TOBB University of Economics and Technology
Department of Computer Engineering

BİL401/501 Distributed Data Processing and Analysis
Spring 2016

Homework 1

Due by Feb 2 (Tuesday) 24:00

Subject: Pairs and Stripes design pattern

(Adapted from Jimmy Lin's class at Waterloo)

By now, you should already be familiar with HortonWorks sandbox (e.g., submitting jobs) and using Maven to organize/compile your assignments.

Before staring this assignment, it is *highly recommended* that you look at the implementations of bigram relative frequency and co-occurrence matrix computation in Bespin.

In this assignment you'll be computing pointwise mutual information, which is a function of two events *x* and *y*:

$$\text{PMI}(x, y) = \log \frac{p(x, y)}{p(x)p(y)}$$

The larger the magnitude of PMI for *x* and *y* is, the more information you know about the probability of seeing *y* having just seen *x* (and vice-versa, since PMI is symmetrical). If seeing *x* gives you no information about seeing *y*, then *x* and *y* are independent and the PMI is zero.

Write a program (two separate implementations, actually—more details below) that computes the PMI of words in the http://www.gutenberg.org/cache/epub/100/pg100.txt (The Complete Works of William Shakespeare) collection you used in the previous assignment. Your implementation should be in Java. To be more specific, the event we're after is *x* occurring on a line in the file (the denominator above) or *x* and *y* co-occurring on a line (the numerator above). That is, if a line contains "A B C", then the co-occurring pairs are:

- (A, B)
- (A, C)
- (B, A)
- (B, C)
- (C, A)
- (C, B)

If the line contains "A A B C", the co-occurring pairs are still the same as above; same if the line contains "A B C A B C"; or any combinations of A, B, and C in any order.

A few additional important details:

- To reduce the number of spurious pairs, we are only interested in pairs of words that co-occur in ten or more lines.
- To reduce the computational complexity of the problem, we are only going to consider up to the first 100 words in each line.
- Just so everyone's answer is consistent, please use log base 10.

Use the same definition of "word" as in the word count demo. Just to make sure we're all on the

same page, use this as the starting point of your mapper:

```
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
    String line = ((Text) value).toString();
    StringTokenizer itr = new StringTokenizer(line);

    int cnt = 0;
    Set set = Sets.newHashSet();
    while (itr.hasMoreTokens()) {
      cnt++;
      String w = itr.nextToken().toLowerCase().replaceAll("(^[^a-z]+|[^a-z]+$)", "");
      if (w.length() == 0) continue;
      set.add(w);
      if (cnt >= 100) break;
    }

    String[] words = new String[set.size()];
    words = set.toArray(words);

    // Your code goes here...
  }
```

You will build two versions of the program (put both in the same package):

1. A "pairs" implementation. The implementation must use combiners. Name this implementation PairsPMI.
2. A "stripes" implementation. The implementation must use combiners. Name this implementation StripesPMI.

Since PMI is symmetrical, PMI(x, y) = PMI(y, x). However, it's actually easier in your implementation to compute both values, so don't worry about duplicates. Also, use TextOutputFormat so the results of your program are human readable.

Make sure that the pairs implementation and the stripes implementation give the same answers! Answer the following questions:

Question 1. *Briefly* describe in prose your solution, both the pairs and stripes implementation. For example: how many MapReduce jobs? What are the input records? What are the intermediate key-value pairs? What are the final output records? A paragraph for each implementation is about the expected length.

Question 2. What is the running time of the complete pairs implementation? What is the running time of the complete stripes implementation? (Tell me where you ran these experiments, in the lab computers, or your own laptop, or somewhere else).

Question 3. Now disable all combiners. What is the running time of the complete pairs implementation now? What is the running time of the complete stripes implementation? (Tell me where you ran these experiments, in the lab computers, or your own laptop, or somewhere else).

Question 4. How many distinct PMI pairs did you extract?

Question 5. What's the pair (x, y) (or pairs if there are ties) with the highest PMI? Write a sentence or two to explain why such a high PMI.

Question 6. What are the three words that have the highest PMI with "tears" and "death"? And what are the PMI values?

Note that you can compute the answer to questions 4—6 however you wish: a helper Java program, a Python script, command-line one-liner, etc.

Upload and submit your assignment via your git repository account. Name the repo **tobb-etu-bigdata-course**. Put all your assignment submissions under this repo (asg1, asg2, …, asgN). In the assignment submission subdirectory (tobb-etu-bigdata-course/asg2) have a readme file explaining programs, input/output files, and also your answers to the above questions 1-6.

Upload date/time should be before the submission date/time to get full credit. 10 points are deducted for each late day, and your submission is not accepted after 2 late-days unless permitted by the instructor for legitimate excuses.

Share your git account with the instructor (edogdu@github, erdogandogdu@bitbucket).