of  $x\_row * \textit{tokens}[c]$ ;
6     Append prawl to prob[ $c$ ];
7   Append to y_pred class with the biggest value from dictionary prob,

```

In the third Bayes classifier, we changed the approach to the dataset. We utilized individual dependencies on the construction of each language - the probability of one letter occurring after another. The formula in this case takes the form:

$$\hat{\theta}_{yj} = \prod_{i=2}^n P(x_{j,i-1}x_{j,i} | y) \quad (2)$$

where: θ_{yj} is the probability $P(x_j | y)$ for x_j contained in the same class y_j . $P(x_{j-1}x_j | y)$ is the probability of the occurrence of letter x_j after x_{j-1} in class y_j . n is the number of letters in the considered text sequence.

This time, the methods are given raw training sets X and Y, and a test set X. The 'fit' method is responsible for creating a 'neighborhood table' of all the letters present in the training set X divided by language classes. They contain the probabilities of the occurrence of a given pair of letters one after the other. The 'predict' method for the test set determines membership in a class based on the probabilities from the 'neighborhood tables'.

Algorithm 3: Method 'LetterProb.fit' training the algorithm

```

Data: Sets  $x\_train$  and  $y\_train$ 
Result: None
1  $dictionaries :=$  empty dictionary;
2  $classes :=$  set of values of  $y\_train$ ;
3 foreach  $c \in classes$  do
4    $x_c := x\_train \in c$ ;
5    $letterCount := 0$ ;
6    $last = ''$ ;
7    $dictionaries[c] :=$  empty dictionary;
8   foreach  $row \in x_c$  do foreach
9      $letter \in row$  do
10       $letterCount += 1$ ;
11      if  $last + letter \in keys[dictionaries[c]]$  then
12         $dictionaries[c][last + letter] += 1$ ;
13      end
14      else
15         $dictionaries[c][last + letter] := 1$ ;
16      end
17       $last := letter$ ;
18    end
19  end
20  foreach  $z \in Keys dictionaries[c]$  do
21     $dictionaries[c][z] = dictionaries[c][z] / letterCount$ ;
22  end
23 end

```

In K-Nearest Neighbors from the library, we use scalar vector multiplication to calculate distances. We multiply this value by -1 to avoid the need to compute the k-farthest neighbors further.

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i \cdot b_i \quad (3)$$

where a,b are vectors

In our k-NN, we used the same formula for calculating distances as in the library algorithm, but additionally, we incorporated weighted computation of the k nearest neighbors.

3. Experiments

To compare the different performance parameters of the used algorithms, we utilized the metrics module from the sklearn library. To improve the accuracy of the results, each algorithm was

executed 10 times, and the final value is the average of all trials. The dataset containing texts in 17 languages with a total length of 10,337 records was divided into training and testing sets in a 70:30 ratio. For each algorithm, we compared parameters such as:

- precision - it is a measure that determines the ratio of correctly predicted class elements to all those marked as the given class

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4)$$

- recall - a measure informing how many elements from a given class were correctly recognized

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5)$$

- f1-score - it is the harmonic mean between precision and recall

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6)$$

- support- a measure of the occurrences of each class in the dataset
- accuracy - it is the ratio of correctly classified samples to all cases in the test set

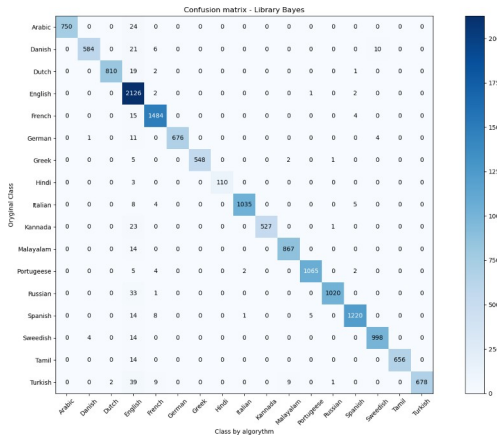
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (7)$$

Meaning of labels:

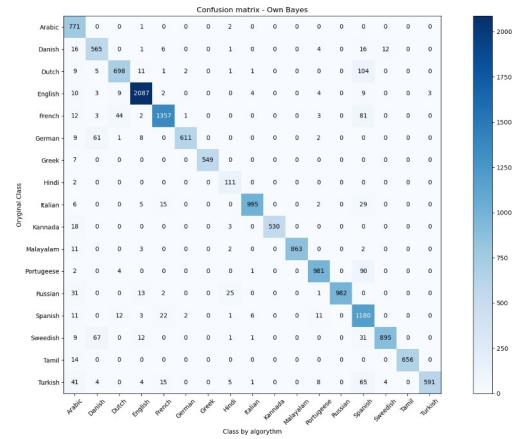
- TP - true positive - cases that were correctly classified as positive by the classifier
- TN - true negative - cases that were correctly classified as negative by the classifier
- FP - false positive - an error where the test result incorrectly indicates the presence of a condition when it is not present
- FN - false negative - an error where the test result incorrectly indicates the absence of a condition when it is actually present

3.1. The Bayesian algorithm from the sklearn library

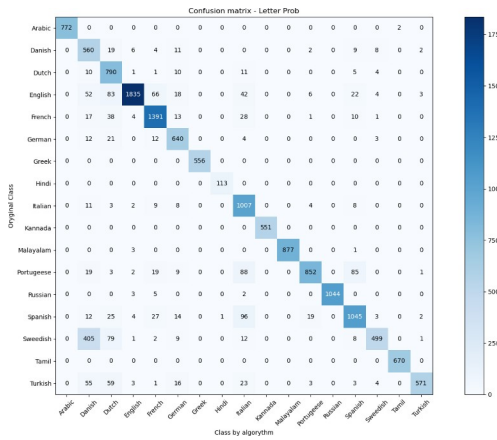
Analyzing the results shown in the above table, we can observe that the algorithm matches most languages with an accuracy ranging from 98-100% (see Tab. 1). The exception is the English language, which has an accuracy of only 89%, which may be due to the fact that English words are borrowed from other languages. The method for the entire dataset has an accuracy of 98%, making it the most accurate of all the solutions we have used (Fig. 1a).



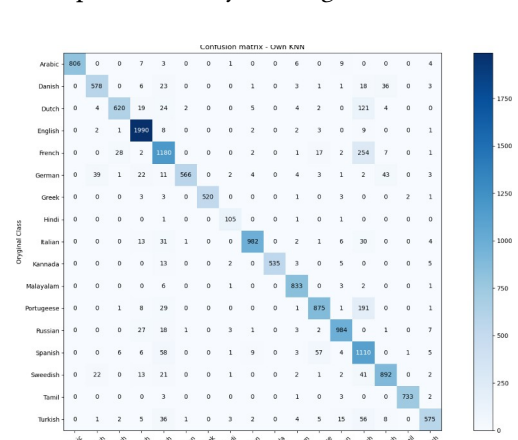
(a) The effectiveness results for the Bayesian algorithm from the sklearn library



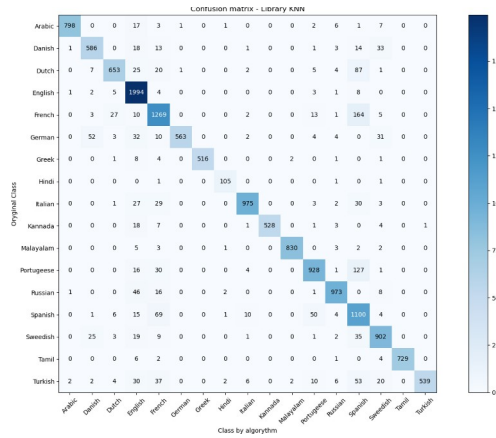
(b) The effectiveness results for the self-implemented Bayesian algorithm



(c) The effectiveness results for a custom-written Bayesian algorithm for letter proximity



(d) The effectiveness results for our k-nearest neighbors (kNN)



(e) The effectiveness results for k-nearest neighbors (kNN) from the library

Figure 1: Comparison of effectiveness results for different algorithms

Table 1

The effectiveness results for the Bayesian algorithm from the sklearn library

	precision	recall	f1-score	support
Arabic	1.0	0.97	0.98	774.0
Danish	0.99	0.94	0.97	621.0
Dutch	1.0	0.97	0.99	832.0
English	0.89	1.0	0.94	2131.0
French	0.98	0.99	0.98	1503.0
German	1.0	0.98	0.99	692.0
Greek	1.0	0.99	0.99	556.0
Hindi	1.0	0.97	0.99	113.0
Italian	1.0	0.98	0.99	1052.0
Kannada	1.0	0.96	0.98	551.0
Malayalam	0.99	0.98	0.99	881.0
Portuguese	0.99	0.99	0.99	1078.0
Russian	1.0	0.97	0.98	1054.0
Spanish	0.99	0.98	0.98	1248.0
Swedish	0.99	0.98	0.98	1016.0
Tamil	1.0	0.98	0.99	670.0
Turkish	1.0	0.92	0.96	738.0
accuracy			0.98	15510.0
macro avg	0.99	0.97	0.98	15510.0
weighted avg	0.98	0.98	0.98	15510.0

3.2. Self-implemented Bayesian algorithm

During the construction of this algorithm, our goal was to achieve results similar to the algorithm from the sklearn library. As observed, our algorithm performs worse with languages that use specific alphabets (e.g., Arabic, Hindi) and struggles more with recognizing languages belonging to the same family due to similarities in words stemming from the shared ancestry of these languages. This is particularly evident in Germanic languages: Dutch - German, Danish - Swedish, and Romance languages: Spanish, French, and Portuguese. However, the issues with languages using specific alphabets and the overall decrease in accuracy of other languages result from the lack of a smoothing parameter in the computational algorithm. Ultimately, though, in general terms, we achieved an algorithm accuracy of approximately 93%. It's the slowest among all algorithms but has average accuracy (see Tab. 2 and Fig. 1b).

3.3. Custom Bayesian algorithm for letter proximity

The algorithm, thanks to a completely different approach to the dataset, achieved results different from the rest. As the measurements show, unlike the previous one, it performs best with languages using specific alphabets. However, it struggles more with languages belonging to the same families. For example, with Germanic languages (Danish, Swedish, and Dutch) and some Romance languages (Italian, Spanish, and Portuguese). This is due to the similar structure of these languages associated with their common ancestry. If more than one language had

Table 2

The effectiveness results for the self-implemented Bayesian algorithm

	precision	recall	f1-score	support
Arabic	0.79	1.0	0.88	774.0
Danish	0.8	0.91	0.85	621.0
Dutch	0.91	0.84	0.87	832.0
English	0.97	0.98	0.98	2131.0
French	0.96	0.9	0.93	1503.0
German	0.99	0.88	0.93	692.0
Greek	1.0	0.99	0.99	556.0
Hindi	0.73	0.98	0.84	113.0
Italian	0.99	0.95	0.97	1052.0
Kannada	1.0	0.96	0.98	551.0
Malayalam	1.0	0.98	0.99	881.0
Portugeese	0.97	0.91	0.94	1078.0
Russian	1.0	0.93	0.96	1054.0
Spanish	0.73	0.95	0.83	1248.0
Swedish	0.98	0.88	0.93	1016.0
Tamil	1.0	0.98	0.99	670.0
Turkish	0.99	0.8	0.89	738.0
accuracy			0.93	15510.0
macro avg	0.93	0.93	0.93	15510.0
weighted avg	0.94	0.93	0.93	15510.0

the same probability (taking into account the rounding error of floating-point numbers), the algorithm chose the first one in alphabetical order, hence the lower accuracy of Danish compared to Dutch, and Dutch compared to Swedish. Similarly for Romance languages. Ultimately, this algorithm has the lowest overall accuracy of the tested trio, at around 89% (Tab. 3 and Fig. 1c). However, this result exceeded our initial expectations for the algorithm.

3.4. KNN algorithms

The first test we conducted for the KNN algorithm was to assess its effectiveness for different values of k ranging from 1 to 9. As shown in Tab. 4, the algorithm exhibited different effectiveness across the different values of k . Therefore, we choose $k=9$, for the algorithm from the library and $k=10$ for our algorithm. As we can also observe, for small values of k , our algorithm has higher effectiveness, which may be related to the use of a weighting table. As k increases, the difference in effectiveness decreases, until eventually, the algorithm from the library starts to exhibit greater effectiveness.

3.5. KNN without library

To shorten the execution time of the algorithm and increase its effectiveness from around 60% using the Euclidean metric, we decided to calculate the distance as the dot product of vectors. This allowed us to save some time and increase the effectiveness to 90%. The results

Table 3

The effectiveness results for a custom-written Bayesian algorithm for letter proximity

	precision	recall	f1-score	support
Arabic	1.0	1.0	1.0	774.0
Danish	0.49	0.9	0.63	621.0
Dutch	0.71	0.95	0.81	832.0
English	0.98	0.86	0.92	2131.0
French	0.91	0.93	0.92	1503.0
German	0.86	0.92	0.89	692.0
Greek	1.0	1.0	1.0	556.0
Hindi	0.99	1.0	1.0	113.0
Italian	0.77	0.96	0.85	1052.0
Kannada	1.0	1.0	1.0	551.0
Malayalam	1.0	1.0	1.0	881.0
Portuguese	0.96	0.79	0.87	1078.0
Russian	1.0	0.99	1.0	1054.0
Spanish	0.87	0.84	0.86	1248.0
Swedish	0.95	0.49	0.65	1016.0
Tamil	1.0	1.0	1.0	670.0
Turkish	0.98	0.77	0.87	738.0
accuracy			0.89	15510.0
macro avg	0.91	0.91	0.9	15510.0
weighted avg	0.91	0.89	0.89	15510.0

Table 4

Comparison for k-nn

k	Classification accuracy with library	Classification accuracy without library
1.0	0.8282	0.8301
2.0	0.8101	0.8301
3.0	0.8765	0.8756
4.0	0.8704	0.8765
5.0	0.8872	0.8852
6.0	0.8975	0.8926
7.0	0.9107	0.901
8.0	0.9175	0.9078
9.0	0.9246	0.9107
10.0	0.9239	0.912

indicate a strong performance of the algorithm across multiple languages. High precision and recall in languages like Arabic, Greek, Kannada, and Tamil show that the algorithm is particularly effective for these languages, achieving near-perfect scores. However, there are areas for improvement, notably in Spanish, which has a lower precision (0.61) and F1 score (0.72), indicating potential difficulties in accurately classifying this language (Tab. 5 and Fig. 1d).

Table 5

The effectiveness results for our k-nearest neighbors (kNN)

	precision	recall	f1-score	support
Arabic	1.0	0.96	0.98	836.0
Danish	0.89	0.86	0.88	670.0
Dutch	0.94	0.77	0.85	805.0
English	0.94	0.99	0.96	2018.0
French	0.8	0.79	0.8	1494.0
German	0.99	0.81	0.89	701.0
Greek	1.0	0.98	0.99	533.0
Hindi	0.88	0.97	0.93	108.0
Italian	0.97	0.92	0.95	1070.0
Kannada	1.0	0.95	0.97	563.0
Malayalam	0.95	0.98	0.97	846.0
Portuguese	0.9	0.79	0.84	1107.0
Russian	0.95	0.94	0.94	1047.0
Spanish	0.61	0.88	0.72	1260.0
Sweedish	0.9	0.89	0.9	997.0
Tamil	1.0	0.99	0.99	742.0
Turkish	0.93	0.81	0.87	713.0
accuracy			0.90	15510.0
macro avg	0.92	0.90	0.91	15510.0
weighted avg	0.91	0.90	0.90	15510.0

3.6. KNN with library

The algorithm from the library shows similar results for individual languages. Some of them achieved higher scores, while others had lower ones. However, the overall accuracy remained unchanged at 90%. The Spanish language, which our algorithm struggled with, still has a much weaker performance compared to the rest, but this result has slightly improved (Tab. 6 and Fig. 1e).

4. Conclusion

Based on our results, the Bayes algorithm from the sklearn library performs the best, achieving 98% accuracy. Our version of this algorithm ranks second with 93% accuracy. However, both KNN-based algorithms and our Bayes classifier based on letter pair probabilities performed the worst among all, still achieving relatively high scores of 90% and 89% accuracy, respectively. Although KNN algorithms handle language classification tasks well, their use in this form is not optimal in terms of both time or memory efficiency. Achieving results similar to our Bayes algorithms, they require almost two orders of magnitude more time. Similarly, in the case of the computational resources of the test platform, difference between both types of algorithms is significant. The KNN classifier from the library performs calculations faster than the one we created, thanks to the use of multi-threaded processing, while our KNN classifier performs calculations using only a single CPU core. However, this impacts memory usage. During tests,

Table 6

The effectiveness results for k-nearest neighbors (KNN) from the library

	precision	recall	f1-score	support
Arabic	0.99	0.95	0.97	836.0
Danish	0.86	0.87	0.87	670.0
Dutch	0.93	0.81	0.87	805.0
English	0.87	0.99	0.93	2018.0
French	0.83	0.85	0.84	1494.0
German	1.0	0.8	0.89	701.0
Greek	1.0	0.97	0.98	533.0
Hindi	0.94	0.97	0.95	108.0
Italian	0.97	0.91	0.94	1070.0
Kannada	1.0	0.94	0.97	563.0
Malayalam	1.0	0.98	0.99	846.0
Portuguese	0.91	0.84	0.87	1107.0
Russian	0.96	0.93	0.94	1047.0
Spanish	0.68	0.87	0.76	1260.0
Swedish	0.88	0.9	0.89	997.0
Tamil	1.0	0.98	0.99	742.0
Turkish	1.0	0.76	0.86	713.0
accuracy			0.90	15510.0
macro avg	0.93	0.90	0.91	15510.0
weighted avg	0.91	0.90	0.90	15510.0

Table 7

Comparison of algorithm runtimes

Test	Bayes library	Bayes without library	Letter Probability	k-nn library	k-nn without library
1	0.1017s	7.0200s	4.8464s	493.7335s	749.5594s
2	0.0717s	6.8577s	5.4612s	500.2711s	743.5833s
3	0.0552s	6.5957s	5.1900s	492.0898s	743.2426s
4	0.0529s	6.3185s	4.9687s	487.9027s	747.1023s
5	0.0500s	6.3520s	4.8404s	484.2074s	743.5345s
Mean	0.0663s	6.6287s	5.0613s	491.6409s	745.4044s
Accuracy	98%	93%	89%	90%	90%

the KNN from the library used over 9.5GB of available RAM on the test platform, while our KNN algorithm required approximately 5GB of memory. In contrast, the Bayes algorithms did not require more than 1GB of RAM and, despite running on a single CPU thread, did not fully load it. None of our developed algorithms came up close to 100%. One of the possible future improvements would be to combine together both our Bayes classifiers, to eliminate their separate weak points. An algorithm created this way would be much closer to 100% accuracy with only slightly lower time efficiency.

References

- [1] Y. Obi, K. S. Claudio, V. M. Budiman, S. Achmad, A. Kurniawan, Sign language recognition system for communicating to people with disabilities, *Procedia Computer Science* 216 (2023) 13–20.
- [2] D. Mengliev, V. Barakhnin, N. Abdurakhmonova, M. Eshkulov, Developing named entity recognition algorithms for uzbek: Dataset insights and implementation, *Data in Brief* (2024) 110413.
- [3] A. Vaitkevičius, M. Taroza, T. Blažauskas, R. Damaševičius, R. Maskeliūnas, M. Woźniak, Recognition of american sign language gestures in a virtual reality using leap motion, *Applied Sciences* 9 (2019) 445.
- [4] M. Nallakaruppan, G. Srivastava, T. R. Gadekallu, P. K. Reddy, S. Krishnan, D. Polap, Child tracking and prediction of violence on children in social media using natural language processing and machine learning, in: *International Conference on Artificial Intelligence and Soft Computing*, Springer, 2023, pp. 560–569.
- [5] G. Dash, C. Sharma, S. Sharma, Sustainable marketing and the role of social media: an experimental study using natural language processing (nlp), *Sustainability* 15 (2023) 5443.
- [6] P. Langley, W. Iba, K. Thompson, et al., An analysis of bayesian classifiers, in: *Aaai*, volume 90, Citeseer, 1992, pp. 223–228.
- [7] K. Prokop, Grey wolf optimizer combined with k-nn algorithm for clustering problem, in: *IVUS 2022: 27th International Conference on Information Technology*, 2022.
- [8] G. Guo, H. Wang, D. Bell, Y. Bi, K. Greer, Knn model-based approach in classification, in: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003*, Catania, Sicily, Italy, November 3-7, 2003. *Proceedings*, Springer, 2003, pp. 986–996.