Justin Quach

CS-410

10/06/2022

Technology Review

When the Apache development community was founded initially in 2004, there was a need for a toolkit that could handle text regardless of scale. With this need, the community developed the Apache OpenNLP, which is an open-source project and a versatile toolkit that mainly focuses on text mining and processes natural language text. Some of OpenNLP's functions include "tokenization, sentence segmentation, part-of-speech tagging, named extraction, chunking, parsing, and coreference resolution" (Manual). OpenNLP also provides an application program interface (API) and a command line interface (CLI) to provide both back-end and front-end functionalities, mainly for making experiments convenient and training text data because of OpenNLP's ability to make use of newer models; older NLP toolkits simply do not have the capacity to handle newer models. The Apache OpenNLP provides a wide range of tools to automate and classify certain tasks, and the main OpenNLP functions will be described below in detail.

To start using OpenNLP, the setup must be done first. The command prompt script **opennlp** should be opened first, as this is the unique entry point to access all the other tools in the toolkit. There needs to be a model loaded and the tool of choice (sentence segmentation, parsing, etc.), which can be invoked as such: **opennlp [Toolname] [model.bin]**. The opennlp command also has optional routing for parsing input to the model.bin, and parsing output to a text file. Currently, the OpenNLP toolkit supports "training NLP models that can be used by OpenNLP… OpenNLP supports ONNX models via the ONNX Runtime for the Name Finder and Document Categorizer. This allows models trained by other frameworks such as PyTorch and Tensorflow to be used by OpenNLP" (Manual). The choice in compatible models with the opennlp command is quite sufficient for performing and processing many different texts. However, the ONNX models must be created outside of the OpenNLP toolkit, as this toolkit does not support creating these models directly.

Tokenization is one of the many functions that OpenNLP provides to segment input character sequences into tokens. Tokens can be strings, singular characters, punctuation, and numerical values, to name a few. These tokens can be split using whitespace or certain characters to segment sentences. OpenNLP provides tokenization through their tokenizer API, and their two applicable tools: the Simple Tokenizer and a learnable tokenizer; given a basic text file and a .bin model. One factor to note is that "the tokenizer offers two tokenize methods, both expect an input String object which contains the untokenized text. If possible, it should be a sentence, but depending on the training of the learnable tokenizer this is not required" (Manual). These tokenizer toolkits also provide the ability to train the models, which can be converted into a training format that can be used directly. The token can also be detokenized using a Detokenizing API to undo the tokenization process of a trained tokenizer, and the detokenized token can be separated using a split character. The tokenization and detokenization processes make the process of creating tokens intuitive and sufficient for those who require OpenNLP's unique tokenization methods.

Part-of-speech tagging is also one of the many functions that OpenNLP provides, marking tokens with their corresponding word type given the token itself and the context of that token. To note, a token can have one or more tags attached to itself. Part-of-speech tagging assigns a specific tag to a word in the corpus and is used to describe terms within the text. To utilize OpenNLP's tagging features, invoke the command: **opennlp POSTagger [pos_model].bin**, which reads a tokenized sentence line-by-line from standard input. The tagger can also be embedded into an application via its API and can be trained on annotated material. With the use of an evaluation tool, we can also evaluate the accuracy of each POS tag attached. This is especially helpful for categorizing certain tokens to be referred to with something like a tag dictionary. "The tag dictionary is a word dictionary which specifies which tags a specific token can have. Using a tag dictionary has two advantages, inappropriate tags cannot be assigned to tokens in the dictionary and the beam search algorithm must consider less possibilities and can search faster." (Manual). This dictionary makes it easy to search and find certain terms through its quick look-up times.

Chunking is also possible with Apache OpenNLP. The concept of chunking is to divide text syntactically into certain categories (for example, noun, verb, and adjective groups). This essentially allows us to grab required phrases and exclude the extraneous details. Chunking can be done via the Chunking API, which loads the chunker model into memory from disk or from another data source. Once the chunker is ready, it can now tag data. Like tagging from before, “it expects a tokenized sentence as input, which is represented as a String array, each String object in the array is one token, and the POS tags associated with each token” (Manual). Like the other tools within OpenNLP, chunking can also be trained for custom-built data using the Chunking Training API. For instance, training chunking so that it can divide text on new corpus for further analyzation is a possible, viable strategy. The other advantage to using chunking in OpenNLP is to perform a 10-fold cross validation of the chunker (in the statistical sense), and contains commands that trains parameters within a file, insists on a cutoff, number of folds, and misclassification, to name a few. Chunking in Apache OpenNLP has a variety of built-in tools to train and evaluate models using data.

Within OpenNLP, the concept of a parser returns a parse tree from a sentence, breaking each word or phrases into each individual category; this essentially breaks down the sentence into its rudimentary version, so that others can understand a sentence’s syntactical structure. The Parser, like the previous components, is capable of training and evaluating data. To train a dataset, invoke the command **opennlp ParserTrainer** and append different rules (for instance, the parameter file itself, the cutoff number of times a feature for the model should be seen). The training API require four steps: “A HeadRules class needs to be instantiated: currently EnglishHeadRules and AncoraSpanishHeadRules are available; the application must open a sample data stream; call a parser train method: this can be either the CHUNKING or the TREEINSERT parser; save the ParseModel to a file” (Manual). In terms of evaluating, the model must be loaded and a Parse ObjectStream must also be created as well; the evaluation can be tested on pre-trained models, a test dataset or cross validation. Parsing is one of the features available for use in Apache OpenNLP that helps to parse sentences into individual syntactical words.

From the analysis, Apache OpenNLP is community-driven by developers that desired to efficiently automate certain natural language processes, which include tokenization, POS-tagging, chunking, and a parser. Each tool within the toolkit was able to effectively train and evaluate data for statistical purposes, and efficiently performed their intended tasks. Apache OpenNLP continues to receive updates on these NLP tasks, with new developers seeking interest in joining this community, and perhaps, new methods of how to conduct and automate new NLP tasks will come in the distant future.

References

https://opennlp.apache.org/docs/2.0.0/manual/opennlp.html#tools.tokenizer.introduction

https://en.wikipedia.org/wiki/Apache_OpenNLP

https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba

https://opensourceconnections.com/blog/2022/06/27/using-modern-nlp-models-from-apache-opennlp-with-solr/

https://www.analyticsvidhya.com/blog/2021/10/what-is-chunking-in-natural-language-processing/

https://medium.com/analytics-vidhya/text-classification-based-on-apache-open-nlp-317f82f4b1b0