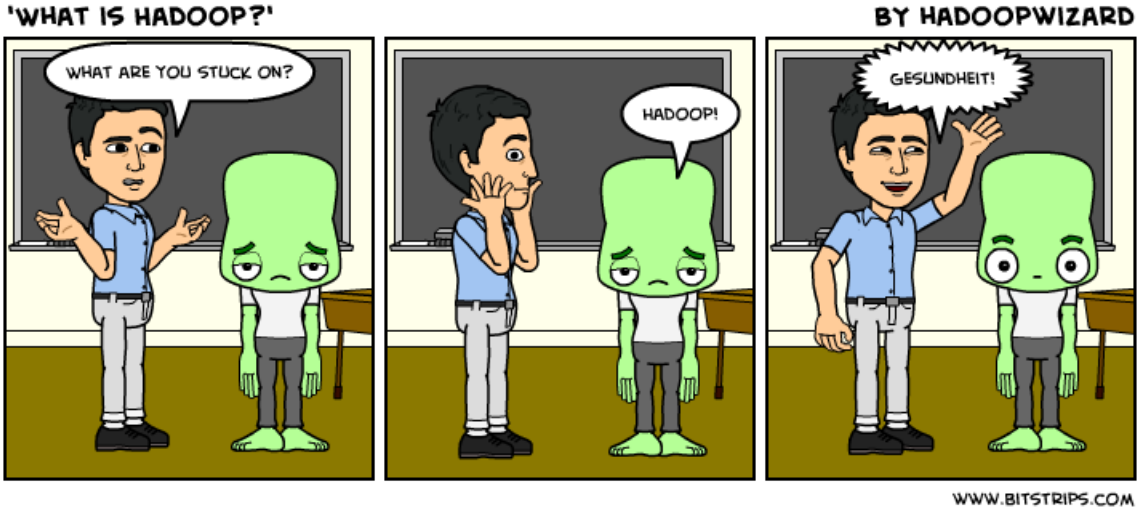
CS451 - Assignment 8

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

Part 1 due: Friday, Nov. 8 before class
Part 2 due: Monday, Nov. 11 before class
Part 3 due: Sunday, Nov. 17 by 11:50pm



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

For this assignment we're going to revisit our Naive Bayes classifier and implement a parallelized version using MapReduce for Hadoop.

1 Part 1: Setup

To get you ready for the assignment and for lab on Friday, we need to do a few configuration things. Login to `basin` and do the following (note, if you're working with a partner, both people should do this from their own account):

1. Edit your the file `~/.bashrc` (which is called whenever you login) and add the following lines:

```
# setup the hadoop environment variables
/usr/local/hadoop/conf/hadoop-env.sh

alias hadoop='/usr/local/hadoop/bin/hadoop'
```

The first line calls a script that sets up some basic environment variables for us and the second line creates an alias so that you can call hadoop by just writing `hadoop`.

After you do this, logout and then back in to `basin`. If all worked well, you should be able to type `hadoop` on the command-line and you should see:

```
Usage: hadoop [--config confdir] COMMAND
(some more stuff here)
```

2. Setup passwordless SSH on the lab network

   Hadoop communicates between machines over SSH. For this to happen without you having to type your password every time two computers want to communicate, we need to setup passwordless SSH.

   Please follow the directions below EXACTLY:

   (a) Type: `ssh-keygen -t rsa`
   (b) hit `enter` to use the default path location
   (c) hit `enter` to use an empty passphrase
   (d) hit `enter` again to confirm an empty passphrase
   You should then see a bunch of information printed to the screen including:

   ```
   You'll see a bunch of stuff, including:
   Your identification has been saved in ...
   Your public key has been saved in ...
   The key fingerprint is:
   ...
   The key's randomart image is:
   +--[ RSA 2048]----+
   (some pretty picture here)
   ```

   (e) Type: `cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys`
   (f) Type: `echo "IdentityFile ~/.ssh/id_rsa" > ~/.ssh/config`
   (g) Type: `chmod 700 ~/.ssh`
   (h) Type: `chmod 600 ~/.ssh/*`
   (i) Type: `chmod 640 ~/.ssh/authorized_keys`

   To make sure it worked type the following:

   `ssh abe.cs.middlebury.edu`

   You may get asked if you want to add this host (yes/no question), however, you should not be asked for your password and should immediately login to abe.

3. Add all of the lab machines as known hosts

   For each IP address from `140.233.20.150` to `140.233.20.175` (that's 26 of them!) do the following:

   ssh to that machine, e.g. `ssh 140.233.20.150`

   When you do this, one of four things will happen:

- If the computer isn't on (or it's having problems), you'll get:

  ssh: connect to host 140.233.20.xxx port 22: No route to host

  or it will hang and you'll have to CTRL+c the ssh.

  If this is the case, you'll just have to skip this one.

- If you've already sshed to this machine previously and added to the key, you'll get something like: Last login: .... prompt $

  If that's the case, you're done with this computer and just type `exit`

- If you haven't sshed to this computer before you'll get the prompt: he authenticity of host '140.233.20.xxx' can't be established. RSA key fingerprint is ... Are you sure you want to continue connecting (yes/no)?

  Type "yes" and you'll see:

  Warning: Permanently added '140.233.20.xxx' (RSA) to the list of known hosts. Last login: ... prompt $

  You've added this computer to the list of known hosts, so now you can exit it by typing `exit`

- You might also get the following:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
...
```

  If this is the case, you need to delete this key from your known hosts file. This happens if you logged into this computer a while ago, but the computer, has been replaced or upgraded.

  To delete the key from your known hosts file:

  * open up the file:
    `~/.ssh/known_hosts`
  * find the line that has the computer with the IP address you're having trouble with (using find/search is the easiest way).
  * delete this line in the file, save the file and exit
    Once you've done this, try and ssh again to this computer. It should know allow you to add it.

Once you're done (sorry, I know 26 is a lot), you should not be able to ssh to any of the lab machines just by type `ssh ...` and it will immediately log you into that machine.

4. Setting up Eclipse

   We'll be using Eclipse (if you don't want to use Eclipse some talk to me) for this assignment, so I want to make sure everyone is setup to write MapReduce programs.

   If you're working on your computer, copy the following file somewhere onto your own computer:

   `/usr/local/hadoop/hadoop-core-1.2.1.jar`

- Open up Eclipse and make a new java project. Call it `hadoop-lab`.
- Right click on the project and select `Build Path -> Configure Build Path...`.
- Under the `Libraries` tab select the `Add External Jars...` button.
- Browse and select the `hadoop-core-1.2.1.jar`. If you're on the lab machines just browse to the location above.
- To make sure it worked, create a new class called `WordCount` and then copy the code from the `WordCount.java` demo we looked at in class on Wednesday (code available on the course web page). If you can get it to compile without any errors, you're all set!

5. A few quick sanity checks

   Once you've done this, you should be all ready to run hadoop. Login to `basin` and then answer questions below. *Put them in a .txt file and then submit the file online (no need to zip it) using the submission script. Enter 8.1 as the assignment number.*

   Question 1: How many files are there in `/user/dkauchak/part1/`?

   Question 2: What is the third line in `/user/dkauchak/part1/other.txt`?

   Question 3: Copy a file called `username.txt` (where username = your actual username) into your `user` directory. Inside the file should be your favorite animal. How many bytes is this file?

   Question 4: The code below (assuming you've set everything up correctly) should approximate pi iteratively:

   ```
   hadoop jar /usr/local/hadoop/hadoop-examples-1.2.1.jar pi 5 10000
   ```

   Run this. For a variety of reasons, you may actually see an exception or two go by, however, it should finish. What is the value of pi that it finds?

2 Part 2: 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 the second part of this assignment you are going to 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 types should be 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. Put your answers in a file (pick some reasonable file format) and submit it online as assignment 8.2. You will be graded on how well you follow the guidelines, how appropriate your solution is to the MapReduce framework and the quality of you 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 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). 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 next week how we can do this.
   - access to 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) we want to 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:

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

   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. 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

3 Part 3: Naive Bayes (again?)

For the last part of this assignment we're going to be implementing the Naive Bayes training algorithm within the MapReduce framework.

3.1 High-level requirements

Write a MapReduce program called `NBTrain` that takes two command-line arguments, an input directory containing training data and an output directory. Within the input directory, the training file(s) will consist of labeled, multi-class, examples of the form:

```
<label><tab><text>
```

where `<label>` is the class label, `<tab>` is a tab and `<text>` is the text representing the example. The wine training data is an example of such a file.

When the program is run, within the output directory it should generate two output directories: `priors` and `counts`. `priors` should contain one or more files that contain counts for how many times each label occurred in the training data in the form:

```
<label><tab><count>
```

The `counts` directory should contain one or more files that contain the feature,label counts, that is the number of times each feature occurred in examples with each label in the form:

```
<label><tab><word><tab><count>
```

Additional requirements:

- If possible, all of your MapReduce programs must properly utilized a combiner class.
- To break up an example text into words, use the `parseFeatureLine` function in the `StringParser` class (see the description of the starter code below).
- All of your classes for this assignment should go inside the `ml.hadoop` package.
- The program should accept the input/output directory as command-line parameters. If the user does not enter the right number of parameters, you should display the usage.
- Try and be as memory and run-time efficient as possible. This means avoiding creating extra objects wherever possible and using static instance variables where appropriate (like we've done in class).

3.2 An Example Run

To make sure the specification is clear, here is an example running of the program.

Let's say you have a directory called `input` that has a single file in it with the following contents:

```
1    a b a c
0    b b
1    c d a
```

To run your program you all hadoop as follows:

```
$ hadoop jar assign8.jar ml.hadoop.NBTrain input output
```

After the program runs you should see and `output` directory. *Within* that output directory should be two directories, priors and counts. If you look inside the priors directory, you will find a file `part-00000` which has the following:

```
0    1
1    2
```

If you look inside the counts directory, there will also be a file called `part-0000` which has the following:

```
0    b    1
1    a    2
1    b    1
1    c    2
1    d    1
```

Notice, in particular, 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.

3.3 Starter Code

At:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs451/assignments/assign8/assign8-starter.tar.gz`

I have included starter code for this assignment:

- `StringParser`: a class containing a static function to be used to split up the text in the examples.

- `NBClassifier`: a version of the Naive Bayes classifier that will take data as output from your MapReduce run and perform classification. This classifier should NOT be run on the hadoop cluster, but instead it should be run locally.

3.4 One path to implementation

There are many ways to implement this problem. Below I outline one possible approach. No matter what approach you take, however, make sure that work incrementally and test each step (map, reduce, etc.) as you go.

1. Think about how to solve the problem on paper. Specifically, think about how many MapReduce programs you will need and what their input and outputs will be.
2. Implement one or more MapReduce programs to generate the prior data. As always, break it up into writing the map, then the reduce step, debugging each one individually.
3. Implement one ore more MapReduce program to generate the counts data.
4. Write a final driver class that runs has the proper user interaction via the command-line and setups up and calls all of the MapReduce programs in the right sequence. If you've done it right, this should just be a few lines of code.

3.5 Hints/Advice

- As always, create some very simple data sets to test your system on where you know what the output should be. The example above is a good place to start, but I'd also recommend using a slightly more involved example (say with more than just two classes) and real words to make sure you're implementing everything correctly.
- Most of the times when you get errors, the feedback is pretty useful. I realize it prints out a lot, but take a look at what the main exception is and often this will point you in the right direction.
- Incremental development is critical here!
- Here are a couple of common errors you might see that aren't as obvious:
  - If you declare your mapper and reducer classes as nested classes, they *must* be declared as `static` classes. If you don't, you will get some error like:

    ```
    java.lang.RuntimeException: java.lang.NoSuchMethodException: ...Reducer.<init>()
    ```

  - Be careful to make sure that your driver setup matches your map/reduce input properly. For example, if your mapper outputs Text values, but in your driver you say it outputs IntWritable values, you might get something like:

```
java.io.IOException: Type mismatch in key from map: expected
```

- If you get an exception, but your program still finishes fine, check the output directory. Due to the somewhat fragile nature of our hadoop cluster, you can occasionally get exceptions that are not your fault and that the cluster recovers from.
- If you're stuck on a particular exception, you *may* search the web to figure out how to solve the exception. Besides that, you should not be trying to find code to solve these problems beyond the documentation and class examples.
- I am asking you to try and be efficient about run-time and variable usage for these programs. Feel free to implement it however you like as a first pass. Once you have it working, go back and look at the code and see if you can come up with ways of reducing the number of variables used, making more things static, etc.

3.6 Experimentation

Once you have your data working, run an experiment! I have provided you with a `NBClassier` class that will use the output from our system (with some copying from the HDFS) to train a model. Run one or two interesting experiments and provide a brief writeup along with your code.

Right now, the only data set that we have available that we can run is the wine data set. I do have a few other data sets available, however, the cluster has been a bit finicky and I don't want to overburden it by running larger jobs. I'm going to see if I can fix some of these robustness issues early next week. If I do, I will publish at least one other text data set for processing.

3.7 Extra Credit

3.8 When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.).

Create a directory with your last name, followed by the assignment number, for example, for this assignment mine would be `kauchak8`. If you worked with a partner, put both last names.

Inside this directory, create a `code` directory and copy all of your code into this directory, maintaining the package structure.

Finally, also include your `experiment` file.

`tar` then `gzip` this folder and submit that file on the submission page on the course web page.

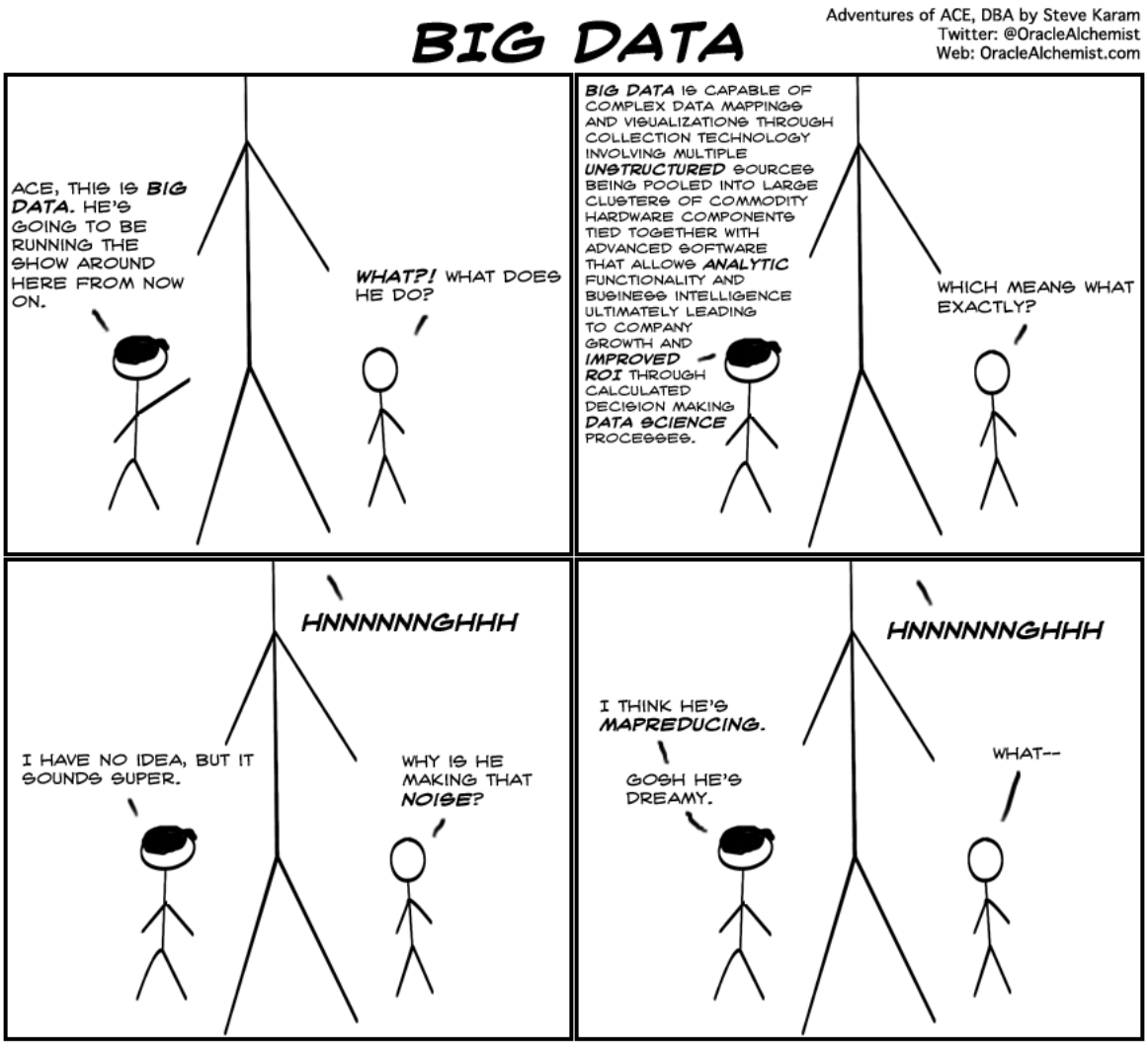
Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate JavaDoc
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

Submit this file online as assignment 8.3.



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