

# Microsoft Malware Classification Challenge

## 3rd place solution



Mikhail Trofimov, Dmitry Ulyanov, Stanislav Semenov

# Our team

## Mikhail Trofimov



Team Name <small>* in the money</small>	Score <small>?</small>	Entries
say NOOOOO to overfitttting	0.002833228	268
Marios & Gert	0.003240502	80
<b>Mikhail &amp; Dmitry &amp; Stanislav</b>	<b>0.003969846</b>	<b>71</b>
Ivica Jovic	0.004470816	11
Octo Guys	0.005191324	37
Oleksandr Lysenko	0.005335339	51
Silogram	0.006328459	98
SSIR	0.006705905	157
gphilippis	0.006810522	86
Gilberto Titericz Junior	0.006849661	47
clustifier	0.006965226	86
David Shinn	0.007070557	24

## Dmitry Ulyanov



## Stanislav Semenov



# Further plan

- Problem description
- Feature extraction
- Feature processing and selection
- Models
- Tricks

# Further plan

- **Problem description**
- Feature extraction
- Feature processing and selection
- Models
- Tricks

# The data

~ 20000 program executables, each given in two forms:

# The data

~ 20000 program executables, each given in two forms:

## 1. HEX dump “*bytes file*”

1	10001000	6A	FF	68	A3	16	00	10	64	A1	00	00	00	00	50	64	89
2	10001010	25	00	00	00	00	83	EC	20	8B	44	24	34	56	50	8D	4C
3	10001020	24	0C	C7	44	24	08	00	00	00	00	FF	15	34	20	00	10
4	10001030	8B	4C	24	3C	51	8B	C8	C7	44	24	30	01	00	00	00	FF
5	10001040	15	3C	20	00	10	8B	74	24	34	50	8B	CE	FF	15	34	20
6	10001050	00	10	8D	4C	24	08	C7	44	24	04	01	00	00	00	C6	44
7	10001060	24	2C	00	FF	15	30	20	00	10	8B	4C	24	24	8B	C6	5E
8	10001070	64	89	0D	00	00	00	00	83	C4	2C	C3	CC	CC	CC	CC	CC
9	10001080	55	8B	EC	83	E4	F8	81	EC	0C	03	00	00	8B	55	08	85
10	10001090	D2	A1	20	30	00	10	53	56	89	84	24	10	03	00	00	57
11	100010A0	0F	84	1B	02	00	00	8B	5D	0C	85	DB	0F	84	10	02	00
12	100010B0	00	B9	10	00	00	00	BE	B0	20	00	10	8D	BC	24	08	01

# The data

~ 20000 program executables, each given in two forms:

## 1. HEX dump “*bytes file*”

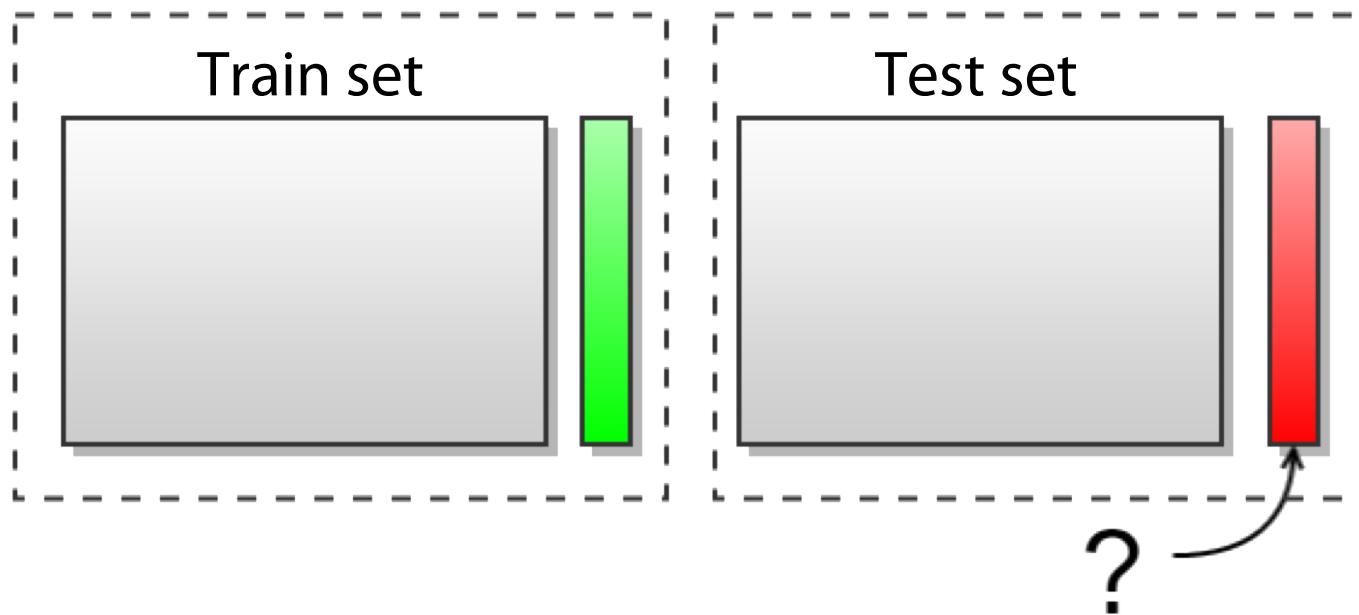
1	10001000	6A	FF	68	A3	16	00	10	64	A1	00	00	00	00	50	64	89
2	10001010	25	00	00	00	00	83	EC	20	8B	44	24	34	56	50	8D	4C
3	10001020	24	0C	C7	44	24	08	00	00	00	00	FF	15	34	20	00	10
4	10001030	8B	4C	24	3C	51	8B	C8	C7	44	24	30	01	00	00	00	FF
5	10001040	15	3C	20	00	10	8B	74	24	34	50	8B	CE	FF	15	34	20
6	10001050	00	10	8D	4C	24	08	C7	44	24	04	01	00	00	00	C6	44
7	10001060	24	2C	00	FF	15	30	20	00	10	8B	4C	24	24	8B	C6	5E
8	10001070	64	89	0D	00	00	00	00	83	C4	2C	C3	CC	CC	CC	CC	CC
9	10001080	55	8B	EC	83	E4	F8	81	EC	0C	03	00	00	8B	55	08	85
10	10001090	D2	A1	20	30	00	10	53	56	89	84	24	10	03	00	00	57
11	100010A0	0F	84	1B	02	00	00	8B	5D	0C	85	DB	0F	84	10	02	00
12	100010B0	00	B9	10	00	00	00	BE	B0	20	00	10	8D	BC	24	08	01

## 2. IDA disassembly

```
421 .text:100013DD ; -----
422 .text:100013DD
423 .text:100013DD ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fo
424 .text:100013DD public DllEntryPoint
425 .text:100013DD
426 .text:100013DD 83 EC 04 DllEntryPoint:
427 .text:100013E0 60 sub esp, 4
428 .text:100013E1 BB 10 00 00 00 pusha
429 .text:100013E6 8D 4C 5B 10 mov ebx, 10h
430 .text:100013EA 51 lea ecx, [ebx+ebx*2+10h]
431 .text:100013EB B9 01 00 00 00 push ecx
432 .text:100013F0 C1 E1 0C mov ecx, 1
433 .text:100013F3 51 shl ecx, 0Ch
434 .text:100013F4 BF 00 04 00 00 push ecx
435 .text:100013F9 81 C7 00 04 00 00 mov edi, 400h
436 .text:100013FF 57 add edi, 400h
437 .text:10001400 6A 00 push edi
438 .text:10001402 FF 15 0C 20 00 10 push 0
439 .text:10001402 FF 15 0C 20 00 10 call ds:VirtualAlloc
```

# The task

## Classify malware into 9 families



- Train: 10k examples
- Test: 10k examples

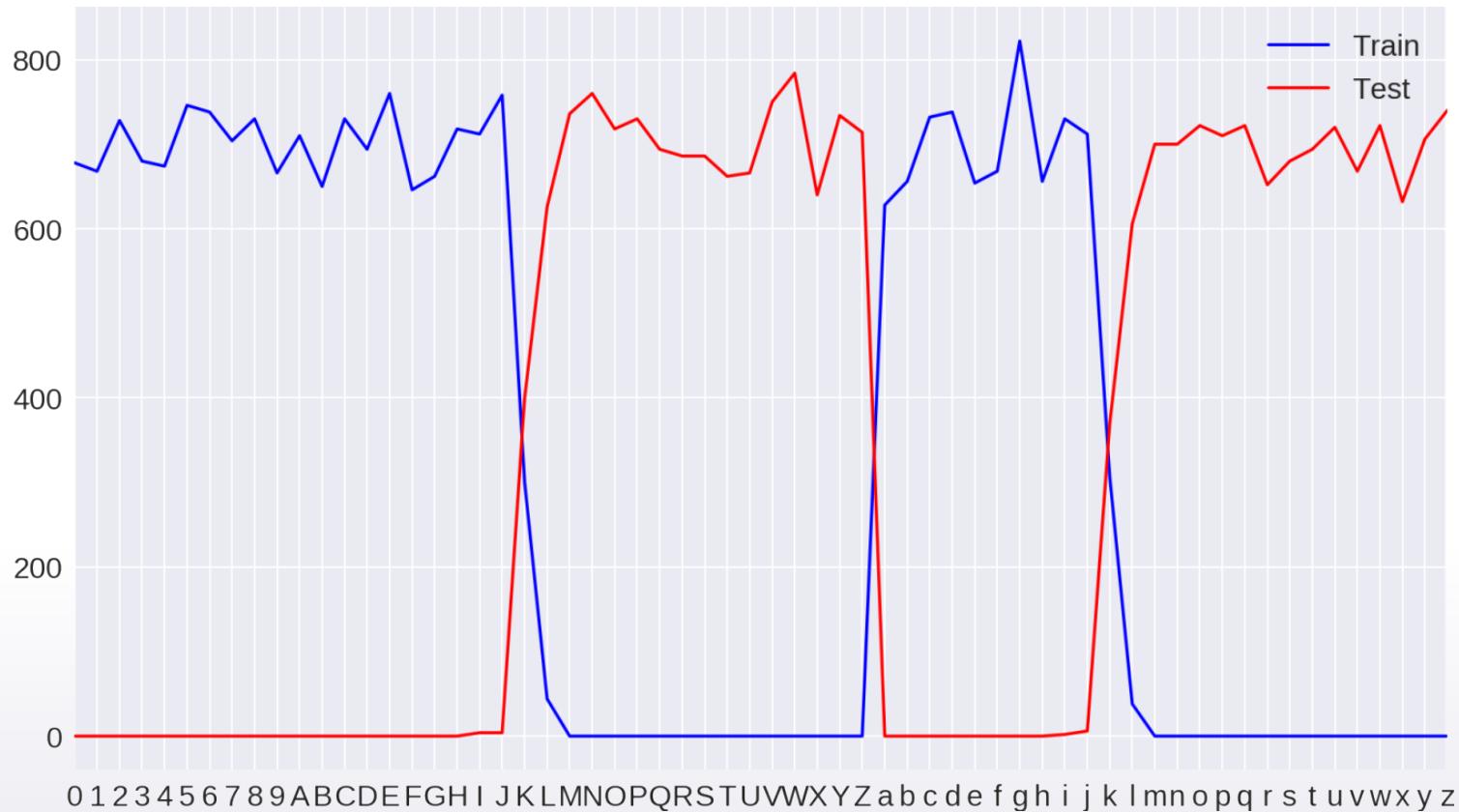
# Evaluation metric

$$\text{LogLoss} = -\frac{1}{N} \sum_{i=1}^L \sum_{j=1}^N y_{ij} \log(\hat{y}_{ij})$$

- **Final LogLoss ~ 0.003**
  - Extremely low for 9-class task
  - Accuracy > 99.7 %

# Quick check for leaks

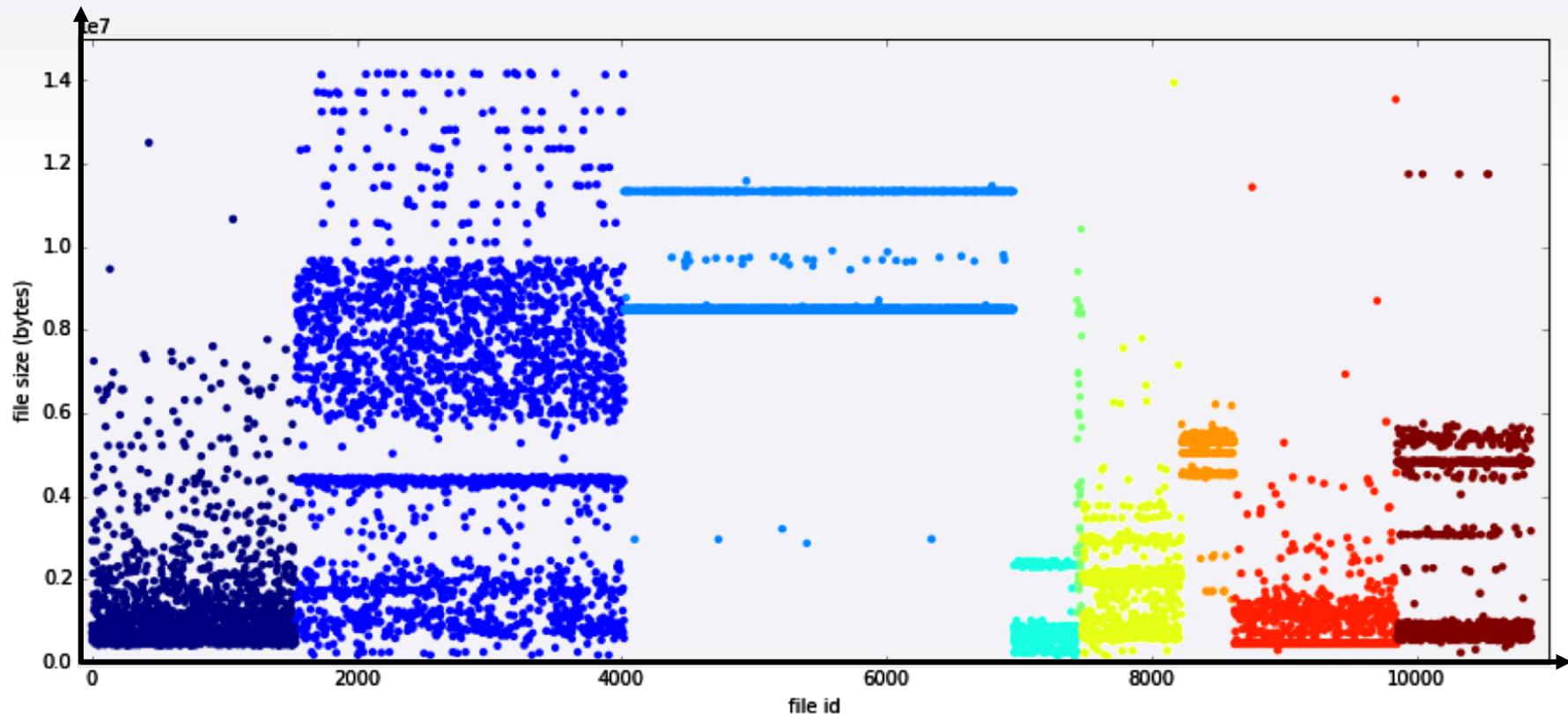
- **Example file names:**
  - 2v819FwJZg4G7MCNB6r
  - lyRMjuo5028fa70EsNBP



# Further plan

- Problem description
- **Feature extraction**
- Feature processing and selection
- Models
- Tricks

# Data exploration: file sizes



Feature set	Error rate, %	Logloss (10-fold CV)
File sizes	12	0.391

# Data exploration: bytes counts

1	10001000	6A	FF	68	A3	16	00	10	64	A1	00	00	00	00	50	64	89
2	10001010	25	00	00	00	00	83	EC	20	8B	44	24	34	56	50	8D	4C
3	10001020	24	0C	C7	44	24	08	00	00	00	00	FF	15	34	20	00	10
4	10001030	8B	4C	24	3C	51	8B	C8	C7	44	24	30	01	00	00	00	FF
5	10001040	15	3C	20	00	10	8B	74	24	34	50	8B	CE	FF	15	34	20
6	10001050	00	10	8D	4C	24	08	C7	44	24	04	01	00	00	00	C6	44
7	10001060	24	2C	00	FF	15	30	20	00	10	8B	4C	24	24	8B	C6	5E
8	10001070	54	89	00	00	00	00	00	83	C4	2C	C3	CC	CC	CC	CC	CC
9	10001080	55	8B	EC	83	E4	F8	81	EC	0C	03	00	00	8B	55	08	85
10	10001090	D2	A1	20	30	00	10	53	56	89	84	24	10	03	00	00	57
11	100010A0	0F	84	1B	02	00	00	8B	5D	0C	85	DB	0F	84	10	02	00
12	100010B0	00	B9	10	00	00	00	BE	B0	20	00	10	8D	BC	24	08	01

Feature set	Error rate, %	Logloss (10-fold CV)
Single bytes counts	1.3	0.044

# Data exploration: sys calls

```
421 .text:100013DD ; -----  
422 .text:100013DD ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fo  
423 .text:100013DD public DllEntryPoint  
424 .text:100013DD DllEntryPoint: sys call  
425 .text:100013DD sub esp, 4  
426 .text:100013DD 83 EC 04  
427 .text:100013E0 60  
428 .text:100013E1 BB 10 00 00 00  
429 .text:100013E6 8D 4C 5B 10  
430 .text:100013EA 51  
431 .text:100013EB B9 01 00 00 00  
432 .text:100013F0 C1 E1 0C  
433 .text:100013F3 51  
434 .text:100013F4 BF 00 04 00 00  
435 .text:100013F9 81 C7 00 04 00 00  
436 .text:100013FF 57  
437 .text:10001400 6A 00  
438 .text:10001402 FF 15 0C 20 00 10
```

Feature set	Error rate, %	Logloss (10-fold CV)
System calls	2.3	0.078

# Data exploration: asm operators

```
421 .text:100013DD ; -----  
422 .text:100013DD ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fo  
423 .text:100013DD ; public DllEntryPoint  
424 .text:100013DD  
425 .text:100013DD  
426 .text:100013DD 83 EC 04  
427 .text:100013E0 60  
428 .text:100013E1 BB 10 00 00 00  
429 .text:100013E6 8D 4C 5B 10  
430 .text:100013EA 51  
431 .text:100013EB B9 01 00 00 00  
432 .text:100013F0 C1 E1 0C  
433 .text:100013F3 51  
434 .text:100013F4 BF 00 01 00 00  
435 .text:100013F9 81 C7 00 00 00 00  
436 .text:100013FF 57  
437 .text:10001400 6A 00  
438 .text:10001402 FF 15 0C 20 00 10  
asm operators ;-----  
;-----  
DllEntryPoint:  
    sub    esp, 4  
    pusha  
    mov    ebx, 10h  
    lea    ecx, [ebx+ebx*2+10h]  
    push   ecx  
    mov    ecx, 1  
    shl    ecx, 0Ch  
    push   ecx  
    mov    edi, 400h  
    add    edi, 400h  
    push   edi  
    push   0  
    call   ds:VirtualAlloc
```

Feature set	Error rate, %	Logloss (10-fold CV)
Asm operators	1.5	0.055

# Data exploration: asm operators

```
421 .text:100013DD ; -----  
422 .text:100013DD ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fo  
423 .text:100013DD ; public DllEntryPoint  
424 .text:100013DD  
425 .text:100013DD  
426 .text:100013DD 83 EC 04  
427 .text:100013E0 60  
428 .text:100013E1 BB 10 00 00 00  
429 .text:100013E6 8D 4C 5B 10  
430 .text:100013F4 E5  
431 .text:100013F5 39 C1 00 00 00  
432 .text:100013F0 C1 E1 0C  
433 .text:100013F3 51  
434 .text:100013F4 BF 00 04 00 00  
435 .text:100013F9 81 C7 00 04 00 00  
436 .text:100013FF 57  
437 .text:10001400 6A 00  
438 .text:10001402 FF 15 0C 20 00 10  
section  
DllEntryPoint:  
    sub    esp, 4  
    pusha  
    mov    ebx, 10h  
    lea    ecx, [ebx+ebx*2+10h]  
    push   ecx  
    mov    ecx, 1  
    shl    ecx, 0Ch  
    push   ecx  
    mov    edi, 400h  
    add    edi, 400h  
    push   edi  
    push   0  
    call   ds:VirtualAlloc
```

Feature set	Error rate, %	Logloss (10-fold CV)
Sections distribution	0.8	0.027

# First results

Feature set	Error rate, %	Logloss (10-fold CV)
File sizes	12	0.391
System calls	2.3	0.078
Asm operators	1.5	0.055
Single bytes counts	1.3	0.044
Sections distribution	0.8	0.027
<b>All together</b>	<b>0.46</b>	<b>0.014</b>

# Features: n-grams

1	10001000	6A	FF	68	A3	16	00	10	64	A1	00	00	00	00	50	64	89
2	10001010	25	00	00	00	00	83	EC	20	8B	44	24	34	56	50	8D	4C
3	10001020	24	0C	C7	44	24	08	00	00	00	00	FF	15	34	20	00	10
4	10001030	8B	4C	24	3C	51	83	C8	C7	44	24	30	01	00	00	00	FF
5	10001040	15	3C	20	00	10	8B	74	24	34	50	8B	CE	FF	15	34	20
6	10001050	00	10	8D	4C	24	08	C7	44	24	04	01	00	00	00	C6	44
7	10001060	24	2C	00	FF	15	30	20	00	10	8B	4C	24	24	8B	C6	5E
8	10001070	64	89	0D	00	00	00	00	83	C4	C5	CC	CC	CC	CC	CC	CC
9	10001080	55	8B	EC	83	E4	F8	81	EC	0C	03	00	00	8B	55	08	85
10	10001090	D2	A1	2e	00	10	53	56	89	84	24	10	03	00	00	57	00
11	100010A0	0F	84	1B	02	00	00	8B	5D	0C	85	DB	0F	84	10	02	00
12	100010B0	00	B9	10	00	00	00	BE	B0	20	00	10	8D	BC	24	08	01

10-gram

4-gram

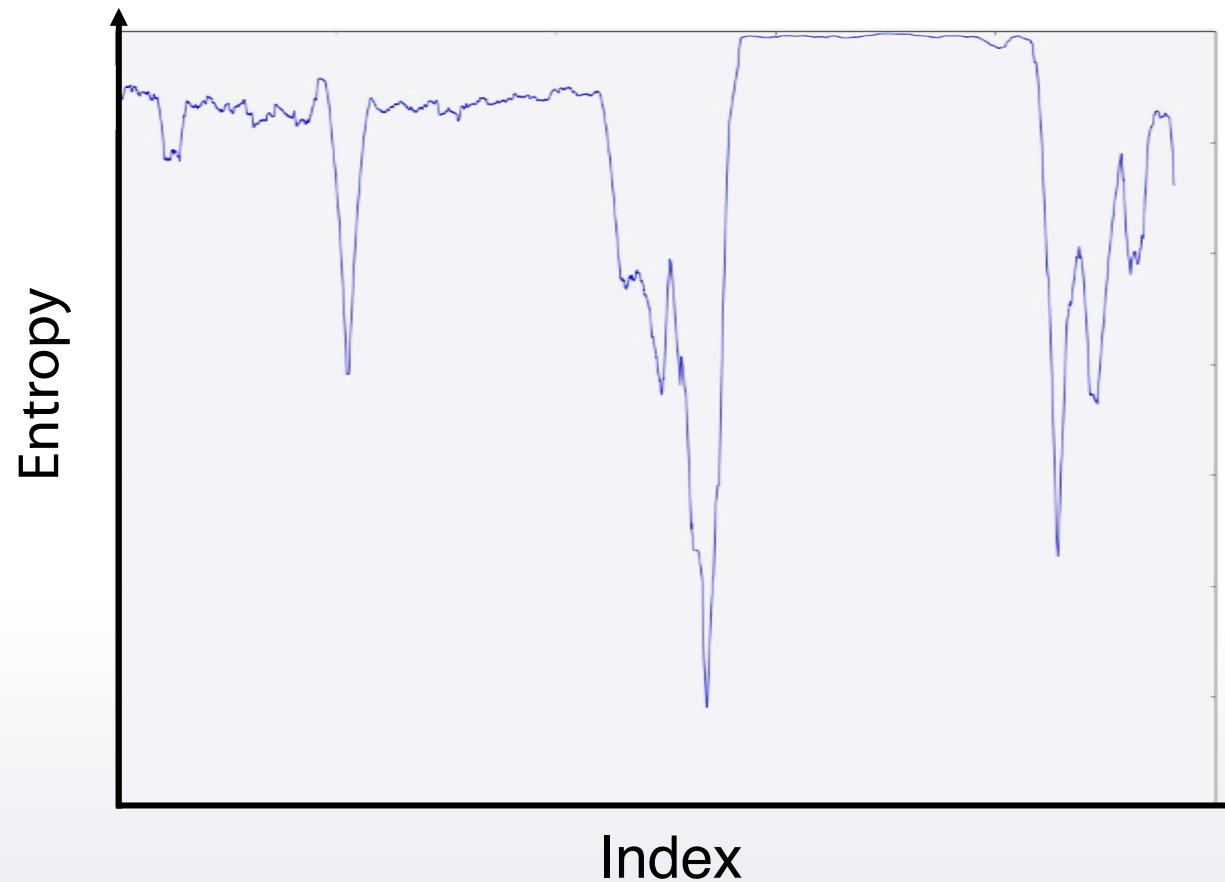
2-gram

# Features: n-grams

n	Possible #features	Selected #features	mlogloss	error rate, %
1	256+1	—	0.044	1.3
2	$(256+1)^2 \sim 65k$	—	0.019	0.6
4	$(256+1)^4 \sim 4.3*10^9$	131	0.016	0.48
10	$(256+1)^{10} \sim 1.2*10^{24}$ <b>hashed into <math>2^{28} = 2.6*10^8</math></b>	831	<b>0.011</b>	<b>0.32</b>

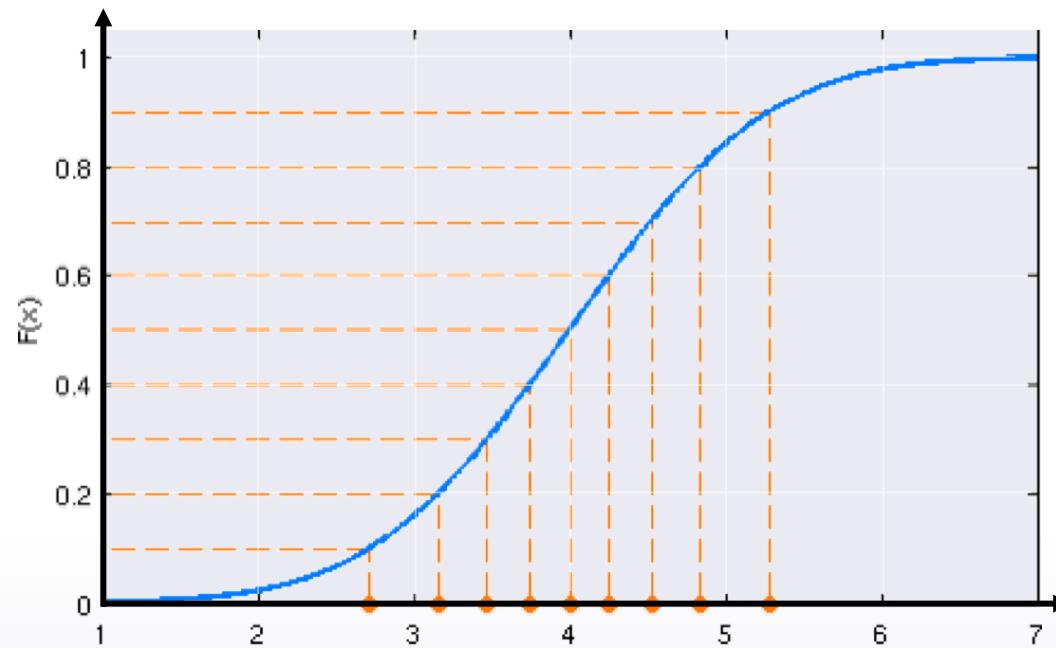
# Entropy

- **Entropy extraction**
  - Compute entropy for a sliding window over byte sequence



# Entropy

- **Features generation**
  - Mean, median, max, min, etc.
  - *Percentiles, inverse percentiles*



- The same with ***diff(entropy sequence)***

# Strings

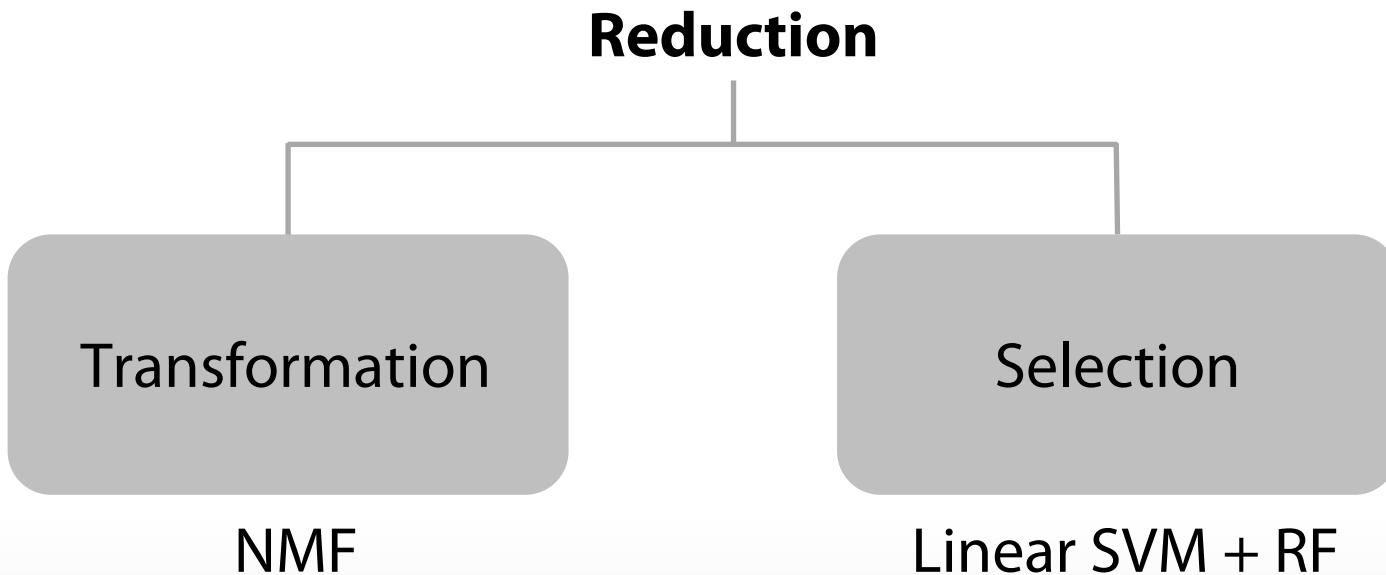
- **Strings extraction**
  - Find printable sequences with “\0” at the end
- **Feature generation**
  - Discard string content
  - Compute strings length distribution in files
  - Quantiles, percentiles, inverse percentiles...

# Further plan

- Problem description
- Feature extraction
- **Feature processing and selection**
- Models
- Tricks

# Dimensionality reduction

A lot of noisy features → Need processing



# Feature transformations: NMF vs PCA

- **Matrix factorizations:**

- Non-negative matrix factorization (NMF)
- Principal component analysis (PCA)

$$\begin{matrix} & 1 & \cdots & n & \cdots & N \\ \begin{matrix} 1 \\ \vdots \\ t \\ \vdots \\ T \end{matrix} & \boxed{\mathbf{X}} & \approx & \begin{matrix} 1 & \cdots & p & \cdots & P \\ T & \boxed{\mathbf{S}} & \times & \begin{matrix} 1 & & & N \\ P & \boxed{\mathbf{A}} & & \end{matrix} \end{matrix} \end{matrix}$$

**PCA**

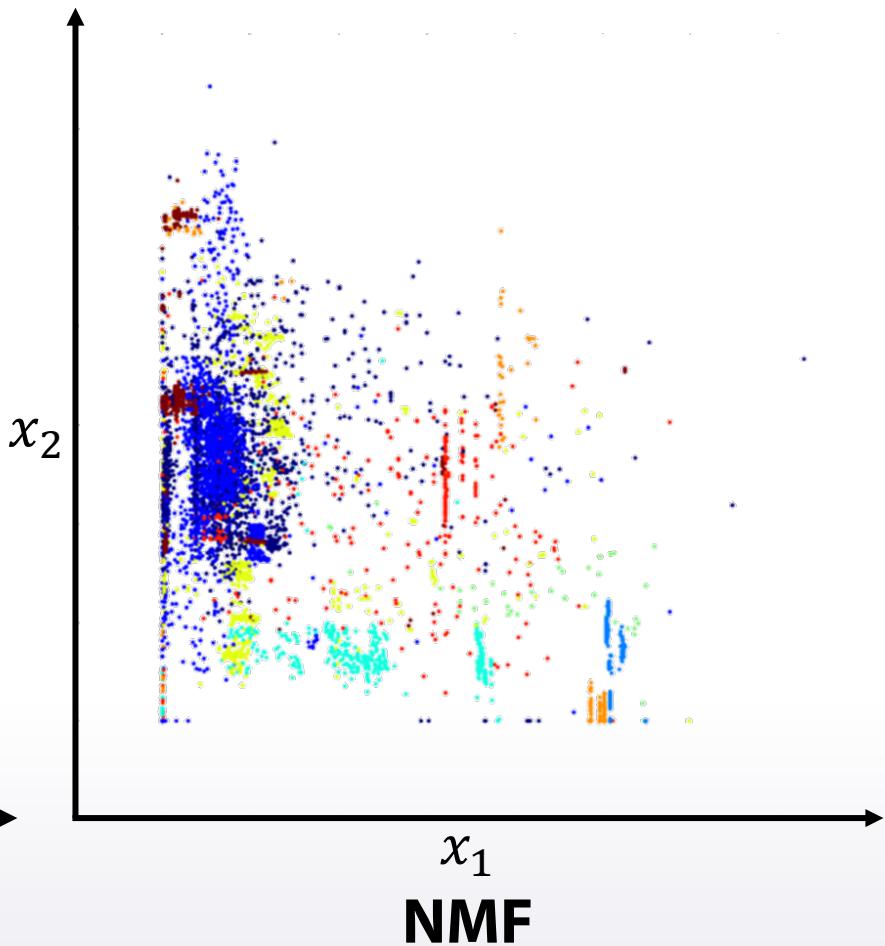
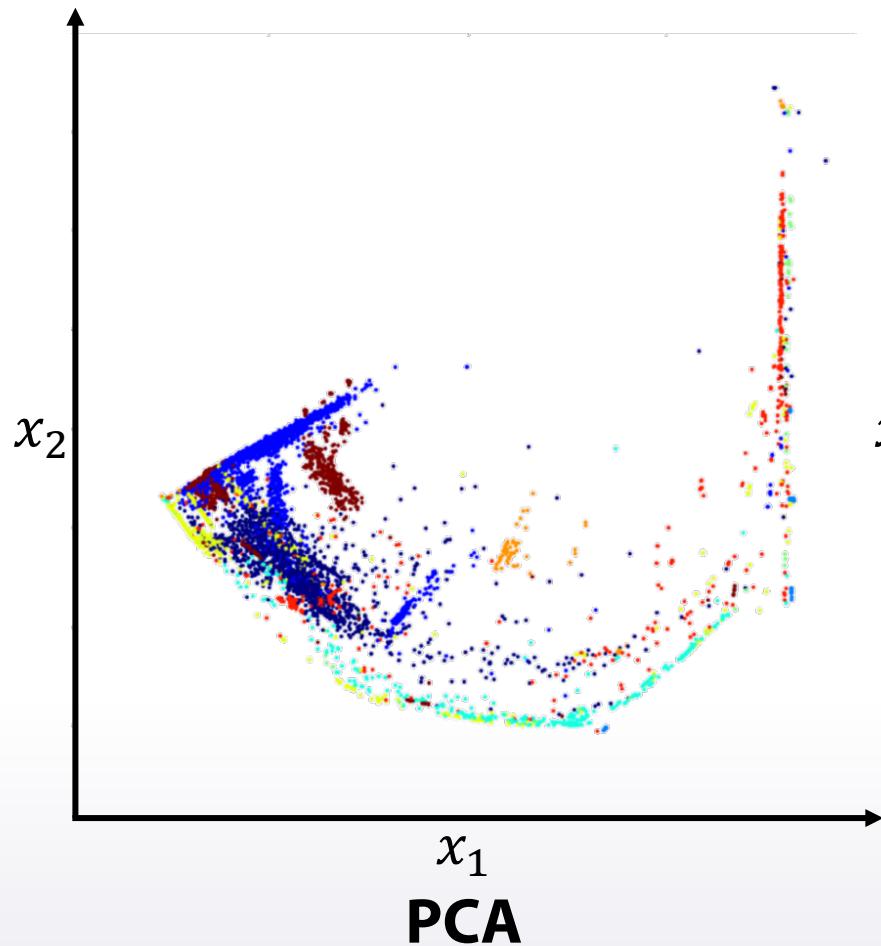
$$\min_{S,A} \|X - SA\|_2$$

**NMF**

$$\begin{cases} \min_{S,A} \|X - SA\|_2 \\ S_{ij} \geq 0 \\ A_{ij} \geq 0 \end{cases}$$

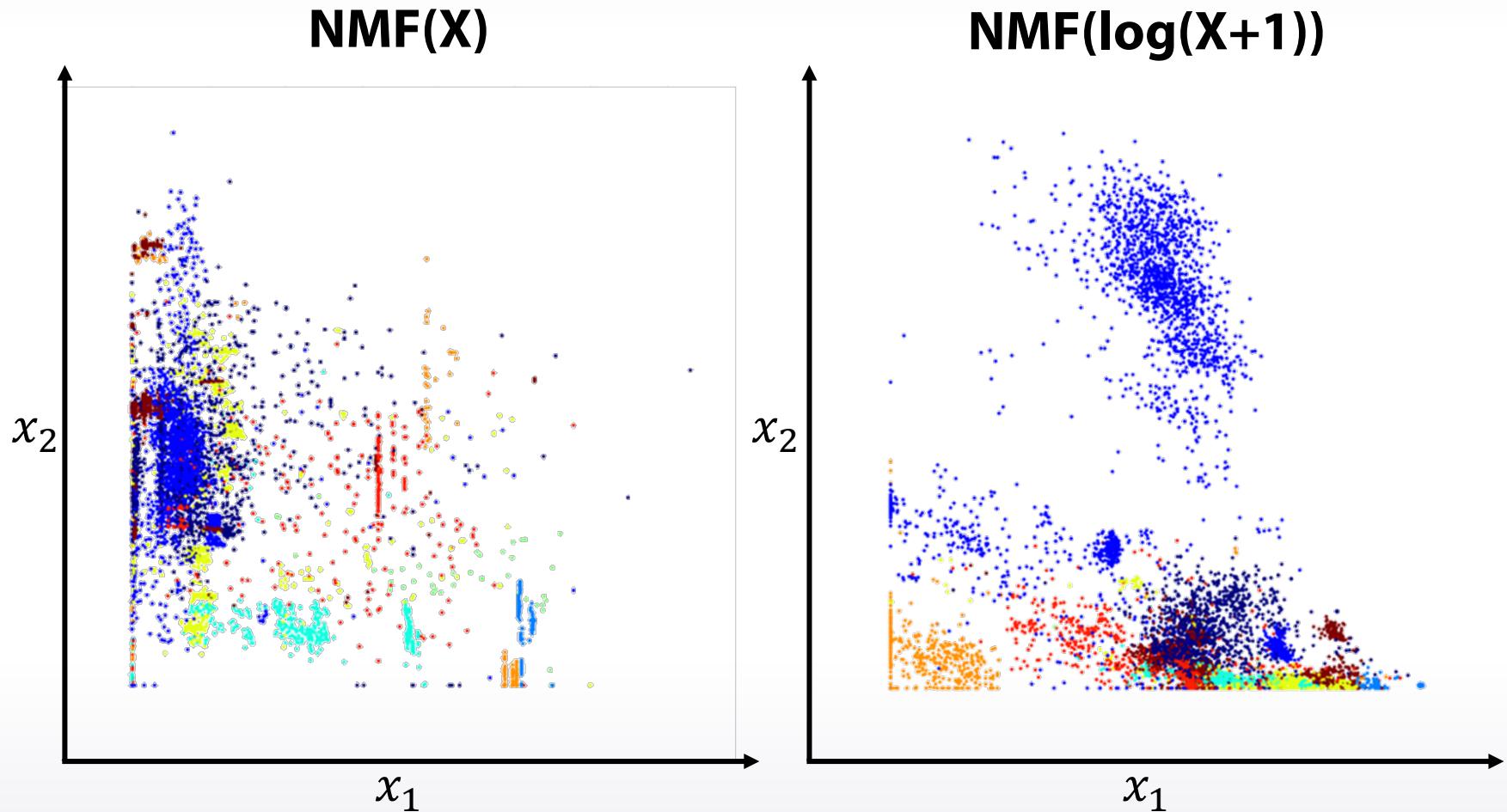
# NMF vs PCA

- NMF is better for using in Random Forest



# A simple way to get different features

- Apply transform to data and run NMF



- Log transform changes objective from MSE to RMSLE

# Feature selection: 4-grams

Stage	Number of features
Original	$(256+1)^4$
Omit rare	4 995 597
Linear SVM + L1-penalty	6369
RF features importances	131



logloss	error rate, %
0.016	0.48

# Feature selection: 10-grams

Stage	Number of features
Original	$(256+1)^{10}$
Hashing	$2^{28}$
Omit rare	3 438 702
Linear SVM + L1-penalty	10865
RF feature importance	861



mlogloss	error rate, %
0.011	0.32

# Feature selection: 10-grams

- We wanted to find good 10-grams *in addition* to the features X that we had.

# Feature selection: 10-grams

- We wanted to find good 10-grams *in addition* to the features X that we had.

## **Our method:**

1. Fit a classifier on the features X and obtain out of fold predictions for the train set.
2. Sort the objects by their loss and create a new dataset:
  - Assign class 1 to top K error-prone objects
  - Assign class 0 to all the others
3. Fit a model on the new dataset and select best-performing features.

# Further plan

- Problem description
- Feature extraction
- Feature processing and selection
- **Models**
- Tricks

# Model

- **At first: Random Forest**
  - Needs manual logloss calibration
- **Moved to GBDT (XGBoost)**
- **Bagging works well with boosting:**
  - Use  $L$  train objects
  - + sample  $\alpha L$  train objects with replacement
  - We use  $\alpha = 7$
- **Use only use half of features for every tree**
- **Average over ~20 runs**

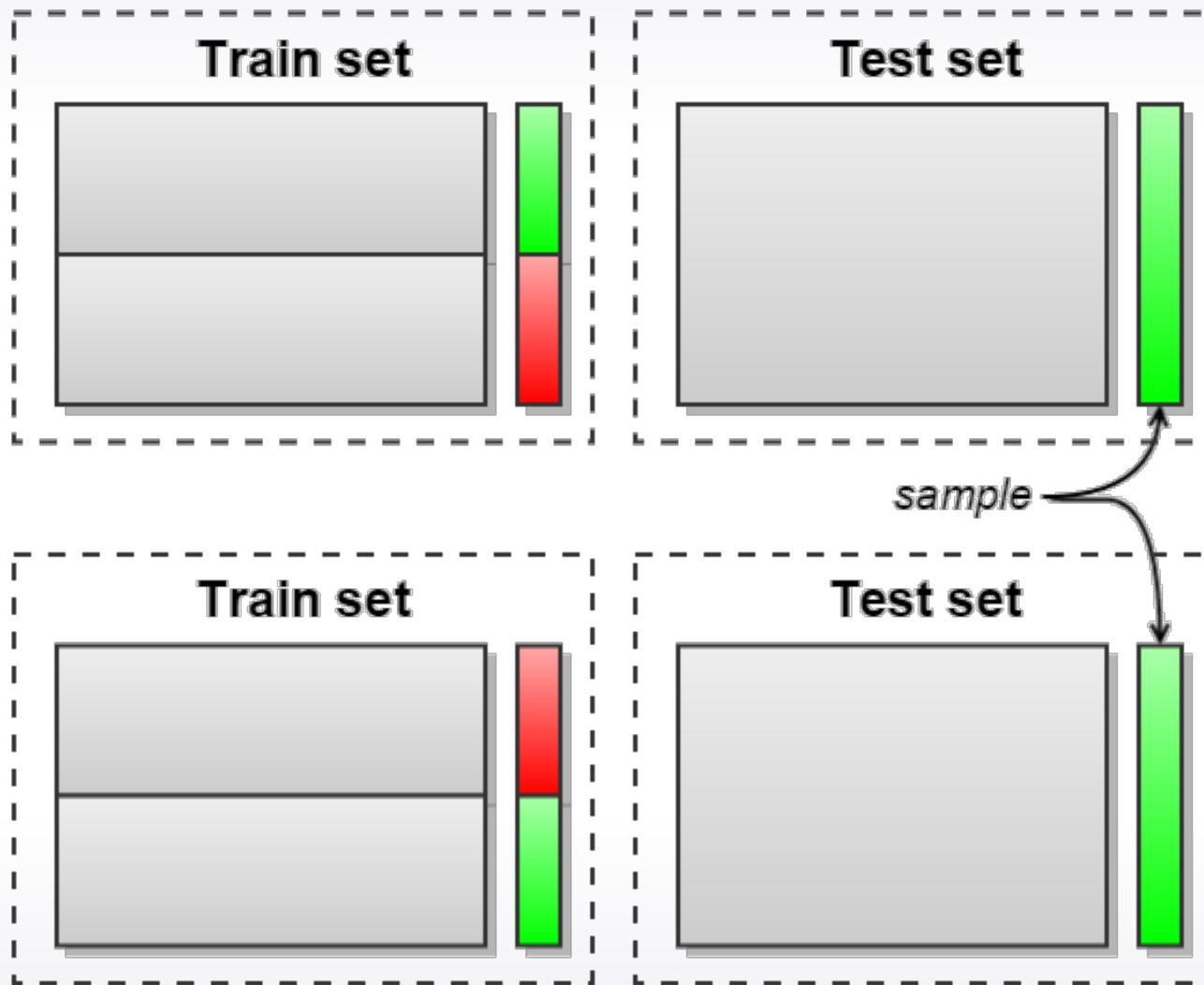
# Further plan

- Problem description
- Feature extraction
- Feature processing and selection
- Models
- **Tricks**

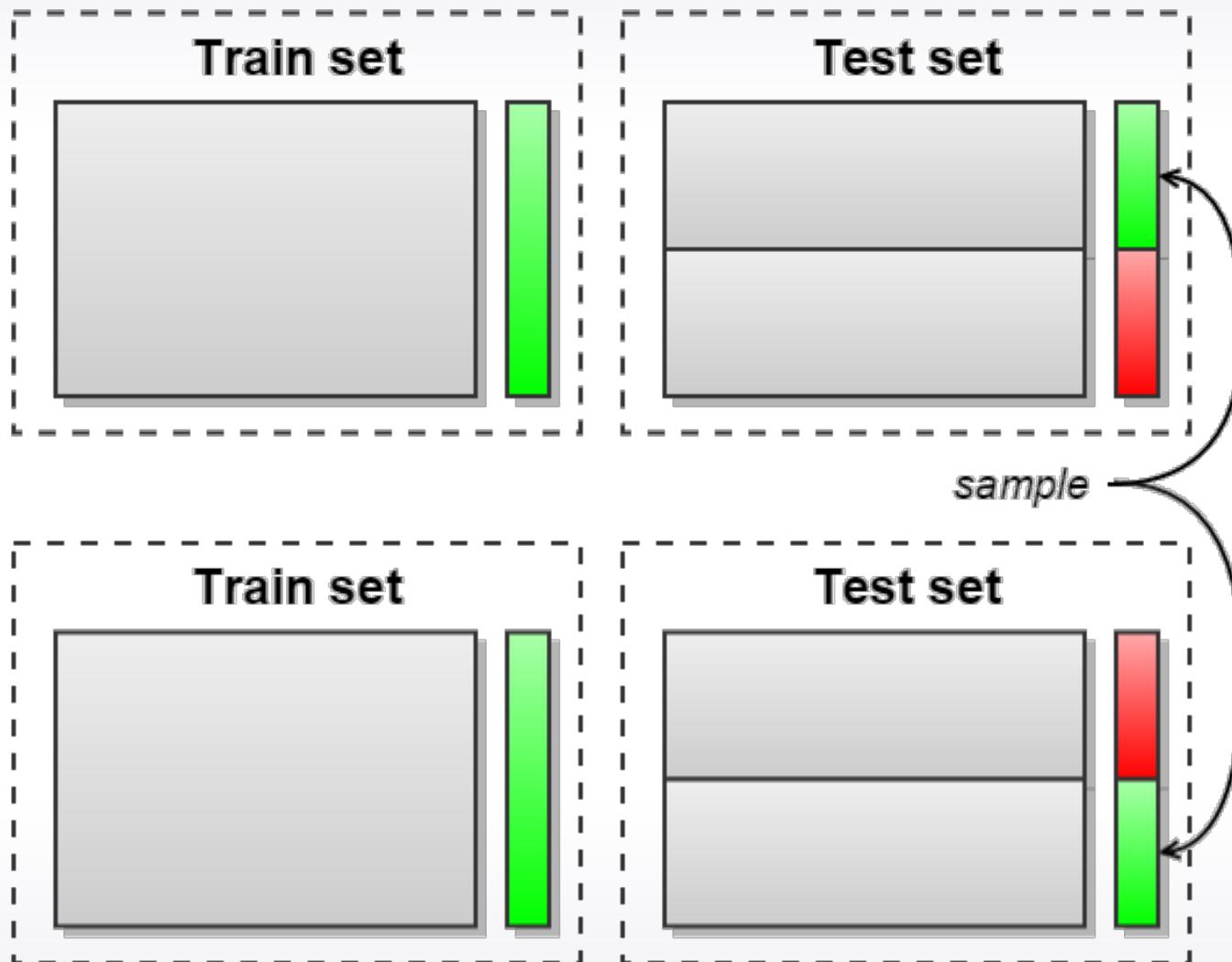
# Pseudo labelling

- CV accuracy is high ( $>0.997$ )
- The more data  the better
  - Use the test data for training
    - Sample labels according to predicted distribution
    - Or just use predicted class

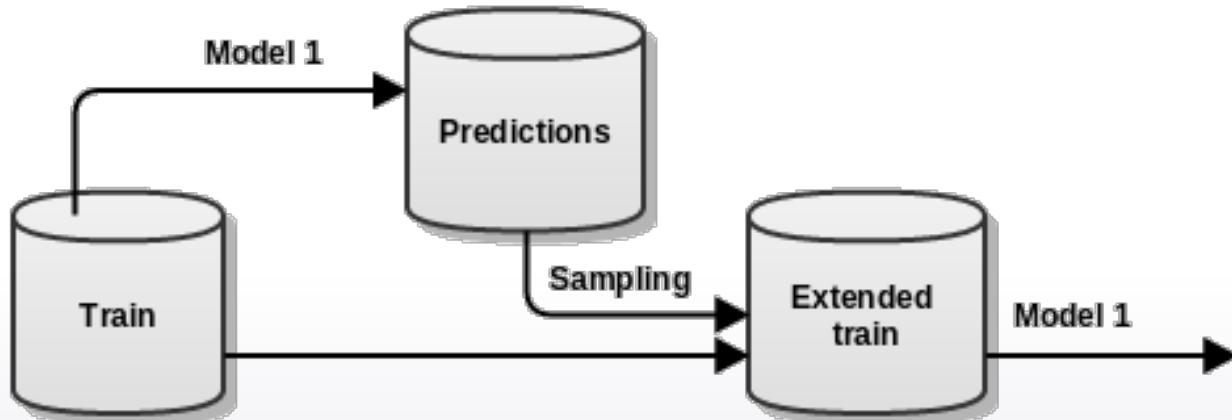
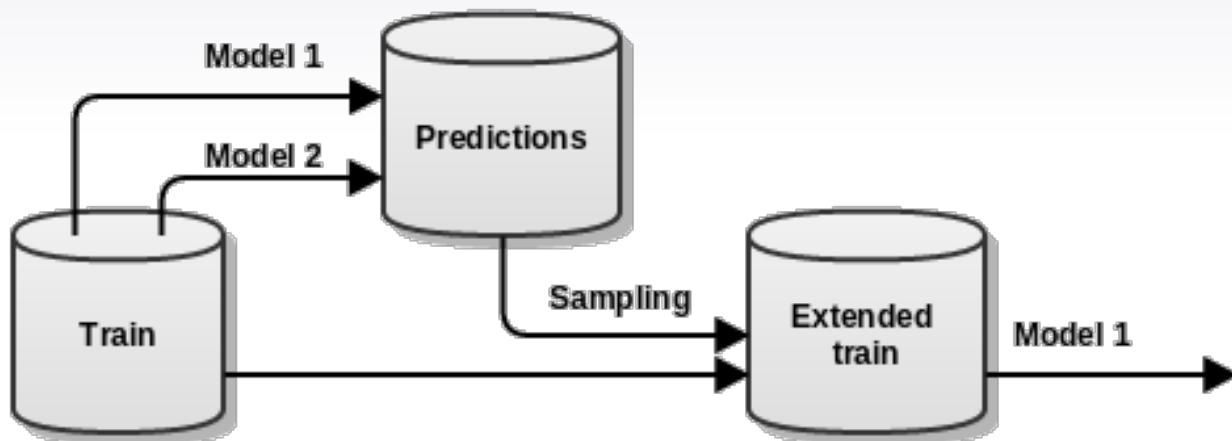
# Pseudo labelling: cross validation



# Pseudo labelling: test prediction



# Pseudo labeling: how To Use



# Tricks: per-class weighting

## Linear mixing

$$y_w = \alpha y_1 + (1 - \alpha) y_2$$

## Per-class linear mixing

$$y_w^i = \alpha^i y_1^i + (1 - \alpha^i) y_2^i$$

# Source code

**The code is available at github:**

<https://github.com/geffy/kaggle-malware>

# Conclusion

- **Technically challenging competition**
  - ~500 GB of data
- **Competitions with raw data are fun!**
  - A lot of room for feature generation
- **Tricks:**
  - Extreme bagging
  - Pseudo labelling
  - Per-class mixing