Assignment 1

Information Retrieval and Web Search
Winter 2015
Total points: 100
Issued: 01/22/2015 Due: 02/06/2015

All the code has to be your own (exceptions to this rule are specifically noted below). The code must run on the CAEN environment without additional installation or additional files (except for the data files specified in the assignment).

You can discuss the assignment with others, but the code is to be written individually. You are to abide by the University of Michigan/Engineering honor code; violations will be reported to the Honor Council.

1. [50 points] Text preprocessing.

Write a Python program that preprocesses a collection of documents. You will test this program on the Cranfield dataset, which is a standard Information Retrieval text collection, consisting of 1400 documents from the aerodynamics field. The dataset is available from the class web page:
http://web.eecs.umich.edu/~mihalcea/courses/498IR/Resources/cranfield.zip

Programming guidelines:
Write a program called *preprocess.py* that preprocesses the collection. Assume that the documents are available in a folder whose name you will read from the command line. For testing purposes, use the *cranfieldDocs/* folder that you will obtain after unpacking the cranfield.zip archive.

Include the following functions in *preprocess.py*:

a. Function that removes the SGML tags.
Name: *removeSGML*; input: string; output: string

b. Function that tokenizes the text.
Name: *tokenizeText*; input: string; output: list (of tokens)

Your tokenizer should correctly address the following cases, among others.:

- tokenization of . (do not tokenize acronyms, abbreviations, numbers)
- tokenization of ' (expand when needed, e.g., I'm -> I am; tokenize the possessive, e.g., Sunday's -> Sunday 's; etc.)
- tokenization of dates (keep dates together)
- tokenization of - (keep phrases separated by - together)
- tokenization of , (do not tokenize numbers)

c. Function that removes the stopwords.
Name: *removeStopwords*; input: list (of tokens); output: list (of tokens)

Use the list of stopwords available on the class webpage
http://web.eecs.umich.edu/~mihalcea/courses/498IR/Resources/stopwords

d. Function that stems the words.
Name: *stemWords*; input: list (of tokens); output: list (of stemmed tokens)

You are allowed to use a publicly available implementation of the Porter stemmer, e.g.,
http://tartarus.org/~martin/PorterStemmer/python.txt

The main program should perform the following sequence of steps:
i. open the folder containing the data collection, provided as the first argument on the command line (e.g., *cranfieldDocs/),* and read one file at a time from this folder.
ii. for each file, apply, in order: *removeSGML*, *tokenizeText*, *removeStopwords*, *stemWords*
iii. in addition, write code to determine:

- the total number of words in the collection
- vocabulary size (i.e., number of unique terms)
- most frequent 50 words in the collection, along with their frequencies (list in reverse order of their frequency)

The *preprocess.py* program should be run using a command like this:
```
% python preprocess.py cranfieldDocs/
```

It should produce the following output:

```
Words  [total-number-of-words]
Vocabulary [total-number-of-unique-words]
Top 50 words
Word1 [frequency-of-Word1]
Word2 [frequency-of-Word2]
…
Word50 [frequency-of-Word50]
```

Write-up guidelines:
Create a text file called answers1.txt, and include the following information:

- Total number of words in the Cranfield collection.
- Vocabulary size of the Cranfield collection.
- Most frequent 50 words in the Cranfield collection, along with their frequencies.
- The minimum number of unique words in the Cranfield collection accounting for 25% of the total number of words in the collection?
  Example: if the total number of words in the collection is 100,

and we have the following word-frequency pairs: airplane - 30 space - 10
clear - 8 cut - 7 etc. the answer to this question will be 1 (1 word accounts for 25%
of the total 100 words)

- Pick two subsets of the Cranfield dataset. Determine and report the number of words and the size of the vocabulary for the subsets you selected. Use this information to derive the K and beta parameters required by the application of the Heaps law. Use these values to predict what would be the vocabulary size if the corpus were to increase to 1,000,000 words, or to 100,000,000 words.

2. [50 points] Language identification.

Implement a language identifier, using the letter-based bigram probability model discussed in class, with add-one smoothing.

The dataset to be used for this assignment is available from the class web page:
http://web.eecs.umich.edu/~mihalcea/courses/498IR/Resources/languageIdentification.data.zip

Programming guidelines:
Once you unpack the data archive, you will obtain a folder called *languageIdentification.data/*. Store this data folder in the same folder as your program. There are five files included in the *languageIdentification.data*/ folder: three files with data to use for training (*English*, *French*, *Italian,* stored under a subfolder called *training*/), one file with data to use for test (*test*), and one solution file to use for evaluation (*solution*).

Write a Python program called *languageIdentification.py*. The program should include the following functions:

a. Function to train a bigram language model.
Name: *trainBigramLanguageModel;* input: string (training text in a given language); output: dictionary with character frequencies collected from the string; output: dictionary with character-bigram frequencies collected from the string

Given an input string, this function will calculate the frequencies for all the single characters and for all the bigram characters in the string.

b. Function to determine the language of a string
Name: *identifyLanguage;* input: string (text for which the language is to be identified); input: list of strings (each string corresponding to a language name); input: list of dictionaries with single character frequencies (each dictionary corresponding to the single character frequencies in a language); list of dictionaries with bigram character frequencies (each dictionary corresponding to the bigram character frequencies in a language); output: string (the name of of the most likely language). Note: in the input lists, elements at a given position K in the lists correspond to the same language L.

The main program should perform the following sequence of steps:
i. Use the *trainBigramLanguageModel* function to build unigram and bigram dictionaries for each of the three language files provided as training.
ii. Open the test file, provided as the first argument on the command line, and for each line in the test file, apply the *identifyLanguage* function.

The *languageIdentification.py* program should be run using a command like this:
% *python languageIdentification.py test*

It should produce the following output:

```
Line1 Language1
Line2 Language2
…
Line300 Language300
```

(where LineN represents the Nth line in the test file, and LanguageN is the language determined as the most likely one by the *identifyLanguage* function)

Write-up guidelines:
Create a text file called answers2.txt, and include the following information:

- Accuracy of your language identifier when comparing the output of your system with the solution provided? (in other words, what is the percentage of predictions that are correct)

General submission instructions:

- Include all the files for this assignment in a folder called *[your-uniqname].Assignment1/* **Do not** include the data folders, i.e., *cranfieldDocs/* and *languageIdentification.data/*
  For instance, lahiri.Assignment1/ will contain answers1.txt, answers2.txt, preprocess.py, file_with_Porter_stemmer_code, languageIdentification.py
- Archive the folder using zip and submit on CTools by the due date.
- Make sure you include your name and uniqname in each program and in the answers*.txt files
- Make sure all your programs run correctly on the CAEN machines.