SAPIENZA
UNIVERSITÀ DI ROMA

MACHINE LEARNING FOR MALWARE ANALYSIS

Computer Science Engineering

Università degli Studi di Roma "La Sapienza"

Andrea Lombardo

Matricola 1893440

November 4, 2019

Contents

1 Introduction

With the improvement of the technology during last years, objects like smartphone has entered people' lives as an essential element of daily survival. This fact could be dangerous for people' security because this object could be victim of numerous attacks against its user through malware. A malware is a software that has the capability to steal, encrypt, or delete your data and spy on your computer activity without your knowledge or permission. A malware is a malicious software that fulfills the deliberately harmful intent of an attacker.
Malware analysis concerns the study of malicious samples with the aim of developing a deeper understanding about several aspects of malware. There're two kind of malware analysis: **dynamic analysis**, is when the malware is executed in a controlled environment and its action on the system are registered and analyzed and **static analysis**, where a malicious file is analyzed and using a disassembler the analyst is able to understand what's going on. We're focus on the statistic analysis.

1.1 Relationship between malware detection and Machine Learning

A large body of research has studied the impact of Machine Learning in these kind of security problem. In malware analysis exists different problems, two of the most famous are:

Malware detection works on detecting if an application could be classifier as a malware application or benign(non-malware) application.

Subsequently, **Malware family classification** works on detecting which class the family belongs to.The class changes accordingly to the behaviour of the malware.

The problem for a machine is to have the capability to discriminate a malware software from a benign one. This topic is a matter of great importance,although we now are focused on analyzing the behaviour insight of the malware. In fact it is well known that compiling the same source code of a single function with different compilers lead to totally different compiled functions. Due to this fact one prototypical problem for reverse engineer is to detect which compiler was used to produce a given function and understanding also the optimizer used for that instruction. The machine Learning is a natural choice to support such a process of knowledge extraction.

2 Dataset

The provided dataset is composed by a training set and a test set. These two are jsonl files so each row is a json file.

- **The training set** is provided in jsonl format. It's composed by 30000 json object where each of them is in the following format: **'instruction'**, a collection of assembly instructions with the register used, an optimizer,with a tag **'opt'**, that could be **High** or **Low** and a compiler, with the tag **compiler**, which could be one of the following: **gcc, icc,** and **clang.** Each row of the file is a json object, we can see an example :


```
{
    "instructions": ["xor edx edx", "cmp rdi rsi", "mov eax 0xffffffff", "seta
                     dl", "cmovae eax edx", "ret"],
    "opt": "H",
    "compiler": "gcc "
}
```


- **The test set** is called blind set.It's a jsonl file which contains only the collection of instruction due to fact that the goal is guessing which is the right optimizer, for a binary problem and right compiler, for a multi-class problem according the classifier used.

3 Developing

First of all I decide to manage the collection of data from jsonl file to a csv file, according the jsonl library. After reading the content of the csv.file, **train-dataset.csv**, I manipulate the data with the purpose of discarding registers and storing mnemonic instructions of the same object in the same data structure, a list. The choice for merging all these assembly instructions into a list split with a blank, is to create a model of bag of word. Now for representing the instruction as a vector I decide first to put all the instruction into a string composed by assembly instruction and it's done through a function **ManageContentToVector** that giving in input a collection of strings(the list), and a number, return the collection vectors. The cited function, according the number in input, develops a different functions of **sklearn library** between HashingVectorizer,TfidfVectorizer and CountVectorize. For the phase of testing I decide to use only the CountVectorize, because I think that in this case could be more relevant to take care on the count of a word instead of giving an important as Tf-IDF model made, to a word that in our case words are assembly instructions. The approach of the vector used let us to composed each entry with the number of occurrences appears in that instruction.

The aim of this work is to compare the behaviour of different learning algorithm(Naive Bayes,Decision Tree) on this dataset with respect to different classification problem(Binary for the optimizer, Multiclass for the compiler) and decide which is the best solution for each classifier problem. About this fact we decide which is the best according performance in measurements of Precision,Recall, F1-score and AUC. During the experiments I decided to split the multi-class problem with two subproblems. In detail I want to compare the behaviour of the classifiers, during the analysis of multi-class problem, where they may be OneVsRestClassifier or OnevsOne. It means that change the way to consider classes.

4 LEARNING ALGORITHM: A NAÏVE BAYES APPROACH

The Naive Bayes Classifier is a probabilistic classifier based on the Bayes theorem with strong Independence assumption between the features that in our case are the word of the instruction. Bayes theorem leads to find a posterior probability of an event given a probability of an event that has already occurred. The criterion on which the algorithm described below has been developed, follows the same logic of the text classification algorithm with the string of instruction and the target that could be different according the problem. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n :

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Considering the independence probability we've the following formula:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

The classification of a new instance can be done by choosing the argmax of this distribution of probability, and because the denominator doesn't depend on class.

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

In our problem, due to the fact that each collection of instructions is represented as vector,this algorithm will compute the probability of having an instance of a specific compiler or optimization given the occurrences of instructions in the string. We implement the Naive Bayes classifier by scratch with 3 different distribution of probability: Multinomial and Bernoulli and Gaussian and decided to use it to detect the compiler and optimizer.

GaussianNB() implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian. It follow the function below:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

In our case we decide to use it in both cases(binary, multi-class) problems.

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification. In our case we decide to use it only in the approach of the Multi-class problem to detect right compiler .

BernulliNB implements the Naive Bayes training and classification algorithms for data distributed according to Bernoulli distributions. There may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors. For this reasons I decide to use it only into a binary classifier problem.

5 DecisionTree

A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label.
It classifies instances by sorting them from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree moving down the tree branch corresponding to the value of the attribute. This process is then repeated for the subtree rooted at the new node.

6 Metrics

In Machine Learning, performance measurement is an essential task. Indeed when it comes to a classification problem we should measure in terms of:
Precision= $\frac{tp}{tp+fp}$
Recall= $\frac{tp}{tp+fn}$
F1-score=$\frac{2PR}{P+R}$
Accuracy=$\frac{tp+tn}{tp+tn+fp+fn}$
Where:

True Positive (tp) is when correctly classified as positive
True Negative (tn) is when correctly classified as negative
False Positive (fp) is when falsely classified as positive
False Negative (fn) is when falsely classified as negative

Another concept sometimes used in evaluation is the Receiver Operating Characteristics (ROC) curve. The ROC curve plots the true positive rate or sensitivity against the false positive rate. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. The AUC is a number between 0 and 1. ROC curve is used for visual comparison of classification models which shows the trade-off between the true positive rate and the false positive rate.

7 Experiments

About the experiment I decide to test each classifier that I defined before for each problem (binary,multi-class(OneVSRest),multi-class(One-vs-One) with 3 different type of test where I change the optional flag(**random-state**, **test-size**). The experiment allows to understand which is the best solution, according metrics, for computing a classification on the blind-set. The experiments are made only on the provided dataset and for test it I decide to split data according different partitions that changes for the experiments did. Into the the routine **train-test-split()** I give in

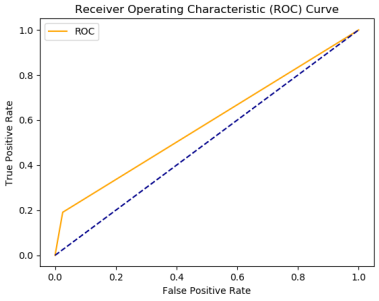
input a float number, test-size, which could be between 0.0 and 1.0 and represents the proportion of the dataset to include in the test split and the random-state which generate a different combination of data.
I should mention that I want to make a comparison on the behaviour of **same** distribution in the same conditions so I used MultinomialNB for multi-class problem and BernoulliNB for binary problem cause the first works with occurrence counts instead the second is designed for binary/boolean features. Remember that the two targets are different one refers to the optimizer (binary problem) and the second refers to the compiler(multi-class problem). I decide to create a plot related on confusion matrix using plot_confusion_matrix() routine only on DecisionTree approach because how we can see after it works better than other configurations.

7.1 Experiment 1

In first experiment I choose as parameters:(random-state=**10** test-size=**0.30**) and I want to analyze performance on different distributions:

Binary problem:

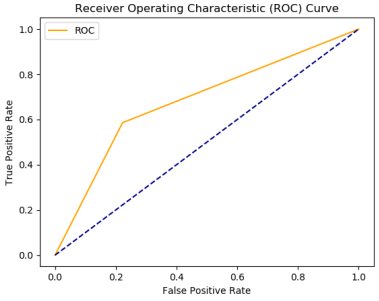
GaussianNB: how can we see in the pictures behind the GaussianNB distribution is not good for this kind of problem we can see in terms of: AUC is **0.58**,it means that the probability of missing the classification is near 50% and it's not good. Another important element to notice is the accuracy that in this case is not so much high **0.66**.



	precision	recall	f1-score	support
L	0.64	0.97	0.77	5355
H	0.84	0.19	0.31	3645
accuracy			0.66	9000
macro avg	0.74	0.58	0.54	9000
weighted avg	0.72	0.66	0.59	9000

```
-Accuracy: 0.6573333333333333
-Precision element y: [0.6390011  0.83754513]
-Recall: [0.97478992 0.1909465 ]
show information in roc with the information:
-AUC: 0.58
```

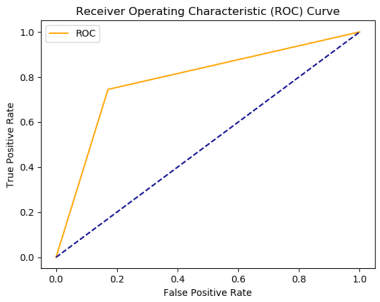
BernulliNB how we can see in the picture behind it behaves better than GaussianNB distribution, in fact we prefer using BernulliNB distribution for this dataset. Mentions terms of : **accuracy:0.7** and **AUC=0.68**



	precision	recall	f1-score	support
L	0.73	0.78	0.76	5355
H	0.64	0.59	0.61	3645
accuracy			0.70	9000
macro avg	0.69	0.68	0.68	9000
weighted avg	0.70	0.70	0.70	9000

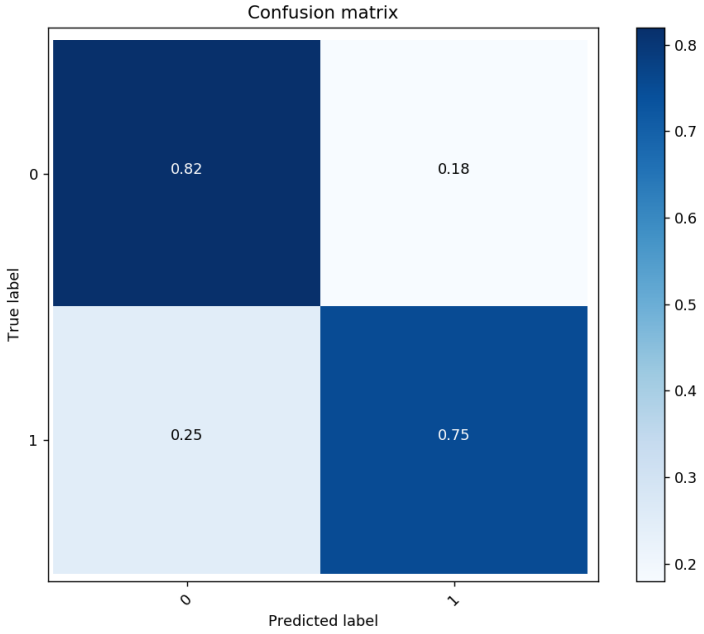
```
-Accuracy: 0.7
-Precision element y: [0.73400317 0.64201984]
-Recall: [0.77759104 0.58600823]
show information in roc with the information:
-AUC: 0.68
```

DecisionTreeClassifier how we can see in the pictures behind the DecisionTreeClassifier allows to have better result for AUC accurancy performance.In fact AUC is almost higher than the probability to have a success is more than 75%. Accuracy is near **0.80**



	precision	recall	f1-score	support
L	0.83	0.83	0.83	5355
H	0.75	0.74	0.75	3645
accuracy			0.79	9000
macro avg	0.79	0.79	0.79	9000
weighted avg	0.79	0.79	0.79	9000

-Accuracy: 0.7948888888888889
-Precision element y: [0.82678339 0.7477279]
-Recall: [0.82894491 0.74485597]
show information in roc with the information:
-AUC: 0.79



Multi-class problem(One-VS-Rest) in comparison(One-VS-One)

NOTE: I decide to make a comparison between same distribution with the two approach: at **left** there's **One-Vs-Rest** version, at **right** there's **One-VS-One**. For this fact I decide to compute the comparison instead analyze AUC computation for each computation.

GaussianNB: how can we see in the pictures behind the GaussianNB distribution is not good for this kind of problem we can see in terms of: accuracy that is in the order of **0.43/0.44**, this is the only common measure for the two tables. If we focus on the recall and the precision rows the information changes a lot.

	precision	recall	f1-score	support
gcc	0.86	0.12	0.21	2997
clang	0.52	0.25	0.34	2966
icc	0.40	0.94	0.56	3037
accuracy			0.44	9000
macro avg	0.59	0.44	0.37	9000
weighted avg	0.59	0.44	0.37	9000

-accuracy: 0.44233333333333336
-recall: [0.12078745 0.25320297 0.94435298]
-precision: [0.86190476 0.52080444 0.40179322]

	precision	recall	f1-score	support
gcc	0.38	0.98	0.54	2997
clang	0.77	0.11	0.19	2966
icc	0.77	0.21	0.33	3037
accuracy			0.43	9000
macro avg	0.64	0.43	0.36	9000
weighted avg	0.64	0.43	0.36	9000

-accuracy: 0.432
-recall: [0.97597598 0.10687795 0.21270991]
-precision: [0.37751678 0.76570048 0.77088305]

MultinomialNB how we can see in the picture behind it behave better than GaussianNB distribution, in fact not only the accuracy is higher but the two approaches OneVsRest and OnevsOne don't have an higher range of differences in terms of numbers. For exam-

ple in the OneVsRest we 've 0.59 for the accuracy that is same of OneVsOne(performing an truncation/approximation). The same topic is valid for the recall and precision lines.

	precision	recall	f1-score	support
gcc	0.56	0.69	0.62	2997
clang	0.57	0.51	0.54	2966
icc	0.68	0.60	0.64	3037
accuracy			0.60	9000
macro avg	0.60	0.60	0.60	9000
weighted avg	0.60	0.60	0.60	9000

-accuracy: 0.5994444444444444
-recall: [0.68601935 0.51416049 0.59729997]
-precision: [0.56297919 0.56839359 0.68067542]

	precision	recall	f1-score	support
gcc	0.55	0.72	0.63	2997
clang	0.58	0.52	0.54	2966
icc	0.69	0.55	0.62	3037
accuracy			0.60	9000
macro avg	0.61	0.60	0.60	9000
weighted avg	0.61	0.60	0.60	9000

-accuracy: 0.5972222222222222
-recall: [0.72038705 0.51652057 0.55449457]
-precision: [0.55344783 0.57550714 0.69101354]

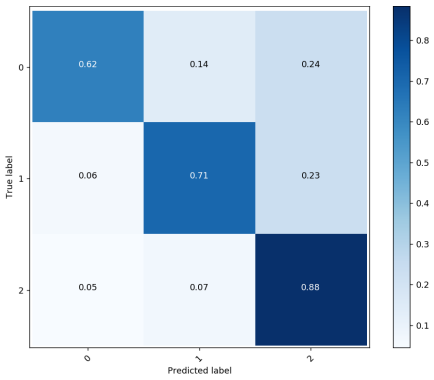
DecisionTreeClassifier how we can see in the pictures behind the DecisionTreeClassifier allows to have better result respect to the others in fact the accuracy is higher and the entry of the precision and recall are a number nearest to 1 than other computations made above.

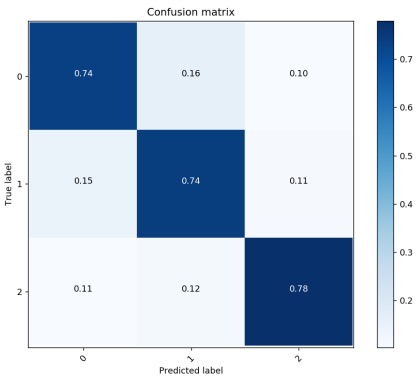
	precision	recall	f1-score	support
gcc	0.86	0.63	0.72	2997
clang	0.77	0.71	0.74	2966
icc	0.66	0.88	0.75	3037
accuracy			0.74	9000
macro avg	0.76	0.74	0.74	9000
weighted avg	0.76	0.74	0.74	9000

-accuracy: 0.7413333333333333
-recall: [0.62696029 0.71105866 0.88376688]
-precision: [0.85916781 0.76970803 0.65897373]

	precision	recall	f1-score	support
gcc	0.74	0.74	0.74	2997
clang	0.72	0.74	0.73	2966
icc	0.79	0.77	0.78	3037
accuracy			0.75	9000
macro avg	0.75	0.75	0.75	9000
weighted avg	0.75	0.75	0.75	9000

-accuracy: 0.7506666666666667
-recall: [0.74007341 0.73836817 0.77313138]
-precision: [0.74007341 0.72444591 0.78791946]



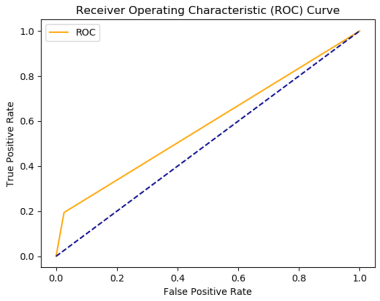


7.2 Experiment 2

In the second experiment I decide to change only parameters random-state=**50** test-size=**0.80** and see what are changes.

Binary problem:

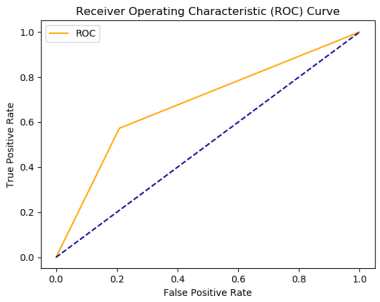
GaussianNB: how can we see in the pictures behind, the GaussianNB distribution is not good for this kind of problem how I have said before, we can notice that the performance doesn't change a lot from previous experiment.



	precision	recall	f1-score	support
L	0.64	0.97	0.77	14338
H	0.83	0.19	0.32	9662
accuracy			0.66	24000
macro avg	0.74	0.58	0.54	24000
weighted avg	0.72	0.66	0.59	24000

-Accuracy: 0.6599583333333333
-Precision element y: [0.64203265 0.83281596]
-Recall: [0.97370624 0.1943697]
show information in roc with the information:
-AUC: 0.58

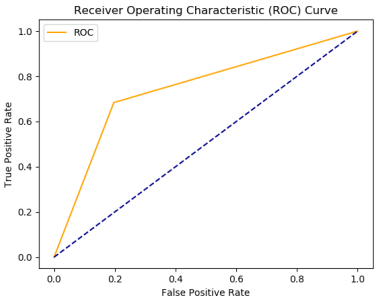
BernulliNB how we can see in the picture behind it behave better than GaussianNB distribution, in fact we prefer BernulliNB distribution in contrast to GaussianNB. How we have said before, we can notice that the performance doesn't change a lot from previous experiment.



	precision	recall	f1-score	support
L	0.73	0.79	0.76	14338
H	0.65	0.57	0.61	9662
accuracy			0.70	24000
macro avg	0.69	0.68	0.68	24000
weighted avg	0.70	0.70	0.70	24000

-Accuracy: 0.703375
-Precision element y: [0.7331869 0.64921958]
-Recall: [0.79153299 0.57255227]
show information in roc with the information:
-AUC: 0.68

DecisionTreeClassifier how we can see in the pictures behind the DecisionTreeClassifier allow to have better result for AUC accurancy performance. We can notice that performance of this test are worse than previous. We can see it also from the ROC curve(orange lines).



	precision	recall	f1-score	support
L	0.79	0.80	0.80	14338
H	0.70	0.68	0.69	9662
accuracy			0.76	24000
macro avg	0.75	0.74	0.74	24000
weighted avg	0.75	0.76	0.75	24000

-Accuracy: 0.7550833333333333
-Precision element y: [0.79028273 0.70067883]
-Recall: [0.80318036 0.68370938]
show information in roc with the information:
-AUC: 0.74

Multi-class problem(One-VS-Rest) in comparison(One-VS-One)

NOTE: I decide to make a comparison between same distribution with the two approach: at **left** there's **One-Vs-Rest** version, at **right** there's **One-VS-One**.

GaussianNB: how can we see in the pictures behind this distribution is not good notice the parameters of accuracy, recall which are too low.

	precision	recall	f1-score	support
gcc	0.53	0.26	0.35	7987
clang	0.43	0.92	0.58	8000
icc	0.84	0.29	0.43	8013
accuracy			0.49	24000
macro avg	0.60	0.49	0.46	24000
weighted avg	0.60	0.49	0.46	24000

-accuracy: 0.492875
-recall: [0.26205083 0.924125 0.29239985]
-precision: [0.53041054 0.42793471 0.84341253]

	precision	recall	f1-score	support
gcc	0.37	0.97	0.53	7987
clang	0.69	0.12	0.21	8000
icc	0.85	0.17	0.28	8013
accuracy			0.42	24000
macro avg	0.64	0.42	0.34	24000
weighted avg	0.64	0.42	0.34	24000

-accuracy: 0.419125
-recall: [0.97032678 0.1205 0.16785224]
-precision: [0.36859127 0.69402448 0.84858044]

MultinomialNB how we can see in the picture behind it behave better than GaussianNB distribution, in terms of accuracy. Also like the previous test the two approaches oneVs-Rest and OnevsOne don't have an higher range of differences in terms of numbers.

	precision	recall	f1-score	support
gcc	0.53	0.67	0.59	7987
clang	0.55	0.52	0.53	8000
icc	0.67	0.54	0.60	8013
accuracy			0.58	24000
macro avg	0.58	0.58	0.58	24000
weighted avg	0.58	0.58	0.58	24000

-accuracy: 0.5754166666666667
-recall: [0.66620759 0.516 0.54424061]
-precision: [0.53252602 0.55150301 0.66855741]

	precision	recall	f1-score	support
gcc	0.52	0.69	0.59	7987
clang	0.55	0.51	0.53	8000
icc	0.67	0.50	0.57	8013
accuracy			0.57	24000
macro avg	0.58	0.57	0.57	24000
weighted avg	0.58	0.57	0.57	24000

-accuracy: 0.5676666666666667
-recall: [0.69224991 0.512 0.49906402]
-precision: [0.51944758 0.55269194 0.67266611]

DecisionTreeClassifier how we can see in the pictures behind this behaviour's distribution demonstrate that this approach for this dataset is better than others. Both the tests prove the best capability of classification of OnevsOne in terms of performance than OnevsRest.

	precision	recall	f1-score	support
gcc	0.80	0.56	0.66	7987
clang	0.74	0.64	0.69	8000
icc	0.60	0.86	0.71	8013
accuracy			0.69	24000
macro avg	0.71	0.69	0.68	24000
weighted avg	0.71	0.69	0.68	24000

-accuracy: 0.68575
-recall: [0.55853262 0.641375 0.85685761]
-precision: [0.79618062 0.73976355 0.59907512]

	precision	recall	f1-score	support
gcc	0.69	0.70	0.70	7987
clang	0.67	0.68	0.68	8000
icc	0.74	0.72	0.73	8013
accuracy			0.70	24000
macro avg	0.70	0.70	0.70	24000
weighted avg	0.70	0.70	0.70	24000

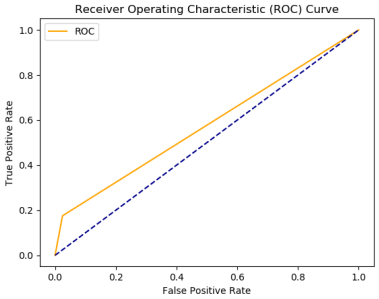
-accuracy: 0.7002083333333333
-recall: [0.70076374 0.67975 0.72007987]
-precision: [0.68954047 0.67318643 0.7392697]

7.3 Experiment 3

In the third experiment I decide to change only parameters random-state=**0** test-size=**0.55**.

Binary problem:

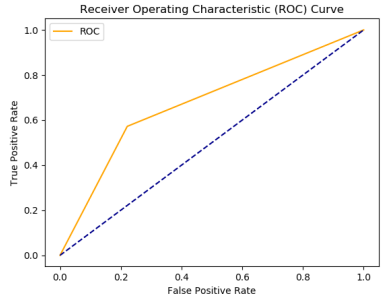
GaussianNB: how can we say before and how is showed in the pictures behind: the performance in all showed cases prove that this distribution works worse for this kind of problem and there are not more alterations from these experiments.



	precision	recall	f1-score	support
L	0.64	0.98	0.77	9871
H	0.83	0.18	0.29	6629
accuracy			0.65	16500
macro avg	0.73	0.58	0.53	16500
weighted avg	0.71	0.65	0.58	16500

-Accuracy: 0.654
-Precision element y: [0.63786935 0.82716927]
-Recall: [0.97538243 0.17544124]
show information in roc with the information:
-AUC: 0.58

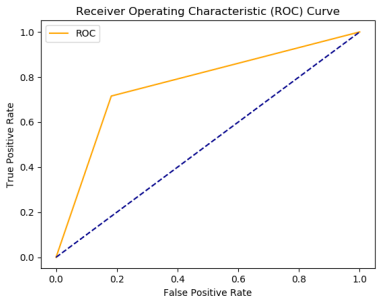
BernulliNB how can we say before and how is showed in the pictures behind: the performance in all showed cases prove that this distribution works better than GaussianNB but the quality of the classification is not enough. The three analyzed cases are not so much different in terms of precision,recall,accuracy and AUC.



	precision	recall	f1-score	support
L	0.73	0.78	0.75	9871
H	0.63	0.57	0.60	6629
accuracy			0.70	16500
macro avg	0.68	0.68	0.68	16500
weighted avg	0.69	0.70	0.69	16500

-Accuracy: 0.6956969696969697
-Precision element y: [0.73046949 0.63449314]
-Recall: [0.77864451 0.57218283]
show information in roc with the information:
-AUC: 0.68

DecisionTreeClassifier how we can say before this approach works better than others. For the three cases we're in the order of accuracy higher than **0.73**, and in this particular case without random AUC as Accuracy is **0.77**.



L	0.81	0.82	0.81	9871
H	0.73	0.72	0.72	6629
accuracy			0.78	16500
macro avg	0.77	0.77	0.77	16500
weighted avg	0.78	0.78	0.78	16500

-Accuracy: 0.7768484848484849
-Precision element y: [0.81066158 0.72534027]
-Recall: [0.81805288 0.71549253]
show information in roc with the information:
-AUC: 0.77

Multi-class problem(One-VS-Rest) in comparison(One-VS-One)

NOTE: I decide to make a comparison between same distribution with the two approach: at **left** there's **One-Vs-Rest** version, at **right** there's **One-VS-One**.

GaussianNB: this is a particular case where OneVsRest works better than OneVsOne in terms of accuracy. Indeed there is a difference far away from the first to the second.

	precision	recall	f1-score	support
gcc	0.86	0.12	0.21	5507
clang	0.44	0.93	0.59	5499
icc	0.71	0.53	0.60	5494
accuracy			0.52	16500
macro avg	0.67	0.52	0.47	16500
weighted avg	0.67	0.52	0.47	16500

-accuracy: 0.5237575757575758
-recall: [0.11875794 0.9276232 0.52548234]
-precision: [0.85826772 0.43725356 0.70898821]

	precision	recall	f1-score	support
gcc	0.38	0.97	0.54	5507
clang	0.66	0.15	0.24	5499
icc	0.90	0.17	0.29	5494
accuracy			0.43	16500
macro avg	0.65	0.43	0.36	16500
weighted avg	0.65	0.43	0.36	16500

-accuracy: 0.43212121212121213
-recall: [0.97457781 0.14893617 0.17182381]
-precision: [0.37763862 0.6636953 0.89563567]

MultinomialNB this is a particular case where OneVsRest works better than OneVsOne in terms of accuracy. Therefore we should mention about the performance parameters from one case to the next that are different.

	precision	recall	f1-score	support
gcc	0.82	0.60	0.69	5507
clang	0.76	0.66	0.70	5499
icc	0.63	0.88	0.73	5494
accuracy			0.71	16500
macro avg	0.74	0.71	0.71	16500
weighted avg	0.74	0.71	0.71	16500

-accuracy: 0.7132727272727273
-recall: [0.5997821 0.6575741 0.88278122]
-precision: [0.82143745 0.75854835 0.62889004]

	precision	recall	f1-score	support
gcc	0.55	0.73	0.63	5507
clang	0.60	0.52	0.56	5499
icc	0.68	0.56	0.62	5494
accuracy			0.60	16500
macro avg	0.61	0.60	0.60	16500
weighted avg	0.61	0.60	0.60	16500

-accuracy: 0.6015151515151516
-recall: [0.72525876 0.51718494 0.56188569]
-precision: [0.54991051 0.6042065 0.68145695]

DecisionTreeClassifier in this case the two classifiers work about with same performance but as always better than other approach for multi-class performance.

	precision	recall	f1-score	support
gcc	0.82	0.60	0.69	5507
clang	0.76	0.66	0.70	5499
icc	0.63	0.88	0.73	5494
accuracy			0.71	16500
macro avg	0.74	0.71	0.71	16500
weighted avg	0.74	0.71	0.71	16500

-accuracy: 0.7132727272727273
-recall: [0.5997821 0.6575741 0.88278122]
-precision: [0.82143745 0.75854835 0.62889004]

	precision	recall	f1-score	support
gcc	0.73	0.73	0.73	5507
clang	0.72	0.73	0.73	5499
icc	0.76	0.76	0.76	5494
accuracy			0.74	16500
macro avg	0.74	0.74	0.74	16500
weighted avg	0.74	0.74	0.74	16500

-accuracy: 0.7387272727272727
-recall: [0.72562194 0.72631388 0.76428831]
-precision: [0.73227048 0.72420671 0.75958755]

8 Conclusion

Therefore based on the above examples,we can see that the four distributions(GaussianNB, BernoulliNB, MultinomiaNB, DecisionTree) try make a linear classification without big differences from alternatives random-state parameters or test-set.
I mustn't use SVC model because, as sklearn documentation says, it works well with a dataset that is less than of thousands of samples and in our case we have a dataset of 30000 element.
Then the Naive Bayes approach is not the best way to solve the problem of detecting the optimizer and the compiler for a particular instruction. The reason why this is not a good classifier is due to the conditional independence assumption among features.

The best approach finds by me is the DecisionTree that is able to classify better in term of performance than other analyzed distributions. The wonderful thing about decision trees is that they really are as simple as they appear. Indeed, the best solution that I find is to use the DecisionTree. You can use them and interpret them correctly without any advanced statistical knowledge. Furthermore another prove, which demonstrate the quality is about ROC curve, in detail orange line shows that DecisionTree works better than others, where it's able to distinguish between positive class and negative class. This is true due is nearer to the ideal case that is when a classifier is able

to guess with a success.
For understanding which is the best configuration between OnevsOne and OnevsRest I decide to analyze the confusion matrix above, where OnevsOne is able to classify better, for the fact that the diagonal matrix of the OnevsOne has higher values in comparison with OnevsRest except for the entry[2,2] that is able to classify with 88% but in general the ore two entry have lower values. In conclusion all this informations are really required to be successful.The main drawback is the fact that Decision trees are also very expensive in terms of sample size. Every time the tree splits the data using a predictor, the remaining sample size reduces. This means that decision trees are not a good choice for small sample sizes.

9 Blind-test

The train-blind-dataset.csv is built through different lines of code where:

- **First step:** I decide to use same code for extract instruction without register then I fit the machine using the train dataset which is giving for training, after that I'm able to transform the two collection of instruction according the same dictionary length.
- **Second step:** I compute the result for binary and Multiclass problem according tree.DecisionTreeClassifier() routine, which is the same for OnevsOne multiclass problem, best than OnevsRest for this dataset and classifier, and Binary problem .
- **Third step:** Without splitting the dataset because I 've already had the train set and test set divided, I give in input to the routine tree.DecisionTreeClassifier().fit the test set that in our case is the dataset provided with targets,and the target that depends on the kind of problem that I want to resolve and in the prediction the dataset-blind that is composed with only instruction without register. This operation return a list of target that according the problem(Multiclass or Binary) is composed by (L and H) or(gcc and icc and clang).
- **Four step:** After this part I decide to recompute the jsonl to csv and now instead that before I don't discard registers, after compute this csv, I'm able to read data on csv and put the content to a list
- **Five step:** Now according itemgetter included into a list construction build a list of instruction with register included, done through a function similar the first used without a split of the content(register), and now I decide to use another function for building csv **buildthecsv** and print the result **printResult**

9.1 Result

I obtain for the binary problem : **H: 1261 L: 1739**
instead for the multiclass problem: **gcc 1014 clang: 1000 icc: 986**
we should to analyze that there could be a 25% of miss-classification which could be proved by the result obtained above.

10 Future works

It could be cool to make a comparison on metrics taking care the order of the assembler instructions, so using neural network based embedding of words into vectors, and takes context into account.In detail it's implemented by a python library for natural language processing **gensim** which contains a good word2vec implementation .

References

- Seminar-slide-Malware-detection
- Machine Learning for Malware Detection
- Library sklearn
- library json
- json to csv
- Bernoulli NB vs MultiNomial NB, How to choose among different NB algorithms
- Understanding ROC curve and AUC
- Discussion on Gaussian-Bernulli-Multinomial
- Detection of Android Malware in Your Pocket
- confusion-matrix