QA_System

September 6, 2023

[2]:
```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk import ne_chunk, pos_tag, word_tokenize
import re
import wikipedia
from nltk.probability import FreqDist
from nltk.util import ngrams
import string


# to remove the stop words
stop_words = set(stopwords.words('english'))

# Cleaning data here
# Tokenizing, removing punctuation and stop words
# also removing the rewrite string from the search result
def clean_data(rewrite_string, data):
    data = data.lower()
    s = data.replace(rewrite_string, "")
    for i in rewrite_string.split(" "):
        if (i not in stop_words):
            s = s.replace(i, "")
    tokenized_text = word_tokenize(s)
    without_punct = [''.join(
        eachcharac for eachcharac in eachword if eachcharac not in string.
 ↪punctuation) for
        eachword in tokenized_text]
    without_stop_words = ""
    for i in without_punct:
        if i not in stop_words:
            without_stop_words = without_stop_words + " " + i
    return without_stop_words

# Searching for results on wikipedia
def get_content_from_wiki(list_of_rewrittenqs_with_wts):
    content_weight_rewrite = []
    for j in list_of_rewrittenqs_with_wts:
```

```
        res = "\"" + j["rewrite_string"] + "\""
        wiki_res = wikipedia.search(res)
        # print(wiki_res)
        super_string = ""
        for i in wiki_res:
            # some results had this error so we had to remove disambiguation
            if (not re.match(".*disambiguation.*", i)):
                tokenized_sentence = []
                result = ""
                try:
                    result = wikipedia.summary(i)
                except Exception:
                    pass
                result_string = result
                super_string = super_string + " " + clean_data(j["rewrite_string"], result_string)
                # Forming a list of dictionaries containing
                # the rewrite string,
                # content from the wikipedia page
                # weight given to each result
        content_weight_rewrite.append(
            {
                "statement": j["rewrite_string"],
                "content": super_string,
                "weight": j["priority"]
            }
        )
        super_string = ""

    return content_weight_rewrite

# to rewrite the input queries
def rewrite_ques_assign_weights(question):
    sample_txt = question
    list_of_dicts_with_weights = []

    # WHO condition
    if re.match(r'^[wW]ho', sample_txt):
        match_res = re.match('^[wW]ho (is|was|[a-z]*) (.*)', sample_txt)
        if (match_res.groups()[0] == 'is' or match_res.groups()[0] == 'was'):
            list_of_dicts_with_weights.append(
                {
                    "rewrite_string": str(match_res.groups()[1]) + ' ' + str(match_res.groups()[0]),
                    "priority": 5
                })
            list_of_dicts_with_weights.append(
```

```
                {
                    "rewrite_string": str(match_res.groups()[1]) + ' ' +␣
↪str(match_res.groups()[0] + " known for"),
                    "priority": 3})
            list_of_dicts_with_weights.append(
                {
                    "rewrite_string": str(match_res.groups()[1]) + ' ' +␣
↪str(match_res.groups()[0] + " famous for"),
                    "priority": 2})
        else:
            res = str(match_res.groups()[1]) + ' was ' + str(match_res.
↪groups()[0] + ' by ')

    # WHEN condition
    if re.match(r'^[wW]hen', sample_txt):
        match_res = re.match('^[wW]hen (is|was) (.*)', sample_txt)
        if (match_res.groups()[0] == 'is' or match_res.groups()[0] == 'was'):
            list_of_dicts_with_weights.append(
                {
                    "rewrite_string": str(match_res.groups()[1]) + ' ' +␣
↪str(match_res.groups()[0] + ' on'),
                    "priority": 5
                })

    # WHAT condition
    if re.match(r'^[wW]hat', sample_txt):
        match_res = re.match('^[wW]hat (is|was) (.*)', sample_txt)
        if (match_res.groups()[0] == 'is' or match_res.groups()[0] == 'was'):
            list_of_dicts_with_weights.append(
                {
                    "rewrite_string": str(match_res.groups()[1]) + ' ' +␣
↪str(match_res.groups()[0]),
                    "priority": 5
                })

    # WHERE condition
    if re.match(r'^[wW]here', sample_txt):
        match_res = re.match('^[wW]here (is|was) (.*)', sample_txt)
        if (match_res.groups()[0] == 'is' or match_res.groups()[0] == 'was'):
            list_of_dicts_with_weights.append(
                {
                    "rewrite_string": str(match_res.groups()[1]) + ' ' +␣
↪str(match_res.groups()[0]) + ' in',
                    "priority": 5
                })
    return list_of_dicts_with_weights
```

```
# Generating n-grams
def gen_ngrams(text):
    temp = []
    u_grams = ngrams(word_tokenize(text.lower()), 1)
    b_grams = ngrams(word_tokenize(text.lower()), 2)
    t_grams = ngrams(word_tokenize(text.lower()), 3)
    f_grams = ngrams(word_tokenize(text.lower()), 4)
    fi_grams = ngrams(word_tokenize(text.lower()), 5)
    #    for i in u_grams:
    #        temp.append(i);
    for i in b_grams:
        temp.append(i);
    for i in t_grams:
        temp.append(i);
    return FreqDist(temp).most_common(1)


# This is where the program begins
exit_flag = False
while (not exit_flag):
    user_input = input("Please Enter Your Question Now: ")
    if (("who" in user_input) or ("when" in user_input) or ("where" in␣
 ↪user_input) or ("what" in user_input)):
        can_do = 1
    elif ("exit" in user_input):
        can_do = 2
    else:
        can_do = 3

    if (can_do == 1):
        value_dicts = rewrite_ques_assign_weights(user_input)
        similar_content_dict = get_content_from_wiki(value_dicts)
        new_diction = {}
        answer_dict = []
        for i in similar_content_dict:
            ans = gen_ngrams(i["content"])
            if (len(ans) != 0):
                output = ""
                for j in ans[0][0]:
                    output = output + j + " "
                answer_dict.append({
                    "answer_statement": i["statement"] + " " + output,
                    "weighted_avg": ans[0][1] * i["weight"]
                })
        #print(answer_dict)
        for i in answer_dict:
```

```
            print('Answer: ', i['answer_statement'])

    elif (can_do == 2):
        print("Thank you, Bye")
        exit_flag = True

    else:
        print('Sorry, we cannot answer your question at the moment')
```

```
Please Enter Your Question Now: who is michael jackson
Answer:  michael jackson is american singer
Please Enter Your Question Now: where is fairfax
Answer:  fairfax is in united states
Please Enter Your Question Now: when is christmas
Answer:  christmas is on 24th december
Please Enter Your Question Now: what is hyderabad
Answer:  hyderabad is old city
Please Enter Your Question Now: how do you know me
Sorry, we cannot answer your question at the moment
Please Enter Your Question Now: exit
Thank you, Bye
```

[ ]: