Problem Statement:

Develop a model for language detection from transliteration text. Given English text input, the model should identify the underlying language represented by the transliterated text. The languages to be detected include Hindi, Tamil, Kannada, Malayalam, and Telugu.

Objectives:

Design and implement a machine learning or deep learning model capable of accurately detecting the language of transliteration text.

What is Transliteration?

Transliteration is the process of converting text from one script or alphabet into another. It involves representing letters or characters from one writing system into corresponding characters of another writing system-making it easier for you to pronounce in your own language.

For example: The Russian word for "hello" is - Здравствуйте. In English, it is written as "Zdravstvuyte" — that's transliteration! The meaning of the word is not changed, we're just changing the script so English speakers can pronounce it.

Types of Transliteration:

Phonetic: Aims to capture the sounds of the original language as closely as possible. Helps non-native speakers approximate pronunciation more accurately.

Representing the Russian word "Привет" as "Privet" in the Latin alphabet and the Arabic phrase "السلام عليكم" as "As-salamu alaykum" in Roman letters.

Orthographic: Orthographic transliteration focuses more on preserving the visual appearance or spelling of the original word. It's less concerned with capturing exact sounds and more with maintaining the integrity of the written form.

Rendering the Japanese greeting "こんにちは" as "Konnichiwa" in Romanization and Transliterating the Greek word "αγάπη" as "agápi" in the Latin alphabet.

Key Differences:

Focus: Phonetic transliteration emphasizes sounds, while orthographic transliteration focuses on visual representation.

Purpose: Phonetic transliteration aids pronunciation for non-native speakers, whereas orthographic transliteration preserves the written form.

Accuracy: Phonetic transliteration aims for sound accuracy, while orthographic transliteration aims for visual accuracy, sometimes at the expense of precise pronunciation.

Translation involves converting the meaning and content of text from one language to another, ensuring that the essence of the original message is preserved. Transliteration, on the other hand, entails converting the script or writing system of text from one form to another, often aiming to represent the pronunciation or form of the original words. While translation deals with languages and meanings, transliteration deals with scripts and forms.

Example: Un gato y un perro – Spanish

A cat and a dog- Meaning in English.

Uses of Transliteration:

* Language Learning: Transliteration can aid language learners in understanding and pronouncing words in a new language by representing unfamiliar scripts in a familiar alphabet.
* Cross-cultural Communication: It facilitates communication between speakers of different languages by providing a way to convey pronunciation or spelling in a mutually understandable format.
* Online Searchability: Transliteration allows users to search for content online in languages with different scripts, as search engines often support both native scripts and their transliterated versions.
* Standardization: In multilingual contexts, transliteration can help standardize the representation of names, terms, or phrases across different languages or regions.
* Accessibility: Transliteration can make content accessible to individuals who are familiar with a different script or alphabet, enabling them to understand and engage with the text.

Advantages of Transliteration:

* Pronunciation Aid: It helps non-native speakers approximate the pronunciation of words in a different language by representing them in a familiar script.
* Cross-script Compatibility: Transliteration enables communication and information exchange between languages with different writing systems, promoting cultural exchange and understanding.
* Ease of Use: Transliteration allows users to interact with content written in unfamiliar scripts without requiring them to learn a new alphabet or characters.
* Search Engine Optimization (SEO): For businesses or organizations, transliteration can improve online visibility by ensuring that content is searchable using both native scripts and their transliterated versions.
* Consistency: Transliteration can help maintain consistency in the representation of names, terms, or phrases across various documents, databases, or platforms.

Datasets:

Bhasha-Abhijnaanam Dataset:

* Bhasha-Abhijnaanam is a language identification test set for native-script as well as Romanized text spanning 22 Indic languages
* The dataset covers languages such as Assamese, Hindi, Maithili, Nepali, Sanskrit, Tamil, Bengali, Kannada, Malayalam, Oriya, Santali, Telugu, Bodo, Kashmiri, Manipuri, Punjabi, Sindhi, Urdu, Gujarati, Konkani, and Marathi
* The dataset includes data instances with unique identifiers, native sentences, romanized sentences, language, script, and source information

Aksharantar Dataset:

* The Aksharantar dataset is the largest publicly available parallel corpus for transliteration tasks across 21 Indic languages
* It contains 26 million word pairs spanning these languages, with each pair consisting of a word in an Indic language and its English transliteration.
* The dataset is split into training, validation, and test subsets for each language pair. The training set is the largest, containing millions of word pairs for languages like Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Tamil, and Telugu.
* Smaller language pairs like Bodo, Kashmiri, and Konkani also have thousands of word pairs in the dataset.

Implementations:

1. IndicXlit (Indic Cross-Lingual Transliterator) Model:

The IndicXlit model is a transformer-based multilingual transliteration model that supports 21 Indic languages for converting text between Roman and native scripts. It focuses on ensuring the accuracy of the transliteration process.

**Arcitecture:**

The model has 6 encoder and 6 decoder layers, 256 dimensional input embeddings, feedforward network (FFN) dimension of 1024 and 4 attention heads.

The IndicXlit model has approximately 11 million parameters.

The IndicXlit model is trained on the Aksharantar dataset, which is described as the largest publicly available parallel corpus containing 26 million word pairs across 20 Indic languages.

The model supports 21 Indic languages for transliteration tasks. These languages include Assamese, Bengali, Bodo, Gujarati, Hindi, Kannada, Kashmiri, Konkani, Maithili, Malayalam, Manipuri, Marathi, Nepali, Oriya, Punjabi, Sanskrit, Sindhi, Sinhala, Tamil, Telugu, and Urdu.

The Dakshina test set is used to evaluate the performance of the IndicXlit model. It serves as a benchmark dataset against which the transliteration accuracy and effectiveness of the model are measured. The IndicXlit model's performance on the Dakshina test set is compared to the results reported in the Dakshina paper-

<https://arxiv.org/pdf/2205.03018> - this paper provides a detailed analysis of the working and evaluation of the model.

1. IndicLID (Indic Language Identifier) Model:

The IndicLID model is a language identifier developed by AI4Bharat that supports all 22 Indian languages listed in the Indian constitution, both in native-script and romanized text and is trained on the Aksharantar dataset.

It is a two-stage classifier that combines a fast linear classifier with a slower classifier fine-tuned from a pre-trained language model. The model can predict 47 classes, including 24 native-script classes, 21 roman-script classes, and additional classes for English.

The IndicLID model consists of three main components:

IndicLID-BERT:

This is a slower but more accurate classifier fine-tuned from a pre-trained language model

It is used for predicting the language of text when the confidence of the faster linear classifier is below a certain threshold.

IndicLID-FTN:

IndicLID-FTN is a fast linear classifier for native-script text

It is used for quickly predicting the language of text written in the native scripts of Indian languages.

IndicLID-FTR:

IndicLID-FTR is a fast linear classifier for romanized text

It is used for quickly predicting the language of text written in the Roman script, which is commonly used for representing Indian languages.

The IndicLID model is evaluated on the Bhasha-Abhijnaanam benchmark, which is a language identification test set for native-script as well as romanized text spanning 22 Indic languages. The evaluation compares the performance of IndicLID with existing language identifiers to assess its accuracy in identifying the language of text, especially for native-script text.

<https://arxiv.org/pdf/2305.15814> - this paper provides a detailed analysis of the working and evaluation of the model.

Fine tuning IndicXlit model:

1. Organize Train/Test/Valid Data: Create a directory named "corpus". Inside "corpus", create subdirectories for each language pair (e.g., en-X). Place the training, testing, and validation data for each language pair in their respective directories. Name the training files as "train\_x.en" for English-Roman characters and "train\_x.x" for English-Indic characters.
2. Download and decompress the model file and joint vocabulary files:

wget https://storage.googleapis.com/indicates/indicxlit/model.zip

unzip model.zip

1. Binarize the files using the joint dictionaries: Use the preprocess.py script provided with the IndicXLit model to binarize the training, validation, and test files. The script takes the joint dictionaries as input and generates the binarized files in a format suitable for fine-tuning.
2. Adjust hyperparameters like learning rate, batch size, and number of training epochs based on your dataset and computational resources.
3. Evaluate the fine-tuned model on a validation set to monitor performance and adjust parameters if needed.

Fine tuning IndicLID model:

1. Download Pre-trained Models: Download the pre-trained components of the IndicLID model from the following links:

IndicLID-BERT

IndicLID-FTN

IndicLID-FTR

1. Hyperparameter Tuning:

Fine-tune the language model by adjusting hyperparameters such as learning rate, batch size, and number of training epochs. Use the provided pre-trained models as a starting point for fine-tuning on your specific dataset.

1. Evaluation:

Evaluate the fine-tuned models using metrics like precision, recall, and F1-score to assess their performance on your language identification task.

XLM-Roberta model:

* XLM-RoBERTa is a multilingual version of RoBERTa. It is pre-trained on 2.5TB of filtered CommonCrawl data containing 100 languages.
* RoBERTa is a transformers model pretrained on a large corpus in a self-supervised fashion.
* More precisely, it was pretrained with the Masked language modeling (MLM) objective. Taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words.

papluca/xlm-roberta-base-language-detection:

* The model is a fine-tuned version of xlm-roberta-base on the Language Identification dataset, which is a collection of 90k samples consisting of text passages and corresponding language label.
* The model supports the following 20 languages: arabic (ar), bulgarian (bg), german (de), modern greek (el), english (en), spanish (es), french (fr), hindi (hi), italian (it), japanese (ja), dutch (nl), polish (pl), portuguese (pt), russian (ru), swahili (sw), thai (th), turkish (tr), urdu (ur), vietnamese (vi), and chinese (zh)
* The model achieves an average accuracy of 98.5% on the test set, outperforming the langid library constrained to the same 20 languages.

from transformers import pipeline

from transformers import XLMRobertaForSequenceClassification, XLMRobertaTokenizer, Trainer, TrainingArguments

model\_ckpt = "papluca/xlm-roberta-base-language-detection"

pipe = pipeline("text-classification", model=model\_ckpt)

This model works for the Indian language Hindi. The model can be fine tuned for detecting other languages such as Tamil, Telugu, Kannada and Malayalam.

The steps include:

* Loading the Aksharantar dataset in the required format
* Preprocessing the data
* Splitting the dataset into training and testing sets
* Loading pretrained tokenizer and model- the text is tokenized
* The model is compiled and trained
* Evaluation of the model is done