Code Review for Shabbir Hussain

Ushang Thakker

10/3/2017

Review for A2:

1. Project doesn't compile when typed `make build` as said in the Makefile. Need to explicitly create `target/classes` folder.
2. The `jar` file doesn't build and gives a `FileNotFoundException` since the `artifacts` folder inside `out` is not automatically generated. Need to create this path explicitly.
3. In the second MR task the files are read line by line and the total k-neighborhood score of the words becomes the sum of the k-neighborhood score of the immediate words in that line only. This means that the neighborhood score for the first & last few words is incorrect when k is small. But when k is large > the length of the words in the line the error will increase exponentially. Thus, the k-neighborhood score calculated is not over the entire corpus but over each line read individually resulting in incorrect k-score.

```
/**
 * Maps text input to letter count.
 * @param kin is the input key. (unused)
 * @param datin is the chunk of text to process.
 * @param context is the output context.
 * @throws IOException
 * @throws InterruptedException
 */
@Override
public void map(LongWritable kin, Text datin, Context context) throws IOException, InterruptedExcep
    final String[] words = datin
            .toString()
            .toLowerCase()
            .replaceAll(validCharsRegex, " ")
            .split("\\s+");

    int score = 0;
    int i = 0;
    if(words.length == 0) return;

    String strCurr = words[i];
    // Initialize score for zeroth element.
    for(int j = 1; j < k; j++){
        score += getWordScore(getSafeWord(words, j));
    }
    // Process elements between 0 to length - k
    do{
        context.write(new Text(strCurr), new WordStatsWritable(score));

        // Prepare for next word
        score += getWordScore(strCurr);         // Add current word score.
        score += getWordScore(getSafeWord(words, i + k));    // Add new entering word score.
```

```
            strCurr = getSafeWord(words, ++i);

            score -= getWordScore(strCurr);         // Subtract new current word score.
            score -= getWordScore(getSafeWord(words, i - k));    // Subtract exiting word from the wind
        }while(i < words.length - k);

        // Process words between length - k and length
        do{
            context.write(new Text(strCurr), new WordStatsWritable(score));

            if(i >= words.length - 1) break;

            // Prepare for next word

            strCurr = getSafeWord(words, ++i);

            score -= getWordScore(strCurr);         // Subtract new current word score.
            score -= getWordScore(getSafeWord(words, i - k));    // Subtract exiting word from the wind
        }while(true);
    }
```

4. The `Makefile` didn't complete execution and the R-Markdown report doesn't generate after program execution ends.
5. NOTE: The code is very well written and very easy to understand. The coding standards and conventions used throughout the code were of supreme quality. I really liked the way the code was written.