Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2020, 7(5):383-386**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Converting Non-English Languages to System Readable Format using Python

Maheswara Reddy Basireddy

Email id: maheswarreddy.basireddy@gmail.com

Abstract In today's globalised society, the capacity to process and comprehend text in several languages is essential for many applications and systems. There is a rising demand for efficient ways to translate non-English text into a format that computer systems can handle due to the volume of data being created in several languages. For natural language processing (NLP), machine translation, and language identification, Python provides a robust ecosystem of libraries and tools. This study investigates the usage of NLP libraries like spaCy and NLTK (Natural Language Toolkit) for text processing and conversion to system-readable forms, as well as Python modules like langdetect and translate for language detection and machine translation, respectively.

Keywords natural language processing, spaCy, NLTK

Introduction

There is an enormous quantity of data being created in many languages as a result of the globalisation of information and the growing usage of the internet. For computer systems and applications that need to analyse and comprehend text in many languages, this linguistic variety presents difficulties. The development of techniques and instruments that translate non-English material into a computer system-readable format is required to overcome this difficulty.

Python is a well-liked and flexible programming language that provides a robust ecosystem of modules and tools for natural language processing (NLP), machine translation, and language recognition. The creation of multilingual systems and applications is made possible by these libraries, which may be used to translate non-English text into a format that is readable by the system.

This study investigates the usage of NLP libraries like spaCy and NLTK (Natural Language Toolkit) for text processing and conversion to system-readable forms, as well as Python modules like lang detect and translate for language detection and machine translation, respectively. An overview of these libraries' features and applications—including how to transform non-English text into forms that are readable by systems—is given in this article. The article also provides code snippets and real-world use cases to demonstrate how these libraries might be implemented.

Language Detection

Finding out the language in which the content is written is crucial before attempting to translate or analyze it. Langdetect is one of the language detection packages available in Python.

A. Langdetect

Based on the efforts of Nakatani Shuyo [1], the langdetect library is a Python adaptation of the Node.js language detection module. It determines the language of a given text using a naïve Bayesian filter. The following demonstrates the use of Langdetect:

```
from langdetect import detect

text = "이것은 한국어 텍스트입니다."
lang = detect(text)
print(lang) # Output: ko
```

The language of the provided text is ascertained in the example above by utilising the detect function. The function yields the ISO 639-1 code corresponding to the identified language, 'ko' in this instance denoting Korean.

The langdetect package is based on Nakatani Shuyo's work, which determines a text's language using a naïve Bayesian filter. The frequency of n-grams, or sequences of n characters, is used by the library, which has been trained on a sizable corpus of text in many languages, to identify the language [1]. The library is a well-liked option for Python language identification jobs due to its accuracy and efficiency.

B. Limitations and Challenges

The langdetect library has some drawbacks and difficulties, despite its effectiveness in identifying the language of a given text. The library's potential inability to reliably identify languages that are underrepresented in the training corpus is one of its limitations. Additionally, writings that move between codes or are multilingual may be difficult for the library to access.

The inability of the library to discriminate between languages that are closely related, like Spanish and Portuguese or Danish and Norwegian, presents another difficulty. In some situations, further language-specific processing might be needed in order to recognize the language correctly.

Machine Translation

The next stage is to translate the text into a language that the system can comprehend, usually English, after the language of the text has been recognized. Translate is one of the machine translation libraries available for Python.

A. Translate

A free and open-source Python module called translate gives users access to a number of translation services, such as Yandex, Microsoft Translator, and Google Translate. Interpret [2]. Here's an illustration of how to utilise Google Translate's translate feature:

```
from translate import Translator

Translator = Translator(to_lang="en")
text = "이것은 한국어 텍스트입니다."
translation = translator.translate(text)
print(translation)

# Output: This is a Korean text.
```

In the above example, we create a Translator object with the target language set to English ('en'). We then use the translate method to translate the given Korean text to English.

The translate library provides a convenient interface for accessing various machine translation services. It supports multiple translation services, allowing users to choose the service that best suits their needs. Additionally, the library provides options for customizing the translation process, such as specifying the source and target languages, and handling additional parameters specific to the translation service.

B. Limitations and Challenges

Although machine translation libraries such as translate provide a practical means of translating text, they are not without restrictions and difficulties. The precision and calibre of the translations, which differ based on the translation provider and the intricacy of the source material, is one drawback. Idiomatic phrases, cultural allusions, and context-specific subtleties can be difficult for machine translation systems to interpret, which might result in erroneous or lacking translations.

Managing language-specific aspects like word order, writing systems, and grammatical norms is another difficulty. Text that adheres to multiple language traditions may not always be correctly translated by machine translation systems, producing translations that are grammatically wrong or lack semantic coherence.

Furthermore, the languages that machine translation libraries support and the longest text that can be translated may be restricted. Certain language pairings might not be supported by translation services, and they could have limits on the amount of text that they can handle.

Natural Language Processing (NLP)

The next stage is to analyse and transform the text into a format appropriate for additional analysis or processing after translating it into a language that the system can comprehend. For this, Python offers a number of NLP libraries, like spaCy and NLTK (Natural Language Toolkit).

A. NLTK (Natural Language Toolkit)

One of the best platforms for writing Python programmes that interact with data in human language is NLTK [3]. It offers user-friendly interfaces for handling a range of natural language processing (NLP) operations, including parsing, tokenization, stemming, tagging, and semantic reasoning. Here's an illustration of how to tag parts of speech and tokenize data using NLTK:

```
import nltk
text = "This is a Korean text."
tokens = nltk.word_tokenize(text)
tagged = nltk.pos_tag(tokens)
print(tagged)

 # Output: [('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('Korean', 'JJ'), ('text', 'NN'), ('.', '.')]
```

Using the word tokenize function, we first tokenize the text in the example above into individual words. The pos_tag function is then used to execute part-of-speech tagging on the tokens, giving each token a part-of-speech tag.

Numerous NLP resources and tools, including as corpora, tokenizers, stemmers, taggers, parsers, and tools for semantic reasoning, are available through NLTK. It is extensively employed in a variety of industrial applications, as well as in academic and research contexts. For text processing applications including named entity identification, sentence segmentation, tokenization, and part-of-speech tagging, NLTK is very helpful.

B. Spacy

A free, open-source toolkit for sophisticated NLP in Python is called spaCy [4]. With its production-ready design, it offers precise syntactic analysis and excellent performance. Here's an illustration of how to utilise named entity recognition, tokenization, and part-of-speech tagging using spaCy:

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "This is a Korean text."
doc = nlp(text)

for token in doc:
print(token.text, token.pos_, token.ent_type_, token.ent_iob_)
```

Using the spacy.load method, we first load the en_core_web_sm language model in the example above. After that, we feed the text to the nlp object to generate a Doc object. Lastly, we loop through the tokens in the Doc object, printing for each token the named entity type, partof-speech tag, token text, and named entity IOB (Inside, Outside, Beginning) tag.

In a variety of natural language processing (NLP) tasks, including as text categorization, dependency parsing, named entity identification, tokenization, and part-of-speech tagging, spaCy is renowned for its exceptional performance and accuracy.

Conclusion

This study investigated how to translate non-English material into a format that is readable by a computer using Python tools for language recognition, machine translation, and natural language processing (NLP). The langdetect library was introduced for language identification, which determines the language of a supplied text using a naïve Bayesian filter. For machine translation, the translate library was added, giving users access to a number of translation providers, including Yandex. Translate, Microsoft Translator, and Google Translate.

Additionally, the article covered two well-known Python NLP libraries: spaCy and NLTK (Natural Language Toolkit). For a wide range of NLP activities, such as tokenization, stemming, tagging, parsing, and semantic reasoning, NLTK provides an extensive collection of tools and resources. However, spaCy is well-known for its great performance and is intended for more complex NLP jobs.

The implementation of these libraries for converting nonEnglish text to formats that are readable by the system was demonstrated in the paper through the use of real-world use cases and code samples. Language identification, machine translation, tokenization, part-of-speech tagging, and named entity recognition were all addressed in the examples.

Although these libraries provide strong tools for translating non-English literature, the study also outlined several drawbacks and difficulties with each strategy. Idiomatic phrases, cultural allusions, and context-specific subtleties can cause machine translation systems to generate erroneous or partial translations, whereas language recognition libraries may have trouble translating texts in many languages or those that are closely related.

For academics, developers, and practitioners dealing with multilingual data and applications, this article provides an extensive guidance. With the help of the libraries and methods covered in this paper, systems may be constructed.

Reference

[1]. Shuyo, Nakatani. "Language Detection Library for Java." GitHub, https://github.com/shuyo/language-detection.

[2]. Artur Karapetyan. "Py-Translate." GitHub, https://github.com/teraly/py-translate.

[3]. Steven Bird, Ewan Klein, and Edward Loper. "Natural Language Processing with Python." O'Reilly Media, Inc., 2009.

[4]. Honnibal, Matthew, and Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing." (2017)