Malware classification through Spark

Project No.2 lightning talk
Group: ChickenBurger

Mojtaba Sedigh-Fazli; BahaaEddin AlAila; Shawn(Shengming) Zhang

Introduce to this project:

1) rooted from the Microsoft Malware Classification Challenge on Kaggle 2015
2) The main task is to classify around 9000 malwares into 9 classes
3) The big difficulties are laid on the large size of the malwares, 200GB
4) Scala spark code has written to solve this project through AMAZON EC2 service
5) Sbt is used for build and package our Scala codes.
6) We hooked up the EC2 manually since Flintrock has security group issues

Outlines:

1) Data Loading
2) Features Engineering
3) Training on Random Forest, Gradient Boosted trees and SVMs
4) Testing
5) Potential improvement and lesson learned

Data Loading

Nasty bug:

```
16/09/21 09:35:17 WARN TaskSetManager: Lost task 0.0 in stage 7.2 (TID 27, 172.19.21.188):
org.apache.spark.shuffle.FetchFailedException: Too large frame: 7069349343
```

Yo Mama's cell number

Data too large bug → **Remote Shuffle Blocks cannot be larger than 2 GB**

Details: https://issues.apache.org/jira/browse/SPARK-5928

Only triggered for remote fetches, not local mode, took us a week to pinpoint

(Thx, lazy execution!)

Once you know the problem, few changes to fix it :

Data Loading -cont'd

Solution

```
val training_set:RDD[(String,Iterable[(String,Double)])] = filesWithPaths.grouped(limit).map(
                    filesAtaTime => {
sc.wholeTextFiles(filesAtaTime.mkString(",")) // path/to/file1.asm,path/to/file2.asm,...,path/to/fileN.asm
//(path,content)
.map({ case (path, content) => (path.split("/").last.split("\\.")(0), processingFunc(content))}).persist(StorageLevel.MEMORY_AND_D

}).reduce((a:RDD[(String,Iterable[(String,Double)])], b:RDD[(String,Iterable[(String,Double)])]) => a union b)
```

Load and process 800 files at a time (Matrix Sketching - Liberty), union all

Feature Engineering:

- ASM files:
  - the segCount: which are the # lines of each section (e.g.: text, idata, rdata, data)
  - opcodes Ngrams: which are the 1gram, 2gram, 3gram and 4 grams of the opcode (e.g.: “push-mov”-> 45)   (97% accuracy)
- Bytes files:
  - hexadecimal Ngrams: which are the 1gram, 2gram, 3gram and 4 grams of the bytestream
    - Fit all 4 grams into one long, lossless but still gentler on reducebykey, eg:
      - 0x10000000000000AC
      - 0x20000000000000AC
      - 0x40000321000000AC
      - 100 for “??”
  - File size : # of bytes (feature 0x000000000000000)

Feature Engineering - cont’d

- Discard features appearing less than 5 times per file
- Discard features appearing less than 20 times in all files
- vectorize, .compressed
- Extract once, serialize, refrigerate
- Load from disk and deserialize

Training on Random Forest and Gradient Boosted Trees

1) Random Forest:
   Parameter sweep on, 5 fold cross validation. (900 combinations)
   Our best 98.2% (sadly did not save the model). And couldn't reproduce
   because RF randomly sample features.

2) Gradient Boosted Trees and LSVM's:
   Spark's mllib, One vs Rest, with RF meta stacker:
   Did not give good results(distribution skewed) =>
   implemented resampling to equalize the class distribution.
   Took forever way past the deadline, and ran out of memory.
   (should've resampled hashes only, then joined)

Potential improvement and lesson learned

1) Only retain features that showed good class correlation
    a) Look into RF scores for features
2) Extract structure from opcodes (model for loops ...etc), regardless of n-grams. (LSTM ?)
3) Write the hex file into binary get McAfee to detect it :p

Questions? and thank you!