Report Homework n°1

Fabrizio Casadei

Matriculation number: 1952529

SAPIENZA
UNIVERSITÀ DI ROMA

November 28, 2020

Chapter 1

Overview

The aim of this project is to solve a classification problem, concerning code written in Assembly language, hence binary functions.
To understand the nature of a certain program, it's important to know what kind of operations it wants to perform, many malware use kinds of well-known pattern useful for the prevention.
For example, if we are looking for ransomware in a bunch of assembly files, very probably these files will have some type of encryption function, utilized for blocking user's files, and so on for other categories of malware.
This operation of classification can automate a lot of work that otherwise would be done in a manual matter, eventually even with less correctness than the one which can guarantee a machine learning approach.
The dataset that we are going to manage it's composed of 4 binary functions kinds :

- **Encryption**, which encrypts some files
- **String manipulation**, doing some kind of operation on strings, as: comparison, cutting, substitution, so on and so forth.
- **Math**, large set of math operations on numerical value and matrices
- **Sorting**, any possible type of sort algorithm used on arrays like: Insertion, Selection, Merge, etc.

1.1 The dataset

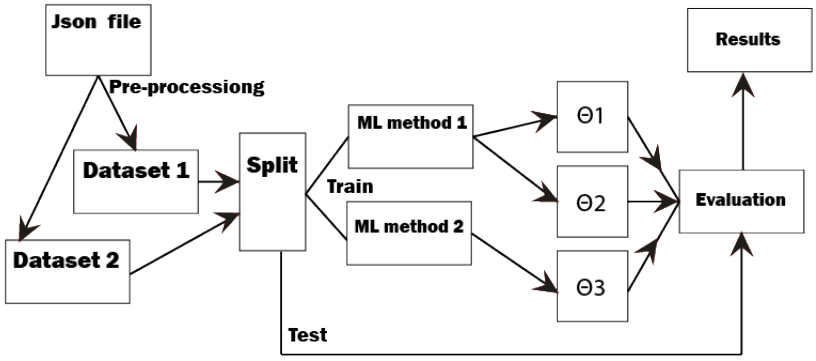
The choice performed for the use of datasets available is the following:
for the learning and testing process has been used a dataset without duplicates, leading to better performance. Then the learned model has been used for the classification of a blind dataset that includes duplicates. All the dataset are in JSON line format, so in each line, there is a JSON object, which is a dictionary with pairs "key-value", the keys are:

- **ID**, the identifier
- **semantic**, the label of each binary function (obviously the blind dataset doesn't have this key)

- **lista_asm**, lines of the assembly code
- **cfg**, the control flow graph, optional data usable for the pre-processing

1.2 Strucuture of the project

In this project has been performed three different types of model, which differs in two main aspects: the vectorization and the ML model used.
In the specific, as we can see in the figure below, a certain ML method will use two different types of pre-processed dataset (in the figure Dataset1 & Dataset2), and this is very interesting, to understand how much the elaboration of data con variate the final result.
The last result is made using the Dataset1, as named in the schema, and another ML model.



Schema of the project

In the next chapters will be discussed the characterization of these components, and then examined the given evaluations in terms of:

- **Confusion Matrix**
- **Precision**
- **Recall**
- **f1-score**
- **support**

All these metrics give to us, an very clear idea of how the model's behaviour in the test phase.
Finally, we will try to give an explanation of the different results obtained and will compare these values to expectations.

Chapter 2

The pre-processing

In this chapter we are going to analyze how has been handled the problem at the root, or rather, how has been managed the dataset, which represents the input for our system. Before starting the explanation, it's important to highlight an aspect, the pre-processing step can be divided in two parts: the first one, where we manage the samples, following certain rules, and the second part, where we use some kinds of vectorization technique. Just the latter could be different in the results we will see, instead, the first is a fixed common step in the models, hence, all the result are based also on this.

The binary function that we want to classify, have several features (which are based on the knowledge of the domain expert), which are present with an high level of probability, almost certainties. Let's see these more specifically, for all the labels, what features have been taken into account.
Encryption features:

- Use of xor, shifts and bitwise operations
- long number of instructions

Sorting features:

- One or two for
- Operations for comparing and moving

Math features:

- Math operations, both on vectors and matrices
- Frequent use floating point instructions

String manipulation features:

- Operations for comparing and swapping of memory

General useful features:

- Number of external calls
- Number of instructions
- Number of accesses to registers

With this assumption, has been created a list of assembly key words which describes all these features, and each sample has been replaced by a string which characterize the occurrences of all the aspects, accordingly with the features.
For an example, in a random sample, we will have the number of instruction, then we could have the times that has been used the command "mov", or could be the number of occurrences for the register "ebx", so on and so forth for all the categories listed above.
All these information are then used by the vectorizer to generate the tokens.
With this process we have extract only the information that is really concerned with the classification problem, thus to reduce the probability of having noisy data, consequently, the performances are increased.

After the vectorization, the dataset will be split into two sub-dataset, and for doing this, an important decision must be taken. How do we divide the dataset?
In this case we have used the 80% of the dataset for learning operation, and the rest 20% for testing.
Evaluating a model using the same dataset for learning and testing is an error because in this case, you aren't evaluating how the system performs in realistic conditions, hence blind data.

Chapter 3

First solution

This is the first solution for the project, here we define a vectorization method and the ML model, which will be alternatively used also in the second and third configuration.

3.1 Vectorization

Let's start the analysis of first solution discussing the vectorization method that has been used.
The vectorization is a fundamental step, to use machine learning method on the dataset, we need to transform data into a vector representation, through a vectorizer. In this case, we have used a CountVectorizer, which are going briefly to discuss.
This vectorizer perform both tokenization and build a vocabulary of the input words.
This vocabulary map the words and membership documents, in this matter the handling of data is more favoured. Using this vocabulary is also possible to add new documents at the collection.

Using this vectorizer we could have the problem of repetitions, that reduce the importance of the words, but this is partially resolved by the previous handling of the data which has been made. So we aspect good performance from this point of view.

This vectorizer is also used in the third solution.

3.2 Classification model

The second aspect which characterizes this solution is the ML model used, which is the **SVM**
This ML model is a very powerful and complete resolutive one, which combines the experiences of other models like **Fisher's linear discriminant**.
In particular the parameters utilized are the kernel and gamma. Gamma defines how far the influences of a single training, hence, for regularizing the sensitivity of values far from the decision boundary. This has been set to Scala, which is equal to $\frac{1}{n_{features} * Variance_x}$
While the kernel used is linear. This 2 parameters after different tries, seem good in terms of performances.

3.3 Results

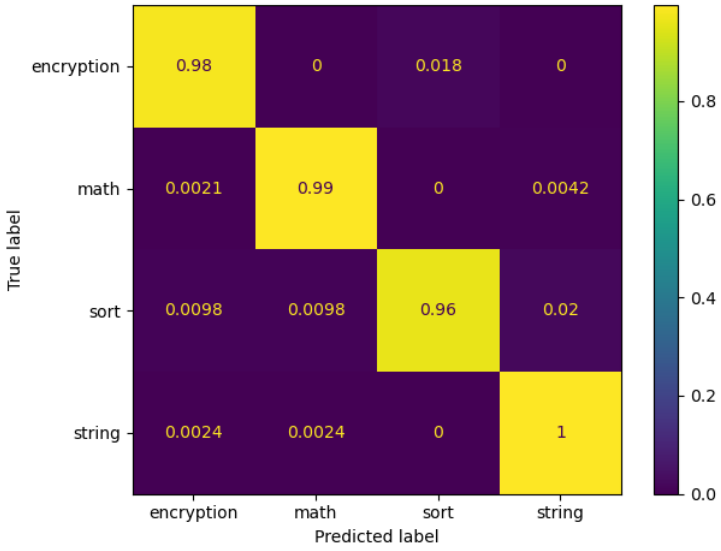
Now it's the moment to see how good has been this model, in terms of performance. We start talking about the time for performing of these operations, and that's what I have recorded:

- *Time for vectorization* $\simeq$ 0.6 seconds
- *Time for learning* $\simeq$ 0.02 seconds
- *Time for predicting* $\simeq$ 0.8 seconds

Then we pass to look at the quoted metrics of the first chapter, and how their values are characterized in this model through the classification report:

	precision	*recall*	*f1-score*	*support*
encryption	0.99	0.98	0.98	219
math	1.00	0.99	0.99	474
sort	0.96	0.96	0.96	102
string	0.99	1.00	0.99	420
accuracy			0.99	1215
macro avg	0.98	0.98	0.98	1215
weighted avg	0.99	0.99	0.99	1215

Finally we have generated the **confusion matrix**, which gives a clear idea of how the model predicts each class.



Confusion matrix first solution

Chapter 4

Second solution

This solution differs from the first one, by a new vectorization algorithm.The model is still a SVM.
Let's how much this new version will change the performance, and how they variate.

4.1 Vectorization

In this case has been chosen the **TfidfVectorization**, which is enough similar to the countVectorization, but it doesn't calculate the number of repetitions of a word, instead, it extrapolates how much is frequent. **TF-IDF** are metrics which try to give more importance to certain words, for example, based on the distribution of the frequency, it measures the "originality". Has been used the parameter n-gram, which goes to consider the lower and upper limit of the range of values for different n-grams to be extracted. All values on n will be used such that $min_n \leq n \leq max_n$.
Hence, this vectorizer performs the usual operation like tokenization and learning of the vocabulary, using an inverse document frequency weightings.

Considering the pre-process data before the vectorization, we can imagine, not a real convenience of using this rather than the countVectorization, based in a major way for the nature of the input. The use of the countVectorization with the previous handling of data will in certain sense, account in a better way the concept of frequency. Obviously, this isn't always true, indeed quite the opposite.

4.2 Results

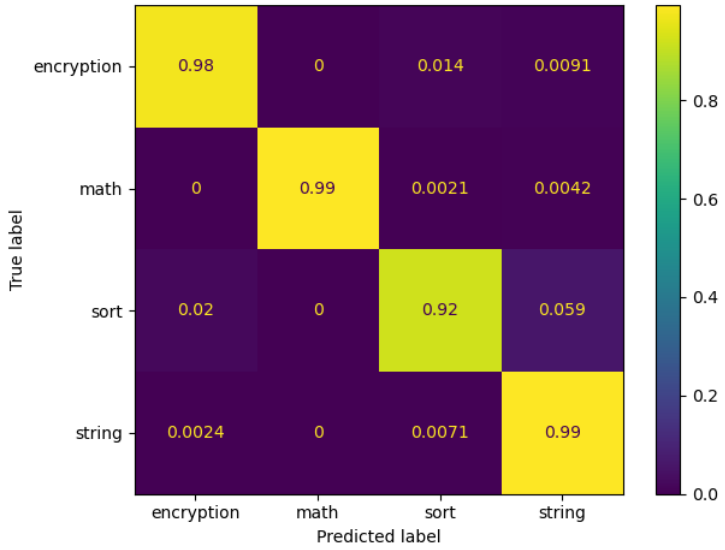
With the given information about the difference between this and the first solution, we can in the end highlight how the pre-processing influences results, in the specific how much will change it another kind of vectorization.
How we did for the first solution, we first give a look at the timing of each of the operations.

- *Time for vectorization* $\simeq 0.07$ seconds
- *Time for learning* $\simeq 0.4$ seconds
- *Time for predicting* $\simeq 0.5$ seconds

The classification report that we have registered is the following:

	precision	*recall*	*f1-score*	*support*
encryption	0.99	0.98	0.98	219
math	1.00	0.99	1.00	474
sort	0.93	0.92	0.93	102
string	0.98	0.99	0.98	420
accuracy			0.98	1215
macro avg	0.97	0.97	0.97	1215
weighted avg	0.98	0.98	0.98	1215

In conclusion for this chapter we give information on the Confusion Matrix.



Confusion matrix second solution

Chapter 5

Third solution

The last solution which we are going to see, present a new model for the classification problem, with the same vectorizer of the first solution (CountVectorizer).

5.1 Classification model

In this version of the implementation, we will see how good will perform a **Multinomial Naive bayes**, which exploits the probability distribution of the 4 classes for new samples.

Using this model we expect not really great performance, it's simpler classifier if compared with the SVM, and is usually used on smaller problems.

5.2 Results

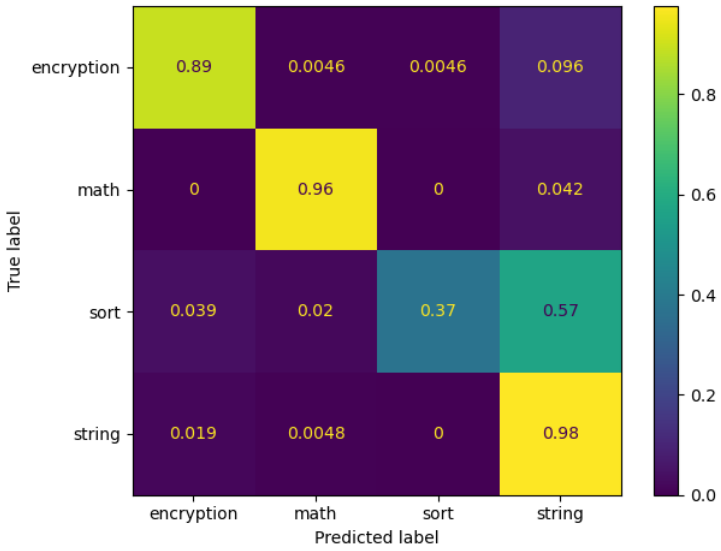
Before passing at the conclusion, and give comparisons through solutions, we show the information retrieved on the third.
The timings:

- *Time for vectorization* $\simeq$ 0.6 seconds
- *Time for learning* $\simeq$ 0.01 seconds
- *Time for predicting* $\simeq$ 0.003 seconds

The classification report:

	precision	*recall*	*f1-score*	*support*
encryption	0.94	0.89	0.92	219
math	0.99	0.96	0.97	474
sort	0.97	0.37	0.54	102
string	0.81	0.98	0.88	420
accuracy			0.90	1215
macro avg	0.93	0.80	0.83	1215
weighted avg	0.92	0.90	0.90	1215

The confusion matrix:



Confusion matrix third solution

Chapter 6

Conclusion

6.1 The comparison

Now it's time to take stock of all the evaluations that we have seen. Firstly we can spend two words on the general result which are seemed overall good, we have noticed a wide descent of performance only in the third solution, while the other two are characterized from very good values, just thinking about the True positive and True negative of confusion matrices, all values are greater than 0.90. But how we know, this doesn't give to us the certainty of good prediction for new samples, cause we could have *overfitting*.
let's analyze in order.
The metrics tell to us, that the first solution is the better, simply using the *Count vectorization* and the *support vector machine* model, the results are great! thus the decision boundary which has been created is very precise for the dataset test which we have used, it's not perfect and this is not a bad thing a priori, because a more general solution can model in a better way new samples.
Then in the second solution, we have still got very good results, but we can see, how much the vectorization method can impact on the correctness of predictions. In fact, we are passed from *Count vectorization* to *TfidfVectorization*, and as expected, in this case given the previous managing of data, *Count vectorization* has better behavior, even if is more simple. In any case, this is an acceptable solution for the problem.
Finally, we compare the result for the last solution, which looks the worse one, as expected.
A simple ML model like the *Multinomial Naive bayes* can classify in a satisfactory way the problem which are modelling. Even because the hypothesis space is too much large for having predictions that are comparable with the others. In the specific as we can see there's a lot of uncertainty in the prediction of Sort sample, that are confused with String manipulation functions.
This is caused by the quite similar nature of some operations in the two categories, in the sort algorithms, we have operations like comparisons, substitutions, so on and so forth, just similar to the String operations. Other models reach a good value even with this difficulty, but the *Multinomial Naive bayes* fails.
The last comment that we want to make is about the learning time, in both the three cases the timing is excellent, with very slight differences, but the *Multinomial Naive bayes* has shown the best performance from this point of view, which is the half of the timing needed for the *SVM* that use the same vectorizer.

6.2 Blind test

Given the previous comparison, the first model has been chosen to predict each class of samples into the *blindset*.
The data will be handled and vectorized, updating the document-term matrix (through the transform operation), then the SVM model, appropriately learned, tries to predict correct labels.
In the end has been created a text file that is made up by each predicted label, one for each line.