Monolingual Written Natural Language Identification with Naïve Bayes and LSTMs

Jacob Lundberg
Linköping University
LiU-id: jaclu010
Email: jaclu010@student.liu.se

***Abstract*—This project aims to correctly classify the language of written text paragraphs. Given a paragraph from the WiLi dataset, one out of 235 languages can be classified. Chosen model solutions to the classification problem is a Naïve Bayes model and an LSTM. They reach a classification accuracy of 93.87% and 63.87% respectively. The Naïve Bayes model performs a lot better than the author of the WiLi dataset's neural network model, which achieves an accuracy of 88%.**

I. Introduction

The art of identifying the language in a text has been around a long time. It is something we humans do on a daily basis. The agent in question needs to understand what language is being read to interpret it correctly. It is an essential part of the pipeline of understanding text.

In this project, the Wi-Li dataset [1] is evaluated. The dataset in question contains phrases from Wikipedia in 235 different languages with a thousand (1000) paragraphs from each language.

As described by Thoma [1], a Multi-Layer Perceptron with character-based tf-idf features is implemented and achieves an overall accuracy of 88%. Also, many existing classifiers are tested against said dataset but only reach a prediction accuracy of 22-36%. The lousy performance is due to these classifiers only being implemented for a subset of languages existing in this dataset. The lack of existing solutions to both suggested classifiers by Thoma and other state-of-the-art models (to the WiLi dataset) makes for an exciting project. Thoma specifically states that models such as Naïve Bayes and recurrent neural networks (RNN) should be investigated further.

As mentioned by Mathur et. al. [2], Naive Bayes, SVM, n-gram, graph-based n-gram and, prediction partial matching (PPM) are models that have been used for the task of language identification. Neural networks, specifically long short-term memory (LSTM) models in conjunction with word embeddings is currently popular in various natural language processing (NLP) tasks [3], [4], [5]. Also, Mathur et. al. reports success in the use of RNNs for the task of language identification [2].

The idea of this project is to use a Naïve Bayes model as a baseline and then see if improvement is made with an advanced model, an LSTM neural network. The proposed LSTM consists of sequences of character n-grams.

II. Theory

This chapter will briefly describe the necessary theory, i.e. the theory behind the models used.

A. *tf-idf*

tf-idf is a common feature selection technique in NLP tasks. It is a short for *term frequency - inverse document frequency* and it is a model to evaluate how relevant a word is given a set of documents.

Given a collection of N documents, the frequency f of word i in document j helps define the *term frequency* tf as

$$tf_{i,j} = \frac{f_{i,j}}{\max_k f_{k,j}},$$

where $\max_k f_{k,j}$ denotes the word in the document with the highest frequency. This means that the word j is normalized on the maximum occurance of any word in the document.

Then, suppose word i occures in n_i out of the N documents. Then, the *inverse document frequency* is defined as.

$$idf_i = \log_2(\frac{N}{n_i}).$$

Finally, the two numbers are multiplied,

$$tf\text{-}idf = tf_{i,j} \times idf_i$$

and the word with the highest score usually carry most information about a document. [6]

B. *Multinomial Naïve Bayes*

A Naïve Bayes model is a derivation of a graph structure organized in such a way that there is assumed independence between all covariates.

Given a class C_k out of k possible classes and a word $x_i \in \boldsymbol{X}$, where $\boldsymbol{X}$ is the corpus, the probability of a specific class is given by

$$p(C_k, \boldsymbol{X}) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k).$$

This is derived from Baye's theorem and the independence assumption between all $x_i \in \boldsymbol{X}$. [7]

Given this formula, in the classification problem, one would have to compute the probability for each class and pick the class with the highest probability as a classification [8].

C. Word embeddings

Neural networks cannot on their own understand a text in string format. The strings need to be transformed into numbers or vectors. *fastText* is one such model and it is a part of a theory called *word embeddings*.

fastText produces embedding vectors by looking at each n-gram of a word with a size of 3-6 characters. Then, a vector representation is assigned to each n-gram. Finally, the sum of the n-gram vectors represents the final embedding for the word. [9]

An advantage of fastText compared with Google's *word2vec* [10] is that fastText can produce embeddings on words not yet seen in the data while training, otherwise known as out-of-vocabulary words. The advantage is limited to that at least some n-gram of a word were present during training. [9]

As described in the original paper, Mikolov et. al. [10] uses 1.6 billion words to train a 300 unit long word vector.

D. LSTM

An LSTM is one of many types of RNNs, specifically *gated RNNs*. It is based on an idea that there can be paths throughout a sequence of inputs. This idea is closely related to another feature within the field called *leaky units*. These paths carry information that may be accumulated through the sequence. As an opposite, it also means that some time information should be forgotten, which is also a feature within gated networks. [11]

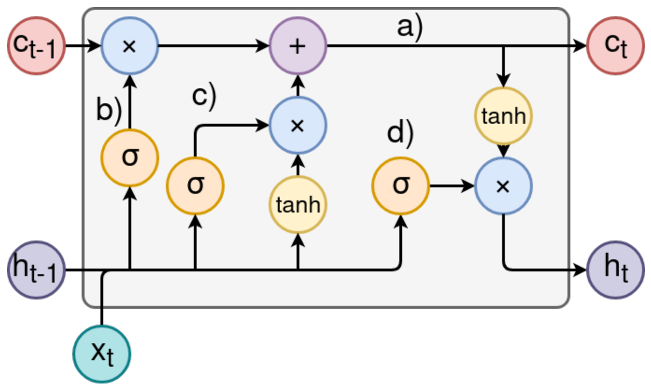

Fig. 1. Layout of an LSTM cell

The remainder of this section is a summary of previous work by Wang and Raj [12]. Figure 1 shows a layout of a simple LSTM cell. One cell is responsible for one of the sequential inputs.

1) States: States can be described as values passed between cells. In particular, the cell state c_t is the memory of the cell (*a*) in figure 1) and can be passed on to other cells and layers. The hidden state h_t can also be passed between cells and layers.

The input state is the linear combination of the previous hidden state and the input, called i_t:

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1}),$$

where W_{ab} are the corresponding weights and σ denotes the *sigmoid* function.

2) Gates: Gates in general uses the *sigmoid* function to prune information or let it pass through a certain spot, since the function assumes a value on $[0, 1]$ and the gate is multiplied to the corresponding spot.

The *forget gate*, f_t (*b*)), decides wether or not to keep the cell state from the previous time step. It is denoted with

$$f_t = \sigma(W_{fi}i_t).$$

The *input gate*, g_t (*c*)), decides if the input state should be fed into the cell state. It is denoted by

$$g_t = \sigma(W_{gi}i_t).$$

Finally, there is the *output gate*, o_t (*d*)), which decides if the current state is passed on as hidden state to the next cell,

$$o_t = \sigma(W_{oi}i_t).$$

These gates decide the flow of information, the final formula for the cell state is

$$c_t = g_t \odot i_t + f_t c_{t-1},$$

where $\odot$ denotes the element wise product. The hidden state is updated by

$$h_t = o_t \odot c_t.$$

A common issue with RNNs is the vanishing/exploding gradient problem [11]. As gradients are backpropagated through the network, they tend to vanish or explode. In a lot of places, 3 times in figure 1, values are multiplied. If the gradient then is continuously multipled with values smaller than one, the gradient will soon enough approach zero. The opposite is also the case when the gradient is continuously multiplied with a value larger than one.

III. Data

This chapter references Thoma [1]. As said, the dataset consists of 235 languages with text extracted from Wikipedia. Each language consists of 1000 paragraphs of text, making it a total of 235 000 paragraphs. The paragraphs were generated from random Wikipedia pages that consist of at least 140 characters in Unicode format. The dataset is already split into train and test data with 117 500 paragraphs each. Each of these sets are combined with labels of the language in pair with each paragraph. The dataset is available online[1].

A. Balancing

Even though each language contains a balanced amount of paragraphs, it does not mean that the paragraphs themselves are balanced. As an example, the shortest paragraph is 140 Unicode points long while the longest is 195 402 Unicode points long. The average length is 371 Unicode points.

[1]https://doi.org/10.5281/zenodo.841984

B. Issues in the dataset

Thoma presents 5 distinct error sources within the data. The most disturbing are presented below.

1) References: Sometimes, books or other authors are mentioned in the article in question and is often written in the original language, which often is English. A concrete example is the second paragraph from the training set:

Sebes, Joseph; Pereira Thomas (1961) (på eng). The Jesuits and the Sino-Russian treaty of Nerchinsk (1689): the diary of Thomas Pereira. Bibliotheca Instituti historici S. I., 99-0105377-3 ; 18. Rome. Libris 677492

The paragraph is from the Swedish Wikipedia, because of the string *(på eng)*, which makes this paragraph hard to classify correct.

2) Translation: Some Wikipedia pages have english translations. A notable example is from the Tonganese Wikipedia [2].

3) Quotes: Many articles contain quotes, which are often written in the original language. This error is closely related to Section III-B1.

IV. Method

This chapter describes the steps taken to produce results. All implementation is done in Jupiter notebooks[3], with Python 3[4] as programming language. The resulting implementation is available online[5].

A. Data collection

The dataset consists of four *.txt*-files. That is the train an test data along with their labels. On top of this, a sheet with more information regarding language shortenings was shipped with the data.

B. Pre-processing

The data consists of long paragraphs of text. As the data is not pre-processed in any way before, there still exists punctuation marks, spaces and other delimiters that may have an impact on the final performance. Therefore a regular expression is applied to each paragraph with the intent to remove all punctuation marks and to some extent correctly split words.

Cleaning the data was harder than expected since Chinese, for example, uses another Unicode character for a comma in comparison to English. This and other examples based on primarily [13], [14], [15] were added to the regular expression. This pre-processing was enough for the Naïve Bayes model.

1) LSTM pre-processing: For the LSTM model, a fastText word embedding was trained on the data. The fastText model used a vector size of 100 with a 10 iteration training procedure. The fastText model was created with the *Gensim*[6] implementation in Python. In total, 6 695 838 words were used to train the vector model.

[2]https://to.wikipedia.org/wiki/Sione_Tupou_Mateialona/en

[3]https://jupyter.org/

[4]https://python.org/

[5]https://github.com/jaclu010/text-mining

[6]https://radimrehurek.com/gensim/models/fasttext.html

Then, each word in a paragraph was concatenated with a blank space. After which all possible penta-grams of the paragraph in question were generated. Also, each paragraph was transformed into a sequence of 10 consecutive penta-grams. This made the dataset very large. To reduce the dataset, only 20% of the training set and 10% of the test set were used, chosen randomly with a uniform distribution. Also 10% of the training data was used as validation data.

C. Multinomial Naïve Bayes

This model was implemented with *sk-learn*'s[7] implementation of the Naïve Bayes model. First, a *count vectorizer* was applied onto the training data, which counts occurrences of each word creating a 2D matrix with documents and counts.

Two models were created. One which only uses the previously mentioned count matrix (further referenced as *NB-cv*) and the second which converts the count matrix to *tf-idf* scores before training (further referenced as *NB-tf-idf*).

The test set was then transformed into a count matrix which is used by both models to make predictions. The models kept an internal mapping to the language in question and were able to produce language labels. Thus, the data was ready for evaluation.

D. LSTM

The LSTM model was created with *Keras*[8] with *Tensorflow*[9] as backend.

The model consists of three layers. One LSTM layer with 100 LSTM units, a dropout layer with a dropout fraction of 0.4 and finally a softmax output layer with 235 neurons, one for each language. For each input, the index of the softmax neuron with the highest score is the resulting prediction.

E. Evaluation

Evaluation was done by calcuating accuracy, precision, recall and F1-score.

V. Results

This section presents the results obtained. They are shown as percentages in table I. A detailed confusion matrix for the NB-cv model can be found in Appendix A, figure 2 and for the LSTM model in figure 3.

TABLE I
Results for the classifiers, numbers in percent %

%	NB-cv	NB-tf-idf	LSTM
Accuracy	93.87	92.99	63.87
Macro precision	**95.89**	**95.25**	**69.27**
Micro precision	93.87	92.99	63.87
Recall	93.87	92.99	63.87
F1-score	94.39	93.52	59.34

In table I, regarding model NB-cv, it is noted that 2411 paragraphs are classified as English. The same phenomena

[7]https://scikit-learn.org/

[8]https://keras.io/

[9]https://www.tensorflow.org/

is noted for the LSTM model where 2216 paragraphs are classified as English. Thus, a lot of paragraphs are wrongly classified and manual inspection shows indication of errors mentionend in Sections III-B1, III-B2 and, III-B3.

The LSTM model performs a lot worse on all aspects. Also noted was the training set accuracy of 39.22% and validation set accuracy of 46.30%.

VI. Discussion

Naïve Bayes work extremely well in this setting. The NB-cv model (94% accuracy) is notably better than Thoma's 88% accuracy neural network. It is also notable that the model performs better when only the *count vectorized* data is used, compared to *tf-idf* transformed data. The performance increase can be explained by features that *tf-idf* intend to scale down is of some use. A frequently appearing sequence of characters in English is *the*. However, these three characters are also a good indication that the language in question *is* English; hence it might answer the improved performance. Also, some languages have unique characters that only their language use, resulting in small to none overlap regarding prediction probabilities. Mathur et al. [2] also concludes this fact.

The LSTM model did not perform as well as expected. A lot of the times, overfitting the training data was a common problem. Despite numerous attempts to prevent overfitting, it happened already on the second epoch. Another sign of a poor fit is the difference in training and test set accuracy. A poor fit may be the result of numerous errors sources; the size of the word embedding vector; the type of n-gram used; not enough dropout in the model; the type of optimizer and, the layout of the model. Not only do the results disappoint but it also proves the difficulty of the classification task. Also, since the results of the models differ a lot, no discussion is made concerning the different metrics used.

One solution that Mathur et al. [2] proposes and uses is an ensemble of RNN classifiers with different type of n-grams, which, in their case improve performance. There are multiple ways of improving the model in this project. As an example, only penta-grams were used as a feature. As so, the model can be extended to multiple n-grams or different types of n-grams.

As previously stated, the original *word2vec* uses 1.6 billion words to train a word vector. The model of this project uses a corpus that is 250 times smaller. At the same time, this relatively small corpus need to account for 235 languages. The small corpus is an issue that impacts performance, as it might introduce unwanted relationships between word vectors.

Some languages have no word boundaries and the fact that one character can represent multiple words, which means that a regular space might not always be a good split between words. All models used in this project is based on space delimited words, and it probably has some impact. However, the NB-cv model still performs very well. A way to tackle this issue is to use byte-grams, where an entire paragraph is split into n-grams taking all delimiters into account.

The dataset shows multiple occurrences were the paragraphs contain mostly English words. It leaves room for errors as even a human could have misunderstood the language of a paragraph. A good example of this is already mentioned in Section III-B1. Manual inspection of all paragraphs is a cumbersome project, but the error sources mentioned in Sections III-B1, III-B2, III-B3 and, described by Thoma [1] do contribute to a decrease in performance. However, in many circumstances today, multiple languages appear together. The added noise to the data proves that our models (at least NB-cv and NB-tf-idf) are very robust.

VII. Conclusion

A simple model such as Naïve Bayes works extremely well in this setting. At this time the NB-cv model is, to my knowledge, the best performing model on the WiLi dataset, with an accuracy of 93.87%.

The LSTM needs further development. Some are discussed in Section VI. This is an example when the time invested in developing an advanced neural network might not be worth it when the Naïve Bayes model performs extremely good.

References

[1] M. Thoma, "The wili benchmark dataset for written language identification," 2018.

[2] P. Mathur, A. Misra, and E. Budur, "Lide: Language identification from text documents." 2017. [Online]. Available: https://login.e.bibl.liu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1701.03682&site=eds-live&scope=site

[3] K. Jiang, S. Feng, Q. Song, R. A. Calix, M. Gupta, and G. R. Bernard, "Identifying tweets of personal health experience through word embedding and lstm neural network," *BMC Bioinformatics*, vol. 19, no. 8, p. 210, Jun 2018. [Online]. Available: https://doi.org/10.1186/s12859-018-2198-y

[4] C. Xu, L. Xie, and X. Xiao, "A bidirectional lstm approach with word embeddings for sentence boundary detection," *Journal of Signal Processing Systems*, vol. 90, no. 7, pp. 1063–1075, Jul 2018. [Online]. Available: https://doi.org/10.1007/s11265-017-1289-8

[5] J. Kim, G. Tur, A. Celikyilmaz, B. Cao, and Y. Wang, "Intent detection using semantically enriched word embeddings," in *2016 IEEE Spoken Language Technology Workshop (SLT)*, Dec 2016, pp. 414–419.

[6] A. Rajaraman and J. D. Ullman, *Data Mining*. Cambridge University Press, 2011, p. 1–17.

[7] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. [Online]. Available: http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html

[8] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[9] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," 2016.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[11] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016, unpublished, Book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[12] H. Wang and B. Raj, "On the origin of deep learning," 2017.

[13] Wikipedia. Chinese punctuation. Read: 2018-12-29. [Online]. Available: https://en.wikipedia.org/wiki/Chinese_punctuation

[14] ——. Lao language. Read: 2018-12-29. [Online]. Available: https://en.wikipedia.org/wiki/Lao_language

[15] ——. Tibetan alphabet. Read: 2018-12-29. [Online]. Available: https://en.wikipedia.org/wiki/Tibetan_alphabet

Appendix

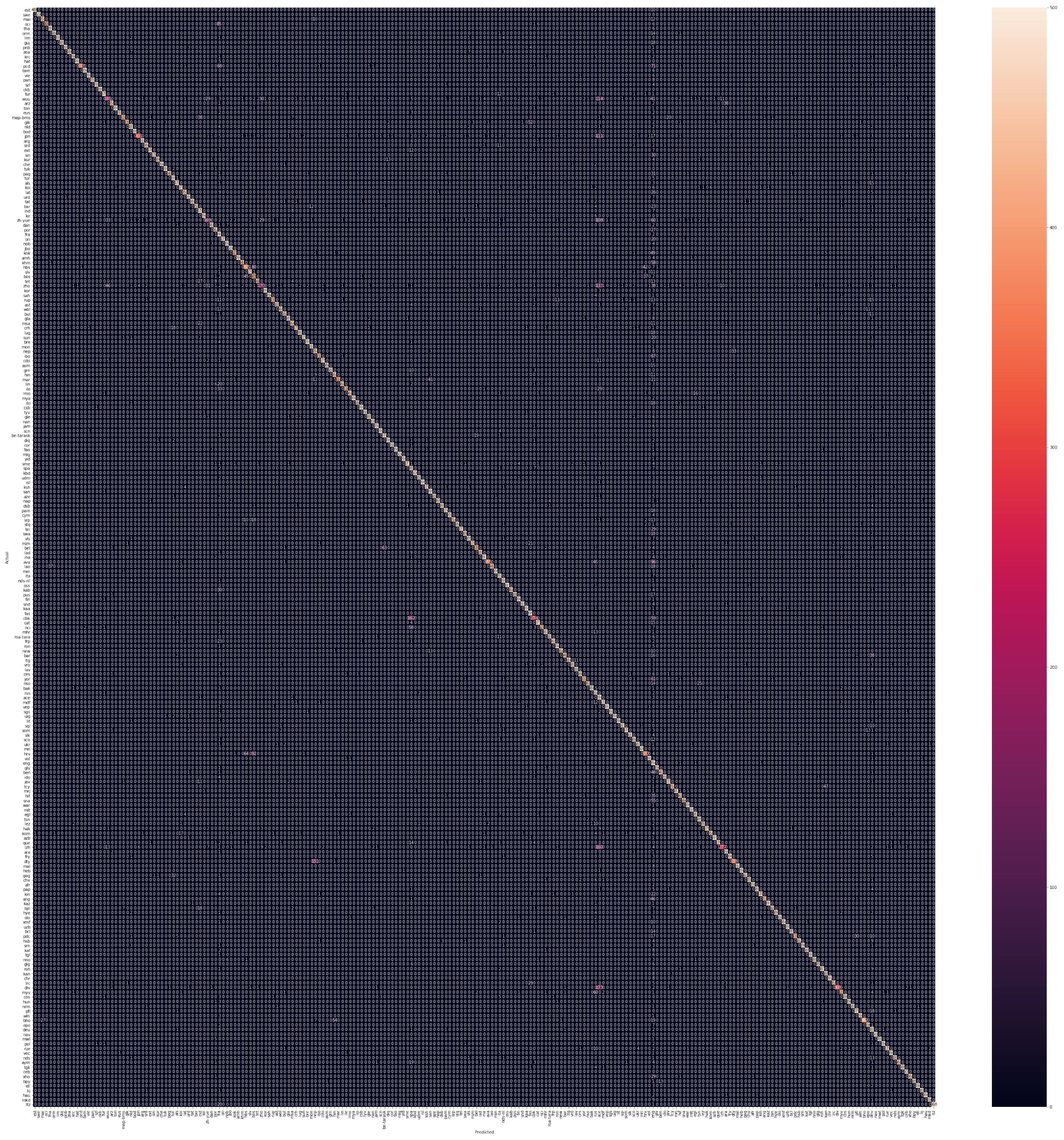
Fig. 2. Confusion matrix for the NB-cv model. The color bar shows prediction counts where black is zero and white is 500

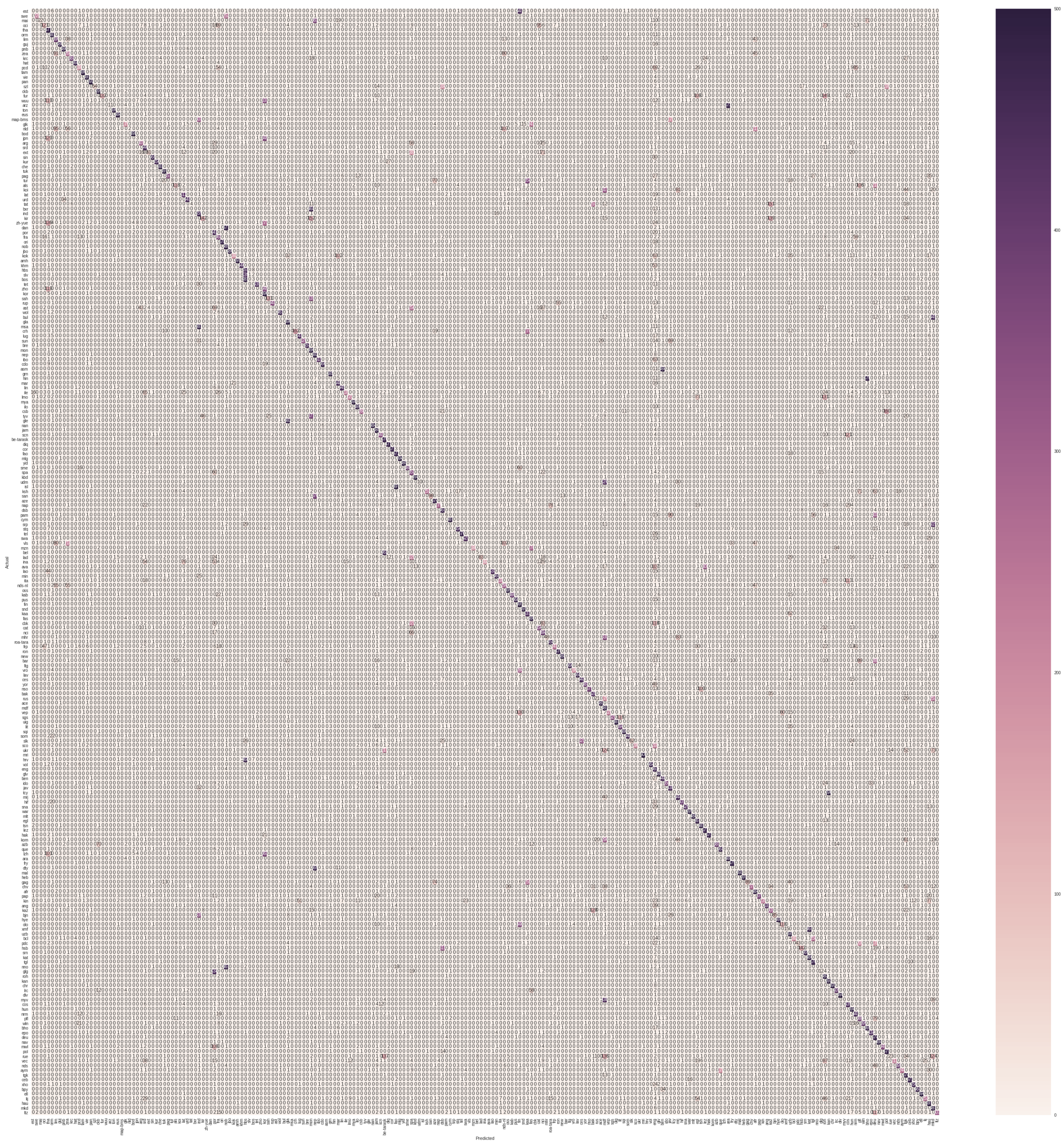

Fig. 3. Confusion matrix for the NB-cv model. The color bar shows prediction counts where white is zero and black is 500