COMP 345 Advanced Program Design with C++

Fall 2017

Assignment #4

Due date (Moodle Submission): November 21
Due date (Lab Demo): November 22 & 29

Task 4: "Recommender" (8%). Recommendations are everywhere: Amazon recommends books you should buy, Netflix tells you which TV series you should watch, and Google recommends news articles to read. Time to write your own C++ program that can *recommend* items!

For this assignment, you have to work on *movie recommendations* (though the program really works with any content). You have to be able to:

1. Read a set of movie descriptions into an index;
2. Ask a user about a movie s/he likes (which must be one in the index);
3. Recommend a list of top-n other movies that s/he would also like.

Dataset. Your program has to work with the CMU movie dataset,[1] which you can download from http://www.cs.cmu.edu/~ark/personas/data/MovieSummaries.tar.gz. This dataset contains 42,306 movies in the file `movie.metadata.tsv`, including a summary for each movie that you can find in the file `plot_summaries.txt`. The important parts in these files for the purpose of this assignment are (details are in the `README.txt` file included with the dataset):

File `movie.metadata.tsv`:

```
Wikipedia movie ID   Freebase movie ID    Movie name        Release Date    ...
975900               /m/03vyhn            Ghosts of Mars    2001-08-24      ...
...
```

File `plot_summaries.txt`:

```
Wikipedia movie ID   Plot description
975900               Set in the second half of the 22nd century...
```

Indexing. Use your classes from the first three assignments to build an index, where each movie m is represented through a vector $\vec{m}$ that you create from the movie's plot description.

Processing the query. Your system takes as input the name of a movie. If the movie exists in the index, you compute the similarity between the input movie's vector and the other movies in the index, using the same similarity function as before.

Recommendation. Output the top-n best (default 5) movies as recommendation (include movie name, release date, ID#, and plot description in the output).

Congratulations, you now have your very own *recommender system*. The method used here is called *content-based recommendation*, which is one of many recommendation methods in use today.[2] Another approach is *collaborative filtering*, which you might have heard about in the context of the 2009 competition organized by Netflix for the best movie rating prediction,[3] where the winner was awarded a prize of US $1,000,000.

[1] CMU Movie Summary Corpus, http://www.cs.cmu.edu/~ark/personas/

[2] If you want to learn more about recommendation engines, you can start with the Wikipedia page https://en.wikipedia.org/wiki/Recommender_system and for a lot more details, download *Recommender Systems: The Textbook* through the Concordia Library: http://mercury.concordia.ca/record=b3281624.

[3] The Netflix Prize, see https://en.wikipedia.org/wiki/Netflix_Prize

Coding guidelines. Develop your program according to the following specification:

a) For all classes, make sure you properly separate your system into *header* (`.h`) and *implementation* (`.cpp`) files. Put each class into its own translation unit. You are free in the choice of an IDE, but your code must be standard, cross-platform C++ code.

b) Document all your classes and functions with *Doxygen*.

c) For your classes, follow object-oriented design principles as discussed in the course; in particular make data members `private` unless you have a good reason not to; use `friend` functions where appropriate to access private members; access `private` members in derived classes through `protected` functions, and make proper use of inheritance (e.g., use `virtual` functions for polymorphism and do not override non-virtual functions in `public`ly derived classes). **New:** for all classes, make use of exception handling for any error cases.

d) Write four separate `main` programs, using the *same* classes (see *e)* below): a new `recommender.cpp` for Task 4, as well as updated versions of `summarizer.cpp` (Task 3), `googler.cpp` (Task 2) and `indexing.cpp` (Task 1). You will have to demo all four main programs.

e) Add these two new classes to your code from the previous assignments:

Class `movie` is a new subclass of `index_item` that holds the information for a movie: `ID`, `name`, and `release_date`, together with accessor methods. The `content` field holds a movie's plot description.

Class `index_exception` is a new exception class, which is a subclass of the standard class `std::exception`. Override the virtual function `what()` to provide an explanation of the exception. Change any existing code in your classes to use the new exception class in case of errors. Make sure you process the exceptions (`try...catch`) so that your `main` program does not terminate. In particular, for Task 4, asking for recommendations based on a movie name that is not in the index will result in an exception.

Note that you will have to find a way to combine the information about a movie (name and plot) through the movie ID from the two input files.

You code must make use of polymorphism, where appropriate. For all these classes, overload the inserter (`operator<<`) to provide meaningful debug output. You can add additional classes if you like, but these must not duplicate the functionality of the classes above. As for the previous assignments, you are responsible for coming up with an object-oriented design that makes good use of these classes, so that they collaboratively solve the stated tasks (e.g., using the mentioned CRC method).