INFO000–2: Markov Model

Pr Laurent MATHY, Gaulthier GAIN, Sami BEN MARIEM

February 21, 2023

1 Introduction

In this assignment, you will implement a simple classification algorithm based on Markov models. The principle is that you will assign probability to sequence of symbols and then, try to infer the author of a text sample.

2 Markov models

A Markov model is a probabilistic model that defines a probability distribution over sequences of symbols (whatever those symbols may represent). For simplicity, in this project, we will consider symbols to be *bytes* represented in C++ by the `char` type.

Given training data, a k^{th}*-order Markov model* computes the probability of occurrence of each input symbol, depending on the preceding k symbols. In other words, for each symbol sequence s of length k in the training data, the model estimates the probability that s is followed by symbol c as $N(sc)/N(s)$, where sc is the sequence resulting from the concatenation of sequence s and symbol c, and $N(.)$ represents the number of times the corresponding sequence occurred in the data.

Unfortunately, when using such a model, after training, for classification of input data, some of the occurrence counts will be zero if the input data contains symbol sequences that did not occur in the training data. To deal with this issue, we will instead use a process called *Laplace smoothing* where the above probabilities are computed as $(N(sc) + 1)/(N(s) + A)$, where A is the size of the symbol alphabet (i.e. the number of distinct symbols). Note that Laplace smoothing ensures that the probabilities of occurrence of a given symbol sums to 1.

Example If the input data is “abccbaabacba” and we are using a second-order Markov model, then the probability of the occurrence of “bcc” is $(1+1)/(1+3) = 1/2$, since $N(bcc) = 1$, $N(bc) = 1$ and the alphabet size is 3.

To deal with “limit conditions” at the beginning and end of the input, the input data is considered *circular*. As a result, the probability of occurrence of “aab” is $(2+1)/(2+3) = 3/5$, since $N(aab) = 2$, $N(aa) = 2$ and the alphabet size is 3.

Finally, the probability of the occurrence of “aaa” is $(0 + 1)/(2 + 3) = 1/5$, since $N(aaa) = 0$, $N(aa) = 2$ and the alphabet size is 3.

3 Markov model construction

Building a Markov model is essentially a counting problem. We will represent a Markov model by the following structure:

```
typedef std::set<char> Alphabet;
typedef std::map<std::string, unsigned int> Model;

struct Markov_model {
  unsigned int order;
  Alphabet alphabet;
  Model model;
};
```

In this structure, the `order` field stores the order of the model, the `alphabet` field stores the alphabet of symbols the model operates on, and the `model` field itself maps each string relevant to the model to its appearance count. Note that the `Markov_model` data structure must hold all the necessary information to allow the computation of Laplace probabilities for any appropriate input data.

In this first part of the project, you will write a function that builds a Markov model from an input string:

```
void markov_model(Markov_model&, unsigned int, const std::string&);
```

where the first parameter represents the model the function is building, the second parameter represents the model's order, and the third parameter is the training data on which the model is built.

This function must throw a standard `length_error()` exception if the input data and the order are incompatible in size.

4 Computing Laplace probabilities

The second part of the project is to write a function that, given a trained Markov model and an appropriate string, returns the probability of occurrence of that string:

```
double laplace(const Markov_model&, const std::string&);
```

Again, this function must throw a standard `length_error()` exception if the input data is incompatible in size with the model.

This function must also throw a standard `domain_error()` exception if the input data contains symbols not present in the alphabet of the Markov model.

5 Sequence likelihood

The third part of the project is about using a Markov model to measure how well this model fits some given input data. This fit will be measured as the *likelihood* of the input data according to the model, which is simply the probability that the model generated the input sequence. For each input symbol c, the likelihood of the symbol is simply the Laplace probability of observing c under the model, given the k previous characters (considering the input to be circular). The overall likelihood of observing the input sequence under the model is then simply the product of each input symbol's likelihood.

While the likelihood as described above is perfectly fine from a theoretical perspective, the resulting value may be very close to zero, risking to test the floating-point precision of C++ (even though we are using `double`).

Therefore to avoid any precision issues, we will use the mathematically equivalent definition of the likelihood to be the *sum of the (natural) logarithm of the Laplace probabilities* (which must then be a negative number, since the probabilities are never greater than 1).

The function to be written is:

```
double likelihood(Markov_model&, const std::string&);
```

where the first argument is a trained Markov model, and the second is the input for which the likelihood computation is required.

The Markov model facilities described in sections 3, 4 and 5 must be appropriately declared and defined in two files:

- header file: `markov_model.hpp`
- source file: `markov_model.cpp`

6 Application

Finally, you will write a test application that takes several input files as training data as well as several input files as test data and "attributes" each test data to the more likely model. The best model for the test data is, of course, the model that maximises the likelihood of this data.

The format of each training file is rather simple:

- it starts with the name of the model on a single line;
- the rest of the file is the training data.

While the test files simply contain the test data.

Your application (e.g. `testApp`) will take several *command line arguments* of the form:

```
./testApp 2 TD1 TD2 TD3 --- I1 I2
```

where the first argument (e.g. 2 the example above) specifies the order to be used for the Markov models; TDx is the x$^{\text{th}}$ training data file; Iz is the z$^{\text{th}}$ input test file; and -- indicates the separation of the training and test files. Note that there is an unspecified number (greater than 1) of each type of file.

For each test file, your application will

1. print out on standard output, on a separate line, the likelihood under each model in the *model_name*: *likelihood_value* format; if the test file is incompatible with the training data, simply output the symbol - instead of a likelihood value for the corresponding model.
2. print a final line indicating to which model the test input is attributed, in the *input_file_name* `attributed to` *model_name* format, if such model exists; output the symbol - instead of the model name otherwise.

The output must always list all models and all attribution results *in the order* in which the corresponding files appeared on the command line.

7 Remarks

7.1 No new classes or structures

In this project, you **must not** define any new `class` or `struct` (except for the given `struct Markov_model`, of course). If needed, you can use `pair` or `tuple`. Also try to reuse the STL algorithms as much as possible, rather than re-implementing them.

You can consult the Standard C++ Library Reference for further documentation about the STL data structures and algorithms, at `http://cppreference.com/`.

7.2 Respect the interface

Submission must scrupulously **follow the interface** defined above. You can of course define auxiliary functions as you see fit, but these should be properly hidden from the users of the interface.

The application must also follow the specified input and output formats.

7.3 Encoding

You can assume that your input is ASCII encoded (each symbol fits into a single char). `\n` and `\r` should never be symbols in the alphabet. In other words, avoid leaving these in your input strings.

The facilities required to manipulate files are defined in the `<fstream>` STL header file.

The facilities for the definition and manipulation of command-line arguments in C++ are identical to those found in C.

7.4 Readability

Your code must be readable:

- Make the organisation of your code as obvious as possible. Remember you can create as many auxiliary functions as you see fit.
- Use descriptive names for functions and variables.
- Complement your self-documenting code with comments, where appropriate.
- Choose a coding convention, and stick to it. *Consistency* is key.

7.5 Robustness

Your code must be robust. The `const` keyword must be used correctly, memory must be managed appropriately, the program must run to completion **without crash**.

7.6 Warnings

Your code must compile **without error or warning** with `g++ -std=c++17 -Wall` on `ms8??` machines. However, we advise you to check your code also with `g++ -std=c++17 -Wall -Wextra`, `clang++ -std=c++17 -Wall -Wextra` or tools like `cppcheck`.

7.7 Evaluation

Your code will be evaluated based on all the above criteria. Failure to comply with any of the points mentioned in **bold face** can (and most often will) result in a mark of zero!

8 Submission

Projects must be submitted through the submission platform before **Monday, 27th March, 23:59 CET**. Late submissions will be accepted, but will receive a penalty of $2^n - 1$ points (/20), where n is the number of days after the deadline (each day start counting as a full day).

You will submit a `s<ID>.tar.gz` archive of a `s<ID>` folder containing your `markov_model.{hpp|cpp}`, your test application (`main.cpp`) and your `Makefile`, where `s<ID>` is your ULiège student ID.

The submission platform will do basic checks on your submission, and you can submit multiple times, so check your submission early!