Project Report
Course Project : CIS 787 (Analytical Data Mining)
Professor: Dr. Reza Zafarani

Chirag Sachdev
680231131

Ransomware Classification

Introduction:

Ransomware Classification by analysis of Data Set containing opcodes for 6 major Ransomware families. The dataset for this project was acquired from the Cyber Science Lab at the University of Guelph. The dataset contains opcodes from 555 ransomware families.

Motivation:
In 2003 Buffer Overflow was ranked as the #1 most severe vulnerability, it is still considered as a sever vulnerability but as technology has progressed, malware has become more sophisticated and can cause damage in ways without establishing control over the systems. Ransomwares are one such category of malware where the user is locked out of their systems without a remote attacker monitoring. The two types of ransomware include Locking Ransomware and Crypto Ransomware. In locking ransomware, the victim is locked out of the System whereas in Crypto Ransomware, the data of the victim is encrypted using a strong symmetric/ asymmetric encryption algorithm. The victim is demanded with high amounts of cryptocurrency or the key used for encryption would be deleted.
Ransomware ranks as the most destructive malware as of 2019[1].
Being an aspiring cyber security specialist, I would like to use my skills of systems and combine it with my knowledge of the machine learning domain to find a prevention from ransomware attacks.

Prior Work:

The dataset obtained is a subset of the original research conducted at the Cyber Science Lab at the University of Guelph[2]. The publication titled *Know Abnormal, Find Evil: Frequent Pattern Mining for Ransomware Threat Hunting and Intelligence*[2], used frequential pattern data mining to match opcode sequences to registry events. The opcodes were captured by collecting logs on a virtual machine from the actual execution of the Ransomware Samples. The algorithms chosen for the original research were J48, Random Forest, Bagging and MLP algorithms.

Problem Statement:

Given a Ransomware, classify the Ransomware into it's Ransomware Family.

Method:

Feature Generation:

The dataset was read and was converted to a json database. Every element of the json Database represents a Ransomware family, every sub entrée is the name of the sample which contains the list of opcodes from the particular sample.

To generate the features, the unique number of opcodes were calculated and found to be 420. Since opcodes are sequential instructions, an n-gram model was adopted to generate the features. For this project unigrams, 2-grams and 3-grams were chosen to make the features.
Number of unique features based on 2-grams and 3-grams based on 420 opcodes

$$P(420,1) + P(420,2) + P(420,3) = 420 + 420^2 + 420^3 = 74{,}264{,}442$$

To reduce the number of features, top opcodes were selected based on their counts. The frequency of the unique opcodes was calculated for the entire dataset and stored as a key-value pair. The list was sorted in descending order of the opcode

counts and the opcodes which make 90% of the data were taken as the top opcodes. These top opcodes were found to be:
{'mov', 'push', 'call', 'add', 'sub', 'pop', 'lea', 'xor', 'jz', 'cmp', 'test', 'jmp', 'jnz', 'retn', 'and', 'movzx', 'or', 'shl'}

Based on these top 18 opcodes, 2-grams and 3-grams were generated by permuting these 18 opcodes using the itertools library[3]. Since the sequence of the opcodes matters, permutations was chosen as the parameter to generate n-grams rather than combinations. With the selected opcodes, the number of unique features generated was reduced to:

$$P(420,1) + P(18,2) + P(18,3) = 420 + 18^2 + 18^3 = 5{,}832$$

The feature labels were generated as 420 unigrams with 324 2-grams and 5832 3-grams. A total of 6576 features were generated from the dataset. The data was then vectorized based on these features using the nltk API[4]. Since the n-grams were generated from permutations rather than actual occurrences, there exists a possibility that the k-sequence of instructions are not present in the entire dataset. Hence the columns where the vector counts were 0 were dropped reducing the number of features to 4612.

The class labels were replaced by integers as:

1: 'Cerber',
2: 'CryptoWall',
3: 'CTB-Locker',
4: 'Locky',
5: 'Sage',
6: 'TeslaCrypt'

Model Training:

The data was then split into test and training sets using stratified sampling to maintain variance of training and testing sample. The data was split in a 67% - 33% train test set getting 371 samples for training and 184 samples for testing.

For the purpose of comparison, two approaches were used:

1. Data was taken as vector counts
2. Data was taken as probabilities, i.e., the vectors were normalized according to the sum of the columns

5 models were trained for comparison using the scikit-learn API[5].

1. Logistic Regression using newton-cg as the solver
2. SVM
3. Random Forest classifier with GINI as the impurity measure and 100 estimators
4. Decision Tree using an Optimized CART algorithm with GINI as the impurity measure
5. KNN classifier for k=3 neighbors with l2 norm for vector counts and Jensen-Shannon divergence[6] as distance measure for normalized vectors

Evaluation Metrics:

The models were then evaluated by generating Confusing Matrices for each model. The evaluation metrics used for the models are Precision, Recall and F-Score. The aggregate scores were calculated by taking a weighted average, where the weights were the number of instances in each class.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{P}}$$

$$FScore = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Testing Models Based on Count Vectors:

The models were trained on count vectors and the evaluation metrics were calculated.

Confusion Matrices:

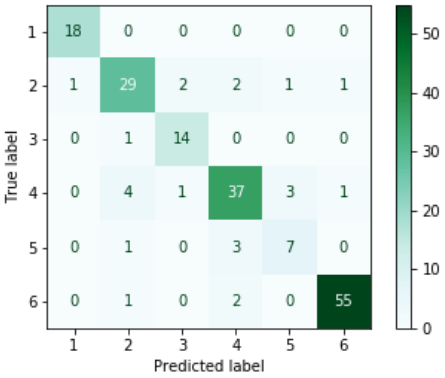

Figure 1. Logistic Regression

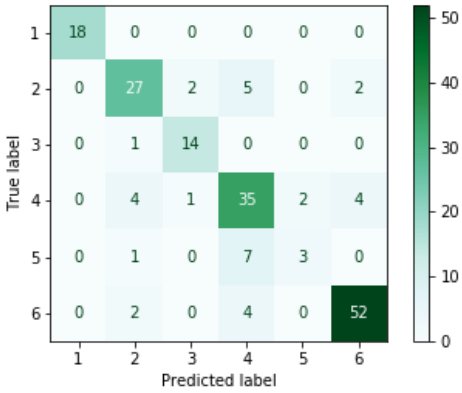

Figure 2. SVM

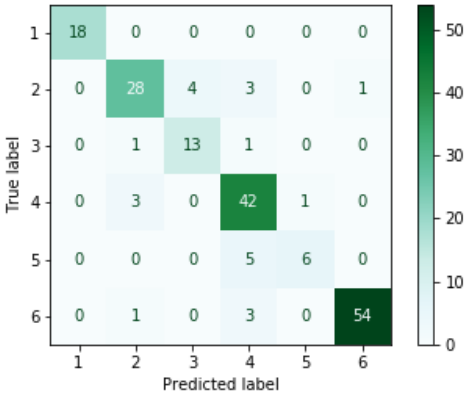

Figure 3. Random Forest Classifier

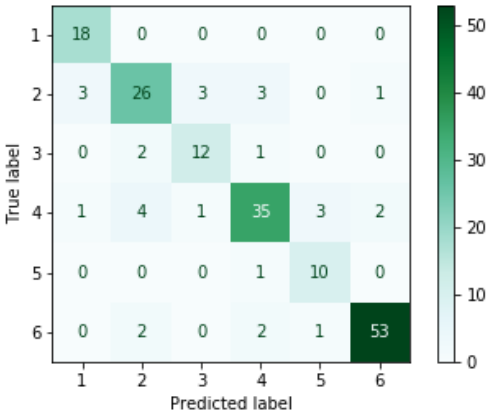

Figure 4. Decision Tree Classifier (CART)

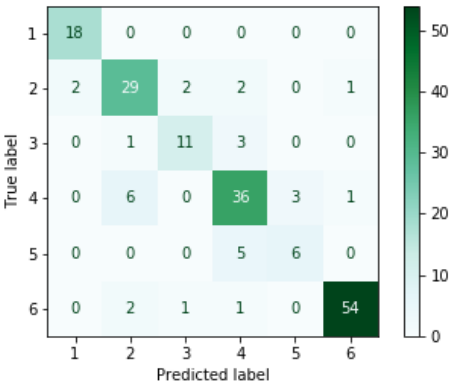

Figure 5. KNN Classifier

Evaluation Metrics:

Classifier	Precision	Recall	F-Score
Logistic Regression	0.8698494740105532	0.8695652173913043	0.8692336772027226
SVM	0.8059402021678237	0.8097826086956522	0.8034030324934818
Random Forest	0.8813469920247414	0.875	0.8739774617188327
Decision Tree	0.8698494740105532	0.8695652173913043	0.8692336772027226
Decision Tree	0.8698494740105532	0.8695652173913043	0.8692336772027226

All the classifiers have comparable metrics, SVM is the least accurate model whereas RandomForest is the most accurate.

Testing Models Based on Normalized Count Vectors (Probabilities):

The models were trained on normalized count vectors and the evaluation metrics were calculated.

Confusion Matrices:

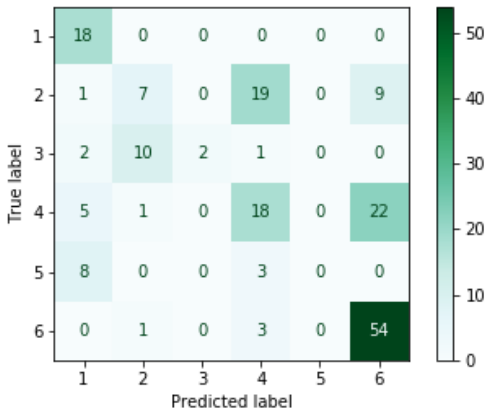

Figure 6. Logistic Regression Normalized Vectors

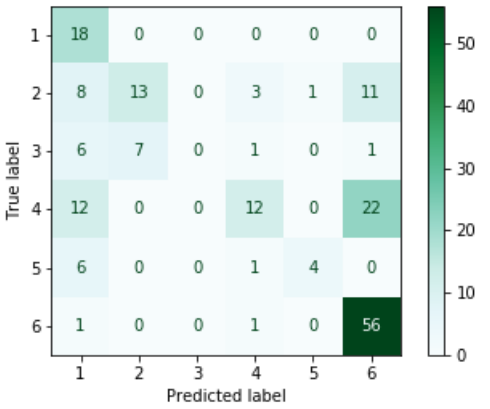

Figure 7. SVM on Normalized Vectors

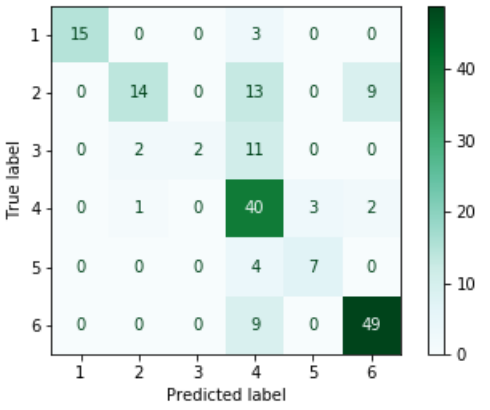

Figure 8. Random Forest on Normalized Vectors

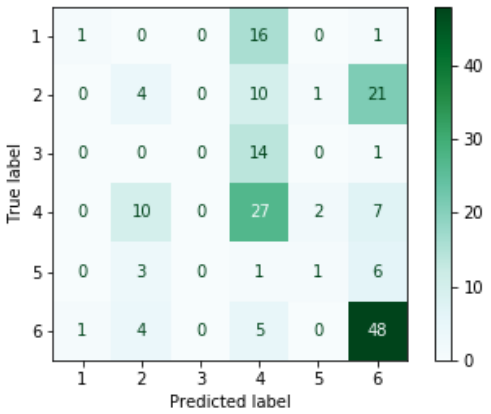

Figure 9. Decision Tree on Normalized Vectors

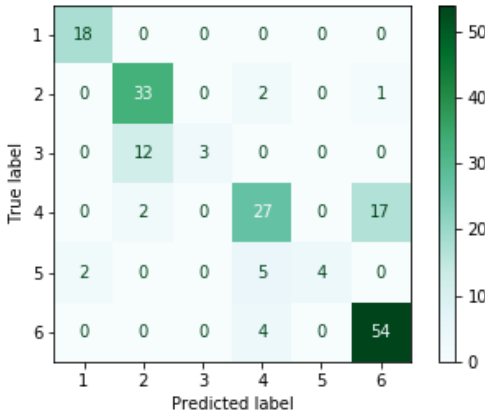

Figure 10. KNN on normalized Vectors

Evaluation Metrics:

Classifier	Precision	Recall	F-Score
Logistic Regression	0.5079228820714888	0.5380434782608695	0.47477599127215 5
SVM	0.5723287865870986	0.5597826086956522	0.5040624019239331
Random Forest	0.7647485080988917	0.6902173913043478	0.6718537506411784
Decision Tree	0.5079228820714888	0.5380434782608695	0.47477599127215 5
KNN	0.7807652514728077	0.7554347826086957	0.7299013490530 13

Here we can see that the normalization of features in fact causes the models to lose performance.

Dimensionality Reduction

After obtaining results from the previous steps, it was decided to perform feature selection based on the Lasso Regression Model[7].

Lasso regression is Linear regression along with a penalty. This penalty is applied to the feature labels as a whole. This penalty is based on the L1 Norm and hence a Zero weight for a particular feature signifies that that particular feature does not have any significance to the model. The Lasso model is obtained by minimizing the function

$$\boldsymbol{min(w)} \frac{\mathbf{1}}{\mathbf{2n_{samples}}} \boldsymbol{||Xw - y||_2^2 + \alpha||w||_1}$$

Here w is minimized by assigning weights to the feature weights. Hence for alpha = 0 for feature weights implies that those features do not contribute to the model.

The lasso model was trained and the coefficients for the features was obtained. According to these coefficients, features with non-zero weights were retained and the ones with 0 weights were discarded.

From the Lasso model, the 50 features retained were:

{'mov', 'sub', 'lea', 'xor', 'jz', 'cmp', 'test', 'jnz', 'and', 'movzx', 'shl', 'imul', 'nop', 'inc', 'fstp', 'shr', 'jb', 'xchg', 'fxch', 'cdq', 'jns', 'adc', 'bt', 'bts', "('mov', 'push')", "('mov', 'add')", "('mov', 'jmp')", "('mov', 'shl')", "('push',

'mov')","('call', 'mov')", "('call', 'push')", "('sub', 'mov')", "('sub', 'add')", "('pop', 'push')", "('lea', 'mov')", "('lea', 'push')", "('lea', 'call')", "('xor', 'cmp')", "('xor', 'test')", "('jz', 'mov')", "('jz', 'xor')", "('cmp', 'jnz')", "('or', 'mov')", "('mov', 'cmp', 'jnz')", "('push', 'call', 'mov')", "('push', 'xor', 'pop')", "('call', 'mov', 'push')", "('call', 'mov', 'lea')", "('call', 'mov', 'jmp')", "('add', 'mov', 'sub')"}

Hence using Lasso Regression, 4562 features were dropped.

Re-Training and Evaluating models based on selected Features

The classification models were trained again on the selected features and evaluated.

Confusion Matrices:

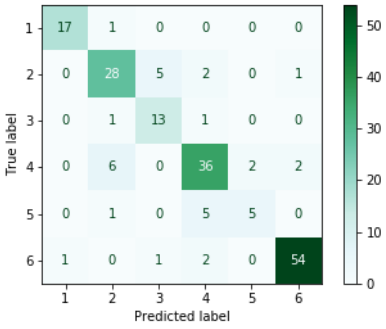

Figure 11. Logistic Regression on Selected Features

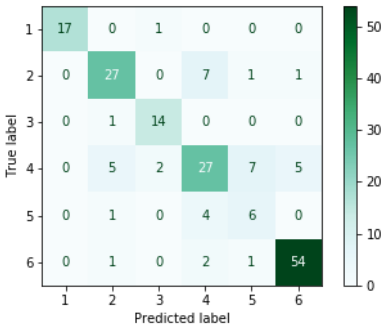

Figure 12. SVM on Selected Features

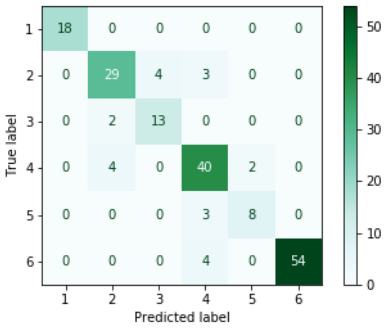

Figure 13. Random Forest on Selected Features

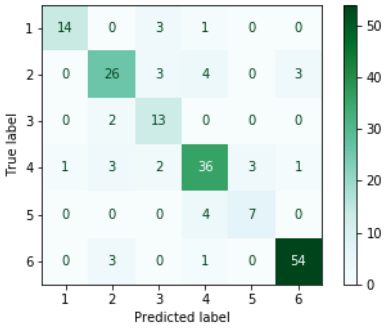

Figure 14. Decision Tree on Selected Features

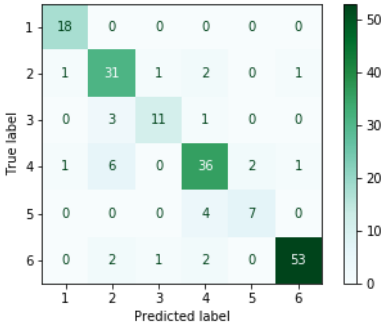

Figure 15. KNN on Selected Features

Evaluation Metrics:

Classifier	Precision	Recall	F-Score
Logistic Regression	0.8332114805226933	0.8315217391304348	0.8297157767389179
SVM	0.7922520094994521	0.7880434782608695	0.7882412268789273

Random Forest	0.8853215199123127	0.8804347826086957	0.8817326513428397
Decision Tree	0.8332114805226933	0.8315217391304348	0.8297157767389179
KNN	0.8516858745119614	0.8478260869565217	0.8475892093568933

Results and Findings:

Analysis of the Dataset reveals the opcodes that make up 90% of the data are:

{'mov', 'push', 'call', 'add', 'sub', 'pop', 'lea', 'xor', 'jz', 'cmp', 'test', 'jmp', 'jnz', 'retn', 'and', 'movzx', 'or', 'shl'}

Random forest works as the best classifier for this dataset. Since the data has to be grouped to be more similar, GINI works as a better metric for impurity for tree - based models.

For distance measure is KNN, L1 norm yields better results.The nearest neighbrs for KNN at k = 3 gives the best results.

Normalizing the vectors leads to the decrease in the model efficiency. Hence the model was retrained using Count Vectors.

Upon fitting the data into a Lasso Regression model, the features selected based on Weight coefficients alpha are:

{'mov', 'sub', 'lea', 'xor', 'jz', 'cmp', 'test', 'jnz', 'and', 'movzx', 'shl', 'imul', 'nop', 'inc', 'fstp', 'shr', 'jb', 'xchg', 'fxch', 'cdq', 'jns', 'adc', 'bt', 'bts', "('mov', 'push')", "('mov', 'add')", "('mov', 'jmp')", "('mov', 'shl')", "('push', 'mov')","('call', 'mov')", "('call', 'push')", "('sub', 'mov')", "('sub', 'add')", "('pop', 'push')", "('lea', 'mov')", "('lea', 'push')", "('lea', 'call')", "('xor', 'cmp')", "('xor', 'test')", "('jz', 'mov')", "('jz', 'xor')", "('cmp', 'jnz')", "('or', 'mov')", "('mov', 'cmp', 'jnz')", "('push', 'call', 'mov')", "('push', 'xor', 'pop')", "('call', 'mov', 'push')", "('call', 'mov', 'lea')", "('call', 'mov', 'jmp')", "('add', 'mov', 'sub')"}

Upon Feature Selection, the evaluation metrics remain similar to the models trained with entire feature set, with the exception that the KNN classifier and Random Forest classifier perform slightly better. Hence the features models work efficiently with the selected 50 features. Thus, the Lasso Regression model is extremely efficient for feature selection.

Future Work:

The data was raw and was not smoothed, the models would be retrained by smoothing the data.

Instead of using permutations as a function to generate features, the TF-IDF vectorizer would be used to generate features from the raw data. The number of grams would be increased to take in 4,5 and 6 grams.

The Lasso Model would be retained and would be used for feature selection. More concentration would be given to Random Forest Classifier and KNN classifier.

Along with this, I would also use the Adaboost classifier to check its performance.

Along with this I would also study the opcodes of windows applications to impose a knowledge based bias by adding weights to the features to give more importance to the relevant features

References:

[1] Update: Most Destructive Malware of All Time. (n.d.). Retrieved from https://www.opswat.com/blog/most-destructive-malware-all-time.

[2] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi and R. Khayami, "Know Abnormal, Find Evil: Frequent Pattern Mining for Ransomware Threat Hunting and Intelligence," in IEEE Transactions on Emerging Topics in Computing.
doi: 10.1109/TETC.2017.2756908
keywords: {Cryptography;Feature extraction;Malware;Tools;Computers;Mathematical model;Buildings;Malware;ransomware;crypto ransomware;ransomware detection;ransomware family detection},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8051108&isnumber=6558478

[3] itertools - Functions creating iterators for efficient looping¶. (n.d.). Retrieved from https://docs.python.org/3/library/itertools.html.

[4] Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. Sebastopol: Oreilly.

[5] Scikit-learn: Machine Learning in Python. (2011). *Journal of Machine Learning Research*, *22*, 2825–2830.

[6] scipy.spatial.distance.jensenshannon¶. (n.d.). Retrieved from https://scipy.github.io/devdocs/generated/scipy.spatial.distance.jensenshannon.html.

[7] 1.1. Linear Models¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/linear_model.html#lasso.