Exploratory Analysis of the UCI Forest Covertype and NSL-KDD Datasets Using Hadoop, Machine Learning, and Big Data Techniques

September 28, 2014

STUDENT

Harry Rybacki

CSC495/693 - Big Data Analytics and Machine Learning
hrybacki@gmail.com

INSTRUCTOR

Dr. Shan Suthaharan

Department of Computer Science, UNC-Greensboro
s_suthah@uncg.edu

Abstract

This paper focuses on exploratory big data analysis of two datasets: The UCI Forest Covertype dataset and the NSL-KDD dataset. After a brief review of the background information in the field, both datasets will be explored using rudimentary statistical analysis. Specifically, we will analyze each dataset for inaccuracy, incompleteness, and imbalance. Next, the computing environment necessary for big data analysis will be discussed as well as providing a detailed installation guide for Hadoop with simple examples of its usage. Finally, big data and machine learning analysis techniques will be used with Hadoop and Python in an attempt to classify observations. Specifically, the Stochastic Gradient Decent (SGD) algorithm is used in collusion with feature hashing to allow for similar scaling of both the size of the dataset and the computational complexity of the analysis. These results will be compared against the aforementioned datasets having been manually altered to be inaccurate, incomplete, or imbalanced to give us a solid benchmark to rate the effectiveness of our algorithm and computing environment choices. This paper hopes to find that Hadoop will prove to be more effective than less distributed computing environments as well as show the effectiveness of SGD and feature hashing by comparing the known good dataset analysis to the manually altered dataset analysis.

I. Introduction

TBA

II. Background

Managing big data is difficult. As the number of features grows, the rate at which observations are stored, and the data varies more the storage and processing becomes increasingly more complex[2]. As a result, a combination of technologies designed to assist in the storage and processing as well as a careful choice of analysis algorithms is essential to getting any use from the data.

In his paper, Suthaharan recommends the use of Hadoop Distributed File Systems (HDFS) and Cloud Technologies to assist in the storing of data as well has the communication infrastructure of the analysis network.

As mentioned by Dalessandro, as datasets grow linearly the computational complexity of standard statistical analysis algorithms grows exponentially[1]. However, using the Stochas-

1: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

tic Gradient Decent (SGD) algorithm can alleviate much of the computational complexity gains. Despite being less optimal on smaller datasets, SGD takes advantage of sparsity within datasets and searches for min/max of individual data points making it scale linearly with the dataset.

Furthermore, the use of feature hashing lessens the aforementioned problems surrounding high feature dimensionality. Although feature hashing degrades the quality of the data it allows for working with incredibly large, millions or billions, amounts of features[1] in a way that scales linearly.

III. Datasets

I. NSL-KDD Dataset

The NSL-KDD Dataset, obtainable here: `http://nsl.cs.unb.ca/NSL-KDD/`, was constructed to resolve inherent problems in a similar dataset that was built in 1999[3]. The previous dataset was imbalanced and lead to bias toward more frequent attacks. This issue is however no longer present with the current dataset. Furthermore it is complete and there are no chunks of the dataset missing. We will have to assume that when this dataset was constructed the tools used to capture this information were accurate and the dataset was not tampered with in a way that would lead to inaccuracy.

With 125,973 observations, each of which has forty features, we can conclude that his is a high dimensional dataset. Furthermore, if an observer were to be collecting similar data in a live environment (an active network) the velocity of number of observations would increase at a non-trivial rate.

INSERT HISTOGRAM OF PROTOCOL TYPES

II. UCI Forest Covertype Dataset

The UCI Forest Covertype Dataset[4], obtainable here: `https://archive.ics.uci.edu/ml/datasets/Covertype`, was constructed for predictive modeling of forested lands the neighbor forested lands under the control of the original dataset owners. This dataset contains 581,012 observations each of which has fifty-four features. Relevant statistical information for each feature was easy to calculate. For example, feature 1, elevation, has a 581,012 observations, a mean of 2,959 meters, and a standard deviation of 280 meters.

INSERT HISTOGRAM OF ELEVATION

We can state that this dataset is not imbalanced as there are an equal number of , inaccurate, or incomplete.

IV. Computing Environment

I opted to set up my Hadoop environment using the online cloud hosting provider, Digital Ocean. Compared to similar services provided by Linode and AWS (Amazon Web Services), Digital Ocean is relatively inexpensive. Furthermore, having contacted individuals that have set up Hadoop environments using Digital Oceans services I know that not only do they offer outstanding customer support should I need it but also that Digital Ocean provides mostly-complete guides describing how to set up Hadoop. For example, `https://www.digitalocean.com/community/tutorials/how-to-install-hadoop-on-ubuntu-13-10` provides thorough documentation for installing a single-node Hadoop environment on Ubuntu 13.10. I will be using and expanding upon this guide as a base for my setup with one exception, I will be using the current version of Ubuntu, 14.04.

The use of Digital Ocean servers allows me to access my Hadoop Environment from virtually anywhere using SSH. For my environment, I will be using a server allocated with 2GB of RAM, a 40GB SSD drive, and two processor cores. I may opt to upgrade to a larger Digital Ocean droplet (sever) if these resources seem inadequate.

2: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

I. Hadoop Configuration

1. Set Up Server on Digital Ocean

   (a) Visit `http://digitalocean.com`

   (b) Sign in – If you have not created an account, follow the "CREATE ACCOUNT" instructions on the landing page.

   (c) Create a Droplet

      i. Click the green "CREATE" button on the top left corner of your menu
      ii. Enter a name in the "Droplet Hostname" Field
      iii. Select the $ 20/mo droplet option
      iv. Select the New York region
      v. Select Ubuntu 14.04x64 from the "Select Image" options
      vi. Click "Create Droplet" at the bottom

   (d) SSH into your droplet

      i. Check your email for a message from Digital Ocean and note the droplet <IP_ADDRESS> and <ROOT_PASSWORD>
      ii. Open up your desired way to SSH into a remote server
      iii. Enter the following into your terminal

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)

Documentation:  https://help.ubuntu.com/

System information as of Thu Sep 11 11:19:12 EDT 2014
System load:  0.01              Processes:           77
Usage of /:   3.9% of 39.25GB   Users logged in:     0
Memory usage: 3%                IP address for eth0: 104.131.45.156
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

0 packages can be updated.
0 updates are security updates.

Last login: Thu Sep 11 11:19:12 2014 from ip-152-13-249-96.uncg.edu
```

Listing 1: *Server Output from Digital Ocean*

2. Install and Test Hadoop

   (a) Install Java

```
# update all of the OS packages
$ apt-get update

# Install Java as it is needed by Hadoop, -y informs apt-get to select
    yes for any options the user might be queried with
$ apt-get install default-jdk -y

# Ensure Java is installed resulting in output akin to:
$ java --version
```

3: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

```
java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.1) (7u65-2.5.1-4ubuntu1
    ~0.14.04.2)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Listing 2: *Server Output from Installing Java*

(b) Create and Setup SSH Certificates

```
# Hadoop uses SSH to access its nodes and setting up SSH keys ahead of
    time will prevent tedious password entry later on.

# create an ssh key using the RSA algorithm with an empty passphraseIf
    you are prompted for a filename just accept the default by hitting
    enter unless you have a specific name convention you would like to
    follow
$ ssh-keygen -t rsa âĂŞP ''

# read and output the new key into the authorized ssh keys for this
    system.
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Listing 3: *Server Output from Setting Up SSH*

(c) Download and Install Hadoop

```
# fetch the tarballed Hadoop from the Apache website
$ wget http://apache.mirrors.lucidnetworks.net/hadoop/common/hadoop
    -2.3.0/hadoop-2.3.0.tar.gz

# uncompress and unpack the Hadoop tarball
$ tar xfz Hadoop-2.3.0.tar.gz

# move the Hadoop install to itâĂŹs correct system location
$ mv hadoop-2.3.0 /usr/local/hadoop
```

Listing 4: *Server Output from Downloading Hadoop*

(d) Edit and Setup Configuration Files:

i. Determine location of Java libraries

```
# Display your java installation directory with output akin to:
$ update-alternatives âĂŞconfig java

There is only one alternative in link group java (providing /usr/bin
    /java): /usr/lib/jvm/java-7-openjdk-amd64/jre/bin/java

# <JAVA_HOME> will be everything in the path preceding /jre/bin/java
# <JAVA_HOME> = /usr/lib/jvm/java-7-openjdk-amd64
```

Listing 5: *Server Output from Editing Determining Java Library Directory*

ii. Edit ~/.bashrc

```
# Add Hadoop specific variables to your \textasciitilde/.bashrc file
    . Upon loading a new shell, these environment variables specific
     to and needed by Hadoop will automatically be loaded. Open up
    ~/.bashrc in your favorite text editor, scroll to the bottom,
    and add the following environment variables
```

4: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

```
# Hadoop Environment Variables
export JAVA_HOME=<JAVA_HOME>
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="Djava.library.path=$HADOOP_INSTALL/lib"
```

Listing 6: *Server Output from Editing .bashrc*

iii. Populate environmental variables

```
# Now, these variables won't be loaded until the next time a shell
    is created so you must manually source the bashrc file

$ source ~/.bashrc
```

Listing 7: *Server Output from Sourcing New Env. Variables*

iv. Edit /usr/local/hadoop/etc/hadoop/hadoop-env.sh

```
# We must now tell Hadoop where Java is located on the system so
    update the 'export JAVA_HOME...' line to match your <JAVA_HOME>.
     This will ensure that Hadoop knows where Java is every time it
    is started. E.g:

# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

Listing 8: *Server Output from Editing hadoop_env.sh*

v. Edit /usr/local/hadoop/etc/hadoop/core-site.xml

```
# core-site.xml contains config properties Hadoop uses upon startup.
     We will specify where the hdfs web server should run by adding
    the following within the <configuration></configuration> tags

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:9000</value>
</property>
```

Listing 9: *Server Output from Editing core-site.xml*

vi. Edit /usr/local/hadoop/etc/hadoop/yarn-site.xml

```
# yarn-site.xml contains config properties MapReduce uses upon
    startup. We need to specific the auxiliary services it uses as
    well as what time of shuffle class to use. We'll stick to the
    defaults as this is a simple environment: Open the file and add
    the following betwee the <configuration></configuration> tags

<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
```

5: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

```
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Listing 10: *Server Output from Editing yarn-site.xml*

vii. Create and Edit /usr/local/hadoop/etc/hadoop/mapred-site.xml

```
# mapred-site.xml specifies which framework used by MapReduce.
    Before modifying it, we must first copy a template withour our
    hadoop directory

$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/
    local/hadoop/etc/hadoop/mapred-site.xml # copy the template

# Now we can specify the yarn framework we setup in yarn-site.xml.
    Open mapred-site.xml and add the following between the <
    configuration></configuration> tags

<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

Listing 11: *Server Output from Editing mapred-site.xml*

viii. Edit /usr/local/hadoop/etc/hadoop/hdfs-site.xml

```
# hdfs-site.xml specifies each host in the cluster. In our case, we
    will be hosting our NameNode and DataNode on the same system. So
    , we must create directories to hold each accordingly.

$ mkdir -p /usr/local/hadoop_store/hdfs/namenode
$ mkdir -p /usr/local/hadoop_store/hdfs/datanode

# Now we most tell Hadoop how many replicas of block storage we want
     the DataNode to keep as well as where these directories are by
    adding the following between the <configuration></configuration>
     tags

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
```

Listing 12: *Server Output from Editing hdfs-site.xml*

(e) Format the Hadoop Filesystem. With our configuration files set up we are ready to format the HDFS so that we can start our dfs.

6: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

```
# With our configuration files set up we are ready to format the HDFS
    so that we can start our dfs.

$ hdfs namenode -format

# A few notes: First, only run this command successfully once. Running
    it subsequent times will format (destroy) any data your HDFS is
    storing. Second, you will see a lot of output from this command.
    Take note of the exit code; if all was successfull it should be 0
    as in the example output below.

root@hadoopbox:~# hdfs namenode -format
14/09/11 21:32:20 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************

...ouput cut out for brevity

14/09/11 21:32:21 INFO namenode.NNStorageRetentionManager: Going to
    retain 1 images with txid >= 0
14/09/11 21:32:21 INFO util.ExitUtil: Exiting with status 0 # This is
    what we want to see
14/09/11 21:32:21 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoopbox/127.0.1.1
************************************************************/
```

Listing 13: *Server Output from Formatting HDFS*

(f) Start Hadoop

```
# Now that everything is set up and the filesystem has been formatted
    we can start Hadoop by executing the following commands

# Note: you will be prompted to enter 'yes' twice after executing the
    first command. These prompts are merely adding the ssh key to
    known_hosts and shouldn't pop up again.

$ start-dfs.sh
$ start-yarn.sh
```

Listing 14: *Server Output from Starting Hadoopl*

(g) Confirm all Hadoop services are running

```
# Everything should be up and running. However, one quick command will
    confirm this as well as provide us with the PIDs for each process.

$ jps # Should lend output akin to

11305 Jps
10887 ResourceManager
10742 SecondaryNameNode
10554 DataNode
10404 NameNode
11018 NodeManager

# And with that, Hadoop is up and running.
```

Listing 15: *Server Output from Confirming Hadoop Install*

7: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG

V. Machine Learning

TBA - Assignment 3

VI. Conclusion

TBA

VII. Acknowledgement

1. Dr. Shan Suthaharan, Associate Professor, University of North Carolina at Greensboro

References

[1] Dalessandro (2013). Bring The Noise: Embracing Randomness is the Key to Scaling Up Machine Learning Algorithms

[2] Suthaharan (2013). Big Data Classification: Problems and Challenges in Network Intrusion Prediction with Machine Learning

[3] ADD: NSL-KDD Dataset

[4] ADD: UCI Tree Cover Dataset

8: This course is developed and taught by Dr. Shan Suthaharan at the Department of Computer Science, University of North Carolina at Greensboro (UNCG) in Fall 2014. The development and delivery of this course is funded by the Center for Science of Information, Purdue University through a sub-award approved by the National Science Foundation, and partially funded by UNCG