sqrrl

Getting Started with Machine Learning for Incident Detection

August 2016 | **Target. Hunt. Disrupt.**

Chris McCubbin, Director of Data Science, Sqrrl
David J. Bianco, Security Technologist, Sqrrl

A story we all know: Regular expressions

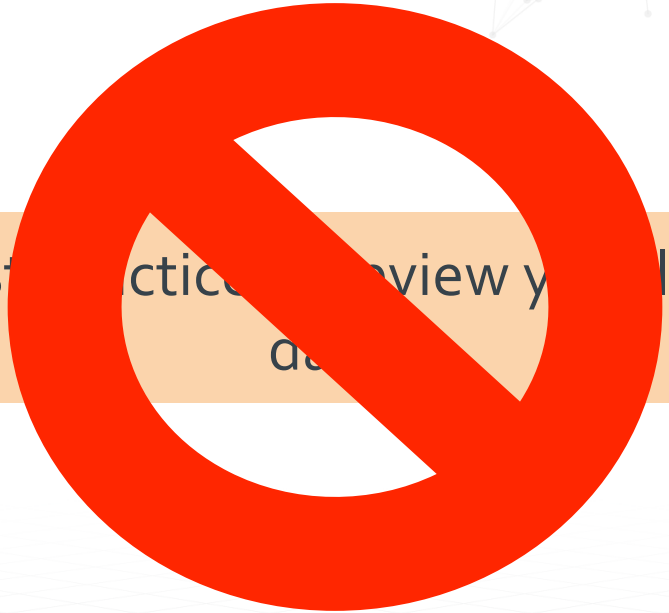
sqrrl

- "Good theory leads to good programs"
- Who here has implemented and optimized a Nondeterministic Finite Automata compiler?
- You probably use one every day
  - Regex: Grep, perl
- You don't care how it works inside
  - But you might need to know some quirks
    - Regex can't count (google up "regex HTML" on stackoverflow)
    - Grep has no 'bad cases'
    - Perl is more powerful (lazy, backreferences)
- But it is helpful to know what it's good for, how to use it, etc.

Agenda

- What is Machine Learning (ML) good at?
- ~~How does ML work?~~ What are the quirks of useful Machine Learning techniques?
- Can I use Machine Learning easily?
- How can you customize & improve our examples?

When's the last time you heard…?



"It's a Best ctice view y logs every
da

Machine-Assisted Analysis

Practical Cyborgism for Security Operations

COMPUTERS

- Bad at context and understanding
- Good at repetition and drudgery
- Algorithms work cheap!

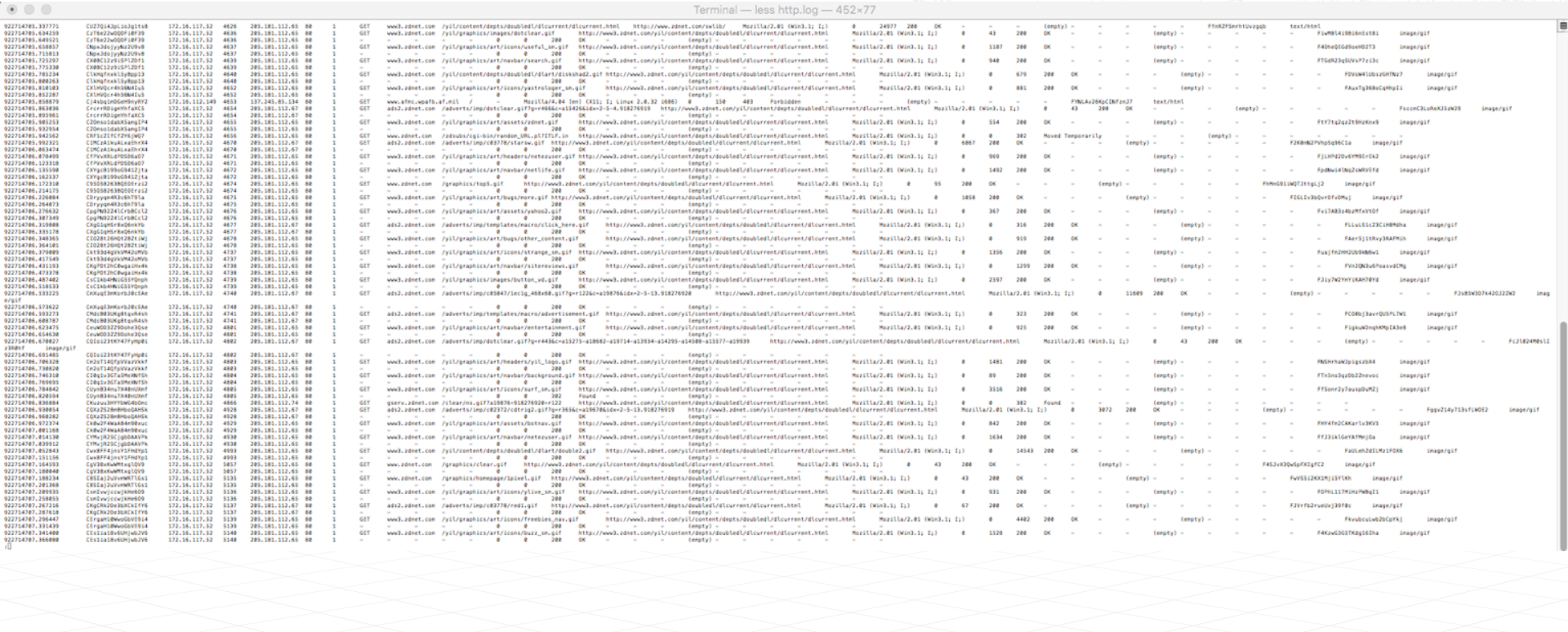
PEOPLE

- Contextual analysis experts who love patterns
- Possess curiosity & intuition
- Business knowledge

EMPOWERED ANALYSTS

- Good results from massive amounts of data
- Agile investigations
- Quickly turn questions into insight

Problem Statement: HTTP Proxy Logs

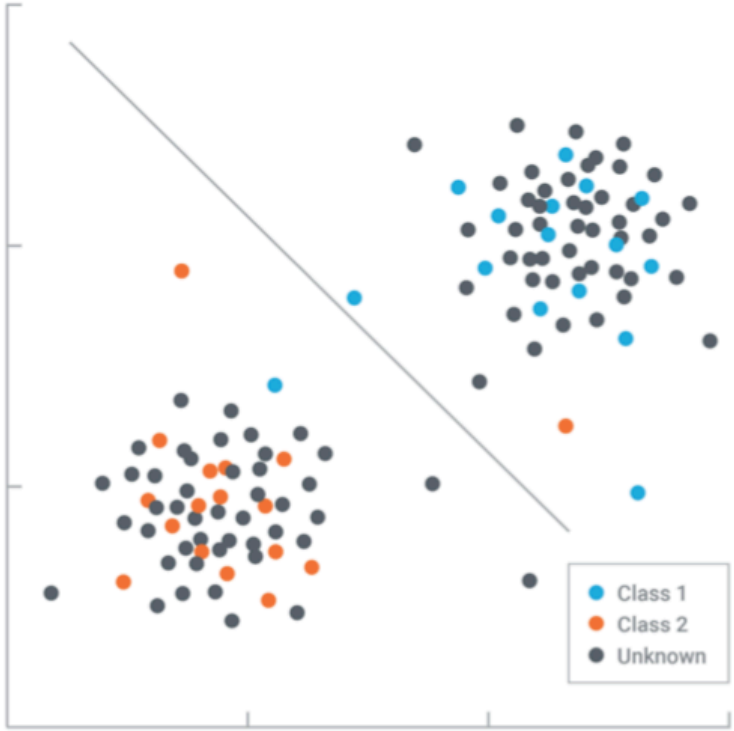


Our solution: Clearcut!

Two different types of machine learning

- **Supervised**
  - Have labeled training data?
  - Classification algorithms
    - Random Forests
- **Unsupervised**
  - No labeled training data
  - Assume attacks are rare
  - Outlier Detection
    - Isolation Forests
  - Clustering

Supervised: Binary Classification



Given a population of two types of "things", can I find a function that separates them into two classes?

Maybe it's a line, maybe it's not.

Nothing's perfect, but how close can we get?

If we derive a function that does reasonably well at separating the two classes, that's our binary classifier!

Fortunately, Python has pantsloads of libraries that can do this for us. The machine can learn the function given enough samples of each class.

Classification With Random Forests

sqrrl

1. Identify positive and negative sample datasets
2. Clean & normalize the data
3. Partition the data into training & testing datasets
4. Select & compute some interesting features
5. Train a model
6. Test the model
7. Evaluate the results
8. 🍺

Generating synthetic abnormal data

Perhaps we don't have any malware data, but we have normal data.

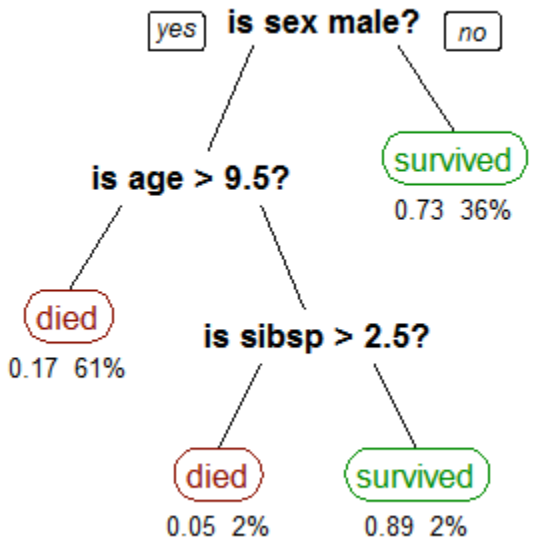
If we could make some synthetic abnormal data, we could still use the same methods

One-class classification

How should we create the data?

One option: 'Noise-contrastive estimation': Generate noise data that looks real-ish, but has no real structure and contrast that to the normal data

Decision Trees



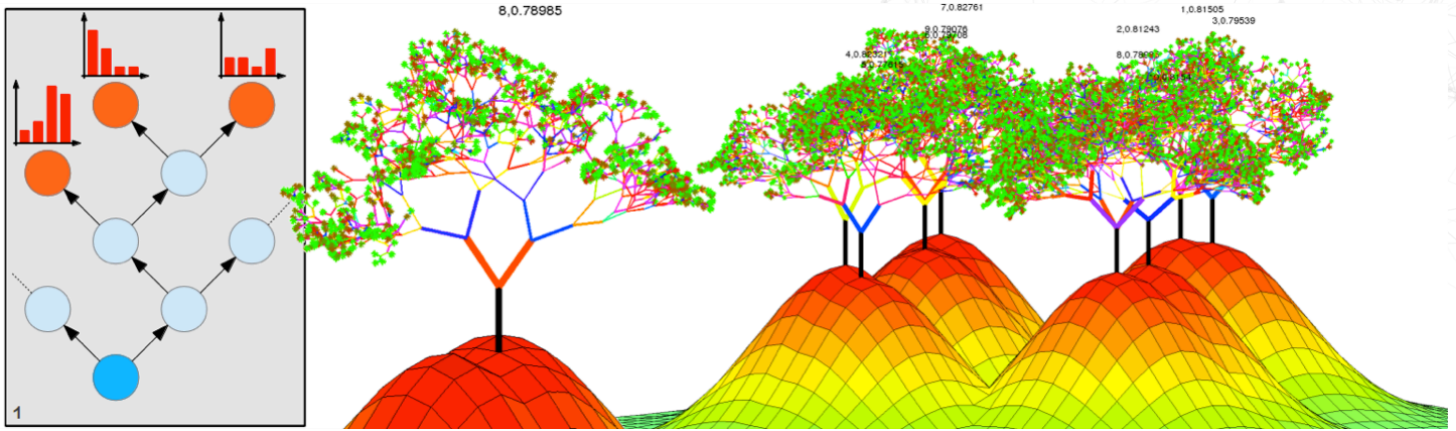
Greedily grow tree by choosing feature that explains the class the most

Split the training set into two sets, repeat

Form a classifier by "walking down the tree"

Issue: overfitting

Random Forests



http://www.rhaensch.de/vrf.html

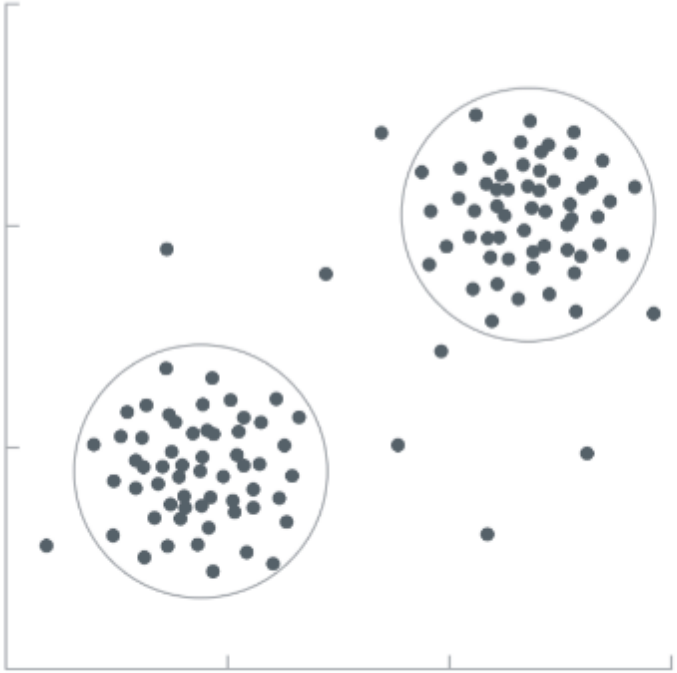
Sample training set with replacement

Fit a decision tree to the sample

Repeat n times

Form a classifier by averaging the n decision trees

Unsupervised: Outlier Detection

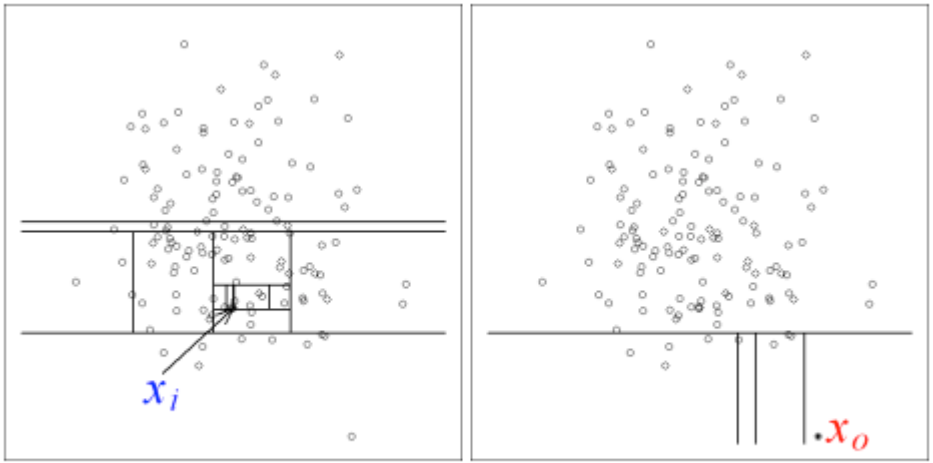


Given a population of "things", can I find a function that tells me which ones look weird?

Can also pretend to be a classifier (class 0 = normal, class 1 = weird)

Loads of ways to accomplish this: distance to your neighbors, angle-based methods, isolation-based methods

Isolation Forests [Liu, Ting, Zhao]



(a) Isolating x_i (b) Isolating x_o

Pick a dimension at random. Pick a value at random.

Make a tree by splitting the set into two sets, repeat. Stop when the set is a single point.

Do this for many trees.

Form an outlier detector by the average depth that a point is isolated in each tree (deeper is more inlier-y)

Issue: enumerated types

http://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdmo8b.pdf

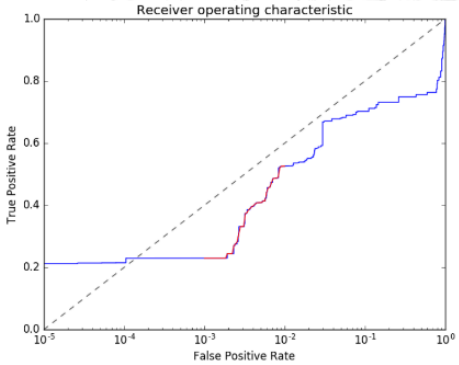
A quick note about parameters

Choosing parameters can be important

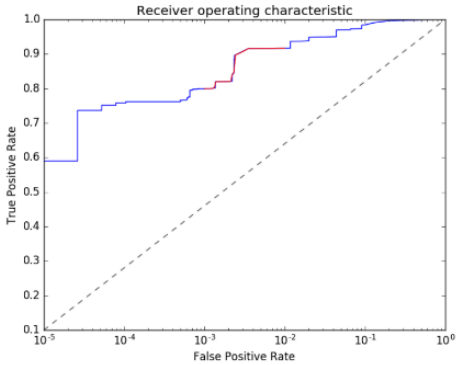
Can use expert knowledge or ad-hoc methods

Dimitar Karev (MIT RSI Intern) tested a range of parameters for Clearcut iforests using exhaustive search (for forest params) and a genetic algorithm (for features)

Result was a huge improvement in F1 (see ROC curves)



(a) The initial parameters.



(b) The modified parameters.

Figure 6: The ROC Curve Produced by the Model under Different Settings of Parameters.

Classification With Isolation Forests

sqrrl

1. Identify positive and negative sample datasets
2. Clean & normalize the data
3. Partition the data into training & testing datasets
4. Select & compute some interesting features
5. Train a model
6. Test the model
7. Evaluate the results
8. 🍺
9. Notice similarities

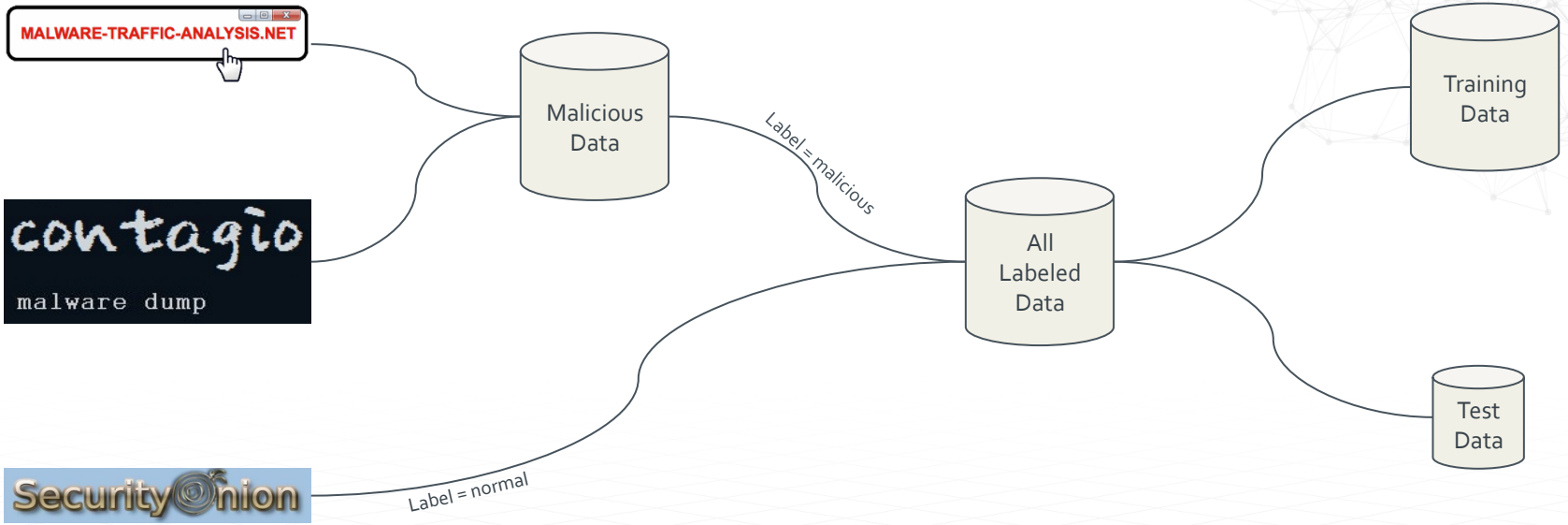
The beauty of scikit-learn & python

- **Gists to perform many types are learning are simple and consistent**
  - Take same data as input (supervised requires an extra column)
  - Signatures of methods are the same
- **Example: RF's vs iForests**
  - Changed a few lines of code for training
  - Classes are a bit different (0/1 vs 1/-1)
  - Can re-use the analysis script with nearly no change

```
#RF
clf = RandomForestClassifier(n_jobs=4,
        n_estimators=opts.numtrees, oob_score=True)
y, _ = pd.factorize(train['class'])

clf.fit(train.drop('class', axis=1), y)
test['prediction'] = clf.predict(testnoclass)
```

```
#iF
clf = IsolationForest(n_estimators=opts.numtrees)



clf.fit(train.drop('class', axis=1))
test['prediction'] = clf.predict(testnoclass)
```

Identifying Training & Test Data



Feature extraction

Many classifiers want to work with numeric features. We use a 'flow enhancing' step to add some convenience columns to the data

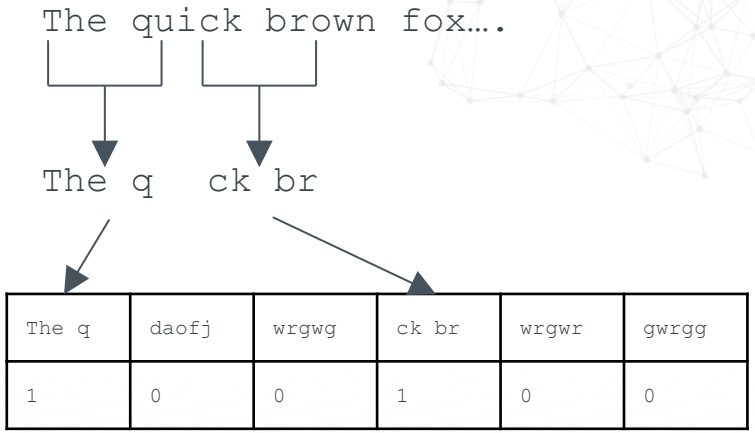
Some columns are already numeric

Some columns have easy-to-extract numeric info: number of dots in URL, entropy in TLD

Categorical columns can be converted to "Bag of words" (BOW): N binary features, one for each category

Text-y columns can be converted to BOW or Bag-of-Ngrams (BON)

Use TF-IDF to determine which features to keep



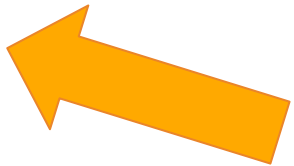
The q	daofj	wrgwg	ck br	wrgwr	gwrgg
1	0	0	1	0	0

Training, Testing & Evaluating a Model

```
% ./train_flows_rf.py -o data/http-malware.log http-training.log
Reading normal training data
Reading malicious training data

Building Vectorizers

Training

Predicting (class 0 is normal, class 1 is malicious)

class  prediction
0      0             12428
       1                15
1      0                19
       1              9563
dtype: int64
F1 = 0.998225469729
```

Training, Testing & Evaluating a Model

```
% ./train_flows_rf.py -o data/http-malware.log http-training.log
Reading normal training data
Reading malicious training data

Building Vectorizers

Training

Predicting (class 0 is normal, class 1 is malicious)

class  prediction
0      0             12428
       1                15
1      0                19
       1              9563
dtype: int64
F1 = 0.998225469729
```

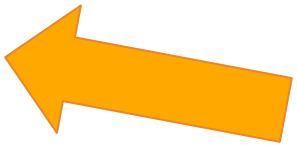


Read the Bro data files into a Pandas data frame.

Each row is labeled either ‘benign’ or ‘malicious’.

Training, Testing & Evaluating a Model

```
% ./train_flows_rf.py -o data/http-malware.log http-training.log
Reading normal training data
Reading malicious training data

Building Vectorizers

Training

Predicting (class 0 is normal, class 1 is malicious)

class  prediction
0      0             12428
       1                15
1      0                19
       1              9563
dtype: int64
F1 = 0.998225469729
```



Random Forest requires numeric data, so we have to convert strings.

Primarily two methods:

- Bag of Words (method, status code)
- Bag of N-Grams (domain, user agent)

Training, Testing & Evaluating a Model

```
% ./train_flows_rf.py -o data/http-malware.log http-training.log
Reading normal training data
Reading malicious training data

Building Vectorizers

Training

Predicting (class 0 is normal, class 1 is malicious)

class  prediction
0      0             12428
       1                15
1      0                19
       1              9563
dtype: int64
F1 = 0.998225469729
```

Split all the labeled data into 'training' (80%) and 'test' (20%) datasets.

Now feed all the training data through the Random Forest to produce a trained model.

At this point, we do nothing with the test data.

Training, Testing & Evaluating a Model

```
% ./train_flows_rf.py -o data/http-malware.log http-training.log
Reading normal training data
Reading malicious training data

Building Vectorizers

Training

Predicting (class 0 is normal, class 1 is malicious)

class  prediction
0      0             12428
       1                15
1      0                19
       1              9563
dtype: int64
F1 = 0.998225469729
```

Now we run the 'test' data through the trained model. It's still labeled, so we know what the answer ***should*** be.

We compare the expected results with the actual prediction and create a little table.

We don't expect perfect results, but we'd like to see ***most*** of the data in the 0/0 and 1/1 rows.

Training, Testing & Evaluating a Model

```
% ./train_flows_rf.py -o data/http-malware.log http-training.log
Reading normal training data
Reading malicious training data

Building Vectorizers

Training

Predicting (class 0 is normal, class 1 is malicious)

class  prediction
0      0             12428
       1                15
1      0                19
       1              9563
dtype: int64
F1 = 0.998225469729
```

It's hard to compare two tables to see how different models compare (due to different datasets or feature choices).

The F1 value is a useful single-number measure for comparison, combining TP & FP rates.

Anything over about 0.9 is considered good, but ***beware*** very high values ("overfitting")!

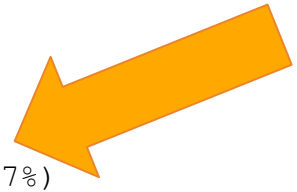
Bonus: Most Influential Features with '-v'

```
Feature ranking:
1. feature user_agent.mac os  (0.047058)
2. feature user_agent. os x 1 (0.044084)
3. feature user_agent.; intel (0.042387)
4. feature user_agent.ac os x (0.037192)
5. feature user_agent.os x 10 (0.031616)
[...]
46. feature userAgentEntropy (0.009144)
47. feature subdomainEntropy (0.007699)
48. feature browser_string.browser (0.007263)
49. feature response_body_len (0.006410)
50. feature request_body_len (0.005506)
51. feature domainNameDots (0.005054)
```

Analyzing Log Files

sqrrl

Percentage of original file left to review.



```
% ./analyze_flows.py http-production-2016-05-02.log
Loading HTTP data
Loading trained model
Calculating features

Analyzing
detected 298 anomalies out of 180520 total rows (0.17%)
----------------------------------------
line  2393
Co7qtw35sGLX6RiG79,80,HEAD,download.virtualbox.org,/virtualbox/5.0.20/
Oracle_VM_VirtualBox_Extension_Pack-5.0.20.vbox-extpack,-,Mozilla/5.0 (AgnosticOS;
Blend) IPRT/64.42,0,0,200,80,Unknown Browser,,,download,virtualbox

----------------------------------------
line  2394
ChpL1u2Ia64utWrd9j,80,GET,download.virtualbox.org,/virtualbox/5.0.20/
Oracle_VM_VirtualBox_Extension_Pack-5.0.20.vbox-extpack,-,Mozilla/5.0 (AgnosticOS;
Blend) IPRT/64.42,0,16421439,200,80,Unknown Browser,,,download,virtualbox
```

Bonus: Classifier Explanations with '-v'

```
line  431
C9WQArVvgv1BjvJG7,80,GET,apt.spideroak.com,/spideroak_one_rpm/stable/
repodata/repomd.xml,-,PackageKit-hawkey,0,2969,200,80,Unknown
Browser,,,apt,spideroak

Top feature contributions to class 1:
   userAgentLength 0.0831734141875
   response_body_len 0.0719766424091
   domainNameLength 0.056790435921
   user_agent.mac os  0.0272829846513
   user_agent. os x 1 0.0252803447682
   user_agent.os x 10 0.0251306287983
   user_agent.ac os x 0.0244848247673
   user_agent.; intel 0.0241743906069
   user_agent. intel  0.0236921809876
   tld.apple 0.020090459858
```

Ideas for improvement

More diverse malware samples

Better filtering for connectivity checks in the malware data

Incrementally retraining the forest ('warm start')

Log type "plugins"

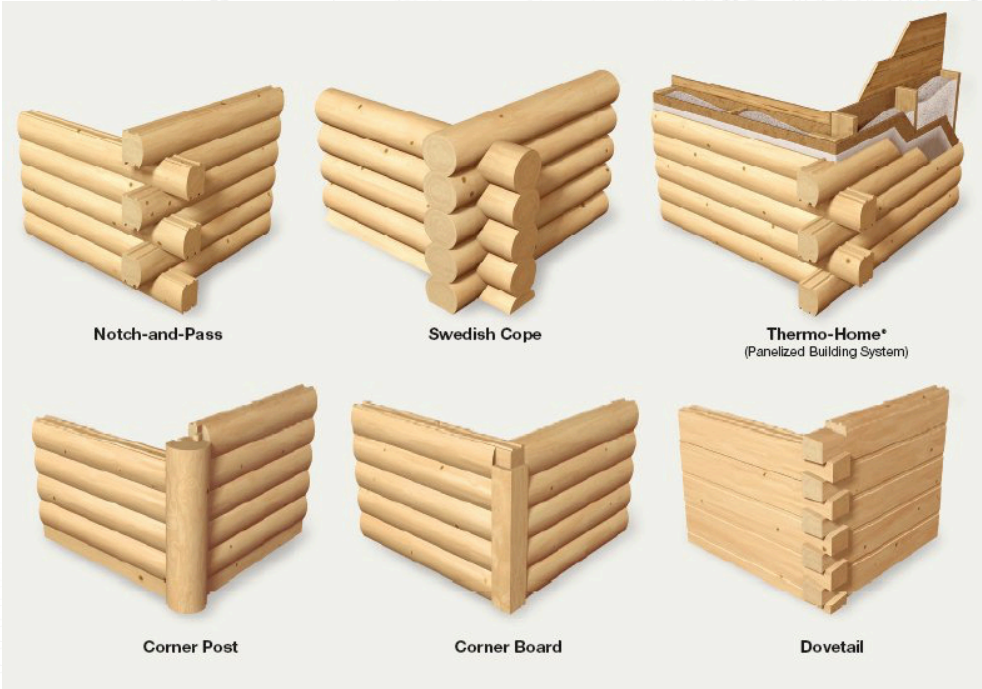
K-class classifier

Adapting to other log sources

Change log input: clearcut_utils.load_brofile
Import your data into a pandas data frame

Change flow enhancer: flowenhancer.enhance_flow
Add any columns that might make featurizing easier

Change feature generator:
featurizer.build_vectorizers
Make any BOW and BON vectorizers that you want
Use featurizers to make BOW/BON features
Add any other features you think might be important



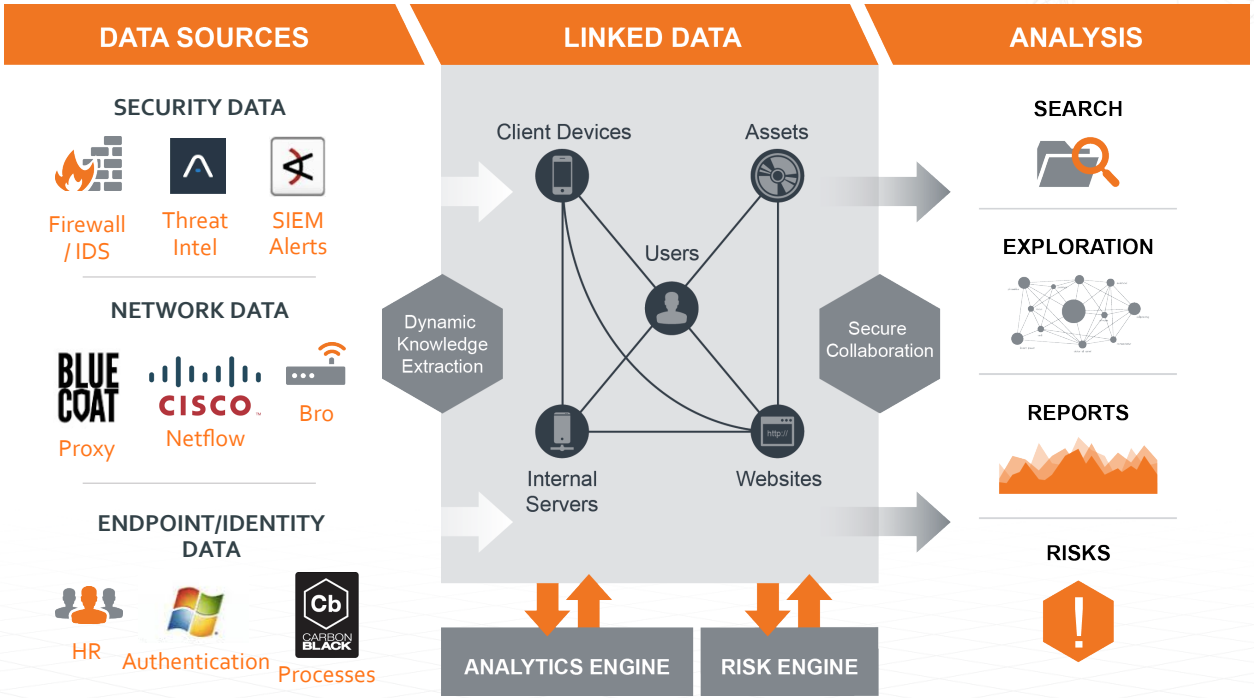
http://www.orwellloghomes.com/greybg.jpg

Takeaways

sqrrl

- Pandas and scikit-learn are highly active python projects that are bringing data science and machine learning tools to the masses
- Security technologists can (should?) leverage these tools as black or grey boxes
- Today, implementing 'standard' ML algorithms is not the long pole in the tent
- Snag Clearcut for an example

The Sqrrl Threat Hunting Platform



How To Learn More?

Go to **sqrrl.com** to…

- Download Sqrrl's Threat Hunting eBook
- Download the Sqrrl White Paper on Threat Hunting Platforms
- Request a Sqrrl Test Drive VM
- Download Sqrrl's Product Paper
- Reach out to us at info@sqrrl.com

More Info

Chris McCubbin
Director of Data Science
@_SecretStache_
chris@sqrrl.com

David J. Bianco
Security Technologist
@DavidJBianco
dbianco@sqrrl.com

Clearcut
Machine Learning for Log Review

https://github.com/DavidJBianco/Clearcut
(iforest branch for iforests)

sqrrl
Target. Hunt. Disrupt.