CS1003 Practical 4: Streaming Data

Overview

In this practical I was asked to create a program that would perform a fuzzy string search across given e-books, taking in 3 arguments, and my program achieves just that. I have completed all these parts.

Design

Main Method

It felt like this programme would not really benefit that much from Object Orientated Programming, so all of this program is run within one class. The main method which takes in the string array of arguments, first checks it has received the correct number of arguments (that number being 3). If not, then it displays an error and quits the program. Once it is checked for the correct number of arguments it then stores them in 2 strings for the text files path and the search term, then it parses the 3rd argument to a float for the similarity threshold.

Now that the similarity threshold has been read in it checks that it is between 0 and 1 inclusive as the Jaccard index can be a minimum of 0 and a maximum of 1. It then creates an array of the search term split by spaces and takes the length of that array and stores it as the search length in words. Then the program sets up the spark connection using basic configurations. Then a java pair RDD is created that reads in all the text files with the whole text files method, the key being the file names. Then all the text from the books is put into one single java RDD by taking the values of the pair RDD. Then a new java RDD of strings is declared which is filled using the find phrases method. Then a new RDD is declared of String Array lists which is populated by mapping the phrases strings using the cut phrases method.

Then yet another java RDD is created which filters the final phrases from the previous RDD using the calculate Jaccard method and comparing it against the similarity threshold. Now that only the appropriate phrases have been mapped to this RDD they are streamed to an output in the terminal.

Find Phrases Method

This method is designed to find valid strings that could become phrases e.g., if the search term were 4 words long you could pull phrases to compare to out of a 20-word long string but not a 2-word long string. This method takes in the books RDD and the search length and returns the RDD of strings that can be broken into phrases. First it filters the phrases by using the get valid phrase lengths method and then simply maps the strings to clean up the data a bit. It then returns that RDD.

Get Valid Phrase Lengths Method

This method helps filter the phrases by returning a Boolean if the phrase is of a valid length i.e., greater than or equal to the search length. It is passed in a string phrase and the search length, cleans the data a bit just to be sure and then splits the phrase into words by splitting by spaces. It then checks if the length of the resulting array is greater than or equal to the search length if so, it returns true otherwise it returns false.

Cut Phrases Method

This method cuts the phrases into strings of the appropriate length i.e., the same number of words as the search term and returns them as an array list. It takes the string phrase to be cut down and the search length.

First it splits the phrase by spaces i.e., into individual words in the form of a string array and then declares an array list that will be used to store all the actual words as to stop any unwanted null or spaces left in the data. So, to clean up the data it loops through the phrase and if it a null string i.e., “” or a space then it is ignored otherwise it is added to the array list. Then the array lost is converted into a string array and replaces the values in the string array created from the split and the beginning of the method.

So next it checks if the current phrase is already of the appropriate length, if so, declare a string and loop through the words in the phrase making extra sure to avoid double spaces and the like by checking the data is clean and if so then concatenating it all together then trimming and adding that phrase to the array list after the loop. Otherwise, the phrase must be longer than the length, so it is cut into chunks of the same length using nested for loops concatenate it by looping through the first words of the search length then the next number of words equal to the search length and trimming them and then adding each of those to the array list. The array list is then returned.

Calculate Jaccard Method

This method takes in a single string phrase and the search term and generates bigrams of each of them, then using them to calculate the Jaccard similarity index between the strings and returning the result of that equation.

So, it starts by creating a set of strings for the bigrams, then it generates the bigrams for the phrase using a for loop and this code was taken from my previous code in CS1003 practical 1. It then does the same for the search term and then again using my own code from CS1003 Practical 1 the Jaccard index is calculated.

It then returns the result of that calculation.

Testing

What is being tested	Name of test method	Pre-conditions	Expected outcome	Actual outcome	Evidence
The basic functionality of the program.	Stacscheck is used.	NA	All tests will pass.	All the tests passed.	The output from stacscheck was: Testing CS1003 CS1003 P4 - Looking for submission in a directory called 'CS1003-P4': Already in it! * BUILD TEST - build : pass * TEST - queries/advice/test : pass * TEST - queries/not-found/test : pass * TEST - queries/sail/test : pass * TEST - queries/tree/test : pass * TEST - queries/what-50/test : pass * TEST - queries/what-62/test : pass * TEST - queries/what-75/test : pass 8 out of 8 tests passed
The check for inputting the incorrect number of arguments.	NA	NA	An error will display, and the program will quit.	The expected outcome.	The input: time java -cp "/cs/studres/CS1003/0-General/spark/*":. CS1003P4 hello world The output from the terminal was: Error: Incorrect arguments passed. Please pass: The path to the directory that contains the text files, the search term, similiarity threshold
The check for putting in an invalid similarity threshold.	NA	NA	An error will display, and the program will quit.	The expected outcome.	The input: time java -cp "/cs/studres/CS1003/0-General/spark/*":. CS1003P4 hello world 1.3 The output from the terminal was: Error: similarity threshold provided must be between 0 and 1 inclusive.

Evaluation

Overall, I would say my program was extremely successful, it passes the stacscheck decently fast for the sheer amount of data. Though I am sure there are ways to improve the efficiency of the code, but it is perfectly serviceable for what it was asked to perform for this practical

Conclusion

This practical has definitely been a struggle as I personally find the concept of streaming difficult to program with. I understand the high-level concepts but putting it into practise with spark was very difficult which is why I am frankly proud I managed to get it working. I definitely get spark and streaming a lot better now on a functional/programming level. If given more time I would try to make the program run a bit faster, maybe even have a go at running the program across multiple computers so the data is distributed.