Tech Review of Apache OpenNLP

Introduction:

Apache OpenNLP is a machine learning toolkit. It is used to process natural language text. For example, part-of-speech tagging, parsing, language detection, and other NLP tasks. This tech review includes an introduction to existing Apache OpenNLP features. Besides, it also analyzes some advantages and disadvantages of Apache OpenNLP. At the same time, this tech review also includes predictions for future Apache OpenNLP development.

Body Part:

The structure of Apache OpenNLP includes Sentence Detector, Tokenizer, Language Detector, Name Finder, chunker, Part-of-Speech Tagger, Document Categorizer, Lemmatizer, etc. These components of Apache OpenNLP can perform natural language processing tasks, train models, evaluate models, and so on. Besides, Apache OpenNLP provides the command line interface (CLI). It makes it convenient to use common commands to access all the tools, and is also easy to set up the parameters. There are several useful common commands. (Example from: https://opennlp.apache.org/docs/1.8.3/manual/opennlp.html) The first command is "$opennlp": prints all tools available in this library. The second command is "$opennlp ToolName model_name.bin" which uses the tool and loads the model, and the input and output of the model must be in the standard format. If we need to select the input and output files, add "<input.txt> output.txt" after this command line to point directly to the input and output files. For setting up the number of iterations or specifying some data locations to be read during the training process, we need to add some keywords to the command line to point to, such as: "-model," "-encoding" and so on. Apache OpenNLP provides many pre-trained models that can support multiple languages and can be used with some OpenNLP components (https://opennlp.apache.org/models.html). This can help save much time.

The language detector classifies the language documents according to the function of the model. Vocabulary is extracted in the context of normalized text by using the n-gram algorithm. Our default size is 1-3 words. However, Language Detector allows using LanguageDetectorFactory to customize n-gram size. We need to pay attention to the training text that each row in the training text represents a row of data. The first column represents the ISO-639-3 code, the code means the specific language. The second column is the important core text after the space. When training, the data set needs to have multiple different text information; otherwise, the training will report an error. The training has several steps. Firstly, we need to load the training data into the LanguageDetectorSampleStream. We need to pay attention that the content of each column of text may contain space characters. The second step is to define the training parameters, such

as the number of iterations, and the cutoff value, where ALGORITHM_PARAM can define the algorithm. The third step uses the train function to train the model. The fourth part uses the model to make predictions and simultaneously evaluates the model.

Sentence Detector can check if punctuation is at the last position of a sentence, but it cannot determine where a sentence ends or begins based on its content. We also can train the API which is provided by the Apache OpenNLP to implement sentence detectors. There are several steps for training. Firstly, we need to open a sample data stream to get the train data. Then we used the SentenceDetectorME.train function to train the model and save the model directly for evaluation. OpenNLP's tokenizer will split input characters into words, punctuation, and numbers. Moreover, the tokenizer will take them as tokens. Tokenization has two steps. The first step is to identify the first position and last position of a sentence. The second step is to identify the tokens in the sentence. OpenNLP provides many different tokenizers, such as Whitespace Tokenizer/Simple Tokenizer/Learnable Tokenizer. These three tokenizers label whitespace, the characters that match the token, and the boundaries of the sentence. Besides, we can use the detokenizer method to undo the tokenization.

If you want to find the name entities inside a text file, the Name Finder is Useful. The name entities include people, numbers, locations, dates, etc. There are many pre-trained models provided by OpenNLP, which are useful for us to use such as en-ner-location.bin, en-ner-person.bin, en-ner-percentage.bin, and en-ner-date.bin, etc.(Example from: https://blog.thedigitalgroup.com/open-nlp-name-finder-model-training) In order to train the model, we can use the command line provided by OpenNLP to train the model. Firstly, provide training data with one sentence per line. The training data of the Name Finder model has a requirement. The requirement is that there should only have one sentence for each line in the file. However, another format can also be used, but the sentence must be tokenized and marked with entities. Examples of command for training is "$ opennlp TokenNameFinderTrainer -model en-ner-location.bin -lang en -data en-ner-location.train -encoding UTF-8".(Example from: https://opennlp.apache.org/docs/1.8.3/manual/opennlp.html) Another way we can train the API which OpenNLP provides. The training method is similar to other models. Firstly, read the data from the training file. Secondly, use the NameFinderME.tarin function to train the model. Finally, use TokenNameFinderModel to store the model as a file.

Document Categorizer can classify text into several categories which are predefined by the users. Because classification works for different specific needs, OpenNLP does not have pre-trained models. (Example from: https://opennlp.apache.org/docs/1.8.3/manual/opennlp.html) The

following is an example of the model training command "$ opennlp DoccatTrainer -model en-doccat.bin -lang en -data en-doccat.train -encoding UTF-8" which uses the Maximum Entropy Model in OpenNLP.

Part-of-Speech Tagger performs tagging by checking the part-of-speech type according to the tag itself and the text meaning environment. Generally, a tag contains multiple pos tags. OpenNLP will use the pos tagger to find the most suitable pos tag based on the probability model. We should pay attention here that using a tag dictionary can improve efficiency and running speed. Here is an example of a pos tagger with part-of-speech tags derived from the Penn Treebank. NNP means proper noun/singular. VBZ means Verb/3rd Person singular present. CD means Cardinal Number. NNS means Noun, Plural. JJ means Adjective.(Example From: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html)

Lemmatizer will be set as a dictionary by parsing the given part-of-speech tag. Generally, a token may be derived from multiple words or phrases. Lemmatizer can be implemented using OpenNLP's Lemmatizer tool. Another method is LemmatizerTrainerME. This tool can train models on various text resources. Chunker can divide the text into related parts of different sentence grammar, such as noun phrases and verb phrases. OpenNLP provides a pre-trained model that can call Chunker Tool through the command line: ChunkerME. Another disadvantage for the chunker is that the pre-trained model may not be able to support all the languages, which leads to the model may not be able to detect some necessary entities.

Conclusion:

Combining the above research for Apache OpenNLP. Apache OpenNLP is very useful for text detection of documents in different languages. Besides, Apache OpenNLP is suitable for tokenizing vocabulary, etc. However, my research found that some languages are still not supported by the pre-trained models provided by Apache OpenNLP. In the future, I think this tool can continue to improve this aspect.

Citation:

Apache opennlp developer documentation. Accessed November 5, 2022. https://opennlp.apache.org/docs/1.8.3/manual/opennlp.html#tools.langdetect.

web_page_person. “Language Detector Example in Apache Opennlp.” TutorialKart. TutorialKart, November 30, 2020. https://www.tutorialkart.com/opennlp/language-detector-example-in-apache-opennlp/.

“Getting Started with Opennlp 1.5.0 – Sentence Detection and Tokenizing.” David Dearing. Accessed November 5, 2022. https://dpdearing.com/posts/2011/05/opennlp-1-5-0-basics-sentence-detection-and-tokenizing/.

Gole, Sagar. “Open NLP Name Finder Model Training.” October 30, 2015. https://blog.thedigitalgroup.com/open-nlp-name-finder-model-training.

Krishna. “OpenNLP: Document Categorizer Training.” openNLP: Document categorizer Training, January 1, 1970. https://self-learning-java-tutorial.blogspot.com/2015/10/opennlp-document-categorizer-training.html.

Penn Treebank P.O.S. tags. Accessed November 5, 2022. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.