Report Homework 1
Machine Learning

By Farooq Ahmad Wani
1946707

Project Overview

The problem statement is to solve the classification task for the given dataset. The dataset is the JSON file containing many entries where each entry belongs to one or another class. We have to devise the algorithmic procedure to classify a similar type of entries where the class of entry is missing.

Dataset

The training dataset contains 6073 distinctive records. The Dataset is of the form of JSON structure where the general structure is:

{ "id": "containing the id of record",
"semantic": “containing the class label”
“lista_asm” : “containing the list of instructions”
“cfg” : “containing the control flow graph information”

Firsly , I loaded the dataset in my python project and renamed some columns which are given as:

columns = {'lista_asm': 'Instructions', 'cfg': 'FlowGraphs' ,'semantic' : 'Class'}

I have captured the general highlight of the data using the pandas-profiling feature of the python and the following results I got from it.

Dataset statistics

Number of variables	3
Number of observations	6073
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	142.5 KiB
Average record size in memory	24.0 B

Variable types

CAT	2
UNSUPPORTED	1

Here you can see that we have information on only the three variables though we have four variables in the original JSON from the dataset. This is because I had eliminated the column named “Id” as it is not going to contribute to the classification.
Other interesting stats I got from the Data are :

`Instructions` has unique values	Unique
`FlowGraphs` is an unsupported type, check if it needs cleaning or further analysis	Unsupported

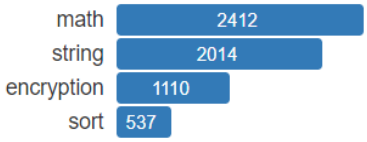
Here we got the confirmation that every record has unique values of instructions and the Flowgraphs; The (“cfg”) column is not understandable to python right now as it is a packed object and needs to be

unpacked.

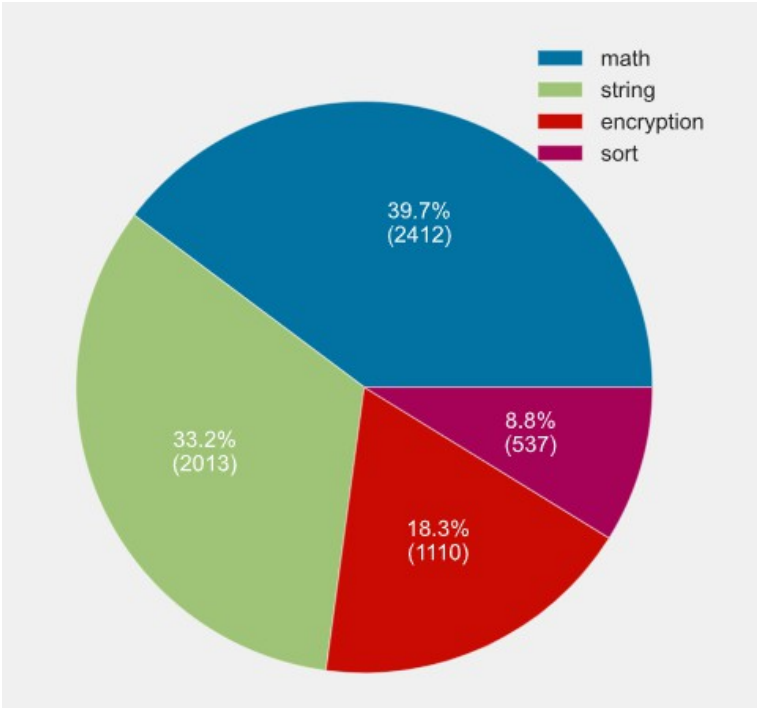
The number of enteries or rows for each class by numbers are as:

Class
Categorical

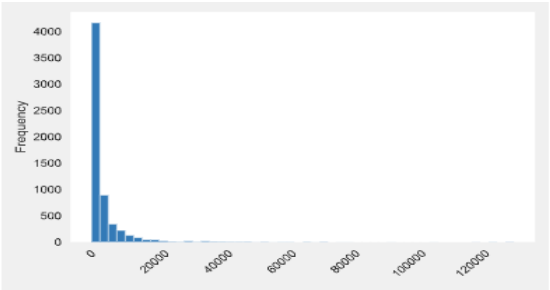
Distinct	4
Distinct (%)	0.1%
Missing	0
Missing (%)	0.0%
Memory size	47.4 KiB



The percentage contribution of each of the Classes is given by the following pie chart, it is evident that the contribution by the “sort “ class is very less compared to the other, which means we have a slightly unbalanced dataset.



we also got the varying lengths of the instructions for every entry which is given as :



Histogram of lengths of the category

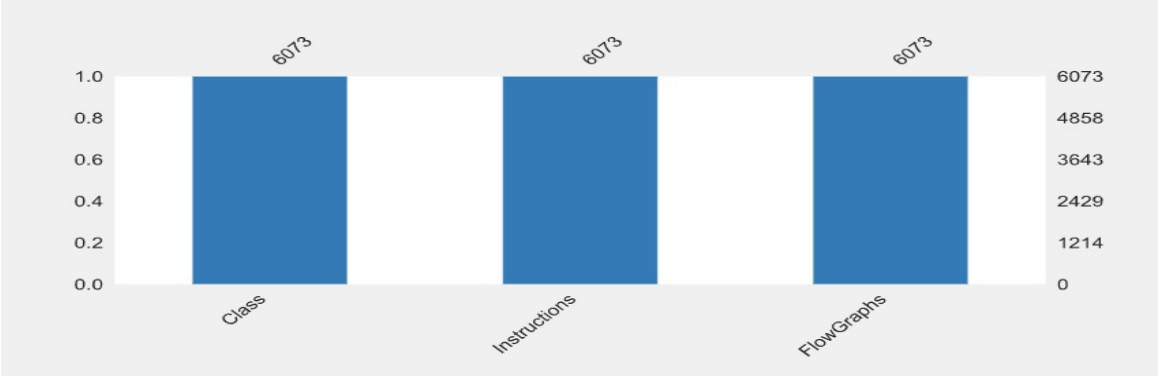
Length

Max length	132832
Median length	1614
Mean length	3947.266754
Min length	8

Hence from this observation it is evident that we can use this property as one of the feature for

clasification.

The Matrix plot of the dataset shows that we don't have any missing value.



Hence our dataset is complete so we don't have to tackle the missing value problem.

Preprocessing and Feature Extraction

After analyzing the data I used two separate procedures to get the features for the classification algorithm :

Firstly, I used the FlowGraph information to get the three features and those three features are :
Node_length: gives the number of nodes that are present in the instruction set.
Double_length: gives the Ratio of the total nodes (node_length) to the number of bi-directional nodes.
Single_length: gives the Ration of the total nodes to the number of unidirectional Nodes.

Secondly, I used the instruction information, there I used the Vectorization technique of Td_idf. The Td_idf is the term document _inverse document frequency.I looped over each row got the instructions set and parsed it to get its operators then merged all these operators present in one row or entry, After that, I sent this Information to the Vectorization function to make the vectors of it. The function returns the frequency of each instruction of assembly language in each row, as I also used the ngram(1,3) it also kept the order and sequence of terms for one to three units which resulted in a much bigger vector than the number of unique terms.

After that, I used the Hstack to combine these two feature techniques as the output from these techniques were of different formats.

Now I have the Feature dataset ready I need to divide it into the training and the testing sets. Here used the help of Sklearn and used its function to give me the training dataset and testing dataset.

MODEL

I selected the logistic regression model of the sklearn library and passed the training dataset without any parameter tuning and the results were quite bad as expected. The Ngram used for this first training was (1,1)

	precision	recall	f1-score	support
encryption	0.89	0.87	0.88	312
math	1.00	0.96	0.98	719
sort	0.00	0.00	0.00	167
string	0.75	0.99	0.85	624
accuracy			0.87	1822
macro avg	0.66	0.70	0.68	1822
weighted avg	0.80	0.87	0.83	1822

after getting this bad accuracy and f1-score, I find out that the main cause of this lower score is that the “sort' and “string” classes are using an almost similar type of assembly instruction.

Therefore I changed the ngram parameters, where now not only the number of instructions contribute but also the sequence of instructions will also contribute to classification.

After doing that , the results saw the improvement as shown below:

	precision	recall	f1-score	support
encryption	0.96	0.95	0.95	312
math	0.99	0.98	0.98	719
sort	0.84	0.63	0.72	167
string	0.91	0.99	0.95	624

0

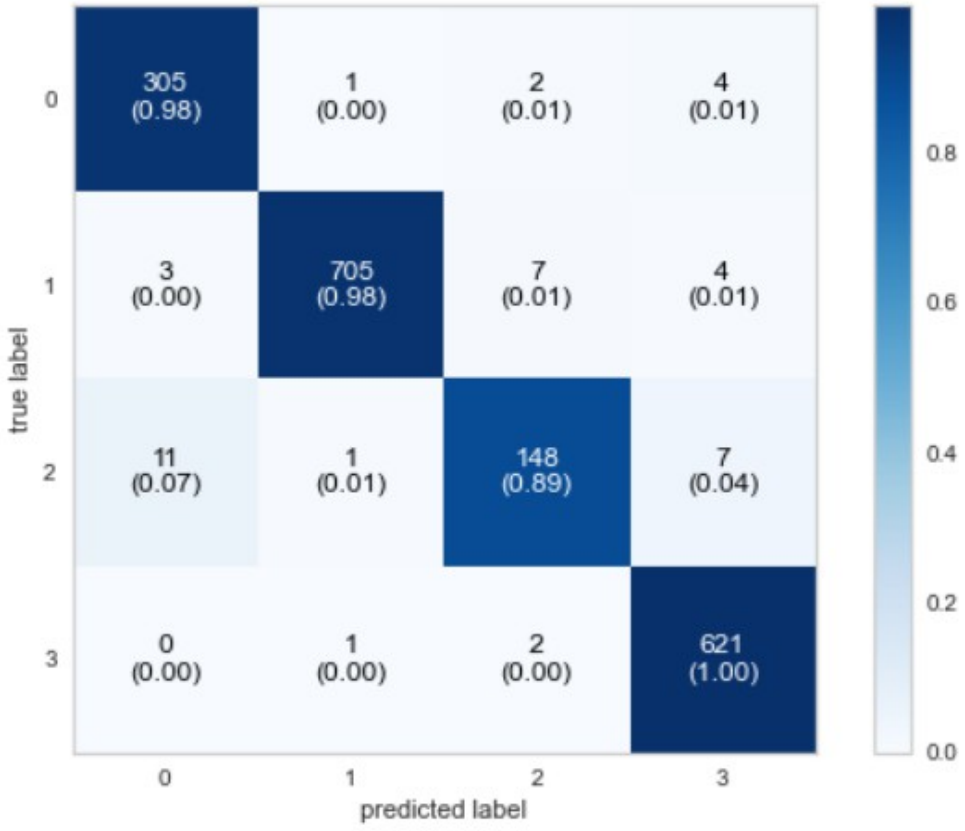
but still, the performance was low so I tried a different solver and replaced the saga solver with the lib linear solver.
There was an improvement but still, it was not that high, I saw that by default this solver uses the L2 regularization.
As I was not overfitting I replaced that with L1 regularization and the results are better.

	precision	recall	f1-score	support
encryption	0.96	0.98	0.97	312
math	1.00	0.98	0.99	719
sort	0.93	0.89	0.91	167
string	0.98	1.00	0.99	624
accuracy			0.98	1822
macro avg	0.96	0.96	0.96	1822
weighted avg	0.98	0.98	0.98	1822

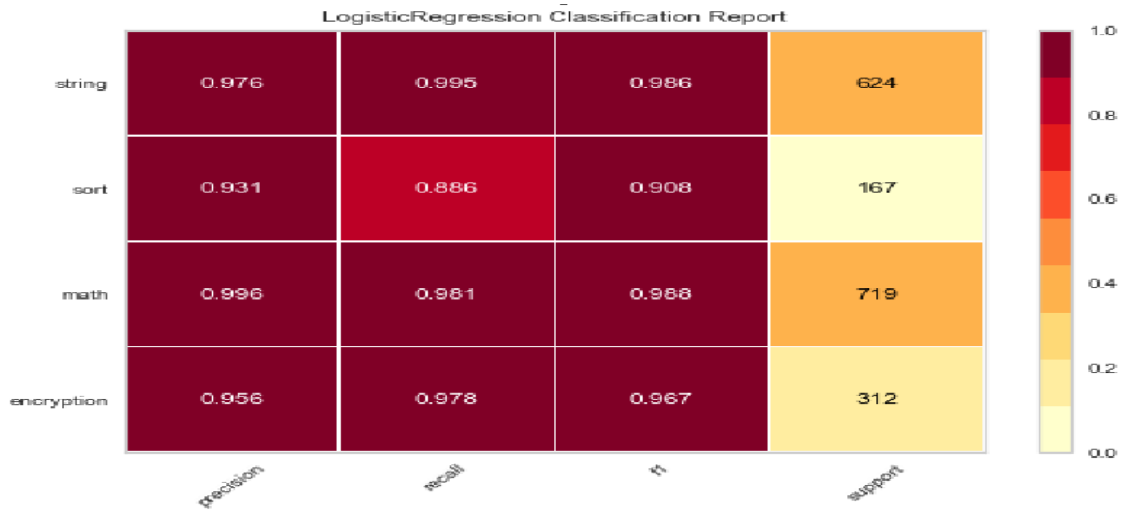
Evaluation and Visualization

I used the confusion matrix and other evaluation techniques to check the results and the weights of the classifier. I also tried to see whether the features I extracted from flow graphs are contributing to the results or not. I used the three very important libraries from python which are Sklearn for confusion matrix, yellow-brick for different visualizations, and the most important eli5 for checking the feature importance.

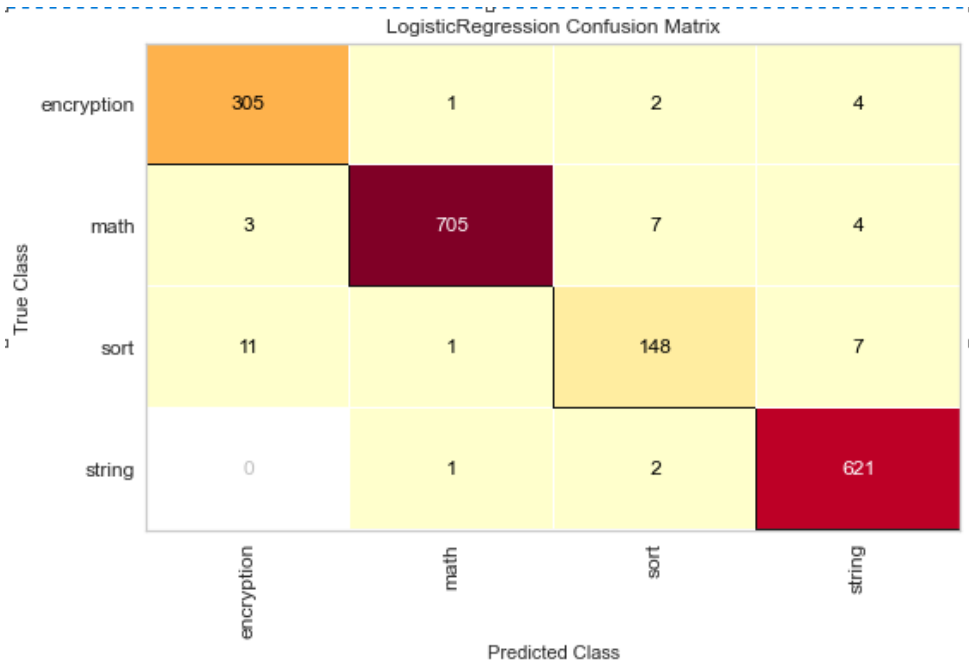
The results are discussedd below:
This is the confusion matrix as we can see that most of the entries are classified correctly, though there is scope for improvement.



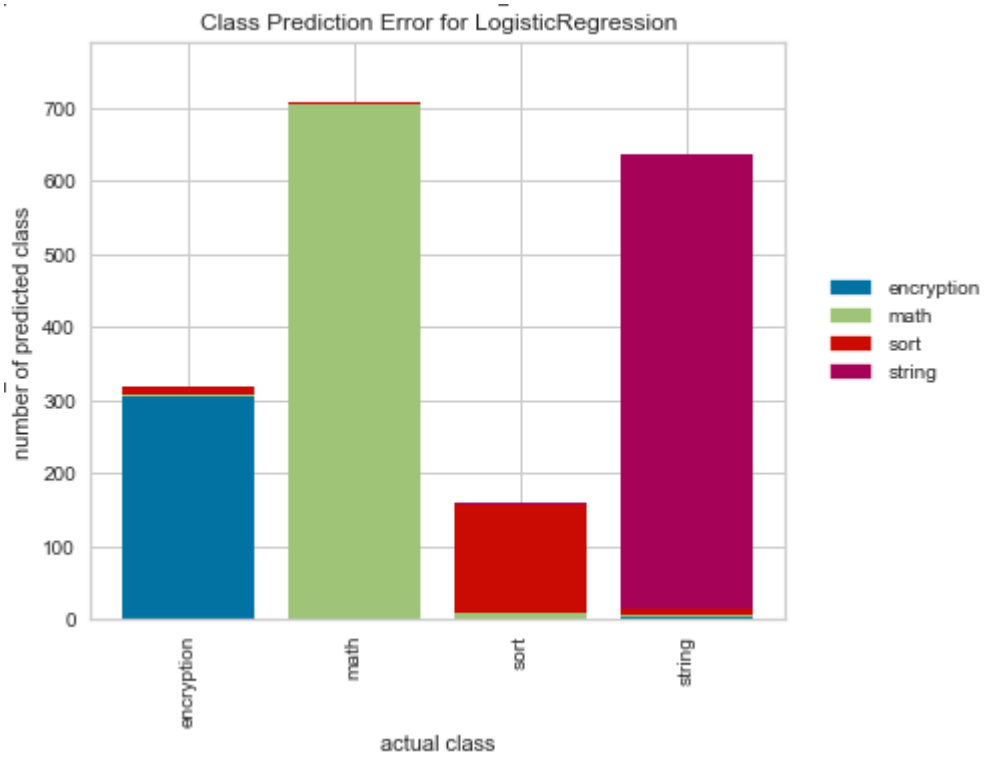
The yellow-brick visualization of the test set shows that the “sort” has minimum accuracy out of all:



To get a more intuitive understanding the result can be viewed as:

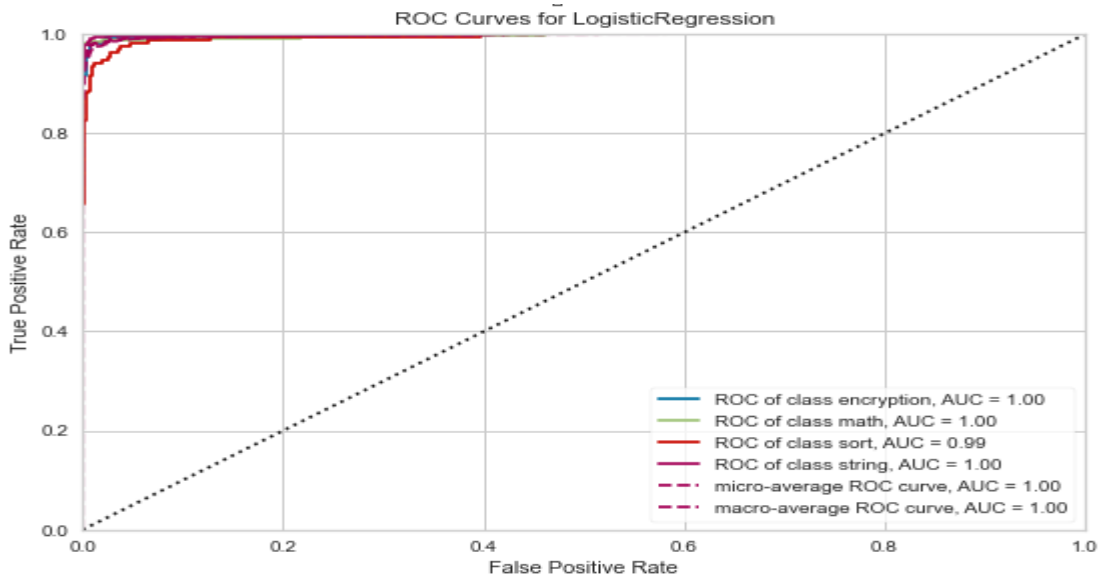


The bartchart view of the above solution can be viewed as :



The Area under the curve of evaluation can be viewed as :

The evident thing from that is the result given by the model is very good and the "sort" class has a minimum area under its curve which is expected as it has the lowest accuracy score out of all the four classes.



Feature importance

Finally, to check which features are contributing to the result I have used the eli5 to check the weights of each feature and the result is as shown in fig:

y=encryption top features		**y=math** top features		**y=sort** top features		**y=string** top features	
Weight[?]	**Feature**	**Weight**[?]	**Feature**	**Weight**[?]	**Feature**	**Weight**[?]	**Feature**
+12.404	xor	+35.937	movsd	+18.093	nop	+20.696	je
+11.211	shl	+11.263	ucomisd	+15.953	ret test	+19.358	js
+9.388	rol	+7.429	lea	+11.147	cmp jae	+12.782	test
+8.439	movaps movaps	+7.327	movsxd add jmp	+9.687	mov nop	+11.906	jmp
+7.369	shr	+7.022	movapd	+9.370	cmp	+10.621	jmp jmp jmp
+7.087	movzx	+1.500	movlpd	+9.188	call test jle	+9.840	mov cmp je
+4.481	mov mov mov	+0.280	call	+8.831	imul	+9.079	mov test
+3.612	and	+0.174	legth_double	+7.903	movsxd mov mov	+8.624	ret mov
+2.715	add mov	+0.036	<BIAS>	+6.544	mov sub	+7.752	cmp
+2.333	ror	+0.012	movss	+6.479	cmp mov	+6.055	mov jmp mov
+1.756	add	-0.032	length_single	+6.314	cmp jbe	+5.996	mov add pop
+1.652	sub sub	-0.099	length_node	+4.908	mov cmp jge	+5.862	mov mov test
+1.649	xor mov	-0.819	add	+3.598	mov mov call	+5.738	movsx
+1.637	mov mov	-0.905	imul	+3.556	lea	+4.713	jmp jmp
+0.412	mov add mov	-1.020	shr	+3.127	movsxd mov	+4.559	ret
+0.027	length_single	-2.221	pop	+2.924	mov cmp jle	+4.323	cmp jl
+0.025	length_node	-3.905	mov	+2.473	cmp jb	+3.990	movdqa
-0.123	legth_double	-5.139	test	+2.403	push push	+3.763	pop
-1.726	<BIAS>	-5.898	cmp	*… 10 more positive …*		+3.068	movzx mov
-4.000	je	-5.998	movzx	*… 8 more negative …*		+2.759	mov add
-4.869	jmp	-7.278	xor	-2.401	shr add mov	*… 9 more positive …*	
-5.529	movsxd			-2.491	shr	*… 6 more negative …*	
-15.177	movsd			-3.293	ret mov	-1.736	mov nop
-23.376	cmp			-3.710	js	-2.056	mov mov call
-28.295	test			-4.036	mov cmp je	-2.240	lea
				-4.112	rol	-2.986	xor mov
				-4.223	mov add pop	-3.013	xor
				-4.339	je mov	-3.452	<BIAS>
				-4.969	mov test	-3.772	push push
				-8.358	je	-10.513	call test jle
				-12.065	movsd	-13.258	imul
				-14.723	movzx	-16.551	movsd

This proves that our method of extracting features like node_length, double_length, and single_length is contributing to the final result of classification and have very good positive weights for math and encryption classes. At the same time to classify sort and string many more features are considered and have further scope for improvement.
Finally, I want to show how these weights contribute by applying them to one example.

y=encryption (probability 0.000, score -9.932) top features		
Contribution[?]	Feature	Value
+1.293	length_node	52.000
+0.115	length_single	4.333
+0.064	xor	0.005
+0.015	add	0.008
-0.174	je	0.044
-0.182	jmp	0.037
-0.213	legth_double	1.733
-0.575	test	0.020
-1.726	<BIAS>	1.000
-8.549	movsd	0.563

y=math (probability 1.000, score 18.518) top features		
Contribution[?]	Feature	Value
+20.241	movsd	0.563
+1.818	movapd	0.259
+1.599	ucomisd	0.142
+0.302	legth_double	1.733
+0.036	<BIAS>	1.000
+0.005	call	0.017
-0.007	add	0.008
-0.038	xor	0.005
-0.065	mov	0.017
-0.104	test	0.020
-0.140	length_single	4.333
-5.130	length_node	52.000

y=sort (probability 0.000, score -10.884) top features		
Contribution[?]	Feature	Value
+0.123	nop	0.007
-0.111	length_single	4.333
-0.364	je	0.044
-0.384	legth_double	1.733
-0.966	length_node	52.000
-2.387	<BIAS>	1.000
-6.795	movsd	0.563

y=string (probability 0.000, score -12.670) top features		
Contribution[?]	Feature	Value
+0.901	je	0.044
+0.446	jmp	0.037
+0.260	test	0.020
+0.080	ret	0.018
+0.001	mov	0.017
-0.000	sub	0.004
-0.002	length_single	4.333
-0.016	xor	0.005
-0.100	movapd	0.259
-0.719	length_node	52.000
-0.746	legth_double	1.733
-3.452	<BIAS>	1.000
-9.322	movsd	0.563