Homework Assignment 2

Pokemon, Covid, Text Processing

Worth 140 points.

Posted Friday, Mar 1

Due Friday, Mar 22 at 11 PM in Canvas

Late Submission: By Saturday, Mar 23, 11 PM in Canvas - 15 point penalty
(15 points will be deducted from your score)

You will work on this assignment individually.

Make sure you abide by the DCS Acadmic Integrity Policy for Programming Assignments.

Write your code in files **pokemon.py** (for Problem 1), **covid.py** (for Problem 2), and **tfidf.py** (for Problem 3).

Each 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 graded.

For any of the problems in this assignment, you may only import and use modules we have covered in lecture before NumPy and Pandas. So you can use math, collections, re, and csv. (Technically you can use json, but it's not required for this assignment since you are not working with JSON-formatted data. Likewise, the random module is not required for this assignment.)

You may NOT use NumPy or Pandas.

Problem 1: Pokemon Box Dataset (45 points)

Given a CSV data file as represented by the sample file pokemonTrain.csv, perform the following operations on it.

1. [7 pts] Find out what percentage of "fire" type pokemons are at or above the "level" 40.
   (This is percentage over fire pokemons only, not all pokemons)

   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, as in:

   ```
   Percentage of fire type Pokemons at or above level 40 = 13
   ```

   Do NOT add % after the value (such as 13%), only print the number

   Print the value to a file named "pokemon1.txt"
   If you do not print to a file, or your output file name is not exactly as required, you will get 0 points.

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 typically a given pokemon weakness has a fixed "type", but there are some exceptions. So, 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 (NaN) 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".

    This result file should have all of the rows from the input file - rows that were modified as well as rows that were not modified.

    If you do not write the modified data to another CSV file, or your output file name is not exactly as required, you will get 0 points.

    The following tasks (#4 and #5) 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", ...],  ...}
    ```

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

    Print the dictionary in the following format:

    ```
    Pokemon type to personality mapping:

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

    Print the dictionary to a file named "pokemon4.txt"
    If you do not print to a file, or your output file name is not exactly as required, you will get 0 points.

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"
    If you do not print to a file, or your output file name is not exactly as required, you will get 0 points.

Testing

We will be testing your code with other similarly formatted files with different values. These test files will renamed as `pokemonTrain.csv` when running your code, so your code doesn't need to accommodate different file names when testing. (Likewise, you should test your code with similarly formatted test files, renaming them as `pokemonTrain.csv` before testing.)

It is ok to have other intermediate files generated during the creation of the result files, we will just ignore all files other than the required result files.

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 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 mm.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.

Watch out!: 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"
```

Also, the symptoms column has values such as "fever 37.7 C" and "fever (38-39 C)". For these values, you shouldn't do any special processing, so the symptoms should be extracted as "fever 37.7 C" and "fever (38-39 C)", as presented in the data.

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

This result file should have all of the rows from the input file - rows that were modified as well as rows that were not modified.

If you do not write data back to another CSV file, or your output file name is not exactly as required, you will get 0 points.

Testing

We will be testing your code with other similarly formatted files with different values. These test files will renamed as `covidTrain.csv` when running your code, so your code doesn't need to accommodate different file names when testing. (Likewise, you should test your code with similarly formatted test files, renaming them as `covidTrain.csv` before testing.)

It is ok to have other intermediate files generated during the creation of the result file, we will just ignore all files other than the required result files.

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 you 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 above, remove "stopwords". These are the non-essential (or "noise") words listed in the file stopwords.txt

  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. 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 above 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".
If you do not print to a file, or your output file name is not exactly as required, you will get 0 points.

- **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
> - tf-idf Wikipedia Page
> - TF-IDF/Term Frequency Technique

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 documents in the input set, you will compute 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. So if a word occurs 5 times, and the total number of words in a document is 100, then TF for that word is 5/100.

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) = log((Total number of documents) / (Number of documents the word is found in)) + 1
```

Note: The log here uses base e. And 1 is added after the log is taken, so that the IDF score is guaranteed to be non-zero.

d. Calculate TF-IDF score: TF * IDF for each distinct word in each preprocessed document. Round the score 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 score, 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. See the Testing section below, in files **tfidf_test1.txt** and **tfidf_test2.txt**, for the exact output format.

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".
If you do not print to a file, or your output file name is not exactly as required, you will get 0 points.

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 page referenced above. Look for it under "Let's cover an example of 3 documents" on this page. (Note: We are adding 1 to the log for our IDF computation.)

2. Next, you can test your code against test1.txt and test2.txt. Compare your resulting preprocessed documents with our results in preproc_test1.txt and preproc_test2.txt, and your TF-IDF results with our results in tfidf_test1.txt and tfidf_test2.txt.

3. Finally, you can try your code on these files: covid_doc1.txt, and covid_doc2.txt, and covid_doc3.txt. 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.

Note: When we test your submissionn, the input test file will be named `tfidf_docs.txt`, as stated at the start of Part 3. However, the file names contained in this file may be different than the samples given above (i.e. "doc1.txt", "doc2.txt", ..) so make sure that your code can read whatever file names appear in the `tfidf_docs.txt` file and work on them.