Automatic Identification of CPU Instruction Sets From Binaries

by

Sebanjila Kevin Bukasa; Benjamin Farinier;
Omar Jaafor; Nisrine Jafri

Airbus
Mouad Abouhali; Philippe Biondi

REDOCS 2016

28 Octobre 2016
Gif sur Yvette

Outline

- Introduction

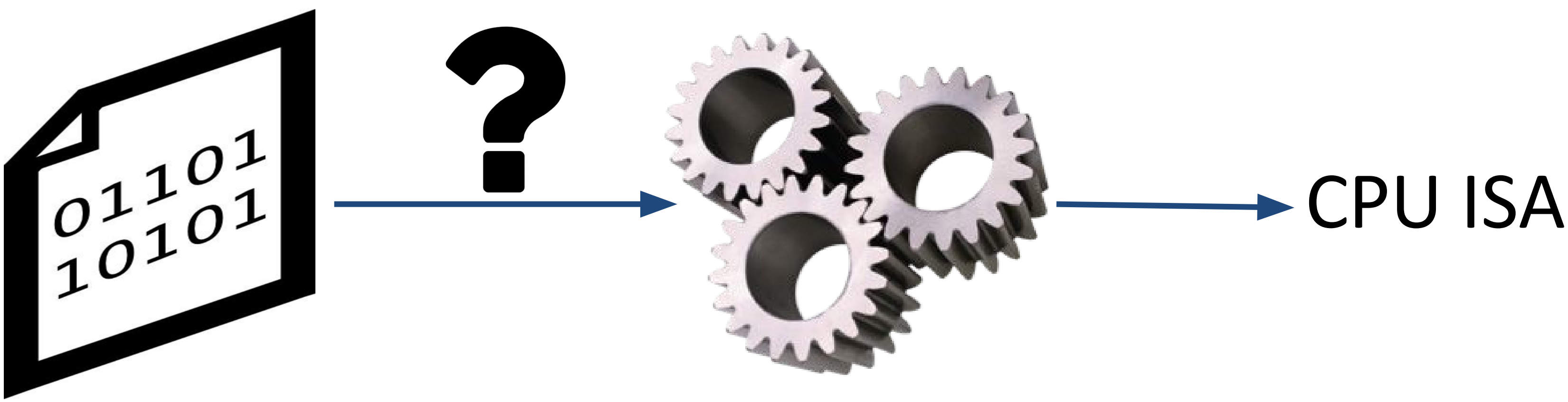
Approach 1 Approach 2

- Background
- Process
- Implementation
- Result

- Conclusions

Goal

Determine an efficient way to automatically identify the CPU ISA based on binary code only.



Motivations

- It is a prerequisite for reverse engineering.
- Allows to test the binaries on its intended platform.
- Automate a task that was performed manually, saving precious time.

Target Architectures

Primary targets

- x86
- ARM
- PPC
- MIPS

Secondary targets

- PIC
- Arc
- ARcompact
- Intel 8051
- etc.

Possible Approaches

- Heuristics / Pattern matching
- Statistical
- Machine Learning

Approach 1: Statistical Discrimination

Determine the distribution of some features that differ from one platform to another.

Pros:

- ❑ Allow analysts to understand decision patterns.

Cons:

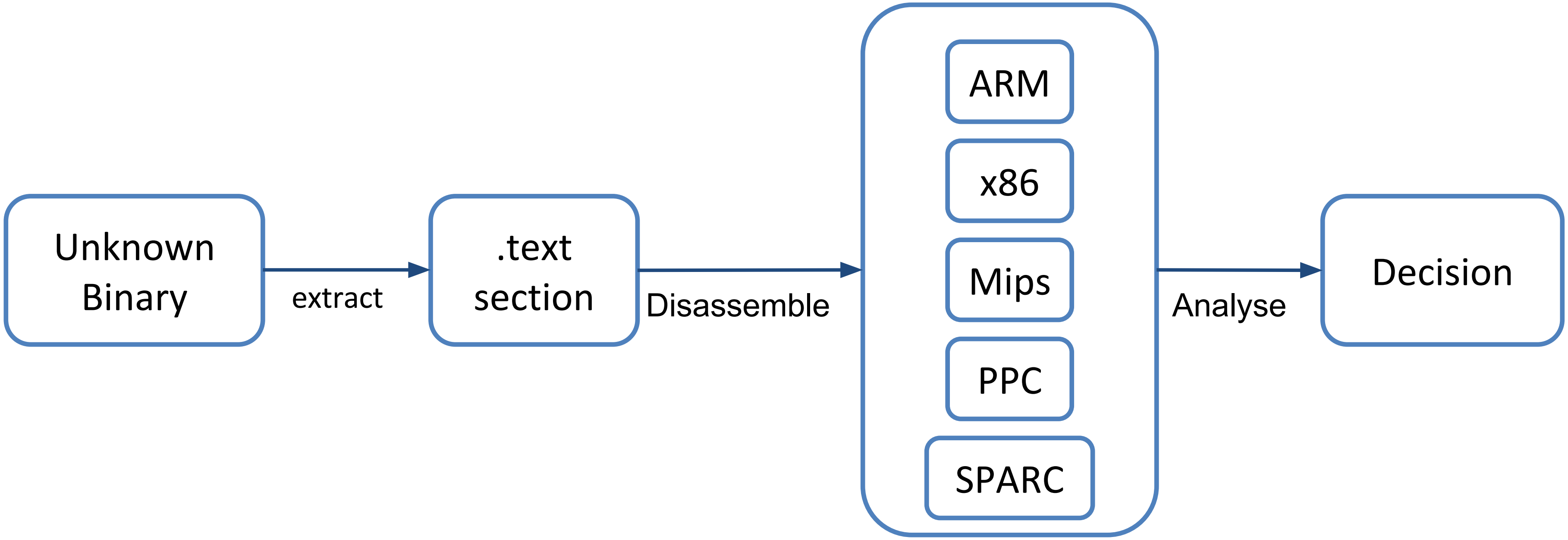
- ❑ Requires time consuming feature engineering.

1.Background : The Shannon entropy

Mathematical function that intuitively corresponds to the amount of information contained in or issued by a source.

$$H_b(X) = -\mathbb{E}[\log_b P(X = x_i)] = \sum_{i=1}^{n} P_i \log_b \left(\frac{1}{P_i}\right) = -\sum_{i=1}^{n} P_i \log_b P_i .$$

1.Process

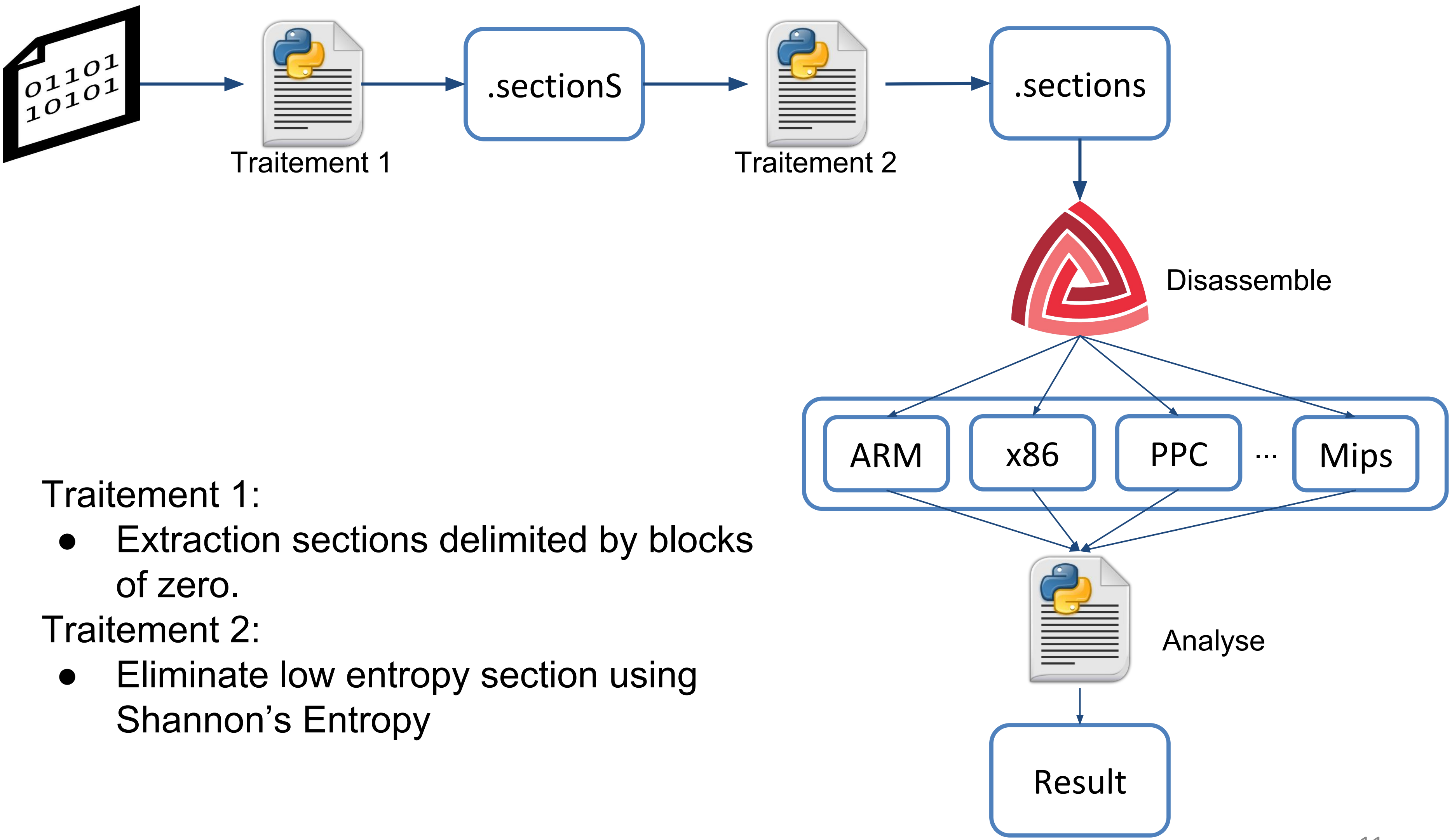


Unknown Binary
extract
.text section
Disassemble
ARM
x86
Mips
PPC
SPARC
Analyse
Decision

1.Implementation:
Finding code section

- First idea : use long sequence of zeros as delimiters in order to find section
- Then try to guess which one of those sections is the .text section
- Shannon entropy as a first approximation

1.Implementation



Traitement 1:

- Extraction sections delimited by blocks of zero.

Traitement 2:

- Eliminate low entropy section using Shannon's Entropy

1.Implementation : Disassemble

- Using Capstone we disassemble all splitted files
    - Code is isolated so we can retrieve interesting things

1.Implementation : Disassemble 2

- We make some statistics about this pieces of code in each languages:
    - Number of jumps
    - Jump addresses
    - Name and number of registers used ...

- Decision made by results

1.Result : Analyse results

- Hypothesis:
  - There is a limited number of jumps:
    - No more than 10% in general
  - Jumps can only be done to regular addresses
    - Jump to 0xFFFF can be suspicious
  - First registers are the more used by compilers
    - Passing arguments, etc.
  - There is no multiple memory accesses in general
- Decision made from these hypothesis

1.Results

- Funny things : unknown files seem to use crypto stuffs ;-)
- Extraction and statistics are working
- No decision made but some ideas:
  - Focus on architectures specifity
  - Jumps are rare compared to branchs
  - First registers are often used

Approach 1: Possible improvement

- Use other measure than Shannon's entropy
  - Ideally measure based on bytes distribution in .text sections
- More architecture based criteria
- Add other disassemblers

Approach 2: Machine Learning

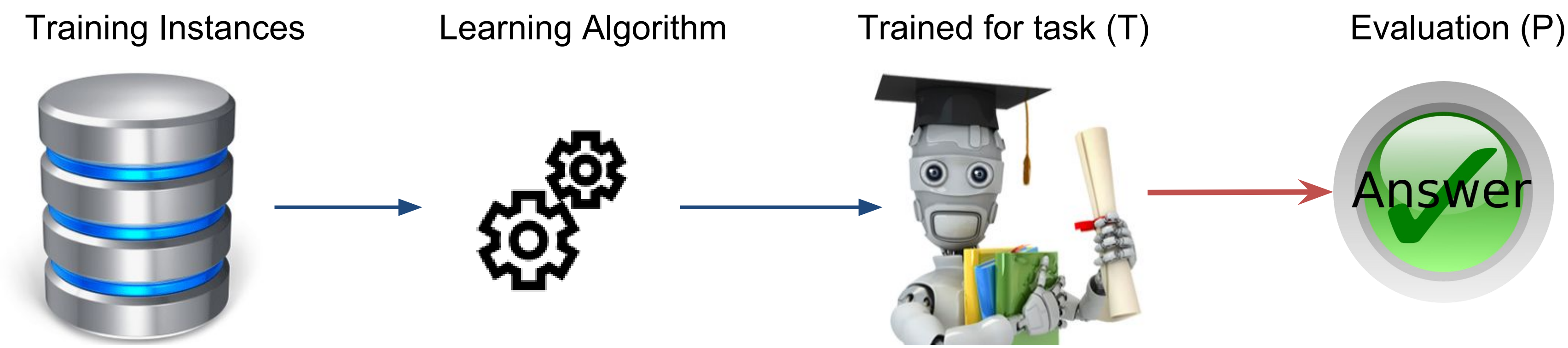
Pros:

- Less complex feature engineering (by comparison with statistical)
- Good performance

Cons:

- Difficult to interpret
- Require large sample for training

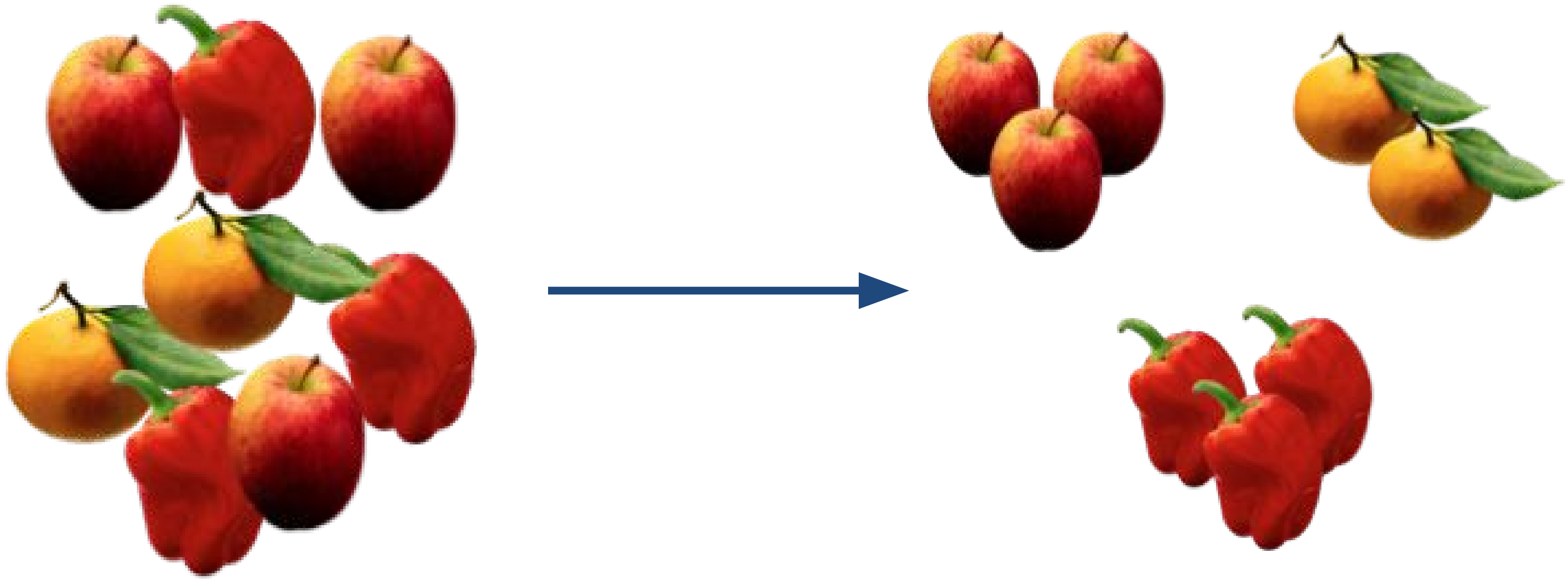
2.Background : Machine Learning



Application area:

- Classification,
- Regression,
- Outlier detection,
- Information retrieval, etc

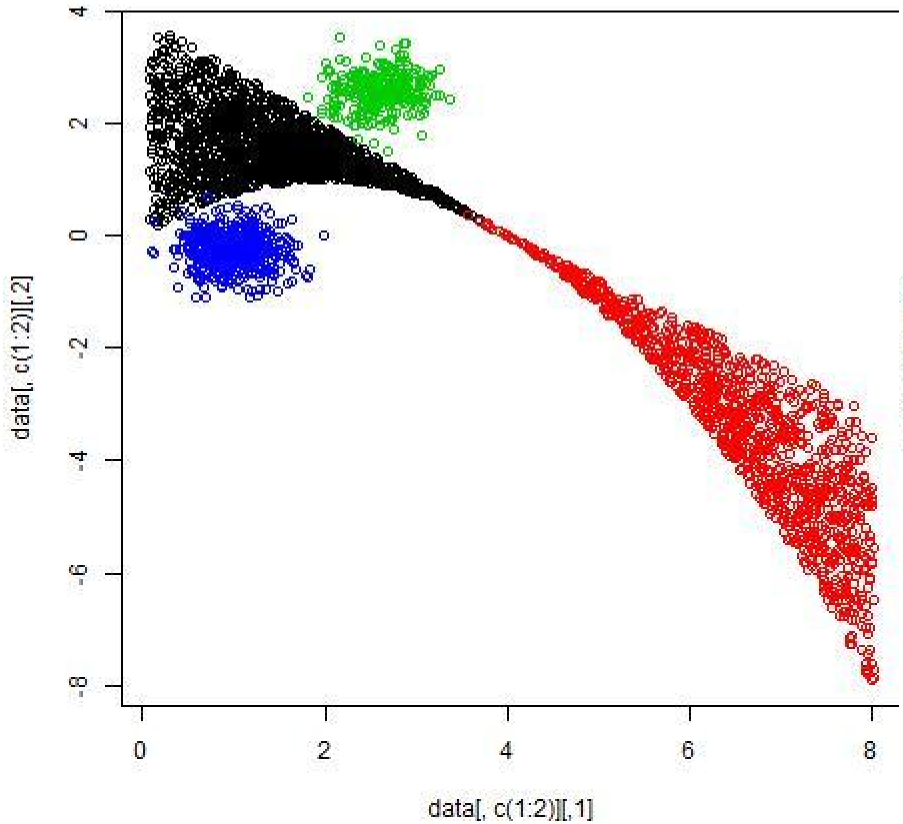
2.Background: Classification



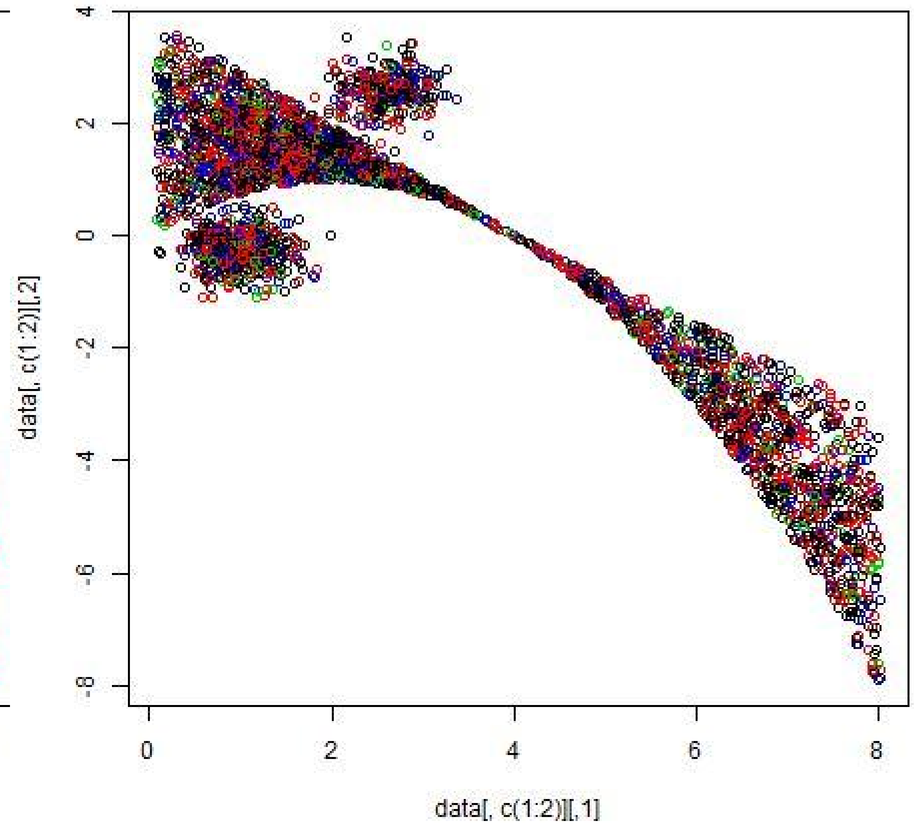
Type:

- Non supervised
- Supervised

2.Background:
Supervised Classification:Feature selection

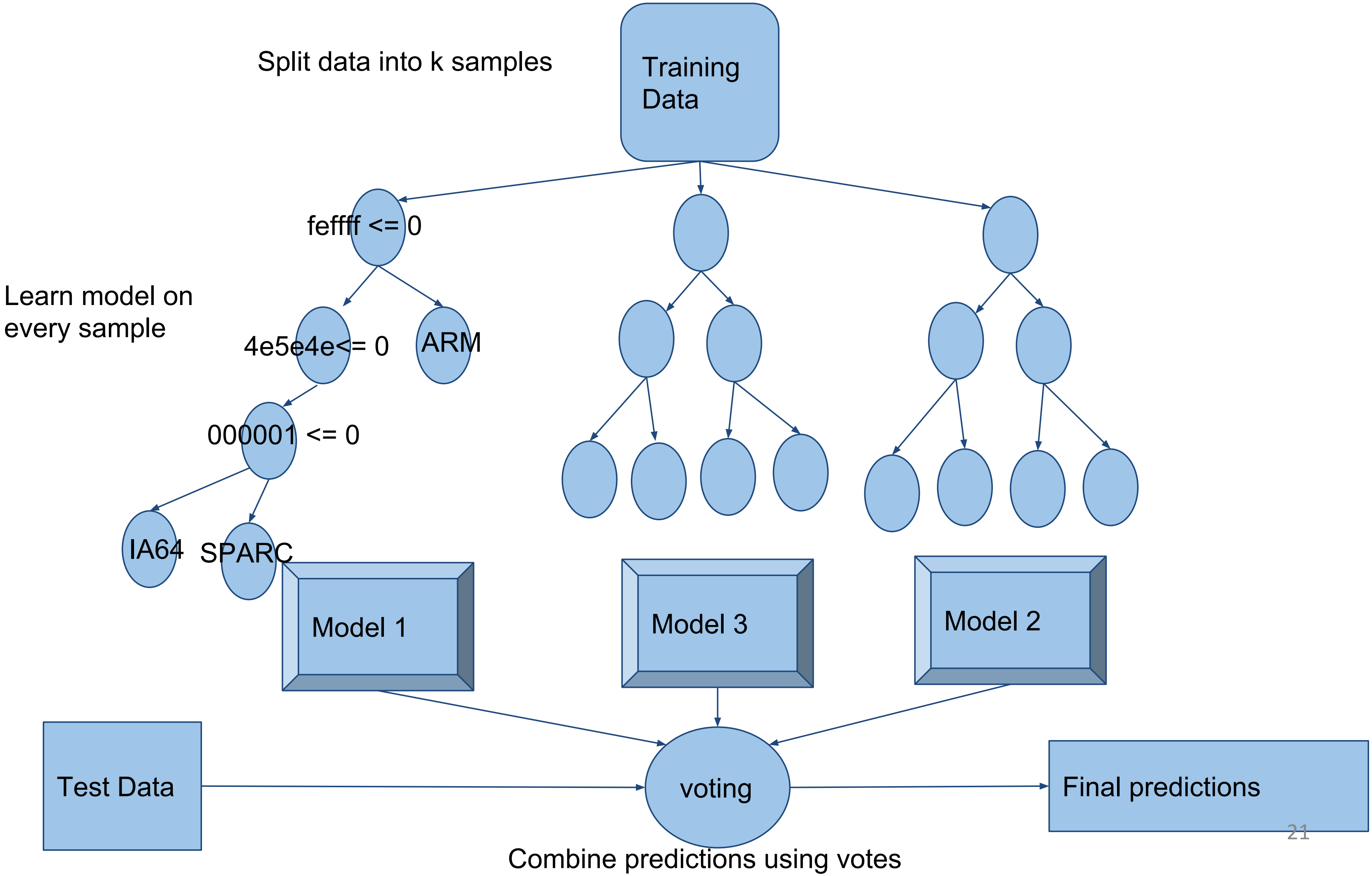


Good feature selection

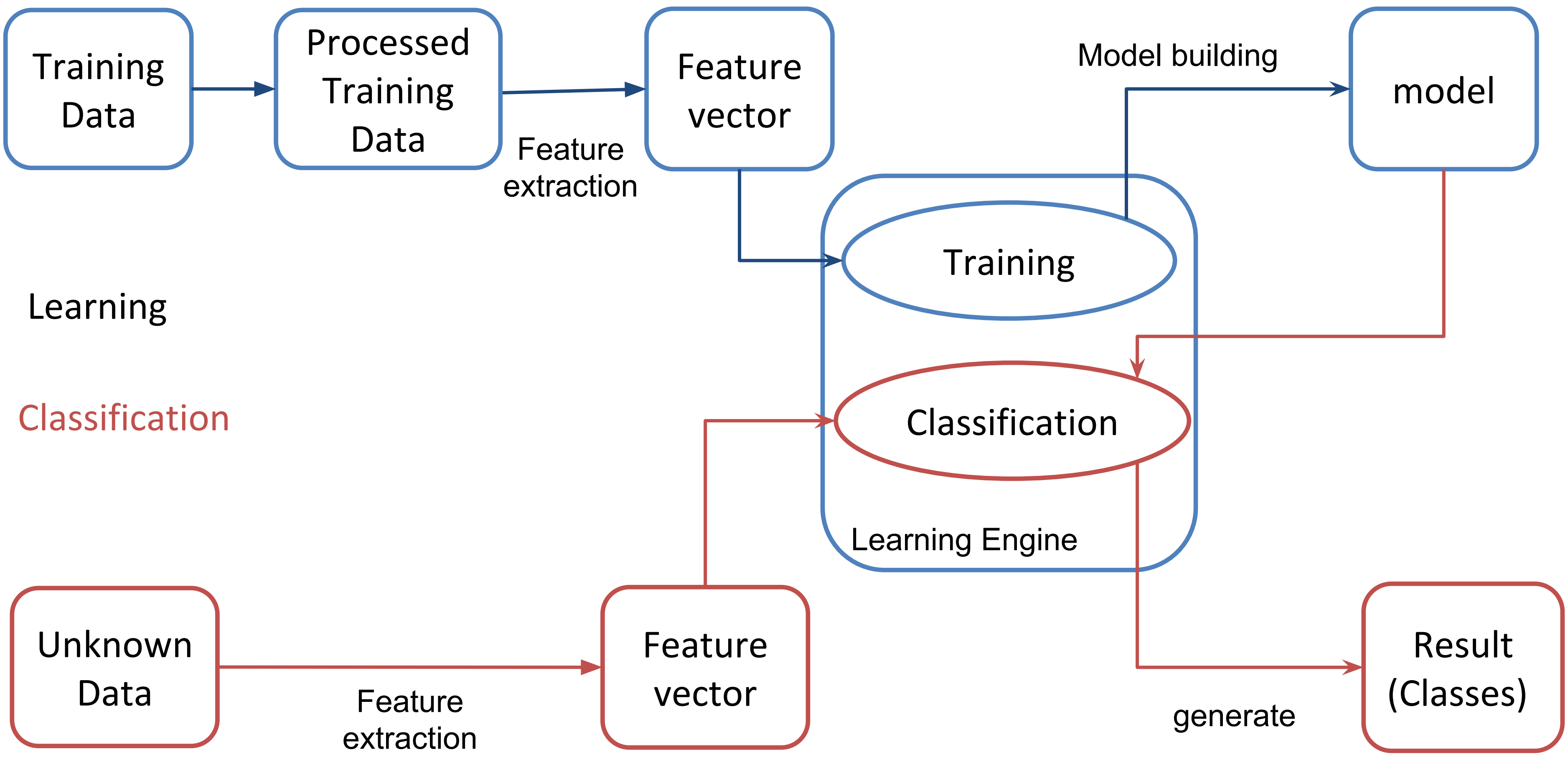


Bad feature selection

2.Background : Random forest classifier

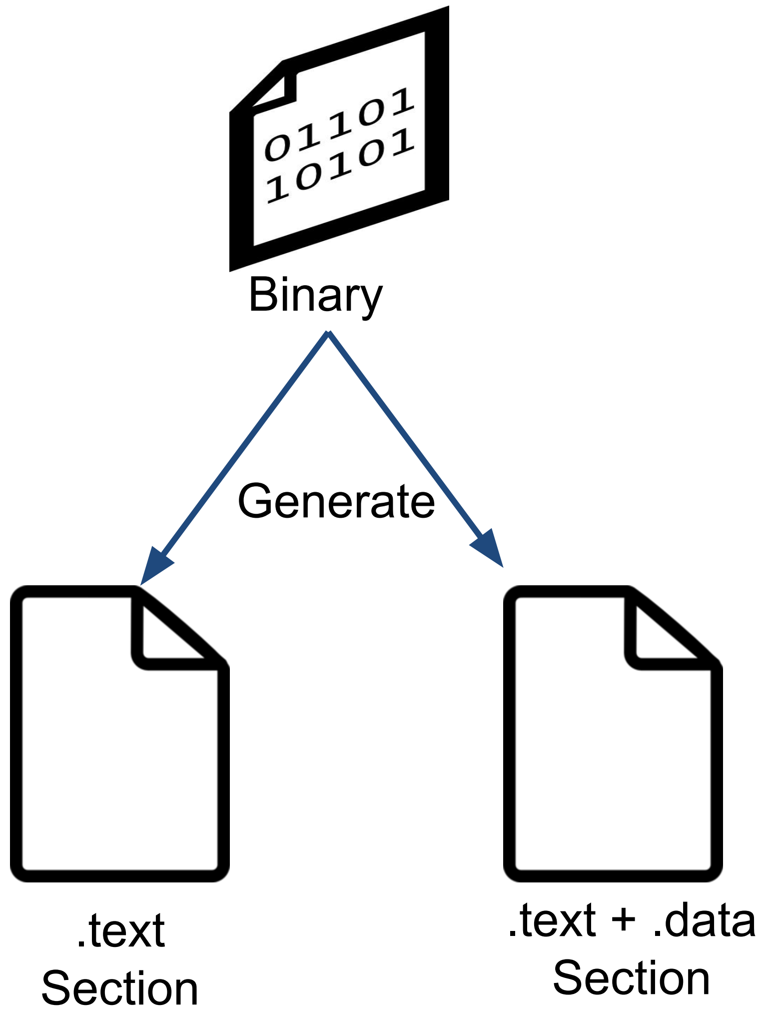


2.Process

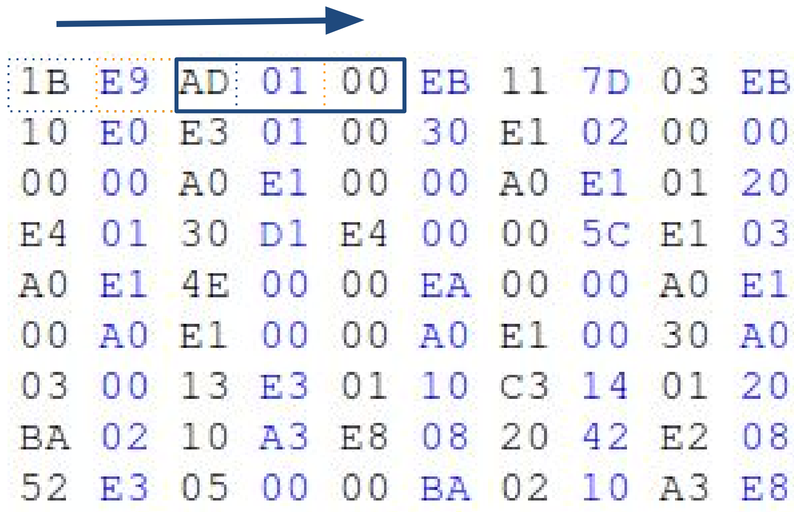


Training Data
Processed Training Data
Feature extraction
Feature vector
Model building
model
Training
Learning
Classification
Classification
Learning Engine
Unknown Data
Feature extraction
Feature vector
generate
Result (Classes)

2.Process : Data Traitement

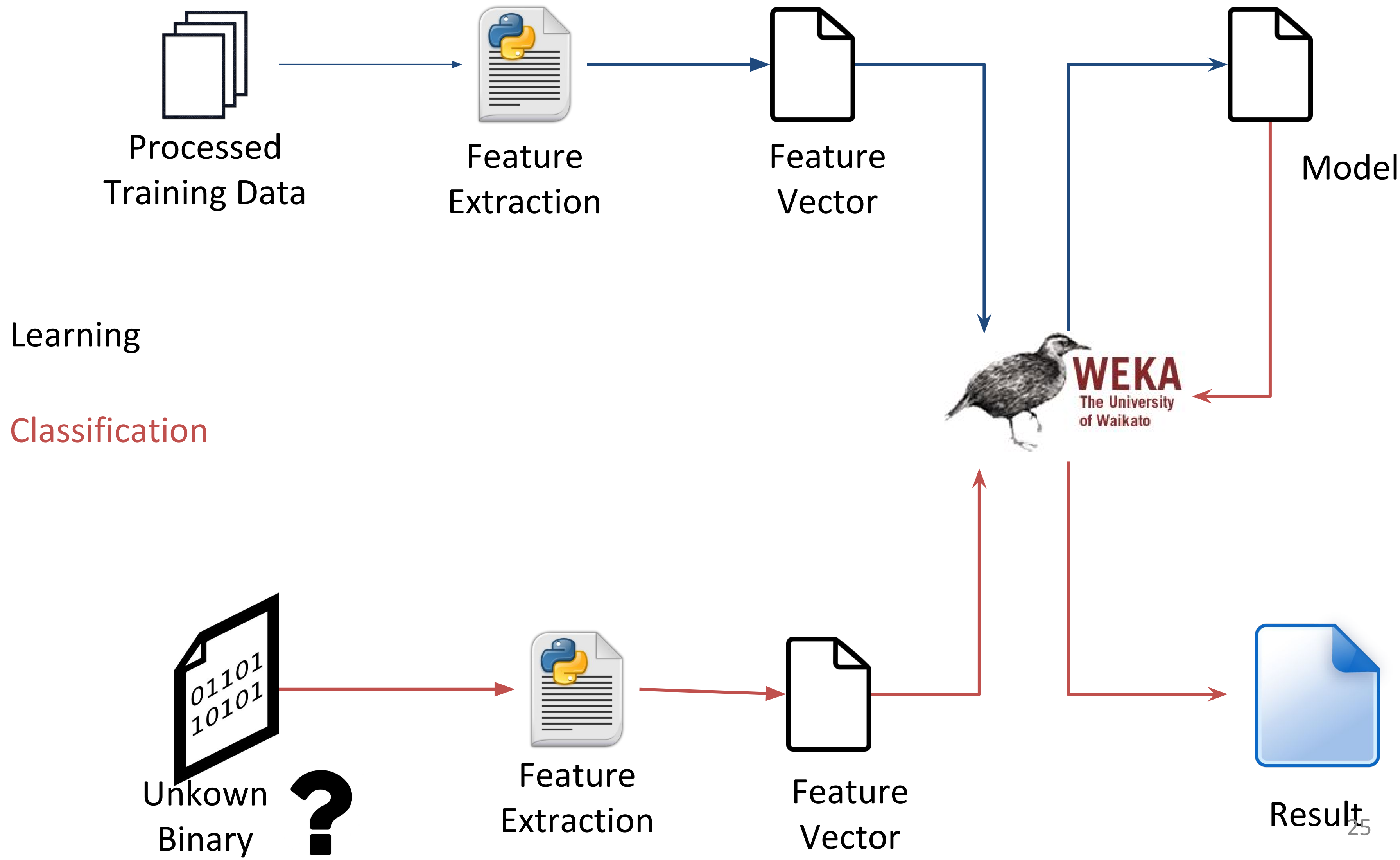


Process : Feature Extraction



- Work on binaries
- Sliding window (3 bytes)

2.Implementation



2.Result

=== Summary ===

Correctly Classified Instances	4726	91.9634 %
Incorrectly Classified Instances	413	8.0366 %

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	<-- classified as
357	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	3	a = ELF_ALPHA
0	0	31	0	1	0	0	140	0	0	0	0	0	0	0	0	0	149	b = ELF_AMD64
0	0	367	0	1	0	0	3	0	0	0	0	0	0	0	0	0	2	c = ELF_ARM
0	4	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	d = ELF_ARM64
0	0	7	0	289	0	0	1	0	0	0	0	0	0	0	0	0	0	e = ELF_ARMEL
0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	f = ELF_ARMHF
0	0	0	0	0	0	345	3	0	0	0	0	0	0	0	0	0	1	g = ELF_HPPA
0	0	0	0	0	0	0	358	0	0	0	0	0	0	0	0	0	0	h = ELF_IA64
0	0	1	0	0	0	0	8	789	0	0	0	0	0	1	0	0	0	i = ELF_M68K
0	0	0	0	0	0	0	2	1	359	0	1	0	0	0	0	0	0	j = ELF_MIPS
0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	k = ELF_MIPS64EL
0	0	1	0	1	0	0	3	1	0	0	353	0	0	0	0	0	0	l = ELF_MIPSEL
0	0	0	0	0	0	0	3	0	0	0	0	378	0	0	0	0	0	m = ELF_PP
0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	n = ELF_PPC64EL
0	0	0	0	0	0	0	1	0	0	0	0	1	0	355	0	0	0	o = ELF_S390
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	p = ELF_S390X
0	0	0	0	0	0	0	22	2	0	0	0	0	0	0	0	354	0	q = ELF_SPARC
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	389	r = ELF_X86

2.Results

3 different test benches :

- .text only -> 97,384%
- .text + .data + .rodata -> 91,963%
- Full Binary -> not enough samples to be relevant
    - Good results
    - Tested on given corpus

Conclusion

Approach 1: Statistical Discrimination

- Difficult to find interesting matching points
- Robust on some architectures
- Restricted to Capstone

Approach 2: Machine Learning

- Simple process
- Robust solution
- Easy to extend

Questions ?

Bonus

- Learned new things (Machine Learning, Python :'( )
- Worked in group (not really)
- Discovered Gif sur Yvette (and it's castle)