UNIVERSITA LA SAPIENZA

Machine Learning homework: Compiler provenance

Katarina Guderud

November 10, 2019

Abstract

When talking about compiler provenance, we refer to information about the compiler family, compiler version, optimization level and compiler-related functions. The method is used in many binary analysis applications today. To mention some of them, we have fingerprinting function, clone detection and authorship attribution. The importance of developing an efficient and automated approach for extracting compiler provenance is thus becoming more and more important. Compiler provenance answers fundamental questions of malware analysis, among other things, to analyse the performance of the program and instrumentation tools specific to particular compilers.

In this rapport I will present a practical approach which analyzes the syntax, structure and semantics of disassembled functions to extract compiler provenance. Given the binary code of a function, we want to identify the compiler and the optimization who produced it. We have looked into two of the classical classification problems; binary classifier and multi class classifier.

Key words: Compiler provenance, Binary classifier, Multi class classifier, Malware detection, feature engineering.

CONTENTS

1 Introduction

1.1 Malware analysis and classification problem

A malware is a contraction of malicious software. Easily explained malware is any piece of software that was written with the intent of damaging devices, stealing data, and generally causing a mess. Malware analysis is the study that concerns malicious samples with the aim of developing a deeper understanding about several aspects of malware. Malware analysis can be performed in two different ways: static analysis and dynamic analysis. Static analysis focuses on looking at the binary code using a disassembler, while dynamic analysis registrars and analyses the actions of the malware in a controlled environment. We will focus om the static analysis.

Often the training set is very large, and therefore we often think Machine Learning can be a good solution to solving the problem. Because of the huge amount of data, the analyst cannot analyze all the code. We therefore need to try to focus on a specific part of the sample which for some reason looks interesting.

When we apply machine learning to malware analysis, we can use two different strategies: supervised and unsupervised learning. Supervised learning needs labelled training set, while unsupervised learning does not need labelled training set. Supervised learning is good for classifying unknown samples, and this is also the strategy we used for the classification problems.

In this rapport I will present and discuss the different techniques used for the classification problems, how we tuned the parameters to get a better accuracy, and at the end I will conclude with what was the best solution.

2 About the project

Before presenting the solution I will give a brief presentation of how the project is given. The project consists of three different parts. The first part is about taking the dataset and manipulating it to something we can use in the learning process. The second part is to make a binary classifier which can predict if a function has been compiled with high or low optimization. The third part is to make a multiclass classifier which can predict which compiler a function has been compiled with. The dataset had three different compilers. In this part I will go trough how the dataset was structured, and in the next part I will focus more on how we solved the two classification problems.

2.1 Dataset

The dataset is formatted as a .jsonl file, in which each line is a .json file. Each .json file has the three following features:

- instructions: the assembly instruction for the function.
- opt: the ground truth label for optimization (H, L)
- compiler: the ground truth label for compiler (icc, clang, gcc)


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


Figure 2.1: The figure shows an example of how a line in the .jsonl file be.

The dataset contains 30 000 functions, in other words, 30 000 lines of .json functions. The compiler distribution is balanced - 10 000 functions for each compiler, the optimization distribution is not balanced, and the instructions were not balanced.

The first thing we had to do was to split the instructions in a way so that we only had to consider the mnemonic of each instruction. For example if a instruction was given as "mov eax 0xffffffff", we split the function by the first space and only keep "mov" as this is the instruction. In the end we had a list of strings for each functions instructions. The optimizations and compilers were also gathered in a list of strings, but since the length of these was constant, it was not necessary to manipulate that much.

The next part was to transform the input. We used the library sklearn's *CountVectorize()* function, which counts the number of occurances of a word, and gathers all the information in a matrix. Now we have input which can be used to train with. We used the function *train_test_split()*, also from sklearn, to split our set of arrays to train- and test-subsets.

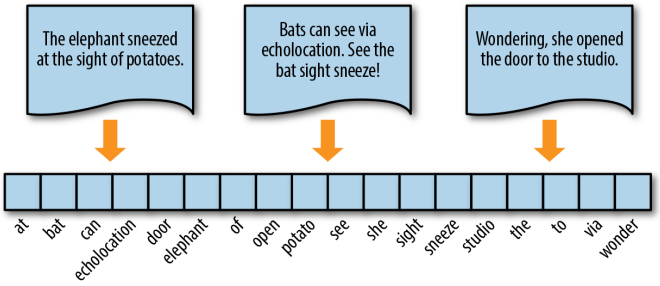


Figure 2.2: The figure shows how the function CountVectorize() works.

3 The Compiler Provenance Problem

The main task of this homework is to solve the Compiler Provenance Problem. That means we are going to build a binary classifier that can predict if a function has been compiled with optimization HIGH or LOW, and build a multiclass classifier that can predict the compiler used to compile a specific function.

3.1 Binary Classification

For the binary classification the classifier utilizes some training data to understand how some input data relates to a class. In our case, known High and Low optimization have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown function.

3.1.1 Gaussian Naive Bayes

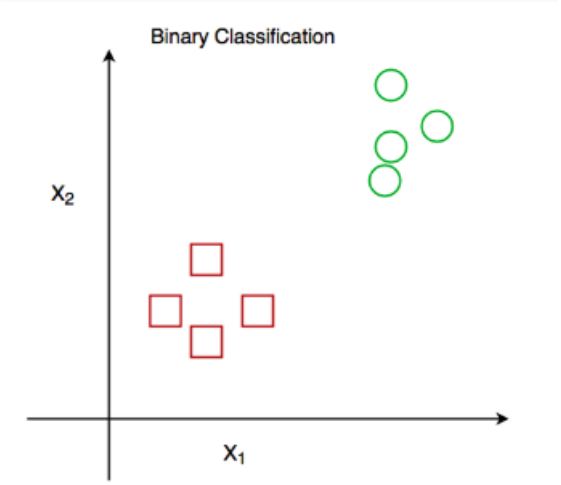
The Gaussian Naive Bayes *GaussianNB()* function, from sklearn library, is a supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

Bayes theorem given the relation between class variable y and dependent feature vector x_1 through x_n:

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

3.1.2 Logistic Regression

Logistic regression classifier is one of the most simple and commonly used Machine Learning algorithms for binary classification. The method describes and estimates the relationship between one dependent binary variable, and an independent variable. In our case the dependent is the optimization, and the independent is the instruction.



classification.png

Figure 3.1: The graph is representing an example of a binary classification. The two dataset (red square and green circle) are separated and classified.

3.2 Multiclass classification

In the multiclass classification problem, the number of classes is more than two. In our case, the number of classes is three. When the classifier is trained accurately it is supposed to predict which compiler the function is compiled with.

3.2.1 SVM

Support Vector Machine is a classifier which works by finding a hyperplane that has maximum distance between the data points. Data points falling on either side of the hyperplane will be classified as one or the other class. The dimension of the hyperplane depends on the number of features. In our case the input features is three, so the hyperplane will be a surface or a two dimensional plane.

3.2.2 Logistic Regression

The logistic Regression method for multiclass classification works in the same way as for binary classification. See 3.1.2.

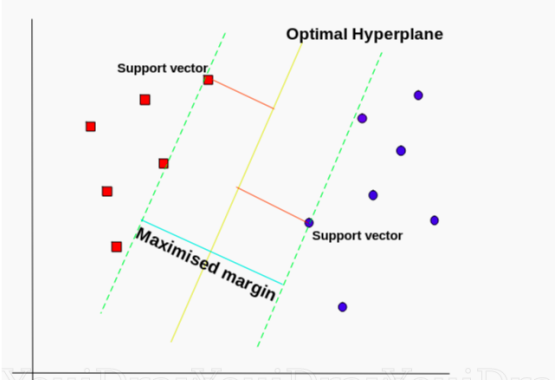


Figure 3.2: The figure shows how SVM divides the two classes into different classes. In our case we have three classes so the line will be presented as a two dimensional plane.

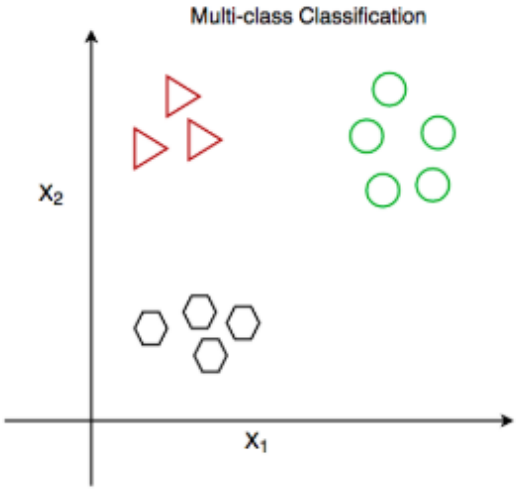


Figure 3.3: The graph is representing an example of a multiclass classification. The three dataset (red square, green circle and black pentagon.) are separated and classified.

4 Results

4.1 Improvement of accuracy

There are a lot of things you could do to improve the accuracy of a classifier. Though, I decided to stick to two simple methods which made the accuracy fulfilling.

By studying the length of the optimization set for the binary classification problem, we noticed that the number of highs and lows in the set was not balanced. Therefore we integrated some functionality in the code which did such that the number of high optimization func-

tion equals number of low optimization functions. By doing this, the accuracy raised with approximately 10% in the binary classification problem.

Also by tuning the parameters for the vectorizer, the accuracy improved some percentages. For example for the multiclass classifier we ended up setting max_features = 100, and for the binary classifier we sat ngram_range = (1,3).

4.2 Binary classification

For binary classification I decided using the Gaussian Naive Bayes classifier and Logistic regression. The results shows that Logistic Regression gives the best accuracy. The reason why may be because Naive Bayes assumes that the features are conditionally independent, while in real life they are never perfectly independent. In other words Gaussian NB have higher bias but lower variance compared to Logistic Regression.

4.2.1 Gaussian Naive Bayes

Figure 4.1 shows us the result for the Gaussian Naive Bayes classifier.

```
Accuracy for Gaussian NB is 0.796
              precision    recall  f1-score   support

           H      0.755     0.875     0.811      4012
           L      0.852     0.718     0.779      4031

    accuracy                          0.796      8043
   macro avg      0.804     0.796     0.795      8043
weighted avg      0.804     0.796     0.795      8043

[[3510  502]
 [1137 2894]]
```

Figure 4.1: The figure shows hows the which numbers the Gaussian Naive Bayes classifier obtained from the classification report.

4.2.2 Logistic Regression

Figure 4.2 shows us the result for the Logistic Regression classifier.

4.3 Multiclass classification

For multiclass classification I decided using the Support Vector Machine and Logistic Regression. The logistic regression was first of all chosen to compare with the binary classifier. The results shows that also here the Logistic Regression classifier got the best result. The two classifiers are very similar, but there is one big difference; the loss function. SVM minimizes hinge loss while LR minimizes logistic loss. Logistic loss diverges faster than hinge loss, so generally

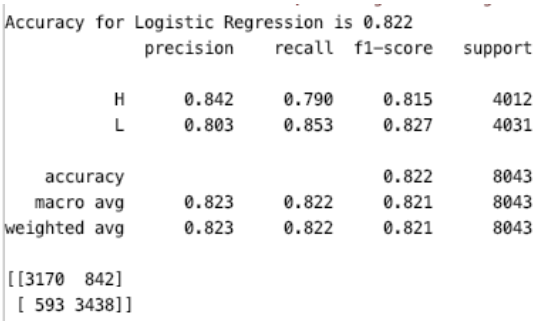


Figure 4.2: The figure shows hows the which numbers the Logistic Regression classifier obtained from the classification report.

Logistic Regression will be more sensitive to outliers. In our case, the accuracy of Logistic Regression was higher than for SVM. The reason for this may be because the dataset was quite balanced, which leads to few outliers. Also it is possible the parameters for the classifiers could be more tuned in.

4.3.1 Support Vector Machine (SVM)

Figure 4.3 shows us the result for the Support Vector Machine classifier.

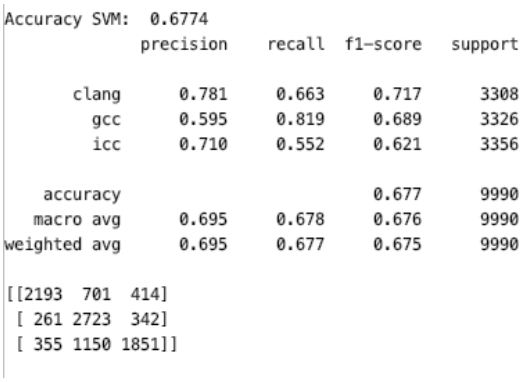
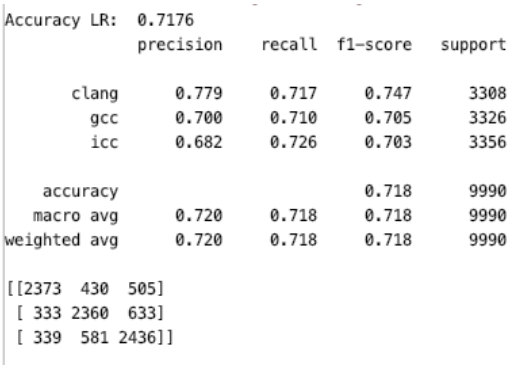


Figure 4.3: The figure shows hows the which numbers the Support Vector Machine classifier obtained from the classification report.

4.3.2 Logistic Regression

Figure 4.4 shows us the result for the Logistic Regression classifier.



```
Accuracy LR:  0.7176
              precision    recall  f1-score   support

       clang      0.779     0.717     0.747      3308
         gcc      0.700     0.710     0.705      3326
         icc      0.682     0.726     0.703      3356

    accuracy                          0.718      9990
   macro avg      0.720     0.718     0.718      9990
weighted avg      0.720     0.718     0.718      9990

[[2373  430  505]
 [ 333 2360  633]
 [ 339  581 2436]]
```

Figure 4.4: The figure shows hows the which numbers the Logistic Regression classifier obtained from the classification report.

5 SUMMARY

The compiler provenance problem gives insight in how the compiler and instructions can influence the optimization. Through this report we have discussed features about the dataset and how we could manipulate the dataset. We have also discussed how we could obtain the most precise classifier, and which methods we used. In the end it ended up being the logistic regression classifier which won the price for best accuracy for both classifiers, even though SVM in theory should have better accuracy than LR. The reason for this may be because of some error with the vectorizer. The logistic regression classifier had an accuracy of 82% for the binary classification, and 72% for the multiclass classification. Still there are a lot of small adjustments one could do to obtain a higher accuracy.

[strings]underscore

REFERENCES

[1] Support Vector Machine vs Logistic Regression
https://towardsdatascience.com/support-vector-machine-vs-logistic-regression-94cc2975433f

[2] Tips and Tricks for Multi-Class Classification
https://medium.com/@b.terryjack/tips-and-tricks-for-multi-class-classification-c184ae1c8ffc

[3] Naive Bayes
https://scikit-learn.org/stable/modules/naive_bayes.html

[4] Logistic Regression
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[5] Naive Bayes vs Logistic Regression
https://medium.com/@sangha_deb/naive-bayes-vs-logistic-regression-a319b07a5d4c

[6] Support Vector Machine
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC