Michela Proietti 1739846

ABSTRACT

In this homework, I explain the preprocessing operations that have been applied to the dataset in order to get useful features for the classification task. Moreover, there is a comparison between the results obtained by using different classification methods, namely SVM and decision trees.

PREPROCESSING

The given dataset contains functions belonging to four different classes: Encryption, String, Math and Sort. This means that the problem we are trying to solve is a multiclass classification problem, so we need to be able to predict, for each new instance, one of the above classes. Each item in the dataset contains a unique ID for each function, the semantic, which represents the label of each function, a list of all the assembly instructions of each function and finally the control flow graph. Functions belonging to different classes have different properties and we have to exploit these differences in order to be able to correctly classify the instances. In particular, we know that encryption functions are very complex, thus having also a complex CFG, and contain a lot of nested for. On the contrary, sorting functions have a simple logic and usually have only one or two nested for, which means that they are simpler than the previous ones. For this reason, I decided to obtain information about the complexity of the functions using the CFG and in particular the number of nodes, the number of cycles and the cyclomatic complexity, which represents the number of independent paths within a source code and can be computed as:

$$M = E - N + 2P$$

where E is the number of edges of the graph, N is the number of nodes and P is the number of connected components. Moreover, another big difference is in the type of instructions that are used in these functions. In particular, encryption functions use a lot of xor, shifts and bitwise operations, math functions use a lot of arithmetic operations and a lot of floating point instructions with special register xmm, and finally string manipulation and sorting functions use a lot of comparisons and swap of memory locations. For this reason, I used the list of assembly instructions available in the dataset to compute the number of instructions of each type. Finally, I have created a pandas data frame, characterized by 10 features.

In the following picture, we can see the first 10 items in the data frame with the relative labels.

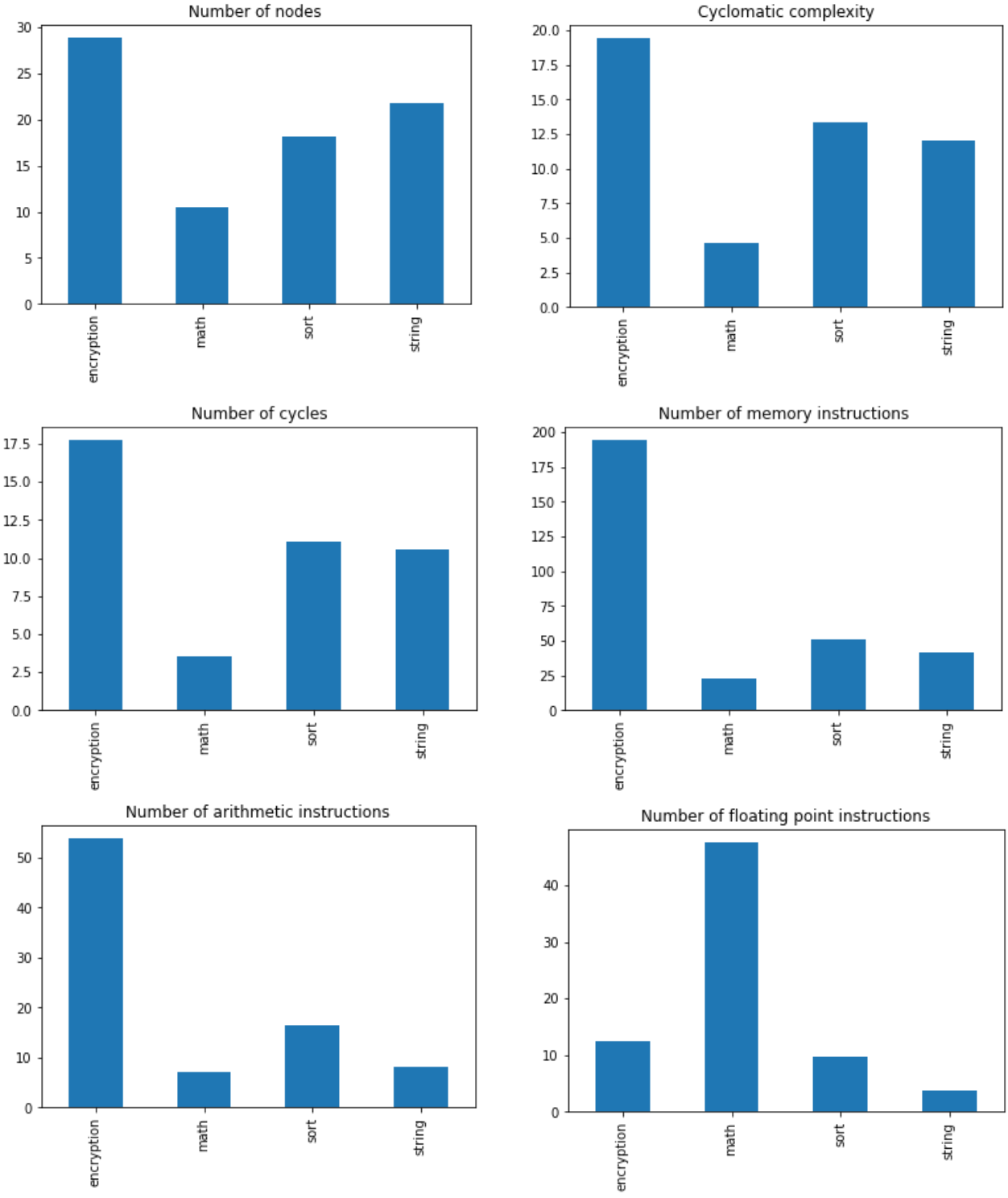
	n_nodes	cyclomatic_complexity	n_cycles	n_memory_ins	n_arithmetic_ins	n_float_ins	n_bitwise_ins	n_jumps	n_comparisons	n_calls	label
0	23	9	8	54	5	4	54	15	8	6	string
1	15	7	6	10	6	52	13	12	4	1	math
2	18	17	16	188	65	17	247	16	17	0	encryption
3	2	1	0	7	1	6	5	1	1	1	math
4	22	15	11	22	10	3	32	14	10	7	sort
5	35	29	28	118	2	0	249	1	1	30	encryption
6	21	21	20	167	90	93	156	20	8	0	encryption
7	3	3	2	63	2	0	127	2	2	0	encryption
8	15	13	12	61	25	1	36	14	20	0	sort
9	14	4	3	37	6	1	34	5	1	7	string

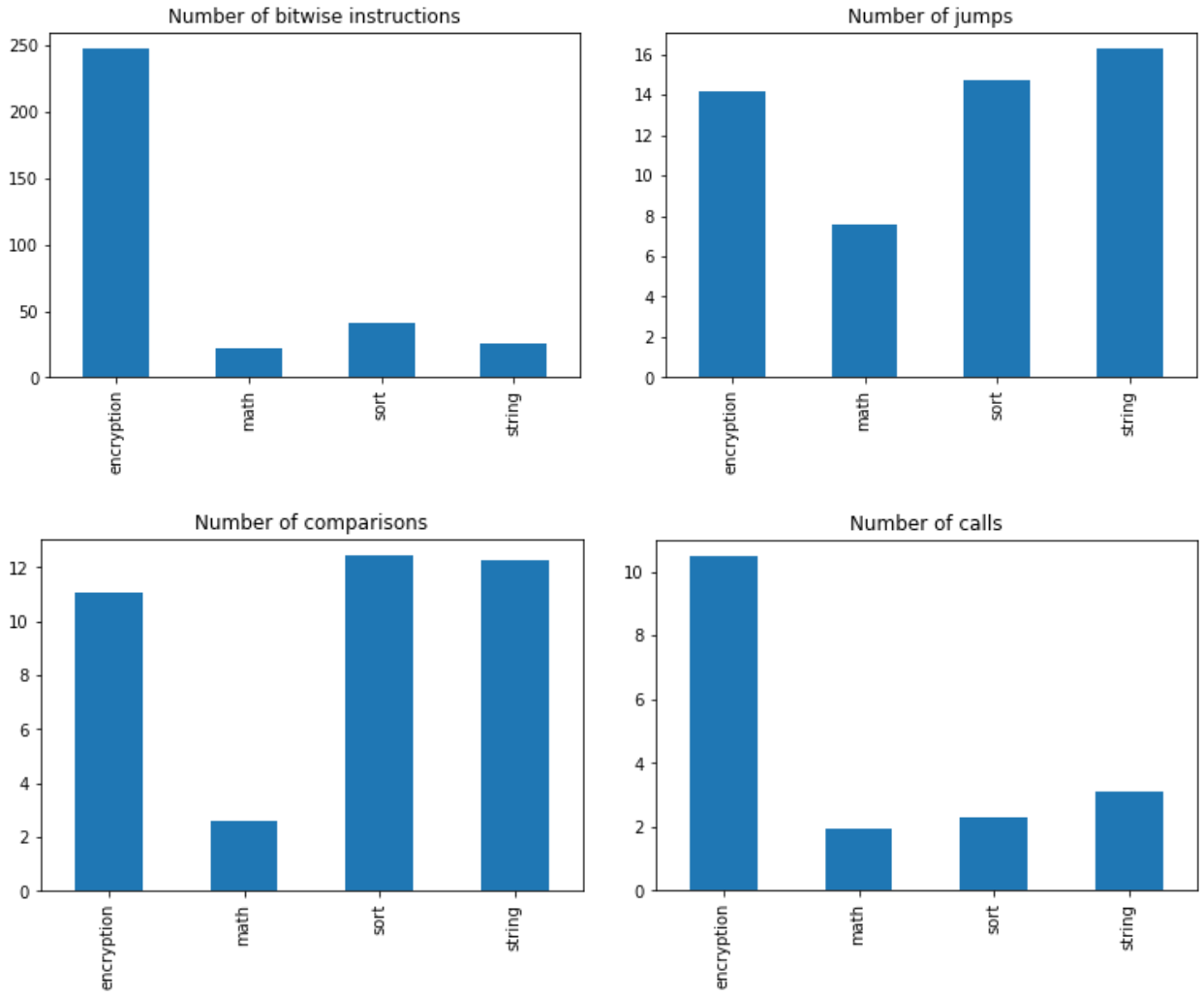
We can further investigate the properties of the data frame, by analysing the following table.

	n_nodes	cyclomatic_complexity	n_cycles	n_memory_ins	n_arithmetic_ins	n_float_ins	n_bitwise_ins	n_jumps	n_comparisons	n_calls
count	14397.000000	14397.000000	14397.000000	14397.000000	14397.000000	14397.000000	14397.000000	14397.000000	14397.000000	14397.000000
mean	18.574217	11.477669	9.872543	67.124540	18.738973	20.801486	70.839342	12.694103	9.063902	3.898659
std	22.189614	17.067398	16.687443	138.409361	45.631754	57.272675	203.778394	14.236532	11.488972	11.795867
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	7.000000	3.000000	1.000000	13.000000	3.000000	1.000000	8.000000	4.000000	1.000000	1.000000
50%	13.000000	8.000000	6.000000	36.000000	7.000000	5.000000	23.000000	9.000000	6.000000	1.000000
75%	23.000000	14.000000	12.000000	68.000000	18.000000	14.000000	56.000000	16.000000	12.000000	4.000000
max	386.000000	384.000000	383.000000	3354.000000	966.000000	1541.000000	3774.000000	158.000000	145.000000	384.000000

The first row represents the number of non-null values that we have for each feature and because of the way the features have been constructed, there is no null value and so we have 14397 values for each feature. The mean represents the mean of all the values for each feature, but this property is not so relevant, because we want to focus mostly on the differences between classes. Therefore min, which is the minimum of the values for each feature, and max, which is the maximum of those values, are more important. In fact, we notice that there is a great difference between the minimum and the maximum value of each feature, but we still don't know if those differences are present between the values of items that belong to the same class or if they highlight differences between the classes and therefore are relevant for the classification task. In order to better understand the properties of each class, items have been grouped with respect to their class and in the following pages it is possible to see histograms showing the mean values of the features for all the four classes. As we can easily see from the plots, the number of nodes, the number of cycles and the cyclomatic complexity have the highest values in the case of the encryption functions, that as we have already said, are fairly complex functions. On the other hand, the functions with the simplest CFGs are the math functions, that differently from all the other types of functions have instead the greatest number of floating-point instructions. We must specify that the feature that represents the number of floating-point instructions also take into account the number of times in which the xmm register is used. These could have been considered as two different features, but since they let us identify the same type of function, they are considered as one. Finally, accordingly to what we have said before, we can see that the classes with the highest number of jumps and comparisons

are the sorting and the string manipulation functions. So, the features we have extracted are coherent to what we had expected. After having created the data frame, data has been split into a training set, which is made up of 11517 samples and a test set composed by 2880 samples.





EXPERIMENTS

The first method that has been used is a Support Vector Machine. A SVM classifier tries to find the maximum margin hyperplane separating different classes. There could be situations in which data is not linearly separable in the input space, so in these cases it is possible to apply some transformations to the data in order to be able to split it effectively. In our case, the SVM classifier has been tested by using different types of kernels, starting from the linear kernel, and then trying also non-linear kernels such as the RBF, the polynomial and the sigmoid kernel. In general, it is common practice to train a SVM with a linear kernel first, because it is less expensive, and to apply non-linear kernels only if it fails in correctly classifying the instances. In fact, SVM with non-linear kernels have more hyperparameters to tune and since they are more complex model they tend to overfit the data, not being able to generalize and correctly classify new instances. The other model that has been used is a decision tree. At each node of

the tree, we consider just one feature and impose a threshold in order to split samples according to the value of that feature. Entropy can be used to measure the purity of the sub-split and it is defined as follows:

$$Entropy = -\sum_{k=1}^{n} p_{i,k} \log(p_{i,k}) \ \ where\ p_{i,k} \neq 0$$

where $p_{i,k}$ represents the probability that sample i belongs to class k and it is computed as the number of samples of class k over the total number of samples. Another way to measure the purity is by using the Gini impurity parameter, which is computed as:

$$Gini_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

The problem with decision trees is that they are very powerful and therefore there is the risk to overfit data. For this reason, I have done several experiments in which I imposed a depth limit and showed the accuracy obtained using gini and entropy. Finally, I selected the lowest depth limit that let us have the highest accuracy.

In order to compare the results obtained with the different models, several metrics have been used. First of all, we have considered accuracy, which is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy represents the number of samples that have been correctly classified over the total number of samples. In the previous formula, TP (true positive) is the number of samples that have been correctly classified as belonging to a certain class, TN (true negative) is the number of samples correctly classified as not belonging to a certain class, FP (false positive) is the number of samples that have been mistakenly classified as belonging to a certain class and FN (false negative) is the number of samples that have been mistakenly classified as not belonging to their actual class. Since the dataset is not particularly unbalanced, accuracy is a good metric to characterize the performances of the models. However, since the results that we get could depend on the way the train and test set are split, k-folds cross validation has been used. More precisely, the dataset has been randomly split into 5 subsets, or folds, with test size equal to 33,3% of the total size. Then, for each of these folds, the cross_val_ score method builds the model to evaluate on the 4 remaining folds and test it. This procedure is repeated for all the 5 folds and then the method returns the average of the 5 accuracy values obtained in the 5 experiments. Other metrics that have been used are precision and recall.

$$Recall = \frac{TP}{TP + FN}$$

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

Recall represents the ability of the model to avoid false negatives, while precision represents the ability of the model to avoid false positives. Since in a malware it is important to find encryption functions, we may want all the encryption functions to be classified correctly, so we may should prefer having high recall instead of having high precision. Another metric we could take into account is the F1-score, that is a combination of precision and recall:

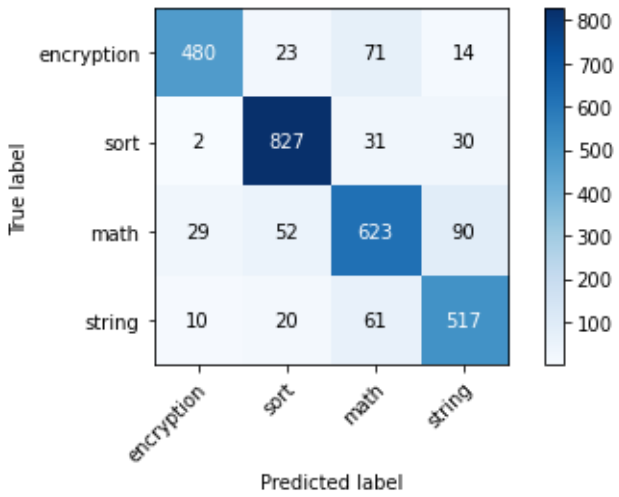
$$F1 - score = \frac{2(precision \cdot recall)}{precision + recall}$$

RESULTS

First of all, I show the results obtained using a SVM with different kernels. The accuracy values are the ones obtained with cross validation.

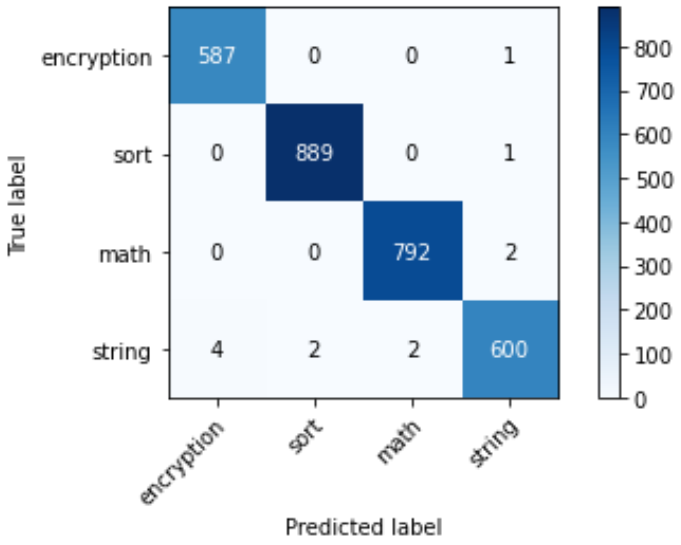
MODEL	ACCURACY	TRAINING TIME (s)
SVM with linear kernel	0.877 0.866	109.28 1.14 (scaled data)
SVM with polynomial kernel	0.478	3.88
SVM with RBF kernel	0.806	2.32
SVM with sigmoid kernel	0.636	2.24

As we can see, the training time of the SVM with linear kernel is greater than the others, and this might sound strange since computations should be easier and faster. However, when we use SVMs it is very important to scale our data, because kernel values depend on the inner product between feature vectors and therefore large attribute values might cause numerical problems. If we add this other preprocessing step, by scaling the data using a simple StandardScaler from sklearn.preprocessing, we get almost the same accuracy but with a lower training time. As we can see, there is no point in using non-linear kernels, since we get better performances using a simple linear kernel. The following picture represents the confusion matrix we obtain using a linear kernel, while the table shows the values obtained for the other metrics. As we can see, the sort class is the one for which the model does the fewest mistakes, but it is also the one with the fewest recall. However, as we have said before, we are particularly interested in identifying the encryption functions and for them the recall is of 82%. That is an acceptable result, but we may want something more.



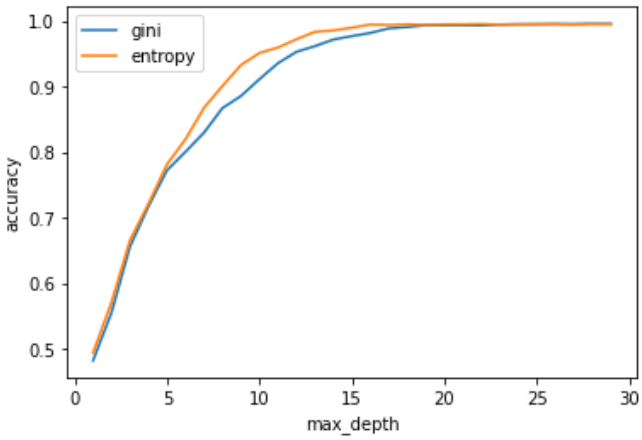
	PRECISION	*RECALL*	*F1-SCORE*	*SUPPORT*
ENCRYPTION	0.92	0.82	0.87	588
MATH	0.90	0.93	0.91	890
SORT	0.79	0.78	0.79	794
STRING	0.79	0.85	0.82	608
ACCURACY			0.85	2880
MACRO AVG	0.85	0.85	0.85	2880
WEIGHTED AVG	0.85	0.85	0.85	2880

For the decision tree, we get the results shown in the following page. As we can immediately notice from the confusion matrix, the accuracy of this method is much higher than the one we obtained with the SVM, and we can see that it does only 12 errors on a total of 2880 test samples. If we look at the table that shows also the values obtained for the other metrics, we see that they are all very close to 1.



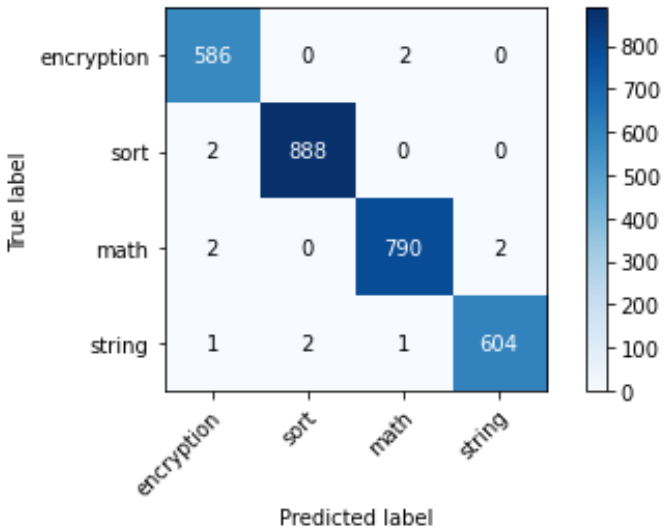
	PRECISION	*RECALL*	*F1-SCORE*	*SUPPORT*
ENCRYPTION	0.99	1.00	1.00	588
MATH	1.00	1.00	1.00	890
SORT	1.00	1.00	1.00	794
STRING	0. 99	0. 99	0. 99	608
ACCURACY			1.00	2880
MACRO AVG	1.00	1.00	1.00	2880
WEIGHTED AVG	1.00	1.00	1.00	2880

As we have said in the previous section of the report, the problem with decision trees is that they are very powerful, and we could risk to overfit data. The following plot shows the accuracy obtained by using different depth limits for the tree and using either gini or entropy.



As we can see, there is not much difference between the results obtained using gini and those obtained using entropy. In both cases, we see that the lowest depth that let us have the highest accuracy is around 17/18. I decided to use entropy with a depth limit of 17 and these are the results I got.

	PRECISION	*RECALL*	*F1-SCORE*	*SUPPORT*
ENCRYPTION	0.99	1.00	0.99	588
MATH	1.00	1.00	1.00	890
SORT	1.00	0.99	1.00	794
STRING	1.00	0. 99	1.00	608
ACCURACY			1.00	2880
MACRO AVG	1.00	1.00	1.00	2880
WEIGHTED AVG	1.00	1.00	1.00	2880



The following table compares the accuracy values obtained using cross validation and the training times of the decision tree with and without the depth limit.

	Accuracy	**Training time (s)**
Without depth limit	0.994	0.043
With depth limit	0.994	0.0379

We can easily see that the accuracy remains the same and the training time, which was already low, decreases. This is therefore the model which was used to get the predictions for the unlabelled dataset.