Assignment 1 - Audio and Video Coding

Pedro Alagoa [72534] Gonçalo Grilo Alexandre [72608]

I. Summary

This report describes the implementation and design of the text analysis tool developed for the class. The purpose of the tool is to extract statistical properties of any given text, such as the entropy, as well as generating text based on those properties. The tool was developed using Markov Models.

II. Running instructions

The tool is called **finite context model (FCM)** and can be used in a variety of ways, depending on the desired functionality. For example:

- Read input from the file *input.txt* and generate a model with order 4.

```
./FCM -i input.txt -k 4
```

- Load the model from the file *model.bin* and generate text with the length of 50 chars.

```
./FCM -r model.bin -l 50
```

There are other options that can be selected when using the tool.

Listing 1: Usage of the tool

```
1 Allowed options:
2   -h [ --help ]                         produce
        help message
3   -i [ --input ] arg                    input
        file
4   -k [ --order ] arg (=1)               set
        order of the model
5   -o [ --output ] arg (=out.txt)        output
        file
6   -l [ --output_length ] arg (=1000) length
        of the output file (in chars)
7   -s [ --save ] arg                     file
        where the model will be saved
8   -r [ --read ] arg                     file
        with the model to be read
9   -a [ --alpha] arg (=0.1)              alpha
        value
```

III. External libraries

This tool uses some libraries of the Boost C++ Libraries [1]. Those libraries are:

- **Serialization** - used to save and load the model from existing files.
- **Program options** - used to parse the input arguments given by the user.
- **Filesystem** - used when validating input arguments.

IV. Practical work

To store the data (a Markov chain) that forms the model, the most important structure that is being used is a *Unordered Map*.

The keys for this map are the contexts found in the analysed text. The lenght of the context is the order (k) of the model.

The value for each key is another map which key is a symbol that can occur after the given context and the value is the number of times that symbol occured after the given context.

```
std::unordered_map<std::string, std::
    unordered_map<std::string, unsigned int
    >> map;
```

A. Text generator

A.1 Description

The text generator takes the model and produces text based on it.

It generates the next symbol based on the **k** characters that came before that symbol.

A.2 Algorithm

As stated in IV, we use 2 unordered maps. The first map uses the context as the key and the value is another unordered map. This inner map has a symbol as key and the value is a counter of the number of times this symbol appeared in the context. All of the instances of the inner map contain all the symbols present in the given alphabet, initiallized to 0.

The inner map is initialized with all the counters at 1.

We consider all the alphabet(upper case and lower case), new line and spaces as valid symbols.

We choose the next symbol using a discrete distribuition of the occurences of the symbols observed during the construction of the model (plus the α value). We then generate a random value to choose which symbol is added to the text.
The code can be found in Appendix A.

A.3 Output

Here are some outputs of the text generator:

- *./FCM -i shakespeare.txt* ***-k 1*** *-l 150*

```
ioutem in ieaty hy a gher deessee
Ay nd be
BU y te w oous arou ham t f llont u l d
ncoreanoganisors t h MI weng
NAn rst h d we nd
NCow ff S ys iry PERY
```

- *./FCM -i shakespeare.txt* ***-k 3*** *-l 150*

```
That his thought ser and knew
PUCKING Perge crossed he is proclamber
Wher chill
CHIDA The daught
Hazarday take make
And fore cannot on my is in with p
```

- *./FCM -i shakespeare.txt* ***-k 5*** *-l 150*

```
ens the wood lady
Alas what please
it flower than shall do young
and far as early dear a greaten
as our honour best man to die
If it be dot Yea letter
```

By observing the generated text, we conclude what was expected - the higher the order, the better is the construction of words.

B. Entropy calculation

For calculating the entropy of the text, we are using the following expression.

$$H = (\sum_{c} P(c)H(c)) + P0 * \log_2 A^k \qquad (1)$$

- k - the order of the model.
- $P(c)$ - the probability of appearance of a given context.
- $H(c)$ - the entropy of a given context.
- $P0$ - this value is the probability of the contexts that did not appear in the input but that are possible given the considered alphabet. Its value is given by the following expression:

$$P0 = \frac{n0 \cdot a}{N + a \cdot A^k} \qquad (2)$$

- $n0$ - number of non discovered contexts that are possible given the considered alphabet.
- N - total number of occurrences of the known contexts.
- A^k - total number of possible contexts in the alphabet.

The code for calculating the entropy can be found in the Appendix A .

V. Benchmarks and Results

To make the tests, we compared the results of 3 different inputs of the same style (books).

- **shakespeare.txt** - The Complete Works of William Shakespeare.
- **thecallofthewild.txt** - The Call of the Wild, by Jack London
- **lusiadas.txt** - Os Lusíadas, by Luís de Camões.

A table with the results can be found in the Appendix B.
By analysing the results, we came to the following conclusions:

- Generally, the **entropy** is higher when the **input length** is larger.
  - **Exception**: For order *k=1*, this is not true, since the total number of contexts is the same, no matter the size of the input (in our case). The entropy could be higher given an input that didn't cover all possible contexts.
- Generally, the **entropy** is higher when the **order** is higher.
  - This only happens because the input size is small. As the order gets higher, the ammount of undiscovered contexts increases drammatically. As such the entropy also increases (in some cases, because the value of alpha is too high). Only on longer inputs (such as *shakespeare.txt*) the entropy decreases when going from order *k=1* to order *k=3*. This is because a longer text probably has more discovered contexts than a shorter one.
- The **entropy** is lower when the α value is lower.
  - A high α means that the unknown contexts of a given model will have a lot of weight when calculating the entropy or generating text. Thus, when the α value is lower, the entropy will also decrease.
  - **Exception**: For order *k=1*, the entropy does not change, because all possible contexts are know by the model.
  - We can also see that lowering the α value has more impact on lowering the entropy when the entropy is high.

VI. Conclusions

As we can see, the number of characters will affect the entropy of the model. In larger inputs, we can see that the entropy will be higher, for the same context size, if we don't count the α value.

The key for choosing the optimal value for α so that the entropy is as low as possible is a logarithmic function that takes into account the order of the model and the lenght of its input.

The resulting expression is an exponential decay function that could potentially estimate the optimal value for the alpha that would minimize the entropy:

$$\alpha = e^{-\frac{k}{n} * \lambda} \tag{3}$$

- **k** - the order of the model (k $\in \mathbb{N}$)
- **n** - the length of the input in character (n $\in \mathbb{N}$)
- λ - the death rate of the created markov chain.

When we increment the context size the entropy will decrease for longer inputs. This is not true if the context size is really large, so if that's the case, the α value should also be substantially low.

To conclude, we learned that this is an simple and efficient way to make a somewhat reliable text generator, given the appropriate inputs (order, input text, alpha).

REFERENCES

[1] David Abrahams Beman Dawes. Boost C++ Libraries. http://www.boost.org/. [Accessed: 14 October 2017].

Appendix A

Code

A.1 guessNext()

```
std::string FCM::guessNext(){
 // Set alpha to prevent loops from happening
 float alpha = 0.1;
 InnerCounter* inner_map = &map[current_context];
 std::string best_guess;
 std::vector<float> weights;
 std::vector<std::string> symbols;
 std::discrete_distribution<unsigned int> dist;

 //check if inner_map is empty or you get a segfault
 if (inner_map->empty())
  inner_map = &symbols_list;

 // Get a vector of the weights and a vector of the symbols
 for (auto it = inner_map->begin(); it != inner_map->end(); ++it)
 {
  weights.push_back(it->second+alpha);
  symbols.push_back(it->first);
 }
 // Create a discrete distribution using the weights
 dist = std::discrete_distribution<unsigned int>(weights.begin(), weights.end()
     );
 // Generate the next symbol using the distribution and a random number from
     0-1
 best_guess = symbols[dist(gen)];
 // Update the context
 current_context.erase(0,1) += best_guess;
 return best_guess;
}
```

A.2 getEntropy()

```
double FCM::getEntropy(){
 double sum = 0;
 std::string c_contx;
 double total_contex_ap = 0;
 double contx_ap = 0;
 for (unsigned int i = order; i < len; ++i)
 {
  c_contx = data.substr(i-order,order);
  sum += std::log2(probOfSymbol(c_contx, std::string(1, data[i]), &contx_ap));
```

```
  total_contex_ap += contx_ap;
 }
 double no = (std::pow(symbols_list.size(), order)) - map.size();
 return (-(1.0/(double)len) * sum) + ((no*alpha) / (total_contex_ap + std::pow(
     symbols_list.size(), order) * alpha)) * (std::pow(symbols_list.size(),
     order));
}
```

Appendix B

Results

File	Number of characters	Order	Time to create model	alpha	Entropy
thecallofthewild.txt	175168	1	0.0260747 s	0.1	3.37218
thecallofthewild.txt	175168	1	0.0260747 s	0.01	3.37218
thecallofthewild.txt	175168	3	0.0370383 s	0.1	29.0855
thecallofthewild.txt	175168	3	0.0370383 s	0.01	4.72285
thecallofthewild.txt	175168	5	0.0882417 s	0.1	294354000
thecallofthewild.txt	175168	5	0.0882417 s	0.01	97201000
lusiadas.txt	318298	1	0.0393808 s	0.1	3.26701
lusiadas.txt	318298	1	0.0393808 s	0.01	3.26701
lusiadas.txt	318298	3	0.0540751 s	0.1	10.8448
lusiadas.txt	318298	3	0.0540751 s	0.01	3.11439
lusiadas.txt	318298	5	0.170842 s	0.1	185371000
lusiadas.txt	318298	5	0.170842 s	0.01	45362300
shakespeare.txt	4830809	1	0.437128 s	0.1	3.56894
shakespeare.txt	4830809	1	0.437128 s	0.01	3.56894
shakespeare.txt	4830809	3	0.860404 s	0.1	2.36609
shakespeare.txt	4830809	3	0.860404 s	0.01	2.29915
shakespeare.txt	4830809	5	2.47319 s	0.1	6214400
shakespeare.txt	4830809	5	2.47319 s	0.01	629186

Table B.1: Program results