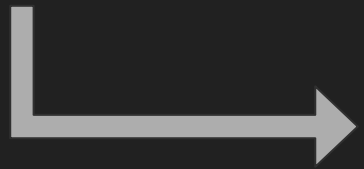
Elizabeth
DSP P2 Lightning Talk

Chris Barrick
Zach Jones
Jeremy Shi

Initial Troubles

Initial Troubles

- Started out using `spark.mllib` with RDDs
- Quickly realized DataFrames were the way to go
  - Exposed better API
  - Better theoretical performance
- Started with User-Defined Functions (UDFs) for Tokenization



```
@elizabeth.udf([str])
def split_bytes(text):
    '''Splits the text of a bytes file into a list of bytes.
    '''
    bytes = []
    for line in text.split('\n'):
        try:
            (addr, *vals) = line.split()
            bytes += vals
        except ValueError:
            # ValueError occurs on invalid hex,
            # e.g. the missing byte symbol '??'
            continue
    return bytes


@elizabeth.udf([str])
def split_asm(text):
    '''Splits the text of an asm file into a list of opcodes.
    '''
    opcodes = []
    pattern = re.compile(r'\.([a-z]+):([0-9A-F]+)'
        r'((?:\s[0-9A-F]{2})+)\s+([a-z]+)(?:\s+([^;]*))?')
    for line in text.split('\n'):
        match = pattern.match(line)
        if match:
            opcode = match[4]
            opcodes.append(opcode)
return opcodes
```

Artist’s depiction of memory usage with UDFs

Initial Troubles

- Explanation:
  - UDFs in Python can’t be implemented by a simple java API call
  - DataFrame has to be serialized to Python worker, processed, then deserialized back to the JVM
  - Python worker consumes huge off-heap memory during the serialization-work-deserialization process
- Solution: try not to use UDFs
- Switched to `spark.ml.feature` for all feature extraction tasks and never looked back!

Methodology

Classifiers

- Naive Bayes model did okay-ish using TFIDF scores of single-byte tokens (74%)
- Naive Bayes gave better result with 2-gram and 4-gram term frequencies (89.56%)
- From previous results, GBTs seemed like the way to go
  - However, spark.ml GBTs are binary only <sad face>
  - Wait, no problem! Just build an ensemble of OneVsRest binary GBTs
  - Nevermind - turns out GBTs don't output raw probabilities due to an oversight during development (fix coming in Spark 2.3)
- Instead tried Random Forest
  - Got promising result using 2-gram and 4-gram term frequencies from binary files (98.8%)
  - Question: what features can we add to the RF classifier to improve?

Feature Engineering

- Further improvement using 2-gram and 4-gram byte frequencies: 98.87%
- Potential for more improvement: Incorporate .asm features
- Built feature extractor that got the length of each asm segment (HEADER, .data, .rsrc, etc) as well as opcode counts
- Created joint feature set that included binary term frequencies as well as .asm features
- Surprisingly, the joint feature set performed slightly worse! (98.01%)

Final Push

- Had the output of a few different classifiers which achieved > 98% accuracy
- Maybe they make a few uncorrelated errors?
- Built simple voting emsemble using the output files
  (no time/resources to train a brand-new ensemble)
- Resulted in a few more correct classifications -> boosted score to 99.04%

Thanks!