Anomaly Detection Challenge

Challenge 4: Malware Anomaly Detection w PEINFO

Ishmeet Kaur (03677735)
Mustika Rizki Fitriyanti (03667399)

January 2017

1 Introduction

The aim of this challenge is to classify the PEINFO binaries as benign or malicious using binary classification.The data consisted of a collection of PEINFO JSON dumps with the keys "sha256" and "peinfo" . The goal of the challenge was to classify the binaries as malicious or benign in the test set using the features derived from the training set.

2 Data Preprocessing and Analysis

The training dataset consisted of 9764 entries and the test set consisted of 46784 json dumps . The malign and the benign samples were almost equally distributed.

2.1 Data Analysis

The json dump consisted of sha identifying the binary and the peinfo information consisting of feature such as the rich header, exports,timestamp,versioninfo, imports,versionvar,debug information,pe_Section and version_information.

```
{   "label": "benign",
    "results":
    {
        "sha256":  "
            a5125a79ca0eb93479ac4f7f53ee45f9617c0..",
             "peinfo": {
                "imphash": "4
                    a128e034b29626b284abefbf44f9089",
                "exports": [
                {
                        "function": "
                            GetCommandManager"
                    },
                    ...],
              "pe_sections": [
                    {
                    "size": 188928,
                    "md5": "
                       b1449631d8a21ac2ce11c65af8d
                       ",
                    "virt_size": 188590,
                    "section_name": ".text\u0000\
                       u0000\u0000",
                    "entropy": 5.8956529232079475,
                    "virt_address": "0x1000"
                    },
                 ...],
              "imports": [
                {
                    "function": "tolower",
                    "dll": "msvcrt.dll"
                    },
                     ...],
              "rich_header": {
                        "Sha256": "6
                            e2606c4f6ae15540ed84710b
                            ..",
                        "values_raw": [ 13500381,1
                            ,13565917,13,....],
                        "checksum": 173844575,
                        "values_parsed": [
                            {
                                "times_used": 1,
                                "id": 205,
                                "version": 65501
                            },
                        ... ]
                },

        "thread_local_storage": [{
            "TLS Callback Function": "0x004025a0",
            "Callback Function": 0},.......}]
            "pehash": "075
               f79fb337c3d3d77d7963b2bf1f0e78",
            "version_info": [
                            {
                                "key": "LegalCopyright",
                                "value": "Copyright 2001-2012 Lexmark .."
                            },
                        ... ],
                        "debug": [ {
                            "MajorVersion": 0,
                            "subtype": "pe_debug",
                            "DebugPath": "LXAA4_iesc.pdb",
                            "PointerToRawData": "0x7df8",
                            "SizeOfData": 39,
                            "Type": 2,
                            "TimeDateString": "2013-04-25 20:23:21",
                            "DebugAge": 1,
                            "MinorVersion": 0,
                            "result": "LXAA4_iesc.pdb",
                            "DebugSig": "RSDS",
                            "DebugGUID": "b7174990aa86f598d7db3d",
                            "TimeDateStamp": 1366921401
                        }],
                    "version_var": [],
                    "timestamp": {
                        "human_timestamp": "2013-04-25T20:23:21Z",
                        "timestamp": 1366921401
            }
          }
        },

        "sha256":"5a79ca0eb93479ac4f7f53ee45f9617c0"
}
```

The json dumps were at first arranged for further analysis. The values of very key pair of every feature (like rich_header,version_var,debug, version_info,imports,pe_section) were arranged in a list for further analysis and every list characterising the dataset with a value_count greater than 1 were treated as separate features. For eg:

```
"rich_header":
{
        "Sha256": "6e2606c4f6ae15540ed84710b..",
        "values_raw": [[ 13500381,1,13565917,13,....],
                        [1,13,67,...],....],
        "checksum": 173844575,
        "values_parsed": [
            {
                        "times_used": [1,6,.....]
                        "id": [205,205   ..]
                        "version": [65501,65501,...]
                },
                ...]
},
```

2.2 Handling Categorical Data

2.2.1 One Hot Encoding:

Different sections of the json binaries were treated as categorical variables rich_header,the categorical variables were made as different features to train the dataset.

2.2.2 Limited One hot Encoding :

As the entire one hot encoding of the features took an extensive memory and runtime, the feature set containing value counts greater than 5 were encoded .Features upon which OHE was applied:

- rich_header
- Thread_local_storage
- pe_sections
- debug
- version_var

2.2.3 Tf-Idf (N grams):

N gram analysis(using one gram) increased the accuracy by 0.4 as compared to one hot encoding of the features .Tf- Idf Vectorizer was applied to the following features:

- Imports(Function and dll)
- Exports

- Error
- version -info(key,value)

3 Feature Engineering

3.1 Dimensionality Reduction by PCA :

Out of 56000 features,selecting 100 features by iterating 5 times gave an accuracy of 0.9688 using PCA

3.2 Dimensionality Reduction Univariate Selection:

Selection of 56000 features , selecting 500 features using a simple method of univariate selection (using chi square test) resulted in an accuracy of 0.967
As the feature selection methods did not increase the accuracy considerably,all features were used to train the model in a sparse matrix format.

4 Create and Evaluate The Model

After the pre - processing step, which consisted of feature engineering and data formulation , we created the model using some algorithms and evaluated the model.

4.1 Using Stratified k- Cross Validation

Stratified cross validation (10) was used for cross validation for the training set to ensure consistency between the classes.

4.2 Implementation of Classification Algorithms

During the model creation, we used the features to train the algorithms such as :

- Decision Tree Classifier
- K- nearest Neighbors
- Random Forest
- Logistic Regression
- AdaboostClassifier
- Extra Trees Classifier

In order to find the best classifier model to fit in the data set, several features in different combinations were tried (both brute force and feature selection as explained above) along with Truncated SVD(which worked with sparse matrices efficiently) ,but using all the features (with and without SVD) gave comparatively better results both in the training and the test set as seen in the Figure 1.

4.3 Enhancing Accuracy and Tuning Parameters

As the Random Forest proved to be the best classifier out of the classifiers after comparing the performance of various classification algorithms on the processed dataset,we tried to enhance the accurracy by tuning the parameters of the Random Forest Algorithm.)After detailed analysis of the random forest algorithm,we observed that higher the number of tress used for iteration ,higher is the accuracy. Using OOB(Out of Bag) Error Rate, we make a list number of maximum estimators for every replacement method, then we process in Random Forest Classifier with cross validation in order to make it comparable to other algorithms as seen in Figure 1.

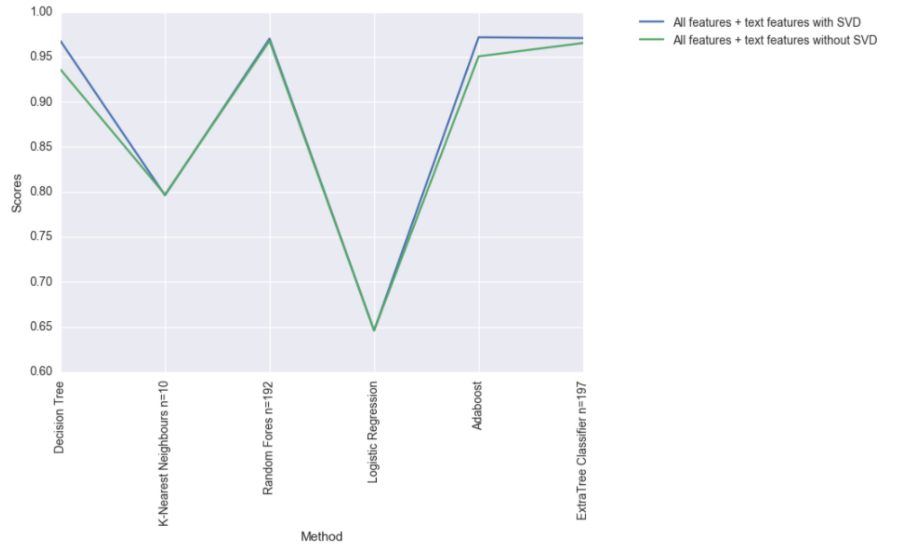

Figure 1. The result using SVD and without SVD

5 Observations and Results

- One hot encoding of all the features resulted in a large feature set resulting in Memory errors.
- Using Univariate selection and selecting only 500 features for every field of the json dump helped increasing the accuracy.

- Limited One Hot Encoding using the value counts per feature for training helped in training the model using limited features and then selecting the top useful features using univariate feature selection for initially training the dataset and to prevent the system running into memory errors.
- Using a sparse matrix representation of the features helped in consuming less memory and prevented memory errors.
- Using N-grams for the textual features increased the cross validation accuracy from 93 to 97.8
- Finding the best number of trees for random forest helped in tuning the parameters for increasing the accuracy.
- Using PCA for feature selection and dimensionality reduction and selecting only 100 features from 52000 feature set gave an accuracy of 96.2
- Selection of features by the Univariate Selection method(500 out of 52000) gave an accuracy of 96.72.
- Increasing the number of trees reduced the system performance and lead to memory management problems but increased accuracy.
- Adding features such as debug info, and behavioural features like number of sections and number of imports and exports decreased the accuracy.
- The highest score resulted using all the features. As seen in Figure1, SVD helped increasing the accuracy.

6 Summary

After evaluating the performance of the Random Forest Algorithm with the number of estimators as 197, we concluded that it worked comparatively better as compared to other algorithms as seen in Figure 1 .The best algorithm found was Random Forest classifier with 197 number of estimators with a public score of 0.97884.

References

[1] $https : //www.sec.in.tum.de/assets/Uploads/deeplearning.pdf$

[2] $http : //scikit - learn.org/stable/modules/feature_selection.htm$