CS 560 Assignment 2

Duwal Shrestha, Mahendra
`mduwalsh@utk.edu`

Nash, Drew
`anash4@utk.edu`

7 April 2015

README

Use the commands identified in this `README` to compile and run the PA2 submission. Please contact Drew and Mahendra if there are any questions. Before any of these commands are run, the Hadoop cluster should be started using the following command while in the Hadoop directory.

```
$ ./bin/start-all.sh
```

The following files are included in `Duwal_Nash_PA2.zip`:

`index.txt` - Generated index with sample input
`LineIndexer.java` - Hadoop Line Indexer program
`preprocess.cpp` - Preprocessor program
`query.cpp` - Query program
`result.txt` - Testing result document, general
`result-ec.txt` - Testing result document, extra credit
`StopWord.java` - Hadoop Stop Word program
`stopwords.txt` - Stop word list generated with sample input
`works.txt` - Sample input: Shakespeare's work, without disclaimers
`works-full.txt` - Sample input: Shakespeare's work, with disclaimers
`works-out.txt` - Preprocessed sample input
`works-full-out.txt` - Preprocessed sample input

1 Preprocessing

The sample text files used were `works.txt`, which contains all works of Shakespeare from the Gutenberg website, and `works-full.txt`, which also contains the licensing disclaimers.

1.1 preprocess.cpp

Use the following commands to compile and run the preprocessor.

```
$ g++ -o preprocess preprocess.cpp
$ ./preprocess
```

```
Enter name of input file for text processing:
works.txt
Enter desired name for output file of processed text:
works-out.txt
$ ./preprocess
Enter name of input file for text processing:
works-full.txt
Enter desired name for output file of processed text:
works-full-out.txt
```

Note that the preprocessor removes all punctuation and converts all letters to upper-case form. Punctuation in the middle of a word is just removed, so "That's" will be converted to "THATS". However, hyphens are converted to spaces, so "Fare-Well" will become "FARE WELL".

Use the following command to put the preprocessed text into the HDFS while in the same directory as `hadoop-core-1.2.1.jar`.

$ ./bin/hadoop fs -put *local_path hdfs_path*

2 Identifying and Removing Stop Words

Use the following command to compile the stop word processor while in the same directory as `hadoop-core-1.2.1.jar` and `StopWord.java`.

$ javac -classpath hadoop-core-1.2.1.jar -d *stopword_class_folder* StopWord.java

Use the following command to create the jar file while in the same directory as `hadoop-core-1.2.1.jar`.

$ jar -cvf stopword.jar -C *stopword_classes_path jar_file_output_path*

Use the following command to run the stop word processor while in the same directory as `hadoop-core-1.2.1.jar`.

$ ./bin/hadoop jar stopword.jar org.myorg.StopWord *input_filepath output_filepath threshold*

The threshold determines which words are added to the stop words list. For example, setting the threshold to 2000 adds every word appearing 2000 or more times in the input files to the stop words list. Words that are on this list are ignored in all input. With a threshold of 2000, `works-out.txt` and `works-full-out.txt` have 117 stop words, which is appropriate, since almost all of these words are prepositions or small nouns. Setting this threshold any lower results in the loss of character names, such as "Caesar" and "Brutus" which is undesirable in this instance. That is why the authors chose a threshold of 2000 for these sample files. Note that the threshold should be adjusted with each input. Also, since all stop words are ignored, queries containing only stop words will return nothing. If this results in a problem, then the threshold should be increased.

3 Building the Inverted Index

Use the following command to compile the line indexer while in the same directory as `hadoop-core-1.2.1.jar` and `LineIndexer.java`.

```
$ javac -classpath hadoop-core-1.2.1.jar -d lineindexer_class_folder
LineIndexer.java
```

Use the following command to create the jar file while in the same directory as `hadoop-core-1.2.1.jar`.

```
$ jar -cvf lineindexer.jar -C lineindexer_classes_path
jar_file_output_path
```

Use the following command to run the line indexer while in the same directory as `hadoop-core-1.2.1.jar`.

```
$ ./bin/hadoop jar lineindexer.jar org.myorg.LineIndexer
input_filepath output_filepath -skip stopword_filepath
```

The line indexer generates a file that is of the following format:

```
word1 file1.txt@1@1,file2.txt@2@1
word2 file2.txt@1@1
```

In this case, `word1` is located in two places: Line 1, Offset 1 of `file1.txt` and Line 2, Offset 1 of `file2.txt`. Furthermore, `word2` is located in one place: Line 1, Offset 1 of `file2.txt`.

4 Query the Inverted Index

Use the following command to extract the map index and list of stop words from the Hadoop cluster while in the same directory as `hadoop-core-1.2.1.jar`.

```
$ ./bin/hadoop fs -get hdfs_path local_path
```

It is a good idea to rename the map index to `index.txt` and list of stop words to `stopwords.txt`. Use the following commands to compile and run the query tool.

```
$ g++ -o query query.cpp
$ ./query
Enter name of the inverted index file generated by MapReduce:
index.txt
Inverted index file loaded successfully.
Enter name of the stopwords file generated by MapReduce:
stopwords.txt
Stopwords file loaded successfully.
Extra credit mode?  Enter ‘y’ or ‘n’:
n
Show query details?  Enter ‘y’ or ‘n’:
n
See README for details.  Send CTRL+D signal to exit.
Enter queries:
```

At this point, users can enter an unlimited number of queries. In non-Extra Credit (non-EC) Mode, one or more words should be entered. Words should be separated by spaces. Lines returned will include all of the words in the query. For example:
`word1 word2 word3` in non-EC Mode is logically equivalent to stating
`word1 and word2 and word3` in Extra Credit Mode.
Not that the ordering of words does not matter.

5 Improve the Query

In Extra Credit (EC) Mode, “and”, “or”, and “not” is supported in queries. Furthermore, ordering of the words does matter. A query for “fare well” will only return lines in which these words are adjacent in that order. Lines returned will fulfill all conditions in the query. Queries must be in Conjunctive Normal Form. Since it has been mathematically proven that any expression can be written in this format, it is an effective grammar. For example
`word1 and word2 or word3 not word4`
is logically equivalent to stating
`(word1) and (word2 or word3) and (not word4).`
Note that “not” literally means ‘and not’. If ‘or not’ is desired, the user must type “or not” together.

Final Notes

Mahendra - `StopWord.java`, `LineIndexer.java`.

Drew - `preprocess.cpp`, `query.cpp`, `README.pdf`.