Assignment 4: Natural Language Processing

Q1: Extract data using regular expression (2 points)

Suppose you have scraped the text shown belone from an online source. Define a **extract** function which:

- takes a piece of text (in the format of shown below) as an input
- extracts data into a list of tuples using regular expression, e.g. [('Grant Cornwell', 'College of Wooster', '2015', '911,651'), ...]
- returns the list of tuples

In [ ]:
```
text='''Following is total compensation for other presidents at privat
e colleges in Ohio in 2015:

Grant Cornwell, College of Wooster (left in 2015): $911,651
Marvin Krislov, Oberlin College (left in 2016):  $829,913
Mark Roosevelt, Antioch College, (left in 2015): $507,672
Laurie Joyner, Wittenberg University (left in 2015): $463,504
Richard Giese, University of Mount Union (left in 2015): $453,800'''
```

Q2: Find duplicate questions by similarity (8 points)

A data file 'quora_duplicate_question_500.csv' has been provided as shown below. Each sample in this dataset has two questions, denoted as (q_1, q_2) (i.e."q1" and "q2" columns). Column "is_duplicate"=1 indicates if the two questions are indeed duplicate; otherwise, they are not duplicate, although they look similar. This dataset has 500 question pairs in total.

In [ ]:
```
import pandas as pd
data=pd.read_csv("../../dataset/quora_duplicate_question_500.csv",head
er=0)
data.head(3)
```

Q2.1. Define a function **"tokenize"** as follows:

- takes three parameters:
  - *text*: input string.
  - *lemmatized*: an optional boolean parameter to indicate if tokens are lemmatized. The default value is False.
  - *no_stopword*: an optional bookean parameter to remove stop words. The default value is False.
- splits the input text into unigrams and also clean up tokens as follows:
  - if *lemmatized* is turned on, lemmatize all unigrams.
  - if *no_stopword* is turned on, remove all stop words.
- returns the list of unigrams after all the processing. (Hint: you can use spacy package for this task. For reference, check https://spacy.io/api/token#attributes (https://spacy.io/api/token#attributes))

Q2.2. Define a function **get_similarity** as follows:

- takes the following inputs: two lists of strings (i.e. list of q1 and list of q2), and boolean parameters *lemmatized* and *no_stopword* as defined in (Q2.1).
- tokenize each question from the both lists using the "**tokenize**" function defined in (Q2.1).
- generates **tf_idf matrix** from the tokens obtained from the questions in both of the lists (hint: reference to the tf_idf function defined in Section 8.5 in lecture notes. You need to concatenate q1 and q2)
- calculates the **cosine similarity** of the question pair (q_1, q_2) in each sample using the tf_idf matrix
- returns similarity scores for the 500 question pairs

Q2.3. Define a function **predict** as follows:

- takes three lists, i.e. list of similarity scores, "is_duplicate" column, and a *threshold* with default value of 0.5 as inputs
- if a similarity > *threshold*, then predicts the question pair is duplicate
- calulates the percentage of duplicate questions pairs that are successfully identified, i.e.

$$\frac{count\ (prediction = 1\ \&\ is\_duplicate = 1)}{count\ (is\_duplicate = 1)}$$

- returns the predicted values and the percentage

Q2.4. Test:

- Test your solution using different options in in the tokenize function, i.e. with or without lemmatization, with or without removing stop words, to see how these options may affect the accuracy.
- Analyze why some option works the best (or worst). Write your analysis in a pdf file.

Q3 (Bonus): More analysis

Q3.1. Define a function "**evaluate**" as follows:

- takes three lists, i.e. list of similarity scores, "is_duplicate" column, and a *threshold* with default value of 0.5 as inputs
- if a similarity > *threshold*, then predicts the question pair is duplicate, i.e. prediction = 1
- calulates two metrics:
  - *recall*: the percentage of duplicate questions pairs that are correctly identified, i.e.
$$\frac{count\ (prediction = 1\ \&\ is\_duplicate = 1)}{count\ (is\_duplicate = 1)}$$
  - *precision*: the percentage of question pairs identified as duplicate are indeed duplicate, i.e.
$$\frac{count\ (prediction = 1\ \&\ is\_duplicate = 1)}{count\ (prediction = 1)}$$
- returns the precision and recall

Q3.2. Analyze the following questions

- If you change the similarity threhold from 0.1 to 0.9, how do precision and recall change?
- Consider both precision and recall, do you think what options (i.e. lemmatization, removing stop words, similarity threshold) can you give the best performance?
- What kind of duplicates can be easily found? What kind of ones can be difficult to find?
- Do you think the TF-IDF approach is successful in finding duplicate questions?

These are open questions. Just show your analysis with necessary support from the dataset, and save your analysis in a pdf file.

In [ ]:
```
import re
import nltk
import pandas as pd
```

In [ ]:
```
def extract(text):

    result = None

     # add your code here

    return result
```

In [ ]:
```
def tokenize(doc, lemmatized=False, no_stopword=False):

    tokens =[]

    # add your code here

    return tokens
```

In [ ]:
```
def get_similarity(q1, q2, lemmatized=False, no_stopword=False):

    sim = None

    # add your code here

    return sim
```

In [ ]:
```
def predict(sim, ground_truth, threshold=0.5):

    predict = None
    recall = None

    # add your code here

    return predict, recall
```

In [ ]:
```
def evaluate(sim, ground_truth):

    precision = None
    recall = None

    return precision, recall
```

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

    # Test Q1

    text='''Following is total compensation for other presidents at pr
ivate colleges in Ohio in 2015:

Grant Cornwell, College of Wooster (left in 2015): $911,651
Marvin Krislov, Oberlin College (left in 2016):  $829,913
Mark Roosevelt, Antioch College, (left in 2015): $507,672
Laurie Joyner, Wittenberg University (left in 2015): $463,504
Richard Giese, University of Mount Union (left in 2015): $453,800'''

    print("Test Q1")
    print(extract(text))

    data=pd.read_csv("../../dataset/quora_duplicate_question_500.csv",
header=0)
    q1 = data["q1"].values.tolist()
    q2 = data["q2"].values.tolist()

    # Test Q2
    print("Test Q1")
    print("\nlemmatized: No, no_stopword: No")
    sim = get_similarity(q1,q2)
    pred, recall=predict(sim, data["is_duplicate"].values)
    print(recall)

    print("\nlemmatized: Yes, no_stopword: No")
    sim = get_similarity(q1,q2, True)
    pred, recall=predict(sim, data["is_duplicate"].values)
    print(recall)

    print("\nlemmatized: No, no_stopword: Yes")
    sim = get_similarity(q1,q2, False, True)
    pred, recall=predict(sim, data["is_duplicate"].values)
    print(recall)

    print("\nlemmatized: Yes, no_stopword: Yes")
    sim = get_similarity(q1,q2, True, True)
    pred, recall=predict(sim, data["is_duplicate"].values)
    print(recall)

    # Test Q3. Get similarity score, set threshold, and then
    prec, rec = evaluate(sim, data["is_duplicate"].values, 0.5)
```

Your output of Q2 may look like:

lemmatized: No, no_stopword: No 0.6304347826086957

lemmatized: Yes, no_stopword: No 0.782608695652174

lemmatized: No, no_stopword: Yes 0.6358695652173914

lemmatized: Yes, no_stopword: Yes 0.7717391304347826

Submission Guideline

- Following the solution template provided below. Use **main** block to test your functions
- Save your code into a python file (e.g. assign.py) that can be run in a python 3 environment. **Make sure your .py file can be executed**.
- Make sure you have all import statements. To test your code, open a command window in your current python working folder, type "python assign.py" to see if it can run successfully.
- **Each homework assignment should be completed independently. Never ever copy others' work**.

In [ ]: