1 Movie Recommendations [35]

If you have used Amazon, Netflix, Pandora, or Last.fm, you are probably familiar with recommendation systems. The idea is that based on some criteria like past user selections, the system will recommend new products. Recommendation systems can be quite sophisticated. In this problem, we will ask you to consider part of a very much simplified system which will determine similar users based on their movie recommendations.

In our system, viewers rate movies on a scale of 1 (terrible) to 5 (excellent). Not all users rated each movie. If a user didn’t rate a movie, we will assume that user did not view the movie and the movie’s rating is 0. There are many measures of similarity or dissimilarity that could be used. In cases where the data is somewhat sparse, a measure called cosine similarity is sometimes used. To measure the similarity between two users, we use cosine similarity, defined:

\[ \text{similarity}(x, y) = \frac{x \cdot y}{||x|| ||y||} \]

\(x \cdot y\) is the dot product of the ratings of users x and y. It is defined \(x \cdot y = \sum_{i=0}^{\text{movies.size()-1}} x_i y_i\) where \(x_i\) is the rating that user x gave movie number i. That is, the dot product is the sum of the products of the ratings for the two viewers:

\[ \text{user}_0 \text{.movie}_0 \text{.rating} \times \text{user}_1 \text{.movie}_0 \text{.rating} + \text{user}_0 \text{.movie}_1 \text{.rating} \times \text{user}_1 \text{.movie}_1 \text{.rating} \ldots \]

\(||x|| = \sqrt{\sum_{i=0}^{\text{movies.size()-1}} x_i^2}\) is the size of the rating vector. Similarity varies from 0 to 1. Users with similar taste should have a high similarity in their ratings.

The system will require two classes: a class for a movie name and its rating, and a class for movie viewers.

The Rating class is defined as:

class Rating {
public:
    Rating(const std::string& name, int rating) : movie_name_(name), rating_(rating) {} 

    const std::string& getName() const {return movie_name_;}
    int getRating() const {return rating_;}

    void setRating(int rating) {rating_ = rating;}
}

private:
    std::string movie_name_; 
    int rating_;}

1.1 Movie Viewer Class [11]

First create the class definition for the movie viewer class. The movie viewer class should hold a movie viewer’s name and a vector of that viewer’s movie ratings, i.e. a vector of Rating objects. It must have an accessor function to return the viewers name and a function that is passed a movie name and returns the viewer’s numerical rating of that movie. If the viewer did not rate a movie, its rating is zero.

The viewer class must also have a function to add movie ratings. If the viewer has already rated a movie (viewers sometimes rate a movie more than once), the current rating is replaced by the new rating. If the viewer has not yet
rated the movie, the new rating is added. You may define helper functions as needed. Don’t worry about `#include` or `#define` statements.

Solution:
```cpp
class MovieGoer {
public:
    MovieGoer(const std::string& name) {name_ = name;}

    // Accessors
    const std::string& getName() const {return name_;}
    int getRating(const std::string& movie_name) const;

    // Modifiers
    void addRating(const Rating& rating);

private:
    std::string name_;  // private
    std::vector<Rating> ratings_;  // private
};
```

1.2 Movie Viewer Class Implementation [14]

Now implement the constructors, member functions, and any helper functions as they would appear in the `.cpp` file.

Solution:
```cpp
int MovieGoer::getRating(const std::string& movie_name) const {
    for (unsigned int i = 0; i < ratings_.size(); ++i) {
        if (ratings_[i].getName() == movie_name) {
            return ratings_[i].getRating();
        }
    }
    return 0;
}

void MovieGoer::addRating(const Rating& rating) {
    // arg doesn’t have to be a Rating object,
    // could be a movie name and rating
    for (unsigned int i = 0; i < ratings_.size(); ++i) {
        if (ratings_[i].getName() == rating.getName()) {
            ratings_[i].setRating(rating.getRating());
            return;
        }
    }
    ratings_.push_back(rating);
}
```

1.3 Similarity Implementation [10]

Now, implement the Similarity function. This function is not part of the movie viewer or rating class. It is passed two movie viewer objects and a vector of movie names. It should return a double representing the similarity between the two viewer’s ratings.

Solution:
```cpp
double similarity(const MovieGoer& mg1, const MovieGoer& mg2,
                  const std::vector<std::string>& movies) {
    double n1 = 0;
    double n2 = 0;
    double dot = 0;
    for (unsigned int i = 0; i < movies.size(); ++i) {
        n1 += pow(mg1.getRating(movies[i]), 2);
        n2 += pow(mg2.getRating(movies[i]), 2);
        dot += mg1.getRating(movies[i]) * mg2.getRating(movies[i]);
    }
    return dot / (n1 * n2);  // Cosine similarity
}
```
2 Parsing Symbols [ /20]

A consultant has decided to make a new programming language. This is a pretty complicated task, so they have decided to ask for help from several people, including you. Since you’re a Data Structures student, they think you can help them with a task called parsing, which is the process of reading in code and breaking it up into pieces the compiler understands. However, even the task of parsing is difficult, so he’s asked you to just do a smaller set of problems that are about handling the results of the parser. One job of the parser is matching opening symbols like (, [, {, and closing symbols like ), ], and }.

To make the matching rules easy, the consultant has put all the opening symbols into one vector and all the closing symbols into another vector. The closing symbol that matches opening symbol i (the ith element in the opening symbol vector) is stored in position i of the closing symbol vector. For example, the two vectors for the symbols we’ve mentioned so far are illustrated below. Keep in mind that they may add more symbols, so you must write code that allows for any number of symbols.

opening_symbols:  ( [ { closing_symbols: ) ] ]

Our goal will be to interpret a parses data structure that has the indices of all pairs of matching symbols in a given line of input. We can assume that we’re given partially processed input in the form of a vector of strings. Every entry in the vector is either an opening symbol, a closing symbol, or input we can ignore. The parses data structure is a 2-D vector. Each entry in the parses vector is a vector of unsigned integers (positions). The vector of positions always has an even length, with even indices (0, 2, 4, . . .) containing the position of an opening symbol in the input, and odd indices containing the position of the closing symbol that matched it. Note that each symbol may only match another symbol once, so there should be no duplicate entries in the parses vector. Additionally, an opening symbol can only match a closing symbol that is further right in the input vector.

As an example using the above opening_symbols and closing_symbols, the input vector (entries separated by spaces)

vector: [ 3 + 41 ] + ( 8 + 9 ) + ( x + y ) = [ 11 + 50 ]
index:  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

Has a parse vector like:

(): 6 10 12 16
{ }: <empty>
[ ]: 0 4 18 22

For this problem you may not use std::find. You are only allowed to use subscript indexing (e.g. a[5]) for accessing arrays and vectors. You should use no pointers and no iterators. You can always assume that the opening symbols and closing symbols two vectors will match in length.

2.1 Printing Output [ /12]

The first order of business is to write a function called PrintParses that takes in a vector of opening symbols called opens, a vector of closing symbols called closes, a partially processed input vector, and a matches vector called parses. This function should loop through the parses vector, and write the output like shown below to standard output.

The example on the previous page will have the following output:

( )
6 10 8 + 9
12 16 x + y
{ }

3
0 4 3 + 41
18 22 11 + 50

Solution:
//Could also use const std::vector<char>& for opens/closes. Need std::string for input
void PrintParses(const std::vector<std::string>& opens, const std::vector<std::string>& closes,
                 const std::vector<std::string>& input,
                 const std::vector<std::vector<unsigned int>>& parses){
    for(unsigned int i=0; i<opens.size(); i++){
        std::cout << opens[i] << " " << closes[i] << std::endl;
        for(unsigned int j=0; j<parses[i].size(); j+=2){
            std::cout << parses[i][j] << " " << parses[i][j+1];
            for(unsigned int k=parses[i][j]+1; k<parses[i][j+1]; k++){
                std::cout << " " << input[k];
            }
            std::cout << std::endl;
        }
    }
}

2.2 Vector Search [ 8]

Before tackling the big problem, a more senior programmer on the parsing team has suggested you write a helper
function for them called WordInVector. This function will take in four arguments: a vector of strings to search
through, a string to look for, an unsigned integer we will store the position in, and an unsigned integer we will use
as a start index. The function should return true if the string we’re looking for is in the vector of strings, and the
position is not smaller than the start index.

If the word cannot be found in the vector, or can only be found before the start index, the function should return
false and the behavior of the position variable is undefined; in other words when returning false, the position can
be set to any value. If the function returns true, the position variable should be set to the index of the first match
that is on or after the start index.

For example:
//vec contains: Hello world goodbye world

unsigned int pos,pos2,pos3;
WordInVector(vec, "world", pos, 2);
WordInVector(vec, "world", pos2, 1);
WordInVector(vec, "there", pos3, 1);
std::cout << pos << " " << pos2 << " " << pos3 << std::endl;

This will print out: 3 1 <any unsigned integer>

Solution:
bool WordInVector(const std::vector<std::string>& vec, const std::string& word, unsigned int& position,
                  unsigned int start_position){
    for(unsigned int i=start_position; i<vec.size(); i++){
        if (vec[i] == word){
            position = i;
            return true;
        }
    }
    return false;
}

3 Bug Catching [ 9]

The fragments of C++ code below contain errors. Your job is to first describe the error in one or two concise and
well-written sentences and then provide an appropriate correction that eliminates the error.

int* p;
int* q = p;
p = new int;
*p = 55;
std::cout << *q << std::endl;

Solution: This code contains a dereference of an uninitialized pointer. This may cause a segmentation fault at runtime, or unexpected output. Add q = p before the cout statement to fix the problem, change *q to *p or put a value in q before new p.

std::vector<std::string> pets;
pets.push_back("cat");
pets.push_back("dog");
pets.push_back("elephant");

Solution: An attempt was made to reference a vector element that was not allocated. The solution is to reduce each index by 1. There was also extra > on the first line, removing the extra > was another solution.

std::vector<std::string>& Vectorfy(const std::string& s) {
    std::vector<std::string> v;
    v.push_back(s);
    return v;
}

Solution: The function is returning a reference to a local variable. Return a copy.

std::vector<std::string> Vectorfy(const std::string& s)

4 Memory [ /24]

Consider the following code:

```c
int **a;
a = new int*[4];
a[0] = new int;
a[1] = new int[3];
a[2] = new int;
int b = 25;
int c[3] = {1,10,100};
*a[2] = 15;
a[0][0] = 8;
a[2] = &b;
a[2][0] = 1986;
a[2] = &c[1];
```

4.1 Memory Diagram [ /18]

First, draw a memory diagram for the above code. Do not erase lines for pointers that change, instead put an x over the middle of old line, and then draw the new pointer line.

4.2 Cleaning Up Memory [ /6]

Right now the code leaks any memory still allocated. Write 3 delete statements that should go immediately after the provided code.

Solution:
```c
delete a[0];
delete [] a[1];
delete [] a;
```
Is there another leak? If there is, how would you fix it?

Solution: Yes, \( a[2] = &b; \) will cause a leak. To fix it, just before this statement add the line of code below.

\[ \text{delete } a[2]; \]

5 iClicker Replay [ /9 ]

Grading Note: -2pts each unanswered or incorrect.

5.1 Which of the following statements is \textit{false}?

(A) The STL sort routine can alphabetize STL strings.
(B) If you have a big dataset and you care about performance, you should write your own sort routine because the STL vector sort routine is inefficient.
(C) The STL sort requires use of something called an \textit{iterator}, and apparently we are expected to use them without knowing much of anything about them. (So this why everyone says this class is impossible and the Data Structures instructors have unreasonable expectations!)
(D) I can store multiple copies of the same value in an STL vector and if that vector is then sorted, those duplicates will still be there, just clustered together.
(E) When you use the STL sort routine to organize a collection of integers, small values will be at the front, big values at the end.

Solution: B

5.2 Which of the following statements is \textit{false} about the keyword \texttt{const} and the symbol \& in C++?

(A) The keyword \texttt{const} tells the compiler that we do not intend to modify \textit{something} and then the compiler helps protect against accidental changes to this data.
(B) Compiler error message related to mismatches in const & reference are long and sometimes confusing to decipher, so it’s OK to ignore them.
(C) You should use pass-by-reference when the data you are sending to a function might be big (e.g., STL strings & STL vectors).
(D) When the data is small (\( \leq 8 \) bytes), it’s OK to pass-by-value (a.k.a. pass-by-copy), unless you want the function to modify it.
(E) The meaning of the single \& has nothing to do with the meaning of the double \&

Solution: B

5.3 Which of the following is a correct use of the \textit{hash} or \textit{pound} symbol, #, in C++?

(A) When modules from STL or other libraries are used we include the library header file within “double quotes”.
(B) When a custom type (C++ class) is used we include the custom header file within <angle brackets>.
(C) Just like Twitter, it is used to tag topics or modules within the program.
(D) It is the most convenient way to convert strings to numbers (integer or floating point).
(E) Simple if/else logic can be performed with the preprocessor before compilation & linking begins.

Solution: E

5.4 What is \textit{not} a reason for making the member variables of a C++ class private?

(A) This is an \textit{abstraction barrier}, meaning the external user does not need to understand the details of the internal representation.
(B) It facilitates future changes to the internal representation to improve efficiency or add new functionality.
(C) It is a security measure that prevents accidental or erroneous modification of the object by external users of the class.
(D) If you don’t specify, everything about a C++ class will be public. In contrast, by default, everything is private for a struct (inherited from the C programming language).
(E) Because the instructor said we should always* make them private.

Solution: D

5.5 Which of the following pairs of code are \textit{not} equivalent?

\[ \text{float } a[5] = \{ 3.14, 6.02, 2.18, 212.0, 42.0 \}; \]
\[ \text{float } *p; \]

(A) \( a[2] \) \( (*a+2) \)
(B) \( a[2] \) \( *(a+2) \)
(C) \( p = a; \) \( p = &a[0]; \)
(D) \( a[0] \) \( *a \)
(E) \( p = a+2; \) \( p = &a[2]; \)

Solution: A