Sapienza Università di Roma
Department of Computer, Control, and Management Engineering Antonio Ruberti
Artificial Intelligence and Robotics

Homework 1

Course: Machine Learning

Student: **Šikalo Amila**
Matricola: **1938032**

November 29, 2020

Data preprocessing

The data was first analyzed in terms of number of data entries or rows in the table written in JSONL file. It was concluded that data consists of columns: ID, CFG, assembler code and semantic. The ID is the ID of the data row. CFG is control-flow diagram describing how the code is running when compiled and assembly code is the code that describes this behavior. Semantic is supposed to represent the type of the code described by assembly and CFG. The semantic column consists of classes: encryption, math, sort and string.
As advised by professor, the data has to be processed in a way that feature vector for classifier (or classifiers) consist of several coordinates, such as attributes of the CFG graph such as number of nodes, cyclomatic complexity, as well as the dictionary of assembly routines obtained by counting the occurrences of assembly commands in the assembly code. These commands are divided into groups based on their meaning such as jump routines, call routines, arithmetic routines, floating point routines, etc.
The Python function is written to achieve this by creating new Pandas data frame with each row containing such a feature vector. The last column is semantic column from the original data, copied accordingly.
There are two Python functions used for graph specific calculations.

```
def cyclomatic_complexity(G):
        N = G.number_of_nodes()
        E = G.number_of_edges()
        P = nx.components.number_strongly_connected_components(G)
        return E - N + P

def number_of_simple_cycles(G):
        if G.number_of_nodes() > 150:
                return -1
        sc = nx.algorithms.cycles.simple_cycles(G)
        return len(list(sc))
```

Please note that determining the number of simple cycles in a graph is very time-consuming problem and for large graphs require lot of time. Therefore, it was observed that for certain rows in the table the function number of simple cycles of G does not return value in minutes. It was observed that excluding those graphs with the number of nodes larger than 150 is a good idea in order to circumvent this problem. This is shown in the code of number_of_simple_cycles function.

The following methods were used:

Support vector machines (SVMs)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are[1]:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features[2].

[1] Available at: https://scikit-learn.org/stable/modules/svm.html
[2] Available at: https://scikit-learn.org/stable/modules/tree.html

All other methods that have been tried weren't even close to results received in these two methods (for example: Gaussian Naive Bayes - Accuracy: 0.834, Logistic regression - Accuracy: 0.848).

The following configurations of the method have been tried:

SVM:

The following configurations of kernel function have been used:

```
svm_model1 = svm_classification(new_dataset, 'linear')
svm_model2 = svm_classification(new_dataset, 'rbf')
svm_model3 = svm_classification(new_dataset, 'poly')
```

Decision Trees:

The following configurations have been tried:

criterion{"gini", "entropy"}, where default="gini"

splitter{"best", "random"}, default="best"

The following combinations have been used:

```
nbc_model1 = decision_tree_classifier(new_dataset, 'entropy', 'best')
nbc_model2 = decision_tree_classifier(new_dataset, 'gini', 'random')
nbc_model3 = decision_tree_classifier(new_dataset, 'entropy', 'random')
nbc_model4 = decision_tree_classifier(new_dataset, 'gini', 'best')
```

Description of the evaluation method used, and obtained results using appropriate metrics.

In order to compare classifiers, standard metrics are used, that are available from scikit learn library. These are precision, recall and accuracy. Since precision and recall are calculated for each class, we use voting system based on the best result for each class and each metric. The classifier that appears mostly (achieves the best results for highest number of classes and metrics) will be declared as the best classifier within classifier type with respect to parameters used to classify data.

SVM classifier

1. linear

	precision	recall	f1-score	support
encryption	0.935	0.952	0.943	228
math	0.953	0.982	0.967	497
sort	0.835	0.760	0.796	100
string	0.979	0.954	0.966	390
accuracy			0.949	1215
macro avg	0.926	0.912	0.918	1215
weighted avg	0.948	0.949	0.948	1215

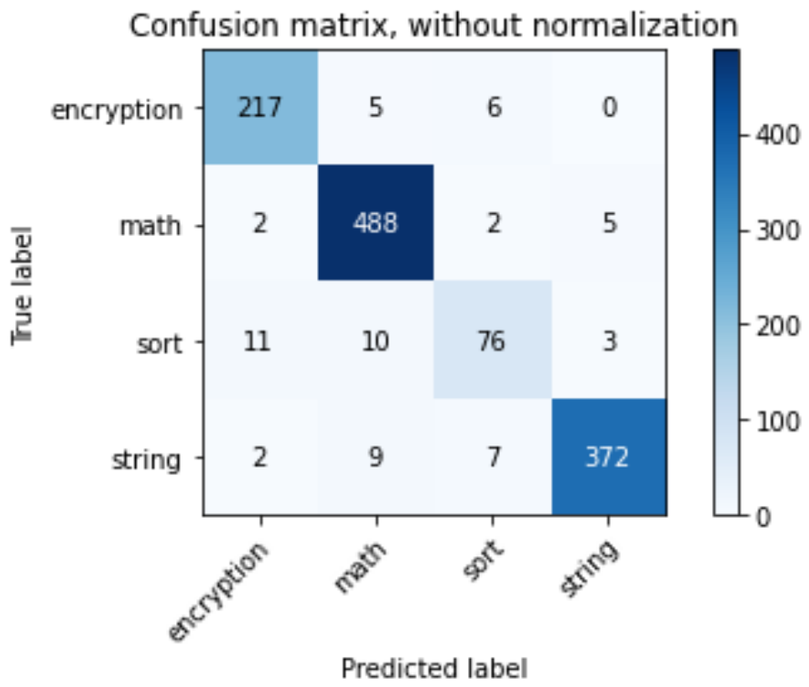

Figure 1: Confusion Matrix for SVM linear classifier

2. rbf

	precision	recall	f1-score	support
encryption	1.000	0.091	0.167	230
math	0.386	1.000	0.557	461
sort	0.000	0.000	0.000	119
string	0.000	0.000	0.000	405
accuracy			0.397	1215
macro avg	0.347	0.273	0.181	1215
weighted avg	0.336	0.397	0.243	1215

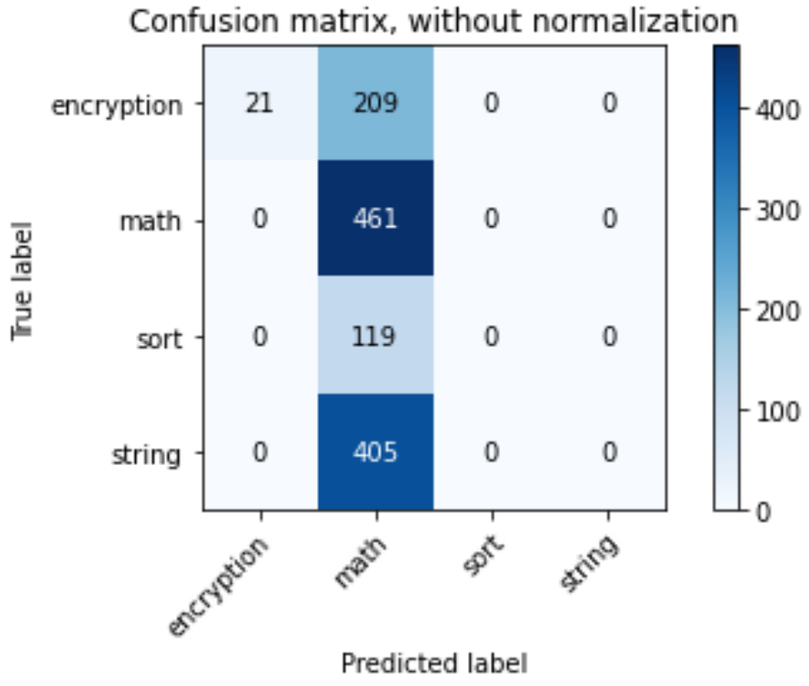

Figure 2: Confusion Matrix for SVM rbf classifier

3. poly

	precision	recall	f1-score	support

encryption	1.000	0.036	0.069	224
math	0.422	1.000	0.593	509
sort	0.000	0.000	0.000	98
string	0.000	0.000	0.000	384
accuracy			0.426	1215
macro avg	0.355	0.259	0.166	1215
weighted avg	0.361	0.426	0.261	1215

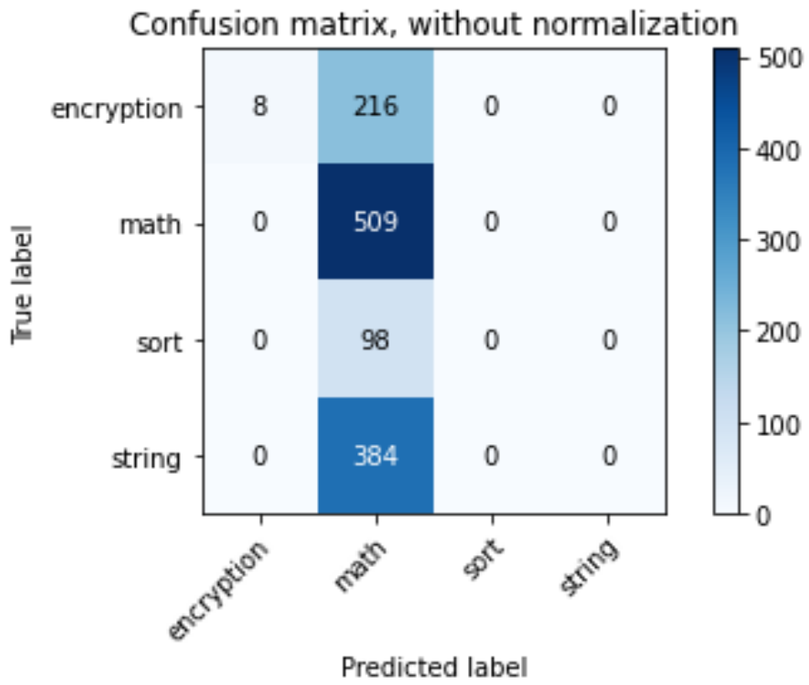

Figure 3: Confusion Matrix for SVM 3-degree polynomial classifier

As described, the results of the voting system for the SVM classifier are given in the following table.

	precision	recall	
encryption	2/3	1	
math	1	2/3	
sort	1	1	
string	1	1	
accuracy			1

Therefore, we can conclude that the Linear SVM gives the best results taking into account all criteria.

Cross validation score for SVM Classifier was not calculated since it would take huge amount of time.

For $\mathbf{C = 1}$:

SVM ran in 257.23009071800004 seconds.

For $\mathbf{C = 100}$**:**
Accuracy: 0.946

	precision	recall	f1-score	support
encryption	0.934	0.934	0.934	227
math	0.943	0.987	0.964	452
sort	0.896	0.735	0.808	117
string	0.969	0.969	0.969	419

SVM ran in 453.393614481 seconds.

For **C=0.01:**

Accuracy: 0.9555

	precision	recall	f1-score	support
encryption	0.962	0.935	0.949	217
math	0.958	0.973	0.965	516
sort	0.842	0.889	0.865	90
string	0.977	0.959	0.968	392

SVM ran in 105.56804294899999 seconds.

It is obvious that regularization parameter C influences the performance of the classifier. Higher value of C means that margins of hyperplanes dividing the space are larger. However, the calculation of the model with higher C requires more time while giving greater accuracy.

Decision tree classifier

1. criterion = entropy, splitter = best

	precision	recall	f1-score	support
encryption	0.969	0.956	0.963	229
math	0.975	0.996	0.986	479
sort	0.897	0.861	0.879	101
string	0.990	0.983	0.986	406
accuracy			0.973	1215
macro avg	0.958	0.949	0.953	1215
weighted avg	0.973	0.973	0.973	1215

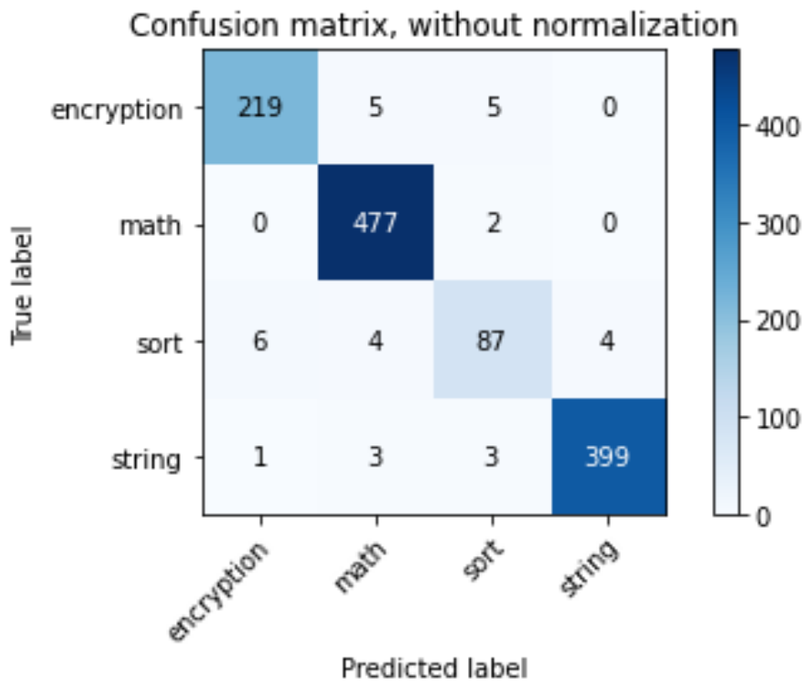

Figure 4: Confusion Matrix for DT classifier, with criterion = entropy, splitter = best

2. criterion = gini, splitter = random

	precision	recall	f1-score	support
encryption	0.956	0.960	0.958	225
math	0.984	0.980	0.982	488
sort	0.852	0.829	0.840	111
string	0.972	0.982	0.977	391
accuracy			0.963	1215
macro avg	0.941	0.938	0.939	1215
weighted avg	0.963	0.963	0.963	1215

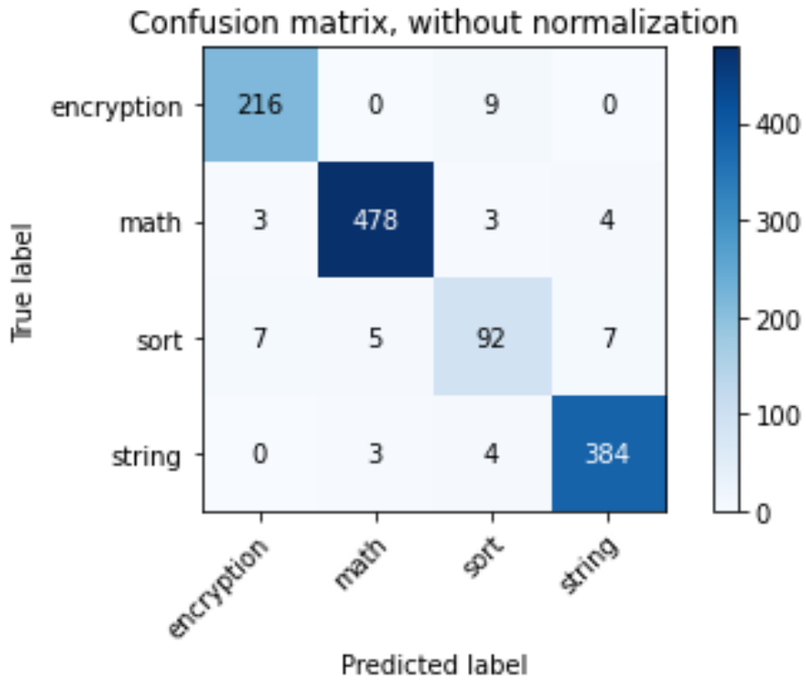

Figure 5: Confusion Matrix for DT classifier, with criterion = gini, splitter = random

3. criterion = entropy, splitter = random

	precision	recall	f1-score	support
encryption	0.970	0.978	0.974	228
math	0.989	0.994	0.991	467
sort	0.909	0.877	0.893	114
string	0.978	0.978	0.978	406
accuracy			0.974	1215
macro avg	0.961	0.957	0.959	1215
weighted avg	0.974	0.974	0.974	1215

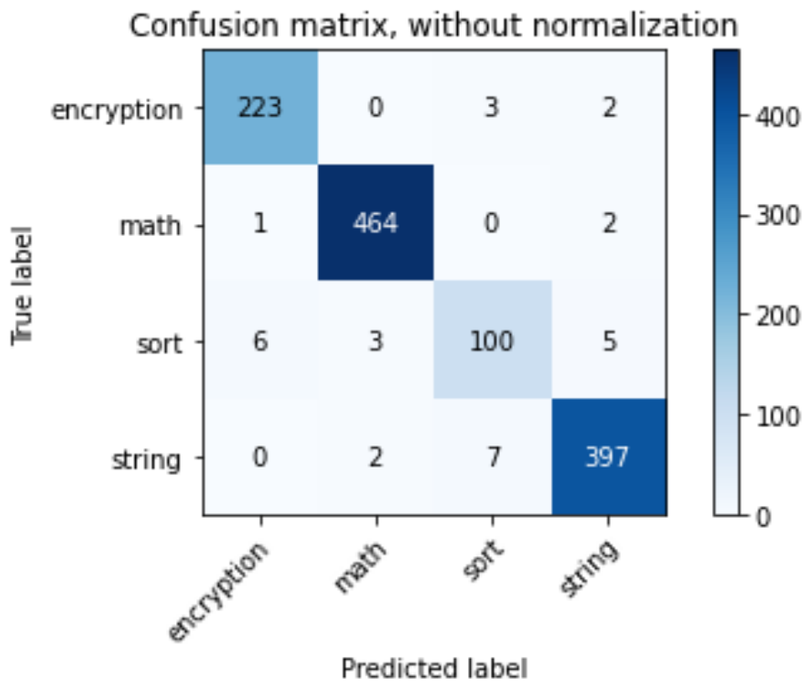

Figure 6: Confusion Matrix for DT classifier, with criterion = entropy, splitter = random

4. criterion = gini, splitter = best

	precision	recall	f1-score	support
encryption	0.923	0.906	0.914	224
math	0.983	0.983	0.983	460
sort	0.880	0.851	0.866	121
string	0.964	0.983	0.973	410
accuracy			0.956	1215
macro avg	0.937	0.931	0.934	1215
weighted avg	0.955	0.956	0.955	1215

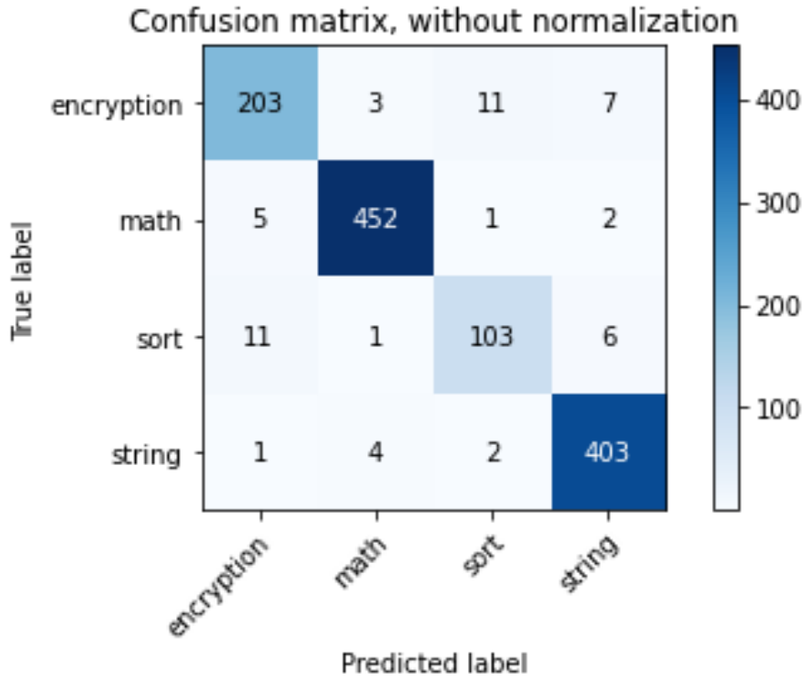

Figure 7: Confusion Matrix for DT classifier, with criterion = gini, splitter = best

As described, the results of the voting system for the Decision Tree Classifier are given in the following table.

	precision	recall	
encryption	3	3	
math	3	1	
sort	3	3	
string	1	1/4	
accuracy			3

Therefore, we can conclude that the Decision Tree with entropy as a criterion and random splitter gives the best results taking into account all criteria.

Based on cross validation for the Decision Tree Classifiers with highest number of votes, we get that the accuracy is 0.969 (+/- 0.01).

DT ran in 0.8378364010000041 seconds.

Blind test

Taking into account all performance metrics as well as time of execution the blind test was ran with Decision Tree Classifier, with parameters criterion = entropy, splitter = random. The results of blind test i.e. the output of predict function of scikit learn model is printed into a text file.

Conclusion

In this homework assignment was to determine the model that best fits the provided data. The data consists of rows, each containing an ID, a CFG, assembly code and semantic. The semantic column represents the class of the data, or the output of the data. Based on the analysis presented in this document, we concluded that the classifier that best fits the data is Decision Tree Classifier with parameters criterion = entropy, splitter = random. We also tested SVM classifier, but it turns out it's not as nearly as performant (time of execution) as Decision Tree Classifier. Moreover, Naive Bayes Classifier and Logistic Regression were also tested, but SVM and Decision Trees are chosen in the end, based on the performance metrics. The classifier that gave the best results was used to generate the output of blind test, provided in the assignment. All the code is written in Python3, using scikit learn and networkX libraries for graph computations.