Android Malware Detection with Weak Ground Truth Data

Jordan DeLoach and Doina Caragea
Department of Computer Science
Kansas State University
{jdeloach,dcaragea}@ksu.edu

Xinming Ou
Deptartment of Computer Science and Engineering
University of South Florida
xou@usf.edu

***Abstract*—For Android malware detection, precise ground truth is a rare commodity. As security knowledge evolves, what may be considered ground truth at one moment in time may change, and apps once considered benign turn out to be malicious. The inevitable noise in data labels poses a challenge to creating effective machine learning models. Our work is focused on approaches for learning classifiers for Android malware detection in a manner that is methodologically sound with regard to the uncertain and ever-changing ground truth in the problem space. We leverage the fact that although data labels are unavoidably noisy, a malware label is much more precise than a benign label. While you can be confident that an app is malicious, you can never be certain that a benign app is really benign or just an undetected malware. Based on this insight, we leverage a modified Logistic Regression classifier that allows us to learn from only positive and unlabeled data, without making any assumptions about benign labels. We find Label Regularized Logistic Regression to perform well for noisy app datasets, as well as datasets where there is a limited amount of positive labeled data, both of which are representative of real-world situations.**

I. Introduction

The area of Android malware detection has recently evolved to increasingly use machine learning [1], [6], [24]. Of critical importance to a machine learning approach, before choosing a classifier or extracting features, is the creation of datasets and the labeling of individual examples as positive (malware) or negative (benign). High quality ground truth is essential to a successful machine learning approach. Low quality ground truth, on the other hand, can make the problem of finding an effective decision boundary to differentiate a malicious app from a benign app much more difficult.

Traditional supervised machine learning algorithms require not just high quality ground truth, but rather a large amount of high quality ground truth. When only a small amount of high quality ground truth is available, standard semi-supervised learning approaches (*e.g.*, expectation maximization or self-training), which can make use of unlabeled data in addition to labeled data, have been successfully used [3]. Similar to the effect on supervised learning, noisy labeled data can cause semi-supervised learning classifiers to shift away from the correct decision boundary [7]. As an alternative to standard semi-supervised learning, Ritter et al. [17] recently proposed a "weakly supervised" approach (a.k.a., one-class semi-supervised learning), which learns from a small amount of high quality positive data *only* and large amounts of unlabeled data, by regularizing the label distribution of the unlabeled data towards a user-provided expected distribution.

High quality ground truth is a rare commodity and exceptionally difficult to obtain for Android malware detection. Benign labels are usually more imprecise than malware labels, in the sense that a new app that is originally labeled benign might later have its label changed to malware. Given these facts about the ground truth in Android app security, our goal is to research the effectiveness of the weakly supervised approach proposed in [17], called Label Regularized Logistic Regression, for the problem of accurately identifying Android malware.

We experiment with different amounts of noise, different amounts of labeled data, and different label distributions. Our working hypothesis is that the one-class (positive only) semi-supervised approach performs better than the standard supervised learning approaches which would use some amount of low quality ground truth.

We summarize our contributions to the field in the following three points:

1) We are the first group to experiment with applying a one-class semi-supervised approach that uses only positive data together with unlabeled data in the Android malware detection domain.
2) We discuss how a one-class semi-supervised approach fits with the specific intricacies of the Android malware domain.
3) We provide extensive analysis of inherent flaws in current techniques for ground truth on Android, and discuss how those flaws can be best remedied with one-class semi-supervised learning approaches.

II. Background

A. Ground Truth

Attempting to concretely label an app as benign or malware, and separating a set of apps into benign and malicious apps is a difficult task. Even for the Google Play Store[1], accurately labeling an app as malicious remains a problem as Google Play Store has been shown to have a relatively substantial

[1]https://play.google.com

amount of malware. Reported levels of malware in the store vary anywhere from Google's self-reported less than 1% [9] to 7% or higher [5], [11]. Either way, the amount of malware in or attempting to get in the app store is drastically growing [16]. Be it Google Play, Amazon Appstore, or Android malware repositories, app sources are inherently noisy, and some means of ground truth preparation are necessary to separate a base dataset for training a classifier.

In order to create ground truth, there are four main existing techniques, each with varying levels of accuracy [10].

1) Some attempt to manually label each instance as either benign or malware. While highly accurate, this approach does not scale beyond hundreds or thousands of apps to the millions that exist in the wild. Zhou et al. [25] did an in-depth analysis and manually labeled more than 1,200 Android malware to conduct their study.
2) Some simply take the instance labels provided with a dataset, e.g. considering all apps or the popular apps in the Google Play Store as benign, and those from malware repositories as malicious. This often results in many false positives and false negatives. MAST [4] used Google Play apps and two malware repositories to label apps as benign and malware, respectively. Wu et al. [23] considered the apps collected from a Chinese app market as benign apps while utilizing the VirusShare dataset as malware apps.
3) Some utilize a single anti-virus product to determine the label, potentially biasing classifier performance towards what an individual anti-virus believes to be malware.
4) Others attempt to verify ground truth by submitting apps to VirusTotal [21]. VirusTotal is a service that takes malware binaries and runs them against a variety of different anti-virus products, over 54 at the time of writing. From there, most approaches determine a threshold, and consider an app to be malware if the number of anti-virus products that flag it as malware is greater than the threshold. Drebin [1] labeled an app as malware if 2 out of a select 10 anti-virus products flagged it as malware. Roy et al. [18] used a threshold of 10 out of 54.

While the fourth option has been shown to perform the best of the four, there exist inherent problems in using anti-virus solutions, specifically with newer malware. As noted by Kantchelian et al. [10], anti-virus products tend to prefer false negatives to false positives, thereby erring on the side of over-labeling as benign and under-labeling as malware. This has been verified by Chen et al. [5] where there were found to be more than 34,000 malicious apps missed by the majority of VirusTotal scanners. While it is nice that anti-virus products do not over-assert malware claims that could impair the popularity of an app, many apps will receive no flags as malware in the instance that the anti-virus is uncertain. Furthermore, anti-virus solutions are based upon signature detection. Security experts must first find and craft a signature before an anti-virus can detect the malware, which means that when a new strain or family of malware comes out (*e.g.*, zero-day malware), the anti-virus will often label it incorrectly as benign. These insights lead us to believe in an imbalance in the precision of anti-virus class labels, with a malware label being comparatively more precise than that of a benign label.

B. *Supervised Learning*

Given a set of ground truth labeled examples of both malware and benign apps, a traditional supervised classifier will attempt to learn the differences between the positive and negative classes by identifying an implicit or explicit decision boundary between the two. The classifier relies heavily upon the provided labels for the examples both when training and testing. In training, the labeled examples are critical for creating an accurate training space and identifying an accurate decision boundary. In testing, a mislabeled app would penalize the performance metric of a classifier if the classifier guessed the correct label. In supervised approaches the presence of mislabeled examples in the training set can easily and substantially degrade classifier performance by leading to an inaccurate decision boundary between the two classes, and thus causing inaccurate predictions. Similarly, mislabeled examples in the test set will negatively affect the performance even when the classifier produces accurate predictions.

C. *Semi-supervised Learning*

Semi-supervised learning, as opposed to supervised learning, attempts to train from a small amount of labeled together with unlabeled data. The benefits of semi-supervised learning come from the fact that when providing ground truth, you are not constrained by a binary model of class labels. Having binary labels means that an app is either positive, negative, or excluded from the dataset. A semi-supervised approach, on the other hand, is able to leverage unlabeled data to augment the existing limited amount of labeled data and learn a better classifier.

There are various approaches to semi-supervised learning, for example, expectation maximization (EM), self-training (ST) and co-training (CT). Such approaches use both positive and negative examples as labeled data, together with unlabeled data. As opposed to semi-supervised approaches that use both positive and negative data, Label Regularized Logistic Regression (LR-LR) [13], [17] is an approach which allows us to learn from positive examples as the only set of labeled data. LR-LR uses the log probability over just the positive class and regularizes the predicted labeled distribution of the unlabeled data towards a constant provided by an expert, where the constant represents the expected distribution of positive apps in the unlabeled dataset. Experimental results in [17] suggested that LR-LR performs better than a modified one-class EM, where the expected distribution of positive apps is used as prior. We also experiment with LR-LR as a one-class semi-supervised classifier, given the difficulty in accurately labeling negative (benign) examples in the Android app security domain.

III. Methods

A. Logistic Regression

Logistic Regression [15] is a linear model (classifier) that uses the logistic function to find the probability of a given feature vector, x, belonging to a certain class, y, given a vector of weights, θ, as follows:

$$p(y=1|x) = \frac{1}{1+e^{-\theta \cdot x}} \tag{1}$$

The optimization function, with L^2 regularization is:

$$O(\theta) = \sum_{i}^{N} \log p_\theta(y_i|x_i) - \lambda^{L^2} \sum_{k} \theta_k^2 \tag{2}$$

where N is the number of examples in the training set. The L^2 regularization is a standard method to balance between maximizing the log likelihood probabilities of the model and achieving a simpler model. The goal of having a simpler model will also lend itself to a model that generalizes better, and is, thus, more applicable on other datasets in the same domain.

B. Label Regularized Logistic Regression

For our work, we utilize Label Regularized Logistic Regression (LR-LR), which is a variant of the expectation regularization for Logistic Regression [13]. This method was applied for detecting tweets related to certain intrusion events using a small amount of positive seed instances and a large number of unlabeled instances [17]. The benefit of LR-LR stems from the fact that it only trains on positive instances and unlabeled instances.

As such, the optimization function for LR-LR, shown in Equation (3), is similar to the Logistic Regression with L^2 Regularization with two modifications. First, the Log Likelihood function (first term) is only calculated over the N_p positive labeled (L_p) examples. Second, the Label Regularization term (second term) is added to regularize the estimation of class distribution between positive and negative instances in the unlabeled dataset. In the equation, $\tilde{p}$ is an expert estimation of the distribution of the positive examples within the unlabeled dataset, and $\hat{p}_\theta$ is the model's posterior predictions on the unlabeled data. The Kullback-Leibler (KL) divergence is used to find the difference between the distributions, multiplied by the regularization constant, λ^U. As the function is optimized, the KL-divergence is minimized, thereby bringing the posterior predictions on unlabeled data into line with the expert's estimation of the true distribution.

$$O(\theta) = \sum_{i}^{N_p} \log p_\theta(y_i|x_i) - \lambda^U D(\tilde{p}||\hat{p}_\theta^{unlab}) - \lambda^{L^2} \sum_{k} \theta_k^2 \tag{3}$$

The gradient for the Label Regularization term to be optimized is:

$$\frac{\partial}{\partial \theta_k} D(\tilde{p}||\hat{p}_\theta) = \frac{1}{N_u} \left(\frac{1-\tilde{p}}{1-\hat{p}_\theta} - \frac{\tilde{p}}{\hat{p}_\theta} \right) * \\ * \sum_{i=1}^{N_u} p_\theta(y_i=1|x_i)(1-p_\theta(y_i=1|x_i))x_{i,k} \tag{4}$$

The gradient shows that the more certain (closer to 0 or 1) the estimation for an individual instance, $p_\theta(y_i = 1|x_i)$, the less it will impact the gradient and thereby change θ. The optimized θ represents a decision surface where $\hat{p}_\theta$ matches the expert provided $\tilde{p}$, and most cleanly separates the positives and the negatives that exist in the unlabeled data.

C. Applications in Android Security

The reason we use LR-LR in our study is because of the uncertainty stemming from weak ground truth data. As noted by Kantchelian [10], most anti-virus vendors tend to bias towards minimizing false positives at the expense of introducing false negatives. In our case, the benign dataset, as scanned by VirusTotal, could likely include apps whose labels in the coming weeks or months could change to malware. Even then, many apps will go undetected by the majority of VirusTotal scanners [5]. Some approaches [10] will only include apps that have been labeled as benign for a set amount of time, attempting to ensure label certainty by delaying introduction into the training dataset.

We believe that LR-LR is a superior option as it allows us to include the maximum amount of data available while decreasing the probability of training over incorrect labels. Instead of attempting to learn from benign apps whose labels could change with time, we instead use LR-LR to learn from positive and unlabeled instances, thereby making us more resilient to those malicious apps that VirusTotal scanners may have missed. As studied previously [10] and verified in Section IV-B, the number of apps whose label is changed from malware to benign, or anti-virus false positives, is exceptionally small. Thus, the precision of the anti-virus products ($Precision = \frac{TP}{TP+FP}$) for malware is exceptionally high, while the precision of benign apps is not nearly as high. Therefore, the only data we can be highly certain of are our malware (positive) examples.

While uncertain of the benign labels, the ability to include these instances in LR-LR as unlabeled examples provides us the ability to introduce additional data that will help in finding an optimal θ as described previously. This is preferable to single-class anomaly detection approaches that only train over positive data. Single-class classifiers, like one-class SVM, have been shown to not perform as well as LR-LR, namely due to the smaller amount of data available to the classifier for training [17].

IV. Experimental Setup

A. Base Data Set

For our experiments we utilized over a million Android apps from a variety of sources. The bulk of our apps were from the Google Play Store via the PlayDrone project [20]. Additionally, we collected 35K malicious apps from VirusShare and 24K malicious apps from Arbor Networks. We verified the authenticity of all labels in the manner described in the following section.

After collecting our base app set, we extracted semantic features mainly relating to permissions, APIs, and intents utilized by the application. APIs can be strong indicators of potential maliciousness. For instance, if a malicious app wants to collect the user's GPS location, they would have to leverage the API by calling the *getCellLocation* function on the *TelephonyManager* class. Likewise, to access certain critical APIs, Android requires that an app garner explicit permission from the user. Those permissions are listed in a metadata file in the APK, and to call the GPS API, an app must have the *ACCESS_FINE_LOCATION* permission. Finally, intents can be used to communicate between apps and with the underlying system. An intent we looked for in the manifest file and used as a feature was *ACTION_PACKAGE_INSTALL*, which attempts to use the Android OS package installer to install an APK. The presence of these types of permissions, APIs, and intents is required in the code for them to effectively execute their malicious mission and, hence, make strong features that together can show patterns that help a classifier to decide if the app is malicious or not. For more discussion of the feature set we utilize and the rationale behind it, see [18], which leverages the same set.

To extract these features, we utilized APKTool [2] to decompile and perform string searches for these APIs, permissions, and intents. As feature extraction can be on the order of around a few seconds per app, for over a million apps this would take several weeks. As such, we utilized a High Performance Computing Cluster to parallelize up to hundreds of cores at a time to speed up the feature extraction process.

B. Label Preparation

As noted in Section II-A, having accurate ground truth is critical to creating a successful machine learning approach. Not only does accurate ground truth improve classifier performance [18], it also ensures that classifiers built would perform accurately on real world data. To determine a label for each app in our dataset, we submitted the MD5 checksum of the binary to VirusTotal [21] and retrieved the count of anti-virus solutions flagging it as malware. We collected these reports for each app in our dataset both in April 2015 and in March 2016. We stored the feature vector along with both its 2015 and 2016 labels in a MySQL database, allowing us to dynamically create and vary datasets along different label confidences and years. Unless otherwise specified, we used the 2016 labels for our experiments. For determining a label, we define a *high-confidence positive app* as an app that has 10 or more VirusTotal anti-virus solutions marking it as malware. For the benign label we require that zero anti-virus solutions have marked it as malware. We note that between our 2015 and 2016 crawls of the anti-virus reports, 1682 apps changed labels, with 1681 having a label change from benign to malware and 1 changing from malware to benign. We note this matches the intuition provided in Kantchelian [10]. Apps that exist between the realm (e.g., less than 10 and more than 0 malware flags), are not used as we seek to have a clean control dataset. We seek a clean dataset so that we can effectively both test the classifier accuracy as well as introduce noise, neither of which would be possible with a high amount of accuracy if the middle, uncertain apps were included.

C. Dataset Noise

Throughout the experiments, we utilize a concept of label noise. As discussed in the previous section, we attempt to leverage ground truth to create as pure a dataset as possible. We do this so that we can systematically introduce noise in a measurable way, representing the noise that is inherent in the dataset due to poor ability to construct ground truth and the inherent inaccuracies in anti-virus software discussed in Section II-A. We represented this by artificially introducing a certain number of positive apps in the unlabeled set, given that it is much more common for apps to be mislabeled as negative, when actually positive, than the other way around. Specifically, if we denote by U_p the set of positive apps to be injected in the unlabeled dataset, and by U_n the set of negative apps in the unlabeled dataset, then

$$|U_p| = \frac{noise * |U_n|}{1 - noise} \tag{5}$$

where $noise$ is a parameter manually set, which represents the fraction of positive apps we place in our unlabeled set.

D. Spark

For conducting our experiments on a large dataset, we leveraged Apache Spark. Spark is a platform for large-scale data processing. Similar to MapReduce, Spark excels in distributed computing. Spark, however, is more effective for many types of data processing, including machine learning, due to the fact that its architecture focuses on in-memory computation, which is critical due to the massive size of the dataset we are working with. Thus, for performance reasons and ease of modification we chose Spark, and more specifically, MLLib as our machine learning framework of choice. We implemented Label Regularized Logistic Regression as a variant of the MLLib-provided Logistic Regression using L-BFGS optimization. Furthermore, we leveraged Spark SQL to dynamically create and vary different selection criteria for creation of experimental training and test sets. Our code implementation is freely available on Github for others to build upon[3].

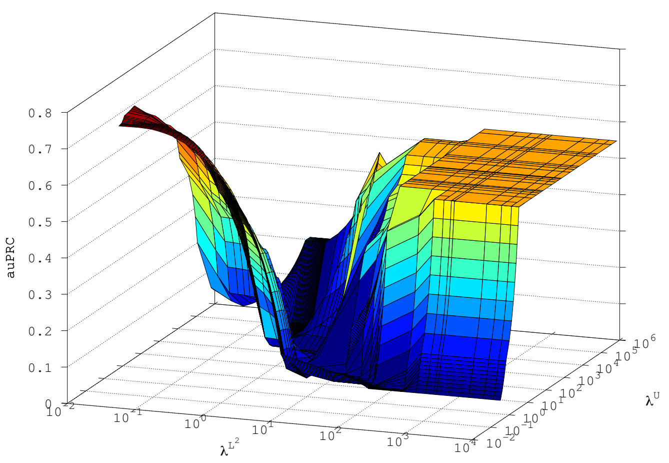
E. Parameter Tuning

We have three main constants in the LR-LR optimization function that we tuned for best performance. To find the best values for the two regularization parameters, we performed a grid search over values of λ^U and λ^{L^2} to find the values that result in the highest area under Precision-Recall curve (auPRC). These results can be seen in Figure 1. For λ^U, the parameter for the label regularization term, we used $\lambda^U = 1$, as verified in Figure 1. For λ^{L^2}, the parameter for L^2 regularization, we used Spark's default value of $\lambda^{L^2} = 0.0$, however this was verified in our parameter tuning. We note that our findings for the best λ^U and λ^{L^2} are quite different than in previous works [13], [17] that suggested much higher values for both parameters. Finally, we varied $\tilde{p}$, the expected

[2]http://ibotpeaches.github.io/Apktool/

[3]https://github.com/jdeloach/mlbSparkExperiments

Fig. 1: LR-LR Performance at varied λ^U and λ^{L^2}.



distribution between positive and negative in the unlabeled dataset, per experiment, as it will be seen in experiments.

F. Supervised Baseline Algorithms

We compare the one-class semi-supervised approach against the default Spark implementations of Naïve Bayes, Logistic Regression, and Support Vector Machines (SVM). We choose these specific classifiers as they are commonly used in the Android security machine learning space and we wanted to see how they will compare against the one-class semi-supervised approach, LR-LR. The supervised algorithms use the high quality positive data that the semi-supervised approach is using as positive examples, and the noisy "unlabeled" data (which contains some positive instances) as negative examples. Logistic Regression provides the most direct comparison with LR-LR. The only distinction between Logistic Regression and LR-LR is the fact that LR-LR does not make use of noisy negative data, as the Logistic Regression does. Instead, LR-LR is using the noisy negative data as unlabeled data to regularize the classifier derived from positive-only data. SVM is an effective, state-of-the-art classifier, but generally needs extensive tuning for achieving best results. We do not tune SVM and only use the default Spark implementation.

G. Evaluation Criteria

For our evaluation criteria, we leverage the area under the Precision-Recall curve (auPRC). As noted for largely imbalanced data sets [8], and verified for Android security data sets [18], auPRC is a better metric due to the large imbalance between benign and malicious apps in the dataset. Furthermore, we leverage 5-fold cross-validation for all of our experiments. Cross-validation allows us to incrementally train and test on all instances in the dataset and thus measure the generalization ability of the classifiers across the entire dataset. Additionally, we utilize the F1-Score in one experiment, and discuss the rationale behind using it in Section V-C.

V. Experiments

A. Research Questions

To motivate our exploration of the one-class semi-supervised approach, LR-LR, we articulate a few research questions. Namely, we want to understand what potential advantages LR-LR may have over supervised approaches when considering two of the biggest difficulties in machine learning approaches for Android security: lack of ample ground truth and also lack of accurate ground truth. To explore these two areas, we form three research questions:

1) Between supervised learning and semi-supervised learning from positive data and unlabeled data, which approach is better when working with inaccurate ground truth, i.e., when the available labeled data contains noise?
2) Between the two approaches, which one is the most effective in terms of predicting future labels?
3) How many labeled malware apps do we need to garner strong classification performance?

To answer these three questions, we devised and executed the three experiments described below.

B. Noisy Ground Truth

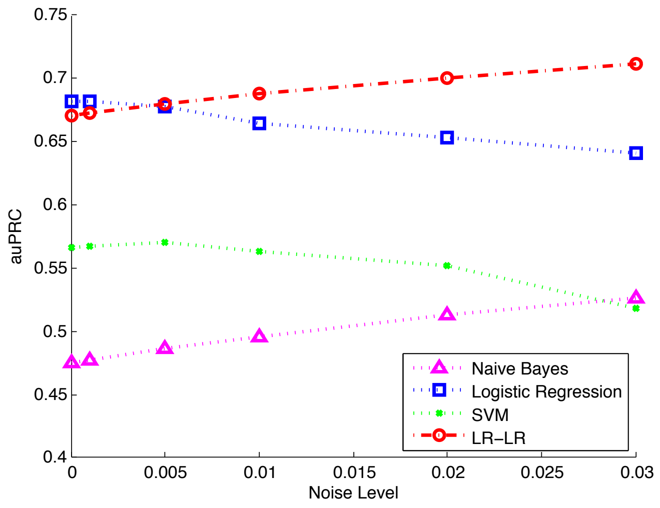
Whether experimental ground truth data is based upon the assumption of all apps collected from Google Play being benign, or the validity of VirusTotal results, incorrect labels will exist in datasets. This noise is inevitable, in the real world and in experimental setups. As we know there is noise inherent in even our cleanest datasets, we attempt to find out how noise affects classifier performance and what are the best methods for dealing with noise in datasets.

To do this, we use a dataset of roughly 50K labeled positive (L_p) apps combined with an "unlabeled" dataset comprised of around 940K negative apps (U_n) and a varied number of positive apps in the unlabeled dataset (U_p) as defined in Equation (5) as a function of the noise level. As we vary the noise level, we set $\tilde{p}$ to be equal to noise as it represents the expert prediction of the amount of positive apps in the unlabeled dataset, *e.g.*, exactly what noise is intended to introduce. In the case of the supervised classifiers, we will provide all "unlabeled" apps as noisy negative apps representative of the false negative class labels that cause malware to be marked as benign and, therefore, present in benign datasets.

For evaluation, we leverage 5-fold cross-validation. Within the training folds, we use L_p and $U_p + U_n$ as the positive and unlabeled/negative examples, respectively. In the held-out test fold, we evaluate over the true labels, e.g. L_p and U_p are positives instances, while U_n contains negatives instances. We then compute auPRC for each fold, and average the 5-folds to yield a composite auPRC for a given noise level.

We compare LR-LR with supervised Logistic Regression, SVM and Naïve Bayes in Figure 2. As can be seen in Figure 2, LR-LR not only has the best performance at nearly all levels of noise, but also performs better as noise is increased. This can be attributed to the fact that LR-LR can learn from unlabeled data, and a higher level of injected noise results in more

Fig. 2: Algorithm performance when varying the noise level



unlabeled data that can be utilized by LR-LR to refine and improve its decision boundary. While our experiments only used as high as a 3% noise rate, we can assume a continued strong performance as noise increases upwards of 7%, the estimate for the malware rate in the Google Play Store[5].

Among the three supervised classifiers, Logistic Regression is the best, being slightly superior to LR-LR for datasets without noise. However, as the noise increases, the performance of Logistic Regression decreases. The next best supervised classifier is SVM, whose performance also decreases with the amount of noise. It is interesting to note that the performance of Naïve Bayes improves with the amount of noise. We speculate this is due to the fact that Naïve Bayes may be skewed towards estimating positives, and the introduction of positive noise in the train and test sets means that it gets less predictions wrong in the testing phase.

C. Predicting the Future

As previously discussed, labels, especially when first given, are often inaccurate. In many instances within a year, labels for an app will change. A vast majority of the time a label will change, it will change from benign to malicious. For each of the apps in our dataset, we have two sets of class labels: one from 2015 and one from 2016. To be considered malicious, we require that at least 10 scanners mark it as so. For benign, we require that zero scanners mark it as malicious. In our dataset, it happens 1681 times that an app goes from benign to malicious, where the opposite flow from malicious to benign only happens one time. The anti-virus precision with respect to malware is exceptionally high, making this the ideal class to learn a decision boundary from as opposed to the less clear benign class. Due to the fact that a supervised classifier will train over both benign and malicious samples, it will reinforce these incorrect labels into the classifier as it learns those 1681 actually malicious apps as benign. We hypothesize that LR-LR will perform better at predicting the correct label, mainly due to the fact that it only trains over apps labeled as malware.

To experiment with this, we train the three supervised classifiers, and LR-LR (with $\tilde{p} = .01$) over our dataset using the 2015 class labels. We then test in two parts. First, we train a classifier on 4 folds with 2015 class labels and evaluate it using auPRC on the fifth, left-out fold using 2016 class labels. Second, we want to additionally focus on finding the best classifiers for detecting those apps with an incorrect label. While auPRC includes the accuracy of the labeling for the 1682 changed instances in its composite measure, the small number of apps that changed labels (i.e., 1682) were insignificant in the million plus apps that the dataset contains. Furthermore, thresholding auPRC over just the changed apps at various values displays an overly-optimistic view due to the fact that there is only one false positive as compared to 1681 anti-virus false negatives, therefore we need a different metric. We use the $F1$-score due to the substantial imbalance of having only one negative in the 1682 apps, precision can be very misleading as there is a maximum of 1 false positive.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

As such, we compute the $F1$-score directly over just the 1682 apps with a changed label in our dataset, using the standard threshold of 0.5. The results for both auPRC and $F1$-score are shown in Table I.

Classifier	auPRC	2016 $F1$-score
Naive Bayes	0.490	0.371
Logistic Regression	0.717	0.164
SVM	0.628	0.097
LR-LR	0.701	0.309

TABLE I: Using 2015 class labels to predict 2016 labels

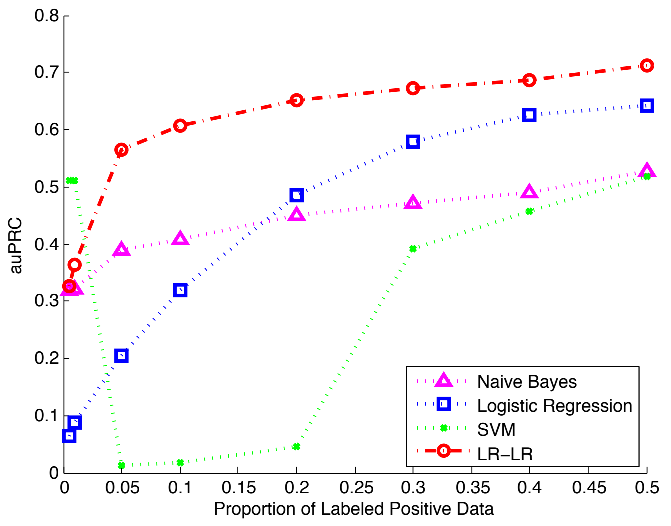
We note that while an $F1$-score of .309 is not outstanding, it is nearly double that of the traditional Logistic Regression. When considering that we trained over all samples, including those with mislabeled labels, the $F1$-score functionally represents the classifiers ability to determine an inaccurate label. While LR-LR did not have the best $F1$-score or auPRC, we feel it is the best balance between overall performance, and performance when considering mislabeled instances. The focus of the experiments is working with noisy and mislabeled data, and for that, LR-LR is the best all-around solution. Furthermore, while Naïve Bayes does have a strong $F1$-score, we note it has a very poor auPRC score, suggesting that in general it has a large amount of false positives, e.g. it will be overly and inaccurately inclined to label an app as malware.

One other interesting result of this experiment is the fact that while we did not inject any noise as in previous experiments, we saw the best LR-LR performance at $\tilde{p} = 0.01$ and not $\tilde{p} = 0.0$. We believe this can be attributed to the fact that even while we have vetted our benign apps via VirusTotal, we know that in the future some of these same apps will be relabeled as malicious. As such, there exists natural noise in our dataset. Unfortunately, we have no way of verifying this hypothesis short of waiting for updated labels or manually verifying malicious apps.

D. Minimizing Necessary Ground Truth

The main benefit of semi-supervised learning techniques is the ability to learn off of unlabeled data. To experiment with this, we explore how LR-LR and the three supervised algorithms perform as we vary the amount of labeled positive data in the training set. We provide the amount of labeled data as a proportion of the total high-confidence malware (10 or more A/V scanners) we have in our database. We vary through 0.005, .01, .05, .1, .2, .3, .4, and .5 of the total roughly 97K malware we have. Our unlabeled dataset has a constant noise factor of .02 in these experiments. As we use more positive data in the training, there will be more positive data also in the test as we distribute the instances equally between train and test, so in effect, when increasing the amount of labeled data, we also change the distribution of the test data from a smaller ratio to a larger ratio.

Fig. 3: Algorithm performance when varying the amount of labeled positive training data



As seen in Figure 3, LR-LR is comparatively more effective in situations containing a low amount of labeled positive data. This is to be expected as LR-LR is uniquely able to learn from the unlabeled data and grow its knowledge, even with a limited starting point. Also as expected, when the amount of positive labeled data is high, LR-LR and vanilla Logistic Regression have very similar performance, as the benefits offered by the inclusion of unlabeled data taper off.

The key insight here that makes LR-LR comparatively advantageous is in the scenario of less common or newer malware. In the case of zero day malware or even malware where not many samples from the family have been obtained, LR-LR's ability to learn from unlabeled data results in higher performance than traditional supervised classifiers. Furthermore, when combined with the rationale from Kantchelian [10] that false negatives are high, there may only be one or two samples from a family that have been labeled as malware and included in the malware training set. The rest of the samples, not labeled as malicious yet, would be considered unlabeled in LR-LR as opposed to being classified as benign, whereby LR-LR would be able to utilize those unlabeled samples to improve classifier performance. A supervised classifier, on the other hand, would consider those mislabeled malware as benign, harming classifier performance.

VI. Related Work

Supervised Android Security. Core works in the supervised Android security area include Drebin [1] which extracts over half a million features relating to both code and metadata attributes and reports very strong performance with SVM. Many of their features included attributes like the names of individual Android Activities within an app, and URLs, many of which were only used once or twice, leading to a very large, and very sparse feature space.

DroidSIFT [24] uses a more computationally-expensive method by extracting features from API dependency graphs and comparing against malware and benign app databases for similar flows. Then a Naive Bayes classifier is used to classify new unseen apps as either benign or malicious.

More recent approaches in supervised learning for Android security in the past year or so have become more sophisticated. One recent work by Sheen et al. [19] focused on utilizing multi-stage and multi-expert ensemble learning. They work with a rather small dataset with less than 2,000 apps and use k-NN, Logistic Regression, and Random Forrest as their classifiers. Another approach by Chuang et al. [6] uses the concept of fused classifiers based off of different sets of features for the same apps. The two sets correlate to the most common APIs for malware apps and benign apps, thereby making classifiers for each, then fusing them together to make one decision. The approach uses the SVM classifier and a dataset of around 6,000 benign apps and 3,400 malicious apps.

Roy et al. [18] explored six research questions derived from current techniques being used by peer works utilizing machine learning for Android app vetting. From these research questions, best practices were extracted including insights about the importance of using real-world class imbalances, effective classifier metrics, and the value of high quality ground truth, among others.

Semi-supervised Security. Only one main work exists in the area of semi-supervised learning for Android security [12]. It, however, does not make use of the key insights of the high precision of positive labels and comparatively low precision of benign apps as it utilizes both positive and negative app labels combined with a tri-training algorithm leveraging Support Vector Machines as the base algorithm. Wang et al. [22] focus on policy analysis and enforcement of the underlying Android operating system using semi-supervised learning on audit logs. Martin et al. [14] focus on checking app behavior against its description in the app store by leveraging self-training. A generic malware detection approach by Bazrafshan et al. [2] focuses on extracting op-code sequences from executables and uses the sequence frequency for a semi-supervised algorithm, Local and Global Consistency. Finally, Label Regularized Logistic Regression [17] has been used to detect security events (e.g. DDoS, hacking, etc.) from tweets on Twitter using a small amount of seed events and a large amount of unlabeled tweets. Our work is the first work to investigate a one-class semi-supervised approach for Android malware detection. Furthermore, we leverage a semi-supervised approach (LR-LR) specific to the strengths and weaknesses of the Android app data. We combine the fact that positive apps are highly certain by only calculating the log likelihood over positive instances, while leveraging the big data elements with the inclusion of unlabeled apps.

VII. Conclusions and Future Work

In this work we leveraged one-class semi-supervised learning for the first time in the Android malware detection field. We carefully observed the strengths and weaknesses in current ground truth in the Android space and how these can be remedied with a one-class semi-supervised approach. Android ground truth, unlike almost all other categories of ground truth has inherent flaws due to the ability of changing class labels over time and the inability to ever know for certain if an app is actually benign. Using these understandings about Android ground truth, we apply Label Regularized Logistic Regression specifically to capitalize on this dynamic. We find in our three separate research questions that LR-LR outperforms other approaches, leading us to believe it is a promising area of further research in the Android security community.

In future work, we would like to continue exploring techniques to deal with the unique difficulties in detecting Android malware, namely in the arena of finding high quality ground truth class labels. We will continue to monitor changing class labels and will be able to form more holistic understandings of how anti-virus ground truth changes and evolves.

Furthermore, we will attempt to add new dimensions of sources when creating ground truth labels. One potential avenue for this is leveraging social media data such as Twitter which has been shown to give some insight on security events that may be happening [17]. We would attempt to catalog the social media posts, map them to a specific app, and then use a form of sentiment analysis to supplement existing anti-virus ground truth.

Acknowledgment

Part of the computing for this project was performed on the Beocat Research Cluster at Kansas State University, which is funded in part by the NSF grants MRI-1429316 and CC-IIE-1440548. This project is partially supported by the National Science Foundation under Grant No. 1622402. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. Drebin: Effective and explainable detection of Android malware in your pocket. In *Proc. of the NDSS*, 2014.

[2] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh. A survey on heuristic malware detection techniques. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*, pages 113–120. IEEE, 2013.

[3] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100. ACM, 1998.

[4] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck. MAST: Triage for market-scale mobile malware analysis. In *Proc. of the WiSec*, 2013.

[5] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 659–674, 2015.

[6] H. Y. Chuang and S. D. Wang. Machine learning based hybrid behavior models for Android malware analysis. In *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on*, pages 201–206, Aug 2015.

[7] F. Cozman and I. Cohen. Risks of semi-supervised learning: How unlabeled data can degrade performance of generative classifiers. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*. MIT Press, 2006.

[8] J. Davis and M. Goadrich. The Relationship Between Precision-Recall and ROC curves. In *Proc. of the ICML*, 2006.

[9] Google. Android security - 2014 year in review. http://bit.ly/1aoqsx0, 2014.

[10] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, AISec '15, pages 45–56, New York, NY, USA, 2015. ACM.

[11] Lookout. 2014 mobile threat report. http://bit.ly/1fktFwe, 2014.

[12] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun. Active semi-supervised approach for checking app behavior against its description. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 2, pages 179–184. IEEE, 2015.

[13] G. S. Mann and A. McCallum. Simple, robust, scalable semi-supervised learning via expectation regularization. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 593–600, New York, NY, USA, 2007. ACM.

[14] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A survey of app store analysis for software engineering. *RN*, 16:02, 2016.

[15] K. P. Murphy. *Machine learning: a probabilistic perspective*. 2012.

[16] RiskIQ. RiskIQ Reports malicious mobile apps in Google Play have spiked nearly 400 percent. http://bit.ly/1siyWOS, 2014.

[17] A. Ritter, E. Wright, W. Casey, and T. Mitchell. Weakly supervised extraction of computer security events from Twitter. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 896–905, New York, NY, USA, 2015. ACM.

[18] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara. Experimental study with real-world data for Android app security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 81–90, New York, NY, USA, 2015. ACM.

[19] S. Sheen and A. Ramalingam. Malware detection in Android files based on multiple levels of learning and diverse data sources. In *Proceedings of the Third International Symposium on Women in Computing and Informatics*, pages 553–559. ACM, 2015.

[20] N. Viennot, E. Garcia, and J. Nieh. A measurement study of Google Play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, pages 221–233. ACM, 2014.

[21] Virus Total. https://www.virustotal.com/. Accessed: December 2014.

[22] R. Wang, W. Enck, D. Reeves, X. Zhang, P. Ning, D. Xu, W. Zhou, and A. M. Azab. Easeandroid: Automatic policy analysis and refinement for security enhanced android via large-scale semi-supervised learning. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 351–366, 2015.

[23] S. Wu, P. Wang, X. Li, and Y. Zhang. Effective detection of Android malware based on the usage of data flow APIs and machine learning. *Information and Software Technology*, 75:17 – 25, 2016.

[24] M. Zhang, Y. Duan, H. Yin, and Z. Zhao. Semantics-aware Android malware classification using weighted contextual API dependency graphs. In *Proc. of ACM CCS*, 2014.

[25] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.