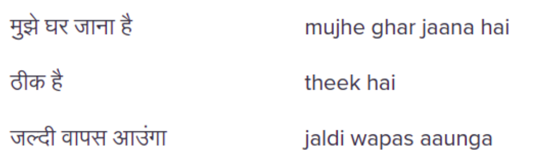
Transliteration

Jerrin John Thomas - 2019114012
Jayant Panwar - 2019114013

Advanced NLP Project Final Report

Introduction

Transliteration is the process of conversion of a text from one script to another. It is a mapping from one system of writing into another, by swapping the letters in predictable ways. Transliteration is concerned with representing the characters, ideally accurately and unambiguously, i.e., one-to-one, so a reader who knows the system can reconstruct the original spelling. It helps pronounce words in a foreign language. For example:



मुझे घर जाना है	mujhe ghar jaana hai
ठीक है	theek hai
जल्दी वापस आउंगा	jaldi wapas aaunga

The major aim of this project is to deal with transliteration of Hindi from its native Devanagari script to the Roman script.

Literature Review

There have been some good research developments in the field of Transliteration. Text for multiple languages has been transliterated into many different scripts using different models/approaches. We take inspiration from such works when building our own Baseline and Baseline++ Models.

Not much literature exists for implementing Transliteration as a Classification task, especially transliterating Hindi from its native script to Roman Script. Nevertheless we have taken inspiration from some of the works that were available. Some techniques we reviewed which have been helpful in some way or the other include:

1. **Sequence Transliteration Using Encoder-Decoder Models:** This paper by Kartik Soni explains the architecture of such a modified neural encoder-decoder model that maximizes parameter sharing across language pair in order to effectively leverage orthographic similarity.
2. **A Deep Learning Based Approach to Transliteration:** This paper talks about how Machine Transliteration can be achieved for Indian Languages using two different sequence-to-sequence architectures based on NMT Framework: RNNs and CNNs.
3. **Hindi and Marathi to English Machine Transliteration using SVM:** This paper from IJNLC explains the architecture of making use of SVMs for transliterating Hindi and Marathi from Devanagari Script to Roman Script.

Dataset Pre-processing

The dataset we have selected for implementing both of our models is the Dakshina dataset. The Dakshina 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 Wikipedia text, 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 divided into 3 different sets with the following splits:

1. Training - 60%
2. Testing - 20%
3. Validation - 20%

For the pre-processing, three phases are followed:

- **Generating Vocabulary:** both the Hindi native script data and the target Roman script data are processed to generate the Vocabulary. This vocabulary consists of characters instead of words.
- **word2vector:** A very simple vector mapping for all the data words is created using the previously built vocabulary. Vectors will be helpful in training and testing our baseline and baseline++ models. The generated vector sequences are padded accordingly so as to run smoothly with our models.
- **vector2word:** A reverse mapping for representing every vector as a word. It helps in debugging and to determine the working performance of our model by looking at the readable output for any given test sentence.

The pairs of words that are out of the vocabulary are ignored and are not transliterated as part of the cleaning process. This generally includes special characters and script specific or language specific characters which do not have a proper defined target transliteration.

Models

Baseline: Seq2Seq Encoder & Decoder

For our Baseline model, we make use of a Sequence-to-Sequence model. A sequence-to-sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ. This overcomes the drawback of the normal LSTM model. Moreover, it is the best way out for implementing transliteration as a single character in Devanagari script may map to one or more characters in Roman script.

Like any other Sequence-to-Sequence model, our baseline model consists of the following important components:

- **Encoder:** The Encoder of our baseline model makes use of the *Embedding, GRU*, and *Dropout* layers. It takes the source (input) vector and source size as the input and is able to generate a proper hidden state.
- **Hidden state:** This state generated by our Encoder is of the most importance to us. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate transliteration predictions. The general formula for building the hidden vector is as follows:
  $h_t = f(W^{hh}h_{t-1} + W^{hx}x_t)$
  where x is the input vector and W are the weights. The weights in our model are initialized using normal initialization which is parameterized using mean and standard deviation.
- **Decoder:** Another essential component of our baseline model is the decoder. Our decoder makes use of the *Embedding, GRU,* and *Dropout* layers like our Encoder but in addition to them it also has two *Linear layers* as a *Sequential* module to help with the generation of the output. It takes the target prediction, the encoder output, and the hidden state generated by the Encoder as its input and generates the output vector and it's own hidden state.
- **Teacher Forcing:** Teacher forcing works by using the target output from the training dataset at the current time step y_t as input in the next time step x_{t+1}, rather than the output generated by the network. This improves the sta-

bility and skill of the model and helps to avoid the slow convergence problem.

Baseline++: Random Forest Classifier

For the baseline++ model, the aim was to solve the transliteration problem by making it a classification task. In the case of transliteration, we have to deal with multi-label classification, i.e., sequences of labels are generated for every input vector. The idea is same as our baseline model, these output label sequences will serve the same purpose as our output vectors in baseline model.

The major components of our baseline++ model include:

- **Vector dataset:** The source and target vectors along with the source size are encapsulated into a single dataset which is later utilized by the model for training. The entries in the generated dataset are later padded to ensure the input vectors and output vectors have same dimensions and training can be done smoothly.
- **Random Forest Classifier:** To solve our multi-label classification problem that is transliteration, it is essential to select an appropriate choice of classifier. We have selected Random forest as classifier as they handle multi-label classifications well. Moreover, they are built on many different combinations of Decision Trees and thus, are a generalized model. They are able to avoid the problem of overfitting. To be able to generate multi label outputs, we connect the Random Forest with the **Multioutput Classifier** of sklearn library. The output is prediction vectors which can be mapped to words to make the prediction readable or even used directly for performance measures.

Evaluation Methodology

To evaluate the obtained results, the accuracy is calculated by converting the predicted vector into a word using vector2word of the target script and comparing it with the truth value.

Results and Analysis

For the baseline, we achieved an average overall accuracy of **63.1 %**.
The overall accuracy for baseline++ was found to be **68.18 %**

The accuracy appears to be low because the evaluation method looks for absolute equivalence between the truth value and the predicted value. Example:- ”astitwa” and ”astitva” were marked inaccurate. Hence even though the accuracy seems to be low, the transliteration is pretty reliable.

Drawbacks and Scope for improvement

Even though our models performed reasonably well, it will be wrong to assume that there is no drawback or there is no scope for improvement. Some of the scope for improvements include:

- Currently, the baseline model is not able to handle out of vocabulary characters. The model can be built in such a way that out of vocabulary characters do not drastically reduce the accuracies.
- In our baseline model, the decoder can be extended to a much better model. For example, we can make use of Attention to better capture information. We can also make use of active and passive beam inference in our general seq-2-seq model. It will improve the model further more.
- For the baseline++ model, we can try a different classifier like SVM with different kernels and proper hyperparameter tuning. This will help us give an idea about the best classifier that we can utilize.

Conclusion

In conclusion, we were able to successfully build baseline and baseline++ models, that were able to transliterate Hindi from its native script into Roman script. The baseline++ model was able to do so as a classification task without any drop in performance measure.

*****