Report Ram Dabas 2021275

In Q1, the beautifulsoup Python library was used to extract data. By providing the HTML content as the first argument and the parser to be used as the second, we parse the content using the BeautifulSoup constructor. After obtaining a BeautifulSoup object, we can extract data from the file's contents using a variety of its methods. Further the preprocessing was done using basic functions of the python.

```python
def preprocess_file(file_path, i):
    with open(file_path, 'r') as file:
        # Read the file
        text = file.read()
        t1 = text

        # Lowercase the text
        text = text.lower()
        t2 = text

        # Remove HTML tags using BeautifulSoup
        soup = BeautifulSoup(text, "html.parser")
        text = soup.get_text()
        t3 = text

        # Perform tokenization
        tokens = word_tokenize(text)
        t4 = tokens

        # Remove stopwords and punctuations
        tokens = [word for word in tokens if word not in stop_words and word not in string.punctuation]
        t5 = tokens

        # Remove blank space tokens
        tokens = [word for word in tokens if word.strip()]
        t6 = tokens
```

In Q2 a Python class named InvertedIndex with an index_document method for indexing documents. The __init__ method initializes an empty dictionary self.index to store the inverted index. Each key in this dictionary is a unique word (unigram), and the corresponding value is a set of document names where the word appears. The index_document method takes a document name and the text of the document as input. It tokenizes the text into words, and for each word, it adds the document name to the corresponding set in the index. If the word is not already in the index, it creates a new set for it. This way, the index ends up mapping each word to the set of documents that contain it.

Code:

```
class InvertedIndex:
    def __init__(self):
        self.index = {}

    def index_document(self, document_name, text):
        # Preprocess the text
        words = preprocess_file(loc2+filename, 0)

        # Update the index
        for word in words:
            if word not in self.index:
                self.index[word] = {document_name}
            else:
                self.index[word].add(document_name)

    def save(self, filename):
        with open(filename, 'wb') as f:
            pickle.dump(self.index, f)

    def load(self, filename):
        with open(filename, 'rb') as f:
            self.index = pickle.load(f)

    def query(self, query):
        # Split the query into words and operators
        elements = re.split(r' (AND|OR|NOT) ', query)

        # Start with the set of all documents
        result = set(self.index.keys())
```

output:

```
1
Car a car
OR
Query 1: car OR car
Number of documents retrieved for query 1: 6
Names of the documents retrieved for query 1: pp_file746.txt, pp_file264.txt, pp_file174.txt, pp_file886.txt, pp_file542
```

Using the relevant regex functions, I extracted all of the tokenized words from the Q1 dataset and used them in Q3. Using a class called Positional_Indexing, I first created a positional index table. After that, I used the query to extract the data from the index table by dumping it into the pickle file. An additional function that I created pulls a query from a positional index table. At last, obtaining our output in the specific format specified in the assignment and receiving the user's input.

Code:

```python
#Question3
class PositionalIndex:
    def __init__(self):
        self.index = {}

    def index_document(self, document_name, text):
        # Preprocess the text
        words = do_pp(text)

        # Update the index
        for position, word in enumerate(words):
            if word not in self.index:
                self.index[word] = {document_name: [position]}
            else:
                if document_name in self.index[word]:
                    self.index[word][document_name].append(position)
                else:
                    self.index[word][document_name] = [position]

    def save(self, filename):
        with open(filename, 'wb') as f:
            pickle.dump(self.index, f)

    def load(self, filename):
        with open(filename, 'rb') as f:
            self.index = pickle.load(f)

    def query(self, query):
        # Split the query into words
        words = do_pp(query)
        print(words)
```

```python
# Start with the set of all positions of the first word
result = self.index.get(words[0], {})

# Apply each word to the result
for i in range(1, len(words)):
    word_positions = self.index.get(words[i], {})
    for doc in list(result.keys()):
        positions = [pos - i for pos in word_positions.get(doc, [])]
        result[doc] = [pos for pos in result[doc] if pos in positions]

# Filter out the documents where the query does not exist
result = {doc: positions for doc, positions in result.items() if positions}

return result
```

output:

```
1
everything acoustic bass ukuleles know smaller model available ukes violins
['everything', 'acoustic', 'bass', 'ukuleles', 'know', 'smaller', 'model', 'available', 'ukes', 'violins']
Number of documents retrieved for query 1 using positional index: 1
Names of documents retrieved for query 1 using positional index: pp_file3.txt
```