Report for Project 1 of COMP 479 by Silas Kalinowski ID: 40256077

Explanation of the modules and Design Choices:

My pipeline consists of five modules that can be executed in a stand-alone fashion. How to start and use the pipeline is properly explained in the Demo file.

The first module: read_extract reads and extracts the body of each of the articles in the given file. The module consists of a loop that iterates over every line of every file. The articles can be separated by testing whether a line starts with <REUTERS, that means that an article begins, or </REUTERS>, that means that an article ends. In the lines following <REUTERS the whole information of the current article is given. Therefore, by extracting all of the lines after <REUTERS, the module extracts the entire information of every article and stores it in a list named article. The next part of the module consists of two for-loops. The first one iterates over the previously extracted list of articles. The second loop iterates over every letter in the current article. In that loop it is tested whether a tag begins by testing if a letter is '<'. If that is the case another if-clause checks whether the body of the article begins. If that's the case the letters after that tag are extracted and temporarily stored as curr_body. The body of the article always ends with the tag </BODY>. Therefore, a second if-clause checks when the body ends and stops the extraction of the letters in that case. Additionally, the extracted letters also are appended to a list which stores all the bodies of the articles. By slicing the curr_body String unnecessary information (every document body ends with Reuter &#3) is removed. The module returns a list which contains the raw text of each article from the corpus.

The second module: tokenize tokenizes a given text. The given text needs to be in a txt file. The module uses the word_tokenize function of NLTK and returns the tokens of the text as a list and an output file in which every token is written in a separate line. If the parameter remove_non_alphanumeric is set to True the module filters out every token which only consists of non alphanumeric characters. Therefore, it assumes that numbers are also valid tokens and are not filtered out.

The third module: make_lowercase makes a given text lowercase. The given text needs to be in a txt file. For the module to work properly the given text should be tokenized and the tokens should be written in separate lines in the given txt. Then the module returns the lowercase text as a List and as an output file in which every lowercased token is written in a separate line.

The fourth module: porter_stemmer stems a given list of tokens. The tokens should be given in a txt file in which every token is written in a separate line. The module iterates over every token of the given tokens and uses NLTK's PorterStemmer to stem every token separately. The module then returns the list of stemmed tokens and an output file in which every token is written in a separate line.

The fifth module: remove_stop_words removes every word in a given list of stop words from a text. The input text should be given as a txt file and be tokenized. Every token of the text should be written in a separate line in the input file. The module reads in the file and iterates over the tokens and checks whether a word is in the given stop words list. If that is not the case it stays in the list of

tokens, otherwise it is removed. Then the remaining words are combined to return the filtered text as a string. The module uses NLTK's list of stop words for the English language as default.

If the given input file path does not exist, the module catches the exception and a message is printed in the console.

Assumptions

I assumed that tokenization also includes filtering out special characters and tokens which only consist of non alphanumeric characters. But the filtering can also be deactivated by setting the parameter remove_non_alphanumeric to False. I also assumed that the input txt file for the module 3, 4 and 5 is in a specific structure. Every token should be written in a separate line for the modules to work properly. When using the pipeline in the given order, the txt file are in the necessary structure automatically. I also assumed that module 1 should only return the raw text of each article. Therefore, other information is not returned, but the module can easily be adjusted to return other information of the articles as well. Also, module 1 only extracts the raw text for articles of the reuters collection because it is based on the articles' structure and does not work for other document structures. Next, module 2, 3, 4 and 5 remove empty tokens or tokens that after filtering/stemming are empty strings.

Testcases:

This assumptions and design choices have advantages and disadvantages that I try to display with my test cases.

The testcases can be found in the testcases folder. The outputs of the modules are in the local directory and are names testcaseNUMBER_moduleNUMBER.txt.

Testcases Module 1: reading and extracting raw text

The first testcase tests whether the module also works for different files than the first five files of the reuters collections. It passed the test and successfully extracted the text of all the articles in reuters21578/reut2-013.sgm. I proofread the result.

The second testcase tests the limits of the module by testing whether a text which has no structure is extracted. This was as expected not the case. The module only extracts text of articles that have the structure of the articles in the reuters collection with a body and reuters tag.

Testcases Module 2: tokenization

The first testcase tests whether words with a hyphen are separated to two tokens. The module does not split words with hyphen and returns them as one token. For further work with the tokens the user should be aware of that.

The second testcases tests whether the module removes punctations and commas as expected.

The third testcase tests how the module processes numbers and special characters. The module removes commas and special characters and returns numbers as tokens.

Testcases Module 3 making text lowercase

The first test case tests if the module functions properly if the input file has the necessary structure.

The second testcase tests how the module processes numbers. The module returns numbers unchanged as tokens.

The third testcase tests how the module processes words that contain special characters. The module lowercases every alphabetic characters and does not change special characters in the words as expected.

Testcases Module 4: stemming

The first testcase tests if the stemmer also works for different languages. I used some German words to test that. The stemmer does not cut the end of German words. It only works properly for English words. The user must be aware of that when using the stemmer. For German text the user should be using a different stemmer.

The second testcase tests the how the stemmer processes numbers. As expected the stemmer does not change numbers at all.

The third testcase tests what happens if the txt file does not have the proper structure. In the test case the words are not written in separate lines. Therefore, the stemmer does not recognize that the file contains three words and stems the three words as if they were one word. The user should therefore only use the stemmer if the file has the necessary structure.

Testcases Module 5: removing stop words

The first testcase tests what happens if the txt file does not have the proper structure. In the test case the words are not written in separate lines. Therefore, the module does not recognize that the file contains three words in the same line and does not remove the first line. For the second line the module works as expected. The user should therefore only use the module if the file has the necessary structure.

The second testcase tests the how the module processes numbers. The module does not remove numbers if they are not stop words and works as expected.

I certify that this submission is my original work and meets the Faculty's Expectations of Originality.

Signature ID: 40256077 Date 2022-09-22