CSE454 – Autumn, 2007

Project Part 1 – to be done ***individually.*** Due on ***10/16 at 12:00pm***

For this first part of the project we would like you to get a feel for a couple of tools that can help with the real thing. We're going to start with a simple classification project and then move the classifier over to Hadoop, a distributed file system and implementation of Google's MapReduce, to run some experiments. Things to turn in are **in bold**. You may e-mail or hand in a paper copy of your results, but please also e-mail the code your produce to eadar@cs.

Note that mostly everything you need to "download and install" is already available in /projects/instr/07au/cse454.

Part 1 – Mallet

For the first part we will just be using Mallet (http://mallet.cs.umass.edu/index.php/Main_Page), a Java system with lots of machine learning algorithms.

1. Download and install Mallet somewhere (http://mallet.cs.umass.edu/mallet/dist/mallet-0.4.tar.gz) . You have access to the unix servers "kiska" and "umnak" for this class and are welcome to work there (they will have things like java and ant installed). You will also find Mallet in "/projects/instr/07au/cse454"
2. Take a look at the tutorials (http://mallet.cs.umass.edu/index.php/Tutorials) and especially the document classification one (http://mallet.cs.umass.edu/index.php/Command_line_tutorial) since that will be the first task. Make sure you are able to run the classification tutorial and get classified results (using the 20_newsgroups dataset: /projects/instr/07au/cse454/20_newsgroups.tar.gz).
3. Text documents in Mallet are converted to *Instances* through a *pipeline*. A pipeline can, for example, throw out stopwords (e.g. the, of, a, etc.), change everything to lowercase, and break apart long pieces of text into words. An instance is the document after it has been passed through the pipeline. Many instances are held in an *InstanceList* which can then be used to train a classifier. Your first task is to construct a pipeline for the input text basically replicating "text2vectors" (which you can use as a reference). A stub for this program (VectorBuilder) is attached at end of this document. **Please turn in a documented version of this class. In particular, please document each step in the pipeline and what it is doing.**
4. Once you have the InstanceList, split it up into a training and test set (try something like 60% and 40%). Try building a classifier and see how well it works. You can start with the NaiveBayesTrainer as below:

```
NaiveBayesTrainer nbt = new NaiveBayesTrainer();
InstanceList train = …
InstanceList test = …

Classifier classifier = nbt.train(train);

Trial trainTrial = new Trial(classifier, test);
System.out.println("accuracy: " + trainTrial.accuracy());
// ConfusionMatrix cm = new ConfusionMatrix(trainTrial).
```

5. Grab a copy of the 4 category wiki dataset from /projects/instr/07au/cse454/ wiki-project0.tgz. The four categories are Celebrities, Cities, Counties, and Countries. Each directory has a number of different files.

Use these in the same way as the 20_newgroup examples and try a few different classifiers and pipeline configurations to get a feel for how they perform and how long they take. You can use a smaller subset of the data to speed up this cycle. Play around with the ConfusionMatrix . **Turn in a short description of what you tried and how well it did. Which categories are most confused for each other?**

6. Try outputting a serialized Classifier object to disk (i.e. through Java's ObjectOutputStream). You will likely want this for the next part.

Hadoop – Making things scale

We will cover Hadoop in more detail in class, but briefly: Hadoop is a system that implements the Google File System (Hadoop's version is called HDFS) and Google's MapReduce. Files are replicated by HDFS partially for availability and in part to speed up the MapReduce operations. MapReduce is a framework for partitioning jobs across many machines by "mapping" each instance to a different place to do some intermediate step and "reducing" to merge the intermediate steps into an output. In our case, we can "map" each document we want to classify to a different task machine and trivially reduce to bring everything together. If you want to know more, take a look at: http://www.cs.washington.edu/education/courses/cse490h/CurrentQtr/lecture/lec2.ppt

7. Grab a copy of Hadoop (please use version 13.1 for now since that is what is on the clusters) from: http://www.apache.org/dyn/closer.cgi/lucene/hadoop/. Also available in /projects/instr/07au/cse454.
8. Take a look at: http://wiki.apache.org/lucene-hadoop/QuickStart , make sure you can run the "Standalone Operation" stuff. Also make sure you understand how to get files onto the Hadoop filesystem. Take a look at the example code and see how jobs are configured and executed. If you like, you can use the eclipse plugin (as described in the cluster information document) to do your work. You can also look at: http://www.cs.washington.edu/education/courses/cse490h/CurrentQtr/labs/lab0.pdf to see some examples in action.
9. Implement a MapReduce class that can classify individual documents/instances (implementing the Mapper interface). You can load the serialized classifier that you trained in the first part of the project. Your system should output a document ID (say the filename), and a classification (one of Cities, Counties, Countries, Celebrities). There are a number of different ways you can do this. **Whatever you choose, please turn in working code *with documentation*. You should provide us with a recipe that we can use to replicate everything from setting up the training and test collections to running on Hadoop. We should be able to build a classifier and classify any arbitrary piece of text through Hadoop using your implementation. Explain why you chose to architect your system in the way you did.** Some possibilities:
    a. You could keep each document in a separate file in an "input" directory and map based on the file name. This will mean that you have to run the text through the pipeline as you did in the first part.
    b. You could keep around all the Instance objects (in a serialized form), one per file, and map those as above.
    c. You could construct your own file format and your own RecordReader
    d. You could use a SequenceFile (http://wiki.apache.org/lucene-hadoop/SequenceFile)
10. Implement a "reduce" operation that calculates the accuracy of the classifier. It should take the output from the map task above (ID + classification) and output a number between 0 and 1. **Turn in documented code for this as above.**
11. Copy over the jar files and anything else you need to the Hadoop cluster (information provided in a separate message) and make sure your system works.

Appendix

```
public class VectorBuilder {
  public static void main(String[] args) throws Exception {
    // Remove common prefix from all the input class directories
    int commonPrefixIndex = Strings.commonPrefixIndex(args);

    // read in the directories, toss out the first common bit of the filename
    // these are our labels
    System.out.println ("Labels = ");
    File[] directories = new File[args.length]; // each class in a directory
    for (int i = 0; i < args.length; i++) {
      directories[i] = new File (args[i]);
      if (commonPrefixIndex < args.length)
        System.out.println ("   "+args[i].substring(commonPrefixIndex));
      else
        System.out.println ("   "+args[i]);
    }
    // construct the pipeline
    Pipe instancePipe;

    instancePipe = new SerialPipes (new Pipe[] {
                //  FILL THIS IN
    });

    // read it all in
    InstanceList ilist = new InstanceList(instancePipe);
    boolean removeCommonPrefix=true;
    ilist.add (new FileIterator(directories,
                                FileIterator.STARTING_DIRECTORIES,
                                removeCommonPrefix));
    }
}
```