Week 3 Homework

Samuel Cuthbertson SAMUEL.CUTHBERTSON@COLORADO.EDU

Note: This write-up references the paper

Detecting Malicious Executables In The Wild

From this point on, the above paper will simply be referred to as "the paper" and the authors as "the authors" to simplify this text.

1. Identifying Executables Review

The authors claim to have found 255 million features (referred to as *n*-grams here) from 1971 benign programs and 1651 malicious programs, diluted down the the "best" features, and run them through 8 different classifiers. They achieved at worst an AUC of 88%, and at best an AUC of 99.6%, with results indicating that the used methods will scale well and increase accuracy.

The most interesting thing about this paper is their method of selecting *n*-grams from the hexdump of each file. They selected overlapping 4-byte sequences and expressed them in hex, such that the sequence 12 FG EE 49 A9 B0 would have the features 12FGEE49, FGEE49A9, and EE49A9B0. As the authors mentioned, this led to Naive Bayes struggling with the feature set since the fundamental assumption of conditional independence is obviously untrue. However, this also massively increased the feature count and contributed to the success of other models.

The selection of "relevant" features is also extremely interesting, and is addressed in the below questions (2.1.1 and 2.1.7).

The descriptions of all the various classifiers used was a fantastic section, and remarkably well written. It was approachable from someone with my level of knowledge, and didn't get bogged down with details of the specific implementations of each classifier, instead focusing on the concept which each classifier used. This was especially useful to me, someone with only a basic understanding of one of the 8 classifiers used. Most of my questions arose surrounding 'boosting', and which classifiers were combined to create "Boosted Naive Bayes", "Boosted J48" and "Boosted SVM".

Another very interesting part of this paper is the way in which the authors tool, MECS, operated. The ideas they through around in the conclusion about further implementations on distributed systems also peaked my interest, and I'm curious if any other entitys have done anything similar to that idea. Another curiosity is how many features could be selected today, even from the same data set. This paper was copyrighted more than a decade ago, and advancing in computing have only made those computations easier.

Overall, this was an excellent paper. All the questions I have from reading it are painfully obviously due to my own lack of knowledge, but it definitely answered most of the questions it rose. The paper was very well written: the authors method was sound, they referenced and described similar research, and their product was effective with their test data.

2. Written Problems

2.1. Regarding The Paper

2.1.1. What is a relevant feature?

In general, relevant features are those which help us classify in our context. For instance, if we wish to classify between a CSV file of data and a file of code, the number of commas in the file would most likely be a relevant feature. The length of the file, however, would be unlikely to help determine the type of file. In the context of this paper, they described relevant features as those which maximized their **IG** formula. The idea behind this was to determine which features were common to only one class, and select a quantity of those to use and compare while classifying new data. I agree with this, as it makes sense to discard features common to all classes since they only take up computational time without providing useful information.

2.1.2. Precision or recall?

In this paper, precision is much more important than recall. We could capture all malware by simply calling all programs malware, but then the system we are trying to protect has no way or reason to run. We would much rather miss some malware on a system than misidentify a non-malicious program as malware.

2.1.3. Example Set of N-Grams

If the given word from the program is

$$1A\ 34\ F6\ 72\ BA\ 33\ 11\ E9$$

Then the *n*-grams for that word will be

$$1A\ 34\ F6\ 72$$

$$34\ F6\ 72\ BA$$

$$F6\ 72\ BA\ 33$$

$$72\ BA\ 33\ 11$$

$$BA\ 33\ 11\ E9$$

2.1.4. Feature and Label Spaces

In this problem, the feature space consists of vectors size 500 (since the top 500 *n*-grams we chosen) consisting of 0's and 1's denoting where the appropriate *n*-gram is present in the program. The label space is simply a vector like

$$< 0, 1 >$$

where v_1 = 0 = not malware, and v_2 = 1 = malware.

2.1.5. Two Questions

One of the questions I'd like to ask the authors is their reasoning for only evaluating Windows PE Executable. I can think of a half dozen reasons off the top of my head, but I'm curious why they didn't go with a much bigger data set of any Windows Executable.

The other question was simply how SVMs work. The language used in this paper, and in the papers they reference, is far too complex and advanced for me to grasp.

2.1.6. Feelings About the Results

I'm very suprised at how accurate the results of this paper were. Given the data being as incomprehensible as it was, it's remarkable that their classifiers worked as well as they did. My best guess for why this was so is that they had many samples of similar malware, developed at similar times in history by similar authors. I'm curious how well the same approach would work on a data set of all Windows Executables, as gathered from a wider set of sources.

2.1.7. IG Function and Entropy

The **IG** function calculates the "Information Gain" from a certain feature, which can also be conceptualized as calculating which features have the highest entropy due to mostly appearing in only one class.

2.1.8. IG Function Prior and Estimation

The Prior in the **IG** Function is $P(C)$, which is estimated simply from the proportion of the training data that is class C.

2.1.9. ArgMax Function

The ArgMax function returns a class C which is most probably the class of the input vector. If a simple Max function was used, then the probability that class C is associated with the input vector would be returned.

2.1.10. Reasoning for Inconclusive Results from Boosted SVMs

The authors explain boosting as a way to increase the stability of unstable classifiers, and say that boosting can adversely affect classifiers that were stable to begin with. Since SVMs are very stable, boosting them decreases their relative accuracy at larger data sets.

2.1.11. Various Metric Questions

Q(1): The C4.5 Algorithm splits nodes based off of which criteria will most completely segregate each subtree to containing one class.

Q(2): Based off of my understanding of the C4.5 Algorithm, it splits nodes in order to **minimize** KL-Divergence from tree to tree.

Q(3): KL-Divergence is a measure of relative entropy between one probability distribution and another.

2.2. Loss Functions of Naive Bayes

The loss function of Naive Bayes is simply given by how many missclassifications the model has given test data. On wild data, another way of getting some idea about how accurate the model is is to store the probabilities for every class of a given vector, and subtract the highest probability from 1. The resulting number is the "Percent Wrongness" of that class and, when averages across multiple input vectors, tells us how poorly that model fits the data.

2.3. Logistic Regression and 0s

Logistic Regression does not run into the same crippling issues with 0's as Bayesian Models, due to mathematical core of that model being based around the exp function, instead of simple multiplication.