CAV - Solving the First Lab Work Assignment

Nuno Humberto Nuno Miguel Silva

***Abstract* –One of the most used approaches to express data dependencies depends on the usage of Markov chains. In lossless data compression, it is used a particular kind, called discrete time Markov chain or finite-context model (FCM). A FCM can be used to reap statistical data from an information source, putting a probability approximation to the symbols of a given alphabet, according to a context that is calculated over a finite and fixed number of past results.**

***Keywords* –data compression, text generation, entropy calculation**

I Scenario

I-A Description

In this project, it is supposed to develop a C++ class called FCM where it should have methods with the objective to reap statistical intel from texts using finite context models. The order of the model should be a parameter of the constructor. The FCM class should have methods to add the occurrence of a character to the model, returning statistical information about a symbol based on a context (current or passed as argument), calculating the entropy of the generated text, writing/reading the model to/from a file and printing data about the model on the terminal.

II FCM class

The class is initialized giving an integer representing the order of the finite context model. Apart from the order, the class is also supplied with the input dataset and a map that will be useful to add statistical information from the dataset. The executable is a file with the following usage: *input prog* <*file name*> <*model order*> <*desired generated text length*>. When executed, the program runs all implemented methods, so it gives a demonstration of what it was stated in the assignment tasks.
Here follows the methods that are implemented in the FCM:

II-A addToDataset

This is a simple method in which a string is passed to initialize the dataset. It can also be used to concatenate more data to the dataset. An alternative to this method was also implemented in which instead of a string, a char is passed as argument.

II-B printContextStatistics

This method is broken in two halves. The first part checks if there is enough data collected to calculate the statistics. If so, a context is retrieved and passed as argument to the second half of the method. Notice that there are two printContextStatistics methods where one of them gets a string as argument. This second method will be used, firstly to check if the retrieved context exists in the map, and secondly prints the occurrences registered after the given context. This method fulfills the requirement of being able to show statistics for the current context or for the context passed as argument (overloading).

II-C printModel

This function attempts to create a textual representation of the currently loaded model. In this representation, it will include every detected context, followed by every detected entry for such context and also every count of these entries. The representation is immediately printed to the user.

II-D calculateModel

This method parses the dataset and fills the map with the contexts and the number of occurrences of the respective following symbols. In order to do so, it goes through all symbols in the dataset, and at a given position, checks if the symbol exists in the map. If it doesn't, it will create a map, adding a key equal to the following symbol and initialize its value to 0. After this, it increments this value.

II-E dumpModel

Being able to dump the model to an external file was a requirement specified in the assignment description. This method writes the currently loaded model to a file, in which each line represents a key in the main data structure, along with its corresponding map. The exported model can be loaded again at any time, using the **loadModel()** method.

II-F loadModel

This method loads a model that was previously exported to a file. It generates a new main internal structure with all the entries present in the given file. For importing the model, each line is parsed individually, from which the context is extracted with the use of string splitting at the tab character. Each entry in the context is then extracted with the use of string splitting at every semicolon character.

II-G printEntropy

This function calculates the entropy of the currently loaded model. To calculate the entropy, the following formula is used.

$$H = \sum (P(c) \cdot H(c)) + P_0 \cdot \log A^k$$

Here, $P_0 = \frac{\alpha \cdot n_0}{N + \alpha A^k}$ where n_0 is the number of unseen contexts and N is the sum of the total of occurrences from all seen contexts.
Two alpha values must be provided for entropy calculation. When calculating the global entropy for each context, one of these alpha values is used to avoid unseen contexts to not be considered for entropy calculation at all. A second alpha value is also used for the calculation of the probability of

each character in a context. These alpha values should be distinct, must be provided by the user at runtime and act as a weight for the contexts with null occurrences. Higher alpha values will yield a higher weight for these contexts and care must be taken when high order values are chosen, since selecting high alpha values will provide a massive increase in total entropy.

II-H generateText

The method that produces text from the statistical intel receives an integer as argument that states the maximum length of the generated text.
This methods starts by choosing a context to start with. The complete this objective, it goes through the map and checks what context appears more often in the input text. After discovering this context, it is added to the start of result string where the produced text will start.
From this point, this method will write the rest of the text out of the given context. To achieve this task, it is used a second map to get statistical data from the generated text, and also a table is created to calculate the percentage of a given symbol in the created text.
To produce a next character in the result string, we go through the map and check if the context exists. If the context belongs in the map, it gets the symbol with most probability, respecting the weight and probability of the other symbols also appearing. This means that it isn't always the same symbol appearing after a given context. The algorithm gives attention to the amount of each symbol and will add a less likely character if the most presumable is outnumbered in the text.
If the context doesn't exist in the map, we look into the dataset and retrieve the most probable symbol. This strategy also takes in consideration the percentages of each symbol in the produced text. So if the most susceptible symbol appears too often it will choose the next presumable character.
Then the symbol is chosen, it is concatenated to the result string. This process is made until the result string gets a length not less than the stipulated size.

III Results

Finally, after the program has been fully developed, it was tested using some benchmarks. An input text was used as follows:

"He said the People of those Counties believed that the opposition to the Excise law–or at least that their dereliction to it, in every other part of the U. States was similar to their own, and that no Troops could be got to March against them for the purpose of coercion; that every acct. until very lately, of Troops marching against them was disbelieved; supposed to be the fabricated tales of governmental men; That now they had got alarmed; That many were disposing of their property at an under rate, in order to leave the Country, and added (I think) that they wd. go to Detroit. That no person of any consequence, except one, but what had availed themselves of the proffered amnesty; That those who were still in the opposition, and obnoxious to the laws, were Men of little or no property, cared but little where they resided; That he did not believe there was the least intention in them to oppose the Army; that there was not three rounds of ammunition for them in all the Western Country. He (I think Mr. Findley also) was apprehensive that the resentments of the Army might be productive of treatment to some of these people that might be attended with disagreeable consequences; on that account seemed to deprecate the March of it: declaring however, that it was their wish, if the people did not give proofs of unequivocal submission, that it might not stop short of its object"

The benchmarks produced strings of 100 symbols and the results were the following:

III-A Test 1

For a first approach, we used the given input text with an order of 1 and alphas of 0.1. The resulting entropy was 3.07596 and the generated text was:

" the of at warer in pro tonthas me sed denopest tithe osiotelemisound t atry be cof therevealan wein"

III-B Test 2

For a second approach, we used the same input text with an order of 2 and alphas of 0.1. The resulting entropy was 14.6149 and the generated text was:

" the to troople of tal in that te proppose was opeops them for tention and ted alarch at it not acco"

We can compare this result to the previous one and conclude that, for a higher order in the finite context model with the same alpha in the entropy calculation, we get a higher value of entropy.

III-C Test 3

For a third approach, we used the same input text with an order of 2, an alpha to calculate the global entropy for each context of 0.00001 and an alpha to calculate the probability of each character in a given context of 0.1. The resulting entropy was 1.61379 and the generated text was:

" the to troople of tal in that te proppose was opeops them for tention and ted alarch at it not acco"

The generated text was the same since the input text and the FCM order did not change. The entropy was much lower this time as the alpha factor that calculates the global entropy for each context was also much reduced.

III-D Test 4

For a third approach, we used the same input text with an order of 2, an alpha to calculate the global entropy for each context of 0.1 and an alpha to calculate the probability of each character in a given context of 0.00001. The resulting entropy was 14.5984 and the generated text was:

“ the to troople of tal in that te proppose was opeops them for tention and ted alarch at it not acco“

The generated text was the same since the input text and the FCM order did not change from the previous 2 examples. The entropy was slightly reduced this time but the alpha factor that calculates the probability of each character in a context was much lower. We can conclude that the alpha to calculate the global entropy for each context has a higher weight in the entropy calculation rather than the alpha that calculates the probability of each character in a given context.

IV Conclusion

The main objectives were achieved successfully. It was an interesting project to deepen our knowledge in the C++ programming language and also to study the purpose of data compression algorithms.

The main issues in the development process of this project were around the calculation of entropy and the implementation of a text generator based on the statistical information of our model. In general, and how these problems were resolved, having worked around these somewhat unfamiliar obstacles helped our growth as future engineers.

References

http://www.public.asu.edu/~gelderen/hel/gwjournal.html