Assignment 4: Natural Language Processing

Q1: Define a tokenize function

which does the following in sequence:

- takes a string as an input
- converts the string into lowercase
- segments the lowercased string into tokens. A token is defined as follows:
    - Each token has **at least two characters**.
    - The **first/last character can only be a letter (i.e. a-z) or a number (0-9)**
    - **In the middle, there are 0 or more characters, which can only be letters (a-z), numbers (0-9), hyphens ("-"), underscores ("_"), dot ("."), or "@" symbols.**
- **lemmatizes** all tokens using WordNetLemmatizer
- **removes stop words** from the tokens (use English stop words list from NLTK)
- generate **token frequency dictionary**, where each unique token is a key and the frequency of the token is the value. (Hint: you can use nltk.FreqDist to create it)
- returns the token frequency dictionary as the output

Note, this question is similar to Q1 in your Assignment 1, but more complicated

Q2: Find duplicate questions by similarity

A data file 'qa.csv' has been provided for this question. This dataset has two columns: question and answer as shown in screenshot blow. **Here we only use "question" column**.

- Define a function **find_similar_doc** as follows:
  - takes two inputs: a list of documents as strings (i.e. *docs*), and the index of a selected document as an integer (i.e. *doc_id*).
  - uses the **"tokenize"** function defined in Q1 to tokenize each document
  - generates **tf_idf matrix** from the tokens (hint: reference to the tf_idf function defined in Section 7.5 in lecture notes)
  - calculates the **pairwise cosine distance** of documents using the tf_idf matrix
  - for the selected document, finds the **index of the most similar document** (but not the selected document itself!) by the cosine similarity score
  - returns the index of the most similar document and the similarity score
- Test your function with two selected questions 15 and 51 respectively, i.e., doc_id = 15 and doc_id = 51.
  - Check the most similar questions discovered for each of them
  - Do you think this function can successfully find duplicate questions? Why does it work or not work? Write down your analysis in a document and upload it to canvas along with your code.

In [1]:
```
import pandas as pd
data=pd.read_csv("qa.csv", header=0)
data.head()
```

Out[1]:

	question	answer
0	Why does Zebras have stripes?	this provides camouflage - predator vision is ...
1	Do animals have a sense of humour?	Dogs don't think that way. You're projecting h...
2	Is the universe flat?	Yes, the Universe is flat. It's a VERY difficu...
3	What is the U.S. Green card procedure after la...	This depends mainly on which state you are . U...
4	motor vehicle agencys in central newjersey?	rt 35 by the monmouth mall, across the street ...

Q3 (Bonus): Retrieve relevant answers to questions by similarity

Each row in "qa.csv" defines a question and its corresponding answer. Now assume we do not know answers to these questions. Let's design an algorithm to retrieve the most relevant answer to each question.

1. Define another function **match_question_answer** as follows:
   - takes two inputs: a list of questions as strings (i.e. *questions*), and a list of answers as strings (i.e. *answers*).
   - uses the "tokenize" function defined in Q1 to tokenize each document
   - generates tf_idf matrix from the tokens (hint: reference to the tf_idf function defined in Section 7.5 in lecture notes)
   - calculates the **cosine distance between every question and every answer** using the tf_idf matrix (hint, you can use scipy.spatial.distance.cdist function)
   - for each question q, **identifies the answer which is the most similar to** q as the most relevant answer (denoted as a^*)
   - returns a list of tuples each with 3 elements, (**index of** q, **index of** a^* , **similarity score**) for every question q in the dataset.
2. Define a function **evaluate** to evaluate the performance of retrieval as follows:
   - takes the returned list from match_question_answer function as an input
   - sets **a minimum similarity threshold** (denoted as min_sim), and selects entries from the list with similarity >= the threshold (denoted as $matching\_pairs$).
   - calculates two metrics for selected $matching\_pairs$:
     - **recall**: the percentage of questions with matching answers, i.e. $len(matching\_pairs)/len(questions)$
     - **precision**: the precentage of questions in matching_pairs indeed matched with the corresponding answers as indicated in the dataset.
   - **varies the similarity threshold** from 0 to 0.6 with 0.05 increase in each round, calculate the recall and precision in each round, and plot **a chart with two lines** where the recall and precision as Y axis and the threshold as X axis.
3. **As the threshold increases, how precision and recal change**? What can be a **good similarity threshold** for retrieving most relevant answers to these questions? Write down your analysis in a document and upload it to canvas along with your code.

In [ ]:
```
# import block

import pandas as pd
```

In [ ]:
```python
# Q1
def tokenize(text):

    token_count = None

    # add your code here

    return token_count
```

In [ ]:
```python
# Q2
def find_similar_doc(doc_id, docs):

    best_matching_doc_id = None
    similarity = None

    # add your code here

    return best_matching_doc_id, similarity
```

In [ ]:
```python
# Q3.1
def match_question_answer(questions, answers):

    result = []

    # add your code here

    return result
```

In [ ]:
```python
# Q3.2

def evaluate(result):

    # add your code here
```

```
In [ ]: if __name__ == "__main__":

            # Test Q1
            text='''contact Yahoo! at "http://login.yahoo.com", select forgot
                    your password. If that fails to reset, contact Yahoo! at
                    their password department 408-349-1572 -- Can't promise
                    their phone department will fix, but they'll know where to
                    go next. Corporate emails from Yahoo! don't come from
                    their free mail system address space. Webmaster@yahoo.com
                    is not a corporate email address.'''

            print("Test Q1")
            for key, value in tokenize(text).items():
                print(key, value)

            # You should get the result look like :
            # contact 2    yahoo 3       http 1           login.yahoo.com 1
            # select 1     forget 1      password 2       fail 1
            # reset 1      department 2  408-349-1572 1   promise 1
            # phone 1      fix 1         know 1           go 1
            # next 1       corporate 2   email 2          come 1
            # free 1        mail 1        system 1         address 2
            # space 1       webmaster@yahoo.com 1

            data=pd.read_csv("qa.csv", header=0)

            # Test Q2
            print("\nTest Q2")
            doc_id=15
            x,y=find_similar_text(doc_id, data["question"].values.tolist())
            print(x,y)
            print(data["question"].iloc[doc_id])
            print(data["question"].iloc[x])

            doc_id=51
            x,y=find_similar_text(doc_id, data["question"].values.tolist())
            print(x,y)
            print(data["question"].iloc[doc_id])
            print(data["question"].iloc[x])


            # Test Q3
            print("\nTest Q3.1")
            result = match_question_answer(data["question"].values.tolist(), \
                                    data["answer"].values.tolist())
            print("\nTest Q3.2")
            evaluate(result)

In [ ]:
```