Compiler Provenance

HW1 - Machine Learning
"Sapienza" University of Rome

Valerio Coretti 1635747

November 11, 2019

Abstract

Nowadays the amount of malicious programs is increasing exponentially and the number of victims are very high. For this reason Malware Analysis is a crucial point in Cybersecurity.

Machine Learning plays a key role in malware analysis beacause it can simplify various processes in several ways. The goal of an analyst is to capture malicious aspects and traits having the broadest scope, and Machine Learning is a natural choice to support this process of knowledge extraction.

In this project we use Machine Learnig techniques to simplify the problem of Reverse Engineering of a malware. When an analyst has a new malware sample to analyze, the first thing to do is to disassemble it. This permits to the analysist to look at the binary code of all the functions contained inside the malware. The problem is that a human cannot analyze all the code but he try to focus on specific part of the sample that for some reason looks more interesting. But in the most cases malware binaries are often stripped or the string inside a binary can be encrypted. In this scenario, an automated approach is a good way for reverse malware binaries.

There are several problems in binary analysis, one of this is the *Compiler Provenance*.

Problem: given the binary code of a function, which are the compiler and the optimization who produced it?

To address this problem, Machine Learning is used with notable results.

In this paper, we will discuss an approach based on *Supervised Learnin* and we will try to solve this two classification problems.

1 Introduction

We have two problems: *Optimization Detection* and *Compiler Detection*. To solve these problems with Machine Learning we analized a large dataset that is a ".jsonl" file that contains 30000 functions. Each row of the file is a json object like that:

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

The first problem is a Binary Classification problem because we have only two types of optimization: High or Low. Instead the second is a Multiclass Classification problem and we have three classes of compiler: gcc, icc, clang.

The compiler distribution is very *balanced*, we have 10000 functions per compiler, instead the optimizations distribution is *unbalanced*, in fact we have 17924 low and 12076 high.

What we will see in the following sections is a description of what we did for solve these problems. We will begin by illustrating how we transformed the dataset up to which model we have definitively chosen. By doing this we explored two ways of features extraction and compared three types of models: *Bernoulli Naive Bayes*, *Multinomial Naive Bayes*, and finally *Support Vector Machine with Linear kernel*.

We worked with *Scikit-Learn* that is a simple and efficient Python Open-Source tools for data mining and data analysis.

2 Preprocessing

The solution we thought is similar to a text analysis, and we based our idea on the Spam Classification Problem. Therefore we have done some operations to transform the dataset for our purpose. Given a function (equivalent to a row) we have transformed the instruction set into a single string. Such as:

> *"instructions"*: ["xor edx edx", "cmp rdi rsi", "mov eax dword [localch]", "seta dl", "cmovae eax edx", "ret"]
>
> becomes
>
> *"instructions"*: ["xor edx edx cmp rdi rsi mov eax dword [localch] seta dl cmovae eax edx ret"]

2.1 Thinning dataset

The main difference with the spam problem is that we have transformed the dataset of the functions in such a way as not to take all the instructions and operands inside, but limiting ourselves to taking the *mnemonics* of the assembly instructions and in addition we have selected *memory accesses* and *register accesses*. This choices are justified by the fact that the set of all instructions is unbounded but if we consider only mnemonics the set is smaller. Memory and register accesses are important because when a compiler optimizes a function it tries to reduce the memory accesses and to use as much as possible the registers that are faster then memory.

To extract memory accesses we based on four key word:

- *"qword", "dword", "word", "byte"*

This commands mean: *"take n bytes from a certain data segment"*. Instead for registers we choosed the main ones:

- Data Registers: *"eax", "ebx", "ecx", "edx"*
- Pointer Registers: *"ebp", "esp"*
- Index Registers: *"edi", "esi"*

Done this, the previous example becomes:

> *"instructions"*: ["xor edx edx cmp rdi rsi mov eax dword seta dl cmovae eax edx ret"]

Now we are ready for the extraction of the features.

2.2 Features Extraction

The list of features can be processed like a list of words in a document. For this reason we use a sklearn vectorizer. The power of a vectorizer is that it not take only term by term (called 1-grams mode) but it can also keep the word order in a text by making more words a single feature. This function is called n-grams. For example for 1-grams if we have only the previous seen string the features is every single word in the string itself, but if we set the range of n-grams at (1, 3) we will have:

> ['cmovae', 'cmovae eax', 'cmovae eax edx', 'cmp', 'cmp rdi', 'cmp rdi rsi', 'dl', 'dl cmovae', 'dl cmovae eax', 'dword', 'dword seta', 'dword seta dl', 'eax', 'eax dword', 'eax dword seta', 'eax edx', 'eax edx ret', 'edx', 'edx cmp', 'edx cmp rdi', 'edx edx', 'edx edx cmp', 'edx ret', 'mov', 'mov eax', 'mov eax dword', 'rdi', 'rdi rsi', 'rdi rsi mov', 'ret', 'rsi', 'rsi mov', 'rsi mov eax', 'seta', 'seta dl', 'seta dl cmovae', 'xor', 'xor edx', 'xor edx edx']

In our project as a first attempt we try with 1-grams mode but as we will see it not perform very well, then swapping to (1, 3)-grams we have really achieved pretty results. Note that, as you can see from the output, for a very large dataset the number of features, with (1, 3)-grams, can be a very high number, for this reason we only consider the top 10000 features ordered by term frequency across the corpus.

Furthermore for the extraction we choose a precise vectorizer: *Tf-Idf Vectorizer*. This object convert a collection of raw documents to a matrix of TF-IDF features. Tf-idf is the short form of *term frequency–inverse document frequency*, it doesn't count the recorrence of a word in a certain row but is a numerical statistic that reflect how important a word is in a document.

The function can be split into two factors: the first factor of the function is the number of terms in the document. In general, this number is divided by the length of the document to prevent that longer documents being privileged.

$$tf_{i,j} = \frac{n_{i,j}}{|d_j|}$$

where $n_{i,j}$ is the number of occurrences of the term i in the document j, while the denominator is simply the size, expressed in number of terms, of the document j.

The other factor of the function indicates the general importance of the term i in the collection:

$$idf_i = log\frac{|D|}{|\{d : i \in d\}|}$$

where $|D|$ is the number of documents in the collection, while the denominator is the number of documents containing the term i

Therefore we have:

$$(tf - idf)_{i,j} = tf_{i,j} * idf_i$$

Now we have our structure. The last important thing to do is divide the dataset into two parts, one for the training and one for the evaluation. The training set is 2/3 of the original dataset and the evaluation set is the remaining 1/3. We decided also to keep the proportion of the classes the same in both problems.

This division is performed choosing random elements, so running different times the preprocessing phase will affect the final results a bit.

3 Model

To get to the final result we explored three different models: *Bernoulli Naive Bayes*, *Multinomial Naive Bayes*, and finally *Support Vector Machine with Linear kernel*. In the following sections we explaines how these models works and then we will discuss about the evaluation.

3.1 Naive Bayes Classifiers

The first chosen model is the *Naive Bayes classifier*, a probabilistic model used mainly for text classification. In this model a document is viewed as a collection of words. As we said before the list of features can be processed like a list of words in a document without losing information, so this model is suitable for our problem.

In our case we divided the problems, in the first we have two classes c_1 and c_2, respcetively *High* and *Low* optimization, in the second we have three classes c_1, c_2 and c_3 as *"gcc"*, *"icc"* and *"clang"* .

We considered two variants of Naive-Bayes, the *Bernoulli* and the *Multinomial.*

The Naive-Bayes model classifies an instance x, represented as a set of features $< x_0 ... x_n >$, with the class c_m which maximizes the probability $P(c_m|x, D)$, where D is the dataset. The class is computed using the following function:

$$argmax_{c \in C}\{P(c|x, D)\}$$

that is equal to:

$$argmax_{c \in C}\{P(x|c, D)P(c|D)\}$$

And, assuming the conditional independece of the features (words) x_i, is also equal to:

$$argmax_{c \in C}\{P(c|D)\Pi_i P(x_i|c, D)\}$$

In the multinomial variant, this function becomes:

$$argmax_{c \in C}\{\frac{P(c|D)\Pi_i P(x_i|c,D)^{n_{x_i x}}}{P(x|D)}\}$$

Where $n_{x_i x}$ is the number of times the word x_i is in the document x. $P(x_i|c, D)$ is defined in the following manner:

$$P(x_i|c,D) = \frac{1 + \sum_{d \in D_c} n_{x_i d}}{k + \sum_j \sum_{d \in D_c} n_{x_j d}}$$

With D_c defined as the collection of documents in the training set of class c. The additional 1 and k (the cardinality of the vocabulary) are the solutions to the zero-frequency problem.

3.2 Support Vector Machine

Here we are in the family of the Linear Classification Models. The basic idea of this type of models is that the instances in a dataset (e.g. binary) are *linearly separable* if exist a *linear function* (hyperplane) such that the instance space is divided in two regions, in one you find only positive examples in the other only negative ones. Support Vector Machine is a solution for the problem above and it is more sophisticated then the Naive Bayes.

SVM is a method in which we don't want to find a line that completely divides instances simply, but which does it in the best way. We want a line that guarantees the maximum distance between points of the two classes from it.

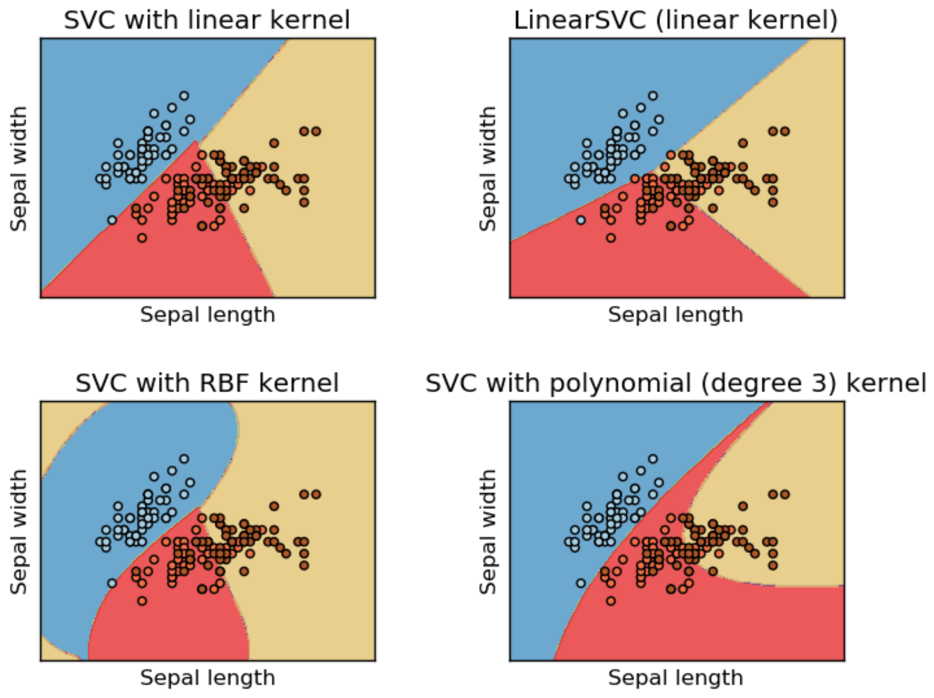
So in Support Vector Machine we always look for 2 things:

- Setting a larger margin
- Lowering misclassification rate

Now the problem is that this two things are kind of contradictory. If we increase margin, we will end up getting a high misclassfication rate on the other hand if we decrease a margin, we will end up getting a lower misclassification rate. The answer is parameter C.

C is the penalty parameter of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly. Increasing C values may lead to overfitting the training data.

Another key point for a better use of SVM are the *Kernel functions*. These selects the type of hyperplane used to separate the data. Using *linear* kernel will use a linear hyperplane. Then we have kernel like *rbf* or *poly* that uses a 'non linear' hyperplane for the classification.



For our problem we use Support Vector Machine with Linear Kernel and for penalty parameter we set the standard value $C = 1$ because it give the best performance.

Now that we have a clearer overview of the methods we have used for our problem, we will discuss the evaluation of the results.

4 Evaluation

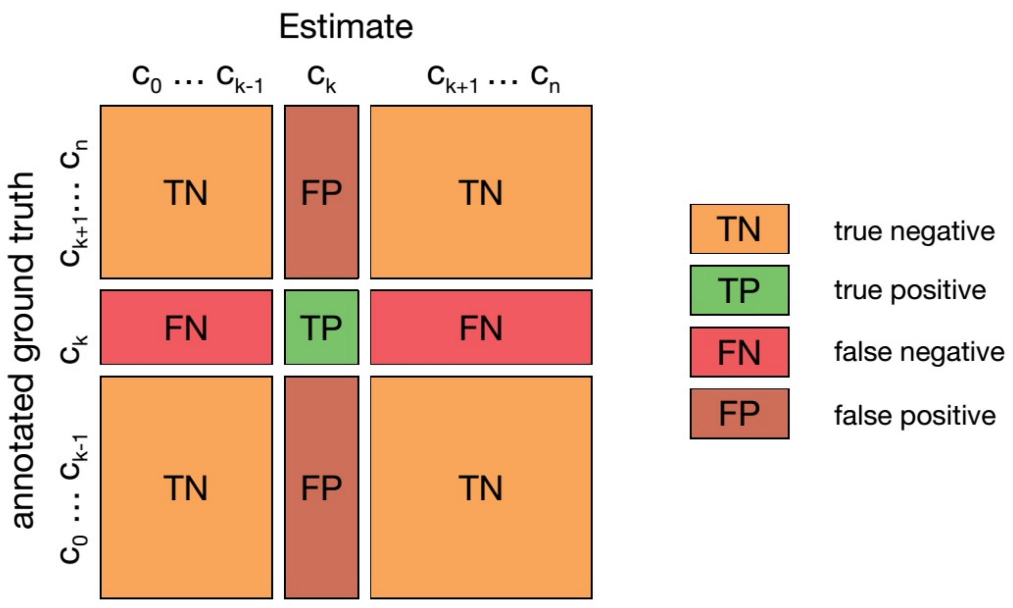
In order to perform a better classification, we evaluated the performance of training the model using two different features: *1-gram* and *3-grams*.

The output parameters that we considered for the evaluation are the following [1]:

- Precision: $TP/(TP + FP)$
- Recall: $TP/(TP + FN)$
- False positives rate: $FP/(FP + TN)$
- False negatives rate: $FN/(FN + TP)$
- Accuracy: $(TP + TN)/(TP + FN + TN + FP)$
- F1-score: $2 * (Recall * Precision)/(Recall + Precision)$

Furthermore we also based our analysis in the results posted by the *Confusion Matrix* that represents in each entry how many instances of class C_i is misclassified as an element of class C_j. So, the main diagonal contains the true positive and true negative respectively for each class.

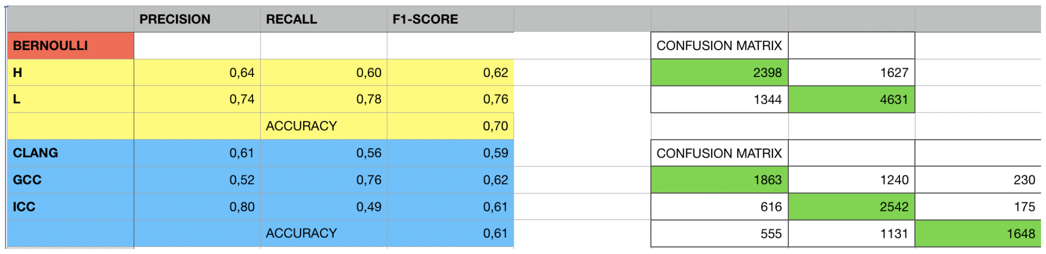
[1]Legend: TP = true positives, TN = true negatives, FP = false positives, FN = false negatives



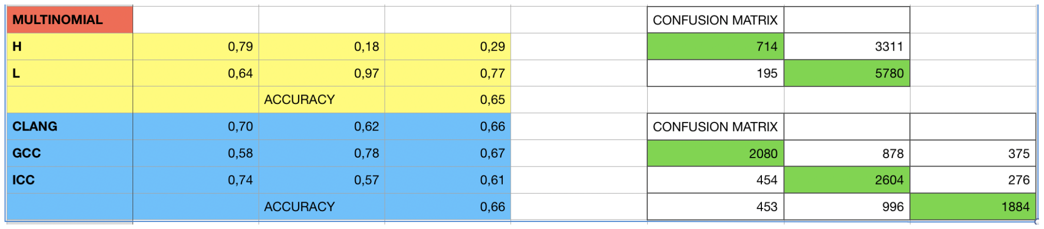
In the following sections we analyze the results and how we arrived at the final choice of the model. We divided the discussion, first we analyze the 1-gram feature and we will show why we decided to change strategy and use the 3-grams feature.

4.1 1-gram feature

This type of feature extraction take only the words inside the functions and don't consider the order, for this reason are very few, about 400 features. As a first attempt we try with the Bernoulli Naive Bayes and Multinomial Naive Bayes. The results for our classification problems are the following:



	PRECISION	RECALL	F1-SCORE				
BERNOULLI					CONFUSION MATRIX		
H	0,64	0,60	0,62		2398	1627	
L	0,74	0,78	0,76		1344	4631	
		ACCURACY	0,70				
CLANG	0,61	0,56	0,59		CONFUSION MATRIX		
GCC	0,52	0,76	0,62		1863	1240	230
ICC	0,80	0,49	0,61		616	2542	175
		ACCURACY	0,61		555	1131	1648



MULTINOMIAL					CONFUSION MATRIX		
H	0,79	0,18	0,29		714	3311	
L	0,64	0,97	0,77		195	5780	
		ACCURACY	0,65				
CLANG	0,70	0,62	0,66		CONFUSION MATRIX		
GCC	0,58	0,78	0,67		2080	878	375
ICC	0,74	0,57	0,61		454	2604	276
		ACCURACY	0,66		453	996	1884

We stopped the experimentation here due to the very low accuracy. Note that the accuracy metric is not very effective for the evaluation of the optimization detection because we have an unbalanced ditribution and for this reason we pay more attention in the False Positive Rate and False Negative Rate. In addition False Positive Rate is a key measure and must be maintained as low as possible.

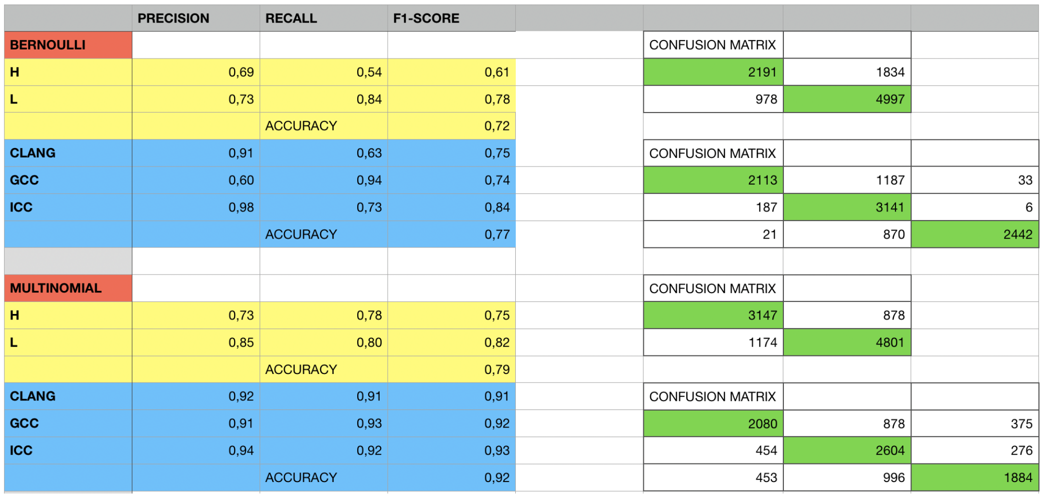
These models with these features seems to classify a High optimization as a Low too often and also in the cases of the compiler detection the numbers of misclassification are very high.

Furthermore as we can see in the picture above the results of the optimization classification with Multinomial are very bad because for the H class we have $F1 - score = 0,29$ and $Recall = 0,18$ this means that of all the functions that have an high optimization this model ranks very few.

To improve performance we first decided to improve features extraction and we analyze the results with the same models.

4.2 3-grams feature

This type of feature extraction making more words a single feature and consider the order, for this reason the number of features is very high but we only consider the top 10000 features ordered by term frequency across the corpus. In general, the multinomial variant is better than the bernoulli but we tried anyway to confirm this trend also for this case. Using the multinomial, the results associated with each set are:



	PRECISION	RECALL	F1-SCORE				
BERNOULLI					CONFUSION MATRIX		
H	0,69	0,54	0,61		2191	1834	
L	0,73	0,84	0,78		978	4997	
		ACCURACY	0,72				
CLANG	0,91	0,63	0,75		CONFUSION MATRIX		
GCC	0,60	0,94	0,74		2113	1187	33
ICC	0,98	0,73	0,84		187	3141	6
		ACCURACY	0,77		21	870	2442
MULTINOMIAL					CONFUSION MATRIX		
H	0,73	0,78	0,75		3147	878	
L	0,85	0,80	0,82		1174	4801	
		ACCURACY	0,79				
CLANG	0,92	0,91	0,91		CONFUSION MATRIX		
GCC	0,91	0,93	0,92		2080	878	375
ICC	0,94	0,92	0,93		454	2604	276
		ACCURACY	0,92		453	996	1884

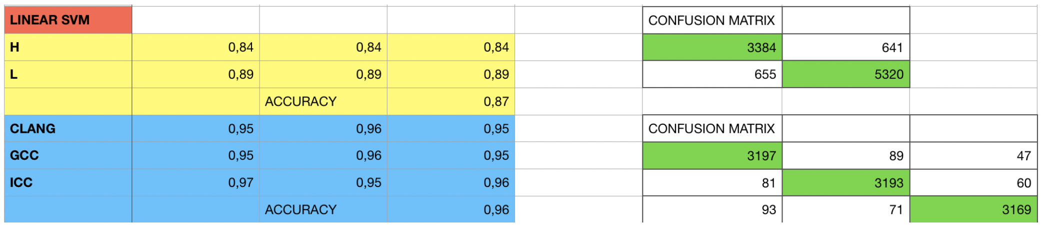
As we can see with BernoulliNB and 3-grams we slightly improve our result but as we expected with MultinomialNB the results start to be really good. Analyze it. About the compiler detection we have an accuracy of 0,92 and this is a relly good result because it ensure a low value for the FPR. Instead for optimization detection the result is still a bit low. We report below the FP-rate and FN-rate and we continue our analysis on this parameters because Precision and Recall can be easily related to FPR and FNR, in fact, recall is simply 1 - FNR and the precision is higher with low FPR values.

Classes	FP-rate	FN-rate
H	0,19648	0,21813
L	0,21813	0,19648

At this point significative better results can be achieved only using a more sophisticated model and for this reason we choose Linear Support Vector Machine.

4.3 Linear SVM

As a last attempt to improve our performances we use Support Vector Machine with Linear kernel, standard penalty parameter $C = 1$ and 3-grams. Goes to show the results.



LINEAR SVM					CONFUSION MATRIX		
H	0,84	0,84	0,84		3384	641	
L	0,89	0,89	0,89		655	5320	
		ACCURACY	0,87				
CLANG	0,95	0,96	0,95		CONFUSION MATRIX		
GCC	0,95	0,96	0,95		3197	89	47
ICC	0,97	0,95	0,96		81	3193	60
		ACCURACY	0,96		93	71	3169

Classes	FP-rate	FN-rate
H	0,10962	0,15925
L	0,15925	0,10962
CLANG	0,02609	0,04080
GCC	0,02400	0,04229
ICC	0,01604	0,04920

Considering this measures, this model is for sure the best than the others in fact we have improved the compiler detection performance even more, but the most convincing result is the fact that we have increased the optimization detection rates bringing them to much better values. As you can see we have taken the FPR from 0.19 to 0.10 for H and from 0.21 to 0.15 for L.

At this point we finish our analysis here and we can draw our conclusions.

5 Conclusion

From these results we could see the fact that SVM is certainly one of the best models, therefore continue to test other models probably wouldn't be worth. The thing that surely can improve our results even more is the extraction of features. In fact at the beginning we showed that we based our analysis only perform the mnemonics of function and the memory and register accesses. Surely by investigating more about the behavior of the various compilers and the many assembly instructions we could find better solutions. For now we end our experience here.

Note that we perform the *Blind-dataset* with the last combination: Linear SVM with 3-grams.

References

[1] *SKlearn Documentation.* `https://scikit-learn.org/stable/documentation.html`

[2] C. Bishop *Pattern Recognition and Machine Learning.*

[3] *Wikipedia - Tf-Idf.* `https://en.wikipedia.org/wiki/Tf{idf`