Heterogeneous and Cloud Computing

Homework #1

Max Points: 35

Due: Wed, Sept 17 2014 By 5:00 PM (EST)

Email-based help cut-off: 24 hours prior to due date

Note: If you are using your personal machine then it is highly recommend that you install Linux on a Virtual Machine and use the Linux virtual machine for your work.

Objective: Develop a program to identify significant terms in a document using a given corpus with the added purpose of:

- Gaining familiarity with standard C++ libraries
- Developing C/C++ programs on a Linux machine.
- Refreshing problem solving skills learnt in undergraduate education

Submission: Once you have successfully tested your program and verified it meets all the formatting requirements, upload just your C++ source file onto Niihka:

- ***MUid*_tfidf.cpp**: A C++ program to display significant terms and sentences in a given document. (*MUid* is your Miami University Unique ID).

Grading Rubric:

This is a graduate course and consequently the expectations in this course are higher. Accordingly, the C++ program submitted for this homework must be operational in order to qualify for earning a full score.

NOTE: Program that do not compile, have methods longer than 25 lines, or just some skeleton code will be assigned zero score.

-5 points: The methods are not implemented concisely and efficiently by using good programming practices and suitable algorithms. Error conditions don't need to be handled and you can just throw an exception.

-1 point: For each style violations as reported by the C++ style checker (a slightly relaxed version from Google Inc.)

-3 points: If the program does not include suitable comments at appropriate points in each method to elucidate flow of thought/logic in each method.

-1 point: For each warning message generated by g++ when compiling with `-Wall` (report all warnings) flag.

-1 point: For each difference in outputs between your program and expected output.

C++ Programming efficiently in Linux

You may use any editor or IDE that you are familiar with for programming. However, it is highly recommended you get comfortable using `emacs`. It is a powerful and rich environment and a good compromise between various tools, particularly when running over a network connection.

Copy my `emacs` configuration

To provide consistent formatting and use of `emacs`, copy my `emacs` configuration into your home directory using the following commands:

```
$ cd
$ rm -f .emacs
$ wget -q http://pc2lab.cec.miamiOH.edu/documents/.emacs
```

Compiling C++ Programs from `emacs`

We will be developing straightforward C++ programs that are relatively straightforward to compile and run. However, the compilation and style checking operations can be streamlined to make development easier. Accordingly, a simple generic `Makefile` is supplied off Niihka in the `Resources→Handouts` folder. Note that, "hand writing" `Makefile` is archaic and should be avoided in any "real" project. The supplied `Makefile` provides the following features:

- Use with `emacs`'s flymake mode (to dynamically highlight compile errors)
- Compile one C++ source file (you can modify it to compile many)
- Download (if needed) and run the style checker on the given source file.

Usage:

The supplied `Makefile` can be used in the following manner in `emacs`:

- From the `emacs` menu choose `Tools→Compile` (or use keystroke CTRL-C C)
- In the mini buffer (at the bottom of `emacs` window) modify the `make -k` default command to:
  - Working with a one C++ source file: `make SRC=hello.cpp`. Change `hello.cpp` to the C++ source file you are working with.
  - Working with all source file(s) in current directory: `make many EXE=hello`. Change `hello` to the executable you would like to generate.

Background on TF-IDF

Retrieving data based on a user-defined query has become ubiquitous in recent years and rarely do user's think twice about Googling for information and expecting significant terms or documents to be retrieved. Several different weighting schemes have been proposed for identifying significant terms in a given document based on a corpus of data. This homework will utilize a simple weighting scheme called Term Frequency Inverse Document Frequency (TF-IDF). Though TF-IDF is a relatively old term weighing scheme, it is simple and effective, making it a popular starting point for other, more sophisticated algorithms.

TF-IDF for a given term (or word) w_d in an individual document d in a corpus (collection of documents) D ($d \in D$), is computed using the formula:

$$w_d = f_{w,d} * log\left(\frac{|D|}{f_{w,D}}\right)$$

Where,

- w_d is TF-IDF of term (or word) w in document d ($d \in D$)
- $f_{w,d}$ is the number of times the term occurs in d
- $|D|$ is the number of documents in the corpus
- $f_{w,D}$ is the number of times the word w occurs in the corpus D.

Homework Program Requirement

Develop a C++ program that accepts the following inputs as command-line arguments in exactly the following order:

1. *Corpus-file*: The first argument will be path to a corpus-file that contains the list of text documents (one per line) that constitute the corpus. See supplied corpus files `number_corpus.txt`, `prog_paradigm_corpus.txt`, `constitution_corpus.txt` for details/examples. The file corresponds to the corpus D in the formula.
2. *Search-file*: The second argument will be path to a text document whose significant terms are to be printed. This argument corresponds to d in the formula.
3. *N-terms*: This is an integer value that specifies the N significant terms to be displayed and used.

And prints the following results:

1. The top N significant terms in the specified *search-file.*
2. The sentences in the *search-file* that contain 2 or more significant terms.

Term generation:

- Each term is separated by a white space. Consequently, the number of terms in a *file* is the value reported by "`wc -w file`".
- For TF-IDF calculations all terms must be converted to lowercase to reduce entropy.
- Ignore terms that have numbers in them (example: ignore terms like "`a70`" "`b52`" etc.)
- Ignore punctuation marks other than hyphens in terms. For example the term "`Object-Oriented;,`" should be converted to "`object-oriented`" for TF-IDF calculations.
- Ignore terms that start with the letters "`http`".

Expected sample outputs

Expected outputs from your program for different corpora are shown below. Note that the lines with significant words are shown in the format: "*Line_number*[*#Sig.Words In Line*]: "

```
$ ./raodm_tfidf number_corpus.txt one.txt 10
Loading corpus using files listed in number_corpus.txt
Loaded corpus of 10 words from 11 file(s).
------[ Starting analysis ]------
Top 10 significant words...
one: 0.740363
You have mail in /var/spool/mail/dmadhava
[dmadhava@dragon tfidf]$ emacs -nw /home/dmadhava/hackArea/c++/tfidf.cpp
[dmadhava@dragon tfidf]$ ~/hackArea/c++/tfidf number_corpus.txt one.txt 10
Loading corpus using files listed in number_corpus.txt
Loaded corpus of 10 words from 11 file(s).
------[ Starting analysis ]------
Top 10 significant words...
one: 0.740363
Lines with 1 or more significant words:
```

```
$ ./raodm_tfidf prog_paradigm_corpus.txt Functional_programming.txt 9
Loading corpus using files listed in prog_paradigm_corpus.txt
Loaded corpus of 16572 words from 110 file(s).
------[ Starting analysis ]------
Top 9 significant words...
cufp: 9.78435
fibonacci: 7.13082
higher-order: 6.85774
combinatory: 6.31621
pure: 6.1824
burstall: 5.98147
fibonaccin: 5.75733
fibrecurrence: 5.75733
first-class: 5.50031
Lines with 1 or more significant words:
248[2]:    that higher-order functions and first-class functions both allow
```

```
$ ./raodm_tfidf constitution_corpus.txt First_Amendment_to_the_United_States_Constitution.txt 10
Loading corpus using files listed in constitution_corpus.txt
Loaded corpus of 62352 words from 296 file(s).
------[ Starting analysis ]------
Top 10 significant words...
jasper: 17.4573
libel: 15.8082
malice: 14.7129
obscenity: 13.7889
fec: 13.5735
highbeam: 13.3263
schenck: 12.8691
defamation: 12.6717
pornography: 12.1602
subscription: 10.6788
Lines with 1 or more significant words:
110[2]:    burden of proof for defamation and libel suits, most notably in new
839[2]:    obscenity or pornography. while the supreme court has generally refused
840[2]:    to give obscenity any protection under the first amendment, pornography
883[2]:    question of obscenity to local authorities.^[142] child pornography is
978[2]:    redefined the type of "malice" needed to sustain a libel case. common
```

Validation (also used for grading)

A compiled version of the sample solution is supplied for validation of your programs. You may verify correct operation and output of your solution using the following procedure:

```
$ ./raodm_tfidf constitution_corpus.txt Fourth_Amendment_to_the_United_States_Constitution.txt 10
> expected_output.txt

$ ./MUid_tfidf constitution_corpus.txt Fourth_Amendment_to_the_United_States_Constitution.txt 10
> MUid_output.txt

$ diff expected_output.txt MUid_output.txt
```

The `diff` command above should not report any differences between your outputs and the expected outputs.

Downloading corpus documents

The files associated with `number_corpus.txt` is already given to you. Use this corpus for your initial development and testing. The data for other corpora are to be downloaded from Wikipedia using the supplied script as shown below:

```
$ ./build_wiki_corpus.sh prog_paradigm_corpus.txt
$ ./build_wiki_corpus.sh constitution_corpus.txt
```

Help and questions

Use Niihka forums for soliciting clarifications regarding the homework, discussions, soliciting generic help, etc. Prefer to use the Niihka forums to Internet resources (particularly `stackoverflow.com` is a not the best resource for C++)

Submission

Once you have successfully tested your program and verified it meets all the requirements, upload just your C++ source file onto Niihka:

- ***`MUid`*`_tfidf.cpp`**: A C++ program to display significant terms and sentences in a given document. (*`MUid`* is your Miami University Unique ID).

Verify that your program meets all the requirements as stated in the grading rubric. No credit will be given for submitting skeleton code or programs that do not meet the base case. Ensure your C++ source files are named with the appropriate conventions. Upload each file individually. Do not upload zip/7zip/tar/gzip or any other archive file formats.