PROJECT 2 – MICROSOFT MALWARE CLASSIFICATION CHALLENGE

Instructions to Execute Code On AWS Cluster:

1. Before running the script, to ensure all python libraries and dependencies are installed, download and install anaconda **on each node**. Steps to download anaconda are given in the appendix.

2. In the conf folder of spark installation, add the following statements (amazon credentials and anaconda python path) in spark-env.sh. This needs to be **done on all the nodes**.

   ```
   export AWS_ACCESS_KEY_ID='aws_id'
   export AWS_SECRET_ACCESS_KEY='aws_key’
   export PYSPARK_PYTHON=$ANACONDA_HOME/bin/python
   ```

3. Install git and clone the git invasion repo on master node.

   ```
   sudo yum install git
   git clone "https://github.com/eds-uga/invasion.git"
   ```

4. To submit the job,

   ```
    ./bin/spark-submit \
   --master spark://<IP address of master node>:7077  \
   --driver-memory 2g \
   --executor-memory 3g \
    --total-executor-cores 20 \
   --conf spark.driver.maxResultSize=8g  \
   --packages org.apache.hadoop:hadoop-aws:2.7.1 \
   <path to random2.py > <bytePath> <namePath> <nameTestPath> <classPath>
   <aws_id> <aws_key>
   ```

   where,
   <bytePath> <namePath> <nameTestPath> <classPath> <aws_id> <aws_key>
   are command line arguments that need to be specified.

   Sample Values:

   bytePath = "s3n://eds-uga-csci8360/data/project2/binaries"

   //s3 path for folder with all .byte files

   namePath = "s3n://eds-uga-csci8360/data/project2/labels/X_train.txt"

   //s3 path for file having file names of training docs

   nameTestPath="s3n://eds-uga-csci8360/data/project2/labels/X_test.txt"

   //s3 path for file having file names of test docs

   classPath = "s3n://eds-uga-csci8360/data/project2/labels/y_train.txt"

   //s3 path for file having labels/classes of training docs

   aws_id = “xxxx”

   //your aws id

   aws_key = “xxxx”

   //your aws key

Parameters like driver memory, executor-memory, total-executor-cores and spark.driver.maxResultSize can be changed based on cluster.

This project has been tested on AWS r3.4xlarge instances in a cluster of 2. With the above parameters it took approximately 38 min time to complete and gave accuracy of 95%.

Sample command:

```
./bin/spark-submit --master spark://172.19.72.238:7077 --driver-memory 2g --executor-memory 3g --total-executor-cores 20 --conf spark.driver.maxResultSize=8g  --packages org.apache.hadoop:hadoop-aws:2.7.1 /home/ec2-user/invasion/random2.py s3n://eds-uga-csci8360/data/project2/binaries s3n://eds-uga-csci8360/data/project2/labels/X_train.txt s3n://eds-uga-csci8360/data/project2/labels/X_test.txt s3n://eds-uga-csci8360/data/project2/labels/y_train.txt aws_id aws_key
```

CLASSIFICATION RATIONALE AND DESIGN:

CLEANING DATA (BYTE FILES):

- Used only byte files for classification.
- Removed the numbers at the beginning of the line and /r/n from the byte files in the clean() function.

FEATURES USED:

- Term frequency of the two digit hexadecimal values and “??” in bytes file as the feature.
- Kept “??” as removing it reduced accuracy from 95 to 94.8%.
- Each hexadecimal digit can take 16 values each, hence 256 total possible values and therefore passed 265 to HashingTF() function as number of features.

MODELS APPLIED:

To get a feel of the data, I earlier applied Naive Bayes classifier but it didn’t give a good accuracy on the small data set. Hence, I decided to use Random forests, which resulted in good accuracy. I tweaked the number of trees and depth to get better accuracy and heuristically determined that 50 trees with max depth 8 using gini impurity give the best accuracy.

APPROPRIATENESS OF CLASSIFIER AND ADVANTAGES OVER NAÏVE BAYES APPROACH:

Naïve bayes approach works well for small training set as it is a high bias and low variance classifier. Though it is fast and does not overfit, it gives a very simple hypothesis that does not represent complex behavior.

Random forests, in contrast, is a bagging technique which uses ensembles of decision trees for prediction. It trains each tree independently, using a random sample of the data and hence model is more robust than a single decision tree, and less likely to overfit on the training data. It is a low bias estimator and can represent complex hypothesis. In my experience, random forest has lower classification error and better f-scores than naïve bayes (Based on yelp dataset challenge I did last semester for machine learning class, in which random forest performed better than single classifiers like KNN,Naïve Bayes,SVM) and based on the following paper: http://bid.berkeley.edu/cs294-1-spring13/images/b/b5/CS_294_Final_Project_Presentation-ABN.pdf

In addition, random forests are fast and scalable and we do not need to tune many parameters as opposed to other classifiers.

The MLlib implementation of random forests is fast in terms of both memory and communication as:

Random Forests utilize different subsample of the data to train each tree. Rather than replicating data explicitly, it utilizes TreePoint structure to save memory. This structure stores the number of replicas of each instance in each subsample. In addition, feature subsampling reduces communication between different nodes.

[Source: https://databricks.com/blog/2015/01/21/random-forests-and-boosting-in-mllib.html]

APPENDIX:

Installing anaconda2:

1. Get the anaconda installation.
   wget https://3230d63b5fc54e62148e-c95ac804525aac4b6dba79b00b39d1d3.ssl.cf1.rackcdn.com/Anaconda2-2.5.0-Linux-x86_64.sh

2. Change permissions to install anaconda script file.
   chmod a+x Anaconda2-2.5.0-Linux-x86_64.sh

3. Run the anaconda installation script
   ./Anaconda2-2.5.0-Linux-x86_64.sh

4. Add anaconda installation to path.
   export PATH = <path of anaconda2 installation>/bin:$PATH"