

User and Developer Documentation

February 1, 2009

1 Setting Up The Eclipse Project

The distribution provided contains the java code and some additional libraries that will be required for compiling the project. In order to build the code the following are required:

1. *code*: Folder with all the java code.
2. *common-logging.jar*: A logging library needed by Map Reduce but not part of the hadoop distribution on the web.
3. *map-reduce*: This is not part of the distribution but can be downloaded from <http://hadoop.apache.org/core/releases.html#Download>. From the link, download version 0.18.2. <http://www.infosci.cornell.edu/hadoop/> has some instructions on how to set up hadoop to run map reduce programs locally.

To setup the eclipse project, create a new Java project in eclipse with all the default settings. Please make sure that the project is using JVM1.5. This is a requirement for the code to work with Hadoop 0.18.2. In case a different version of Hadoop is used, please use the recommended JVM. Once the project has been setup, do the following:

1. Import the java files in *code* into the *src* folder of your project. To do this right click on the project and click Import. Then select From Filesystem under the general tab. Import the contents of *code* into the *src* folder of the project.
2. Next we add in references to the external libraries. To do this right click on the project and select Build Path → Add External Archives. Select the file *hadoop-*.core.jar* from your hadoop installation folder. Repeat the above process to add in the reference to *common-logging.jar* that is part of the distribution.

Eclipse should autocompile the code and generate the relevant classes in the *bin* folder. Finally we will put all the required into a single jar which will be used for execution. To generate the jar file, right click on the project folder and select Export. Under the Java tab select JAR file. Follow the instructions to create a JAR file. In the rest of this document we will assume that this jar file is called *plotgenerator.jar*

2 Generating Plots On A Single CPU

Before discussing how to use the code to generate plots, we briefly discuss the various inputs that the code expects. Tree models for the code are generated using the IND package. Therefore, many file formats are picked up from IND.

1. *attrfile*: File containing information about the different attributes, names and types represented as categorical or continuous. The file format in this case is the same as that used by IND.
2. *dfile*: This file contains the data records that will provide values for nonsummary attributes. It is usually a sample of the training data set. Each line in the file contains a single data record with the values of the different attributes separated by blanks. If any attribute of a record does not contain a value, it is represented using a ?. Attributes in a data record should be ordered in the same way that they are *attrfile*.

3. *maxdpnts*: This number should be greater than the number of records in the *dfile*.
4. *vpnts*: This file specifies the set of visualization points for the different attributes at which summary values are computed. Each line of this has the following format

attribute_name : value₁, ..., value_n

For an ordered attribute $n = 3$, with $value_1$ = minimum value, $value_2$ =maximum value of the attribute and $value_3$ = the number of visualization points along the dimension of the attribute. For a categorical attribute the values are simply an enumeration of the different visualization points.

Plots of higher dimensions are generated by taking the cross product between the visualization points along each dimension.

5. *plots*: This is a file with a set of plots that should be generated. Each line as the following format

plotid : attr_name₁, ..., attr_name_d

Note that plotid is a simple integer and the code assumes that for a set of plots the plotids are numbered in ascending order and continuous.

6. *numtrees*: The total number of trees in the model.
7. *treefile*: A concatenated version of the all the trees in the model. In order to generate this input we have an additional utility that is described in Section 2.1
8. *outputfile*: The output for writing all the summary output. Each line as the following format

plotid, visualizationpoint, summaryoutput

Note that it seems wasteful to have plotid written out repeated but this is done to make the output formats consistent for map reduce and the one CPU case.

9. *method*: It is a flag that should be used to run different versions of summary generation algorithms. By default always set this value to 2.

2.1 Generate Tree File

In order to generate the *treefile* for the summary generation code, we have an util ConcatINDTreesUtil which given a set of IND trees and tree weights generates a concatenated tree file that is input to summary generation. The command line for that is

```
java -classpath newdpdp.jar:../hadoop-0.18.2/hadoop-0.18.2-core.jar ConcatINDTreesUtil basetreefile=../paw/paw.tree
startid=0 numtrees=2 treeweights=../paw/paw.weights outputfile=../paw/paw.concat
```

The above utility concatenates *numtrees* number of single IND treefiles stored as individual treefiles named as *basetreefile+startid* into a single file *outputfile*. *treeweights* is a file with the weight of each tree in the model. Each line in this file has the following format *treeid : weight*. Note that for treeids we simply use the integer value that is present at the end of a treefile. For example a tree model with 3 trees will have three tree files tree0,tree1,tree2 and treeids 0,1,2.

2.2 Executing The Code

The command line for generating the summaries for the example data is given below

```
java -classpath plotgenerator.jar:../hadoop-0.18.2/hadoop-0.18.2-core.jar MainGenerator attrfile=../paw/paw.attr
dfile=../paw/paw.train maxdpnts=400 vpnts=../paw/paw.v plots=../paw/paw.plot numtrees=2 treefile=../paw/paw.concat
outputfie=../paw/paw.opsplit method=1
```

Make the commandline point to the your mapreduce installation folder to include the hadoop jar in the class path. Second the different command line arguments can be specified in any order but the format parameter=value must be followed and all parameters must be specified.

3 Generating Plots Using Map Reduce

The class *MRMainGenerator* which is also part of *plotgenerator.jar* is the main routine for running map reduce jobs. In addition to the inputs specified in Section 2, mapreduce jobs have the following additional commandline parameters

1. *configtype*: When running jobs locally (primarily for debugging) set this value to 0, else set it to 1 when jobs are run on a hadoop cluster.
2. *numreducers*: The number of reduce tasks in the job. Set this proportional to the number of plots being computed in the job.
3. *splitconfig*: This parameter determines how the input is partitioned.
 - *splitconfig=0*: This means that set of records used to derive values of non summary attributes is partitioned across a set of mappers. In order to set this up, split the entire file of data records into smaller files and write a special line "E-OF" (no quotes) at the end of each smaller file. Put all these smaller files into a directory and make the parameter *dfile* point to this directory. The number of map tasks will be equal to number of smaller files that you generate. Writing the special "E-OF" line is important because mappers perform computation after reading in all the input.
 - *splitconfig=1*: This means that the set of plots are partitioned across a set of machines. In order to set this up like the previous case split the set of plots into smaller files which each file containing "E-OF" at the end. Each smaller file should contain a list of plots arranged in continuous ascending order from some starting plotid. Make *plotfile* point to the directory containing all the smaller plot files. The number of maptasks will be equal to the number of smaller plot files that you have.
 - *splitconfig=2*: In this case each mapper processes a subset of trees in the model. Setting up this scenario is a little different from the above case. For ease of explanation lets illustrate setting up this scenario with an example. Suppose we have a model with 4 trees and we want to distribute computation such that each machine makes computations on two trees. Follow the following steps.
 - Create a file with the tree weights for all the 4 trees.
 - Generate the input treefiles: The first tree file will be generated by calling *ConcatINDTreesUtil* with *startid* as 0 and *numtrees* as 2. The second tree file will be generated by calling *ConcatINDTreesUtil* with *startid* as 2 and *numtrees* as 2. Call the two treefiles *mergedtrees-0* and *mergedtrees-1*.
 - In the map-reduce commandline pass some additional parameters as follows *numtreefiles=2*, and *treefile=mergedtrees-*, In addition set up a new directory with two files: The first containing the lines 0 and "E-OF" and the second 1 and "E-OF" and make *mrtreefile* point to this directory in the command line. When a mapper reads one of the files in the directory and sees a 0, it will pick up the first mergedtree file and process it. Similarly for the other mappers.

An example commandline for running mapreduce jobs locally is given below

```
./bin/hadoop jar ../newpdep/bin/newpdep.jar MRMainGenerator configtype=0 splitconfig=0 dfile=../newpdep/pfw/dfile.vpoints=../newpdep/pfw/pfw.v plots=../newpdep/pfw/pfw.plot numtrees=2 numtreefiles=1 treefile=../newpdep/pfw/pfw.co  
outputfile=../newpdep/pfw/opshckt/ method=1 attrfile=../newpdep/pfw/pfw.attr maxdpoints=400 numre-  
ducers=1
```

4 Code Structure

This section provides a brief description of the java code that will help someone modifying the code to understand where to look for the relevant classes.

- *MainGenerator*: This class is the entry point for the code when plots are generated on a single machine.
- *MRMainGenerator*: This class is the entry point for the map reduce code.
- *ParameterLookup*: A utility class for storing and retrieving command line parameter values.

- **DataInformation:** This class is used to store, retrieve and provide other functionality to access the *attrfile* and *dfile*.
- **AttributeValue:** Ordered and categorical data items are wrapped in this class. All code assume that data record are vectors of this type.
- **INDParser:** A parser for parsing trees in the IND format, that are output by the *ConcatINDTreesUtil*.
- **TreeNode:** Representation of a single node in a tree. Also contains additional information for storing short circuit information.
- **Tree:** Class that contains a tree root and defines operations on a tree like traversal and short circuiting.
- **ShCktInfo:** Class that represents the short-circuit information stored at nodes in the tree.
- **ShortCktResult:** This class stores information that gets past back while recursively shortcircuiting a tree.
- **Plot:** Class that stores information about a single plot.
- **PlotSet:** Class that stores a set of plots and the visualization point for the plots.
- **ComputationAlgorithm:** Class that implements the different plot generation algorithms. This class is the starting point to see how the different algorithms are implemented.
- **MyNonSplitableFileInputFormat:** We use this class to deterministically split up map reduce input. This class should be dropped if one wants a more flexible way to split files across mappers.