Implement and train a Bag of Trees

This assignment has two parts; Build a random forest and then extending the random forest into MapReduce. The division of the assignment into these two parts is to facilitate your ability to complete the homework. You only need to submit the assignment after completing part 2 – there is no need to submit part 1.

DataSet: You will use this dataset to build your model:

<http://groupware.les.inf.puc-rio.br/static/har/dataset-har-PUC-Rio-ugulino.zip>

# Part 1 Build a Random Forest

**Task:**

Using the decision tree you have created you will now create a bag of trees ensemble (a simplified random forest). The long-term goal is to create a MapReduce random forest implementation, but the first step in learning how to accomplish that goal is to use the decision tree weak learner you already have made in a non-map reduce random forest implementation. This provides a basis for creating the fully MapReduce distributed ensemble method.

This random forest need only take a single input parameter N. N specifies the number of trees to grow. None of the other traditional random forest/bag of trees input parameters need to be supported at this point. The random forest model your implementation will train should be serialized in JSON format.

Your implementation must either have:

Two modes:

1) Training mode where a model is learned and built, and

2) classification mode where a test set is classified using a pre-existing model that is read in at runtime.

- OR -

Two separate programs:

one for training (required to be MapReduce enabled) and one for classification based on a trained model that is read in at runtime (MapReduce enabled not required).

**Suggested Completion Date:**

In order to provide yourself enough time to complete the migration to MapReduce in part two, it is recommended you complete this assignment by March 7th. You do not need to submit any deliverables from part 1 of this assignment.

**Requirements:**

The random forest should be implemented using good OOP, so that it can later be easily translated into the MapReduce paradigm. Java or Python are acceptable languages to use.

The random forest method does not require using cross validation. You may either utilize Out-Of-Bag (OOB) error reporting or split your data into training and testing sets for verifying your random forest functionality. The split must be 80% train, 20% test.

**Hints**:

* Remember that random forest makes its final decision by a vote of all the weak learners
  + Classification decision is the majority weak-learner classifications
  + Regression decision is the mean of the weak-learner regressions
* Again, keep your data structures clean! Adhering to good object oriented principles and taking a little extra effort to comment your code well can go a long way.
* Don’t forget that you train each tree of the forest on only a subset of the data
  + Randomly selected subset of about 2/3 of the instances (rows)
  + Randomly selected subset of about .
* There are tutorials for random forest methods readily available online.
* The basic algorithm for growing a random forest is:
  + Grow your tree and add it to the forest
  + Send the 1/3 of data not used in training down the forest and get the performance statistics. Persist these in a data structure.
  + If N > 1, iterate until N trees are grown.
  + Persist the grown forest for later use.

**Readings and Resources:**

*Bootstrap Aggregation:*

https://www.youtube.com/watch?v=5Lu1eTiX7qM

*Random Forest:*

https://en.wikipedia.org/wiki/Random\_forest

http://www.bios.unc.edu/~dzeng/BIOS740/randomforest.pdf

https://www.youtube.com/watch?v=5Lu1eTiX7qM

https://www.youtube.com/watch?v=loNcrMjYh64

*Random Subspace:*

https://en.wikipedia.org/wiki/Random\_subspace\_method

*Trees, Bagging, Boosting in General:*

http://jessica2.msri.org/attachments/10778/10778-boost.pdf

# Part 2 Extend the Random Forest to Map Reduce

**Task:**

You will now create a MapReduce implementation of the random forest you have created in part 1. This will help you come to understand at a deeper level how MapReduce enables massively parallel work to be done. It will also give you practice in learning how to divide a problem up into embarrassingly parallel chunks that fit the MapReduce paradigm well. This MapReduce random forest need only take a single input parameter N, which specifies the number of trees to grow. If you elected to create two separate programs in the prior part of the homework, you only need to extend the training program into MR (the classification program does not need to be MR enabled).

The random forest model your MapReduce implementation will train should also be serialized in JSON format.

**Minimum deliverables:**

Source code of the MapReduce random forest implementation

README.txt if more than one source file

JSON serialized model

**Due Date:** March 7th, 2017 at 9:00am

**Requirements:**

Your random forest implementation must make use of the MapReduce programming pattern. Your code must be commented, and if there is more than one source code file, you must include a README.txt file that explains the relationship of all source code files.

**Hints**:

* The driver is where the data should be read and written
* The mapper is where the tree should be grown
* The reducer is where the trees should be aggregated

**Readings and Resources:**

*MapReduce Tutorial*

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

*MapReduce IO*

https://hadoop.apache.org/docs/current/api/

https://hadoopi.wordpress.com/2013/05/27/understand-recordreader-inputsplit/

*MapReduce Random Forest Example Implementation*

http://www.biomedcentral.com/1471-2105/14/S16/S6