# Assignment 2

(i) Describe the steps you have performed for data preprocessing.

## 1. Parsing document

In data processing, created a function called `tokenize(input= file)`

- Takes a .sgm file as a input
- Returns a dictionary
- dictionary.keys = $\text{ID}_1, \ldots\ldots, \text{ID}_\text{N}$
- dictionary.values = dictionary(content)$_1, \ldots_\text{N}$
  - content.keys = "title", "body"
  - content.values = [tokens]

### a. Extracting Content

First, gets a match for every news

```
reuter_start = re.compile(
        r"<REUTERS [(A-Z)(\W)]*(\d*)\W NEWID=\W(\d*)\W>(.|\n)*?(</REUTERS>)")
reuter_matches = re.finditer(reuter_start, text)
```

- `finditer()` method only fetches one match at a time which is less memory consuming
- Selected with the Reuters tag since ID is extracted in loop

### b. Document

Then for each match, extracts the corresponding string from starting index and ending index

```
s = match.start()
e = match.end()

document = text[s:e]
```

**ID** In string, selects and assigns ID of the document

```
start_tag = re.search(r"NEWID=\W(\d*)\W>", document)  # LIST
currentID = int(start_tag.group(1))
```

- Before tokenization of each content, extracts stop words as a list from `stopwords.txt`

```
stopwords = open("stopwords.txt", "r").read()
stopwords = stopwords.split("\n")
```

- For splitting, `re.split()` function splits for all the given punctuations

```
r = re.compile(r'[\s{}]+'.format(re.escape(string.punctuation)))
```

- For TITLE and BODY content, try and except statements are used since some news might not have TITLE content and BODY content along with the tags themselves. None-type raises AttributeError. When encountered, the empty content is inserted

**Title**

```
try:
    # Capture title tag and content
    title = re.search(r"(<TITLE>)((.|\n)*?)(</TITLE>)", document)  # LIST
    title_content = title.group(2).lower() # Group 2 --> ((.|\n)*?)
# In case of NO TITLE assign Empty content
except AttributeError:
    title_content = " "
    pass
```

After capturing content, splits the content with word boundaries and digits; then comprehends the list with conditions that it is not a stop word and only containing alphabetic characters

```
title_tokens_raw = re.split(r, title_content) # Slip punctuation
title_tokens = [
    token for token in title_tokens_raw
        if token not in stopwords           # Remove stopword
        ]
```

**Body** Same capturing process and tokenization process in Title content is applied to Body con

```
try:
    # Capture TITLE tag and content
    body = re.search(r"(<BODY>)((.|\n)*?)(</BODY>)", document)  # LIST
    body_content = body.group(2).lower() # Group 2 --> ((.|\n)*?)
# In case of NO BODY assign Empty content
except AttributeError:
    body_content = " "
    pass

body_tokens_raw = re.split(r, body_content) # Slip punctuation
body_tokens = [
    token for token in body_tokens_raw
        if token not in stopwords           # Remove stopword
        ]
```

Then categorized content for each document is add to dictionary items is returned

```
# A dictionary as the key: ID and value: dict{title, body}
news_update = {currentID: {"title": title_tokens, "body": body_tokens}}

# Update All news
news.update(news_update)
```

## 2. Creating Data

First, fetches the files with .sgm format in given directory with `os.listdir()` method

```
# Document List
documents = []

# Get directory as argument
path = sys.argv[1]

document = [os.path.join(path,file) for file in os.listdir(path) if file.endswith(".sgm")]
# Get the current directories of the file
for files in os.listdir(path):
    # Get ".sgm" files
    if files.endswith(".sgm"):
        documents.append(os.path.join(path,files))
```

## JSON

Then file “inverted_index.json” is created and open for writing. The data for inverted_index is stored in dictionary. The dictionary is a hash object in Python with search complexity of O(1). The inverted index is a regular list.

The common approach is the storing inverted indexes as linked lists, yet we have one word query so there is no need for search speed optimization used in merge operations

```
file = "inverted_index.json"
f = open(file, "w", encoding="latin-1")

# Get inverted index for each word
inverted_index = {}
‘# {token: [id....]}
```

After appending dictionary, dictionary object is written on the Json file with `json.dumps()` method

```
x = json.dumps(inverted_index, indent= 1, ensure_ascii=False)
f.write(x)
f.close()
```

## Trie

Then Trie object is created.

```
_dictionary = Trie()
```

Trie Node has an instance named `self.children` which is dictionary object that holds the children nodes of that node. And `self.isend` is the instance which shows whether appended words last char is this node, in default False.

```
class Trie_Node:
    def __init__(self):
        self.children = {}
        self.isend = False
```

The root of Trie is an empty Node. `insert()` requires to traversing top to bottom until the end of the given word is reached and creating nodes in dictionary with the key: letter and value: {node...}

```
class Trie:
    def __init__(self):
        self.root = Trie_Node()

    def insert(self, word: str):
        current_node = self.root

        for letter in word:
            if letter not in current_node.children:
                current_node.children[letter] = Trie_Node()
            current_node = current_node.children[letter]
        current_node.isend = True
```

`search()`, starts from the root and traverse top to bottom while letter in the given string has a matching node key-value pair. Returns a Boolean.

```
    def search(self, word: str):
        current_node = self.root

        for letter in word:
            if letter not in current_node.children:
                return False
            current_node = current_node.children[letter]

        if current_node.isend == True:
            return True
        else:
            return False
```

Two methods are used for getting the prefix end node and its children, namely `startWith()`, and getting the all possible words start with the prefix, namely `PrefixSearch()`.

- `startWith()` method gets an input prefix as a string and returns the node that prefix search starts with traversing in tree.
- `PrefixSearch()` method first get the last node comes from prefix and traverse with the `findEach()` method which takes the current node as the starting point and the prefix to concatenate the traverse keys of each traversal. `findEach()` method appends the word list in the enclosing scope.

```
    def startWith(self, prefix: str):
        # -i: prefix: str
        # -o: Bool
        current_node = self.root

        for letter in prefix:
            if letter not in current_node.children:
                return False

            current_node = current_node.children[letter]

        return current_node

    def PrefixSearch(self, prefix: str):
        # -i: prefix: str
        # -o: word_list: list

        word_list = []

        current_node = self.startWith(prefix)

        # Traverse the tree and Finds a word, appends the word list
        def findEach(node, word):
            if node.isend:
                word_list.append(word)

            for a, n in node.children.items():
                findEach(n, word + a)

        findEach(current_node, prefix)

        return word_list
```

After appending trie object with all the tokens, the object is pickled and written on the `trie.pickle` file.

```
# Pickle trie and write on pickle file
trie_pickle = open("trie.pickle","wb")
pickle.dump(_dictionary, trie_pickle)
trie_pickle.close()
```

For each document, tokenization method is called and returned dictionary is searched. When encountered with a token; tries append the ID list, if encounters a KeyError, creates a new word and ID list

```
for token in in_content:
```

```
# Try to append the dictionary_items.IDlist
    try:
        if inverted_index[token][len(inverted_index[token])-1] != ID:
        inverted_index[token].append(ID)

        # If key is not found append inverted_index
        except KeyError:
            inverted_index[token] = [ID]
            _dictionary.insert(token)
```

## Preprocessing Execution Times

Tokenization –> Average: 23-25 sec

JSON Dumping –> 1 sec

Pickling –> 1 sec

Total –> 25 sec

After Preprocessing json formatted inverted index and pickled trie is loaded in `query.py`

## Examples

Enter input: ban

[235, 540, 659, 893, 933, 959, 991, 1520, 1710, 1720, 1730, 1911, 2739, 2936, 2942, 2957, 3112, 3998, 4046, 4124, 4191, 4757, 4762, 4793, 4923, 5009, 5031, 5088, 5542, 5852, 5887, 5891, 5892, 5964, 6411, 6823, 6979, 7771, 7952, 7972, 8400, 8656, 9518, 9532, 9582, 9629, 10055, 10101, 10334, 10697, 11075, 11214, 11720, 11769, 11883, 12729, 12794, 13246, 13253, 13649, 14419, 14839, 14944, 14957, 15322, 15798, 15806, 16095, 16748, 16765, 16769, 16784, 16943, 17273, 17650, 17668, 17962, 18554, 19165, 19425, 19555, 19852, 20250, 20439, 20989]

Enter input: tur*

[28, 45, 178, 223, 248, 276, 295, 331, 334, 335, 342, 372, 389, 444, 627, 635, 652, 730, 748, 809, 835, 862, 894, 899, 920, 922, 936, 1019, 1074, 1081, 1097, 1252, 1254, 1478, 1480, 1498, 1539, 1579, 1580, 1611, 1685, 1696, 1749, 1808, 1867, 1911, 1917, 1927, 1935, 1959, 1967, 1975, 2001, 2025, 2034, 2084, 2137, 2190, 2223, 2246, 2318, 2400, 2475, 2495, 2514, 2517, 2522, 2552, 2662, 2666, 2758, 2759, 2765, 2768, 2900, 2960, 2961, 2962, 2963, 2968, 2970, 3004, 3076, 3090, 3188, 3217, 3269, 3273, 3432, 3434, 3449, 3454, 3456, 3462, 3463, 3468, 3480, 3505, 3517, 3518, 3528, 3539, 3563, 3597, 3613, 3635, 3640, 3701, 3745, 3759, 3792, 3798, 3818, 3837, 3841, 3938, 3966, 4002, 4017, 4026, 4045, 4057, 4081, 4085, 4089, 4092, 4094, 4100, 4106, 4108, 4123, 4132, 4137, 4196, 4202, 4223, 4226, 4266, 4286, 4328, 4470, 4562, 4574, 4610, 4625, 4636, 4647, 4649, 4661, 4662, 4691, 4692, 4713, 4727, 4736, 4755, 4756, 4776, 4809, 4860, 4897, 4916, 4931, 4946, 5022, 5038, 5130, 5150, 5157, 5163, 5171, 5180, 5205, 5206, 5210, 5219, 5221, 5225, 5230, 5244, 5248, 5250, 5257, 5264, 5265, 5267, 5274, 5278, 5376, 5386, 5391, 5394, 5395, 5410, 5432, 5456, 5514, 5515, 5527, 5561, 5574, 5630, 5655, 5656, 5715, 5719, 5726, 5761, 5769, 5787, 5879, 5888, 5891, 5978, 5985, 6112, 6280, 6328, 6335, 6346, 6349, 6358, 6363, 6372, 6395, 6400, 6472, 6481, 6521, 6633, 6706, 6722, 6870, 6896, 6935, 6949, 6972, 6999, 7003, 7032, 7051, 7075, 7102, 7103, 7105, 7155, 7254, 7280, 7319, 7366, 7540, 7552, 7578, 7593, 7649, 7654, 7740, 7757, 7797, 7811, 7825, 7871, 7892, 7934, 7951, 7966, 8029, 8038, 8085, 8087, 8110, 8125, 8136, 8155, 8178, 8191, 8192, 8193, 8434, 8435, 8439, 8441, 8510, 8578, 8611, 8614, 8615, 8618, 8619, 8627, 8644, 8671, 8743, 8839, 8884, 8902, 8905, 8944, 8961, 8963, 8980, 9048, 9096, 9127, 9139, 9158, 9160, 9175, 9177, 9180, 9187, 9189, 9229, 9244, 9248, 9302, 9374, 9537, 9603, 9638, 9658, 9662, 9706, 9707, 9728, 9734, 9739, 9741, 9759, 9760, 9761, 9763, 9769, 9778, 9784, 9819, 9831, 9848, 9908, 9926, 9969, 9974, 9977, 10080, 10106, 10111, 10123, 10133, 10168, 10175, 10194, 10228, 10252, 10255, 10278, 10280, 10282, 10317, 10325, 10328, 10348, 10367, 10395, 10433, 10452, 10511, 10522, 10539, 10548, 10588, 10595, 10605, 10611, 10612, 10617, 10620, 10621, 10627, 10630, 10641, 10642, 10648, 10667, 10668, 10688, 10694, 10699, 10700, 10703, 10715, 10739, 10743, 10744, 10778, 10797, 10862, 10863, 10905, 10910, 10915, 11056, 11118, 11124, 11167, 11186, 11205, 11221, 11227, 11243, 11252, 11254, 11279, 11404, 11413, 11472, 11512, 11535, 11654, 11661, 11714, 11763, 11768, 11773, 11781, 11793, 11795, 11797, 11810, 11823, 11827, 11830, 11833, 11837, 11840, 11854, 11870, 11885, 11887, 11894, 11918, 11944, 11987, 12123, 12135, 12180, 12292, 12311, 12320, 12328, 12330, 12349, 12396, 12410, 12432, 12463, 12479, 12484, 12522, 12524, 12598, 12701, 12709, 12720, 12759, 12799, 12806, 12815, 12848, 12877, 12878, 12905, 12909, 12917, 12943, 12948, 12983, 12994, 13012, 13052, 13073, 13123, 13129, 13179, 13244, 13250, 13251, 13259, 13314, 13331, 13350, 13379, 13406, 13419, 13458, 13501, 13561, 13606, 13629, 13662, 13689, 13752, 13807, 13832, 13907, 13919, 14005, 14051, 14068, 14132, 14190, 14196, 14266, 14279, 14287, 14419, 14454, 14588, 14609, 14698, 14749, 14799, 14839, 14857, 14860, 14862, 14863, 14869, 14874, 14880, 14881, 14890, 14912, 14925, 14930, 15014, 15063, 15085, 15194, 15200, 15254, 15338, 15369, 15372, 15374, 15380, 15381, 15386, 15387, 15390, 15397, 15421, 15427, 15438, 15551, 15556, 15639, 15766, 15781, 15884, 15894, 15957, 16009, 16028, 16072, 16086, 16113, 16116, 16137, 16140, 16141, 16159, 16175, 16181, 16191, 16207, 16217, 16220, 16239, 16268, 16310, 16322, 16352, 16427, 16491, 16504, 16540, 16556, 16602, 16636, 16669, 16736, 16750, 16756, 16773, 16785, 16935, 16948, 16953, 16957, 16961, 17012, 17018, 17058, 17083, 17103, 17148, 17170, 17173, 17201, 17203, 17218, 17243, 17248, 17250, 17252, 17264, 17296, 17305, 17306, 17325, 17329, 17368, 17372, 17396, 17402, 17433, 17474, 17508, 17549, 17553, 17693, 17700, 17731, 17737, 17779, 17810, 17839, 17862, 17884, 17886, 17891, 17896, 17900, 17901, 17915, 17921, 17922, 17937, 17962, 18034, 18061, 18085, 18102, 18138, 18143, 18174, 18231, 18288, 18293, 18350, 18399, 18438, 18467, 18469, 18483, 18495, 18497, 18507, 18514, 18563, 18570, 18676, 18709, 18776, 18796, 18823, 18855, 18888, 18917, 18934, 18990, 18994, 19028, 19038, 19039,

```
19068, 19087, 19157, 19250, 19285, 19353, 19378, 19385, 19436, 19476, 19483,
19494, 19499, 19532, 19559, 19561, 19565, 19566, 19584, 19588, 19627, 19672,
19740, 19750, 19786, 19797, 19836, 19884, 19909, 19968, 19976, 20029, 20048,
20076, 20102, 20204, 20259, 20389, 20485, 20492, 20496, 20614, 20632, 20635,
20636, 20683, 20703, 20705, 20760, 20764, 20772, 20781, 20823, 20860, 20867,
20868, 20870, 20897, 20940, 21202, 21340, 21392, 21500, 21503, 21508, 21536,
21561, 21565, 21577]

Enter input: turkey

[835, 1611, 1975, 2246, 2517, 2970, 3518, 3745, 3792, 3837, 4328, 5130, 5244,
5274, 5574, 5655, 6870, 6935, 7105, 7951, 9374, 9831, 9908, 10175, 10367, 10395,
10452, 10511, 10522, 10539, 10621, 10627, 10630, 10641, 10797, 10862, 10910,
11795, 11823, 11885, 11894, 12522, 12877, 12909, 12943, 12948, 13129, 13179,
15200, 15338, 15884, 16191, 16239, 16491, 16504, 16948, 18034, 18061, 18293,
18676, 18994, 19285, 19499, 19559, 19740, 19884, 20760, 21392]
```