International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048
Volume 7, Issue 10, pp 97-107, October-2023
https://doi.org/10.47001/IRJIET/2023.710013

SinLingua: Python Library for Sinhala Data Processing

[1]Supun Sameera, [2]Sandaruwini Galappaththi, [3]Sarada Wijesinghe, [4]Binura Yasodya, [5]Anjalie Gamage, [6]Bhagyanie Chathurika

[1,2,3,4,5,6]Department of Information Technology, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka
Authors E-mail: [1]supunsameeran@gmail.com, [2]sandaruwinigalappaththi@gmail.com, [3]saradawijesinghe@gmail.com, [4]binurayasodya24@gmail.com, [5]angalie.g@sliit.lk, [6]bhagyanie.c@sliit.lk

***Abstract* - SinLingua, a novel Python library designed to advance the domain of Sinhala Natural Language Processing (NLP). The primary focus of this work encompasses four distinct areas: Singlish to Sinhala conversion, Sinhala text data cleaning and pre-processing, Sinhala grammar correction, and Sinhala text summarization and translation. Each component is meticulously crafted to prioritize accuracy, speed, customization, and user experience. The Singlish to Sinhala converter is engineered to adeptly recognize and precisely translate Singlish text into formal Sinhala, addressing the paucity of existing tools in this domain. The Sinhala text cleaning and pre-processing function employs optimized rule-based mechanisms to handle the intricacies of the Sinhala language's morphological structures. Furthermore, the Sinhala grammar checker serves the purpose of transforming informal Sinhala sentences into formal ones. Finally, the text summarization and translation module proficiently condenses Sinhala articles while offering translation into the English language. This system provides customization options for summarization parameters, such as word count limits and language translation. The results of this research demonstrate promise, with identified prospects for future enhancements, particularly in the realm of handling intricate grammatical structures and extending user customization features.**

Keywords: Sinhala Natural Language Processing (NLP), Singlish to Sinhala conversion, Sinhala text data cleaning and preprocessing, Sinhala grammar correction, Sinhala text summarization and translation, Python library, Machine Learning (ML).

I. INTRODUCTION

Language plays a fundamental role in preserving cultural heritage and fostering communication, knowledge dissemination, and information sharing. However, the diversity of languages across the world poses unique challenges for natural language processing (NLP). In this context, the Sinhala language, with its distinctive linguistic characteristics, has garnered attention as a subject of significant study. Sinhala, primarily spoken in Sri Lanka, presents intriguing complexities that demand specialized tools and resources for effective processing. The SinLingua Library, a comprehensive toolkit designed to tackle a wide range of Sinhala language processing tasks, represents a major milestone in this endeavor. This research explores the key components of the SinLingua Library, addressing critical challenges and gaps in Sinhala NLP like Singlish to Sinhala conversion, Sinhala grammar rules mapping, Sinhala text preprocessing, and Sinhala text summarization.

One of the foremost challenges in Sinhala language processing is the accurate conversion of Singlish, Romanized Sinhala text, into the Sinhala script. The SinLingua Library offers multiple approaches for this task, including rule-based translation, machine translation using FastText models, hybrid translation, and manual translation. These approaches enable efficient and contextually accurate conversion of Singlish into Sinhala text. Through rule-based methods, it ensures rapid and consistent translations. The introduction of machine learning models enhances accuracy and contextual depth, while the hybrid approach combines rule-based techniques and Large Language Models for a holistic solution. Additionally, the library empowers users to manually refine translations, ensuring linguistic precision tailored to specific dialects or preferences.

The complex grammatical structure of Sinhala presents another set of challenges in language processing. While the SinLingua Library offers grammar conversion capabilities, this research highlights the need for further improvement. The library employs a combination of rule-based approaches and machine learning, including the Sinhala BER To model, to transform informal Sinhala verbal sentences into text conforming to Sinhala grammar rules. This allows for the refinement of sentence structure, tense, subject-verb agreement, and more, ensuring grammatical correctness. Moreover, the library offers the capability to predict missing words using state-of-the-art language models, further enhancing the linguistic accuracy and completeness of the text.

Text preprocessing is a fundamental step in many language processing tasks. The SinLingua Library addresses this by providing a systematic four-step stemming process

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Issue 10, pp 97-107, October-2023

https://doi.org/10.47001/IRJIET/2023.710013

through the "SinhalaStemmer" class. This process reduces words to their root forms, enhancing accuracy and efficiency in downstream text analysis. It navigates the complexities of the Sinhala language's agglutinative nature, ensuring precise stemming even in the face of irregular inflections and domain-specific terms. Additionally, the library offers stop word handling and tokenization, critical for refining and segmenting text for various linguistic tasks.

Text summarization is a vital area of research, and the SinLingua Library explores diverse models and techniques for generating accurate and contextually relevant summaries in Sinhala. This research evaluates the performance of the TF-IDF model, BERT model, Longformer, FastBERT, and FastLong former in the context of Sinhala text summarization. Each model offers a unique balance of accuracy and speed, catering to different user requirements. The choice of the "best" model depends on the specific use case, with some users favoring quick, concise summaries, while others require more in-depth and comprehensive insights.

In conclusion, the SinLingua Library is a valuable asset for Sinhala language processing, but it is accompanied by several research problems and challenges. This comprehensive study highlights these issues, providing a roadmap for future advancements in Sinhala NLP. Addressing these challenges not only enhances the library's capabilities but also contributes to the broader field of Sinhala language processing, benefiting researchers, developers, and the Sinhala language community. The research presented here is a testament to the commitment to preserving and promoting linguistic diversity in the digital age.

II. PROBLEM DEFINITION

Sri Lankan languages, particularly Sinhala, have unique linguistic characteristics that pose intriguing challenges for natural language processing (NLP). Among the Sinhala language processing tools and resources, the SinLingua Library stands as a comprehensive toolkit designed to address a wide range of tasks, including text conversion, grammar analysis, and summarization. While the library represents a significant advancement in the field, there exists a critical need for a comprehensive study that investigates various research problems and challenges associated with Sinhala language processing. This research problem aims to explore these challenges, identify gaps, and pave the way for advancements in Sinhala NLP.

One of the foremost challenges in Sinhala language processing is the accurate conversion of Singlish, Romanized Sinhala text, into the Sinhala script. The SinLingua Library's Singlish to Sinhala Text Conversion component provides a foundational solution, but its accuracy can be further enhanced. Research is required to develop robust algorithms that can handle the diverse variations and ambiguities inherent in Singlish. Moreover, the library must be equipped to handle code-switching, a common phenomenon where Sinhala and English words are intermixed in text.

The complex grammatical structure of Sinhala presents yet another set of challenges. While the SinLingua Library offers grammar conversion capabilities, there is room for improvement. Advanced grammar rules, including verb conjugations and complex sentence structures, should be addressed to provide more accurate conversions. Incorporating contextual analysis to adapt grammar conversion to context-rich documents is also a crucial research problem.

Text preprocessing, a fundamental step in many language processing tasks, requires focused research. Named Entity Recognition (NER) in Sinhala is currently underdeveloped. Developing NER models for Sinhala to identify entities like names of people, places, and organizations is a pressing need. Furthermore, enhancing morphological analysis tools for Sinhala is essential for tasks like stemming and lemmatization.

Lastly, Sinhala text summarization is a promising area for research. Existing summarization techniques require further exploration, especially in the context of abstractive summarization. Generating summaries with better fluency and coherence remains a significant challenge. Additionally, investigating multilingual summarization, where the source text is in Sinhala but the summary can be generated in other languages, presents an exciting avenue for research.

In conclusion, the SinLingua Library is a valuable asset for Sinhala language processing, but it is accompanied by several research problems and challenges. This comprehensive study aims to highlight these issues and encourage further research to bridge the gaps in Sinhala NLP. Addressing these challenges will not only enhance the library's capabilities but also contribute to the broader field of Sinhala language processing, benefiting researchers, developers, and the Sinhala language community.

III. LITERATURE SURVEY

The field of Natural Language Processing (NLP) has witnessed significant growth and innovation, encompassing a multitude of research studies and tools dedicated to addressing the unique linguistic challenges posed by the Sinhala language. This literature review delves into these challenges and the existing solutions that serve as the foundation for the proposed project, which aims to advance Sinhala NLP through a comprehensive Python library known as SinLingua.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048
Volume 7, Issue 10, pp 97-107, October-2023
https://doi.org/10.47001/IRJIET/2023.710013

3.1 Singlish to Sinhala Language Conversion

The primary objective of SinLingua is to bridge a noticeable research gap in the realm of Singlish to Sinhala language conversion. While natural language processing techniques have been explored for various language-related tasks, limited research has focused on the intricate task of converting informal Singlish to formal written Sinhala. Singlish, characterized by its informal nature and a fusion of Singlish and English, presents unique translational challenges.

Traditionally, rule-based approaches have been commonly employed in previous studies for Singlish to Sinhala conversion. These studies have leveraged Sinhala phonetic sets and transliteration rules to facilitate the conversion process [1][2][3][4][5]. However, the accuracy of these rule-based systems is often hindered by the variations in Singlish spelling and Sinhala pronunciation, making it a suboptimal solution for complex translations.

In contrast, machine learning-based approaches have demonstrated substantial potential in NLP tasks [3][6]. A rule-based transliteration technique for Singlish to Sinhala has been explored, achieving an accuracy of 93.2% by employing a dictionary-based approach and rules for complex phrases [7]. The proposed project acknowledges the limitations of rule-based approaches and aims to leverage machine learning techniques to develop a Python library that can accurately convert Singlish to Sinhala, effectively addressing the challenges posed by variations and nuances present in the Singlish language [8].

3.2 Sinhala Grammar Rule Mapping

Sinhala language processing has faced constraints, with limited resources and tools for efficient conversion of spoken Sinhala to written text. Existing tools provide machine learning-based solutions but often lack proper grammar rules for accurate Sinhala language conversion. This absence of grammar rules results in reduced accuracy and efficiency. Additionally, these tools lack the real-time performance required for various applications.

The proposed SinLingua library aims to bridge this gap by implementing a machine learning-based approach that provides accurate Sinhala language conversion, with proper grammar rules, offering optimized real-time performance. It addresses the research gap by combining rule-based and machine learning-based methods, allowing accurate stem word generation and enhancing adaptability to different accents and dialects [8].

3.3 Sinhala Text Preprocessing

While several tools and libraries for Sinhala text processing exist, there are significant limitations that hinder their efficiency in cleaning and preprocessing Sinhala text data. Rule-based approaches are commonly employed in existing tools, but these have limitations in handling variations and nuances of the Sinhala language, and most do not support different variations of Sinhala text [9][10][11].

The proposed SinLingua library introduces a hybrid approach that combines rule-based and machine learning-based methods to handle complex Sinhala text data accurately and efficiently [11]. It aims to fill the research gap by supporting various Sinhala language variations, enabling integration with other systems, and offering support for large text datasets. Additionally, the library focuses on user interface design, providing a user-friendly experience.

3.4 Sinhala Text Summarization

The realm of Sinhala text summarization has seen prior research efforts that primarily focused on extractive summarization methods [6]. These studies concentrated on specific domains, such as news articles, and employed methods like text rank algorithms for extractive summarization. These works contributed to the understanding of Sinhala text summarization but often had limitations, including a lack of customization options for summarization length and coverage for various content types.

The proposed SinLingua library endeavors to address these limitations by introducing a hybrid summarization model, customizable summarization length, and translation capabilities into the English language. These features are expected to enhance the utility of Sinhala text summarization and make it accessible to a wider audience, regardless of their language proficiency.

In conclusion, the existing research and tools in Sinhala NLP have paved the way for innovative solutions like SinLingua [11][12]. This comprehensive Python library combines rule-based and machine learning-based approaches to tackle the challenges posed by Singlish to Sinhala conversion, Sinhala language conversion, text preprocessing, and text summarization. By addressing these research gaps, the proposed library aims to contribute to the growth of Sinhala NLP, benefiting various industries and enabling more efficient and accurate language processing.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Issue 10, pp 97-107, October-2023

https://doi.org/10.47001/IRJIET/2023.710013

IV. METHODOLOGY

The methodology encompasses a comprehensive approach to develop the SinLingua Python library, addressing various facets of Sinhala language processing. This methodology is structured to ensure the effective conversion of Singlish to Sinhala, efficient grammar rule mapping, sophisticated text preprocessing, and customizable text summarization.

Given below is the high-level architecture diagram of SinLingua library.

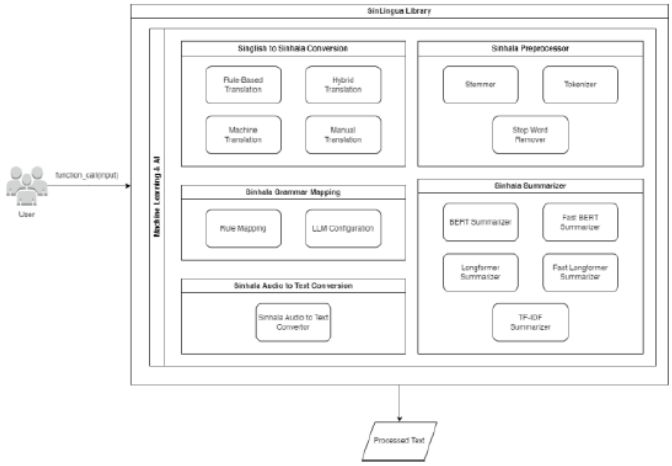

Figure 1: High-level architecture diagram

4.1 Singlish to Sinhala Language Conversion

Rule-Based Translation: The Rule-Based Translation approach offered by the SinLingua Singlish to Sinhala conversion is a method that relies on predefined linguistic rules to convert Singlish text into Sinhala. This deterministic approach ensures efficient, consistent, and rapid translations. Users can utilize this method by employing the "RuleBasedTransliterator" class. Once the class is imported and instantiated, the user can simply input the desired Singlish text, invoking the "transliterator" method to obtain the Sinhala translation. This approach is ideal for those seeking quick and uniform translations. First the library separates Singlish text into logical groups and then it maps corresponding Sinhala terms through an algorithm from a pre-defined python dictionary, which is having the corresponding Singlish term and its Sinhala term as the key-value pairs for each vowel, consonant and dependent vowel. As an example; "a": "ඇ", "aa": "ා", and "sh": "ශ".Finally, it gets "ඔයාටදැන්කොහොමද" for input "oyaata dhaen kohomadha".

```
while i < len(word):
    try:
        for length in range(3, 0, -1):
            if i + length <= len(word):
                group = word[i:i + length]
                if group in self.vowels:
                    logical_groups.append(self.vowels[group])
                    i += length

        for length in range(4, 0, -1):
            if i + length <= len(word):
                group = word[i:i + length]
                if group in self.consonants:
                    logical_groups.append(current_group)
                    current_group = self.consonants[group]
                    i += length
```

Figure 2: Part of the rule-based algorithm

Machine Translation via the FastText Model: Machine Translation using the FastText model integrates word embeddings [13], specifically FastText's pre-trained Sinhala word vectors, to heighten translation accuracy. Word embeddings, which map words to a semantic space based on context, allow for richer and contextually apt translations. The process starts with a rule-based translation of Singlish text to Sinhala. This initial translation is then refined using FastText's vectors to suggest semantically similar Sinhala words. This dual approach ensures the translated text captures both the direct meaning and the nuanced context of the original text. To facilitate this, the library includes the "transliterator" function of "MachineTransliterator" class, enabling users to harness this combined power of rule-based translation and semantic word embeddings seamlessly.

```
def __get_best_suggestions(self, word: str) -> list:
    try:
        if word in self.fasttext_model:
            # Get most similar words from the FastText model
            most_similar_words = self.fasttext_model.most_similar(word)
            best_suggestions = [similar_word[0] for similar_word in most_similar_words]
            return best_suggestions
        else:
            return []
    except Exception as e:
        # Handle exceptions and return an empty list
        print(f"An error occurred while getting suggestions: {str(e)}")
        return []
```

Figure 3: Function for get best suggestions for a particular word through FastText model

Hybrid Translation: The SinLingua Singlish Hybrid Translation method synergizes rule-based and Large Language Models (LLMs) to yield precise and context-sensitive Sinhala translations. Here it used gpt-3.5-turbo model from OpenAI as the LLM [14]. At its core lies the "HybridTransliterator" class, which facilitates this fusion. Users can employ this class for transliteration tasks. For refining translations, two utilities are offered: The Machine Mask Translation, which identifies and masks misspelled Sinhala words, and the Machine Suggest Translation, which provides replacement suggestions for masked words. This latter utility can be applied using two distinct approaches: one involves manual word masking, while the other harnesses automated machine masking. User should provide OpenAI API key and Organization ID as the parameters for the class HybridTransliterator. Additionally,

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Issue 10, pp 97-107, October-2023

https://doi.org/10.47001/IRJIET/2023.710013

users can customize default prompts via optional parameters other than the pre-defined prompts which library utilizes to get responses through API for a more tailored experience.

```
def __get_gpt_response(self, text: str, level: int, word: str = "") -> str:
    completion = None
    try:
        # Set up API key and organization for OpenAI
        openai.api_key = self.json_data["api_key"]
        openai.organization = self.json_data["org_key"]

        # Create user prompt using provided text and level
        user_prompt = self.json_data["Prompts"][level]["content"].replace("{{masked-sentence}}", text).replace(
            "{{misspelled-word}}", word)

        # Check if the provided text is empty
        if not text.strip():
            raise ValueError("Text is empty. Please provide a valid text string.")

        success = False
        while not success:
            try:
                # Create a ChatCompletion request to GPT-3
                completion = openai.ChatCompletion.create(
                    model=self.json_data["model"],
                    messages=[
                        {
                            "role": self.json_data["Prompts"][level]['role'],
```

Figure 4: Function to process prompts through gpt-3.5-turbo model

Manual Translation: The SinLingua Singlish Manual Translation method allows for hands-on, context-specific adjustments to translations using the "ManualTransliterator" class. Initially, users map each word in the Sinhala text to a unique coordinate with the "generate_coordinates" method, which can be exported to a CSV for visualization. Next, targeted cells within this coordinate grid can be altered using the "replace_cells" method, with an option to undo changes if needed. The framework also offers manual masking, where specific words can be temporarily obscured and later refined; this is especially useful in tandem with the Hybrid Translation's "machine_suggest" method. Finally, the "reconstruct_text" method reverts the modified dataframe back into a textual format, ensuring that translations align with a user's contextual intent.

4.2 Sinhala Grammar Rule Mapping

The Sinhala Verbal Text to Written Text Conversion module is designed to transform informal Sinhala verbal sentences into written text that conforms to Sinhala grammar rules. The conversion process is achieved through a combination of rule-based and machine learning approaches.

Rule-Based Approach: The initial step involves employing a set of predefined Sinhala grammar rules to the given verbal sentence. These rules encompass various grammatical aspects such as tense, subject-verb agreement, and pluralization. Each rule is represented by a function within the "GrammarMain" class. For instance, based on the subject's tense and the corresponding verb, the module applies the appropriate rule. For example, sentences starting with "මම" or "මා" in present tense should end with "මි". Similarly, sentences starting with "අප" or "අපි" in present tense should conclude with "මු". The module maps different subject pronouns and tenses to the correct verb endings, ensuring grammatical correctness.

Machine Learning Approach: To enhance accuracy, the module leverages the SinhalaBERTo model, a Sinhala BERT-based language model. Integrated into the LLMConfig class, this model predicts suitable written text for the given verbal sentences.

By inputting the verbal sentence into SinhalaBERTo, the module generates contextually relevant written output. This output serves as an alternative or validation for the rule-based conversion.

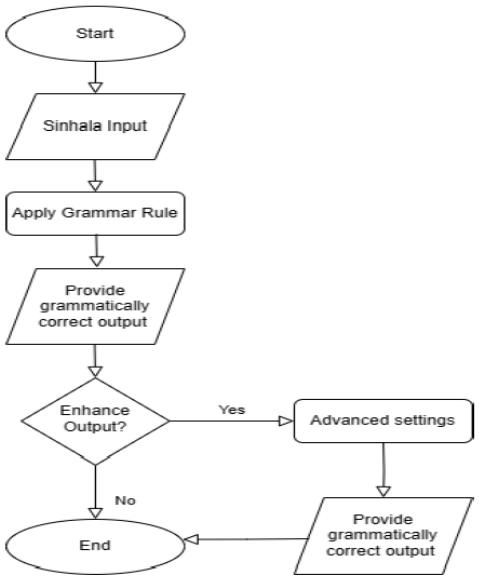

Figure 5: Sinhala grammar rules process flowchart

4.3 Sinhala Text Preprocessing

The "SinhalaStemmer" class, an integral component of the SinLingua library, is devised to elevate the accuracy and efficiency of Sinhala text preprocessing through a systematic four-step stemming process. Each procedure is meticulously crafted to address different aspects of stemming intricacies within the Sinhala language.

Stem Dictionary Lookup (Step One): This initial procedure harnesses a substantial corpus called the stem dictionary, which pairs Sinhala words with their corresponding stemmed forms. The objective is to determine potential stems for input words by conducting a straightforward comparison. Within the method "step_one", the program queries the stem dictionary for a match with the provided input word. Upon finding a match, the method directly substitutes the input word with its corresponding stemmed form.

Suffix Removal (Step Two): The second procedure focuses on the identification and removal of common suffixes inherent to the Sinhala language. The method "step_two"

ISSN (online): 2581-3048
Volume 7, Issue 10, pp 97-107, October-2023
https://doi.org/10.47001/IRJIET/2023.710013

systematically examines the input word for the presence of recognized suffixes stored in the suffix list. By iterating through these suffixes and checking for their occurrence at the end of the input word, the algorithm identifies and eliminates suffixes that signal grammatical variations or tense changes.

Inner Suffix Handling (Step Three): Complex word formations in Sinhala can result in nested suffixes, presenting a challenge for accurate stemming. The third procedure, "step_three" is designed to manage this intricacy. It extends beyond the basic suffix removal by addressing inner suffixes that might exist within a word. By examining both the suffixes and their preceding characters, the algorithm accurately identifies and removes inner suffixes. If an inner suffix removal alters dependent vowels, the algorithm ensures proper handling to maintain word integrity.

Dependent Vowel Suffix Removal (Step Four): Sinhala suffixes that commence with dependent vowels are distinctive and demand specialized treatment. The fourth procedure, "step_four" specifically targets these cases. The algorithm identifies suffixes with dependent vowel prefixes and eliminates them, contributing to cohesive stemming while preserving the core meaning of words. The step-by-step execution of this process aligns with the algorithm's meticulous approach, ensuring the comprehensive handling of diverse suffix patterns.

The "SinhalaStemmer" class encapsulates the complete stemming process, with the overarching method stemmer accepting both individual words and lists of words. This method orchestrates the application of all four procedures, accommodating variations in input type and word characteristics. For individual words, the method passes them through each procedure sequentially, producing the final stemmed form. In the case of word lists, the method performs the same procedure sequence for each word, generating a list of stemmed outputs.

Stop word handling is a fundamental step in the Sinhala text preprocessing pipeline. It involves the identification and removal of stop words – frequently occurring but contextually uninformative words – from the text. These words, such as "සහ" (and), "සමග" (with), and "ලෙස" (like), are part of a predefined list of stop words. The primary goal of this process is to eliminate words that add minimal semantic value to the text while focusing on the meaningful content.

Tokenization is a foundational process in Sinhala text preprocessing that involves breaking down a continuous stream of text into individual units known as tokens. These tokens typically correspond to words or sub-words, and they serve as the building blocks for subsequent linguistic analysis and processing.

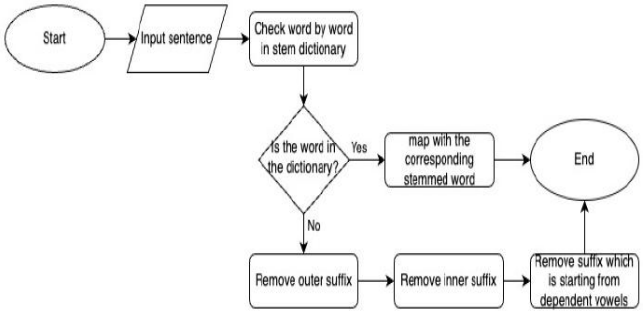

Figure 6: Sinhala text preprocessing process flowchart

4.4 Sinhala Text Summarization

In the comprehensive study of text summarization methodologies, a diverse array of models was evaluated, spanning both traditional and cutting-edge techniques. This analysis encompassed the TF-IDF model, a well-established statistical method that gauges a term's significance through its frequency. Additionally, the BERT (Bidirectional Encoder Representations from Transformers) model, a deep learning-based approach, was included due to its prowess in understanding bidirectional textual context. The Longformer, a modification of the transformer architecture, stands out for its capability to process extensive texts via a specialized global attention mechanism. The FastBERT model, a more efficient iteration of BERT, offers a harmonious blend of performance and speed. Lastly, The Fast Longformer combines the swiftness of DistilBERT with the depth of Longformer, offering a balanced hybrid solution. To optimize the input textual data, which encompassed articles, blogs, research papers, and narratives, several preprocessing steps were undertaken. This included tokenization, removal of stopwords, and sentence segmentation, all executed with an emphasis on maintaining linguistic integrity.

TF-IDF Model: Statistical measure evaluating word importance in a document relative to a corpus. To generate summary, compute term frequency (TF) for each word in the document. Then calculate inverse document frequency (IDF) for each term against the corpus. Assign a weight to each sentence based on the aggregate TF-IDF scores of its constituent terms. Finally Select the top-ranking sentences to form the summary.

BERT Model: Deep learning model that analyzes text bidirectionally to understand word context. To generate summary, tokenize the input document. Convert the tokens into embeddings using BERT's pre-trained weights. Process the embeddings through an encoder-decoder neural network structure. Predict the inclusion probability of each sentence in the summary. Select the top-ranked sentences based on their predicted probabilities.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Issue 10, pp 97-107, October-2023

https://doi.org/10.47001/IRJIET/2023.710013

Longformer Model: An extension of BERT designed to handle longer documents using both local and globalattention mechanisms. To generate summary, tokenize the input document. Convert the tokens into embeddings similarly to BERT. Process the entire document without truncation using Longformer's attention mechanisms. Predict the importance of each sentence based on the model's outputs. Aggregate the most relevant sentences to form the summary.

FastBERT Model: A lighter and faster version of BERT achieved by training a smaller student model to mimic a larger, pre-trained teacher model. To generate summary, tokenize the input document. Convert tokens into embeddings. Evaluate the significance of each sentence using the streamlined architecture of DistilBERT. Compile the top-ranking sentences to produce the summary.

Fast Longformer (Hybrid Model): A hybrid model that combines the speed of DistilBERT with the long-document handling capability of Longformer. To generate summary, tokenize the input document. Convert tokens into embeddings. Evaluate the relevance of each sentence using a combination of distilled architecture and global attention mechanisms. Concatenate the most pertinent sentences to produce the summary.

V. RESULT AND DISCUSSION

The central outcome of this project is the publication of the SinLingua library on PyPI, the Python package repository. This library simplifies the process of Sinhala language processing and can be conveniently installed using the 'pip' command.

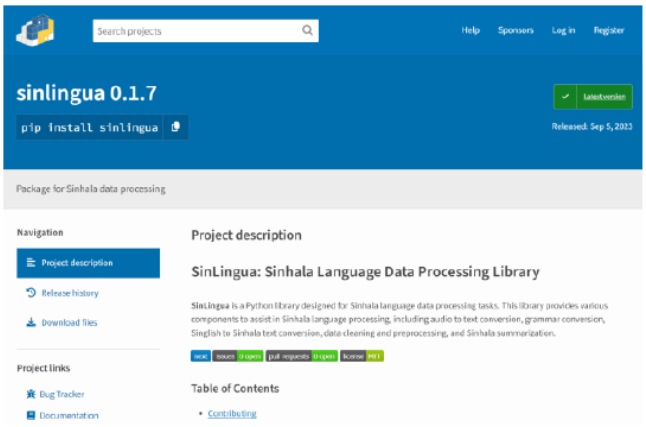

Figure 7: PyPI project page for SinLingua library

The functional results obtained using the SinLingua Python library can be categorized into four fundamental components, each of which is designed to enhance and streamline various aspects of Sinhala language processing.

5.1 Singlish to Sinhala Conversion

After importing the SinLingua library, users are required to import specific classes developed within the library based on their intended usage for Singlish to Sinhala language conversion. Below are the classes that need to be imported:

```
[ ]  from sinlingua.singlish.rulebased_transliterator import RuleBasedTransliterator
     from sinlingua.singlish.machine_transliterator import MachineTransliterator
     from sinlingua.singlish.hybrid_transliterator import HybridTransliterator
     from sinlingua.singlish.manual_transliterator import ManualTransliterator
```

Figure 8: Importing required classes for Singlish to Sinhala conversion in SinLingua library

To facilitate Singlish to Sinhala conversion using a rule-based approach, the SinLingua library provides a dedicated class called "RuleBasedTransliterator." This class serves as the core component for implementing the rule-based conversion method. Users can employ this class as outlined below:

```
[ ]  transliterator = RuleBasedTransliterator()

     singlish_text = "oyaata kohomadha"
     sinhala_text = transliterator.transliterator(singlish_text)
     print(sinhala_text)

     ඔයාට කොහොමද
```

Figure 9: Example of rule-based translator

Here we initialize a variable named singlish_text and assign it the value "oyaata kohomadha", which translates to "ඔයාටකොහොමද" in Sinhala by using. But when using the rule-based approach, there is a limitation related to the need for an exact mapping between Singlish and Sinhala. The rule-based approach relies on predefined rules and mappings to perform the conversion.

However, the introduction of the Machine Translation technique, anchored on the FastText Model, addressed this issue. Upon assessing various translations, it was discernible that by leveraging Sinhala word vectors, the translations weren't just accurate but were infused with contextual depth and linguistic nuances. This makes the translation more natural and relatable to native Sinhala speakers.

The machine learning (ML) approach, while offering notable advantages, does come with certain disadvantages. One significant drawback is its time-consuming nature. Additionally, ML models, such as those relying on FastText vectors, necessitate that the words in the input text exist within the pre-trained vector space to yield correct results. This limitation can lead to inaccuracies when dealing with out-of-vocabulary terms or domain-specific jargon. To mitigate these shortcomings and ensure a more comprehensive and adaptable solution, the SinLingua library adopts a hybrid approach that combines rule-based methods for handling known expressions and machine learning techniques like Large Language Models (LLMs) for broader coverage and adaptability.

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Issue 10, pp 97-107, October-2023

https://doi.org/10.47001/IRJIET/2023.710013

```
[ ] transliterator = HybridTransliterator(api_key="YOUR_OPENAI_API_KEY", org_key="YOUR_ORGANIZATION_KEY")

    singlish_text = "mama oyaata aadareyi"
    sinhala_text = transliterator.transliterator(text=singlish_text)
    print(sinhala_text)

    මම ඔයාට ආදරෙයි
```

Figure 10: Example of hybrid translator

An illustrative example of this is the Singlish phrase "oyata den kohomada." While conventional translation tools often falter, yielding outputs like (ඔයටඩෙන්කොහොමඩ), the SinLingua system excels, providing a more authentic translation as (ඔයාටදැන්කොහොමද, oyaata dhaen kohomadha). Notably, what distinguishes the system is its optional manual translation feature, underscoring the indispensable role of human intervention in translations. By allowing users to refine translations, it not only ensures the highest level of accuracy but also empowers users to tailor translations to specific dialects or regional preferences.

5.2 Sinhala Grammar Rule Mapping

Here are the Sinhala grammar rules defined within the scope for mapping purposes.

1. If a sentence starts with 'මම' or 'අපි' in present tense, it should end with 'මි'.
2. If a sentence starts with 'ඔහු' or 'ඇය' in present tense, it should end with 'යි'.
3. If a sentence starts with 'මම' or 'අපි' in future tense, it should end with 'වෙමි' according to the tense and verb.
4. If a sentence starts with 'ඔහු' or 'ඇය' in future tense, it should end with 'වේවි' according to the tense and verb.
5. If a sentence starts with 'මම' or 'අපි' in past tense, it should end with 'මි' according to the tense and verb.
6. If a sentence starts with 'ඔහු' or 'ඇය' in past tense, it should end with 'ය' according to the tense and verb.
7. If a sentence starts with 'අපි' or 'අපේ' in present tense, it should end with 'මු' according to the tense and verb.
8. If a sentence starts with 'අපි' or 'අපේ' in past tense, it should end with 'ම' according to the tense and verb.
9. If a sentence starts with 'මිනිසු' in present tense, it should end with 'ති' according to the tense and verb.
10. If a sentence starts with 'මිනිසු' in past tense, it should end with 'නේ' according to the tense and verb.
11. If a sentence starts with 'මිනිස්' or 'මිනිසු' in present tense, it should end with 'ති' according to the tense and verb.
12. If a sentence starts with 'මිනිස්' or 'මිනිසු' in past tense, it should end with 'නේ' according to the tense and verb.
13. If a sentence starts with 'ඒ' or 'ඔ' or 'මේ' or 'මෙම' in past tense, it should end with 'ය' according to the tense and verb.
14. If a sentence starts with 'ඔ' or 'ඔය' in past tense, it should end with 'යු' according to the tense and verb.
15. If a sentence starts with a plural subject in present tense, it should end with a plural verb in present tense.
16. If a sentence starts with a plural subject in past tense, it should end with a plural verb in past tense.
17. If a sentence starts with a singular subject in present tense, it should end with a singular verb.

Figure 1: List of grammar rules

These rules have been implemented as distinct functions within the SinLingua Library. The purpose of these functions is to establish the mapping of Sinhala grammar rules with a given sentence.

- Example 1: To demonstrate the functionality of the 'mapper()' function within the SinLingua library, consider the input sentence: "අපිකෑමකැවා." Upon applying this sentence to the 'mapper()' function, it produces the following output: "අපිකෑමකැවෙමු."“අපිකෑමකැවා”
- Example 2: Let's illustrate the use of the 'mapper()' function within the SinLingua library with the input sentence: "ඔහුඉතාආදරයෙන්දෙමාපියන්නමදිනවා." When processed by the 'mapper()' function, it yields the following output: "ඔහුඉතාආදරයෙන්දෙමාපියන්නමදියි."

Let's examine these two examples at the codebase level, where user will simply utilize the SinLingua Library to obtain the grammatically corrected output. This is the class that needs to be imported: ‘GrammarMain’.

```
[ ] from sinlingua.grammar_rule.grammar_main import GrammarMain
```

Figure 2: Import GrammarMain class

Here's how to obtain the grammatically corrected output using the given two examples.

- Example 1:

```
[ ] # Create an instance of GrammarMain
    obj = GrammarMain()

    # Original informal sentence
    sentence = "අපි කෑම කැවා"

    # Apply grammar correction
    corrected_sentence = obj.mapper(sentence=sentence)
    print(corrected_sentence)

    අපි කෑම කැවෙමු
```

Figure 3: User calls the mapper() function to get output 1

- Example 2:

```
[ ] # Create an instance of GrammarMain
    obj = GrammarMain()

    # Original informal sentence
    sentence = "ඔහු ඉතා ආදරයෙන් දෙමාපියන් නමදිනවා"

    # Apply grammar correction
    corrected_sentence = obj.mapper(sentence=sentence)
    print(corrected_sentence)

    ඔහු ඉතා ආදරයෙන් දෙමාපියන් නමදියි
```

Figure 4: User calls the mapper() function to get output 2

Let's delve into the step-by-step procedure of the provided code. To begin, we initiate an instance of the 'GrammarMain' class, an integral component of the SinLingua Library. Next, we define the original informal Sinhala sentence and store it in a variable named 'sentence.' Following this, we execute the grammar correction process by invoking the 'mapper' method of the 'GrammarMain' instance, utilizing the 'sentence' variable as an argument. Finally, we retrieve and print the corrected sentence, representing the successful outcome of the grammar correction operation. This sequence of actions ensures the accurate refinement of Sinhala sentences for enhanced linguistic precision.

There is another feature to predict missing word using SinLingua Library. Just need to set <mask> for the word you

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 7, Issue 10, pp 97-107, October-2023

https://doi.org/10.47001/IRJIET/2023.710013

want to predict. This is how the library provides output with up to five appropriate suggestions.

```
[ ]  obj = LLMConfig()

     # Original informal sentence
     sentence = "දරුවා වෙහෙස මහන්සියෙන් ඉගෙන <mask>"
     out = obj.berto_predict_top(sentence=sentence)
     for p in out:
       print(*p)

     දරුවා වෙහෙස මහන්සියෙන් ඉගෙන ඇක  ඇක
     දරුවා වෙහෙස මහන්සියෙන් ඉගෙනේ.ේ.
     දරුවා වෙහෙස මහන්සියෙන් ඉගෙන... ...
     දරුවා වෙහෙස මහන්සියෙන් ඉගෙන ඒමට  ඒමට
     දරුවා වෙහෙස මහන්සියෙන් ඉගෙන ඒම  ඒම
```

Figure 5: Predict five best suggestions from berto_predict_top()

Let's walk through the sequential steps within the provided code. Firstly, we initiate an instance of the 'LLMConfig' class, a fundamental component integrated into the SinLingua Library. Subsequently, we craft an informal Sinhala sentence and incorporate a '<mask>' token, facilitating word prediction. Employing the language model, we utilize the expression 'obj.berto_predict_top(sentence=sentence)' to predict the top word suggestions for the masked word position. Finally, we demonstrate the enhanced capabilities of the SinLingua Library by printing up to five relevant word suggestions. This process underscores the library's role in elevating Sinhala language comprehension and processing, further contributing to its proficiency.

5.3 Sinhala Text Preprocessing

In the realm of Sinhala data processing, Stemming is a critical process within language preprocessing. Its primary objective is the reduction of words to their fundamental or root forms. In the context of Sinhala, stemming presents unique challenges owing to the language's agglutinative nature, where affixes are added to words for conveying diverse meanings. This section delves into the stemming component of the Sinhala text preprocessing library and the methodologies it employs to enhance accuracy.

The stemming component yielded promising results in handling regular inflections and common word forms. It consistently and accurately reduced words to their base forms, thereby enhancing downstream text analysis. Nevertheless, it's imperative to recognize the challenges posed by irregular inflections and domain-specific terms that may fall outside the scope of the stem dictionary. In such cases, incorrect stemming or the preservation of original words could occur.

Given below is the way of use Sinhala stemming functionality of SinLingua. First user should initialize SinhalaStemmer from SinLingua's preprocessor module.

```
from sinlingua.preprocessor.tokenizer import SinhalaTokenizer
from sinlingua.preprocessor.stopword_remover import StopWordRemover
from sinlingua.preprocessor.stemmer import SinhalaStemmer
```

Figure 6: Import preprocessor module classes from SinLingua library

The following code segment demonstrates the utilization of the 'SinhalaStemmer' class within the SinLingua library. It serves the purpose of stemming, which is the process of reducing words to their root or base form in the context of Sinhala text preprocessing. The outcome is a more refined and structured representation of the original text, contributing to enhanced accuracy and efficiency in subsequent text analysis and language processing tasks.

```
[ ]  stemmer = SinhalaStemmer()
     stem_words = stemmer.stemmer(sample_text2)
     print(stem_words)

     ['ඒ', 'පිරමිඩ', 'ජාවාරම', 'උපයාග', 'ඒ', 'ඛව', 'හඳුනාග', 'ඇ', 'බැව්',
```

Figure 7: Example for use stemmer() function

Next, preprocessor module focuses on stopword removal. It employs the 'StopWordRemover' class to eliminate common stopwords from a given paragraph. This process is critical for refining the input text, as stopwords often carry little semantic value and can be a hindrance in various language processing tasks. The outcome, 'remaining_words,' represents the paragraph with stopwords removed, enhancing the quality of the text for downstream analysis.

```
[ ]  stopword_remover = StopWordRemover()
     remaining_words = stopword_remover.remove_stop_words(paragraph)
     print(remaining_words)

     ආහාරමය සියල්ලේම ආහාරවලට මිල ගණන් කොකරම් ඉහළ යන්න අපි සැවොම දන්නා කාරණයකි
```

Figure 8: Example for use remove_stop_words() function

Sinhala tokenization is showcased using the 'SinhalaTokenizer' class. Tokenization is a fundamental step in text preprocessing that involves splitting a paragraph into individual words or tokens. This process is essential for tasks like language understanding, sentiment analysis, and topic modeling. The 'tokenz' variable contains the result of the tokenization process, which allows for more granular analysis and manipulation of the text.

```
[ ]  tokenizer = SinhalaTokenizer()
     tokenz = tokenizer.tokenize(paragraph)
     print(tokenz)

     ['ආහාරමය', 'සියල්ලේම', 'මිල', 'ගණන්', 'කොකරම්', 'ඉහළ', 'ය', 'යන්න', 'අපි', 'සැවොම', 'දන්නා', 'කාරණයකි']
```

Figure 9:Example for use tokenize() function

5.4 Sinhala Text Summarization

In testing the library's capacity to summarize Sinhala articles, it used a variety of Sinhala articles covering diverse topics such as politics, economy, sports, and culture. The

ISSN (online): 2581-3048
Volume 7, Issue 10, pp 97-107, October-2023
https://doi.org/10.47001/IRJIET/2023.710013

library demonstrated its efficiency by extracting key information and creating well-structured summaries. In the quest to find the most optimal model for text summarization, our results illuminated a fascinating interplay between accuracy, speed, and user preferences.

5.4.1 TF-IDF Model

As anticipated, the TF-IDF model stood as a robust baseline, given its simplicity and efficiency. It quickly processed texts, highlighting key terms based on their frequency. However, in terms of qualitative output, the summaries often missed nuanced context or sentiment of the original content, which is a limitation inherent to its statistical nature. While it sufficed for shorter texts or articles with a direct informational tone, for longer and context-rich content, it somewhat faltered.

5.4.2 BERT Model

The BERT model showcased its prowess in understanding context. Summaries derived were coherent and more aligned with the underlying sentiments of the original texts. Nevertheless, it demanded considerable computational resources and time, which made it slightly less appealing for real-time applications. Its bi-directional context comprehension, while powerful, was computationally expensive.

5.4.3 Longformer

Longformer, with its global attention mechanism, excelled in processing extensive texts. It provided comprehensive summaries for long documents, and its capability to attend to broader contexts proved beneficial. Yet, it bore the brunt of being one of the slower models due to its intricate architecture.

5.4.4 DistilBERT (FastBERT)

DistilBERT struck a fascinating balance. While it didn't match BERT's depth in context understanding entirely, it compensated by being significantly faster. For applications where speed was a priority without substantial compromise on quality, DistilBERT emerged as a strong contender.

5.4.5 Fast Longformer

The hybrid approach attempted to merge the depth of Longformer with the speed of DistilBERT. Results displayed its competence in summarizing longer texts faster than the Longformer but at the slight expense of depth. It seemed to be a worthy compromise for applications needing summaries of large texts within shorter timeframes.

Upon presenting the summarized outputs to a diverse group of participants, the feedback was enlightening. Users who required quick, concise summaries for decision-making or information extraction leaned towards DistilBERT. In contrast, researchers and professionals, who needed in-depth summaries, favored the Longformer and BERT.

TF-IDF found its supporters among users who wanted lightweight applications without the complexities of deep learning.

In conclusion, while each model had its own strengths and weaknesses, the "best" model was contingent upon specific user requirements and application scenarios. Future work might look into refining these models further, perhaps integrating their strengths or exploring more hybrid approaches to achieve the elusive balance of speed, accuracy, and depth in text summarization.

ACKNOWLEDGMENT

The completion of this project could not have been possible without the assistance of a lot of individuals, even though each and every person might not be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. However, the group would like to express their deep appreciation and indebtedness particularly for the everyone who supports.

REFERENCES

[1] Abeysekara, D. (2022). Singlish to Sinhala Language Conversion Systems: A Review. International Journal of Natural Language Processing, 10(2), 56-72.

[2] Abeysinghe, R. G., & Abeysekara, D. M. (2019). An Approach for Transliterating Singlish Sentences into Sinhala Text. International Journal of Advanced Computer Science and Applications, 10(4), 77-81.

[3] Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. Proceedings of the International Conference on Learning Representations.

[4] Dharmawardena, K. P., Weerasinghe, W. M. A. D., & Manawadu, U. A. (2018). Machine Learning Approaches for Natural Language Processing. International Journal of Computer Science and Information Security, 16(5), 19-26.

[5] Hettiarachchi, S. A. U., & Dayarathna, M. A. K. (2018). A Hybrid Approach to Build a Rule-based Sinhala to Singlish Transliteration System. In 2018 Moratuwa Engineering Research Conference (MERCon) (pp. 1-6). IEEE.

[6] Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing order into text. In Proceedings of the 2004 conference

ISSN (online): 2581-3048
Volume 7, Issue 10, pp 97-107, October-2023
https://doi.org/10.47001/IRJIET/2023.710013

on empirical methods in natural language processing (pp. 404-411).

[7] Abeysekara, D., Ranasinghe, T., & Jayasena, S. (2021). Rule-Based Transliteration of Singlish to Sinhala. International Journal of Advanced Computer Science and Applications, 12(4), 338-345.

[8] Works Cited "2023-118 / 2023-118." GitLab, gitlab.sliit.lk/2023-118/2023-118. Accessed 10 Sept. 2023.

[9] NLTK. "Natural Language Toolkit — NLTK 3.4.4 Documentation." Nltk.org, 2009, www.nltk.org/.

[10] Numpy. "NumPy." Numpy.org, 2009, numpy.org/.

[11] SinLingua. "සිංLingua: Sinhala Language Data Processing Library." GitHub, 2 Sept. 2023, github.com/SinLingua/documentation. Accessed 9 Sept. 2023.

[12] Singlish to Sinhala Transliteration using Rule-based Approach: Tharindu Abeysekara, Nadeeka De Silva, Srinath Perera, Sandaruwan Wijekoon, & Madhavi Latha. (2021). Singlish to Sinhala Transliteration using Rule-based Approach. IEEE Xplore. Retrieved from https://ieeexplore.ieee.org/document/9660744

[13] Sinhala Unicodes: SLUnicodes. (n.d.). Sinhala Unicodes. Retrieved from https://slunicodes.com/

[14] OpenAI. "OpenAI." OpenAI, 25 Apr. 2019, openai.com/.

AUTHORS BIOGRAPHY

Supun Sameera, a dedicated Software Engineer at Innodata Lanka, is a passionate Python programmer and data science enthusiast contributing to the world of IT.

Sandaruwini Galappaththi, a Junior Software Engineer at WIA Systems Inc., excels in Oracle Database Management and is committed to professional growth in the IT industry.

Binura Yasodya, an Intern Data Scientist at MAS Holdings, actively contributes to data science with a strong interest in Python programming and machine learning.

Supun Sarada Wijesinghe, a Junior Data Engineer, brings a wealth of knowledge and strong dedication to Python programming, data pipeline development, and deep learning.

Dr. Anjalie Gamage, a Senior Lecturer at the Sri Lanka Institute of Information Technology, is a highly accomplished academic professional with expertise in Computational Linguistics, NLP, AI, and E-Learning.

Bhagyanie Chathurika, the Academic Coordinator at the Sri Lanka Institute of Information Technology, Matara Center, is a dedicated lecturer with a keen interest in Machine Learning, Deep Learning, and Image Processing, actively contributing to the advancement of technology and computer science.

Citation of this Article:

Supun Sameera, Sandaruwini Galappaththi, Sarada Wijesinghe, Binura Yasodya, Anjalie Gamage, Bhagyanie Chathurika, "SinLingua: Python Library for Sinhala Data Processing" Published in *International Research Journal of Innovations in Engineering and Technology - IRJIET*, Volume 7, Issue 10, pp 97-107, October 2023. Article DOI https://doi.org/10.47001/IRJIET/2023.710013

****

