BUILDING A MACHINE LEARNING APPLICATION WITH AWS LAMBDA

Ludi Rehak

ludi@h2o.ai

Silicon Valley Big Data Science Meetup

March 17, 2016

(+ help from Tom and Prithvi)

BUILDING A MACHINE LEARNING APPLICATION WITH AWS LAMBDA

Q: What is AWS Lambda?

A: AWS Lambda is a compute service that runs code – a ***Lambda function*** - on-demand. It simplifies the process of running code in the cloud by managing compute resources automatically.

Offloads DevOps tasks related to VMs:

- Server and operating system maintenance
- Capacity provisioning
- Scaling
- Code monitoring and logging
- Security patches

MAJOR STEPS

Step 1: Identify problem to solve

Step 2: Train model on data

Step 3: Export the model as a POJO

Step 4: Write code for Lambda handler

Step 5: Build deployment package (.zip file) and upload to Lambda

Step 6: Map API endpoint to Lambda function

Step 7: Embed endpoint in application

A CONCRETE USE CASE: DOMAIN NAME CLASSIFICATION

Malicious domains

- Carry out malicious activity - botnets, phishing, malware hosting, etc
- Names are generated by algorithms to defeat security systems

Goal: Classify domains as legitimate vs. malicious

Legitimate	Malicious
h2o	zyxgifnjobqhzptuodmzov
zen-cart	c3p4j7zdxexg1f2tuzk117wyzn
fedoraforum	batdtrbtrikw

FEATURES

- String length
- Shannon Entropy
  - Measure of uncertainty in a random variable

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

- Number of substrings that are English words
- Proportion of vowels

DATA

- Domains and whether they are malicious
  - http://datadrivensecurity.info/blog/data/2014/10/legit-dga_domains.csv.zip
  - 133,927 rows
- English words
  - https://raw.githubusercontent.com/dwyl/english-words/master/words.txt
  - 354,985 rows

MODEL INFORMATION

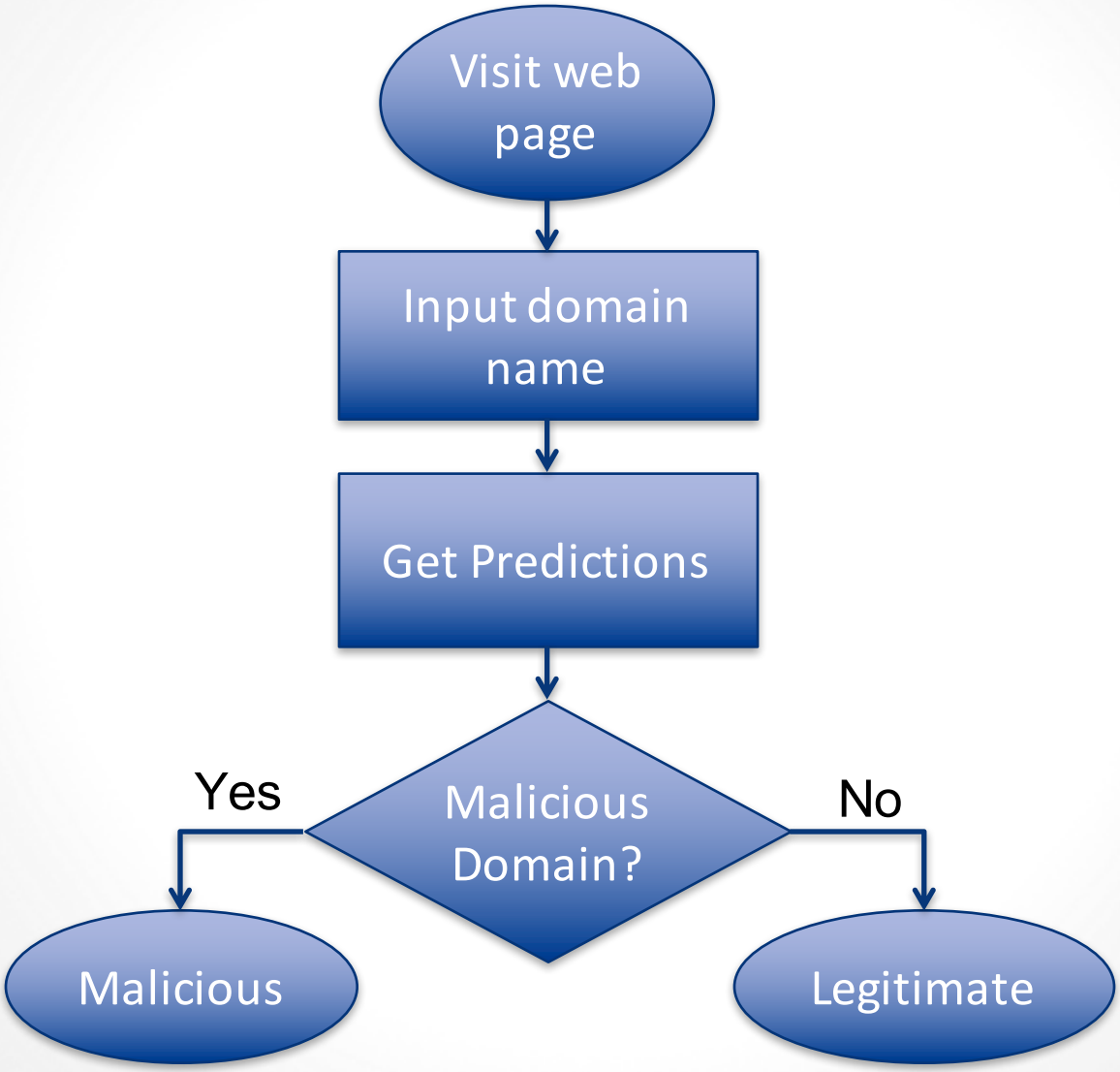
Malicious Domain Model

Algorithm:	GLM
Model family:	Binomial
Regularization:	Ridge
Threshold (max F1):	0.4935

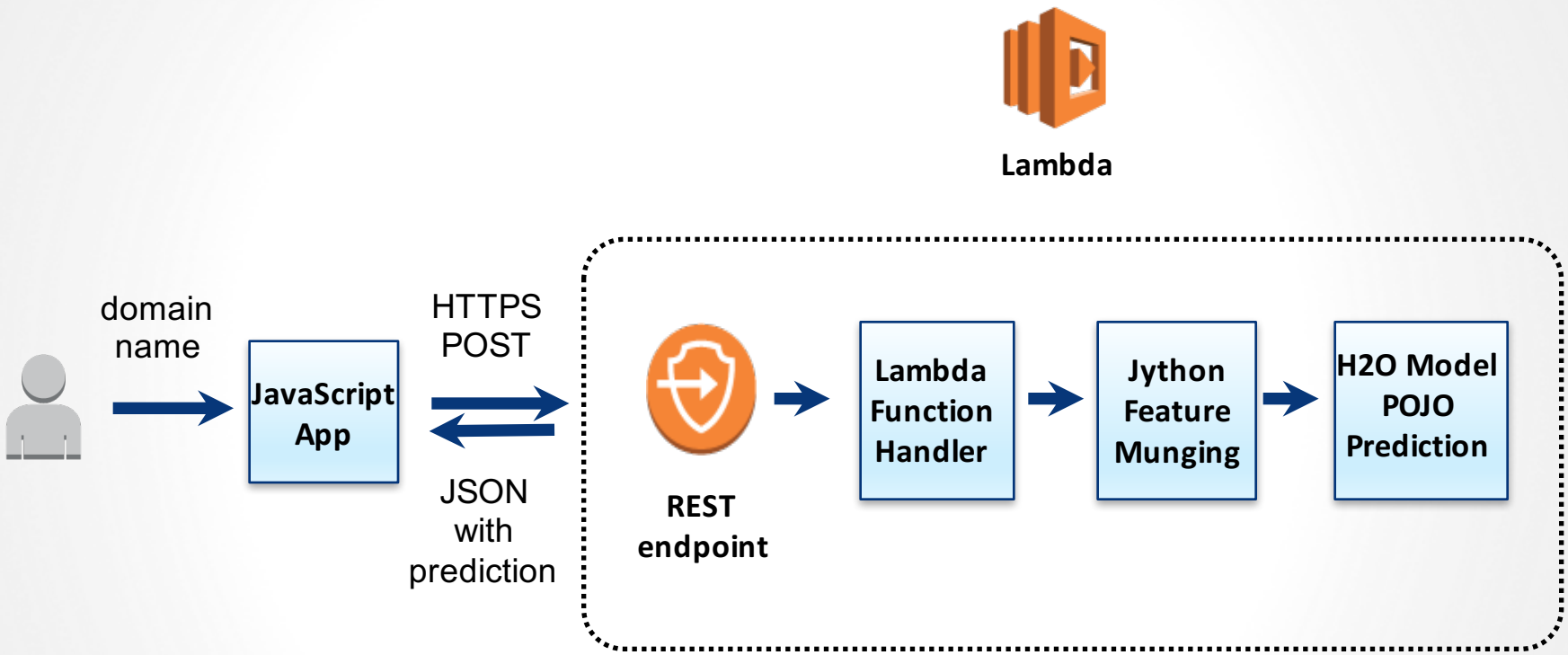
Confusion matrix on validation data

		Predicted		
Actual	Class	0	1	Error
	0	15889	315	FPR 0.0194
	1	346	10043	FNR 0.0333

WORKFLOW FOR THIS APP

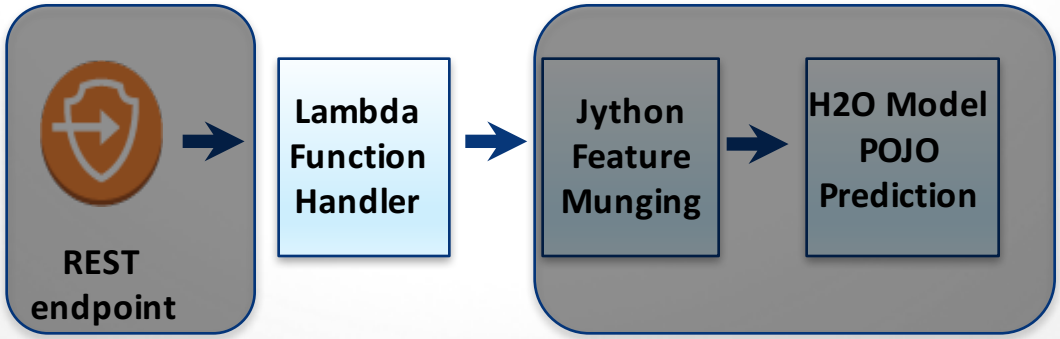


APP ARCHITECTURE DIAGRAM



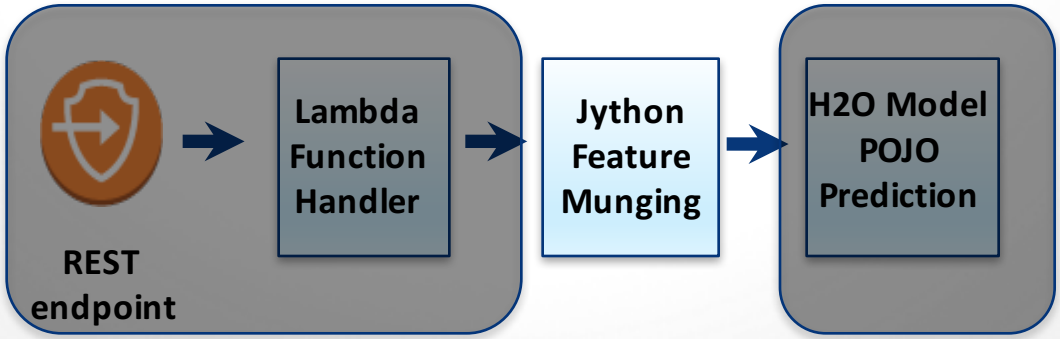
LAMBDA FUNCTION HANDLER

```
public static ResponseClass myHandler(RequestClass
request, Context context) throws PyException {

  PyModule module = new PyModule();

  //Prediction code is in pymodule.py
  double[]predictions=module.predict(request.domain);
  return new ResponseClass(predictions);
}
```



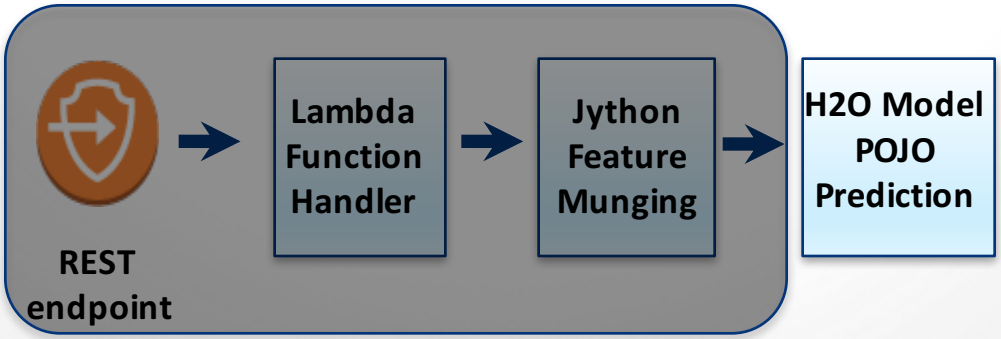
JYTHON FEATURE MUNGING

```
def predict(domain):
  domain = domain.split('.')[0]
  row = RowData()
  functions = [len, entropy, p_vowels, num_valid_substrings]
  eval_features = [f(domain) for f in functions]
  names = NamesHolder_MaliciousDomainModel().VALUES
  beta = MaliciousDomainModel().BETA().VALUES
  feature_coef_product = [beta[len(beta) - 1]]
  for i in range(len(names)):
    row.put(names[i], float(eval_features[i]))
    feature_coef_product.append(eval_features[i] * beta[i])

  #prediction
  model = EasyPredictModelWrapper(MaliciousDomainModel())
  p = model.predictBinomial(row)
```



H2O MODEL POJO

```
static final class BETA_0 implements java.io.Serializable {
  static final void fill(double[] sa) {
    sa[0] = 1.49207826021648;
    sa[1] = 2.8502716978560194;
    sa[2] = -8.839804567200542;
    sa[3] = -0.797706503462465 5;
    sa[4] = -14.94132841574946;
  }
}
```



HANDS-ON DEMONSTRATION

STEP 1: Build
$ git clone https://github.com/h2oai/app-malicious-domains
$ cd app-consumer-loan
$ gradle wrapper
$ ./gradlew build

STEP 2: Create Lambda function and set API endpoint
See instructions and screenshots in README.md

STEP 3: Use the app in a web browser
$ ./gradlew jettyRunWar –x generateModel
http://localhost:8080

TROUBLESHOOTING

- Common Py errors
  - Another H2O is already running
    - Py script can't find the data in h2o.import_file()
- Common Java errors
  - Java not installed at all
    - Also, must install a JDK (Java Development Kit) so that the Java compiler is available (JRE is not sufficient)
  - Not connected to the internet
    - Gradle needs to fetch some dependencies from the internet
- Common Lambda errors
  - Error in uploading .zip file
    - Check if the function already exists and, if not, try again. For slower internet connections, try uploading .zip file with S3 link.
  - Timeout error when testing Lambda function
    - Go to advanced settings and increase Timeout value
  - Gateway Timeout (504 error)
    - This is Lambda's cold start behavior. Keep trying, eventually Lambda kicks in

CAVEATS

- Stateless
  - Can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.
- Cold start behavior
  - containers are instantiated and reused after the first request and stay active for a window of time (10-20 minutes)
  - “the longer I leave it between invocations, the longer the function takes to warm up”
- API Gateway timeout of 10 secs
  - Can request longer timeout

CONFIGURING LAMBDA FUNCTIONS

- Memory
  - Allocates proportional CPU power, network bandwidth, and disk I/O
  - Easy single-dial solution
  - Log shows how much memory was used for tuning and cost savings
- Timeout

LAMBDA RESOURCE LIMITS

Resource	Default Limit
Memory	512 MB
Number of file descriptors	1,024
Number of processes and threads (combined total)	1,024
Maximum execution duration per request	300 seconds
Invoke request body payload size	6 MB
Invoke response body payload size	6 MB
Concurrent executions per region	100

Item	Default Limit
Lambda function deployment package size (.zip/.jar file)	50 MB
Size of code/dependencies that you can zip into a deployment package (uncompressed zip/jar size)	250 MB

LAMBDA PRICING

- Lambda
  - Requests
    - First 1 million per month are free
    - $0.20 per 1 million requests thereafter
  - Duration
    - First 400,000 GB-seconds of compute time per month are free
    - $0.00001667 for every GB-second thereafter
- API Gateway
  - $3.50 per million API calls received plus data transfer costs
- Estimate for Malicious Domain Application:
  - Lambda: $0.37/hour with 10 threads after free-tier
  - API Gateway: $0.71/hour
  - Total: ~$1/hr

LAMBDA PERFORMANCE

Memory (MB)	Threads	Loops	Samples	Median (ms)	Min (ms)	Max (ms)	% Error	Throughput (calls/sec)
512	1	10000	10000	102	85	2137	0	8.4
512	10	1000	10000	102	85	30330	0.18	44
512	100	100	10000	149	85	30307	0.43	168

LAMBDA SCALING

- Automatically scales to support the rate of incoming requests
- “No limit to the number of requests your code can handle”
- Starts as many instances of Lambda function as needed

RELATED EXAMPLES

- H2O Generated Model POJO in a Java Servlet container
  - Github: h2oai/app-consumer-loan
- H2O Generated Model POJO in a Storm bolt
  - GitHub: h2oai/h2o-world-2015-training
  - tutorials/streaming/storm
- H2O Generated Model POJO in Spark Streaming
  - GitHub: h2oai/sparkling-water
  - examples/src/main/scala/org/apache/spark/examples/h2o/CraigslistJobTitlesStreamingApp.scala

RESOURCES ON THE WEB

- Slides
  - GitHub h2oai/h2o-tutorials/tree/master/tutorials/aws-lambda-app
- Source code
  - GitHub h2oai/app-malicious-domains
- Latest stable H2O for Python release
  - http://h2o.ai/download/h2o/python
- Generated POJO model Javadoc
  - http://h2o-release.s3.amazonaws.com/h2o/rel-turan/3/docs-website/h2o-genmodel/javadoc/index.html
- AWS Lambda
  - http://docs.aws.amazon.com/lambda/latest/dg/welcome.html

Q & A

- Thanks for attending!

- Send follow up questions to:

Ludi Rehak

ludi@h2o.ai