HW01: Matlab String Processing Exercises

Where to get help in Matlab?

1. Function **help**: the most useful function in matlab. By itself shows a list of available toolboxes.
2. **help strfun**: using help on a toolbox, shows a list of available functions in that toolbox.
3. **help strfind**: using help on a particular function, will give information about how the function is used and often show examples of how to use the function.
4. **Google**: Search engine is always your best friend while coding.

1 Exercise One – 40 points

In one single matlab script file (exercise01.m), implement the following:

1. Create a variable, A, whose value is the string 'hello'.
2. Create a cell array, C, containing the strings: 'hello', 'goodbye', 'hola', 'hello hellen', 'helmet', 'hellorheaven', 'hillsboro', 'say hello', 'myfellow'. Each index in your cell array should contain one of these strings. Notice you may initialize a cell array by setting it to an empty cell, e.g. *C* = []; and you then can add a string to the end of the cell array by using *C*{*end* + 1} = '*hello*' ; .
3. Count how many times the *exact string* 'hello' occurs in your array. And then count how many times the *word* 'hello' occurs in your array. (Notice the difference). Useful functions: **for, find, strcmp, isempty, strfind**.
4. Find the most similar string to 'goodfellow' in your array, where the similarity is measured by the number of letters in common. Useful function: **intersect**. E.g. intersect ('good', 'goodfellow') -> 'dgo' .
5. Based on 1.4, can you suggest an improved similarity scheme by considering whether or not a character is in the correct position in addition? Provide your explanation in your documentation file **results.doc**. If you can implement your proposed design, it would give you extra points. Spoiler alert: Under such scoring scheme, when compared to 'goodfellow', 'good' will score higher than 'bye', but 'goodflow' will score even higher than 'good'.
6. Capture the outputs and save them in table format (see below) to a file called **results.doc**.

2 Exercise Two – 20 points

Repeat Exercise 1.5 above but instead of using similarity based on number of letters in common, use the function *stringdist* provided with the instructions. Read the instructions on how to choose the two function parameters. Compare the results to your method in Exercise One and tell us what parameter configuration work the best for you.

You may re-use the code from Exercise One by replacing the similarity function with a call to *stringdist*, or you may call *stringdist* for each pair of strings and save the outputs in a text file. The important part is the comparison of the results and the parameters chosen for *stringdist*. You then report the comparison in a tabular format and save it to **results.doc**.

Here is an evaluation table example.

	Exercise One		**Exercise Two**
Term	Similarity to 'goodfellow' by common letters	Similarity to 'goodfellow' by common letters with correct letter position considered	Similarity to 'goodfellow' by using *stringdist*
hello			
goodbye			
…			

3 Exercise Three – 40 points

The input for this exercise is the file **review.txt** provided with the instructions.

In one single matlab script file (exercise03.m), implement the following:

1. Read in the lines of the text file using the **fgetl** function. Store the lines of this file in a cell array. Useful functions: **for, fopen, fgetl**.
2. Preprocess your lines: Remove punctuations, convert to lower case, remove all stop words (the list of stopwords is provided in a txt file). You can do this before or after tokenizing the strings. Use function **lower** for lower case conversion.
3. Parse your stored lines of text into their constituent words using the **strtok** function. Store all of the words in the entire document in a cell array with one word per index.
4. Create a lexicon consisting of all of the unique words that have appeared in your input file. Useful function: **unique**.
5. Create a column vector representing how many times each lexicon word occurs in the document. This is technically a word vector representation for the document. Useful function: **zeros**.
6. Prune the column vector by removing the words that only occur for very few times. The threshold can be 1, 2, ..5. You make the choice that deems reasonable at your own discretion.
7. Report the two column vector versions in a table, both with and without pruning, into the **results.doc** file.

4 Submission Instruction

Please upload your code files **exercise01.m, exercise02.m** alongside the documentation file **results.doc**. Make sure your documentation results.doc has separated sections for each exercise.

PS. If you use other languages, i.e., Python, name your code files exercise01.py, exercise02.py, respectively.