REPORT MACHINE LEARNING HOMEWORK 1

MALWARE CLASSIFICATION

“Classification of Malware through Machine Learning Models”

George Ciprian Telinoiu - 1796513

29/11/2020

ABSTRACT

The task that we have been assigned was to analyze a JSON file populated with a dictionary containing several Assembly Instructions related to different kinds of functions that usually are exploited to create malwares.

Our job was to define and implement several Classification Models in order to successfully classify the various functions based on the analysis of the given dataset.

Moreover we had 4 possible functions:

- *Encryption*
- *Math*
- *Sort*
- *String*

Finally, after evaluating our models we were asked to perform a blind test prediction on another JSON file that had only the set of instructions, leaving blank the ground truth.

0. PRELIMINARIES

The following report will be divided into several subsections, following the order in which I implemented and reasoned around the models presented for the given dataset, specifically:

1. Pre-Processing & Dataset Analysis
2. Model Selection & Evaluation
3. Results
4. Blind Test
5. Conclusion

All the implementation was developed within the *Google Colaboratory* environment, thoroughly making use of several python libraries such as *Pandas*, *Numpy* and *Sklearn*.

1. Pre-Processing & Dataset Analysis

Before reasoning on the possible implementations for the given dataset, it was necessary to transform the Assembly instructions into a representation that could be put to use with a Classification Model. Therefore it meant that I had to perform a Vectorization. Vectorizing text is what allows us to have numerical representation of text that we can work with. Simply put, to vectorize text is to represent text as numerical vectors. Before doing such a thing though, I had to decide what kind of words I wanted to include in my analysis given the fact that not every word within the ones provided in the dataset were of real interest for our task.
I therefore decided to work exclusively with the x86 instructions set and ignore all variables for this analysis.

I loaded the dataset using the *"read.json()"* Pandas function, that returned a Pandas DataFrame object that was used afterwards to work on the Pre-Processing.

At this point I decided to choose the features that I would have taken into consideration for the following Models that I would have used and settled on the analysis of the x86 Instructions, ignoring the cfg graph feature.
Moreover, after several tests I also came to the conclusion that basing my analysis on the frequency of the set of instructions, even if basic, was definitely enough for the given task.

In order to optimize the Vectorizations of the instructions I performed some cleaning of the data that came with the dataset, explicitly getting rid of any symbols and variables that could only increase the size of the Vector without bringing any kind of benefit to this process.
Therefore I implemented a function called *"extract_instr"* that utilised a Regular Expression, found all and only the instructions from the "lista_asm" label for each function of the dataset, saved them in a local variable and returned a string containing all the instructions in order separated by a white space.
This process was then repeated for all the entries of the Dataset, substituting the value of the DataFrame object for the label *"Lista_Asm"*.

What remained to be done now was the selection of the Vectorization method to be applied to what we had built previously. Before settling for the *"tfid"* method, I tried playing with the *"hashing"* and *"count"* methods; but being able to provide the *"ngram_range"* in the *"tfid"* turned out to be crucial.
In fact, the first set of tests has been done with ngram set to 1, therefore considering each word individually. I wanted to set a base-ground for the following modification, that being to increase its value up to 5, showing tangible results that we will comment later. Also, it's reasonable to have more interest in a set of instructions compared to the single one, considering that certain patterns, especially for encryption for example, are noticeable only if we consider the dependency between adjacent instructions.

I saved the result of the Vectorization in the *"X_all"* variable that will be used for the training and testing procedure. In order to prepare for the split, I have also saved the labels of my dataset into the *"y_all"* variable.

Finally I proceeded to perform my *train_test* split with a *test_size* set to 0.33, and a constant *random_state* in order to keep the various tests. Also no shuffle was performed since it would have brought no benefit for the given task.

2. Model Selection & Evaluation

I decided to utilize several Classification Models, hypothesizing SVM and the SGDClassifier to perform optimally for the given task. I have also decided to perform a comparison between the *std* Decision Tree Classifier (*no max_depth)* and a *limited* version of it. Finally the last Model analysed has been the one of Logistic Regression following the *One vs Rest* approach. The Train & Test Set was common among all models. As mentioned in the previous chapter, the Vector has been normalized in order to help the classification process.

Before proceeding to the Model Evaluation, a quick summary of the names used:

● SVM	*(model 1)*
● Decision Trees (std)	*(model 2)*
● Decision Trees (limited)	*(model 21)*
● Logistic Regression	*(model 3)*
● SGD Classifier	*(model 31)*

After training all the mentioned models, it was time to compare the results.
First of all I proceeded to analyze the accuracy of each model, which turned out to be very high, especially after increasing the ngram value to 5. This sort of improvement will be also discussed when comparing the prediction of the Blind Test Set.
Overall the very high percentage was given by the optimal separation of the data, given that each function has distinct features, both in length and sequence of instructions.
Moving onto Confusion Matrix & F1-Score analysis, it's interesting to look at the values returned by the Decision Tree with limited depth, which is the only model that had several cases of FP, specifically with the "*Sort*" function.
Another interesting result is the comparison of Logistic Regression and SGDClassifier that present identical results when looking at their Confusion Matrix (*more on this in a bit)*.
Besides those irregularities that were purposely inserted in order to analyze them , we find no remarkable outliers.
Finally we have introduced a Cross Validation analysis to better understand the value of the accuracy that we found, and indeed all models perform similarly to what we have observed; with only one exception, that being if we compare Logistic Regression and SGDClassifier. Even though their CMs were similar, we can see clearly that the SGD Model is more robust and returns a better result when compared to the Logistic Model.

3. Results

Here we’ll find a summary of the Numerical Results of Training & Prediction that have been commented in Section 2.

3.1 SVM

CLASSIFICATION REPORT

	precision	recall	f1-score	accuracy
encryption	0.989	0.992	0.990	
math	0.996	0.991	0.994	
sort	0.949	0.971	0.960	
string	0.996	0.994	0.995	
				0.991
macro avg	0.982	0.987	0.985	
weighted avg	0.991	0.991	0.991	

CONFUSION MATRIX

```
[[359   2   1   0]
 [  0 792   5   2]
 [  3   1 167   1]
 [  1   0   3 686]]
```

CROSS-VALIDATION SCORE

```
[0.99011369 0.99209095 0.98863075 0.99307958 0.99060801]
Accuracy: 0.991 (+/- 0.00)
```

3.2 DECISION TREE (STD)

CLASSIFICATION REPORT

```
              precision      recall        f1-score      accuracy
encryption    0.980          0.959         0.969
math          0.984          0.992         0.988
sort          0.897          0.860         0.878
string        0.976          0.987         0.981
                                                         0.973
macro avg     0.959          0.950         0.954
weighted avg  0.973          0.973         0.973
```

CONFUSION MATRIX

```
[[347   4   7   4]
 [  0 793   5   1]
 [  6   6 148  12]
 [  1   3   5 681]]
```

CROSS-VALIDATION SCORE

```
[0.97330697 0.96984676 0.96885813 0.97528423 0.9742956]
Accuracy: 0.991 (+/- 0.00)
```

3.21 DECISION TREE (LIMITED)

CLASSIFICATION REPORT

```
              precision      recall        f1-score      accuracy
encryption    0.929          0.906         0.917
math          1.000          0.954         0.976
sort          0.611          0.866         0.716
string        0.970          0.933         0.951
                                                         0.931
macro avg     0.877          0.915         0.890
weighted avg  0.944          0.931         0.935
```

CONFUSION MATRIX

```
[[328   0  26   8]
 [  1 762  34   2]
 [ 13   0 149  10]
 [ 11   0  35 644]]
```

CROSS-VALIDATION SCORE

```
[0.92436975 0.92881859 0.92684132 0.93030153 0.92634701]
Accuracy: 0.927 (+/- 0.00)
```

3.3 LOGISTIC REGRESSION

CLASSIFICATION REPORT

	precision	recall	f1-score	accuracy
encryption	0.992	0.983	0.988	
math	0.992	0.991	0.992	
sort	0.959	0.942	0.950	
string	0.986	0.996	0.991	
				0.987
macro avg	0.982	0.978	0.980	
weighted avg	0.987	0.987	0.987	

CONFUSION MATRIX

```
[[356   3   2   1]
 [  0 792   4   3]
 [  3   1 162   6]
 [  0   2   1 687]]
```

CROSS-VALIDATION SCORE

```
[0.98220465 0.9871478  0.98368759 0.98961938 0.98319328]
Accuracy: 0.985 (+/- 0.01)
```

3.31 SGD CLASSIFIER

CLASSIFICATION REPORT

	precision	recall	f1-score	accuracy
encryption	0.992	0.983	0.988	
math	0.992	0.991	0.992	
sort	0.959	0.942	0.950	
string	0.986	0.996	0.991	
				0.987
macro avg	0.982	0.978	0.980	
weighted avg	0.987	0.987	0.987	

CONFUSION MATRIX

```
[[356   3   2   1]
 [  0 792   4   3]
 [  3   1 162   6]
 [  0   2   1 687]]
```

CROSS-VALIDATION SCORE

```
[0.99406822 0.99505685 0.98912506 0.99456253 0.99209095]
Accuracy: 0.993 (+/- 0.00)
```

4. Blind Test

Now that we have evaluated the various models, we are ready to truly test them on a new set of instances, and try to understand what kind of results we get.
Considering the results of Section 3, the Model that has been chosen to populate the txt file required was the SGDClassifier, with a very close contender of the SVM Model *(as hypothesized at the beginning of the Report).*

In order to make the predictions, we had to Vectorize again the instructions as described in Section 2. After that using the *"open()"* and *"write()"* functions we have populated our document.

Given the fact that I trained several models, I decided to use them to make predictions also on this Blind Dataset and compare the returned results in order to better understand how each Model worked.

In order to do this systematically, I implemented a functions that given 2 Models, compared the results they gave and populated an array composed of 2 parameters:

- C = Number of collisions *(so every time model1[i] != model2[i]);*
- cont = A python Dictionary containing the number of conflicts based on the predictions of the 2 models (*the decision was valid either if one or the model predicted a certain class compared to the other).*

Here's a recap of the values returned by the comparison given 2 values on ngram (1-5)

NGRAM == 1

SVM vs Decision Tree(std)
```
{
'encryption vs math':      5,
'encryption vs sort':      16,
'math vs sort':            21,
'encryption vs string':    0,
'math vs string':          1,
'sort vs string':          29
}
```

SVM vs Logistic Regression
```
{
'encryption vs math':      4,
'encryption vs sort':      9,
'math vs sort':            3,
'encryption vs string':    1,
'math vs string':          0,
'sort vs string':          61
}
```

Decision Tree (std) vs Logistic Regression
```
{
'encryption vs math':      2,
'encryption vs sort':      16,
'math vs sort':            13,
'encryption vs string':    1,
'math vs string':          11,
'sort vs string':          56
}
```

Decision Tree (std) vs Decision Tree (limited)
```
{
'encryption vs math':      10,
'encryption vs sort':      21,
'math vs sort':            20,
'encryption vs string':    16,
'math vs string':          4,
'sort vs string':          56
}
```

Logistic Regression vs SGD Classifier

```
{
'encryption vs math':      3,
'encryption vs sort':      5,
'math vs sort':            4,
'encryption vs string':    2,
'math vs string':          0,
'sort vs string':          31
}
```

SVM vs SGD Classifier

```
{
'encryption vs math':      1,
'encryption vs sort':      9,
'math vs sort':            3,
'encryption vs string':    2,
'math vs string':          0,
'sort vs string':          45
}
```

NGRAM (1-5)

SVM vs Decision Tree(std)

```
{
'encryption vs math':      5,
'encryption vs sort':      6,
'math vs sort':            16,
'encryption vs string':    1,
'math vs string':          1,
'sort vs string':          13
}
```

SVM vs Logistic Regression

```
{
'encryption vs math':      3,
'encryption vs sort':      6,
'math vs sort':            12,
'encryption vs string':    2,
'math vs string':          2,
'sort vs string':          7
}
```

Decision Tree (std) vs Logistic Regression

```
{
'encryption vs math':      4,
'encryption vs sort':      12,
'math vs sort':            21,
'encryption vs string':    3,
'math vs string':          6,
'sort vs string':          11
}
```

Decision Tree (std) vs Decision Tree (limited)

```
{
'encryption vs math':      6,
'encryption vs sort':      23,
'math vs sort':            22,
'encryption vs string':    16,
'math vs string':          5,
'sort vs string':          32
}
```

Logistic Regression vs SGD Classifier

```
{
'encryption vs math':      2,
'encryption vs sort':      6,
'math vs sort':            20,
'encryption vs string':    2,
'math vs string':          3,
'sort vs string':          4
}
```

SVM vs SGD Classifier

```
{
'encryption vs math':      3,
'encryption vs sort':      0,
'math vs sort':            13,
'encryption vs string':    0,
'math vs string':          0,
'sort vs string':          2
}
```

I think this kind of comparison gives us a deeper insight on the Blind Predictions, especially on the effectiveness of the features that we selected.
First of all, with NGRAM set to 1 we have way more collisions between our models, especially when trying to classify *"sort"* and *"string"* functions. This shows us that analysing only 1 instruction at the time brings into play a lot of false positives, since the model can't truly decide between the two. Another quick observation that validates what we have seen in Section 3 is the difference between the 2 Decision Tree Models, where due to the fact that the Limited Model gives birth to lots of miss classifications, the predictions collide a lot, even for easier classification such as *"encryption vs math"*.
Moving to NGRAM(1,5) we can see that we alleviated the issue that we had with the *"sort"* function (even if it's still there) and the same goes for the differences between the Decision Tree Models. Finally it's interesting to see how the similar accuracy presented in Section 3 between SVMs and SGDClassifier gets highlighted here with very similar predictions, with the only difference being on *"math vs sort"*. We can also then understand that the predictions that they agreed on with N GRAM equal to 1, were kind of misleading since now they diverge way more.

5. Conclusion

Based on the analysis that we have developed we found that the 2 best Models that classified the dataset based on the features provided were SVM and SGD Classifier. The models don't seem to suffer from overfitting considering the Cross Validation results and the considerations conducted on the Blind Set.

Finally I add that several other feature extractions could have been conducted, for example defining a more refined bag of instructions, maybe classifying them based on the repercussions they had in each function; or considering the nodes of the CFG so uniquely identify each function.

Rome, 29.11.2020
George Ciprian Telinoiu