Text Preprocessing Pipeline and Proofreading Results COMP 6751 Project 1 Report

Haochen Zou (40158179)

0. Expectations of originality

I certify that this submission is my original work and meets the Faculty's Expectations of Originality. Name: Haochen Zou; I.D: 40158179; Date: 2021.10.1

1. Introduction

For the project required in the assignment, implemented a text preprocess and proofreading results program. The program allow user to enter a file name from the NLTK Reuters corpora, after user entering the file name, program will get the text file from NLTK corpus and generate a .txt file. The PreProcess.py script will then execute the following steps in pipeline: tokenization, sentence splitting, POS tagging, number normalization, date recognition and date parsing. The proofreading solutions will display to user.

2. Materials and Methods

2.1. Techniques

2.1.1. NLTK

NLTK (Natural Language Toolkit), a natural language processing toolkit, is the most used Python Library in the field of natural language processing. It has its own corpus and part of speech classification database. It has its own classification and word segmentation function.

2.1.2. Reuters Corpus

The Reuters Corpus is a corpus from NLTK corpora, it contains 10,788 news documents totaling 1.3 million words. The documents have been classified into 90 topics, and grouped into two sets, called “training” and “test”.

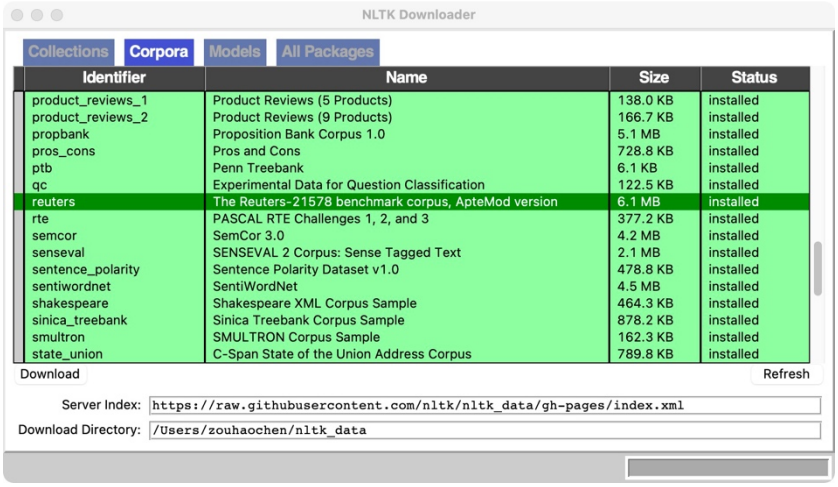


Figure 1. Installed the Reuters corpus from NLTK corpora.

2.1.3. RegexpTokenizer

Regexptokenizer comes from the nltk.tokenize library. Some word segmentation tools are already written rules. If prople want to segment words according to their own rules, they can use regexptokenizer.

2.1.4. Num2words and Words2num

Num2words is a library that converts numbers like 42 to words like forty-two. It supports multiple languages and can even generate ordinal numbers like forty-second.

Words2num inverse text normalization for numbers. Currently it only supports en-US locale.

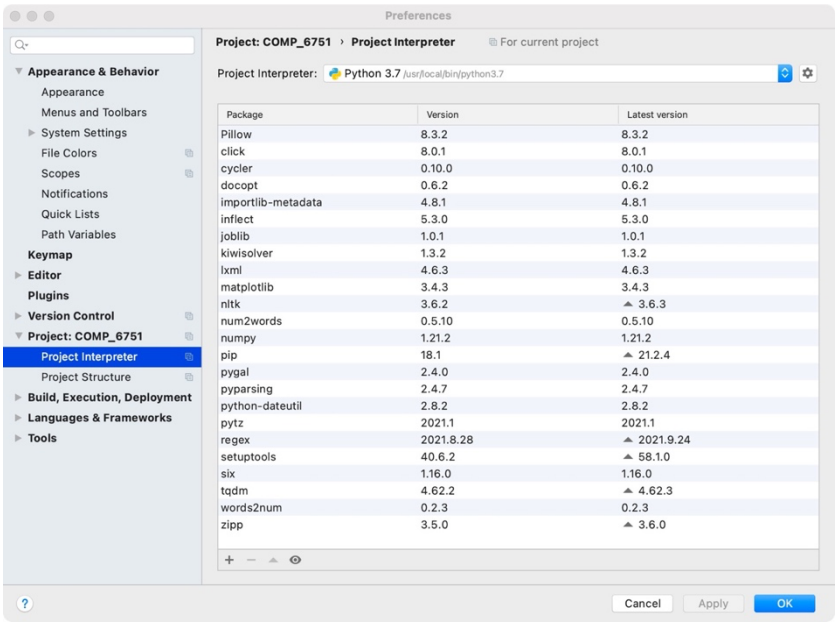


Figure 2. Packages installed for the project.

2.2. Architecture

In the text preprocess and proofreading result program, six operations can be performed:

1. Give tokenization results of the text file.
2. Display the split sentences.
3. Reveal the result of part-of-speech tagging.
4. Show the number normalization from text file.
5. Recognized the date from different formats in the text file.
6. Parse the date information.

Flow chart of the text preprocess and proofreading result program is shown in the last page.

3. Implementation

3.1. Tokenization

For the project requirement, the tokenization start with a regular expression-based tokenizer from NLTK. The project not only used the **basic NLTK's regular expression tokenizer**, but also **improved the tokenizer types** for best result. Information about basic NLTK's regular expression tokenizer show in Figure below from book Natural Language Processing with Python page 111

NLTK's Regular Expression Tokenizer

The function `nltk.regexp_tokenize()` is similar to `re.findall()` (as we've been using it for tokenization). However, `nltk.regexp_tokenize()` is more efficient for this task, and avoids the need for special treatment of parentheses. For readability we break up the regular expression over several lines and add a comment about each line. The special `(?x)` "verbose flag" tells Python to strip out the embedded whitespace and comments.

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

When using the verbose flag, you can no longer use `' '` to match a space character; use `\s` instead. The `regexp_tokenize()` function has an optional `gaps` parameter. When set to `True`, the regular expression specifies the gaps between tokens, as with `re.split()`.

Figure 3. Information about basic NLTK's regular expression tokenizer from book.

The basic and improved regular expression tokenizer is defined as follows:

```
if tokenizer_type == tokenizer_list[0]:
    # basic regular expression tokenizer
    pattern = r'''(?x)                  # set flag to allow verbose regexps
              (?:[A-Z]\.)+              # abbreviations, e.g. U.S.A.
              | \w+(?:-\w+)*            # words with optional internal hyphens
              | \$?\d+(?:\.\d+)?%?      # currency and percentages, e.g. $12.40, 82%
              | \.\.\.                  # ellipsis
              | [][.,;"'?():-_`]        # these are separate tokens; includes ], [
              '''
elif tokenizer_type == tokenizer_list[1]:
    # improved regular expression tokenizer
    pattern = r'''(?x)                          # set flag to allow verbose regexps
              (?:[A-Z]\.)+                      # abbreviations, e.g. U.S.A.
              | \w+(?:-\w+)*                    # words with optional internal hyphens
              | \$?\d+(?:,\d+)?(?:\.\d+)?%?     # currency and percentages and numbers normalization
              | \.\.\.                          # ellipsis
              | [][.,;"'?():-_`]                # these are separate tokens; include ], [
              | \'[sS]                          # possessive nouns
              | \w+                             # word characters
              '''
else:
    raise Exception("ERROR: Tokenizer type【\'" + str(tokenizer_type) + "\'】does not exist in【" + (
        ', '.join(tokenizer_list)) + '】.')

regexp_tokenizer = RegexpTokenizer(pattern)
title_tokens = regexp_tokenizer.tokenize(title)
body_tokens = regexp_tokenizer.tokenize(body)

print('\nText Preprocess and proofreading results display as follows:')
print('\n【Tokenization】')
print(title_tokens)
print(body_tokens)
```

3.2. Sentence Splitting

Sentence splitting as for tokenization, start with a NLTK module. Standard sentence tokenizer **sent_tokenize** return a sentence-tokenized copy of text file using NLTK's recommended sentence tokenizer. In the project, the sentence splitting part is written in the code shown below.

```
body_sentences = nltk.sent_tokenize(body)
print('\n【Sentences Splitting】')
print(body_sentences)
```

3.3. POS Tagging

In this project, A part-of-speech tagger, or POS tagger, processes a sequence of words, and attaches a part of speech tag to each word. Next step is representing tagged tokens and reading tagged corpora. Finally mapping words to properties using python dictionaries. The code about this module is displayed as follows.

```
pos_tags: List[List[str]] = list()
for body_sentence in body_sentences:
    body_tokens = regexp_tokenizer.tokenize(body_sentence)
    body_pos_tags = nltk.pos_tag(body_tokens)
    pos_tags.append(body_pos_tags)
print('\n【POS Tagging】')
print(pos_tags)
```

3.4. Number Normalization

Improve the regular tokenizer which combined the split number segments (finished in 3.1. tokenizer). Created a grammar NUMBER to select number, the grammar contains words with POS tag <CD>. Display the number list to users and convert the numbers to words as well as the words to numbers using packages num2words and words2num. Relevant code is shown below.

```
class NumberNormalization:
    def __init__(self, pos_taggings: List[List[str]]):
        self.pos_tags = pos_taggings
        self.number_normalize: List[str] = list()

    def number_select(self) -> List[str]:
        number_grammar = r"""
                                NUMBER: {<CD>}
                                """
        number = nltk.RegexpParser(number_grammar)

        for i in range(len(self.pos_tags)):
            tree = number.parse(self.pos_tags[i])
            for subtree in tree.subtrees():
                if subtree.label() == 'NUMBER':
                    number_list = subtree.leaves()
                    number_element = number_list[1:]
                    for number in number_element:
                        self.number_normalize.append(''.join(word[0] for word in subtree.leaves()))

        return self.number_normalize

number = NumberNormalization(pos_tags)
number_list = number.unit_detection()
print('\n【Number Normalization】')
print(number_list)
```

3.5. Date recognition

Define format for month and date. Created a date recognition CFG by using the POS tags to detect date information format from the text files. Then check the validate of date information which satisfied the date format defined above. Finally print results to users. Relevant code about this module is displayed below.

```
class DateRecognition:

    def __init__(self, pos_tag_list: List[List[str]]):
        self.pos_tag = pos_tag_list
        self.date_list: Set[str] = set()
        self.month = ('January', 'February', 'March', 'April', 'May', 'June', 'July',
                      'August', 'September', 'October', 'November', 'December',
                      'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')
        self.date_patterns = "(\\d{4}[-/]?\\d{2}[-/]?\\d{2})" \
                             "|(\\w+\\s\\d{1,2}[a-zA-Z]{2}\\s?,?\\s?\\d{4}?)" \
                             "|(the\\s\\d{1,2}[a-zA-Z]{2}\\sof\\s[a-zA-Z]+)" \
                             "|(the\\s\\w+\\sof\\s\\w+)" \
                             "|(\\w+\\s\\d{1,2}[a-zA-Z]{2})" \
                             "|(\\w+\\s\\d{1,2})"
        self.date_regexp = re.compile(self.date_patterns)

    def date_recognition(self) -> Set[str]:
        date_recognition_cfg = r"""
                                DATE:  {<NNP> <CD> <,>? <CD>}       # October 1 2021
                                       {<DT> <NN> <IN> <NNP>}       # the first of October
                                       {<DT> <CD> <IN> <NNP>}       # the 1 of October
                                       {<IN> <NNP> <CD>}            # on October 1
                                       {<NNP> <CD>}                 # October 1
                                       {<IN> <CD>}                  # on 2021
                                       {<IN> <JJ>}                  # on 2021-10-1
                                """
        date_information = nltk.RegexpParser(date_recognition_cfg)

        for i in range(len(self.pos_tag)):
            tree = date_information.parse(self.pos_tag[i])
            for subtree in tree.subtrees():
                if subtree.label() == 'DATE':
                    tokens = [tup[0] for tup in subtree.leaves()]
```

```
            if '/' in tokens or '-' in tokens:
                date = ''.join(ch for ch in tokens)
            else:
                date = ' '.join(word for word in tokens)
            validate = self.date_validate(date, tokens)
            if validate:
                self.date_list.add(date)
        return self.date_list

    def date_validate(self, date_str: str, tokens: List[str]) -> bool:
        for token in tokens:
            if token not in ['/', '-', ','] and not token.isalnum():
                return False

        check = self.date_regexp.findall(date_str)
        if len(check) == 0 or check == []:
            return False

        return True
```

After recognized the date information, create a date parse CFG that takes a date string as input to parse dates. The grammar includes nonterminal DATE, DAY, MONTH, YEAR. Then the words in a date are split and parse. The program will display the parsing result in picture and words, code shown as follows.

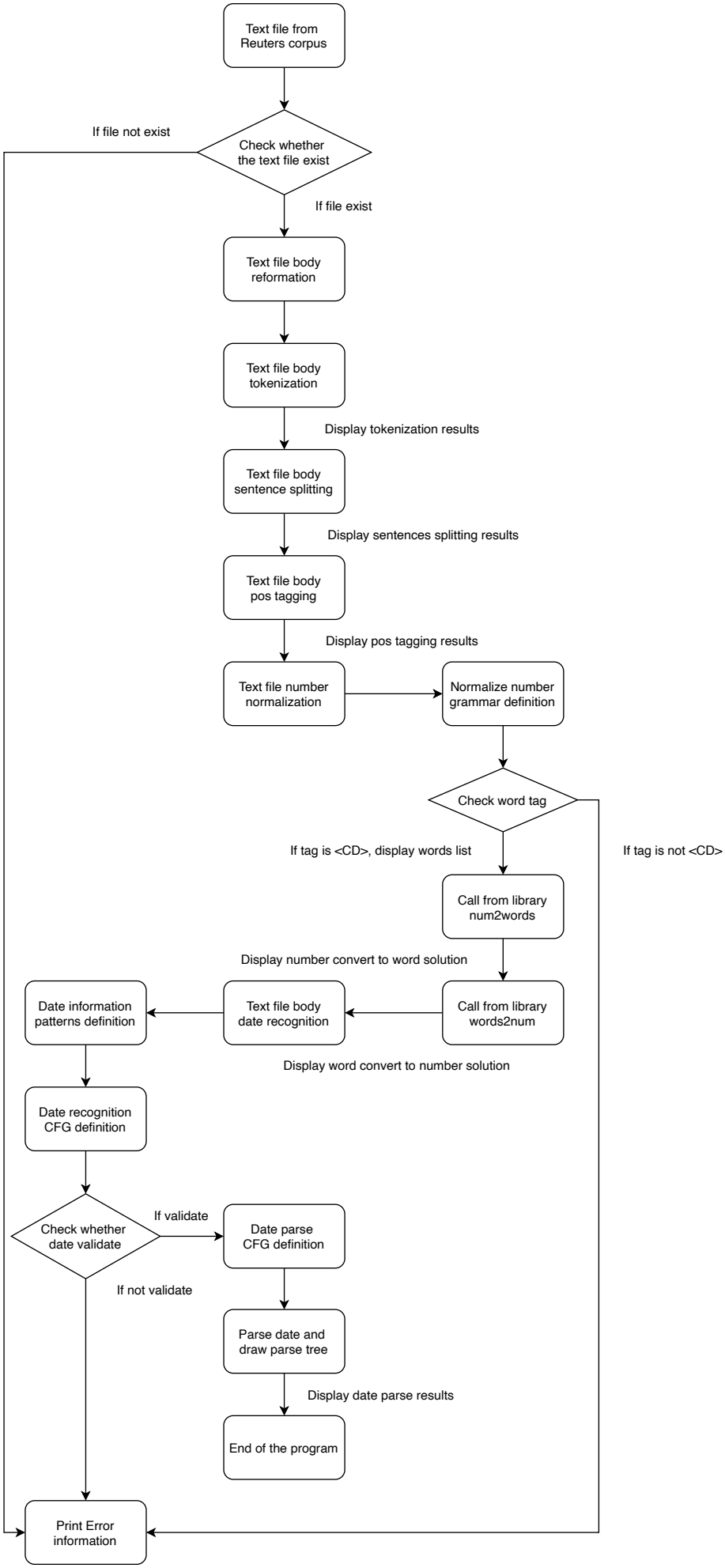
```
def date_parse(text_date: Set[str]):
    date_parse_cfg = nltk.CFG.fromstring(
    """
    DIGIT -> "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
    DATE -> PREP YEAR SEP MONTH_NUM SEP DAY | YEAR SEP MONTH_NUM SEP DAY |
                MONTH_STR DAY SEP YEAR | MONTH_STR DAY | MONTH_STR YEAR | PREP MONTH_STR NN_NUM |
                MONTH_STR NN_NUM YEAR | MONTH_STR NN_NUM SEP YEAR | MONTH_STR NN_NUM |
                DT NN_STR PREP MONTH_STR | DT NN_NUM PREP MONTH_STR | PREP YEAR | MONTH_STR YEAR
    SEP -> "/" | "-" | ","
    YEAR -> DIGIT DIGIT DIGIT DIGIT
    MONTH_NUM -> DIGIT | DIGIT DIGIT
    DAY -> DIGIT | DIGIT DIGIT
    DT -> "the"
    PREP -> "of" | "in" | "on" | "by" | "to" | "from"
    NN_STR -> "first" | "second" | "third" | "fourth" | "fifth" | "sixth" | "seventh" | "eighth" | "ninth" | "tenth" |
                "eleventh" | "twelfth" | "thirteenth" | "fourteenth" | "fifteenth" | "sixteenth" | "seventeenth" |
                "eighteenth" | "nineteenth" | "twentieth" | "twenty-first" | "twenty-second" | "twenty-third" |
                "twenty-fourth" | "twenty-fifth" | "twenty-sixth" | "twenty-seventh" | "twenty-eighth" |
                "twenty-ninth" | "thirtieth" | "thirty-first"
    MONTH_STR -> "January" | "February" | "March" | "April" | "May" | "June" | "July" | "August" | "September" |
                    "October" | "November" | "December"
    NN_NUM -> "1st" | "2nd" | "3rd" | "4th" | "5th" | "6th" | "7th" | "8th" | "9th" | "10th" | "11th" | "12th" | "13th" |
                "14th" | "15th" | "16th" | "17th" | "18th" | "19th" | "20th" | "21st" | "22nd" | "23rd" | "24th" |
                "25th" | "26th" | "27th" | "28th" | "29th" | "30th" | "31st"
    """)
    date_parser = nltk.ChartParser(date_parse_cfg)

    for date in text_date:
        if date.find('/') != -1 or date.find('-') != -1:
            tokens = [ch for ch in date]
        else:
            tokens = []
            for t in date.split():
                if t.isnumeric():
                    tokens.extend([num for num in t])
                else:
                    tokens.append(t)

        for tree in date_parser.parse(tokens):
            print(tree)
            tree.draw()
```

4. Errors and Limitations

Please check the Demo File.

5. Reference

Natural Language Processingwith Python, Steven Bird, Ewan Klein, and Edward Loper
https://pypi.org/project/num2words/
https://pypi.org/project/words2num/


Text file from Reuters corpus
Check whether the text file exist
If file not exist
If file exist
Text file body reformation
Text file body tokenization
Display tokenization results
Text file body sentence splitting
Display sentences splitting results
Text file body pos tagging
Display pos tagging results
Text file number normalization
Normalize number grammar definition
Check word tag
If tag is <CD>, display words list
If tag is not <CD>
Call from library num2words
Display number convert to word solution
Date information patterns definition
Text file body date recognition
Call from library words2num
Display word convert to number solution
Date recognition CFG definition
Check whether date validate
If validate
Date parse CFG definition
If not validate
Parse date and draw parse tree
Display date parse results
End of the program
Print Error information