

Overview

We were asked to create a Java program to search for the most similar word in a text file to a given search query. Similarity here is the similarity of the sets of bigrams for each word. The proportion of bigrams in both compared to the total number gives a score from 0-1. Higher numbers suggesting higher similarity. This score was to be printed along with the closest match.

I successfully implemented each part of the practical specification, with the more accurate bigram creation “top and tail” method. The program allows the user to input a filename and query string before finding the closest match.

Design

I decided to start by getting the list of the words, so that if they entered an invalid file path/name then I wouldn't need to waste computational time getting the bigrams of the query word.

Using “Files.readAllLines()” gives a List<String> variable which is useful as it can be put through a for loop to get each of the bigrams for each line. The program assumes that each line is a singular word as it doesn't cause the program to break if the words list file is formatted incorrectly. It will just mean the matching of those words will be less accurate.

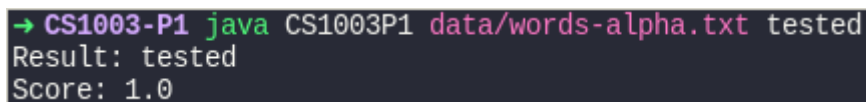
I decided not to keep track of every bigram as I went because I just needed to get the top scoring Jaccard index and word. This means that there is just one output given to the user if multiple words in the list give the same highest Jaccard index score as the top scoring word is only overwritten if the score is greater than the current highest.

Given more time, it would have been interesting to keep track of all the bigram Jaccard index scores in a Map variable which could then be sorted by their value which could give the top suggestions.

As an extension, I added a NGRAM static constant to the program which can be changed to change the n-gram, instead of bigrams. This is interesting and may produce different accuracy, I noticed that when the NGRAM is set to high, the accuracy is a lot lower

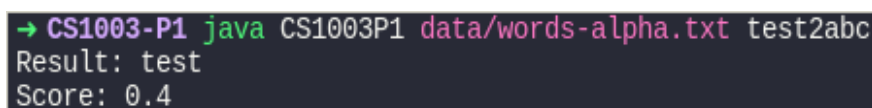
Testing

Test 1 – Working Correctly with words file and query



```
→ CS1003-P1 java CS1003P1 data/words-alpha.txt tested
Result: tested
Score: 1.0
```

Figure 1



```
→ CS1003-P1 java CS1003P1 data/words-alpha.txt test2abc
Result: test
Score: 0.4
```

Figure 2

Figure [1] and [2] show the program working as intended. With the first argument being the relative file path and the second argument as the search query. Figure [1] shows a perfect match to a word in

the words list file with a score of 1 and Figure [2] shows what happens when a query doesn't perfectly match any word in the words list file, giving the closest match.

Test 2 – Stackscheck Success

```
gp87@pc7-148-l:~/Documents/CS1003/CS1003-P1 $ stacscheck /cs/studres/CS1003/Practicals/P1/Tests/
Testing CS1003 CS1003 P1 Text processing
- Looking for submission in a directory called 'CS1003-P1':
- Warning: in a directory called 'CS1003-P1', with subdirectories also with the same name.
- Guessing your submission is in the current directory.
- If that's wrong, go into the correct directory.
* BUILD TEST - build : pass
* COMPARISON TEST - args/0arg/prog-expected.out : pass
* COMPARISON TEST - args/1arg/prog-expected.out : pass
* COMPARISON TEST - args/2args-nofile/prog-expected.out : pass
* COMPARISON TEST - args/3args/prog-expected.out : pass
* COMPARISON TEST - queries/beans/prog-expected.out : pass
* COMPARISON TEST - queries/beanz/prog-expected.out : pass
* COMPARISON TEST - queries/cheese/prog-expected.out : pass
* COMPARISON TEST - queries/sausage/prog-expected.out : pass
* COMPARISON TEST - queries/soosage/prog-expected.out : pass
* COMPARISON TEST - queries/sosage/prog-expected.out : pass
11 out of 11 tests passed
```

Figure 3

Figure [3] shows the successful passing of the stacscheck tests. Showing correct outputs for when incorrect argument numbers are given. This is because error messages are displayed for the user when exactly 2 arguments are not given, the filename and the query string.

This test also shows that the program successfully handles an incorrect filename given, for when the words file doesn't exist. It lets the user know and exits.

Test 3 – Empty File

```
→ CS1003-P1 java CS1003P1 data/empty.txt test
No words in file: data/empty.txt
```

Figure 4

Figure [4] shows how the program deals with a word list file which exists, but is empty and contains no words. This displays an error message to the user and exit.

Test 4 – Extension n-grams

```
→ CS1003-P1 java CS1003P1_Ex data/words-alpha.txt thisistest
Result: moistest
Score: 0.3846154
→ CS1003-P1 java CS1003P1 data/words-alpha.txt thisistest
Result: teethiest
Score: 0.5833333
```

Figure 5

Figure [5] shows the difference between n-grams of length 3, and length 2 (bigrams), showing that changing the size of the n-grams works in the extension program and gives a different Jaccard index score.

Evaluation

My execution of the task was successful in searching for the most similar word in a file using the Jaccard index. I am happy that I have completed all areas of the specification.

Conclusion

Overall, I achieved a successfully accurate word similarity searcher. I managed to use IO to read from a file and use the modern Java.nio packages for more modern implementation of using IO.

I also successfully managed to use the remote tools successfully and efficiently for checking the work with stacscheck from my local machine.

Given more time, potentially also implementing multiple n-grams together would have been an interesting concept as it would involve tweaking the importance of the similarities of the different n-grams used between two words to improve the accuracy of matching as using different sizes n-grams gives different jaccard index scores.

Summarise what you achieved, what you found difficult, and what you would like to do given more time.