The Long & Winding Road to "Production-Worthy"

Emily Heath, PhD

DHS HSSEDI FFRDC

Operated by the MITRE Corporation

HSSEDI
Homeland Security Systems Engineering & Development Institute

Acknowledgement for DHS Sponsored Tasks

The Homeland Security Act of 2002 (Section 305 of PL 107-296, as codified in 6 U.S.C. 185), herein referred to as the "Act," authorizes the Secretary of the Department of Homeland Security (DHS), acting through the Under Secretary for Science and Technology, to establish one or more federally funded research and development centers (FFRDCs) to provide independent analysis of homeland security issues. MITRE Corp. operates the Homeland Security Systems Engineering and Development Institute (HSSEDI) as an FFRDC for DHS under contract HSHQDC-14-D-00006.

The HSSEDI FFRDC provides the government with the necessary systems engineering and development expertise to conduct complex acquisition planning and development; concept exploration, experimentation and evaluation; information technology, communications and cyber security processes, standards, methodologies and protocols; systems architecture and integration; quality and performance review, best practices and performance measures and metrics; and, independent test and evaluation activities. The HSSEDI FFRDC also works with and supports other federal, state, local, tribal, public and private sector organizations that make up the homeland security enterprise. The HSSEDI FFRDC's research is undertaken by mutual consent with DHS and is organized as a set of discrete tasks. This report presents the results of research and analysis conducted under:

70RNPP19FR0000012
Cybersecurity and Infrastructure Security Agency (CISA) Cybersecurity Division Network Security Deployment (NSD)
The purpose of the task is to provide systems engineering, integration, acquisition, program management, and cyber security subject matter expertise to define, develop, and deploy NCPS across the Federal Departments and Agencies (D/As) (the .gov domain).
The results presented in this report do not necessarily reflect official DHS opinion or policy.

HSSEDI
Homeland Security Systems Engineering & Development Institute

| 3 |

Problem Introduction

- **Fraudulent domains: malicious domains posing as well-known services or websites**
  - Commonly used by APT & criminal groups for phishing attacks and delivering malware
    - Ex: DarkHotel APT group used microsoft-xpupdate[.]com and adobearm[.]com to target political leaders in Asia
  - Content of domain is often identical or nearly-identical to legitimate webpage
    - Same logos, templates, fonts, images, color schemes, etc.

- **How can we reliably detect these* types of domains?**

***generic fraudulent domains, not domains unique to an organization's threat profile**

FraudDomains v1.0

- **Idea: Fraudulent & masquerading domains are more likely to contain certain words (targeted services/brands, ‘update,’ ‘download,’ etc.)**
  - List of 37 key terms from expertise & analysis of malicious domains

Match Results for Corpus of 65,625 Fraudulent Domains		
Number of Matches	Count	Percent
$[1, \infty)$	50648	77.18
$[2, \infty)$	18606	28.35
$[3, \infty)$	6490	9.89
$[4, \infty)$	2321	3.54
$[5, \infty)$	875	1.33

- **v1.0 looks for these key terms after filtering with whitelist**

| 5 |

FraudDomains v1.5

- **Idea: Extend the reach of v1.0 by adding shingle matching**
  - Shingle: trigram of characters, form list by taking the set of all trigrams from key term list (105 shingles)

Match Results for Corpus of 65,625 Fraudulent Domains		
Number of Matches	Count	Percent
$[0,4]$	57673	87.88
$[5,\infty)$	7952	12.12
$[6,\infty)$	5770	8.79
$[7,\infty)$	4244	6.47
$[8,\infty)$	3146	4.79

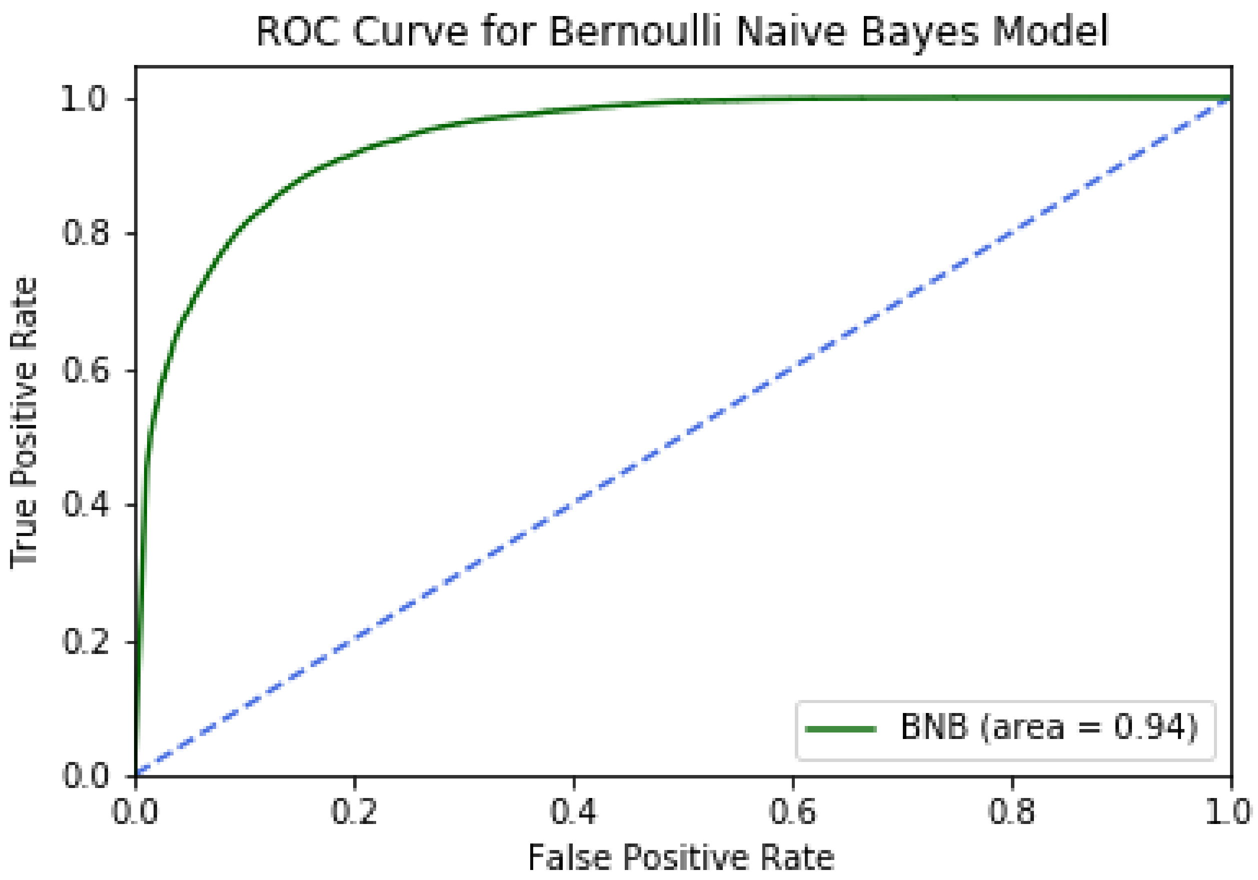
Method	Number of Domains Flagged
Full Match Only	11301
Shingle Match Only	647
Full or Shingle Match	7952

- **v1.5 looks for key terms or shingles, either can trigger a result**

| 6 |

FraudDomains v2.0

- **Idea: Incorporate machine learning model to extend reach of v1.5**

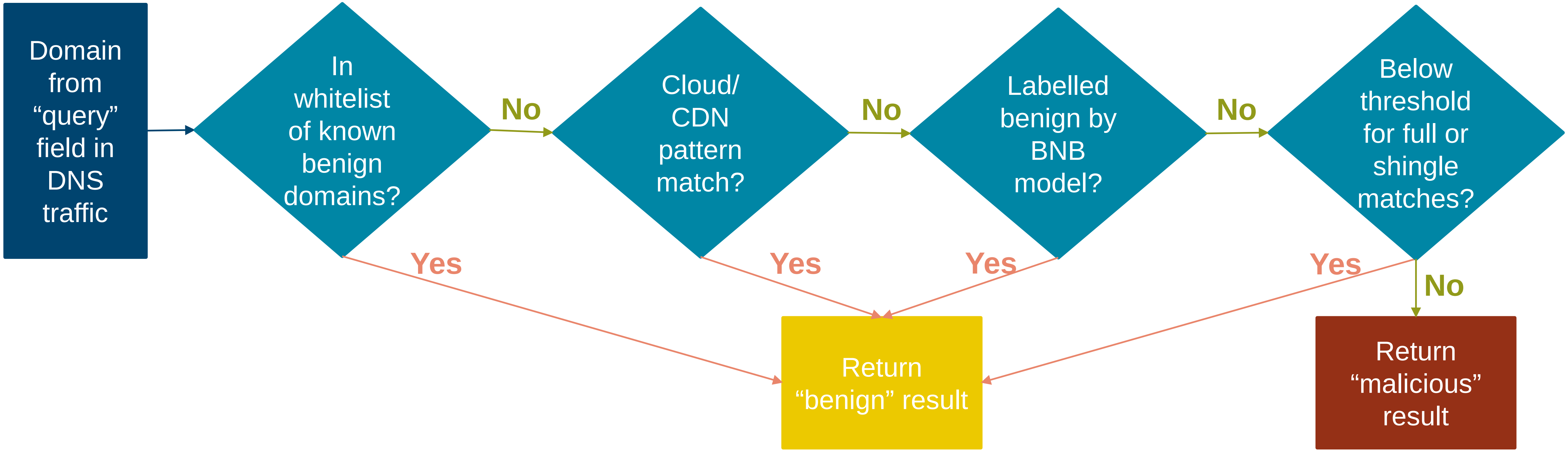


Method	Number of Domains Flagged
BNB Model Only	9688
Match Scheme Only	373
BNB & Match	5049

- **Plot twist: operational testing showed v1.5 was noisier than anticipated (too many FPs)**
  - New plan: use BNB model as first-pass filter

v2.0 Major Problem 1: False Positives

- **Additional operational testing showed v2.0 was frequently marking cloud/CDN domains as malicious**
  - Result: LOTS of alerts when running on real DNS traffic
  - Solution: implement a second whitelist filter looking for identified patterns



| 8 |

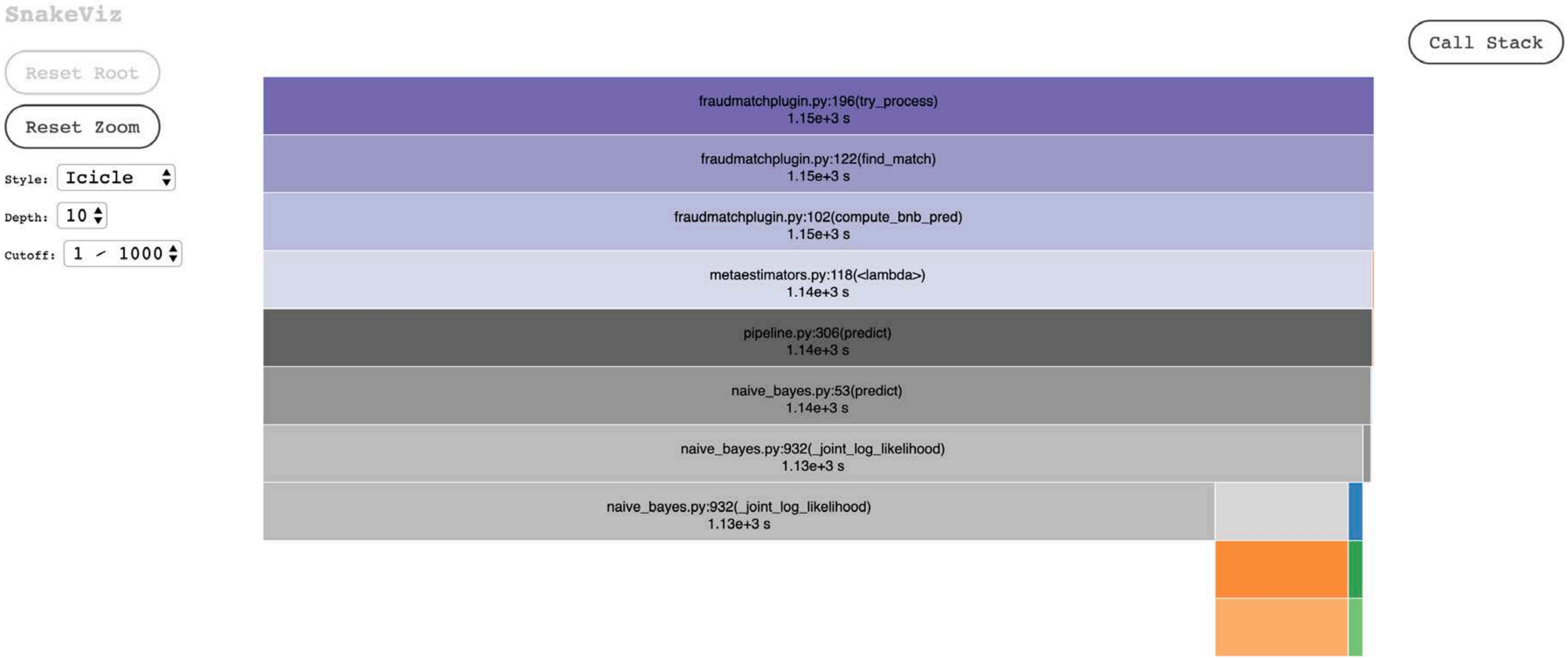
v2.0 Major Problem 2: Scaling

- **Integration testing showed FraudDomains could process around ~4.5 domains per second**
  - – Doesn’t stand a chance in production: average load is ~23K per second
    - Goal: Get close to 1.9K domains per second

To find a solution, we need to first find the problem

Improve integration testing → Review code → cProfile → lineprofiler

Using cProfile



ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
5275	981	0.186	1135	0.2151	naive_bayes.py:932(_joint_log_likelihood)

| 10 |

Using line_profiler

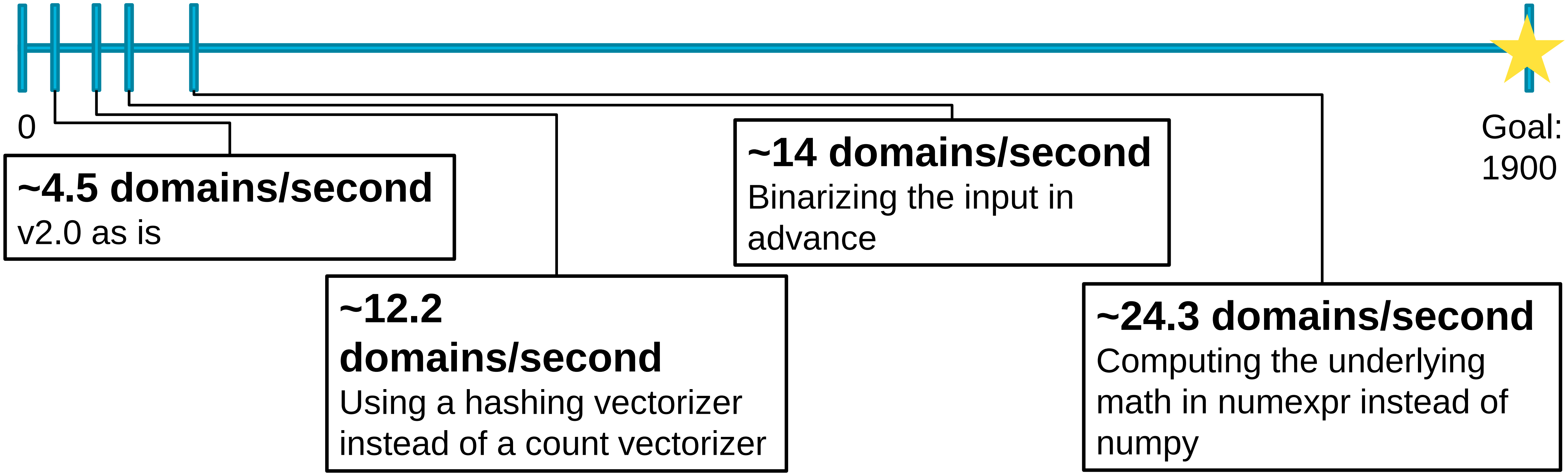
```
Timer unit: 1e-06 s

Total time: 0.259871 s
File: lineproffraud.py
Function: _joint_log_likelihood at line 300

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   300
   301                                               @profile
   302                                               def _joint_log_likelihood(self, X):
   303         1         17.0     17.0      0.0          """Calculate the posterior log probability of the samples X"""
   304                                                   check_is_fitted(self, "classes_")
   305         1        120.0    120.0      0.0
   306                                                   X = check_array(X, accept_sparse='csr')
   307         1          1.0      1.0      0.0
   308         1      15325.0  15325.0      5.9          if self.binarize is not None:
   309                                                       X = binarize(X, threshold=self.binarize)
   310         1          4.0      4.0      0.0
   311         1          1.0      1.0      0.0          n_classes, n_features = self.feature_log_prob_.shape
   312                                                   n_samples, n_features_X = X.shape
   313         1          1.0      1.0      0.0
   314                                                   if n_features_X != n_features:
   315                                                       raise ValueError("Expected input with %d features, got %d instead"
   316                                                                        % (n_features, n_features_X))
   317         1     171261.0 171261.0     65.9
   318                                                   neg_prob = np.log(1 - np.exp(self.feature_log_prob_))
   319         1      70239.0  70239.0     27.0          # Compute  neg_prob · (1 - X).T  as  ∑neg_prob - X · neg_prob
   320         1       2900.0   2900.0      1.1          jll = safe_sparse_dot(X, (self.feature_log_prob_ - neg_prob).T)
   321                                                   jll += self.class_log_prior_ + neg_prob.sum(axis=1)
   322         1          2.0      2.0      0.0
   323                                                   return jll
```

| 11 |

The Changes We Tried Along the Way

- **Profiling showed Naïve Bayes computation was expensive**

(A Very Not-to-Scale) Domains/Second Progress Chart

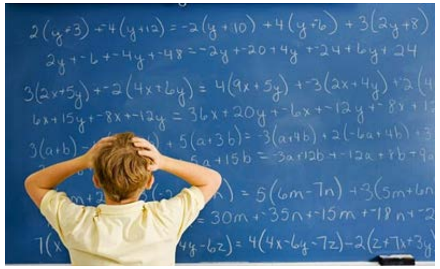


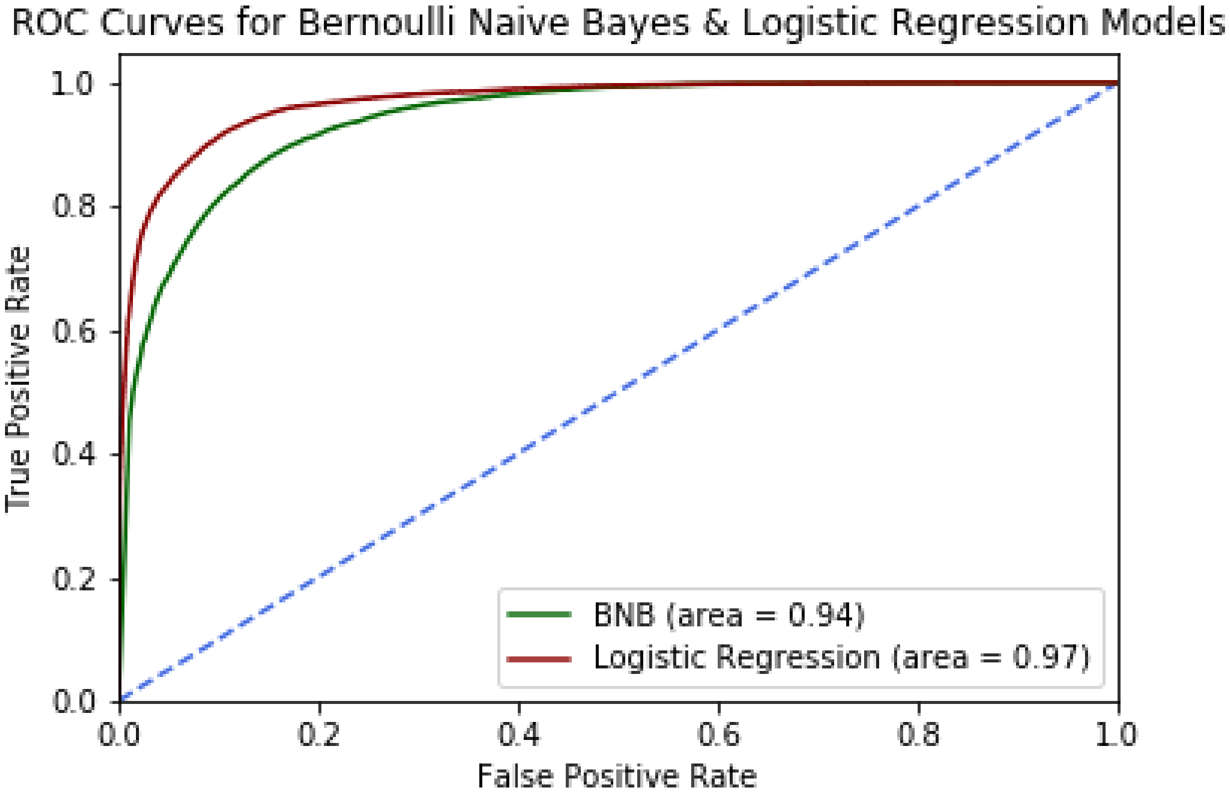
- **The scale of these changes was insufficient**

| 12 |

Arriving at v3.0

- **Lessons we learned: math is hard**
- **New plan: do less math!**
  - Switched to logistic regression model: ~1600 domains/second





Method	Number of Domains Flagged
Log Model Only	10823
Match Scheme Only	246
Log & Match	5176

- **Bonus perk: thresholding on probability**

| 13 |

A Miscellaneous Details Detour

- **Pytest is great, and pytest as part of a CI/CD pipeline is an actual lifesaver**
  - Pylint is ~~a nightmare~~ useful too

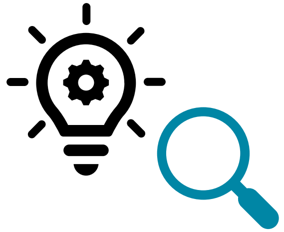
- **Version control is your friend**
  - Code, packages, data & models

- **Obligatory comment about Docker**

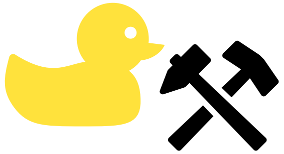
| 14 |

Conclusions

- **There are many aspects of performance to consider for an analytic**
  - Experimental results demonstrate an initial path forward
  - Operational results expose unforeseen weaknesses



- **Writing good software is hard (for me)**
  - Robust testing & tools to dig into the code help identify areas that can be modified/improved/fixed



- **Keep on keepin' on**
  - Continuously evaluate the analytic to find opportunities for refinements, enhancements, improvements