Investigating optimal Machine Learning Techniques for the Detection of 4 Devanagari Languages in Roman script

Aditya Mehta ADITYA.1810018@DAIS.EDU.IN
IBDP, Year 12
Dhirubhai Ambani International School
Mumbai, MH 400098, India

Abstract

Due to diversity in languages in India and lack of support for Indic languages in digital and physical keyboards, a common phenomenon, especially in online modes of communication, is the utilization of the roman script for Indic languages. This form of transliteration is quite common. As such, identification of the root language which is being transliterated can have many potential uses in translation, messaging, and search systems. It is therefore necessary to develop a rapid, accurate, and light model for the purpose of this detection. This paper presents an exploration of various standard textual classification techniques to achieve such a model. The paper is focused on 4 Devanagari languages: Hindi, Gujrati, Mahrati and Sindhi. The machine learning models tested were a Multinomial Naïve-Bayes algorithm, along with a Recurrent Neural Network and a Convolutional Neural Network. The highest accuracy achieved was 97.3%.

Keywords: Natural Language Processing, Script Identification, RNN, LSTM, Transliteration

1. Introduction

In sociolinguistics, transliteration is the representation of words and phrases of one language by the alphabets, script or notations of another language, while keeping the original pronunciations intact, Chatterjee (2017). Transliteration is most common when documentation/ communication is limited to the script of a certain language, and thus expression in all other languages is forced to occur in that same script.

Roman or Latin script is the alphabetic writing system evolved from the Latin language, used for reading and writing predominantly in Western European languages. It is the most adopted writing system in the world, Chanda et al. (2010) Devanagari is used as writing and reading script, which is extensively spread over a wide belt of India. Devanagari is used to write many languages of India, such as, Sanskrit, Hindi, Marathi, Rajasthani, Sindhi, Prakrit, Konkani and Nepali, Ajmire and Warkhede (2018)

Despite the popularity of Devanagari script languages, the script itself has not penetrated significantly online, a20 (2019). Instead, majority of internet communication of Indic languages occurs in transliterated form. This can be attributed to two main reasons. Firstly, there are very few digital and physical keyboards which support Devanagari script, which makes typing messages, documents, e-mails, etc. difficult in the script in the absence of external provisions, Joshi et al. (2011). Moreover, since Devanagari is a phonetic language, there are far more unique alphabets (48) compared to roman script (only 26), Mhaiskar (2014). This makes even simple sentences more complex to write in Devanagri as compared to Roman script. As a result, Devanagari languages are often transliterated to roman script. Detection of the root form of the text can have many purposes, especially in the field of translation, text-to-speech and optical character recognition (OCR), where knowledge of the root language can allow for faster processing.

This paper investigates various approaches to accurately detect four Devanagari languages — Hindi, Gujrati, Mahrati and Sindhi. The models utilized include a simple Naïve-Bayes algorithm, a recurrent neural network (RNN) with a bidirectional Long short-term memory (LSTM), and a convolutional neural network (CNN).

The rest of the paper is structured as such: Section II reviews the existing detection techniques, Section III details the data used in the study, and Section IV details the proposed detection algorithms and text-processing. Finally, Section V reports the experimental testing results and a discussion of the same. The paper is concluded by Section VI.

2. Existing Techniques

Jacob et al. (2014) introduces an autonomous system for detection of Hindi, Malayalam, and Bengali all in romanized script using SVM regression. Adouane et al. (2016) presents a language automatic identifier for both Romanized Arabic and Romanized Berber including a wide range of Arabic dialects as well as the most popular Berber varieties through prediction by Partial Matching (PPM) and dictionary-based methods. The methods reach a macro-average F-Measure of 98.74% and 97.60% respectively. Sharma et al. (2018) suggests an approach to identify text-based content written in Roman script which conveys meaning in Hindi language. This is done through an artificial neural network(ANN) whose ouptput is fed through a autoencoder and a semi-supervised generative adversarial network(GAN). Bangalore (2014) presents N-grams model for romanized Kannada and Hindi with 80.4% accuracy.

3. Data Acquisition

For training, testing, and validation, the Dakshina Dataset was used proposed by Roark et al. (2020). This dataset is a collection of text in both Latin and native scripts for 12 South Asian languages. For each language, the dataset includes a large collection of native script text extracted from public source, a romanization lexicon which consists of words in the native script with attested romanizations, and some full sentence parallel data in both a native script of the language and the basic Latin alphabet. The dataset is summarized below in Table 1.

Thus, there are approximately 10,000 sentences for each language to be trained on, with an average of approximately 15 words per sentence, and a total of 164,755 words. For the combined dataset, the summary of the number of words in each sentence can be seen below in Figure 1.

As can be seen, for the chosen corpus, the sentences generally contain between 10-20 words, with an upper quartile of 20. This is significant for later, when the size of the tensors and neural network layers is chosen.

Table 1: summary of training dataset (Dakshina)

	Language			
Features	**Gujrati**	**Hindi**	**Marathi**	**Sindhi**
Total Size (Sentences)	9,981	9,744	9,969	9,868
Total Words	153,506	176,392	97,977	184,606
Unique Words	49,620	37,124	38,536	39,475
Average words / sentence	15.38	18.10	9.82	18.71

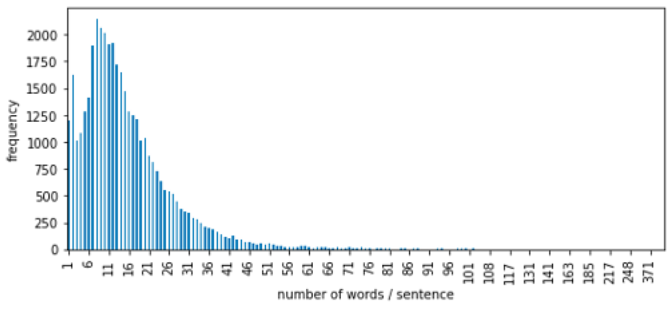

Figure 1: The distribution of words per sentence in chosen corpus

4. Model Selection and Training

Three models were trained on the dataset for the purposes of language classification. They were: Multinomial Naive Bayes, a Recurrent neural network, and a Convolutional neural network. The pre-processing, training, validation and testing of all 3 models was conducted on a system with 16 gigabytes of memory, with 8 cores clocked at 3228 MHz.

To normalize the data, the following steps of text preprocessing were carried out to the base corpus:

1. All non-Latin character words, numbers and punctuation were removed.
2. Excess white space was removed
3. All text was made lower-case

4.1 Multinomial Naïve-Bayes

The Multinomial Naïve Bayes model was chosen as the simplest model of the three used, relying on probabilistic measures as opposed to deep-learning, Pedregosa et al. (2011)

4.1.1 Pre-processing

Prior to training, model-specific pre-processing was required. First, the entire corpus was converted to a n-gram model, using a count-vectorizer, i.e. the model was trained with the vocabulary of the dataset and the text corpus was mapped to a numerical entry in the learnt vocabulary. For each such 'bag of n-gram' representation, the weightage of words was also accounted for using a term frequency(tf) -inverse document frequency (idf) transformation. The combined formula for tf-idf for a given word x is given as:

$$tf - idf = tf_x \times \log(\frac{N}{df_x}) \tag{1}$$

Where, tf_x is the frequency of the word x in the entire corpus, N is the total number of sentences and df_x is the number of sentences containing the word x.

4.1.2 Model Definition

After the data was pre-processed, it was fit to a probabilistic Multinomial Naïve Bayes model, defined by the following formula:

$$P(c|w_k\ ) = \frac{P(w_k|c)\quad \times P(C)}{P(w_k)} \tag{2}$$

$$P\,(c|W) = P(C) \prod_{1}^{k=n} \frac{P(w_k|c)}{P(w_k)} \tag{3}$$

Where, $P(c|w_k)$ gives the (posterior) probability that a word w_k falls in the category c , while in equation three this posterior probability is shown for a set of words W i.e. a set of words W consisting of features w_k for all $k \in\ Z,\ 1-n$ such that $P(c|W)$ is representative of $P\,(c|W) =\ P(c|w_1 \cap\ w_2 \cap\ w_{3....}w_n)$. The Multinomial Naïve Bayes model was tested on the metric of accuracy.

4.2 Recurrent neural network

A recurrent neural network (RNN) was trained on the dataset as well. The RNN allows for a bidirectional model, such that the output from step of the learning/ predicting process can be utilized as the the input for the next step, Sherstinsky (2018)

4.2.1 Pre-processing

To train the the RNN, the data was split into training and validation datasets (80:20), and pre-processed again. A tokenizer was fit on the training data with a vocabulary size of 90,000 to limit size of the network tensors while being near the number of unique words in the chosen corpus (164,000 unique words). The tokenizer was used on both training and validation data. Each sample (a sentence) was limited to a maximum of 25 words, around the upper quartile of the corpus, with smaller sentence being padded. As a result, embedding dimension of the layers of the neural network was taken as 25.

4.2.2 Network Layers

The RNN itself was trained with 4 layers as reflected in Figure 2. The first embedding/ input layer accepts tensor inputs of the shape equivalent to the vocabulary size set for the tokenizer and returns an output to the next bidirectional layer with an embedding dimension of 25. The Biderictional LSTM(Long short-term memory) layer allows for the recurrent nature of the model, capable of learning order dependence in sequence prediction, Hochreiter and Schmidhuber (1997). The ReLU (Rectified Linear Unit) activation layer normalizes all the node outputs from the LSTM. It outputs the input from the LSTM layer directly if it is positive. Else, it outputs zero. ReLU, unlike the sigmoid and tanh activation prevents saturation and vanishing gradient. Lastly, the softmax layer converts the layer output to probabilities for each class. The RNN was trained with sparse categorical crossentropy loss function, with adam optimizer, and accuracy as the testing metric. To summarize, the RNN model has four layers, the embedding layer (input layer), the LSTM, the ReLU and the softmax.

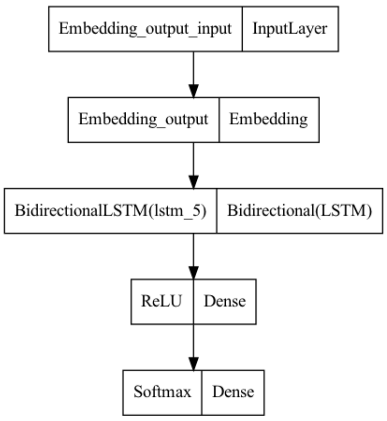

Figure 2: Layers of Recurrent Neural Network

4.2.3 Model Parameters

The model's hyper-parameters were tuned on the basis of analysis of text corpus as well as through the testing process. Embedding dimensions of each model were kept at 25 which was the upper quartile of words/ sentence in the dataset. The vocab size was kept as 90,000 words to ensure a good learning of the corpus (170,000 words) while preventing over-fitting. LSTM layer dropout was 0.2 to ensure optimal compromise between preventing model over-fitting and retaining model accuracy. The number of epochs was kept at 10.

The loss function chosen was sparse categorical crossentropy, to account for the multiple language classes which are represented by integers (0,1,2,3 respectively) as opposed to one-hot encoded vectors. One advantage of using sparse categorical cross entropy is time saved in memory as well as computation due to use of a single integer for a class, rather than a whole vector. The RNN was trained with Adam optimizer since it has little memory requirements and is efficient with the sparse text and class vectors. The training and testing metric was accuracy. All model parameters are summarized in the table below:

4.3 Convolutional neural network

A convolutional neural network (CNN) was trained on the dataset as well. CNN is usually not preferred over RNN for text applications due to ability of RNN to interpret the reversability of data/ corpus. However, the advantages of CNN such as higher feature com-

Table 2: RNN Hyper-parameters

Hyper-parameter	**Selection/ Value**
Embedding Dimensions	25
Vocabulary Size	90,000 words
LSTM Dropout	0.2
Epochs	10
Loss Function	Sparse Categorical Crossentropy
Optimizer	Adam
Metric	Accuracy

patibility, and feed-forward processing may present higher accuracies in case of language detection.

4.3.1 Pre-processing

The pre-processing is the same for the CNN as for the RNN. Seed was also kept constant to ensure the training points were constant. Embedding dimmensions and vocabulary size were also kept the same.

4.3.2 Network Layers

The CNN consisted of 5 layers, which are summarized in Figure 3. The first layer of the CNN was kept same— an embedding/ input layer for the following convolution. The convolutional layer used creates a convolution kernel that is convolved with the layer input over a single dimension to produce a tensor of outputs, Ghosh et al. (2020). The subsequent pooling layer progressively reduces the spatial size of the input image, so that number of computations in the network are reduced. The last two layers of normalization—ReLU and Softmax— are also kept constant.

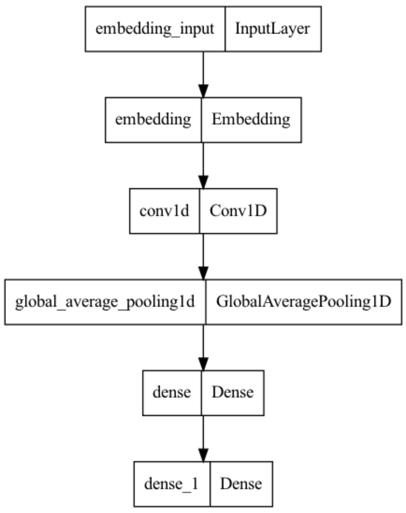
embedding_input | InputLayer
embedding | Embedding
conv1d | Conv1D
global_average_pooling1d | GlobalAveragePooling1D
dense | Dense
dense_1 | Dense

Figure 3: Layers of Convolutional Neural Network

4.3.3 Model Hyper pramaters

The CNN's hyper-parameters were largely same. Embedding dimensions of each model were kept at 25 and the vocab size was kept as 90,000 words. LSTM layer dropout was 0.2 to ensure optimal compromise between preventing model over-fitting and retaining model accuracy. The number of epochs was kept at 20. The default activation of the convolutioned layer was 'Relu'.The CNN was compiled on the same loss function with sparse categorical crossentropy, with the same Adam optimizer, and same testing metric of accuracy.

Table 3: CNN Hyper-parameters

Hyper-parameter	**Selection/ Value**
Embedding Dimensions	25
Vocabulary Size	90,000 words
Activation	Relu
Epochs	20
Loss Function	Sparse Categorical Crossentropy
Optimizer	Adam
Metric	Accuracy

5. Results

5.1 Training and Validation Results

The training of the models was done on a subset of 5000 samples from the Dakshinayan Dataset, with a 80:20 split between validation and testing.

The training of the Multinomial Naïve-Bayes was fastest, with a training time of less than 1 second. The RNN on the other hand took longest to train due to the large number of parameters to train each layer on. The training time over 20 epochs was 15.7 minutes or 942 seconds (at 47.1s per epoch). The training accuracy was 99.6%. In contrast, the CNN was trained much faster since its feed-forward. The model was trained in 1 minute 39 seconds for 10 epochs at a rate of 9.9 seconds per epoch. The CNN achieved a training accuracy of 97.3%. The training results for all three models are summarized below.

Table 4: Training Results

	Metrics		
Model	**Training Time**	**Epochs**	**Training Accuracy**
Naïve-Bayes	<1s	-	-
RNN	15.7 minutes	20	99.6%
CNN	1 minute 39 seconds	10	97.4%

5.2 Testing Results

All three models were tested on the same subsection of 5000 sentences from the Dakshinayan Dataset. The accuracy and inference time for each model is summarized below:

Table 5: Testing Results

	Metrics	
Model	**Accuracy**	**Inference (per second)**
Naïve-Bayes	94.2%	333.33
RNN	97.3%	1.20
CNN	95.8%	2.74

5.3 Benchmarking

The model performance was benchmarked with the method presented by Palakodety and Khudabukhsh (2020) to test the relative accuracy on the same dataset. The benchmarking model was tested for the Hindi language on the same testing dataset as models presented in this paper. The model yielded an accuracy of 89.8%. Though this may be indicative of higher accuracy in presented model, it may also stem from a difference in source of training dataset, since transliteration is not a uniform process. The model proposed by Palakodety and Khudabukhsh (2020) was trained on social media data, which may include different transliteration features than the Wikipedia sourced dataset used for this paper.

6. Conclusion

The paper presents 3 models for the purpose of language detection each with different accuracy as well as training and inference time. The analysis of the limitation and future uses of each model is given below.

6.1 Limitations

The primary limitations of all three models are:

1. Vocabulary learnt from solely one source of data (Dakhshinayan, which is extracted from Wikipedia). As a result, in case of encountering different styles of romanization of same words, models may not be effective.

2. These models are limited in training to the Devanagari script and as a result future transfer of weights to other languages may be less accurate

3. The dataset may also be a limiting factor in terms of the vocabulary size. Since model is effectively trained on 90,000/4 = 22,500 words per language, while each language actually contains over 100,000 unique words, it may not know sufficient words for some real world uses (for example, highly technical language).

There are a few model specific limitations as well. Naïve-Bayes being probabilistic will simply chose class with most features in training data set when encountering a sentence with insufficient known words in a sentence. RNN having high training and low inference time might be limited to cloud computing as opposed to edge-deployment. CNN has lower inference and training time and inference but comprimises in accuracy.

6.2 Future Scope

The three models have unique use cases. Naïve-Bayes, due to high inference time can be used for real time detection purposes, such as in live translation or speech-to-text. RNN due to higher accuracy can be used in more streamlined tasks including OCR, messaging services, and search-optimization. CNN can be use interchangably with the two based on situation-specific needs.

Further research can be used to expand on the work presented in this paper. Primarily, expanding the model architectures to more languages, while mantaining similair levels of accuracy and inference time. A hybrid RNN-CNN is also suggested which can use less memory due to feed-forward layers and pooling (thus having higher inference time) while mantaining bidirectionality for higher accuracy.

Acknowledgments

We would like to acknowledge support for this project from Reverie Language Technologies.

References

India's digital future: Mass of niches - kpmg india, 08 2019. URL `https://home.kpmg/in/en/home/insights/2019/08/india-media-entertainment-report_2019.html`.

Wafia Adouane, Nasredine Semmar, and Richard Johansson. Romanized arabic and berber detection using prediction by partial matching and dictionary methods. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–7, 2016. doi: 10.1109/AICCSA.2016.7945668.

Prafulla Ajmire and Shankar Warkhede. An analytical study of devanagari script recognition. 12 2018.

Supriya Anand Bangalore. Language identification for transliterated forms of indian language queries. 2014.

Sukalpa Chanda, Umapada Pal, Katrin Franke, and Fumitaka Kimura. Script identification a han and roman script perspective. pages 2708–2711, 08 2010. doi: 10.1109/ICPR.2010.1127.

Amitabha Chatterjee. Chapter x - standards for iod activities. In Amitabha Chatterjee, editor, *Elements of Information Organization and Dissemination*, pages 483–495. Chandos Publishing, 2017. ISBN 978-0-08-102025-8. doi: https://doi.org/10.1016/B978-0-08-102025-8.00024-7. URL `https://www.sciencedirect.com/science/article/pii/B9780081020258000247`.

Anirudha Ghosh, A. Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. 01 2020. ISBN 978-3-030-32643-2. doi: 10.1007/978-3-030-32644-9_36.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997. URL `http://www.bioinf.jku.at/publications/older/2604.pdf`.

Sibi Jacob, Kochumol Abraham, and Dr K Sridharan. Romanized language identification and transliteration system for security with an authentication system using persuasive cued click points - rlits. *International Journal of Engineering Research and Development*, 10, 2014. URL `https://www.academia.edu/7511902/Romanized_Language_Identification_and_Transliteration_System_for_Security_with_an_Authentication_System_Using_Persuasive_Cued_Click_Points_RLITS`.

Anirudha Joshi, Girish Dalvi, Manjiri Joshi, Prasad Rashinkar, and Aniket Sarangdhar. Design and evaluation of devanagari virtual keyboards for touch screen mobile phones. pages 323–332, 08 2011. doi: 10.1145/2037373.2037422.

Rahul Mhaiskar. Devanagari: A historical overview and a study of writing style in aga khan palace photo gallery. pages 197–208, 01 2014. doi: 10.2307/26264698.

Shriphani Palakodety and Ashiqur Khudabukhsh. Annotation efficient language identification from weak labels. In *Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020)*, pages 181–192, Online, November 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.wnut-1.24`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Brian Roark, Lawrence Wolf-Sonkin, Christo Kirov, Sabrina J. Mielke, Cibu Johny, Işin Demirşahin, and Keith Hall. Processing South Asian languages written in the Latin script: the Dakshina dataset. In *Proceedings of The 12th Language Resources and Evaluation Conference (LREC)*, pages 2413–2423, 2020. URL `https://www.aclweb.org/anthology/2020.lrec-1.294`.

Deepak Kumar Sharma, Anurag Singh, and Abhishek Saroha. *Language Identification for Hindi Language Transliterated Text in Roman Script Using Generative Adversarial Networks*, pages 267–279. Springer Singapore, Singapore, 2018. ISBN 978-981-13-2348-5. doi: 10.1007/978-981-13-2348-5_20. URL `https://doi.org/10.1007/978-981-13-2348-5_20`.

Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL `http://arxiv.org/abs/1808.03314`.