Android Malware Detection Using Network-based Approaches

by

Emily Alfs

B.S., Doane University, 2016

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Mathematics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2018

Approved by:

Major Professor
Nathan Albin

Copyright

Abstract

This thesis is focused on the use of networks to identify potentially malicious Android applications. There are many techniques that determine if an application is malicious, and they are ever-changing. Techniques to identify malicious applications must be robust as the schemes of creating malicious applications are changing as well. We propose the use of a network-based approach that is potentially effective at separating malicious from benign apps, given a small and noisy training set.

The applications in our data set come from the Google Play Store and have been scanned for malicious behavior using Virus Total to produce a ground truth dataset. The apps in the resulting dataset have been represented as binary feature vectors (where the features represent permissions, intent actions, discriminative APIs, obfuscation signatures, and native code signatures). We use the feature vectors corresponding to apps to build a weighted network that captures the "closeness" between applications. We propagate labels, benign or malicious, from the labeled applications that form the training set to unlabeled applications (which we aim to label), and evaluate the effectiveness of the proposed approach in terms of precision, recall and F1-measure.

We outline the algorithms for propagating labels that were used in our research and discuss the fine tuning of hyper-parameters. We compare our results to known supervised learning algorithms, such as k-nearest-neighbors and Naive Bayes, that can be used to learn classifiers from the training labeled data and subsequently use the classifiers to label the unlabeled test data. We discuss potential improvements on our methods and ways to further this research.

Table of Contents

List of Tables

Acknowledgments

This project is partially supported by the National Science Foundation under Grant No. 1717871. 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.

Chapter 1

Introduction

1.1 Details of the Problem

The purpose of this thesis is to efficiently and effectively apply classification techniques to Android applications; more specifically, the goal is to classify applications from the Google Play Store as malicious or benign. Malicious applications are referred to as malware, a shorthand of malicious software.

Android applications are produced and published at a very high rate in the Google Play Store. Therefore, efficient and effective malware detection is necessary. One major hurdle with this data, Android applications, is the amount of noisy data points. Malware can have many different purposes and perhaps the most prolific is obtaining personal information. This can include passwords, credit card information, and much more which can be used for malicious purposes. With the threat of information such as this getting into the wrong hands, finding a solution to this problem is a necessary one.

1.2 Differences in Methodology

Correctly labeling data is a very important problem in general, thus many people have studied labeling techniques in many different applications. Previous research into malware

detection specifically has taken many different forms. Machine learning has been used in malware detection by DeLoach et al. (2016); Roy et al. (2015) and many others. Social networks have also given researchers new avenues to detect malware (Ni et al., 2016; Chen et al., 2015).

Our approach is to use the properties of graphs to determine "closeness" of applications and propagate labels. We based our definition of closeness on the Hamming distance between two applications permissions vectors. The smaller the distances, the closer (or more similar) the apps. This permissions vector was extracted and derived from previous research done by Roy et al. (2015).

Once we have the distance between all of the applications, we create a graph and propagate the labels. Label propagation is a standard technique that can be applied in various ways. Alternative methods to label propagation include k-nearest-neighbors (k-NN) and Niave Bayes (NB). We apply an iterative method, which simplifies to a linear equation, to propagate the labels. We compare our results against k-NN and NB as baselines. respectively. We test Label Propagation and Label Spreading as implemented in Python through scikit-learn (Pedregosa et al., 2011).

Chapter 2

Label Propagation Algorithms

We use a mix of two algorithms proposed by Zhu and Ghahramani (2002) and Zhou et al. (2004). These algorithms are described in Sections 2.1 and 2.2, respectively, while our proposed variant is described in Section 2.3.

2.1 Hard Clamping Algorithm

The hard clamping algorithm described in this section was proposed by Zhu and Ghahramani (2002). Let $(x_1, x_2)...(x_l, y_l)$ be our labeled data where the set of $Y_l = \{y_1, ...y_l\}$ are the class labels. The number of classes, c, is assumed to be known. In our specific case, the number of classes is 2, class 1 being benign and class 2 being malicious. We also have a set of unlabeled data, $(x_{l+1}, y_{l+1})...(x_{l+u}, y_{l+u})$, for which we are trying to assign labels. Let $Y_u = \{y_{l+1}, ...y_{l+u}\}$ be the set of unobserved labels and $X = \{x_1, ...x_{l+u}\}$ be the data points.

For our purposes, we have a graph $G = (V, E, W)$, where V is the set of nodes, specifically the applications, E is the set of edges, and W is the weight matrix where $w_{i,j} \in W$ is the weight between nodes i and j. This weight is given by the similarity of the feature vectors associated with both of the applications; thus the more similar applications are, the larger the edge weight.

We define a $(l + u) \times (l + u)$ matrix T. This is a probabilistic transition matrix where

$T_{ij} = P(j \to i) = \frac{w_{ij}}{\sum_{k=1}^{l+u} w_{kj}}$. We can think of this as the probability that we will move from node j to node i. Then we define matrix Y which is a $(l+u) \times c$ matrix where each row, i, corresponds to the probability distribution of node x_i.

We start by initializing Y for the first l nodes that we know are labeled. For example, if we know node i is a malicious app, then $Y_i = [0\ 1]$, and if node j is benign, then $Y_j = [1\ 0]$. The values for the rows of Y which correspond to unlabeled nodes do not matter when we initialize Y. We chose to use $Y_i = [0.5\ 0.5]$ as the initial values for unlabeled nodes. The algorithm goes as follows:

1. Propagate $Y \leftarrow TY$
2. Row-normalize Y
3. Clamp the labeled data. Repeat step 1 until Y converges

To expand, in step two, we want to row normalize Y as this represents a probabilistic matrix, thus each row must sum to one. In step 3, by clamping the labeled data, we are setting the values of Y for the labeled points back to the initial values. Thus we are ensuring that our nodes, which have a ground truth associated with them, do not change their labels. The final labeling for application i is given by the maximum of row Y_i. Zhu and Ghahramani (2002) prove that this iterative method converges to Equation 2.1 where $\overline{T}$ is the matrix T row-normalized split into four blocks.

$$Y_U = (I - \overline{T}_{uu})^{-1} \overline{T}_{ul} Y_L \tag{2.1}$$

This split happens such that $\overline{T}_{ul}$ is the bottom left block of $\overline{T}$ composed of rows $l+1$ through $l+u$ and columns 1 through l. Similarly, $\overline{T}_{uu}$ is the bottom right block of $\overline{T}$ composed of rows $l+1$ through $l+u$ and columns $l+1$ through $l+u$.

2.2 Soft Clamping Algorithm

The soft clamping variant of the label propagation algorithm was proposed by Zhou et al. (2004), and will be described in this section. Given a set of points $X = \{x_1, ..., x_l, x_{l+1}..., x_n\}$, where the first l points are labeled and the remaining are unlabeled, let Y be an $n \times c$ matrix. Entry $Y_{i,1}$ is the probability of application i being benign and and $Y_{i,2}$ the probability of application i being malware. We build an affinity matrix W, which in our case is the matrix where the w_{ij} entry is the edge weight between application i and j and we define $w_{ii} = 0$. In Section 3.1, there are more details to how this weight is found. By this definition, W is a symmetric matrix. We construct $S = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$, where D is a diagonal matrix such that D_{ii} is the sum of the elements of the ith row of W. Entry-wise we have $S_{ij} = \frac{W_{ij}}{\sqrt{D_{ii}D_{jj}}}$

Let $F = [F_1^T, ..., F_n^T]^T$ where each F_i is an $1 \times c$ matrix that corresponds to the ith application's classification. The algorithm then iterates

$$F(t+1) = \alpha SF(t) + (1-\alpha)Y \tag{2.2}$$

until convergence. In this algorithm, $Y = F(0)$, our initial labels, and α is a scalar. We can interpret α as the amount of information we are taking from neighboring applications and the application's own initial labeling. Zhou et al. (2004) prove that equation 2.2 converges to the following where F^* is the solution

$$F^* = (1-\alpha)(I - \alpha S)^{-1}Y \tag{2.3}$$

To get the final label for application i, we look at row i of F^* and the label is chosen to correspond the column where the maximum occurs.

2.3 Proposed Variant for Label Propagation

As mentioned, the final algorithm we used is a combination of both soft and hard clamping. We build the matrix $S = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ as outlined in Section 2.2. We also define Λ as a

diagonal matrix with the same dimensions as S, where $\lambda_{i,i} = \alpha_i$. Note that α_i corresponds to the i-th application and $0 \leq \alpha_i \leq 1$ for all i. For applications in which we are confident on our initial labeling, such as the malicious applications that were verified by hand, we have a low α value and vice versa with applications that we are treating as unlabeled. In particular, applications whose labels we are confident in have $\alpha = 0$ and those that we are not confident in $\alpha = 0.9$. This is utilizing the hard clamping for our ground truth data and soft clamping for the unlabeled data. Our evaluation is based off of the iterative equation 2.2. We use instead the following.

$$F(t+1) = \Lambda S F(t) + (1 - \Lambda)Y \tag{2.4}$$

As with equation 2.2, we can prove that it will converge to something very similar to equation 2.3.

Theorem 2.3.1. *Iterating $F(t+1) = \Lambda S F(t) + (1 - \Lambda)Y$ will converge.*

Proof. Without loss of generality, we assume $F(0) = Y$, our initial labels. Applying the iterative equation, we get that

$$F(t) = (\Lambda S)^{t-1} Y + \sum_{i=0}^{t-1} (\Lambda S)^i (I - \Lambda) Y$$

In order to show convergence, we must show that $\rho(\Lambda S) < 1$ where ρ is the spectral radius.

$$\rho(\Lambda S) \leq \rho(\Lambda)\rho(S) \tag{2.5}$$

$$< 1 * 1 \tag{2.6}$$

$$= 1 \tag{2.7}$$

Therefore, $\rho(\Lambda S)$ converges which implies

$$\lim_{t\to\infty}(\Lambda S)^{t-1} = 0$$

and

$$\lim_{t\to\infty}\sum_{i=0}^{t-1}(\Lambda S)^i = (I - \Lambda S)^{-1}$$

Thus our algorithm converges to the following,

$$F^* = \lim_{t\to\infty} F(t) = (I - \Lambda S)^{-1}(I - \Lambda)Y \tag{2.8}$$

□

We also compare different assessment techniques. For both the hard-clamping and soft-clamping, the algorithms use the maximum of the two column values. Another method is to use class mass normalization initially mentioned by Zhu and Ghahramani (2002). We see further explanation in Bengio et al. (2006).

Let $y_{i,k}$ be the initial label of application i and let p_k be the prior probability of class k from the labeled data. We define p_k as the following:

$$p_k = \frac{1}{l}\sum_{i=1}^{l} y_{i,k}$$

The average of estimated weights for class k on the unlabeled data is given by:

$$m_k = \frac{1}{u}\sum_{i=l+1}^{l+u} \hat{y}_{i,k}$$

where $\hat{y}_{i,k}$ is the evaluated class from our hard clamping evaluation. Then the class normal-

ization occurs by taking $\arg\max_k w_k \hat{y}_{i,k}$ where:

$$w_k = \frac{p_k}{m_k}$$

Using this evaluation, we take into account the imbalance of data.

Chapter 3

Experimental Setup

3.1 Details of the Data

The data set that we are working with was curated by Roy et al. (2015). The total data set contains over one million applications but we work with a smaller subset. The total data set has approximately 18,000 known malicious applications which were verified manually by Wei et al. (2017). For consistency of sample size, we select a subset of 18,000 benign applications as well. We consider these subsets to be our ground truth set.

Associated with each of the applications, benign or malicious, is a feature vector. For all of these applications the feature vector has 471 binary entries which represent various features. These features were extracted and organized by previous research done by Roy et al. (2015). The features they extracted fall into the following categories: permissions, *intent actions*, discriminative APIs, obfuscation signatures, and native code signatures.

3.2 Experimental Process

In order to ensure the accuracy of our algorithm, we had the following incremental testing procedure shown in Table 3.1. We read the table as follows: in row one, our first incremental step, we started with 50 labeled malicious and 50 labeled benign applications, and 5,000

'unlabeled' malicious and 5,000 'unlabeled' benign. By 'unlabeled', we are referring to the fact that we know the ground truth for these applications but we have the program treat them as though they are unknown; allowing us to tell the true accuracy of our program. It is also worth noting that each step builds incrementally, such that in the second round of testing, the data points from the first round of testing are included.

Table 3.1: *The number of test instances in each step*

Labeled Malware	**Labeled Benign**	**Unlabeled Malware**	**Unlabeled Benign**
50	50	5000	5000
100	100	5000	5000
500	500	5000	5000
1000	1000	5000	5000
5000	5000	5000	5000

Once a round of testing was completed, we evaluated the precision and recall of each algorithm for the positive results. We consider positive results to be those applications which were assigned a label of malicious and negative results to be those which are assigned labels of benign. Precision is measured by the ratio of true positive to the total number of positive results, both true and false positive. Recall is measured by the ratio of true positives to the total of true positives and false negatives. We also looked at the F-measure which is $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$. We compared the results of the semi-supervised label propagation approach with the results of standard supervised algorithms, k-Nearest-Neighbors (k-NN) and Naive Bayes (NB).

The next steps are to change the ratio of malicious to benign. Up to this point, our ratio is 1:1, i.e. when we have 50 malicious applications there are 50 benign applications. We would like to experiment by changing this ratio to say 1:10, meaning for every benign application there are 10 malicious applications in our labeled and unlabeled set.

Chapter 4

Experimental Results

4.1 Comprehensive Balanced Results

We outline the results from the testing of our algorithm described in Sections 2.2 and 2.3 and compare these against other well known alternative label propagation algorithms, k-nearest-neighbors and Naive Bayes, Sections 4.1.3 and 4.1.4 respectively. Further, we test Label Propagation and Label Spreading as implemented in Python through scikit-learn (Pedregosa et al., 2011) in Sections 4.1.5 and 4.1.6 respectively.

4.1.1 Hard Clamping

In the case of hard clamping, as described in Section 2.2, there are no parameters that need tuning, as in soft clamping with α. We shuffled our data set a total of 5 times to get the average results of this technique to verify the results. In Table 4.1, we see the results of these tests, where LM is labeled malware, LB is labeled benign, UM is unlabeled malware, and UB is unlabeled benign. The last column shows the standard deviation of F-1 measure over the 5 samplings.

Table 4.1: *Hard Clamping Balanced Results*

LM	LB	UM	UB	Precision	Recall	F-1	SD of F-1
50	50	5000	5000	0.798	0.704	0.748	0.045
100	100	5000	5000	0.890	0.819	0.853	0.032
500	500	5000	5000	0.857	0.692	0.766	0.009
1000	1000	5000	5000	0.863	0.692	0.768	0.004
5000	5000	5000	5000	0.885	0.769	0.823	0.003

4.1.2 Soft Clamping with Fixed Alpha

In the soft clamping technique, we have the hyper-parameter α. We tested different values of this parameter to find an optimal value. We shuffled the data set 5 times to verify that the results weren't just a product of the order of the applications. The following tables show the average of those results and the fifth column shows the standard deviation of F-1 measure over the 5 samplings. Rows of tables where '-' are seen in the precision and F-1 column, we had no positive labels, true positive or false positive.

In Table 4.2, we see the average results for the set with 50 applications labeled malware, 50 labeled benign, 5000 unlabeled malware, and 5000 unlabeled benign. In this test phase, $\alpha = 0.1$ was the best value with an F-1 measure of 0.793.

Table 4.2: *50 labeled malware, 50 labeled benign, 5000 unlabeled malware, 5000 unlabeled benign*

Alpha	Precision	Recall	F-1	SD of F-1
0.1	0.937	0.690	0.793	0.047
0.2	0.942	0.683	0.791	0.047
0.3	0.946	0.666	0.781	0.048
0.4	0.952	0.651	0.771	0.056
0.5	0.959	0.627	0.756	0.069
0.6	0.968	0.587	0.726	0.088
0.7	0.978	0.496	0.650	0.123
0.8	0.990	0.337	0.476	0.226
0.9	-	0.094	-	-

Table 4.3, shows the average results for the set with 100 applications labeled malware, 100 labeled benign, 5000 unlabeled malware, and 5000 unlabeled benign. Again, we see the

best F-1 measure, 0.791, when $\alpha = 0.1$.

Table 4.3: *100 labeled malware, 100 labeled benign, 5000 unlabeled malware, 5000 unlabeled benign*

Alpha	Precision	Recall	F-1	SD of F-1
0.1	0.938	0.684	0.791	0.028
0.2	0.943	0.676	0.787	0.028
0.3	0.948	0.662	0.779	0.031
0.4	0.955	0.644	0.768	0.035
0.5	0.961	0.607	0.744	0.029
0.6	0.970	0.568	0.714	0.055
0.7	0.982	0.492	0.650	0.092
0.8	0.982	0.492	0.650	0.180
0.9	-	0.040	-	-

Table 4.4 shows the results for the set with 500 applications labeled malware, 500 labeled benign, 5000 unlabeled malware, and 5000 unlabeled benign. We see the best F-1 measure, 0.782, when $\alpha = 0.1$.

Table 4.4: *500 labeled malware, 500 labeled benign, 5000 unlabeled malware, 5000 unlabeled benign*

Alpha	Precision	Recall	F-1	SD of F-1
0.1	0.942	0.669	0.782	0.004
0.2	0.947	0.664	0.781	0.004
0.3	0.952	0.657	0.778	0.006
0.4	0.959	0.643	0.769	0.011
0.5	0.965	0.612	0.749	0.017
0.6	0.975	0.555	0.707	0.029
0.7	0.986	0.436	0.604	0.044
0.8	0.995	0.220	0.357	0.088
0.9	-	0.004	-	-

Table 4.5 shows the average results for the set with 1000 applications labeled malware, 1000 labeled benign, 5000 unlabeled malware, and 5000 unlabeled benign. We see the best F-1 measure, 0.780, when $\alpha = 0.1$.

Table 4.6 shows the results for the set with 5000 apps labeled malware, 5000 labeled benign, 5000 unlabeled malware, and 5000 unlabeled benign. We see the best F-1 measure, 0.780, when $\alpha = 0.1$.

Table 4.5: *1000 labeled malware, 1000 labeled benign, 5000 unlabeled malware, 5000 unlabeled benign*

Alpha	Precision	Recall	F-1	SD of F-1
0.1	0.938	0.668	0.780	0.003
0.2	0.943	0.663	0.778	0.005
0.3	0.948	0.655	0.775	0.006
0.4	0.948	0.655	0.775	0.011
0.5	0.964	0.608	0.745	0.014
0.6	0.973	0.535	0.660	0.024
0.7	0.986	0.424	0.593	0.036
0.8	0.995	0.201	0.333	0.063
0.9	-	0.002	-	-

Table 4.6: *5000 labeled malware, 5000 labeled benign, 5000 unlabeled malware, 5000 unlabeled benign*

Alpha	Precision	Recall	F-1	SD of F-1
0.1	0.939	0.668	0.780	0.001
0.2	0.945	0.664	0.780	0.001
0.3	0.952	0.657	0.778	0.002
0.4	0.958	0.642	0.769	0.007
0.5	0.964	0.616	0.752	0.006
0.6	0.974	0.535	0.691	0.006
0.7	0.986	0.428	0.597	0.009
0.8	0.995	0.206	0.342	0.015
0.9	1	0.001	0.003	0.001

4.1.3 KNN

With k-NN using 3 neighbors, we found the results shown in Table 4.7, where LM is labeled malware, LB is labeled benign, UM is unlabeled malware, and UB is unlabeled benign. The results using k-NN were better in F-1 than soft-clamping for all of the rounds of testing. However, the precision for soft-clamping and k-NN were comparable but the recall for soft-clamping was much lower than that of k-NN.

Table 4.7: *KNN Balanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
50	50	5000	5000	0.919	0.805	0.858
100	100	5000	5000	0.959	0.906	0.931
500	500	5000	5000	0.923	0.952	0.937
1000	1000	5000	5000	0.919	0.965	0.941
5000	5000	5000	5000	0.937	0.981	0.958

4.1.4 NB

With NB, we found the results shown in Table 4.8 where LM is labeled malware, LB is labeled benign, UM is unlabeled malware, and UB is unlabeled benign. The results using NB were better in F-1 than soft-clamping for all of the rounds of testing. Again like k-NN, the precision for soft-clamping and NB were comparable but the recall for soft-clamping was much lower than that of NB.

Table 4.8: *NB Balanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
50	50	5000	5000	0.827	0.722	0.771
100	100	5000	5000	0.913	0.837	0.873
500	500	5000	5000	0.875	0.741	0.803
1000	1000	5000	5000	0.890	0.746	0.811
5000	5000	5000	5000	0.910	0.826	0.866

4.1.5 Scikit Learn Label Propagation

In the scikit learn implementation of label propagation, one can choose from two kernels, k-NN and the radial basis function (RBF). The results for label spreading can be seen in Table 4.9.

As with k-NN and NB, the precision values for label propagation and soft-clamping were comparable but the recall for soft-clamping was much lower than that of label propagation.

Table 4.9: *Scikit Learn Label Propagation Balanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
50	50	5000	5000	0.979	0.579	0.727
100	100	5000	5000	0.979	0.759	0.855
500	500	5000	5000	0.983	0.904	0.942
1000	1000	5000	5000	0.982	0.928	0.955
5000	5000	5000	5000	0.984	0.962	0.973

4.1.6 Scikit Learn Label Spreading

The scikit learn implementation of label spreading claims to be more robust to noise than the label propagation implementation. This method differs in modification to similarity matrix. The results for label spreading can be seen in Table 4.10.

Table 4.10: *Scikit Learn Label Spreading Balanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
50	50	5000	5000	0.977	0.580	0.728
100	100	5000	5000	0.979	0.760	0.856
500	500	5000	5000	0.979	0.760	0.856
1000	1000	5000	5000	0.982	0.928	0.955
5000	5000	5000	5000	0.984	0.962	0.973

As with k-NN and NB, the precision values for label spreading and soft-clamping were comparable but the recall for soft-clamping was much lower than that of label spreading.

4.2 Initial Unbalanced Results

While testing of the unbalanced dataset is not complete, we do have some initial results. We have implemented the algorithm from section 2.3 using equation 2.8. For the values of α, we have selected $\alpha = 0$ for labeled applications and $\alpha = 0.9$ for unlabeled applications.

The intuition is that we are applying the hard-clamping to the applications which we are treating as labeled and encouraging change for the applications which we are treating as 'unlabeled'. Presently we are unable to properly label malicious applications as our program labels everything benign. We have been able to verify, using the iterative process outlined

in section 2.3, that our program giving the expected results. Areas of improvement can be found in section 5.2. We continued testing to compare KNN, NB, Label Propagation, and Label Spreading.

The results for KNN can be seen in Table 4.11, NB in Table 4.12, Label Propagation in Table 4.13, and Label Spreading in Table 4.14. Overall, Label Propagation, which implements the hard clamping method, and Label Spreading, which implements the soft clamping method, perform better than Naive Bayes. KNN does perform better than both Label Propagation and Label Spreading.

Table 4.11: *KNN Unbalanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
10	100	500	5000	0.929	0.21	0.343
25	250	500	5000	0.919	0.632	0.749
50	500	500	5000	0.89	0.698	0.783
250	2500	500	5000	0.92	0.894	0.907

Table 4.12: *Naive Bayes Unbalanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
10	100	500	5000	0.544	0.392	0.456
25	250	500	5000	0.449	0.652	0.532
50	500	500	5000	0.414	0.692	0.518
250	2500	500	5000	0.417	0.714	0.526

Table 4.13: *Scikit Learn Label Propagation Unbalanced Results*

LM	LB	UM	UB	Precision	Recall	F-1
10	100	500	5000	0.8	0.192	0.31
25	250	500	5000	0.925	0.466	0.62
50	500	500	5000	0.914	0.51	0.655
250	2500	500	5000	0.957	0.836	0.892

Table 4.14: *Scikit Learn Label Spreading Unbalanced Results*

LM	**LB**	**UM**	**UB**	**Precision**	**Recall**	**F-1**
10	100	500	5000	0.8	0.192	0.31
25	250	500	5000	0.925	0.466	0.62
50	500	500	5000	0.914	0.51	0.655
250	2500	500	5000	0.957	0.836	0.892

Chapter 5

Future Work and Conclusions

5.1 Future Work

There are many ways that we can alter the process to test for improvement. One of those would be introducing a threshold for the weights, meaning if the weight between two applications is less than a certain value, between zero and one, then don't connect the two applications. We could also weight each property differently. If there is a property that tends to be more malicious than others, we could weight those more heavily. Similarly, if there is a set of properties that are particularly nefarious, we could increment the weight if all of those appear. These would both potentially require more initial knowledge of the data.

Another appropriate next step would be to work more with the imbalanced data. We can try different values for the imbalance ratio, say 1:2. We will also test other known label propagation techniques to make a more thorough analysis on what is happening with the data points.

5.2 Conclusions

For the soft clamping method on balanced results, $\alpha = 0.1$ produced the best $F1$-measure. While the precision for all three methods were comparable, the recall for soft-clamping was

much lower than that of k-NN and NB. Initial results do favor k-NN and NB as techniques of labeling. We are hopeful that using the algorithm we have described and adapted, will show more positive results when it comes to unbalanced data.

Bibliography

Bengio, Y., Delalleau, O., and Roux, N. L. (2006). Label propagation and quadratic criterion. In Chapelle, O., Schölkopf, B., and Zien, A., editors, *Semi-Supervised Learning*, pages 193–216.

Chen, L., Hardy, W., Ye, Y., and Li, T. (2015). Analyzing file-to-file relation network in malware detection. In Wang, J., Cellary, W., Wang, D., Wang, H., Chen, S.-C., Li, T., and Zhang, Y., editors, *Web Information Systems Engineering – WISE 2015*, pages 415–430, Cham. Springer International Publishing.

DeLoach, J., Caragea, D., and Ou, X. (2016). Android malware detection with weak ground truth data. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3457–3464.

Ni, M., Li, T., Li, Q., Zhang, H., and Ye, Y. (2016). Findmal: A file-to-file social network based malware detection framework. *Knowledge-Based Systems*, 112:142–151.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Roy, S., DeLoach, J., Li, Y., Herndon, N., Caragea, D., Ou, X., Ranganath, V. P., Li, H., and Guevara, N. (2015). 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. ACM.

Wei, F., Li, Y., Roy, S., Ou, X., and Zhou, W. (2017). Deep ground truth analysis of current android malware. In Polychronakis, M. and Meier, M., editors, *Detection of Intrusions*

and Malware, and Vulnerability Assessment, pages 252–276, Cham. Springer International Publishing.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schlkopf, B. (2004). Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328. MIT Press.

Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. Technical report, School of Computer Science Carnegie Mellon University.