Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

Machine Learning: homework 1

Bug finding classification

Student:
Federica Cocci

ID:
1802435

November 25, 2021

Contents

1 Problem to be resolved

This homework has been assigned during the seminar "Bug Finding in Compiler Toolchains".
A compiler toolchain is made of compiler and debugger and other modules; inside it, there can be bugs in different points of the chain. We have our program written in any language (for this homework I have worked with C instructions lines) which is passed to the compiler, this one transforms the input in a machine language code so in executable code.

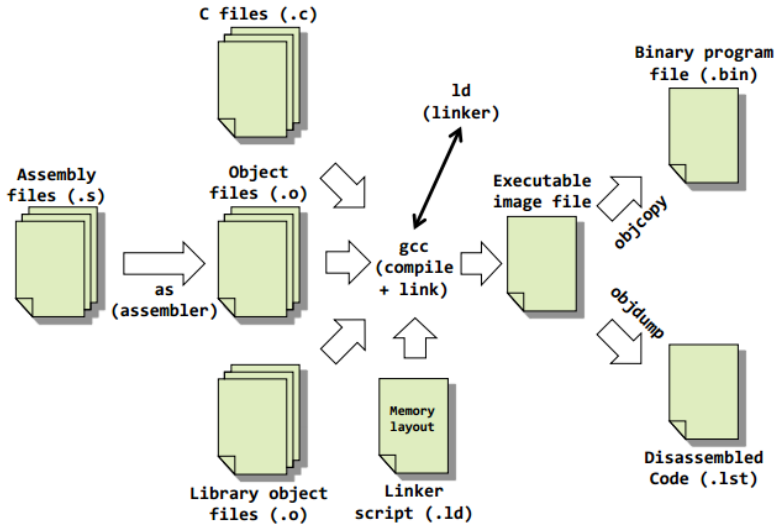


Figure 1: Compiler Toolchain

A way to find bugs is following a debug execution trace which means following a sequence of steps from the entry point to the program exit by repeatedly stepping over assembly instructions. Along the path we map assembly instruction to source instruction and look either for a correspondence or for a bug.

2 Dataset

The given dataset is a mapping between Assembly and C instructions.
The reported informations inside the csv file are: assembly instructions, source line, number of the line inside the program, function name where the line is from, name of the program and the presence or not of a bug.

	A	B	C	D	E	F
1	instructions	source_line	line_number	function_name	program	bug
2	movl HIGHVAL I 19	int32t I 19 1 = (-8	2517	func_47__0	/home/stepping/data	0
3	movl HIGHVAL I 82 (	int32t I 25 73 = (-1	1994	func_25__0	/home/stepping/data	1
4	movq I 15 07 5 %rcx	(* * I 15 06 1) = I	2171	func_1__0	/data_source_asm_t	0
5	movabsq HIGHVAL %	((g 14 5 = = & I 1	2096	func_1__0	/home/stepping/data	1
6	movw HIGHVAL I 41	int16t I 41 89 = (-5	1955	func_4__0	/home/stepping/data	0
7	movl MEM %eax mo	g 13 9 = (I 39 3 =	1950	func_19__0	/home/stepping/data	0
8	leaq I 94 70 %rdi mo	uint32t I 43 90 = H	2021	func_1__0	/data_source_asm_t	1
9	movabsq HIGHVAL %	int32t I 17 10 = (-2	1891	func_1__0	/home/stepping/data	1
10	leaq I 15 6 %rax mov	uint16t * * I 15 5 =	2709	func_56__0	/home/stepping/data	0

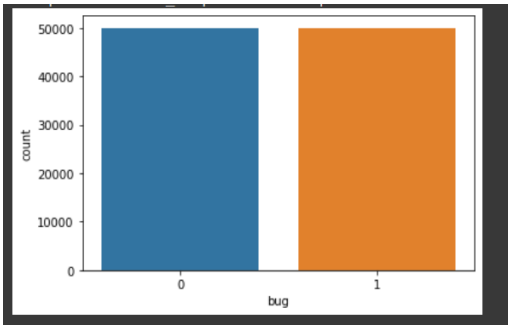
Figure 2: CSV file

The first thing I did when I read the file is selecting only the three columns which are important for the solution of the problem and these are: assembly instructions, source line and bug. The dataset is a balanced dataset so no problem of unbalanced data to take into account in the application of models.

	instructions	source_line	bug
0	movl HIGHVAL I 19 1	int32t I 19 1 = (-8) ;	0
1	movl HIGHVAL I 82 6	int32t I 25 73 = (-1) ;	1
2	movq I 15 07 5 %rcx movq I 15 06 1 %rax movq M...	(* * I 15 06 1) = I 15 07 5 ;	0
3	movabsq HIGHVAL %rax cmpq %rax MEM je MEM mova...	((g 14 5 = = & I 15 8) ? (void) (0...	1
4	movw HIGHVAL I 41 89	int16t I 41 89 = (-5) ;	0
...	...	...	...
99995	movsbl MEM %eax addq 48 %rsp popq %rbp retq	return g 88 ;	0
99996	movl HIGHVAL I 74 2	int32t I 74 2 = (-1) ;	0
99997	movw MEM %ax movb %al MEM	struct S 0 I 17 58 = { HIGHVAL , 6 , 14...	1
99998	leaq I 53 %rax movq %rax I 87	int32t * I 87 = & I 53 ;	0
99999	movb -92 I 15 43 9	uint8t I 15 43 9 = 164 ;	0

100000 rows × 3 columns

(a) Dataset columns



(b) The dataset is balanced

2.1 Preprocessing

Actually I have decided not to apply any kind of method to extrapolate particular features because I think all the components of the line are useful for the solution: bug can be related to the name of the variable, to the value it works with, to the memory it accedes,... Using only mnemonics I think it wouldn't help me.

Before splitting the dataset in train and test set, because it is composed of text data I have to encode it. To do this I have used CountVectorizer function of scikit learn. The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

3 Models

Which are the methods used to resolve this classification problem?
I have applied several methods to understand which is the best for this study case. I have used 3 methods of Naive Bayes type, then the decision tree (not pruned and then (partially) pruned) and at the end I have used the random forest.

3.1 Naive Bayes Classifiers

These methods are based on Bayes's probabilistic theory where the naive assumption is made. The Naive Bayes Assumption says that all the attributes are mutually independent.

Assuming a target function f:X→V
where each x is described by a set of attributes $a_1, a_2, ..., a_n$

NAIVE BAYES ASSUMPTION $\quad \mathrm{P}(\mathrm{a}_1, a_2, ..., a_n | v_j, D) = \prod_i P(a_i | v_j, D)$

NAIVE BAYES CLASSIFIER $\quad \mathrm{v}_{NB} = \underset{v_j \in V}{argmax} P(v_j|D) \prod_i P(a_i|v_j, D)$

Depending on the probability distribution used to compute $P(a_i|v_j, D)$
we get different results. Here I decided to use Bernoulli distribution, Multinomial distribution and Gaussian distribution.

3.1.1 Bernoulli Naive Bayes

Bernoulli distribution assumes that features are binary-valued.

3.1.2 Multinomial Naive Bayes

It works with occurrence counts (e.g. bag of words).

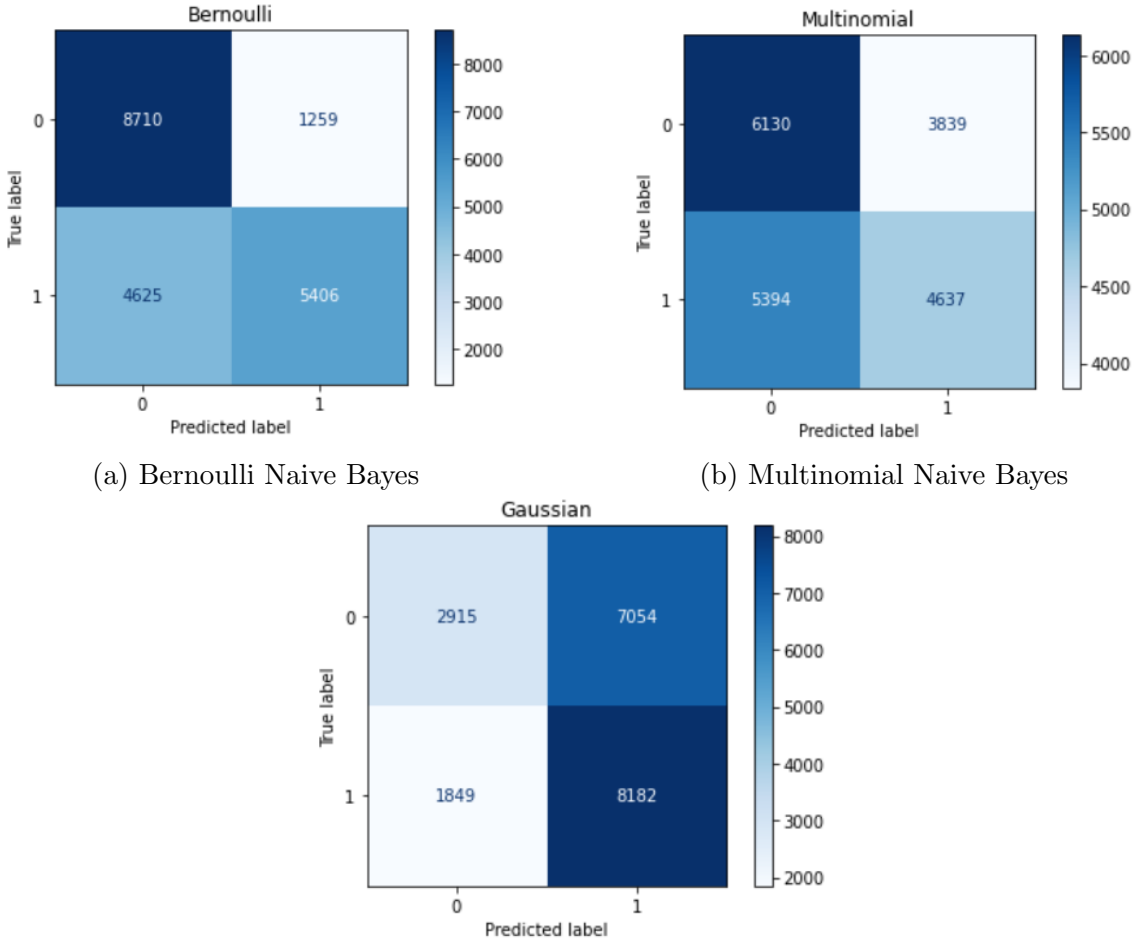
3.1.3 Gaussian Naive Bayes

It is based on continuous distribution.

3.1.4 Discussion about obtained results

Classifier	Accuracy on train set	Accuracy on test set
Bernoulli Naive Bayes	0.708538	0.7058
Multinomial Naive Bayes	0.554937	0.53835
Gaussian Naive Bayes	0.563438	0.55485

Figure 4: Accuracies



(a) Bernoulli Naive Bayes

(b) Multinomial Naive Bayes

(c) Gaussian Naive Bayes

Figure 5: Confusion matrices

The model which has the highest accuracy among the three is the Bernoulli Naive Bayes Classifier. Actually before doing the computation, I thought the best one would have been the Multinomial one because it works with the frequencies of words and CountVectorizer returns vectors with the frequency of each word inside the line. At the end I gather that inside the vectors there are more 0 and 1 than other integer numbers because maybe there are more single instructions as initialization of variable or in general assignments.

```
movl HIGHVAL | 19 1 int32t | 19 1 = ( -8 )
movl HIGHVAL | 82 6 int32t | 25 73 = ( -1 )
```

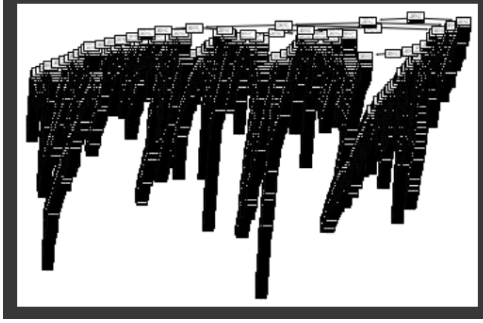
Figure 6: Assignment instructions

Gaussian Naive Bayes is the one with less accuracy (it should work with continuous values while here I work with discrete values)

3.2 Decision Tree

A Decision Tree is a method used for classification.
In my solution I started with a decision tree which doesn't have any limit in depth or in number of leaves and what I have obtain is a tree with:

```
1 if model_type == "decision tree":
2   tree.plot_tree(model)
```



(a) Decision Tree

```
Depth:  119
Number of leaves:  6601
Number of nodes:  13201
```

(b) Characteristics of decision tree

Then I have computed the accuracy on training set and test set and what I have got is:

```
Accuracy on train set 1.000
Accuracy on test set 0.918
```

Figure 8: Accuracy of not pruned tree

Looking at these values, also because the tree hasn't modes to stop its growth fitting to much on the training set and not generalizing well (it is sensible to any change in input data), I decided to apply post-pruning to decrease the accuracy on train set increasing the accuracy on test set.

3.2.1 Post-pruning

When the size of the tree increases then the accuracy on test data decreases respect to the accuracy on training data and, following the definition of overfitting, it means that increasing the number of nodes keeps the model to overfitting.
How can we reduce overfitting?

In general we should stop training for too much time and to do this thing, there is the algorithm of post-pruning. Post-pruning means letting the model builds the entire tree and then start pruning so keep cutting the tree until we start to have a decrease in accuracy, at this point it stops or cuts a different piece of the tree.
To decrease the overfitting there are different methods:

- Pruning: to prune there are different ways:
  - max_depth: how deep the tree will be. The bigger it is, the deeper the tree is increasing the possibility of overfitting.
  - min_samples_split: minimum number of samples required to split an internal node.
  - min_samples_leaf: minimum number of samples required to be in the leaf node. The bigger it is, the deeper the tree is increasing the possibility of overfitting.
  - pruning techniques (this is the adopted one in the solution)
- Using a random forest in the place of decision tree.

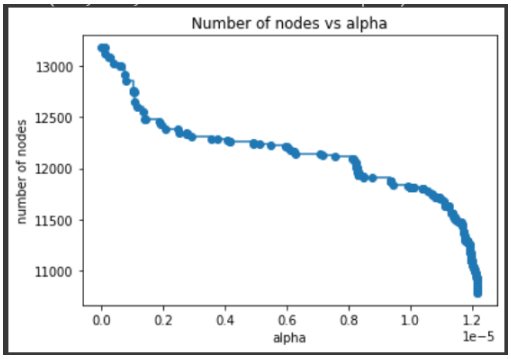
I created and fitted another decision tree but this time passing a value to the parameter ccp_alpha. This parameter is used for the algorithm of pruning implemented in scikit-learn which is called "Minimal Cost-Complexity Pruning" and in particular the subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen: greater values of ccp_alpha increase the number of nodes pruned.
Actually the appropriate value of cpp_alpha is found in a constructive way:

1. scikit-learn provides DecisionTreeClassifier.cost_complexity_pruning_path that returns the effective alphas and the corresponding total leaf impurities at each step of the pruning process.

2. I train a decision tree using the effective alphas.

3. Because there were found 2244 ccp_alphas I decided to analyse only the first 200 finding among them the one which is the best looking how the train accuracy changed with respect to test accuracy. Also the total number of nodes of course changes.

(a) Training accuracy VS test accuracy



(b) Decreasing of number of nodes

I noticed that the train accuracy decreased while test accuracy increased so I take the last value of these 200 analyzed ccp_alpha

4. I created and fitted another decision tree with this selected ccp_alpha

With this approach I got a tree with these characteristics:

```
Depth:  93
Number of leaves:  5361
Number of nodes:  10721
```

Figure 10: Characteristics of decision tree

And the accuracy in this case is:

```
Accuracy on train set 0.995
Accuracy on test set 0.919
```

Figure 11: Accuracy of pruned tree

N.B. If I had analyzed all of 2244 alphas then for sure I would have obtain better accuracies but I had have to make this choice about analyzing only the first 200 for computation time problems.

3.2.2 Discussion about obtained results

To discuss about these results, I will compare structure of the two trees, accuracies and confusion matrices.

```
Depth:  119
Number of leaves:  6601
Number of nodes:  13201
```

(a) Characteristics of not pruned tree

```
Depth:  93
Number of leaves:  5361
Number of nodes:  10721
```

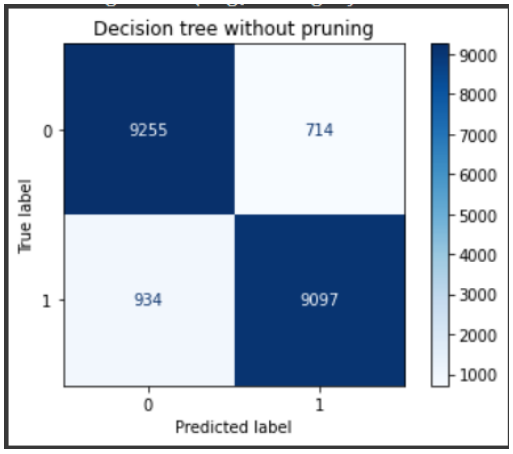
(b) Characteristics of pruned tree

```
Accuracy on train set 1.000
Accuracy on test set 0.918
```

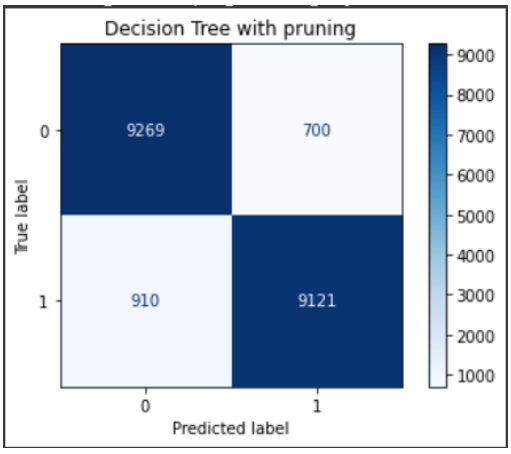
(a) Accuracy of not pruned tree

```
Accuracy on train set 0.995
Accuracy on test set 0.919
```

(b) Accuracy of pruned tree



(a) Not pruned tree - confusion matrix



(b) Pruned tree - confusion matrix

As it is shown, I obtain a better result by pruning the decision tree also if it can be improved further by continuing the analyses on the other values of ccp_alpha. As it was expected, the pruned tree less overfits (the accuracy on the train set decreases and the one on the test set increases).
I obtained a better accuracy working on a smaller tree with less nodes and depth. However this small variation on accuracies actually doesn't have a big consequence on the other metrics which are precision, recall and f1-score:

	precision	recall	f1-score	support
0	0.91	0.93	0.92	9969
1	0.93	0.91	0.92	10031
accuracy			0.92	20000
macro avg	0.92	0.92	0.92	20000
weighted avg	0.92	0.92	0.92	20000

(a) Not pruned tree - other metrics

	precision	recall	f1-score	support
0	0.91	0.93	0.92	9969
1	0.93	0.91	0.92	10031
accuracy			0.92	20000
macro avg	0.92	0.92	0.92	20000
weighted avg	0.92	0.92	0.92	20000

(b) Pruned tree - other metrics

3.3 Random Forest

As already said, an alternative to pruning is using the random forest.
A random forest builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
Also to the random forest that I have created and fitted, I passed the value of ccp_alpha to prune the trees. This value is the same of the previous pruned decision tree.

3.3.1 Discussion about obtained results

```
Accuracy on train set 0.993
Accuracy on test set 0.954
```

Figure 16: Accuracy

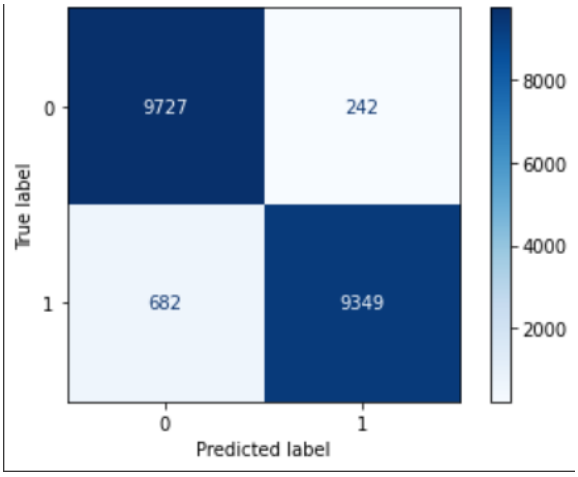


Figure 17: Confusion Matrix

	precision	recall	f1-score	support
0	0.94	0.98	0.96	9969
1	0.98	0.93	0.95	10031
accuracy			0.95	20000
macro avg	0.96	0.95	0.95	20000
weighted avg	0.96	0.95	0.95	20000

Figure 18: Other metrics

4 Final observations and comparisons

Among all these solutions, the best one is the random forest because it has the highest accuracy on the test set and doesn't overfit as much as the pruned tree.

Naive Bayes Classifiers have a low computation cost also in term of time (respect to decision trees or forest) but they have a lower accuracy.

About the other metrics so precision, recall and f1-score, the random forest is the best with respect to decision tree so it is very good not to label as positive a sample that is negative (high precision) and it has also a good ability to avoid false negatives(high recall). Based on the performance metrics above, I will choose overall accuracy. Since our data is balanced, meaning a split between 50/50 true and negative samples, I can choose accuracy.

4.1 The final model used for blind test

In light of this, I decided to use the random forest as the model which works with the blind dataset.