- Gaurav Agarwal
- Rana Alshemali

Documentation of Project 1
TF-IDF Computation Using Map-Reduce

CSCI 8790 | Advanced Topics in Data Intensive Computing | Fall 2018 | Dr. Ramaswamy

Hadoop version 3.1.1
Master : vm169
Slave1 : vm170
Slave2 : vm171

Command to run the MapReduce code:

- `Hadoop jar <.jar file> <package.class> <path to vocab.txt file> <input files' directory path> <output directory path>`

(Note: The output folder must not already exist)

Ex.:

```
hadoop jar /tmp/tf-idf/tfidf.jar td.code /user/student/vocab.txt /user/student/input/ /user/student/output/
```

We have created 2 different codes:

1. TF-IDF without normalization.
2. TF-IDF with normalization (i.e. it additionally divides the term_frequency by the total words in a document)

TF-IDF without normalization

The key-value pairs for TF-IDF without normalization are:

- Mapper1 → (word@doc, 1)
- Combiner1 → (word@doc, count)
- Reducer1 → (word@doc, count)

- Mapper2 → (word, doc=count)
- Reducer2 → (word doc, tfidf_score)

Files included:

- Java program: code.java
- Hadoop Jar file: tfidf.jar
- Final output: output/output2/part-r-00000
- Final output lines: 72,198

Logic:

- The words in the vocab.txt file is stored in an ArrayList.
- In Mapper1, every input file content is split on blank space character to get words and a pattern is applied to filter out words having unwanted symbols attached (eg.: " Hi? " will be processed as " Hi ", however an unattached symbol " ? " will also be treated as a separate word to compare.).
- The words of every file are iterated over and if they exist in the ArrayList of vocab.txt then they are mapped out otherwise they are discarded from further processing.

- We introduced a combiner here to reduce the burden on the reducer. It will reduce the words in the dataset local to a mapper.
- The Reducer1 further sums up the count of these words with respect to the documents.
- The Mapper2 just rearranges the output key-value pairs of Reducer1 so as to process it better for Reducer2.
- The Reducer2 iterates over the documents in which a word(key) appears and gets the total document count in which it appears. The total number of input documents is counted dynamically in the code and the tf-idf is calculated. The Reduer2 outputs the final result in the format of "word document tf-idf score".

TF-IDF with normalization

The key-value pairs for TF-IDF with normalization are:

- Mapper1 → (word@doc, 1)
- Reducer1 → (word@doc, count)

- Mapper2 → (doc, word=count)
- Reducer2 → (doc=total_words, word=count)

- Mapper3 → (word, doc=count/total_words)
- Reducer3 → (word doc, tfidf_score)

Files included:

- Java program: code_norm.java
- Hadoop Jar file: tfidf_norm.jar
- Final output: output_norm/output3/part-r-00000
- Final output lines: 73,644

Logic:

- The words in the vocab.txt file is stored in an ArrayList.
- In Mapper1, words of the input files are retrieved using a regex pattern.
- The words of every file are iterated over and if they exist in the ArrayList of vocab.txt then they are mapped out otherwise they are discarded from further processing.
- The Reducer1 sums up the count of these words with respect to the documents.
- Addition Mapper2 and Reducer2 are introduced to get the total count of words per document so as to divide the tf-score by total words in the document.
- The Mapper2 outputs the document as the key and word_count as the value.
- The Reducer2 counts the total words appearing in a document and outputs it to be processed by the next MapReduce.
- The Mapper3 just rearranges the output key-value pairs of Reducer2 so as to process it better for Reducer3.
- The Reducer3 iterates over the documents in which a word(key) appears and gets the total document count in which it appears. The total number of input documents is counted dynamically in the code and the tf-idf is calculated. The Reduer3 outputs the final result in the format of "word document tf-idf score".

- Gaurav Agarwal
- Rana Alshemali

Assumptions made:

- We have tried to match as many words from the vocab.txt as possible, however certain attached symbols, for example in " Hi? ", which is attached to a word without any blank space may get rejected due to splitting over blank space character and patter matching.
- Master has no data-node running, it only has name-node. It will not do any data processing.
- Our code takes in 3 command line arguments:
    - path to vocab.txt file
    - input files' path
    - output folder path