Proceedings of Blackhat 2017
July 26-July 27, 2017, Las Vegas, USA

GARBAGE IN, GARBAGE OUT: HOW PURPORTEDLY GREAT ML MODELS CAN BE SCREWED UP BY BAD DATA

Hillary Sanders
Data scientist
Sophos Group

Joshua Saxe
Chief Data Scientist
Sophos Group

ABSTRACT

As machine learning has become more common within computer security, researchers have focused on improving machine learning architectures to better detect attacks. But what if the data we're using to train and evaluate our models is sub-optimal?

Cybersecurity machine learning researchers face two main problems when acquiring data to train models. First, any available data is necessarily old and potentially outdated in comparison to the environment the model will face on deployment. Second, researchers may not even have access to relevant past data, often due to privacy concerns.

The greatest model, trained on data inconsistent with the data it actually faces in the real world, will at best perform unreliably, and at worst fail catastrophically. In short, the worse the data that goes into a machine learning system, the worse the results that come out. How do we mitigate this problem in a security context?

In this paper, we describe one remediation that we've found helps us estimate the impact of, and mitigate this issue. Specifically, we present sensitivity results from the same deep learning model designed to detect malicious URLs, trained and tested across 3 different sources of URL data. We review the methodology, results, and learnings from this analysis.

NOMENCLATURE

AUC	Area under the curve, where the curve is a ROC curve (Receiver Operating Characteristic). A value of .5 is bad - it represents the predictive capability of a coin flip, whereas 1 represents perfectly ordered predictions.
train set	Data used to train a machine learning model. Independent (separate) from the test set data.
test set	Data used to test a machine learning model. Independent (separate) from the training data. Often, test data is referred to as 'holdout' data, because it is withheld from training so as to be used for testing.

INTRODUCTION

As the volume and sophistication of new cyber threats has grown, signature based attack detection methods have struggled to keep up. The security community has adapted by increasingly pursuing machine learning based detection approaches.

Often, papers and discussions focus only on the machine learning systems themselves, with little mention of identifying the right data with which to train and test these systems. But in a cybersecurity context, where attackers and networks evolve over time and data differs from network to network, the training and test data we use to create and evaluate our systems matters greatly.

How do we pick the best data with which to train our systems, and how do we use our data to predict how well our systems will detect attacks once we deploy them in a new data environment?

In cybersecurity machine learning, we never have perfect data with which to train and test our machine learning models, due to several factors:

1. By the time we deploy our models, our data will already be 'old' with respect to the evolution of our adversaries and the evolution of our customers' networks.
2. We often don't have access to the types of data our models will face upon deployment, generally due to privacy reasons. (For example, we may want to develop an accurate URL

detector, but don't have access to all customers' past private URL histories).

These two issues are very important to keep in mind during development, because model accuracy is *incredibly* dependent on factors other than just model architecture - namely, the training and test data used. In short, the more our training and test data differs from deployment data, the worse and more unpredictable our deployment accuracy is going to be. Because of this, it's important to be able to predict and mitigate the effects of these two issues.

The first issue can be dealt with by using time decay analysis - training a model on past data up to time t, and then testing it on data from different intervals after time t. Model configurations and training data should be selected based on these results, rather than the results from classical holdout sets (withholding a random percentage of training data to be used as test data, or performing k-fold cross validation).

The second issue, however, is harder to deal with. If we're unable to accurately simulate the data we want our model to eventually perform on, we can't be sure of how it will do on deployment - which is essential.

When precise deployment test data simulation is not possible, it is still possible to map how sensitive a given model is to changes in test data distributions. In this paper, we showcase a sensitivity analysis that allows us to assess the impact training and test data selection has on model accuracy. An ideal model is consistently accurate across a wide variety of test data circumstances. Even if a model is extremely accurate on one set of test data, if it proves unable to handle significant variation in input data (assuming consistent labels), it may not be safe to deploy.

The rest of the paper is structured as follows. To contextualize our analysis, we survey some machine learning research on sensitivity analyses and training data selection. Then, we describe the methodology of the sensitivity analysis. Then, we describe the data and model used in our example. Finally, we describe our results and discuss how they informed our training dataset selection.

Related Work

Sensitivity analyses are the study of how uncertainty in the output of a system (like test accuracy) can be attributed to different sources of uncertainty in its inputs (like what our deployment data looks like). In our case, we're uncertain about our deployment test data, and so we perform a sensitivity analysis by tracking how our test accuracy changes as we change the training and test datasets used.

The use of sensitivity analyses in machine learning is not novel, and similar approaches have been used in machine learning.

- Hunter et al. [1] uses a sensitivity analysis to see how AUC changes with input features used in a neural net, in order to best pick the input features. This is similar to what we do, except instead of trying to pick the best input features, we are trying to pick the best training datasets, and modifying both the training and test data sources to do so.
- Vasilakos et al. [2] does the same (sensitivity analysis of input features for a neural net), but for fire ignition risks.
- Gevrey et al. [3] does the same, but with two-way input variables.
- Zhang et al. [4] performs a sensitivity analysis on a convolutional neural nets, to explore the effect of model architecture components on model performance.

Our method is very similar, though instead of exploring the uncertainty surrounding feature selection, we're exploring the uncertainty surrounding dataset selection, which is particularly relevant in cybersecurity since deployment dataset distributions are always unknown to at least some extent. In other words - instead of being unsure about which features to use, we're unsure about what data to use, and we want to better understand how severely a model's accuracy can change as a result of using different datasets for training and testing.

Golbraikh et al. [5] focuses on train and test set selection based on diversity principals, as a means to produce models that generalize well to data not used during model development. However, this is done in the context of QSAR modeling and uses different methods.

Our core contribution has to do with our results and findings, which are specific to our proprietary URL model, described in detail here [6]. Our actual methodology should not be considered novel (though perhaps underutilized), because at its core, it's just a specific way to implement a sensitivity analysis.

Method

Sensitivity analyses help estimate uncertainty in the output of a model, based on uncertainty in its inputs. Because we're uncertain about future data models will face in the wild, it can be useful to see how our accuracy metrics change as we change our definition of training and test sets. This can help us understand how much we can expect our accuracy to vary on deployment, and which training datasets tend to generalize better to others.

This method can be especially useful when we expect the data our model will eventually face in deployment to be significantly different than our training data. The idea is to see how sensitive our model's accuracy is to changes in training and testing data.

First attain K inherently different datasets, ideally covering the same general type of data (for example, URLs) over the same period of time. Ideally, the differences in our datasets should attempt to mimic the types of differences we expect might ex-

TABLE 1. Dataset Descriptions

Data Sources	Training Sets	Test Sets	Preprocessing
VirusTotal	10 million URLs from January 1-31st 2017, 400,000 labeled as malicious, the rest benign.	28,981,874 URLs ranging from February 1st 2017 - March 31st 2017.	Each observation (scan) from VirusTotal is annotated with a number of vendors who classified the item (at the time of scan) as either malicious or benign. We gave URLs with 0 malicious vendor flags a label of 0, URLs with 5 or more vendor flags a label of 1, and discarded the rest.
Sophos Internal Test Data	10 million URLs from January 1-31st 2017, 400,000 labeled as malicious, the rest benign.	3,812,130 URLs ranging from February 1st 2017 - March 31st 2017.	This dataset is a sample of unique URLs from internal Sophos URL test data, with whitelisted URLs and domains removed.
CommonCrawl & PhishTank	9,982,107 CommonCrawl URLs from January 1-31st 2017, and 17,893 PhishTank URLs from on or before January 31, 2017 (in an attempt to get more data).	18,141,565 URLs ranging from February 1st 2017 - March 31st 2017.	URLs gathered from CommonCrawl were assumed benign, and observations gathered from PhishTank were labeled as malicious. Note that the number of malicious files PhishTank has available is quite small.

ist in future deployment data. For example - if we are *certain* that deployment data will always use a certain type of character encoding, we shouldn't include datasets that use different encodings.

1. For each dataset i in 1:K:
   (a) Split the dataset i up into two disjoint groups: the train set $train_i$, containing data from before date d, and a test set $test_i$, containing data after a date d. (If this is not possible, simply split the datasets into two disjoint groups using other means).
   The number of each label class observations (for us, malicious and benign URLs) should be roughly equal across all K datasets (the amount of data in the validation sets matter less). d should also be the same across all K datasets.
   (b) Train a model M_i on the training set $train_i$.
2. After all K models have been separately trained, for each model M_i for i in 1:K:
   (a) for each test dataset $test_j$ for j in 1:K:
      i. evaluate model M_i on test dataset $test_j$ in the manner of your choice. Example metrics: AUC, AUC by day, or detection rates at selected fixed false positive rates.

For single-point evaluation metrics, this should yield a KxK accuracy matrix: M_i X $test_j$ (training set source X test set source).

There are two main aspects of the evaluation.

First is the general variability of your test accuracies. If you expect deployment data to be less different than the difference between your datasets, then the variability in your test accuracies can be interpreted as a rough upper-limit estimate as to how accurate your model will be on deployment (given a training dataset $train_i$). For example, if you're considering using dataset $train_i$ to train your deployment model, inspect the variability of accuracy values of column i in your accuracy matrix.

Second is the success each trained model M_i had across various test datasets. Training sets that generalized well across all test datasets (especially the ones most likely to look like future deployment data) are best.

In other words, training datasets that result in high average, low variance accuracy across test sets are ideal.

Model Description

In this paper, we use a deep convolutional neural net developed at Sophos to classify short strings. The structure of the model is out of scope for this document, but more can be learned about it from [6].

Data Description

To showcase the problems we are describing and attempting to track and mitigate, we use three separate sources of data. Each

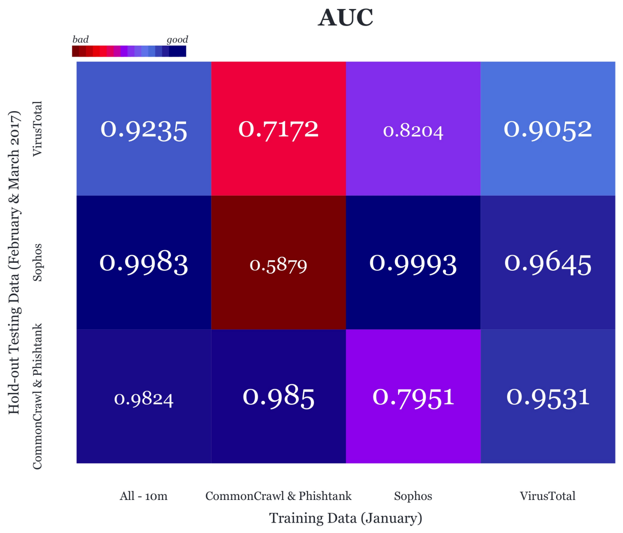

FIGURE 1. Above, AUC rates are shown. Each box represents the outcome of a model trained (x-axis) and tested (y-axis) on a different dataset. *Note that VirusTotal x VirusTotal AUC would usually be in the high 90's, but in this dataset, a single popular domain's labels flipped over time in VirusTotal data (many URLs), which caused AUC to drop significantly. See the appendix, specifically figure 8 for more details.*

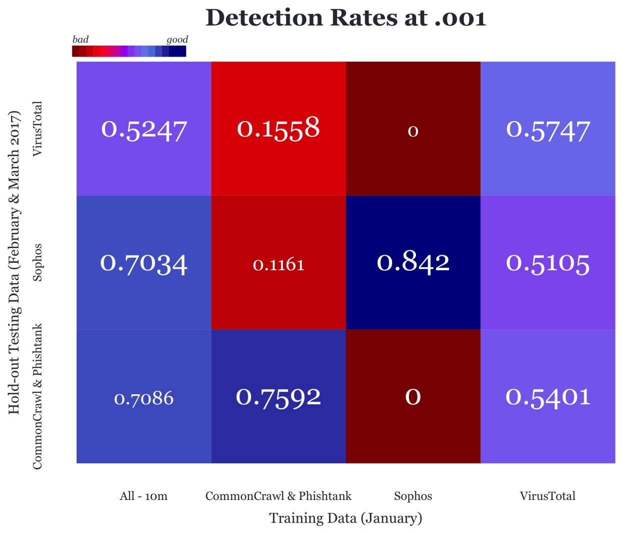

FIGURE 2. Above, detection rates at a fixed false positive rate of 10−3 are shown. Each cell represents the outcome of a model trained (x-axis) and tested (y-axis) on a different dataset.

source contains 10 million URLs from January 2017 (for training), and a secondary, independent set of test URLs from January 2017 to March 2017. All analyses and plots shown in this paper refer to these data (unless otherwise specified, as is done in the appendix). Details are shown in Table 1.

Results

In what follows, we outline the results we obtained from applying the train test sensitivity analysis method to the model architecture and datasets described above. Along with the models trained with the three datasets described in table 1, we've also included a model trained on a random third of each of the three original training datasets.

The sensitivity plots above (figures 1 & 2) yield two insights:

1. First, we have a better idea of how variable our models' accuracies can be when trained and tested on different datasets.
2. Second, we get information about what datasets we should use to train our deployment model. Training on VirusTotal data yielded a model that's dramatically better at generalizing to the other sources of test data, while Sophos did best by far on its own holdout dataset.

While we expected the CommonCrawl / PhishTank trained models to do fairly badly when it came to generalizing to Sophos and VirusTotal URLs (due to lack of malicious data volume, and PhishTank focusing on solely Phishing websites), we weren't sure how Sophos and VirusTotal trained models would perform. Sophos data performed fairly well, but VirusTotal was the clear winner in terms of generalizing to the other datasets. The poor performance between CommonCrawl and Sophos suggests that their distributions differ significantly, while the VirusTotal model's success suggests that it contains patterns common to both CommonCrawl and Sophos data. Another possible explanation is that the benign data on VirusTotal tends to look more suspicious, and is thus closer to the malicious / benign decision boundary, which might make for a better model.

Even if we theoretically had access to all Sophos client URLs, these results suggest that we still might want to include VirusTotal data in an attempt to keep from overfitting to past Sophos client data, which may not fully represent future Sophos data.

The leftmost column of the plots above (figures 1 & 2) show a model trained on a third of each of our original dataset (totaling 10 million training observations). Test accuracies across all three datasets are near or even higher than the maximum of the rows to each cell's right!

In this example, we used three entirely different sources of URL data. This isn't at all necessary: you could also run the same analysis training and testing on, for example, VirusTotal data with different labeling schemes, or data from different months, etc... The basic idea is to train and test on inherently different set of data, evaluate the results, and hopefully learn something about the sensitivity and generalizability of each trained model.

Summary

Deep learning uses a massive amount of unseen complex features to predict results, which enables them to fit beautifully to datasets. But it also means that if the training and testing data are even slightly biased with respect to the real-world test case data, some of those unseen complex features will end up damaging accuracy instead of bolstering it. Even with great labels and a lot of data, if the data we use to train our deep learning models doesn't mimic the data it will eventually be tested on in deployment, our models are liable to fail in deployment.

Unfortunately, it's impossible to train on future data. And often we don't have access to that that even mimics past deployment data. But it is quite possible to simulate the errors we expect to have upon deployment by analyzing the sensitivity of our models to differences in training and testing data. By doing this, we can better develop training datasets and model configurations that are most likely to perform reliably well on deployment.

ACKNOWLEDGMENT

Thanks to Konstantin Berlin for his work with Joshua Saxe in developing the convolution neural net model used in this analysis. Also, thanks to Richard Harang for help with analyzing encoding differences amongst our datasets, his excellent theories about the results, and help with editing. And thanks to Madeline Schiappa for her help in acquiring internal test data.

REFERENCES

[1] Andrew Hunter, Lee Kennedy, Jenny Henry and Ian Ferguson. Application of neural networks and sensitivity analysis to improved prediction of trauma survival. 2000.

[2] Christos Vasilakos, Kostas Kalabokidis, John Hatzopoulos, and Ioannis Matsinos. Identifying wildland fire ignition factors through sensitivity analysis of a neural network. 2009.

[3] Muriel Gevrey, Ioannis Dimopoulos, and Sovan Lek. Two-way interaction of input variables in the sensitivity analysis of neural network models. 2006.

[4] Ye Zhang, Byron Wallace. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. 2016.

[5] Alexander Golbraikh, Alexander Tropsha. Predictive QSAR modeling based on diversity sampling of experimental datasets for the training and test set selection. 2002.

[6] Joshua Saxe, Konstantin Berlin. A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys. 2017.

Appendix

Here, we include some extra plots for readers who just can't get enough of them!

First, ROC (Receiver Operating Characteristic): each plot shows how each model did on a specific test dataset.

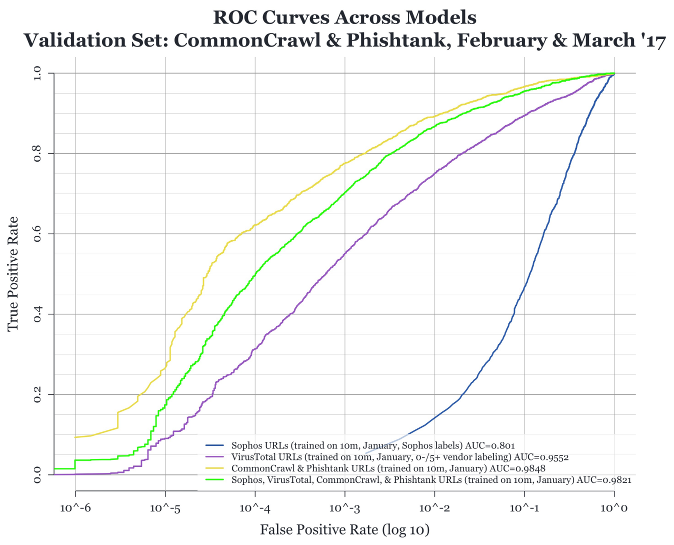

FIGURE 3. Model Performances on CommonCrawl data. Note how poorly Sophos performs on CommonCrawl data, whereas VirusTotal and the combined dataset perform quite well.

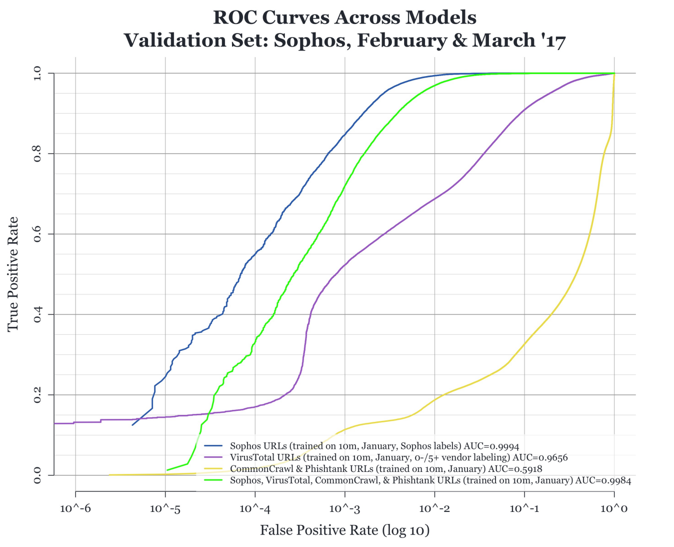

FIGURE 4. Model Performances on Sophos data. Again, not how VirusTotal and the combined dataset do surprisingly well.

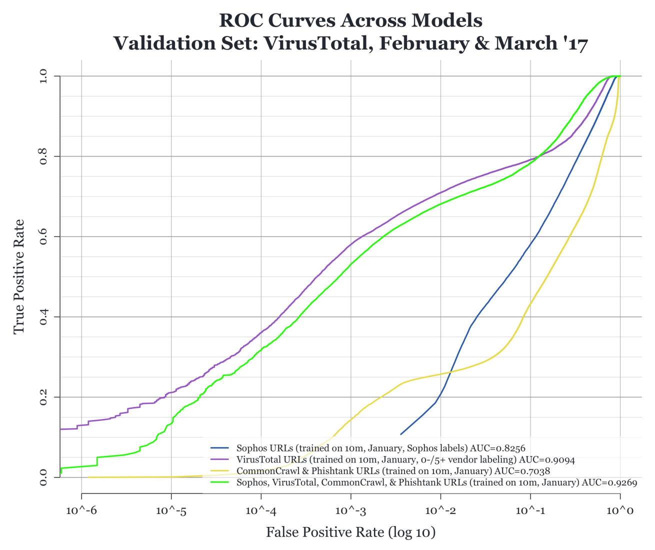

FIGURE 5. Model Performances on VirusTotal data. Note how poorly both Sophos and CommonCrawl do. The VirusTotal and combined datasets do *significantly* better..

Next, we show how accuracies decayed over time. To show this, we extended each holdout test set to include data from January, April, and May - *except* for PhishTank. Due to lack of data, we didn't have sufficient pre-February malicious PhishTank data for a holdout test set, so training data was used (bad). As such, the high accuracy seen from CommonCrawl / PhishTank data in *January* should be disregarded.

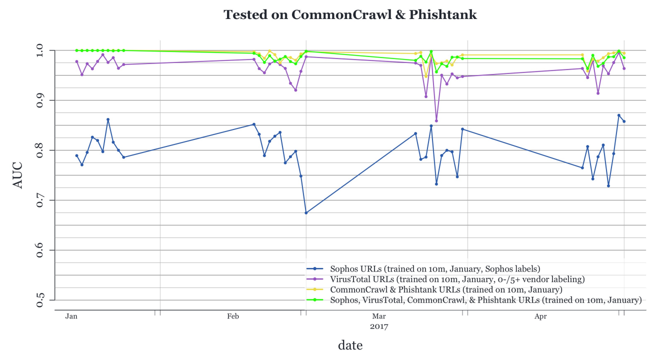

FIGURE 6. Model Performances on CommonCrawl data over time. Because of the way in which CommonCrawl data was downloaded, data is unfortunately grouped into a small number of dates.

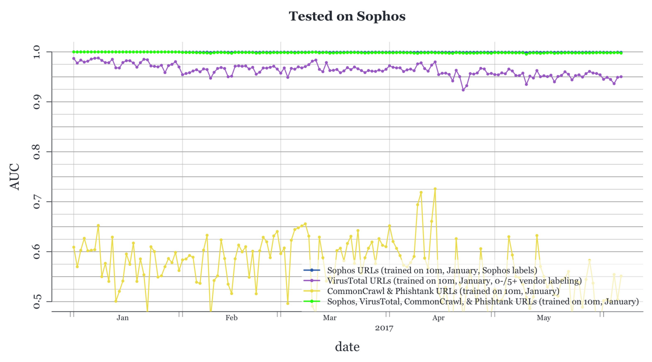

FIGURE 7. Model Performances on Sophos data over time.

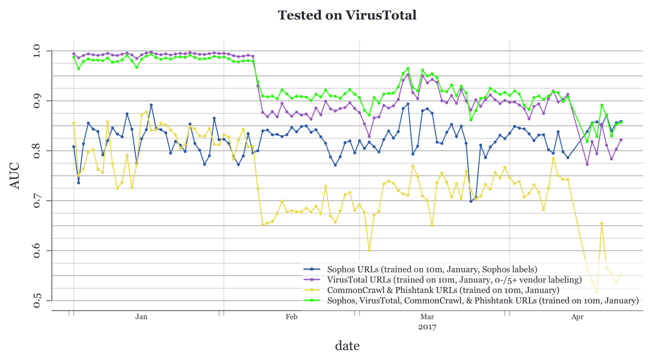

FIGURE 8. Model Performances on VirusTotal data over time. Note that the significant drop from February onwards in VirusTotal accuracy is caused almost entirely from a *single* popular VirusTotal domain (many URLs), whose labels changed over time as vendors changed their decisions. This drop is a somewhat rare occurrence (usually VirusTotal trained URL models decay much more slowly on their own data source), but was nevertheless included.