Assignment 2

Start Assignment

Due Friday by 11:05pm   **Points** 140   **Submitting** a file upload
File Types zip   **Attempts** 0   **Allowed Attempts** 3
Available Oct 13 at 11am - Oct 30 at 11:05pm 18 days

Homework Assignment 2
Pokemon, Covid, Text Processing

Worth 140 points.

Posted Wednesday, Oct 13

Due Friday, Oct 29 at 11 PM in Canvas

Late Submission: By Saturday, October 30, 11 PM in Canvas - 15 point penalty

You have the option to work individually or with a partner. But no groups over 2 people!

We are going to roll over the partner information from Assignment 1 for this assignment, so if you want to continue working with your partner from Assignment 1, you don't need to inform us.

However, if you want to change your partner from Assignment 1, or if you were working alone for Assignment 1 but now want to work with a partner, you will need to let your TA know.

Make sure you abide by the **DCS Academic Integrity Policy for Programming Assignments.** **(https://www.cs.rutgers.edu/academics/undergraduate/academic-integrity-policy/programming-assignments)**

For the problems in this homework, write your code in files **pokemon.py**, **covid.py**, and **tfidf.py** respectively.

▶ h of these files should be executable, which means that when we run your program, it should do an end-to-end run of the tasks in each problem, and produce results as required. For instance:

```
> python pokemon.py
```

should execute the tasks in Problem 1 and produce the required output files.

Zip all of these Python files into a single file named **hw2.zip** and submit this to Canvas. Do not include any input or output files, we only need your Python code.

You are allowed up to 3 submissions (total over regular and late submissions), only the last submission will be retained for grading. Note: If you have a late submission, it will override any submissions you may have made before the regular deadline, and you will be assessed the 15 point penalty.

Problem 1: Pokemon Box Dataset (45 points)

Given a CSV data file as represented by the sample file **pokemonTrain.csv** ⤓ **(https://rutgers.instructure.com/courses/133454/files/19785941/download? download_frd=1)** perform the following operations on it.

1. [7 pts] Find out what percentage of "fire" type pokemons are at or above the "level" 40.

   Your program should print the value as follows (replace ... with value):

   ```
   Percentage of fire type Pokemons at or above level 40 = ...
   ```

   The value should be rounded off (not ceiling) using the **round()** function. So, for instance, if the value is 12.3 (less than or equal to 12.5) you would print 12, but if it was 12.615 (more than 12.5), you would print 13.

   Print the value to a file named "pokemon1.txt"

2. [10 pts] Fill in the missing "type" column values (given by NaN) by mapping them from the corresponding "weakness" values. You will see that usually, a given pokemon weakness has a fixed "type", but there do exist some exceptions. Hence, fill in the "type" column with the most common "type" corresponding to the pokemon's "weakness" value.

For example, most of the pokemons having the weakness "electric" are "water" type pokemons but there are other types too that have "electric" as their weakness (exceptions in that "type"). But since "water" is the most common type for weakness "electric", it should be filled in.

In case of a tie, use the type that appears first in alphabetical order.

3. [13 pts] Fill in the missing values in the Attack ("atk"), Defense ("def") and Hit Points ("hp") columns as follows:

a. Set the pokemon level threshold to 40.

b. For a Pokemon having level above the threshold (i.e. > 40), fill in the missing value for atk/def/hp with the average values of atk/def/hp of Pokemons with level > 40. So, for instance, you would substitute the missing "atk" value for Magmar (level 44), with the average "atk" value for Pokemons with level > 40. Round the average to one decimal place.

c. For a Pokemon having level equal to or below the threshold (i.e. <= 40), fill in the missing value for atk/def/hp with the average values of atk/def/hp of Pokemons with level <= 40. Round the average to one decimal place.

After performing #2 and #3, write the modified data to another csv file named "pokemonResult.csv"

The following tasks should be performed on the pokemonResult.csv file that resulted above.

4. [10 pts] Create a dictionary that maps pokemon types to their personalities. This dictionary would map a string to a list of strings. For example:

```
    {"fire": ["docile", "modest", ...], "normal": ["mild", "relaxed", ...],
...}
```

Note: You can create an empty default dictionary of list with
`defaultdict(list)`

▶ Your dictionary should have the keys ordered alphabetically, and also items ordered alphabetically in the values list, as shown in the example.

Print the dictionary in the following format:

```
Pokemon type to personality mapping:

    fire: docile, modest, ...
    normal: mild, relaxed, ...

 ...
```

Print the dictionary to a file named "pokemon4.txt"

5. [5 pts] Find out the average Hit Points ("hp") for pokemons of stage 3.0.

Your program should print the value as follows (replace ... with value):

```
    Average hit point for Pokemons of stage 3.0 = ...
```

You should round off the value, like in #1 above.

Print the value to a file named "pokemon5.txt"

Problem 2: Covid-19 Dataset (35 points)

Given a Covid-19 data CSV file with 12 feature columns, perform the tasks given below. Use the sample file **covidTrain.csv** ↓ **(https://rutgers.instructure.com/courses/133454/files/19786046/download? download_frd=1)** to test your code.

1. [5 pts] In the age column, wherever there is a range of values, replace it by the rounded off average value. E.g., for 10-14 substitute 12. (Rounding should be done like in 1.1). You might want to use regular expressions here, but it is not required.

2. [6 pts] Change the date format for the date columns - date_onset_symptoms, date_admission_hospital and date_confirmation from dd.mm.yyyy to [▶] m.dd.yyyy. Again, you can use regexps here, but it is not required.

3. [7 pts] Fill in the missing (NaN) "latitude" and "longitude" values by the average of the latitude and longitude values for the province where the case was recorded. Round the average to 2 decimal places.

4. [7 pts] Fill in the missing "city" values by the most occurring city value in that province. In case of a tie, use the city that appears first in alphabetical order.

5. [10 pts] Fill in the missing "symptom" values by the single most frequent symptom in the province where the case was recorded. In case of a tie, use the symptom that appears first in alphabetical order.

Note: While iterating through records, if you come across multiple symptoms for a single record, you need to consider them individually for frequency counts.

Very Important!: Some symptoms could be separated by a ';' , i.e., semicolon plus space and some by ';' , i.e., just a semicolon, even within the same record. For example:

```
"fever; sore throat;cough;weak; expectoration;muscular soreness"
```

After performing all these tasks, write the whole data back to another CSV file named "covidResult.csv"

Problem 3: Text Processing (60 pts)

For this problem, you are given a set of documents (text files) on which you will perform some preprocessing tasks, and then compute what is called the TF-IDF score for each word. The TF-IDF score for a word is measure of its importance within the entire set of documents: the higher the score, the more important is the word.

The input set of documents must be read from a file named "tfidf_docs.txt". This file will list all the documents (one per line) you will need to work with. For instance, if ▶ need to work with the set "doc1.txt", "doc2.txt", and "doc2.txt", the input file "tfidf_docs.txt" contents will look like this:

```
doc1.txt
doc2.txt
doc2.txt
```

- **Part 1: Preprocessing (30 pts)**

  For each document in the input set, clean and preprocess it as follows:

  1. [15 pts] Clean.
     - Remove all characters that are not words or whitespaces. Words are sequences of letters (upper and lower case), digits, and underscores.
     - Remove extra whitespaces between words. e.g., "Hello World! Let's   learn   Python!", so that there is exactly one whitespace between any pair of words.
     - Remove all website links. A website link is a sequence of non-whitespace characters that starts with either "http://" or "https://".
     - Convert all the words to lowercase.

     The resulting document should only contain lowercase words separated by a single whitespace.

  2. [7 pts] Remove stopwords.

     From the document that results after #1, remove "stopwords". These are the non-essential (or "noise") words listed in the file **stopwords.txt** ↓ **(https://rutgers.instructure.com/courses/133454/files/19787014/download? download_frd=1)**

  3. [8 pts] Stemming and Lemmatization.

     This is a process of reducing words to their root forms. For example, look at the following reductions: run, running, runs → run. All three words capture the same idea 'run' and hence their suffixes are not as important.

▶ (If you would like to get a better idea, you may want read this **article (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html)** . This is completely optional, you can do the assignment without reading the article.)

Use the following rules to reduce the words to their root form:

a. Words ending with "ing": "flying" becomes "fly"
b. Words ending with "ly": "successfully" becomes "successful"
c. Words ending with "ment": "punishment" becomes "punish"

These rules are not expected to capture all the edge cases of Stemming in the English language but are intended to give you a general idea of the preprocessing steps in NLP (Natural Language Processing) tasks.

After performing #1, #2, and #3 for each input document, write the modified data to another text file with the prefix "preproc_". For instance, if the input document is "doc1.txt", the output should be "preproc_doc1.txt".

- **Part 2: Computing TF-IDF Scores (30 pts)**

Once preprocessing is performed on all the documents, you need to compute the
Term Frequency(TF) — Inverse Document Frequency(IDF) score for each word

What is TF-IDF?

In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

Resources:

- **TFIDF Python Example** **(https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76)**
- **tf-idf Wikipedia Page** **(https://en.wikipedia.org/wiki/Tf%E2%80%93idf)**

▶   ○ **[TF-IDF/Term Frequency Technique](https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3)**   **(https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3)**

Steps:

a. For each preprocessed document that results from the preprocessing in Part 1, compute frequencies of all the distinct words in that document only. So if you had 3 documemnts in the input set, you will 3 sets of word frequencies, one per document.

b. Compute the Term Frequency (TF) of each distinct word (also called term) for each of the preprocessed documents:

```
    TF(t) = (Number of times term t appears in a document) / (Total number
of terms in the document)
```

Note: The denominator, total number of terms, is the sum total of all the words, not just unique instances. Another way to think of this is it is the sum of the number of times each word appears in the document.

c. Compute the Inverse Document Frequency (IDF) of each distinct word for each of the preprocessed documents.

IDF is a measure of how common or rare a word is in a document set (a set of preprocessd text files in this case). It is calculated by taking the logarithm of the following term:

```
    IDF[t] = (Total number of documents)/(Number of documents the word is f
ound in).
```

To avoid division by zero, set the IDF value as 0 if the 'Number of documents the word is found in' is 0.

Also, add 1 to the IDF score so that the TF-IDF score is non-zero. You can read more about IDF here.

d. Calculate TF-IDF score: TF * IDF for each distinct word in each preprocessed document. Use 2 decimal places for the score. Round the average to 2 decimal places.

▶

e. Print the top 5 most important words in each preprocessed document according to their TF-IDF scores. The higher the TF-IDF score, the more important the word. In case of ties in a document, pick words in alphabetical order. You should print the result as a list of (word,TF-IDF score) tuples sorted in descending TF-IDF scores.

Print to a file prefixed with "tfidf_". So if the initial input document was "doc1.txt", you should print the TF-IDF results to "tfidf_doc1.txt".

Testing:

1. You can begin with the following three sentences as separate documents against which to test your code:
   - #d1 = "It is going to rain today."
   - #d2 = "Today I am not going outside."
   - #d3 = "I am going to watch the season premiere."

   You can match values computed by your code with this same example in the **TF-IDF/Term Frequency Technique** **(https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3)** page referenced above. Look for it under "Let's cover an example of 3 documents" on this page.

2. Next, you can test your code against **test1.txt** ⤓ **(https://rutgers.instructure.com/courses/133454/files/19786243/download?download_frd=1)** and **test2.txt** ⤓ **(https://rutgers.instructure.com/courses/133454/files/19786254/download?download_frd=1)** . Compare your resulting preprocessed documents with our results in **preproc_test1.txt** ⤓ **(https://rutgers.instructure.com/courses/133454/files/19786290/download?download_frd=1)** and **preproc_test2.txt** ⤓

(https://rutgers.instructure.com/courses/133454/files/19786297/download?
download_frd=1) , and your TF-IDF results with our results in **tfidf_test1.txt** ⬇
▶ **ttps://rutgers.instructure.com/courses/133454/files/19786304/download?**
download_frd=1) and **tfidf_test2.txt** ⬇
(https://rutgers.instructure.com/courses/133454/files/19786328/download?
download_frd=1)

3. Finally, you can try your code on these files: **covid_doc1.txt** ⬇
(https://rutgers.instructure.com/courses/133454/files/19786393/download?
download_frd=1) , **covid_doc2.txt** ⬇
(https://rutgers.instructure.com/courses/133454/files/19786400/download?
download_frd=1) and **covid_doc3.txt** ⬇
(https://rutgers.instructure.com/courses/133454/files/19786405/download?
download_frd=1) . Results for these are not provided, however the files are small
enough that you can identify the words that make the cut and manually compute
TF-IDF.