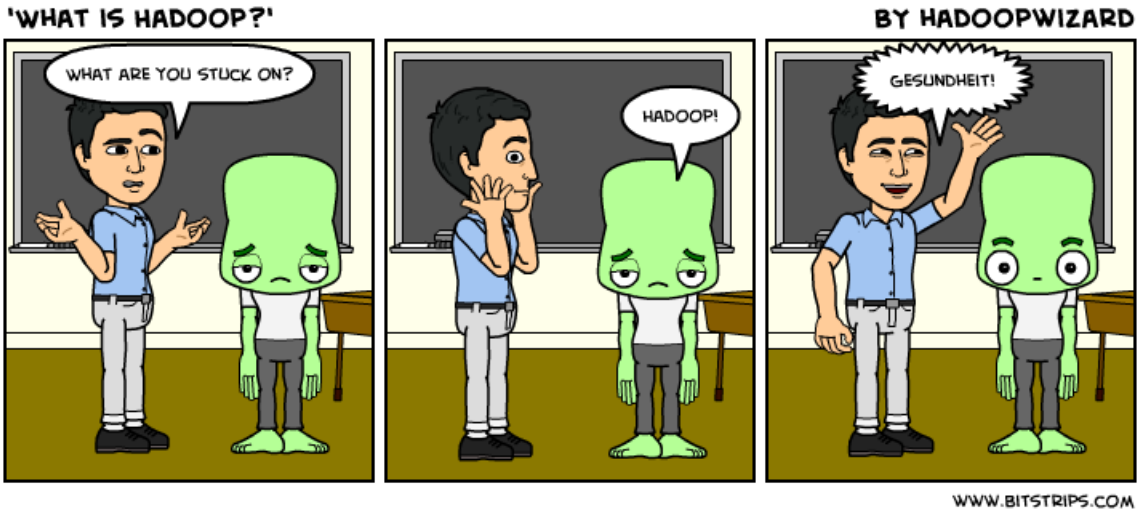
CS158 - Assignment 9

Faster Naive Bayes? Say it ain't so...

Due: Sunday, November 12 by 11:59pm



http://www.hadoopwizard.com/what-is-hadoop-a-light-hearted-view/

For this assignment we're going to dive into how we can implement some of the machine learning algorithms and tools in MapReduce to make them parallelizable.

MapReduce on paper

Viewing problems within the MapReduce framework can take a bit of practice. In addition, because these jobs run on a cluster, it can be very challenging to debug. Therefore, it is even more critical[1] to think through your MapReduce problems before you start coding them.

To give you practice with this, for this assignment we will write pseudocode to solve a few problems within the MapReduce framework.

A pseudocode description of a MapReduce problem should include the following for both the `map` function AND the `reduce` function:

- The types for the input key/value pairs AND the output key/value pairs (that's four types). You should use the built-in hadoop types, specifically one of `Text, IntWritable, LongWritable, DoubleWritable and BooleanWritable`

[1] I know you all know that it's critical for normal programming too!

- Pseudocode describing what needs to happen in the function. These often will just be one or two statements. I don't want you to write actual code, but you should provide enough detail that someone reading it can understand exactly what the function should do and could implement it based on your pseudocode.

For example, for our word counting example, the following would be a reasonable submission:

```
- map
Input: key = LongWritable, value = Text
Output: key = Text, value = IntWritable

* split up the value text into words
* for each word, add an output key/value pair of <word, 1>

- reduce
Input: key = Text, value = IntWritable (or Iterator<IntWritable>, either way of
   writing this is fine for me)
Output: key = Text, value = IntWritable

* calculate the sum for each of the values associated with the key
* add an output key/value pair of <key, sum>, that is the word and how many times it
   occurred
```

Write MapReduce pseudcode for the following problems. You will be graded on how well you follow the guidelines, how appropriate your solution is to the MapReduce framework and the quality of your descriptions. Note, for some of these problems you may have to write more than one pair of map/reduce functions.

1. Anagrams

   Given a file containing text, the program should output key/value pairs where the value is a comma separated list of lines in the file that are anagrams, i.e. use exactly the same letters (ignoring spaces). The output key is up to you and will depend on your implementation. For example:

   Input:

   ```
   captain over rome
   lives
   cat
   banana
   emperor octavian
   act
   tac
   elvis
   ```

Output:

```
<some_key>    cat, act, tac
<some_key>    elvis, lives
<some_key>    emperor octavian, captain over rome
<some_key>    banana
```

2. K-NN (almost)

   Given a CSV file containing a list of training examples (like we've been using all along in this class) the program should calculate the distance from each example in the file *from* a particular test example (which we'll assume is hard-coded into the code). The output from the program should be pairs of label/distance where label is the label of the training example and distance is the distance from that example to the test example. These examples should be output *in sorted order* by distance. You may assume:

   - the test example is available as a variable (lets call it `test`) in both the map and reduce functions. We'll see in class how we can do this.
   - access to a function `readExample` that takes an example line and gives back an `Example`.
   - access to a function `getDistance` that gives the distance between two examples.

3. Naive Bayes (almost)

   Given a file containing labeled text examples where the the line consists of the label followed by the example text associated (e.g. like the wine data) output the number of times a given feature (i.e. word) occurs in an example with a particular label, that is, the *numerator* for all features and labels in:

   $$p(x_i|y) = \frac{count(x_i, y)}{count(y)}$$

   Notice, that we are counting the number of examples with a particular label that contain a particular feature (i.e. word) and NOT the number of times that a word occurs in text of a particular label, that is, if a word occurs multiple times in an example, it still only gets counted once.

   You may choose any reasonable output key (be specific) that would allow someone reading the file to be able to figure out both the feature (word) and label. The value should be the number of times that feature/label combination occurred.

4. Naive Bayes (even closer)

   Given a file containing labeled text examples where the the line consists of the label followed by the example text associated (e.g. like the wine data) output the number of times each label occurs in the dataset, that is, the *denominator* in:

   $$p(x_i|y) = \frac{count(x_i, y)}{count(y)}$$

   Note that with this MapReduce program and the previous, we would have all of the data that we'd need to actually calculate $p(x_i|y)$!

5. Feature normalization

   Given a file containing a list of examples of the form:

   ```
   <label> <feature1> <feature2> <feature3> ... <featurem>
   ```

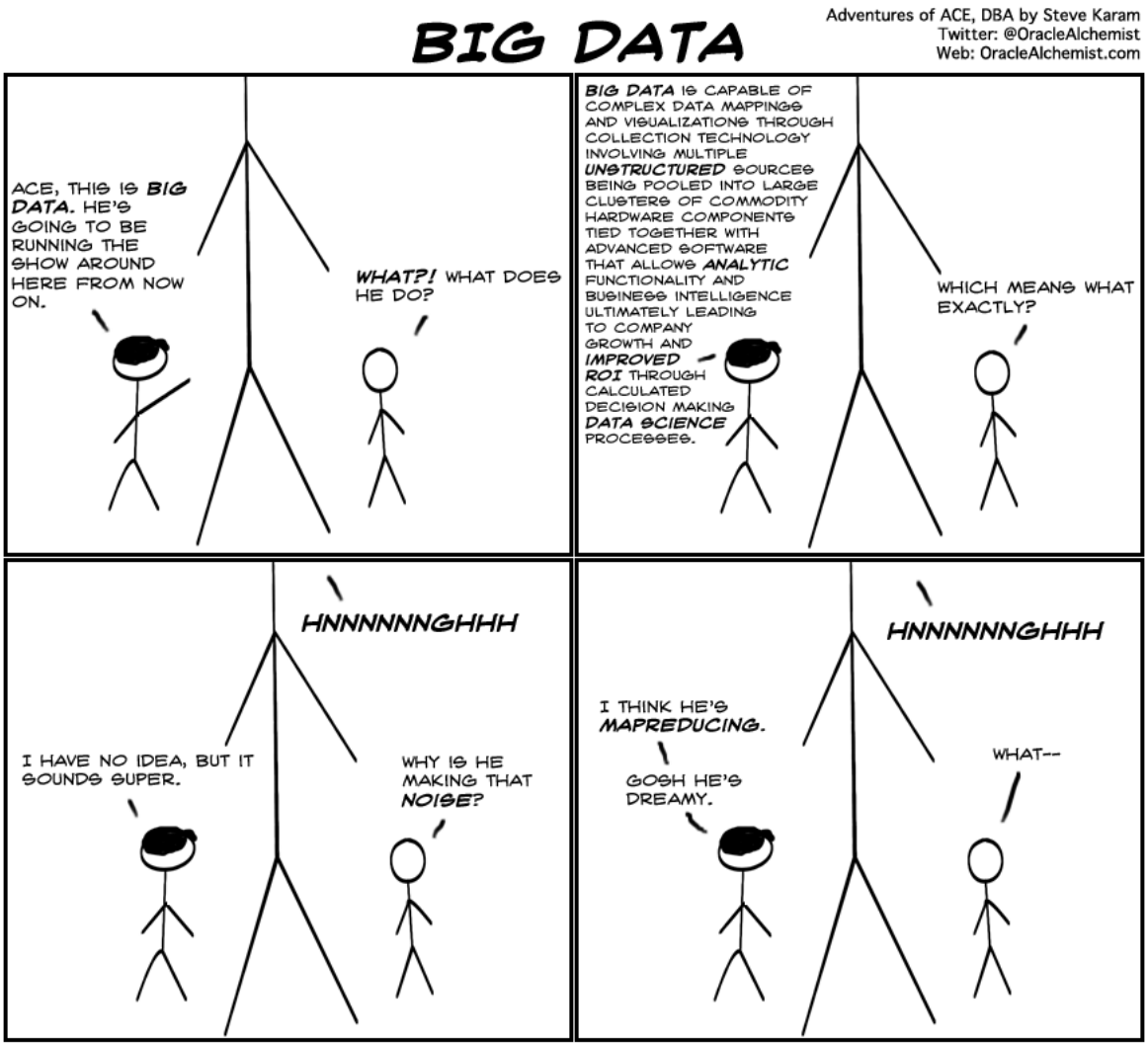
   we want to generate a version of this file where the feature values have been *mean centered*. The output examples should include the label and should also appear in the same order as the original file.

   You may assume:

   - that each example has all of the features defined
   - if (*hint!*) you use multiple map/reduce phases, you may assume that future phases have access (as, say, instance variables) to data produced in previous stages

When You're Done

Submit your answers to the above questions in a single file to Gradescope.



http://knowledgehubnetworks.com/information-technology/big-data-ace-comic/