CSC510: Big Data Analysis with Hadoop DFS

Student: Harrison Brown Instructor: Shan Suthaharan

***Abstract*—We will explore two distinct data sets: "carpet.csv" and "hardwood.csv" through RStudio and the Hadoop Distributed File System with Spark.**

***Index Terms*—IEEEtran, LATEX, Big Data, RStudio, Hadoop, Spark, Scala**

I. Introduction: Understanding your data

Our dataset is comprised of the two classes: "carpet" and "hardwood" provided by Dr. Shanmugathasan. The RGB color model assigns each pixel in an image three distinct red, green, and blue values ranging from 0 to 255. To generate the CSV files Dr. Shan translated the images to greyscale for analysis. Now pixels within these image are assigned a single value from 0 to 255. With 0 representing a dark black pixel and 255 representing a bright white pixel.

II. Background: Describing the Data

These classes were then divided into 1,024 8x8 pixel blocks making up an area of 65536 pixels. With each block containing an area of 64 pixels. This produces 64 columns each with 1024 pixel values. The 64 pixels represent the features or dimensions of our data. Our observations come from the blocks making 1024 observations for 64 features for each class. This means we have a total of 2048 observations for our two classes with a dimensionality of 64.

III. Datasets: Key Statistics

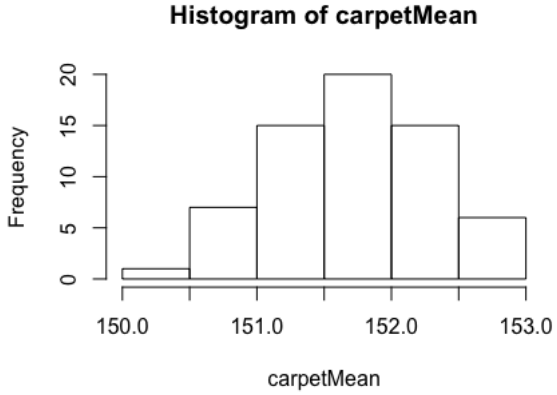
When understanding a dataset there are many key statistics one must consider. Our dataset is balanced because amongst the two classes we have the same number of observations. The dataset is complete because there are no values missing from any of our features. Next when considering the dimensionality of a dataset we need to look for an imbalance in the number of features and observations. Our dataset is not high dimensional because our 1024 observations is not much lower than our 64 features. Finally our dataset does not have any scalability problems because we don't have an increasing number of features or a complex data structure to deal with.

A. Analyzing Our Data in R

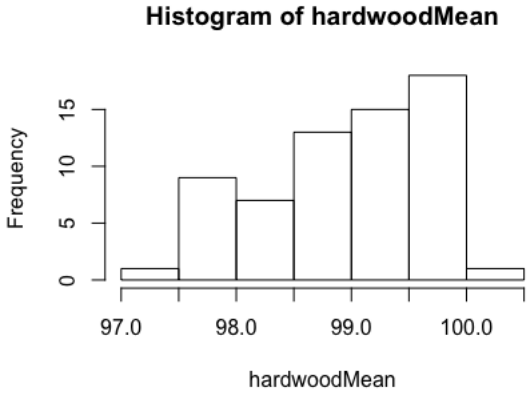
Now that we understand the data on a conceptual level it is time to analyze it using statistical models. Using RStudio I read in the two classes to begin the statistical analysis, R immediately recognizes the 64 features and 2048 observations in the right "Data" pane. Import statistics about our dataset:

B. Mean of Carpet and Hardwood

Then I viewed the mean of each feature in RStudio's console producing a range of 150.1920 to 152.9666 .To make sense of these findings I used a further line to produce the mean of all 64 features. I found carpet's mean to be 151.7124 which corresponds to a grey pixel value trending towards the brighter side of the greyscale. Then I did the same for the hardwood dataset finding a range of 97.47233 to 100.17909 corresponding to the darker side of the greyscale. Next I found the mean of all of hardwood's features to be 98.94751. Finally I plotted these means as histograms:



Nearly normal distribution



Right skewed distribution

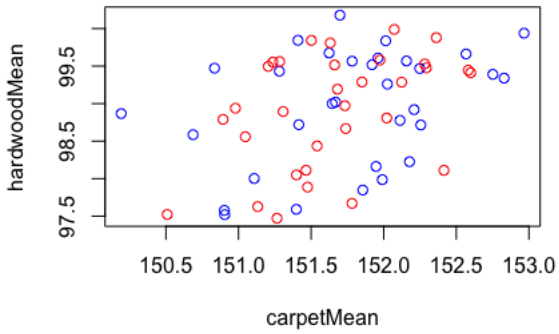
C. Standard Deviation of Carpet and Hardwood

Now it's time to consider the standard deviation of each dataset. First, considering carpet I found each feature to have a standard deviation between 21.5795 to 23.38229. This indicates a relatively uniform dataset just like the histogram revealed. Then I looked at hardwood and found it's standard

deviation of each feature to be within the range 17.38904 to 18.77171.

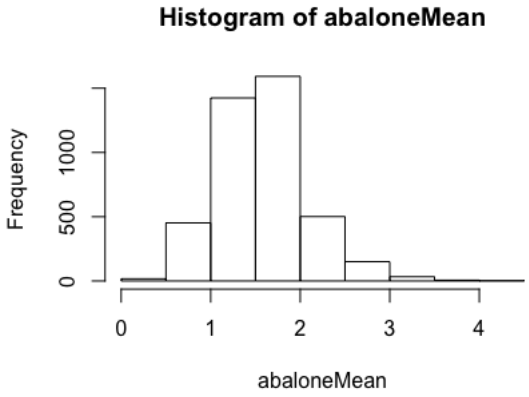
D. Conclusion: Carpet and Hardwood

To further examine these findings, I plotted the carpet and hardwood means. Data points from carpet were colored coded red and points from hardwood were colored blue. These plots



E. New Data Set: Abalone

I selected the data set abalone.csv from http://archive.ics.uci.edu/ml/datasets.html. Abalones are a type of mollusk which is sea dwelling creature with a shell. This data set represents an attempt to predict the age of an abalone based on the number of rings in it's shell. I went through a similar process with this data set to determine it's mean and plotted it in the histogram below:



Summary of Dataset:

1) This dataset has three classes: Male with 1528 observations, Female with 1307 observations, and Infant with 1342 observations for a total of 4177 observations.
2) For each observation there are 7 features
3) This dataset is balanced since for each class there is a similar number of observations
4) It is complete and accurate because these are all the datapoints from University of Tasmania's study who donated it to UCI. Nothing appears to be missing from our dataset and it was used for classification of abalone.
5) The dataset is not high dimensional because the number of observations is much lower than the number of features.
6) The dataset does not have any scalability issues, it has clear class labels and a consistent number of features
7) This process has been an example of supervised learning since Dr. Shan provided the labels for our datasets.

IV. Summary of Paper

Network intrusion is a complex issue for Big Data analysis. To detect such an intrusion on a network it must be constantly monitored in real time. Not only that but the system must be able to learn characteristics about the network traffic as they appear. This is in order to determine what traffic is acceptable and which is an intrusion. This consistent collection leads to Big Data issues as all of this data is generated. Not only does it have to be generated and archived by the system but meaningful associations must be made through machine learning.

Big Data is commonly defined using the three V's: velocity, variety, and volume. Specifically network intrusion is a Big Data problem because there are no accurate systems to predict it 100 percent of the time. A potential solution for this problem is to view the data in three dimensions based on three C's. The three C's are cardinality, complexity, and continuity. These variables create a more focused definition of Big Data based on the terms mathematical counterparts.

To manage data in such a system Hadoop is needed using cardinality as a measurement of network traffic. This begins the representation of network traffic to reveal unwanted traffic or an intrusion. This culminates in a machine learning system that can begin to detect network intrusion. The C's are crucical here because of their mathematic representations.

V. Assignment 2: Computing Environment

A. Hadoop Configuration

1) VirtualBox Installation: My first task was installing VirtualBox 5.1.8 for OS X from the following link: "VirtualBox Downloads".

2) Ubuntu Installation: Next I downloaded the disk image for Ubuntu 16.04.1 LTS, which is the latest 64-bit version for desktop PCs from this link "Download Ubuntu Desktop". Then I configured a VirtualBox called Hadoop Environment using Ubuntu as the boot disk. For this virtual machine I allocated 4 gigabytes of ram, a duo-core processor, and dynamic storage space.

3) Setting up Cloudera Hadoop: First, on my VirtualBox I downloaded the Cloudera CDH4 binary from their archive here "Cloudera CDH4 Single Click Install". Then I installed the package in the terminal, checked for updates, and installed the MapReduce JobTracker with the following commands:

```
$ sudo dpkg i cdh4repository_1.0_all.deb
$ sudo apt-get update
$ sudo apt-get install hadoop-0.20-mapreduce-
  jobtracker
```

Second, I specified the directories for HADOOP HOME and HADOOP CMD respectively:

```
$ export HADOOP_HOME=/usr/local/hadoop
$ export HADOOP_CMD=$HADOOP_HOME/bin/hadoop
$ export PATH=$HADOOP_HOME/bin:$PATH
```

4) Spark and Scala Standalone Deploy Mode: Now, I was ready to install Spark and chose the standalone deploy mode which relies on their own clustered environment. To begin I downloaded the Java repository, checked for updates, and installed Java 7.

```
$ sudo apt-add-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installe
```

Then I established the reference to Java in my Hadoop Environment:

```
$ export JAVA_HOME=/usr/lib/jvm/jdk1.6.0_45/
$ export PATH=$JAVA_HOME/bin:$PATH
```

Next, I prepared for the installation of Scala by creating a folder for it's binary file called Scala. Then I downloaded Scala from "Scala-2.11.7.deb", moved it to the Scala folder, and installed.

```
$ sudo mkdir /usr/local/src/Scala
$ sudo cp /home/hpbrown/Downloads
    /scala-2.11.7.deb / usr/local/src/Scala/
$ sudo cp /usr/local/src/Scala
$ sudo dpkg -i scala-2.11.7.deb
$ sudo apt-get update
$ sudo apt-get install scala
```

5) Installing Spark: Now, I was ready to install Spark and download their pre-built binary here: "Spark-1.2.0 Binary". This produced no problems for me until I went to decompress the tgz file to install Spark. About half-way through extracting the file my terminal returned the following error: "No space left on device". At first, I was incredibly confused, how could my Virtual Machine run out of space when storage was dynamically allocated? I discovered the problem using:

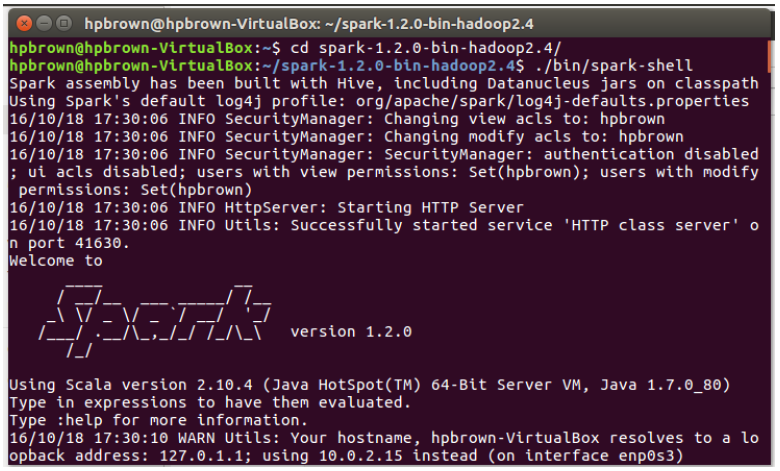
```
$ df-h
```

It turns out one of temp folders had reached it's maximum capacity, specifically: "/dev/sda1" was 100 percent full. I knew deleting files from this directory at random had tremendous potential to ruin the progress I had made in setting up Hadoop and began a Google search. Eventually, determining I had used too many kernels and needed to remove some that weren't currently in use. I followed these instructions from "Removing Unused Kernels":

```
$ sudo dpkg -l linux-image-\* | grep ^ii
$ sudo kernelver=
$(uname -r | sed -r 's/-[a-z]+//')
    dpkg -l linux-{image,headers}-"[0-9]*" |
    awk '/ii/{print $2}' | grep -ve
$kernelver
$ sudo apt-get purge
$(dpkg -l linux-{image,headers}-"[0-9]*" |
    awk '/ii/{print $2}' | grep -ve
"$(uname␣-r␣|␣sed␣-r␣'s/-[a-z]+//')")
```

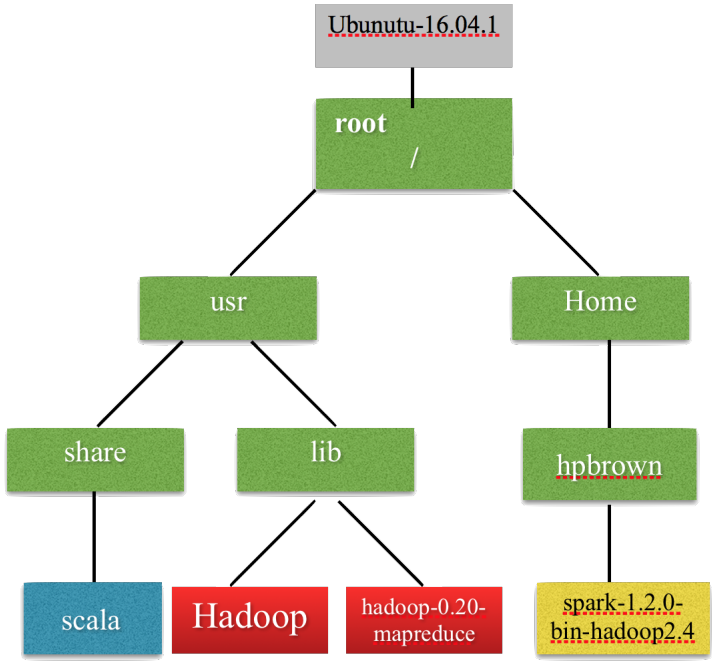
These commands allowed me to view all kernels in use, then view all kernels that weren't active, and finally remove them. Then hoping for the best, I checked the status of "/dev/sda1" and was thrilled to see it was only 93 percent full. Now, I was able to install Scala and finally access the Spark shell.

```
$ df-h
$ tar -xzvf spark-1.2.0-bin-hadoop2.4.tgz
$ cd spark-1.2.0-bin-hadoop2.4/
$ ./bin/spark-shell
```



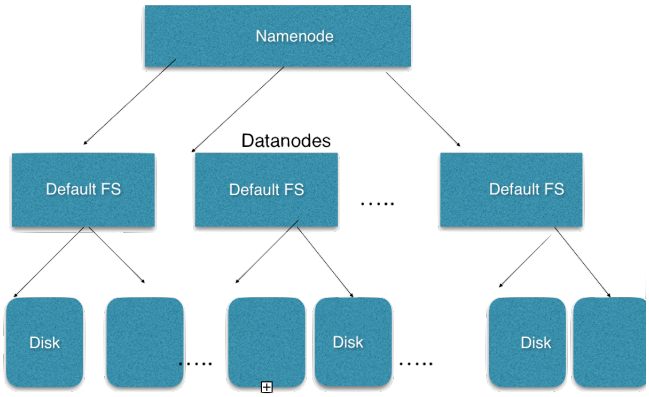
Spark Shell

B. Understanding Hadoop Distributed File System



Directory Structure

C. Understanding Hadoop Distributed File System



Hadoop Structure

D. Programming Example

In assignment 1 I selected the abalone.csv dataset for analysis which is comprised of three classes: male, female, and infant. For assignment 2 I wrote a program in RStudio to read in the abalone.csv dataset and separated it based on these classes. This program can be found in the folder Assignment2 along with the csv files:

- abaloneMale.csv
- abaloneFemale.csv
- abaloneInfant.csv

E. MapReduce Examples

1) Standard Example: After installing Hadoop, you are given a MapReduce example for testing. Wordcount is a fundamental example of MapReduce in action, it's like the "Hello World" of Big Data. For this example we will run Worcount in the Spark Shell. I've selected a document called "transcript.txt" for analysis containing the full transcript of the third presidential debate between Hillary Clinton and Donald Trump. Once we've completed this example we will discover the frequency of words used during the debate.

- Once we are in the Spark Shell the first step is to read our input "transcript.txt" with the following command:

  val inputFile = sc.textFile("/home/hpbrown/Desktop/transcript.txt") (1)

- Next we create a variable called "counts" to hold our wordcount and split the file based on the spaces between words:

  val counts = inputFile.flatMap(line => line.split(" ") (2)

- Then we apply the function map to this variable to define our key value pair. By selecting word and 1 here we define a relationship where the key is each word and the value is incremented by 1 each time the key is discovered. This is written in Scala as:

  $$.map(word => (word, 1)) \quad (3)$$

- The next step is to apply our reducer function to reduce each word to a single instance and increment our value by one for each occurrence:

  $$.reduceByKey(\_+\_); \quad (4)$$

- Finally, we can save the result from our variable "counts" in a folder called output:

  counts.saveAsTextFile("/home/hpbrown/Desktop/output") (5)

- These files can be found in my submission in the path: /Assignments1;3/Assignment2/Standard Example/

2) My Example: For my own example, I've decided to use the MapReduce program to count the occurrences of each class in the abalone dataset. Within this dataset each class is determined by a letter at the beginning of each row: M, F, or I.

- First we begin by reading in the abalone dataset

  val abalone = sc.textFile("/home/hpbrown/Desktop/abalone.csv (6)

- Next we split the file based on the each new line since that is how the classes are defined:

  val abaloneCounts = abalone.flatMap(line => line.split("\n")) (7)

- Then we map our class labels using the map function:

  $$.map(word => (word.chartAt(0), 1)) \quad (8)$$

- Then we reduce by the and add the occurrences of each class:

  $$.reduceByKey(\_+\_); \quad (9)$$

- Finally, we can save the result from our variable "abaloneCounts" in a folder called abaloneOutput using the default partition of two:

  abaloneCounts.saveAsTextFile("/home/hpbrown/Desktop/abaloneOutput") (10)

3) Conclusion: In conclusion we find there are 1307 female, 1528 male, and 1342 infant observations respectively.

- These files can be found in the path: /Assignments1;3/Assignment2/My Example/

4) Hadoop VS MacOS: After completing "My Example" in the Spark Shell with Scala in Hadoop, I ran it again to time it. Then ran the same program in RStudio on MacOS. I was unable to detect any significant time difference in the runtimes. This is to be expected when dealing with a dataset that is the size of abalone.csv.

VI. Assignment 3: Machine Learning

A. Support Vector Machine

Suppose we have two classes of objects plotted on an X and Y plane and we want to create a clear division between the two. A Support Vector Machine will help us optimize this class division by producing a plane that separates them. Not only will this line separate the classes but by optimizing it we create the largest possible division in terms of the planes

length and width. While an SVM is an excellent and applicable tool for solving Big Data problems today it has one major drawback. Not only is it complex in terms of mathmatetics, it is computationally time consuming.

1) Linear SVM Example: Consider the X and Y plane mentioned above with two classes comprised of a single observation each: a "+" and a "-". Using a Support Vector Machine we will create the longest and widest linear domain division possible.

- To begin we have a hyperplane to represent our normal vector to determine the lines orientation called 'v'. Then we consider the bias of the line or its displacement, we can call this bias 'b': Together they form the equation:

$$(v^T) * x + b = 0 \tag{11}$$

- The width of our vector is determined by 2 divided by the length of our vector:

$$2/||v|| \tag{12}$$

To create our SVM we must solve an optimization problem consisting of three linear equations: For the side closest to the "+" class we have:

$$(v^T) * x + b = 1 \tag{13}$$

For the center of the SVM we have:

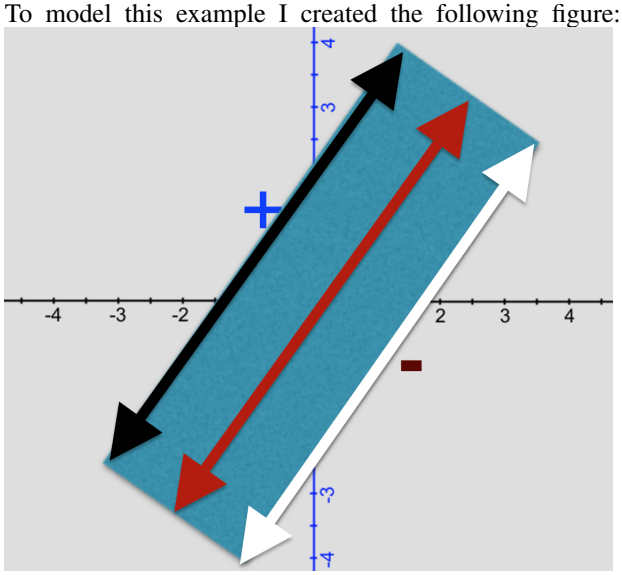
$$(v^T) * x + b = 0 \tag{14}$$

For side closest to the "-" class we have:

$$(v^T) * x + b = -1 \tag{15}$$

2) Figure:

- To model this example I created the following figure:



- Where the black line is the equation:

$$(v^T) * x + b = 1 \tag{16}$$

- The red line is the equation:

$$(v^T) * x + b = 0 \tag{17}$$

- The white line is the equation:

$$(v^T) * x + b = -1 \tag{18}$$

- Lastly, our linear support vector machine is generated by solving the optimization problem resulting in the blue SVM.

B. Decision Tree Learning

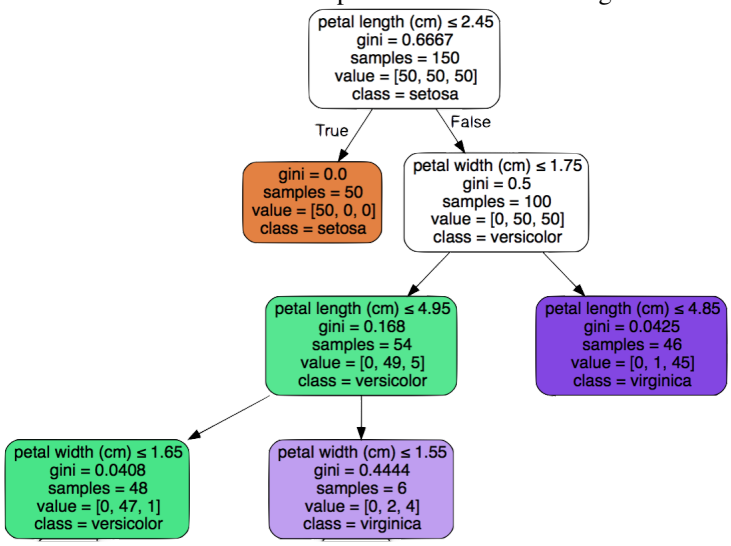
The decision tree technique is unique in the field of Big Data for its ability to produce an easy to understand representation of our machine learning solution. It is also well equipped to deal with multi-class datasets. To begin a dataset is selected and imported into your programming environment. Then this dataset is broken up into two groups: training and testing. Our testing class will be set aside and later used to see how well our classifier was trained. Then using our training group we train our classifier to predict the label. We then go back to our testing group to see how well we trained for classifier. This generates a decision tree that begins with a root feature that leads down a path of leaves based on decisions. As each decision is made moving down the leaves you get closer to the correct class.

1) Example: Consider the dataset https://en.wikipedia.org/wiki/Iris_flower_data_set that describes three different types of the flower Iris: Iris setosa, Iris virginica, and Iris versicolor

- The dataset contains 50 observations of each of these classes
- Each observation contains four features: sepal length, sepal width, petal length, and petal width. After machine learning the result is a complex but easy to understand decision tree.
- For the sake of our example let's suppose only two features are needed to classify the Iris into their correct species: petal length and width

2) Figure:

- Following a decision learning tree technique our two features would produce the following:



- You can see here our classifier was trained to produce this decision tree, where decisions made along the leaves lead us down different paths. Once enough decisions have been made to classify our flower a resulting class label is assigned.

- Say for example we had a petal length greater than or equal to 2.45 centimeters the class label setosa is assigned.
- The above diagram and example has been modified from the following source: http://scikit-learn.org/stable/modules/tree.html

C. Random Forest

This machine learning technique is best suited for large datasets and is an extension of the Decision Tree algorithm. The Random Forest technique follows the same steps as the Decision Tree algorithm but instead of making a decision tree based on the entire dataset it makes a collection of trees. This collection is produced from random subsets of our dataset, giving the algorithm it's name: Random Forest. These selections are totally random and may or may not containing overlapping data points. These trees have two key features:

- The first being most of the time a random tree can provide the correct label for most of our data
- Second, while there are mistakes in an individual tree since our trees are random mistakes are spread out throughout the forest.

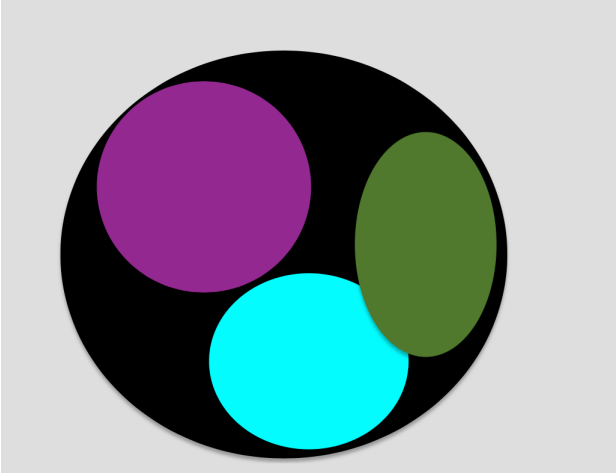
This means when running classification analysis on a dataset with the random forest method, the class predicted by most of the trees is the assigned label.

1) Example: Let's consider a large dataset with two distinct classes of whales: blue and killer whales.

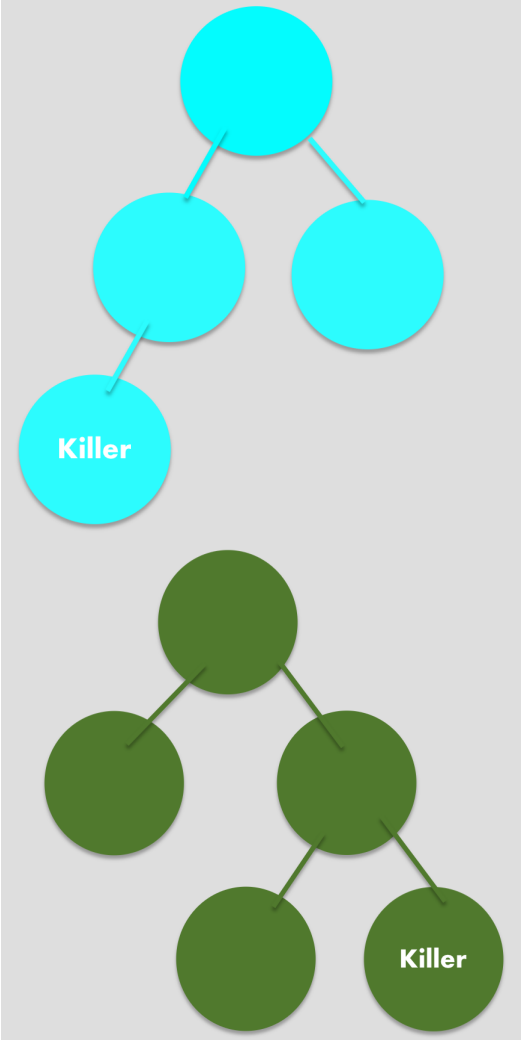
- We would begin our implementation of the Random Forest much like the decision tree by selecting two subsets of our dataset for training and testing.
- Next, we would begin training our classifier by producing a collection of random decision trees.
- Then, we would see what label is predicted by most of the trees in our forest.
- Finally, the class label is assigned based on the selection most of the trees make and a degree of accuracy is assigned. The better we've trained our classifier the better our accuracy at predicting the correct species of whale.

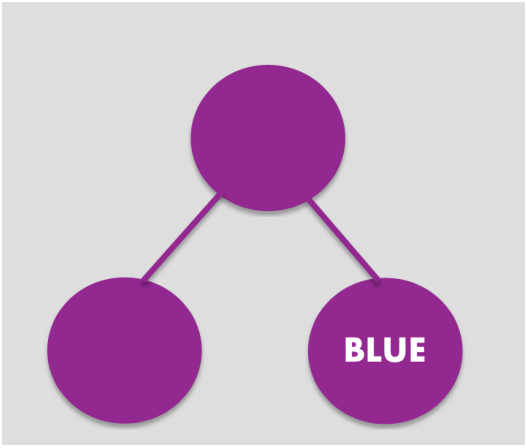
2) Figures: Now let's visualize this example

- First our algorithm makes random selections from our dataset, that could look something like this. With the black circle representing our entire dataset and the purple, green, and blue circles representing randomly selected subsets.:



- Next these random subsets produce a Random Forest of decision trees. In the figures below we see our forest and the predicted class labels for a given observation:





- In conclusion since most of the trees predicted the class killer corresponding to killer whale, the label killer is assigned to the observation

VII. Simulation, Results, and Discussions

A. Selected Machine Learning Technique

For the next phase of the assignment, we will consider two datasets produced from the abalone dataset mentioned previously: maleAbalone.csv and femaleAbalone.csv. I've selected the Random Forest algorithm for analysis for a few reasons:

1) First, we could use a single decision tree but since we have a high volume of data the Random Forest should be better. It was developed after all to account for some of the decision tree's shortcomings.
2) Second, the random forest is great when using a balanced dataset like abaloneMale and abaloneFemale.
3) Finally, since we are working on a supervised learning example in Hadoop the Random Forest technique will work well with distributed file system and MapReduce frameworks.

B. Simulation and Results

1) Random Forest in R:

- We begin by increasing our data using the function rnorm:

$$rnorm(100, mean = 0.5614, sd = 0.1027) \tag{19}$$

- Then we established our class labels
- Generated a Random Forest
- It predicted with the following accuracy: 82.3 percent
- The full code can be found in the assignment3 folder included with this submission.

2) Random Forest in Scala:

- First we begin by reading in the abalone dataset after increasing it's volume and only including Male and Female classes:

$$val\ abalone = sc.textFile(\text{"/home/hpbrown/Desktop/abalone.csv} \tag{20}$$

- Second we convert our dataset to a RDD

$$val\ rdd = sc.parallelize(csv) \tag{21}$$

- Next we split the lines using ","

$$val\ data = rdd.map(\_.split(\text{","})) \tag{22}$$

- Drop missing values from column 2

$$.filter(\_(3) != \text{"?"}) \tag{23}$$

- Drop IDs from the first column

$$.map(\_.drop(1)) \tag{24}$$

- Next convert our values to floating points

$$.map(\_.map(\_.toDouble)) \tag{25}$$

- Now our data is ready for classification, we need to import a few packages for our Random Forest

$$import\ org.apache.spark.mllib.linalg.Vectors \tag{26}$$

$$import\ org.apache.spark.mllib.regression.LabeledPoint \tag{27}$$

- Now we create variable to hold our features and labels, we will use 0 for male and 1 for female.

$$vallabeledPoints = data.map(x => LabeledPoint \tag{28}$$

$$(if(x.last == 0.5614) \tag{29}$$

$$1else0, Vectors.dense(x.init))) \tag{30}$$

- Split labeledPoints into a test dataset of 80 and training data of 20 perecent

$$val\ splits = labeledPoints.randomSplit(Array(0.7, 0.3) \tag{31}$$

$$, seed = 5043l) \tag{32}$$

$$valtrainingData = splits(0) \tag{33}$$

$$valtrainingData = splits(1) \tag{34}$$

- Define our trees and data parameters which requires a few more Spark imports. We will use a maximum tree depth of 6 and 30 trees. The other parameters are standardized when using the Random Forest learning technique.

$$import\ org.apache.spark.mllib.tree.configuration.Algo \tag{35}$$

$$import\ org.apache.spark.mllib.tree.impurity.Gini \tag{36}$$

$$val\ algorithm = Algo.Classification \tag{37}$$

$$val\ impurity = Gini \tag{38}$$

$$val\ maximumDepth = 6 \tag{39}$$

$$val\ treeCount = 30 \tag{40}$$

$$val\ featureSubsetStrategy = \text{"auto"} \tag{41}$$

$$val\ seed = 5043 \tag{42}$$

- Next we will build our training model using the import "Strategy" for configuration and Spark's Random Forest package. (long equations like initializing our model have been split across multiple lines)

$$import\ org.apache.spark.mllib.tree.configuration.Strategy \tag{43}$$

$$import\ org.apache.spark.mllib.tree \quad (44)$$

$$.configuration.RandomForest \quad (45)$$

$$val\ model = RandomForest.trainClassifier(trainingData, \quad (46)$$

$$newStrategy(algorithm, impurity, maximumDepth), \quad (47)$$

$$treeCount, featureSubsetStrategy, seed) \quad (48)$$

- Evaluate the model for accuracy using MulticlassMetrics package.

$$import\ org.apache.spark.mllib.evaluation.MulticlassMetrics \quad (49)$$

$$valevaluationMetrics = \quad (50)$$

$$newMulticlassMetrics(labeledPredictions.map(x => (x._1, x._2))) \quad (51)$$

- Finally, we Asses the accuracy of our Random Forest

$$evaluationMetrics.precision \quad (52)$$

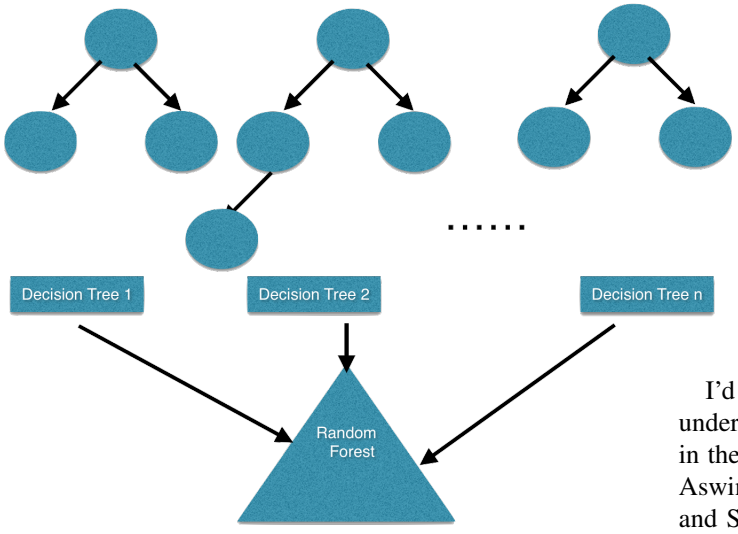
- It printed: 94.3%.

C. Discussion and Findings

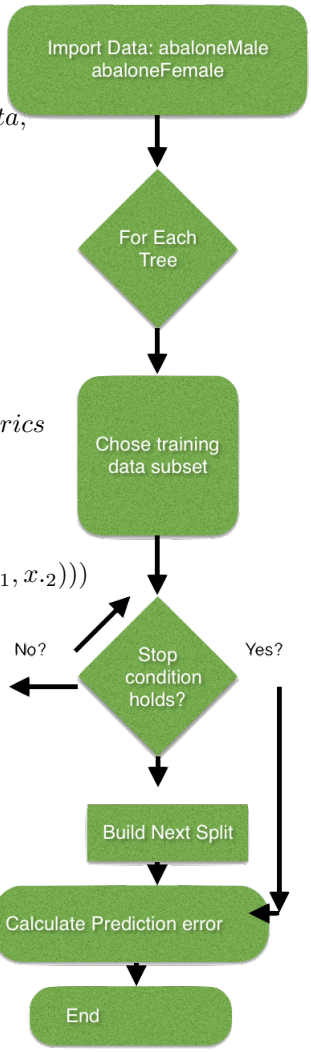
Hadoop with the help of Spark was the clear winner here, for multiple reasons:

1) In final two examples it moved noticeably faster than R.
2) Scala's syntax was more compact and easy to understand.
3) Not only was it faster it computed a result with greater accuracy.

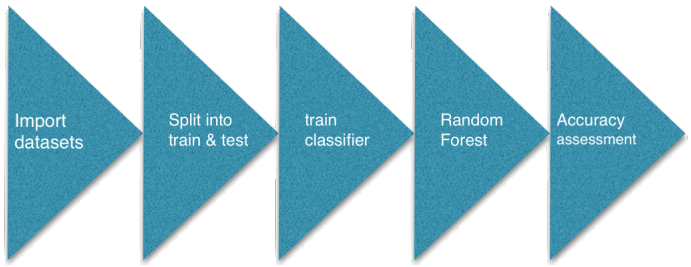
D. Structure Diagram



E. Data Flow Diagram



F. Process Diagram



VIII. Conclusion

Acknowledgments

I'd like to thank Dr. Shan Suthaharan for helping me understand the field of Big Data through immersing myself in the Hadoop Distributed File System. Also, i'd like to thank Aswini Sen for her fantastic presentation on installing Spark and Scala.

REFERENCES

[1] S. Suthaharan, Machine learning models and algorithms for big data classification: Thinking with examples for effective learning, 1st ed. Springer.

[2] [Online]. Available: http://archive.ics.uci.edu. Accessed: Nov. 1, 2016.

[3] S. Suthaharan, "Big data classification," ACM SIGMETRICS Performance Evaluation Review, vol. 41, no. 4, pp. 70?73, Apr. 2014.

[4] J. Dean and S. Ghemawat, "MapReduce," Communications of the ACM, vol. 51, no. 1, p. 107, Jan. 2008.

[5] "Apache Hadoop 3.0.0-alpha1 ? HDFS architecture," 2016.

[6] Image, "1.10. Decision trees ? scikit-learn 0.18.1 documentation," 2010. [Online]. Available: http://scikit-learn.org/stable/modules/tree.html. Accessed: Nov. 1, 2016.

[7] Google Developers, "Visualizing a decision tree - machine learning recipes 2," in YouTube, YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=tNa99PG8hR8. Accessed: Nov. 1, 2016.

[8] A. Tait, "How to predict outcomes using random forests and spark — learning tree Blog," in Big Data, Learning Tree Blog, 2015. [Online]. Available: http://blog.learningtree.com/how-to-predict-outcomes-using-random-forests-and-spark/. Accessed: Nov. 1, 2016.

[9] T. Kodali, "Predicting wine quality using random forests," R-bloggers, 2016. [Online]. Available: https://www.r-bloggers.com/predicting-wine-quality-using-random-forests/. Accessed: Nov. 1, 2016.