Homework 1 Machine Learning
Function classification

Santilli Sofia
matricola 1813509

November 2020

1 Description of the problem

The goal of this work is to classify binary functions which belong to four different classes:

- *Encryption*, functions with the aim of encrypting some code;
- *Math*, which do mathematical operations;
- *Sorting*. They sort arrays;
- *String manipulation*, in order to modify, compare strings or other.

A dataset of 6037 samples is given. Each sample is a function which belongs to one of the classes. We have:

- 1110 samples are Encryption functions;
- 2412 samples are Math functions;
- 2014 are String Manipulation functions;
- 537 are Sort functions;

In Figure 1 is plotted this distribution of samples between the four classes.

Each function is composed of a set of assembly instructions, set that has no predefined lenght. Each item in the dataset is a dictionary; so it has the form {key1: value1, key2: value2, ...}. There are four keys:

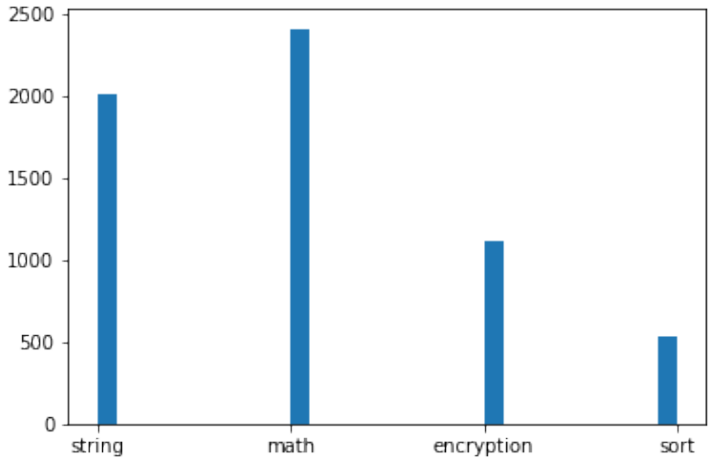

Figure 1 Distribution of the samples between the four classes.

- *ID*. It's unique for each function;
- *semantic*, it is the label which denotes the belonging class of that function;
- *lista_asm*, a linear list of assembly instructions
- *cfg*, the control flow graph, encoded as a network graph.

2 Preprocessing

The first thing to do is preprocessing the dataset, prepare the row data we have in order to make them suitable for a building and training Machine Learning model.
So before doing this, it is necessary to know well what are the features that distinguish one class from another. In general we know that:

- encryption are complex functions, which has a lot of *for*, *if*, *xor*, *shifts* and *bitwise operations*;
- math instructions are obviously characterised by *arithmetic operations*. They also use a special register, *xmm**;

- string manipulation, uses a lot of *comparisons* and *swap of memory locations*;
- sorting uses *compare* and *moves* operations and only one/two *for*.

With such information above, the type of class can be recognized by analyzing the control flow graph (analyzing its complexity, the number of loops) or the lista_asm.
In this work the graph has been left out; only the list of assembly instructions has been considered.
All the operations found in these instructions are expressed in the dataset through assembly commands such as: jmp, cmp, xor, shl, shr, inc, dec, mov, mul,... . We note that these 'words' are always at the first place in the assembly instruction. So the main features that allow us to distinguish the belonging class of a function consists of the first words of the instructions in the samples of the dataset. Therefore, after having converted the dataset's json file into a panda object (through the panda.read_json), it's considered a sample at the time. Each sample is splitted (using the comma as separator) in a list of instructions. Then, each instruction is divided in turn into the words that form it (using again the split, this time with the default separator, the blank space). So now, considering only the first word of the instruction we've just splitten, we obtain the feature we wanted.
The split of a string results in a new list of strings. So after the previous operations, for each sample in the dataset we have a list of lists. We rearrange these aftermath, in order to have a simple list for all the samples. This final list will contain N strings, where N in this case is 6073, the number of samples. Every string is formed by all the first words (assembly commands), divided by a blank space, in the instructions of the respective sample. This list is subjected to a vectorization, as explained in the next paragraph.

In the preprocessing done, xmm* registers have not been considered; they are never the first word in and assembly instruction. So the preprocessing chosen doesn't consider this feature useful in order to distinguish instructions of the math class. Nevertheless, the results we'll see below have a good accuracy; therefore the consideration of xmm* registers doesn't seem to be fundamental for the classification./it doesn't seem to be fundamental taking into account the xmm* registers for the classification.

3 Vectorization

Now we have a list of strings, with one string for each sample in the dataset. This list can be passed as argument to a function, in order to do the vectorization. The chosen type of vectorization is the CountVectorizer, which converts a collection of text documents to a matrix of token counts. In this way it's possible counting the frequency of the commands (jmp, mov, add,...) in a certain type of function.
A choice was not to enforce a minimum word count; so all the commands which appear in the asm instructions are taken into account in this work. We already know the number of rows of this matrix, as it corresponds to the number of samples. So we have 6073 row. The number of columns is found printing the shape of the matrix in our code: 177. Therefore in our dataset there are 177 different commands at the beginning of instructions.

4 Split in Training set and Test set

The next step is a procedure that involves taking the dataset and dividing it into two subsets. The first subset is the Training set, used to train the model. The second is the Test set: an input element in this set is provided to the model, then predictions are made and compared to the expected values. There are different ways in order to split the dataset. Here we consider the 20% of the samples for the test set and the 80% for the training. So we obtain the distribution of samples showed in Figure 2. It's easy to see that, adding samples of the two sets which belong to the same class, we obtain the number of samples seen in Figure 1[1].

5 Models

5.1 SVM: support vector machines

Support vector machines are a set of supervised learning methods, effective in high dimensional spaces. It's also a versatile method: it's possible to spec-

[1]The sum can't be done exactly, because the labels on the chart axes defines only some ranges

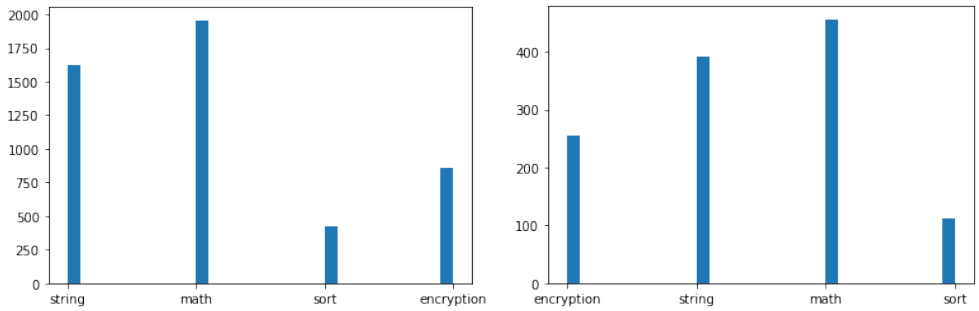

Figure 2 Distribution of the samples between the four classes. On the left the Training set, on the right the test set.

ify different Kernel functions for the decision function. The central idea of SVM is to find a maximum marginal hyperplane in multidimensional space that best divides the dataset into classes.
SVM uses a technique called *kernel trick*. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space, converting non-separable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem.

In this work we'll see the application of the SVM model with different types of kernel, in order to see the results in accuracy we obtain and to establish which is the best model.
For this purpose, it was only necessary to change the kernel parameter in the "svm.SVC(c, kernel, degree)" instruction, and in the case we're considering the polynomial kernel we also establish the degree of the polynomial, setting the parameter 'degree'.

First, we take into account ***kernel = "linear"***.
It is one of the most common kernels to be used. It is mostly used when there is a *large number of features* in a dataset, as in the case we're analysing. It is largely applicable in text classification. Indeed, through linear kernel, an accuracy of 0.983 is obtained.
The results are showed in Figure 3, where we can see also the results obtained from the classes, considered individually. From the confusion matrix, we can observe how the main number of samples misclassified is on the third row of the matrix, row which corresponds to the functions belonging to the sort class.

	precision	recall	f1-score	support
encryption	0.98	0.99	0.98	256
math	0.98	0.99	0.99	456
sort	0.97	0.89	0.93	112
string	0.98	0.99	0.99	391
accuracy			0.98	1215
macro avg	0.98	0.97	0.97	1215
weighted avg	0.98	0.98	0.98	1215

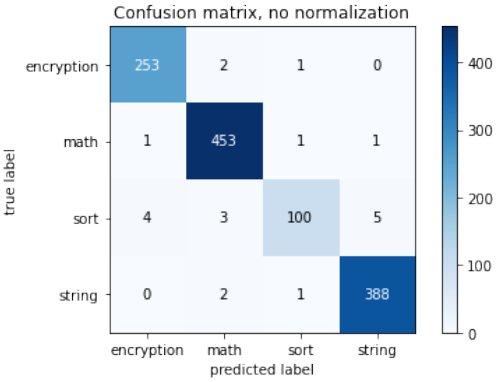

Figure 3 Left: table with the classification report with the SVM method, kernel='linear'. Right: confusion matrix.

In fact, we can see that there are five samples classified by the SVM as string manipulation functions, while in reality belong to sort function class; four samples are classified as encryption and tree as math, while in reality in both cases are sort functions.
Regarding the classification of the remaining samples, it is almost perfectly done.
Therefore, if the main purpose of our work is to find the sort functions, considering the linear SVM will cause some errors.
At the same time, such result is understandable. In fact, in the given dataset, the number of samples belonging to sort functions is smaller with respect to the other types: there are only 537 sort samples on 6037, so the model can train less in order to recognize this type of functions.

Now we consider ***kernel = "polynomial"***.
The polynomial kernel represents the similarity of vectors training samples in a feature space over polynomials of the original variables, allowing learning of non-linear models. It looks not only at the given features of input samples to determine their similarity, but also combinations of these. Initially the degree parameter is set at 2. The accuracy obtained is 0.663.
It is represented in Figure 4. In this case, there is no success in classifying sort functions (it doesn't manage in recognize anything) and just a little in

	precision	recall	f1-score	support
encryption	0.99	0.84	0.91	256
math	0.57	0.99	0.73	456
sort	0.00	0.00	0.00	112
string	0.66	0.36	0.46	391
accuracy			0.66	1215
macro avg	0.56	0.55	0.52	1215
weighted avg	0.64	0.66	0.61	1215

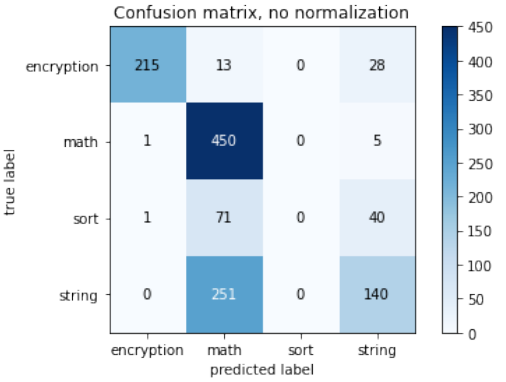

Figure 4 Left: table with the classification report with the SVM method, kernel='polynomial', degree=2. Right: confusion matrix.

classifying string manipulation ones. It is recognizable looking at the sort and string classes' rows in the classification report. We can observe it also from the confusion matrix: the darkest cells are not all on the diagonal; the misaligned are right in correspondence of the sort and string functions.
Then we can see that the most of the functions misclassified are the ones which have been erroneously predicted as math functions and a few classified as string functions. In particular most of the math prediction samples indeed were string manipulation and sort functions. Therefore we can easily conclude that, unless we're only interested in classifying encryption functions (and also in this case with linear kernel we would have better results) the polynomial kernel is not a recommendable method.

More the degree is increased, worst the results will be (accuracy will decrease). In particular, more the degree increase, more the samples are classified as math functions instead of sort and string functions.
In fact, considering a degree equal to 7, we have an accuracy of 0.500. We can do the same considerations done for degree=3, but in this case also the number of samples classified as string manipulation is very low. Despite this, the precision is still almost good (0.78). The previous values can be seen in Figure 5.

	precision	recall	f1-score	support
encryption	0.99	0.57	0.72	256
math	0.43	1.00	0.60	456
sort	0.00	0.00	0.00	112
string	0.78	0.02	0.03	391
accuracy			0.50	1215
macro avg	0.55	0.40	0.34	1215
weighted avg	0.62	0.50	0.39	1215

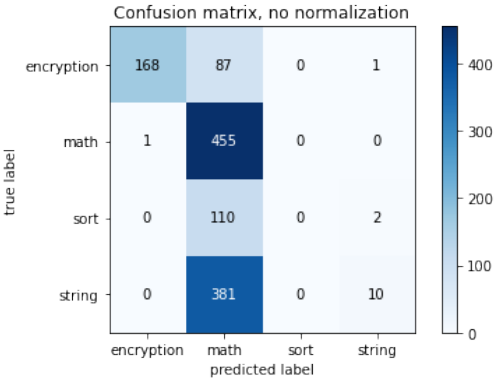

Figure 5 Left: table with the classification report with the SVM method, kernel='polynomial', degree=7. Right: confusion matrix.

Recurring to ***sigmoid kernel***, is still unsuccessful: an accuracy of 0.762 is obtained, sort samples are not predicted in the right way and there are different type of misclassifications.

Instead, using the ***rbf kernel*** we get more acceptable results: an accuracy of 0.884, a few good predictions of sort classes samples (a little better than sigmoid) and quite successful predictions in the classification of the other three classes.

5.2 Decision Tree

Decision trees are one of the most used machine learning algorithms given their intelligibility and simplicity. This model uses, as predective model, a decision tree to go from observations about an item (represented in the branches) to conclusions about the item's target value (in the leaves). Tree models, where the target variable can take a discrete set of values, are called classification trees: in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.
Applying the Decision Tree model, very good predictions are made, except for a few samples misclassified. An accuracy of 0.981 is obtained. As it is recognizable from the confusion matrix in Figure 6, all the darkest cells are on the diagonal. Also from the classification of the sort functions we obtain results above 90%.

	precision	recall	f1-score	support
encryption	0.98	0.98	0.98	256
math	0.99	0.99	0.99	456
sort	0.91	0.94	0.92	112
string	0.99	0.98	0.99	391
accuracy			0.98	1215
macro avg	0.97	0.97	0.97	1215
weighted avg	0.98	0.98	0.98	1215

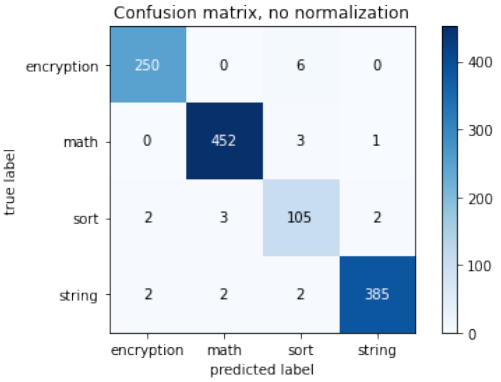

Figure 6 Left: table with the classification report with the Decision Tree. Right: confusion matrix.

6 Conclusions

In this work is given a dataset, that is first subjected to a preprocessing which considers only the first command of every instruction, Different models have been applied to this preprocessed dataset, in order to compare the results obtained in classifying the samples in four different classes.
The most unsuccessful results are in the classification of sort functions. It is understandable: in fact in the dataset sort samples are the ones in lower number. So the model, in the case of sort samples, can't train as well as it does with the other samples.
We have seen how, between the models considered, the most successful results have been obtained through the Decision Tree model and the SVM linear model. Here accuracy tends to 1, value that is reached if all the samples of a certain class have been classified correctly.
On the other hand, SVM model with polynomial kernel brings us to worst results. More the degree of the kernel is increased, more the results get worse. Unacceptable results are get also through SVM models characterised by the sigmoid kernel or the rbf kernel.