Can twitter predict language? A computational approach to language classification

Kiram Ben Aleya[1]*, Kilian Sennrich[2]*

Abstract
Classifying Language is an essential component of almost every Natural Language Processing (NLP) pipeline. Working with mulitilingual data often requires Machine Translation (MT) as a preprocessing step. Some Libraries for MT in Python and R do not provide automatic language detection, which poses an issue if, exampli gratia, one works with a corpus containing documents in dozens of different languages, as appears in scraped twitter data. In this paper, we describe our findings of a machine learning approach to automatically detect languages. In particular we trained our models directly on twitter data, which might be an advantage over existing models when working with this type of data. This paper is not meant to be an academic paper, but rather an informal description of our project. We will outline our preprocessing, and detail our models. In particular we trained a Logistic Regression Model, which serves as a baseline. We compare this to our second model, which is of deep neural nature. We find that the Neural Net outperforms the baseline model by X%.

Keywords
NLP — Language Classification — Linear Classifier — Deep Neural Nets

[1] *Department of Computer Science, University of Zurich, Altstetten, Switzerland*
[2] *Department of Computer Science, University of Zurich, Unterentfelden, Switzerland*
***Corresponding authors**: kiram.benaleya@uzh.ch, killian.sennric@uzh.ch

Contents

1. Introduction

This is the our report for the first assignment of the Machine Learning for Natural Language Processing 1 course. The task was to train a linear, as well as a Deep Neural Net model on twitter data, which can predict the language of the tweet. The goal of this exercise is to:

- understand linear models and use them for multiclass classification tasks
- be able to implement different machine learning models, including MLPs, in scikit-learn
- understand the role of hyper-parameters, regularisation, and the problems of class imbalance.
- perform an error analysis of machine learning models.
- observe the most important features that lead to a prediction of a specific class and explains to some degree what the model does (XAI).

2. Task & Data Description

2.1 Data

For both parts of this exercise, we will work with the same data. The goal is to classify the language of twitter tweets. This is a challenging extension of the problem described in Goldberg, chapter 2. However, we will work with more languages than just six and the text segments we need to classify are much shorter. The material folder in the exercise section on OLAT contains the two files “train_dev_set.tsv” and “test_set.tsv”.

2.2 Tasks

Task 1 Scikit-learn is a useful Python library for all kinds of machine learning tasks. In the following, you will train several models in sklearn to solve this task. The aim is to become acquainted with a few different classifiers, as well as with the basic functionality of sklearn.

1. Create a suitable pipeline in sklearn to preprocess the data. Think about extending the feature space. What other features could you use to determine the language? Please include extra features linguistic features to your machine learning model for this task.
2. Train the following classifier: LogisticRegression
3. To find the optimal hyperparameter settings for the classifier, use sklearn's GridSearchCV. You are supposed to experiment with the following hyperparameters: Penalty (Regularisation), Solver, Experiment with parameters of the Vectoriser (not required, highly advised).

Report the hyperparameter combination for your best-performing model on the test set. What is the advantage of grid search cross-validation? Use a confusion matrix to do your error analysis and summarize your answers in your report. Now that you have your best model, it's time to dive deep into understanding how the model makes predictions. It is important that we can explain and visualize our models to improve task performance.

Explainable models help characterize model fairness, transparency, and outcomes. Let's try to understand what our best performing logistic regression classification model has learned. Generate a feature importance table for the top ten features (please have the features named) for the languages [‘en, ‘es’, ‘ja’]. What is more important, extra features or the outputs of the vectorizer, discuss. We recommend using the ELI5 library as it supports sklearn pipelines to explain the model weights. For more details, see their documentation on dealing with text classification. We will accept answers from any explanation library/method as long as the explanations for the model weights are provided in a structured/clear way.

Task 2 Let's see if you can beat the best linear model you've trained with sklearn with a non-linear MLP.

1. Train an MLP classifier. You can also use GridSearchCV, but be aware that training an MLP model takes much more time. Play around with 5 different sets of hyperparameters including layer sizes, activation functions, solvers, early stopping, vectorizer parameters, and report your best hyperparameter combination Do you achieve higher performance? Why/why not. Important: use the same data splits as for Part 1.

If you need help on that, Raschka's [2015] chapter 2 provides an introduction to MLPs. The Google Machine Learning Crash Course also offers good material.

3. Data Preprocessing

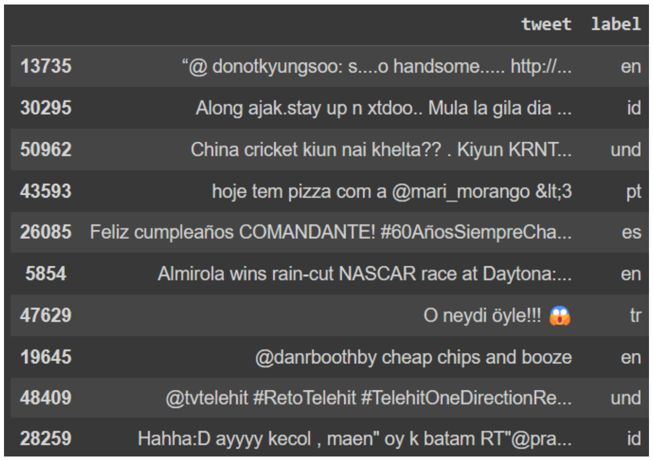
3.1 Data Cleaning

As described above, the data set we use is rather messy (Figure 1.). This is not of great surprise considering this is raw data scraped from Twitter. The data includes tweets in 69 languages. To properly train a model on this data, it must first be cleaned. For that purpose we made a Python object which contains the full data cleaning pipeline. The object works as follows:

1. First it cleans out all links from the data. This is done with a regular expression. Links are non-language-specific and likely lead to worse prediction accuracy when not cleaned out. Some tweets consist only of a link. After these have been deleted, the corresponding lines are empty or consist of just one or two spaces. Therefore they are not further considered in the data analysis and are dropped form the data set.
2. Second, Twitter usernames were removed from the dataset using a regular expression. Twitter usernames are defined here as a string starting with an @ followed by a random sequence of characters. Although Twitter usernames might contain some cues that are language-specific, they were not included in our models. This since it is likely that users of different nationalities are likely to use an English username. We call this phenomenon ”Tendency Towards Global Language” (TTGL).
3. Third, hashtags were removed from the dataset using a regular expression. Hashtags are defined here as a string starting with an # followed by a random sequence of characters. Similar to @, we assume that hashtags are strongly affected by TTGL.
4. Fourth, Unicode Normalization (UN) was applied. ”Programs should always compare canonical-equivalent Unicode strings as equal” (https://www.unicode.org/faq/normalization.html). This ensures better comparability of similar words/phrases utilizing different Unicode encodings. We used the ”unicodedata” library (https://docs.python.org/3/library/unicodedata.html) to perform UN.
5. Fifth, all Arabic numerals were removed from the data. Numerals are not of use for language classification because they are not language-specific, but are used uniformly in almost all languages.
6. Sixth, all special characters were deleted from the dataset. Although one might assume that users in some languages use more special characters (e.g., Spanish users use ! more often because their tweets are more emotionally affected), we found no concrete evidence for this and therefore deleted all of these characters from the dataset using a regular expression.
7. Finally, all Twitter posts were converted to lowercase. This step is common in NLP pipelines because models are supposed to treat lowercase and uppercase letters

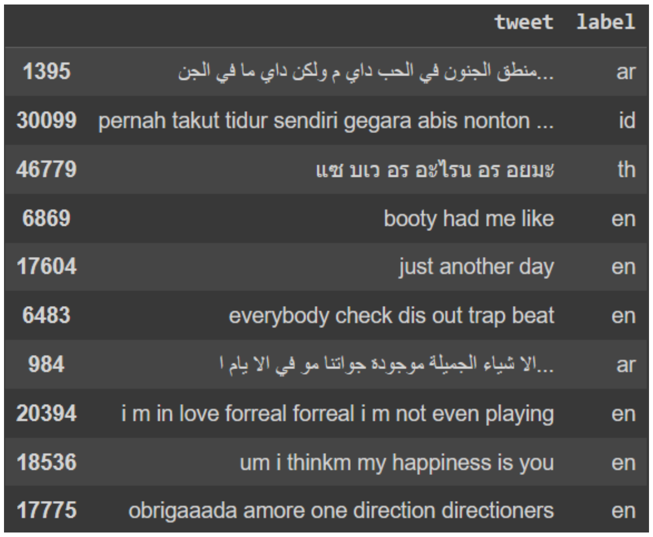
equally. Specifically, on Twitter, users sometimes capitalize their words to express certain sentiments. Since the purpose here is to classify speech and not to perform sentiment analysis, all data is lowercased. Of note, some languages are not case sensitive.

Examples of the cleaned data can be found in Figure 2.



	tweet	label
13735	“@ donotkyungsoo: s....o handsome..... http://...	en
30295	Along ajak.stay up n xtdoo.. Mula la gila dia ...	id
50962	China cricket kiun nai khelta?? . Kiyun KRNT...	und
43593	hoje tem pizza com a @mari_morango <3	pt
26085	Feliz cumpleaños COMANDANTE! #60AñosSiempreCha...	es
5854	Almirola wins rain-cut NASCAR race at Daytona:...	en
47629	O neydi öyle!!! 😱	tr
19645	@danrboothby cheap chips and booze	en
48409	@tvtelehit #RetoTelehit #TelehitOneDirectionRe...	und
28259	Hahha:D ayyyy kecol , maen" oy k batam RT"@pra...	id

Figure 1. Messy Data: Smileys, # and @ are not language-specific.



	tweet	label
1395	منطق الجنون في الحب داي م ولكن داي ما في الجن...	ar
30099	pernah takut tidur sendiri gegara abis nonton ...	id
46779	แซ บเว อร อะไรน อร อยมะ	th
6869	booty had me like	en
17604	just another day	en
6483	everybody check dis out trap beat	en
984	الا شياء الجميلة موجودة جواتنا مو في الا يام ا...	ar
20394	i m in love forreal forreal i m not even playing	en
18536	um i thinkm my happiness is you	en
17775	obrigaaada amore one direction directioners	en

Figure 2. Cleaned data: Data which was run through our custom data cleaning pipeline.

3.2 Data Augmentation

Looking at the distribution of labels in the data, it is apparent that some labels are severely underrepresented (see Figure 3). For example, the labels ”tn,” ”dv,” ”ta_LATN,” ”si,” and ”ko_LATN” are each found in only one of 52675 Twitter posts. To address this issue, it would probably be best to download more data with the underrepresented labels. However, assuming that our data set is a representative sample of all Twitter data, this would unfortunaltely not be computationally feasible. Suppose the training dataset has 52675 observations and some labels are represented only once. To get about 1000 examples of an underrepresented label, one would need to download $52675 * 1000 = 52,675,000$ tweets. Not to mention that one would probably find even rarer labels in the new data, which would then have to be considered as well.

Since it is not possible to download more data for the reasons mentioned above, Data Augmentation (DA) techniques can be used to artificially augment the underrepresented labels. The problem here is that many DA techniques for NLP require language-specific datasets. Exempli gratia, Back-Translation (BT) uses Deep Neural Nets trained on specific languages to translate a document into another language. The translated document is then back-translated into the corresponding original language. Since the languages of underrepresented labels are rarely spoken, it is unlikely to find such translators for all of them.

To address this, we have identified two augmentation techniques that work in a relatively language-independent manner. The two techniques are:

- Random Deletion (RD): RD randomly deletes one or more words from a given string.
- Random Swap (RS): RS randomly swaps two or more words in a given string.

It is evident that these two techniques are rather weak in oppose to BT. This further becomes clear when looking at the confusion matrix in Appendix 1 & 2. The underrepresented labels of the training data set were, if they were even contained in the test set, not accurately predicted. Further explanation can be found in the results section.

We defined that each label must appear at least a thousand times in the training dataset. To boost the underrepresented labels, we used the following procedure:

1. Identify all languages, where less than 1000 tweets exist in the training data set.
2. Identify the number N of up-samples that must be performed to obtain 1000 labels (e.g. The training data set contains 981 tweets in Albanian. –¿ 1000-981 = 19 tweets must be augmented).
3. Randomly choose N tweets in a given language and augment them.
4. Append these augmented tweets to the training data set.

This procedure was packed inside a Python Object to simplify the application. The augmented dataset contained 107627 tweets.

3.3 Feature Engineering

”Feature engineering is one of the most important steps in machine learning. It is the process of using domain knowledge

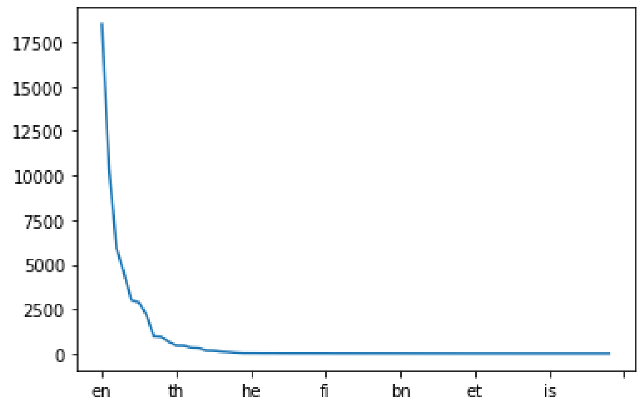


Figure 3. Label Distribution: Some of the languages are grossly underrepresented in the dataset.

of the data to create features that make machine learning algorithms work." (https://www.analyticsvidhya.com/blog/2021/04/a-guide-to-feature-engineering-in-nlp/). We vectorized the cleaned and augmented data set using sklearn CountVectorizer. We also included the Term Frequency-Inverse Document Frequency (TFIDF) metric so that our machine learning models could better detect the variations in meaning between different tweets. For this, we used sklearn TfidfTransformer.

Furthermore an additional feature was added: We hypothesized that the mean length of tweets of different languages might vary, as some languages have shorter overall mean word length than others. For that purpose we created our own sklearn class and added it to the training pipeline.

4. Model Training

In the following, we outline our machine learning setting, followed by remarks on the Baseline Model and the Deep Neural Net Model.

4.1 Cross-Validation and General Remarks on Model Training

Train-Test-Split The augmented training data set was split into a training and a validation data set. For that purpose we used the train_test_split function from sklearn. To keep the two data sets balanced, we shuffled the data before the split. We assigned 90% of the data to the training data set and 10% of the data to the validation data set. After the split, the training set contained 96864 observations and the validation set 10763 observations. By obvious reasons we did not put our hands on the test data set.

Cross-Validation We used Repeated K-Fold Cross Validation (RCV) to get robust estimates of the models accuracy. RCV repeatedly splits the training data into train and validation sets, estimates the model on the training set and evaluates the accuracy on the validation set. The "mean result [of the accuracy] is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error." (https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/).

Pipeline Training We used Pipeline from sklearn to train our models. This is helpful as we wanted to tune the hyper-parameters of the CountVectorizer as well. Note however that we did not add our custom preprocessing function to the pipeline (which could be done with FunctionTransformer() from sklearn). We believe that this would lead to overhead, since it has no hyper-parameters to estimate. By tuning the hyper-parameters of the pipeline, this function would be recomputed with every possible combination of the hyper-parameter grid. By leaving it out of the pipeline this function has a runtime of $O(1)$, by including it, it would have a runtime of $O(\prod_{i=1}^{n} m_{i,\theta})$, with $m_i \in \mathbb{N} > 0$ being the number of possible specifications of hyper-parameter θ_i.

Hyper-parameter Tuning For hyper-parameter-tuning we used GridSearchCV from sklearn. We allowed for the following parameters to vary:

Table 1. Table of possible hyper-parameters

Function	Parameter	Allowed
CountVectorizer	ngram_range	$(1,1),(1,2)$
CountVectorizer	ngram_range	$(1,3),(1,4)$
CountVectorizer	analyzer	$"word","char"$
CountVectorizer	analyzer	$"char_wb"$
LogisticRegression	penalty	$"l1","l2"$
LogisticRegression	penalty	$"elasticnet","none"$
LogisticRegression	solver	$"newton-cg","lbfgs"$
LogisticRegression	solver	$"liblinear","sag","saga"$

4.2 Logistic Regression

Logistic Regression was used to establish a baseline for the Deep Neural Net to beat. The implementation from sklearn automatically one-hot-encodes the criterion, such that we did not have to include this step in our pipeline.

What is logistic regression? - A quick explanation from IBM "This type of statistical model (also known as logit model) is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

$$Logit(\pi) = 1/(1+exp(-\pi)) \tag{1}$$

$$ln(\pi/(1-\pi)) = \beta_0 + \beta_1 * X_1 + \ldots + \beta_k * K_k \quad (2)$$

In this logistic regression equation, $logit(\pi)$ is the dependent or response variable and x is the independent variable. The beta parameter, or coefficient, in this model is commonly estimated via maximum likelihood estimation (MLE). This method tests different values of beta through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimate. Once the optimal coefficient (or coefficients if there is more than one independent variable) is found, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability. For binary classification, a probability less than .5 will predict 0 while a probability greater than 0 will predict 1. After the model has been computed, it's best practice to evaluate the how well the model predicts the dependent variable, which is called goodness of fit. The Hosmer–Lemeshow test is a popular method to assess model fit." (https://www.ibm.com/topics/logistic-regression)

4.3 Deep Neural Net

What is a Multilayered Perceptron? - A quick explanation from IBM "Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm. (https://www.ibm.com/cloud/learn/neural-networks)

5. Results and Discussion

The results section is structured into two parts. First we describe results from the baseline model, second we describe the results from the Deep Neural Net model.

5.1 Baseline Model

The following accuracy scored were observed from the baseline model:

Table 2. Accuracy Table

Data Set	Accuracy	Variance
Training Data Set	0.932885282457879	
Validation Data Set	0.920747003623525	
Test Data Set	0.8500510564763176	
Repeated K-Fold CV	-	-

The lower accuracy on the test data set in comparison to the training data set in Table 2 is an indication of overfitting. The more robust estimate of the RCV is closer to the estimate of the test data set. Overall, we conclude that the baseline model has some predictive power. That is, it would outperform a model that allocates all the data to the most represented label of the training dataset (in our case English (en) with 18508 observations).

The hyper-parameter tuning resulted in the following optimal parameter combination:

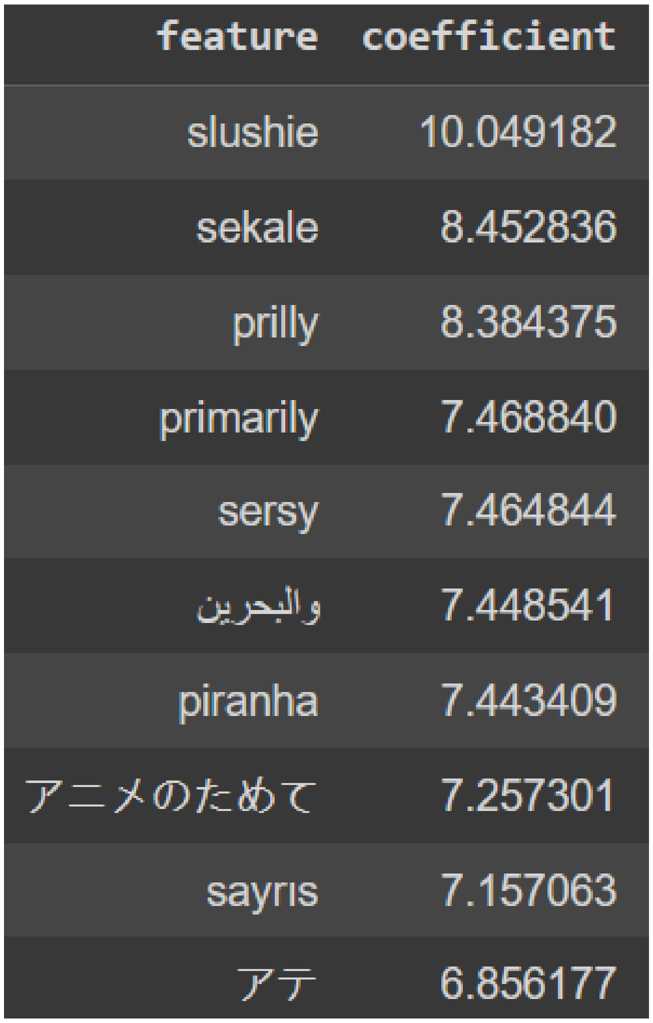
Table 3. Optimal Parameters According to Hyper-Parameter Tuning

Parameter Name	Selected Parameter
ngram_range	-
analyzer	-
penalty	-
solver	-

Why use grid search cross validation? Hyper-parameter optimization is a difficult task and can quickly lead to overfitting. That is, the parameter combination found to be optimal during grid search, is fitting very well to the training data, but not to the test data. It is therefore of great importance to get good, robust metric estimates when choosing a hyper-parameter combination. To achieve such robustness, cross validation on the hyper-parameter-tuner can be used. GridSearchCV from sklearn provides users with this functionality (argument: cv = X).

Performance Evaluation using the Confusion Matrix. From the confusion matrices in Appendix 1 & 2, computed on the test data set, it becomes evident that the model performed decent. Most of the classified labels are on the main diagonal, which indicates accurate prediction. There are also some labels that were predicted far to often. These can be identified as the vertical colored "lines" in Appendix 1 (e.g. labels "de" and "hi_latn"). The proximate explanation is that the model simply assigns tweets to these two labels when it cannot accurately assign them otherwise.

Performance Evaluation using the Classification Report. As can be interpreted from the classification report, the model performed exceptionally well on languages that come with its own set of ”letters” (signs). Exampli gratia, the label ”ar” (Arabian) was predicted with 0.99% accuracy and the label ”ru” (Russian), which was by the way subject to augmentation, was predicted with 0.99% accuracy. Furthermore, labels that are strongly represented in the training data set show good accuracy. For Example, English (en) was predicted with 0.95% accuracy and Spanish (es) was predicted with 0.90% accuracy. A closer look at the labels that were severely underrepresented, e.g., with only one observation in the training data, reveals that data augmentation worked very poorly in these cases. For example, in ko_LATN or ta_LATN, the one observation that was present in the test data was incorrectly predicted.



feature	coefficient
slushie	10.049182
sekale	8.452836
prilly	8.384375
primarily	7.468840
sersy	7.464844
والبحرين	7.448541
piranha	7.443409
アニメのためて	7.257301
sayrıs	7.157063
アテ	6.856177

Figure 4. Feature Importance: Displayed are the top ten most important features and their coefficients.

Most Important Features Figure 4 shows the ten most important features for the languages English (en), Spanish (es) and Japanese (ja). This table was extracted with the show_weights() function from the ELI5 library. For all of these languages our custom feature (i.e. the length of tweet metric) wasn’t among the top ten. We therefore conclude that the mean word length of a tweet is not as important for the classification of these three languages as the bag of words itself.

5.2 Deep Neural Net Model

6. Conclusion

7. Appendix

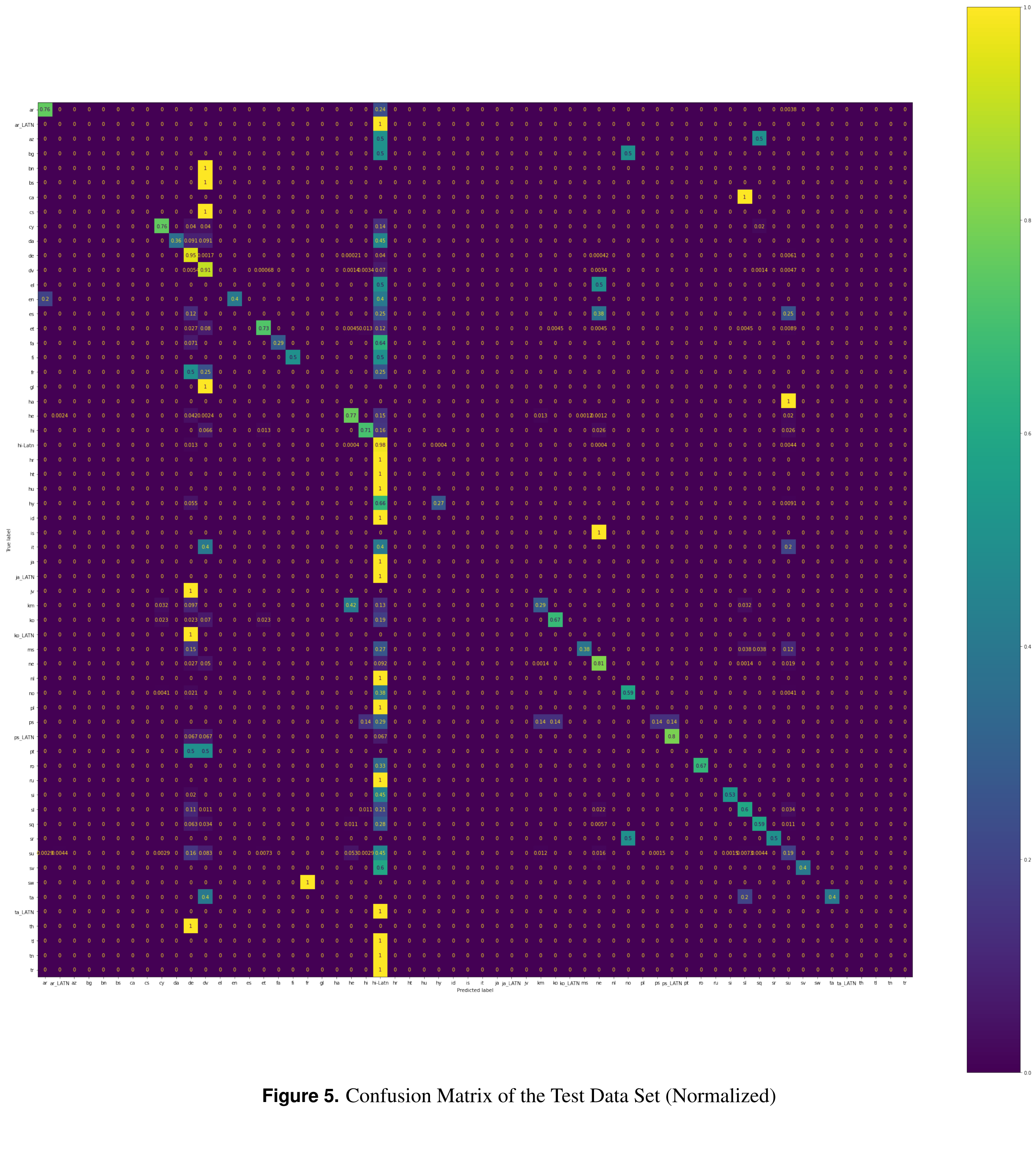


Figure 5. Confusion Matrix of the Test Data Set (Normalized)

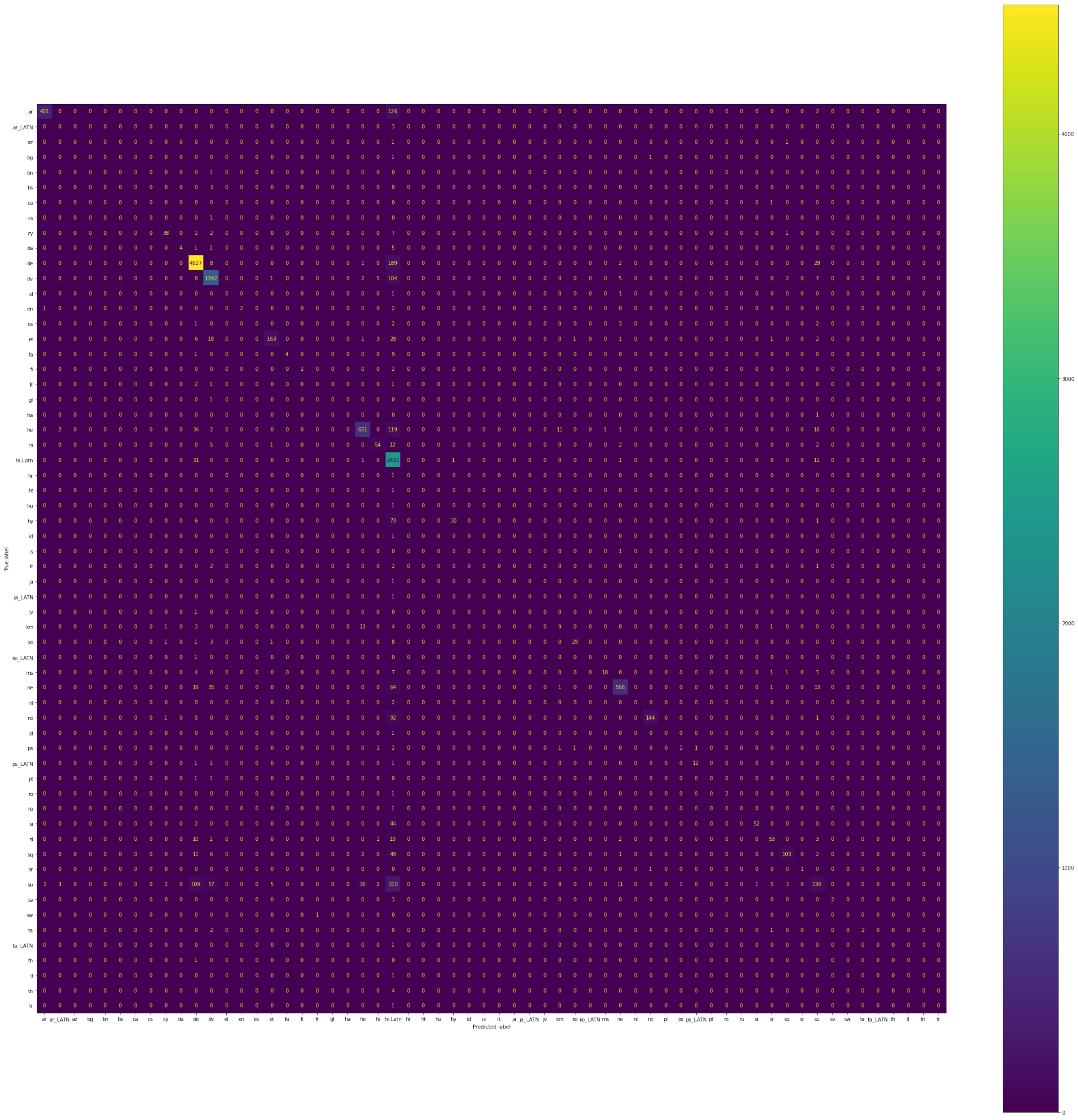


Figure 6. Confusion Matrix of the Test Data Set (Not Normalized)

Table 4. Classification Report for the Testing Data Set 1/2

Language	Precision	Recall	F1-score	Support
ar	0.99	0.76	0.86	529
ar_LATN	0.00	0.00	0.00	3
az	0.00	0.00	0.00	2
bg	0.00	0.00	0.00	2
bs	0.00	0.00	0.00	1
ca	0.00	0.00	0.00	3
cs	0.00	0.00	0.00	1
da	0.00	0.00	0.00	1
de	0.86	0.76	0.81	50
el	1.00	0.36	0.53	11
en	0.95	0.95	0.95	4756
es	0.90	0.91	0.90	1476
eu	0.00	0.00	0.00	2
fa	1.00	0.40	0.57	5
fi	0.00	0.00	0.00	8
fr	0.96	0.72	0.82	224
he	1.00	0.29	0.44	14
hi	1.00	0.75	0.86	4
hi-Latn	0.00	0.00	0.00	4
hr	0.00	0.00	0.00	1
ht	0.00	0.00	0.00	1
id	0.92	0.77	0.84	817
it	0.85	0.72	0.78	76
ja	0.64	0.98	0.78	2476
ja_LATN	0.00	0.00	0.00	1
jv	0.00	0.00	0.00	1
km	0.00	0.00	0.00	1
ko	0.96	0.23	0.37	110
ko_LATN	0.00	0.00	0.00	1
la	0.00	0.00	0.00	1
lv	0.00	0.00	0.00	5
mk	0.00	0.00	0.00	1
mn	0.00	0.00	0.00	1
mr	0.00	0.00	0.00	1
ms	0.30	0.29	0.30	31
nl	0.93	0.65	0.77	43
no	0.00	0.00	0.00	1
pl	0.85	0.42	0.56	26
pt	0.95	0.81	0.87	699
ro	0.00	0.00	0.00	2
ru	0.99	0.59	0.74	243
sk	0.00	0.00	0.00	1
sr	0.00	0.00	0.00	7
sv	0.92	0.80	0.86	15
sw	0.00	0.00	0.00	2
ta	1.00	0.67	0.80	3
ta_LATN	0.00	0.00	0.00	1

Table 5. Classification Report for the Testing Data Set 2/2

Language	Precision	Recall	F1-score	Support
th	0.98	0.52	0.68	98
tl	0.74	0.60	0.66	89
tr	0.93	0.57	0.71	174
uk	1.00	0.50	0.67	2
und	0.58	0.18	0.27	685
ur	1.00	0.40	0.57	5
ur_LATN	0.00	0.00	0.00	1
vi	1.00	0.60	0.75	5
xh	0.00	0.00	0.00	1
yo	0.00	0.00	0.00	1
zh-CN	0.00	0.00	0.00	1
zh-TW	0.00	0.00	0.00	4
zu	0.00	0.00	0.00	1
accuracy	**-**	**-**	**0.84**	**12731**
macro_avg	**0.40**	**0.27**	**0.31**	**12731**
weighted_avg	**0.85**	**0.84**	**0.83**	**12731**