Need for Speed

NumPy@CAID

Marco Necci
29/05/2020

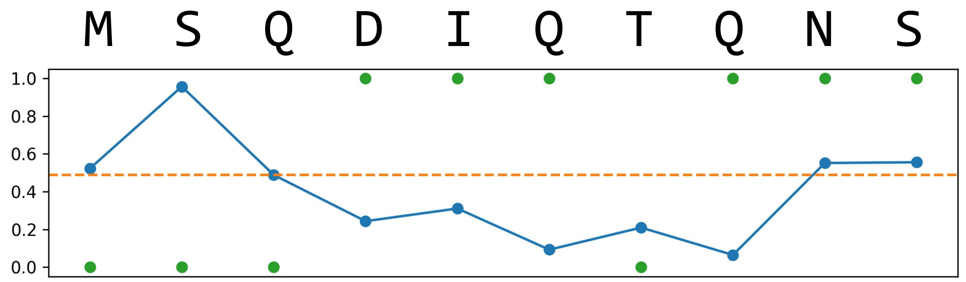
Outline

● Introduction	What is CAID?
● Problem statement	It’s too slow!
● Solution	NumPy
● Results	Now it’s fast

Introduction

What is CAID?

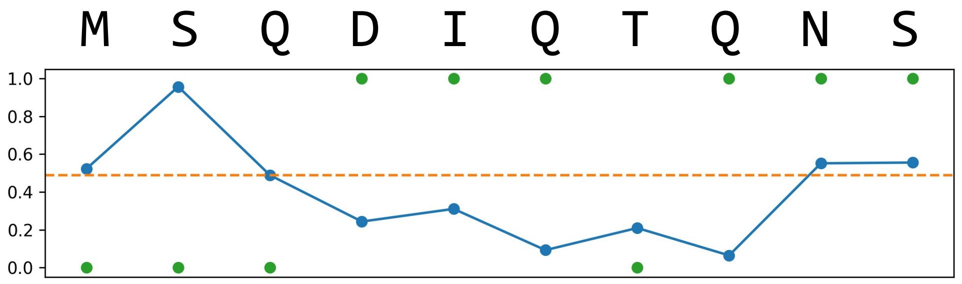
CAID



Amino Acids

- ● True = DisProt
- ● Pred = published predictors

CAID

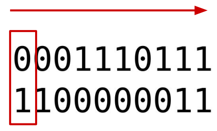


Amino Acids

● True = DisProt

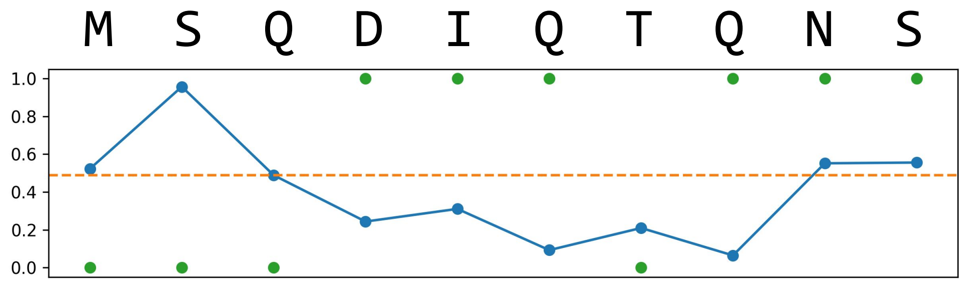
● Pred = published predictors

– Threshold



```
true: 0001110111
pred: 1100000011
```

CAID

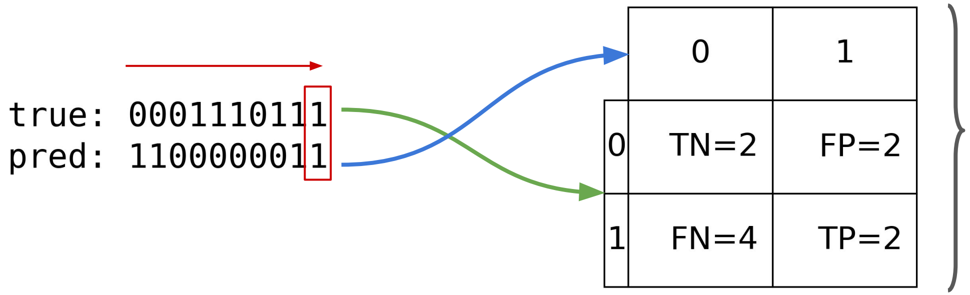


Amino Acids

● True = DisProt

● Pred = published predictors

– Threshold

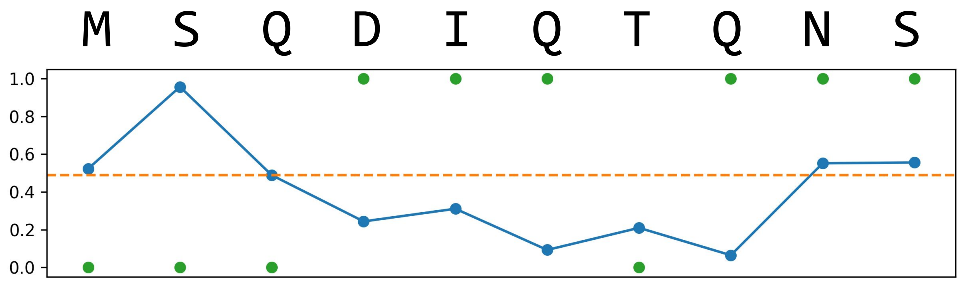


```
true: 0001110111
pred: 1100000011
```

	0	1
0	TN=2	FP=2
1	FN=4	TP=2

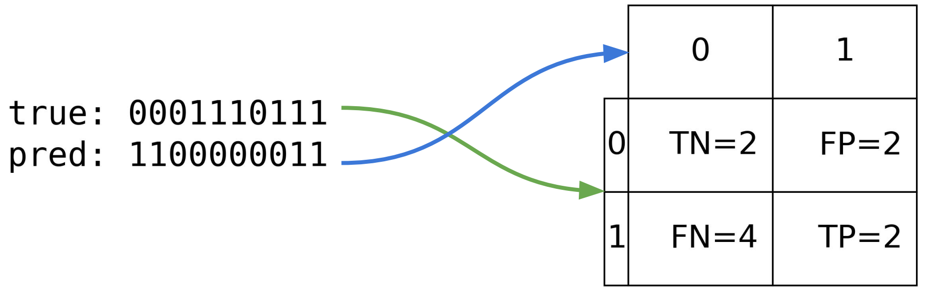
Confusion matrix

CAID



Amino Acids

- ● True = DisProt
- ● Pred = published predictors
- – Threshold



```
true: 0001110111
pred: 1100000011
```

	0	1
0	TN=2	FP=2
1	FN=4	TP=2

Confusion matrix

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2+2}{2+2+4+2} = 0.40$$

Metrics (e.g. accuracy)

Problem Statement

It’s too slow!

Code complexity

Dimension	Elements in dimension	Cumulative product	Order of magnitude	Execution time
Metrics	10	10	10^1	~1 second

Code complexity

Dimension	Elements in dimension	Cumulative product	Order of magnitude	Execution time
Metrics	10	10	10^1	~1 second
Proteins	646	~6,500	10^3	~1 minutes

Code complexity

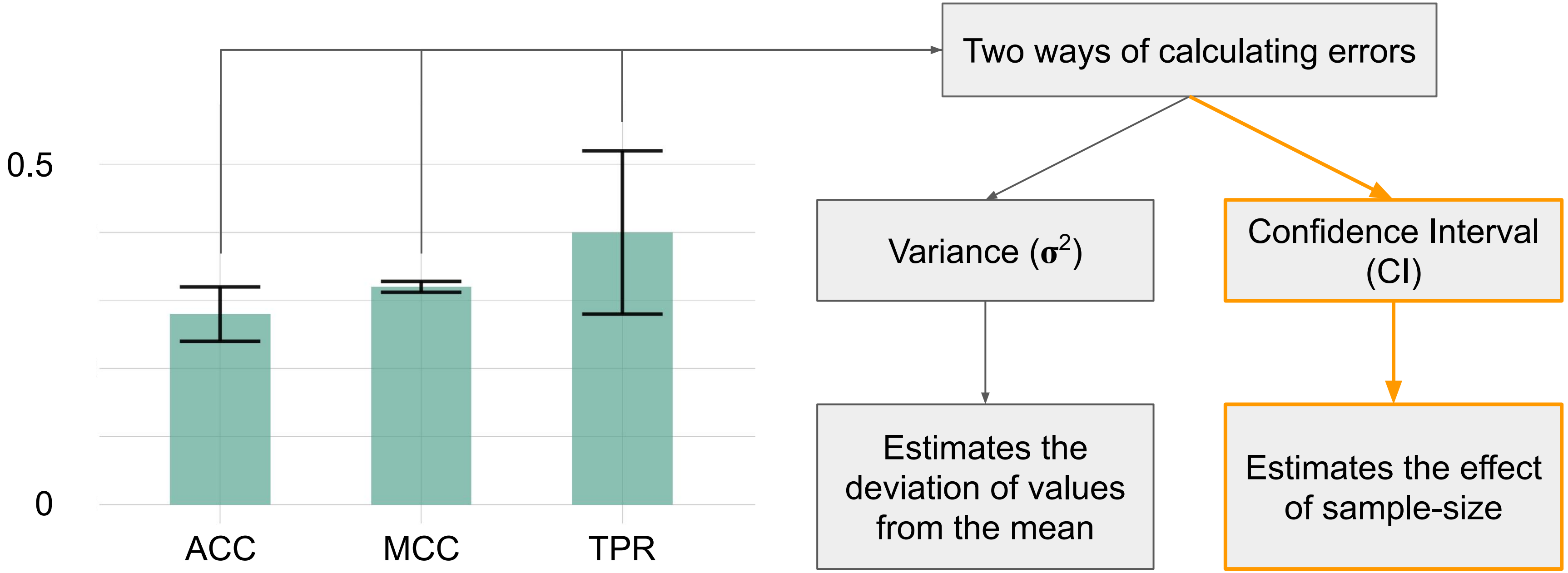
Dimension	Elements in dimension	Cumulative product	Order of magnitude	Execution time
Metrics	10	10	10^1	~1 second
Proteins	646	~6,500	10^3	~1 minutes
Thresholds	5	~32,500	10^4	~20 minutes

Code complexity

Dimension	Elements in dimension	Cumulative product	Order of magnitude	Execution time
Metrics	10	10	10^1	~1 second
Proteins	646	~6,500	10^3	~1 minutes
Thresholds	5	~32,500	10^4	~20 minutes
Predictors	30	~1,000,000	10^6	~1,5 hour

Confidence Intervals

- We want to evaluate the **error** associated to a metric


0.5
0
ACC
MCC
TPR
Two ways of calculating errors
Variance (σ^2)
Confidence Interval (CI)
Estimates the deviation of values from the mean
Estimates the effect of sample-size

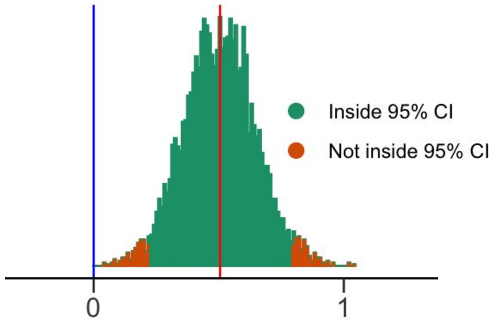
1. Efron, Bradley. "Bootstrap methods: another look at the jackknife." In *Breakthroughs in statistics*, pp. 569-593. Springer, New York, NY, 1992.
2. Pierre-Simon Laplace, 1810

Confidence Intervals

- We want to evaluate the ***error*** associated to a metric

Bootstrap resampling[1] → Confidence Interval (CI)

Creates multiple **resamples** from a set of observations and computes the metric on each of these resamples



Central Limit Theorem[2]

Large number of random samples will approach a normal distribution even if the underlying population is not normally distributed.

1. Efron, Bradley. "Bootstrap methods: another look at the jackknife." In *Breakthroughs in statistics*, pp. 569-593. Springer, New York, NY, 1992.
2. Pierre-Simon Laplace, 1810

Confidence Intervals

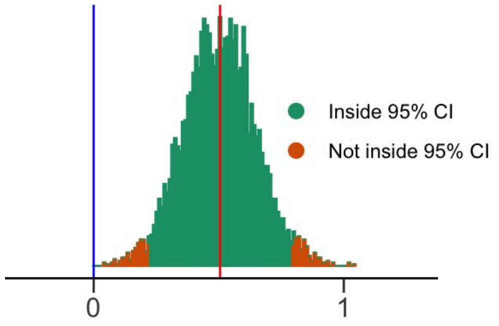
- We want to evaluate the **error** associated to a metric

$$\begin{tikzcd}\boxed{\text{Bootstrap resampling}^{1}} \arrow[r, "\text{Creates multiple resamples from a set of observations and computes the metric on each of these resamples}"] & \boxed{\text{Confidence Interval (CI)}}\end{tikzcd}$$

Usually 1,000

Central Limit Theorem[2]

Large number of random samples will approach a normal distribution even if the underlying population is not normally distributed.



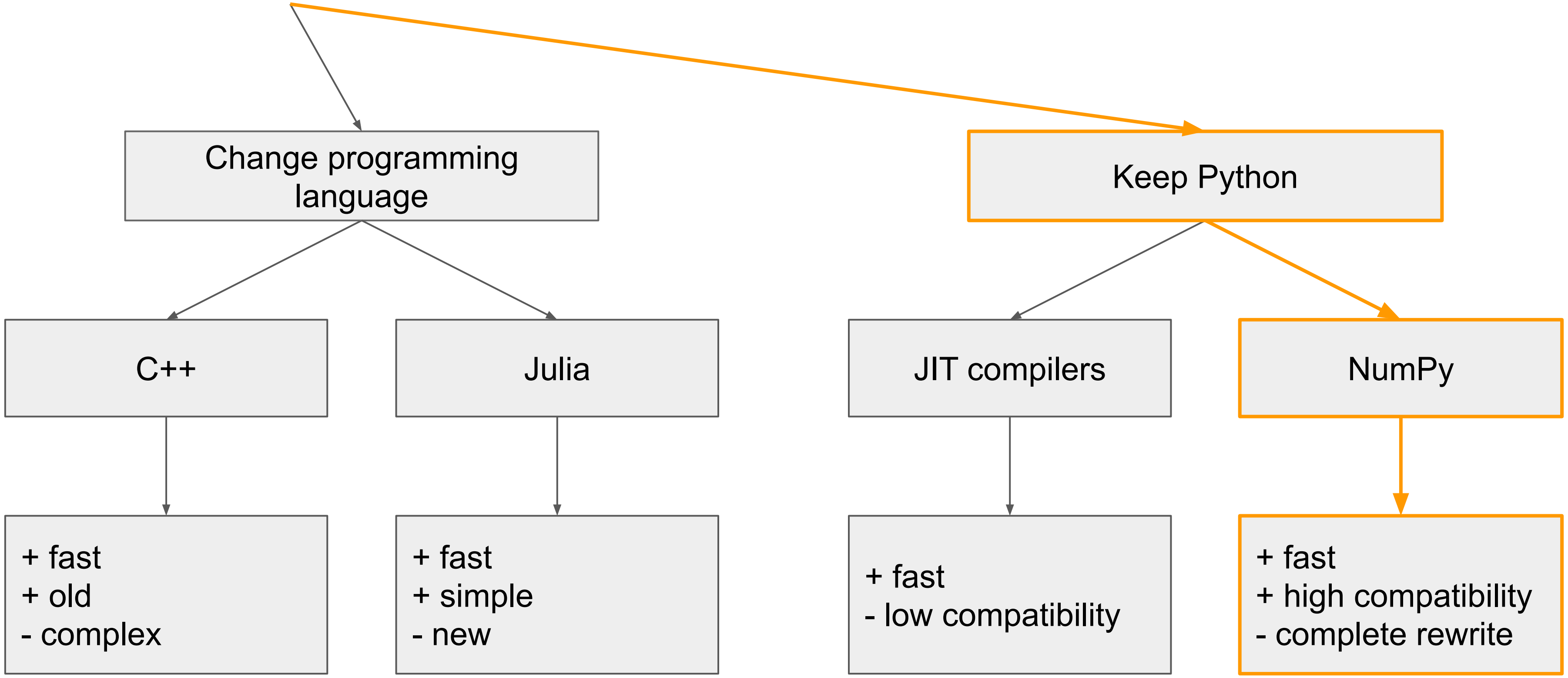
Code complexity

Dimension	Elements in dimension	Cumulative product	Order of magnitude	Execution time
Metrics	10	10	10^1	~1 second
Proteins	646	~6,500	10^3	~1 minutes
Thresholds	5	~32,500	10^4	~20 minutes
Predictors	30	~1,000,000	10^6	~1,5 hours
Bootstrap	1,000	~1,000,000,000	10^9	**~15 hours**

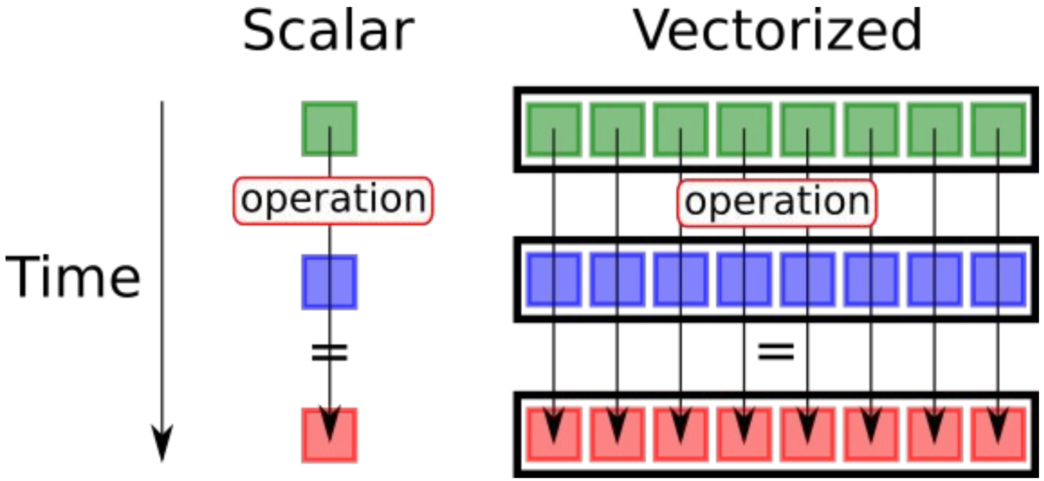
Solution

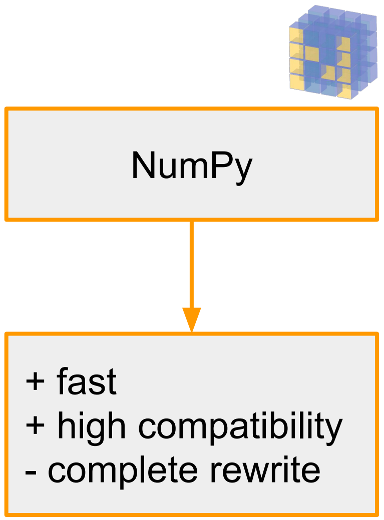
NumPy

Possible solutions



Vectorization





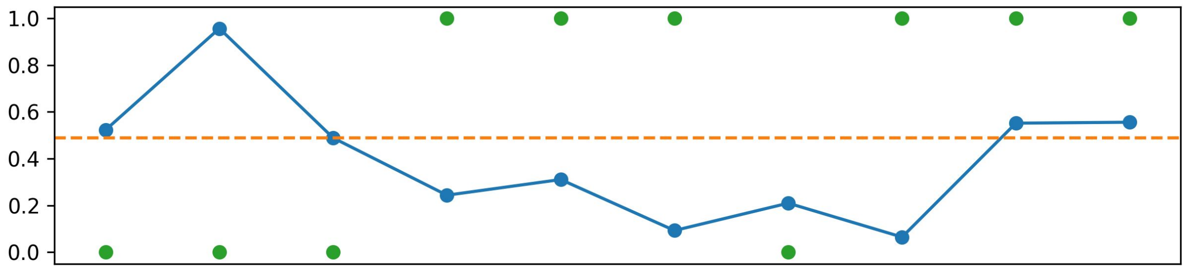
+ fast
+ high compatibility
- complete rewrite

Vectorization

Calculates confusion matrix at all relevant thresholds

```
def binary_clf_curve(y_true, y_score):
    pos_label = 1.0
    y_true = (y_true == pos_label)

    desc_score_indices = np.argsort(y_score, kind="mergesort")[::-1]
    y_score = y_score[desc_score_indices]
    y_true = y_true[desc_score_indices]

    threshold_idxs = np.r_[np.where(np.diff(y_score))[0], y_true.size - 1]

    tps = np.cumsum(y_true, dtype=np.float64)[threshold_idxs]
    fps = 1 + threshold_idxs - tps
    thr = y_score[threshold_idxs]

    return fps, tps, thr
```

Vectorization

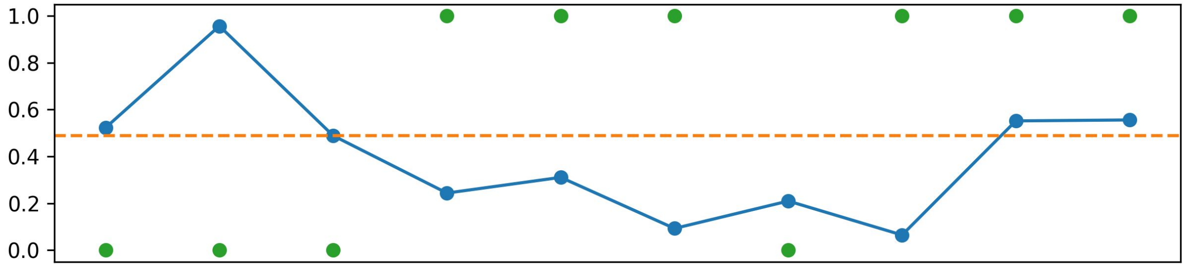


```
array([0,     0,     0,     1,     1,     1,     0,     1,     1,     1])
array([0.523, 0.957, 0.489, 0.244, 0.311, 0.093, 0.210, 0.065, 0.552, 0.556])
```

true

pred

Vectorization



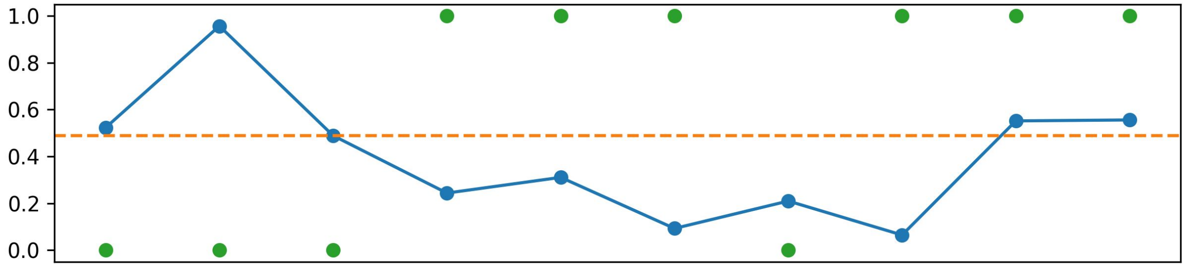
```
array([0,     0,     0,     1,     1,     1,     0,     1,     1,     1])
array([0.523, 0.957, 0.489, 0.244, 0.311, 0.093, 0.210, 0.065, 0.552, 0.556])

array([1,     9,     8,     0,     2,     4,     3,     6,     5,     7])
```

true

pred

Get the indexes of pred sorted in descending order

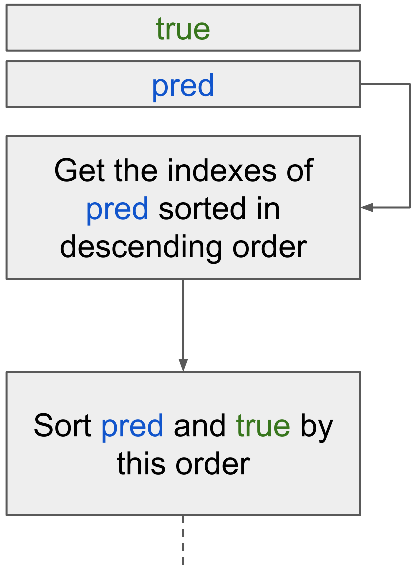
Vectorization



```
array([0,     0,     0,     1,     1,     1,     0,     1,     1,     1])
array([0.523, 0.957, 0.489, 0.244, 0.311, 0.093, 0.210, 0.065, 0.552, 0.556])


array([1,     9,     8,     0,     2,     4,     3,     6,     5,     7])
```

All relevant thresholds

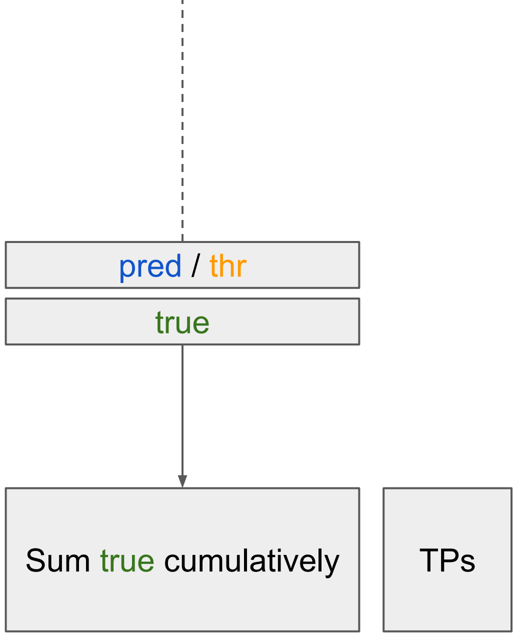
```
array([0.957, 0.556, 0.552, 0.523, 0.489, 0.311, 0.244, 0.210, 0.093, 0.065])
array([0,     1,     1,     0,     0,     1,     1,     0,     1,     1])
```

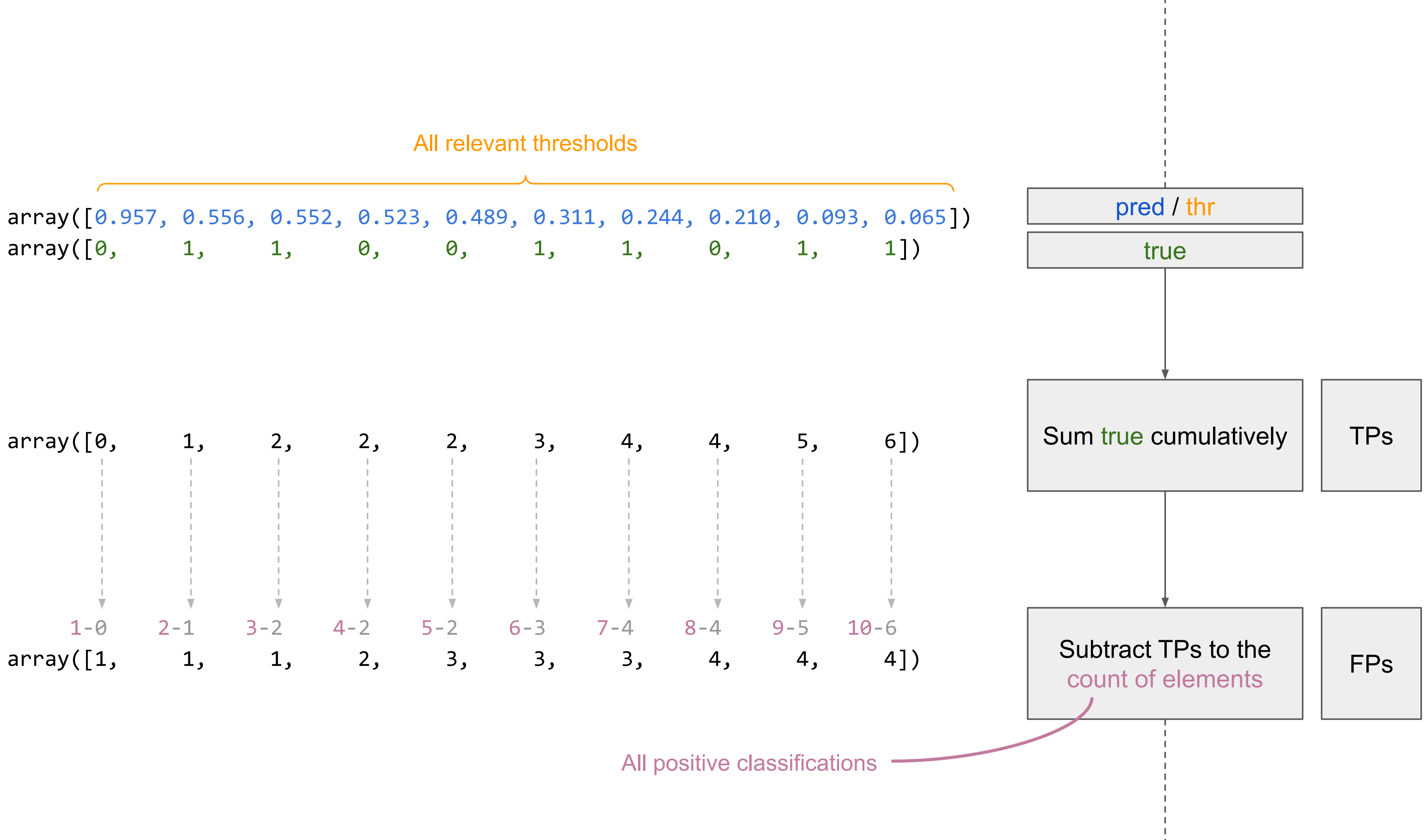

true
pred
Get the indexes of pred sorted in descending order
Sort pred and true by this order

All relevant thresholds

```
array([0.957, 0.556, 0.552, 0.523, 0.489, 0.311, 0.244, 0.210, 0.093, 0.065])
array([0,     1,     1,     0,     0,     1,     1,     0,     1,     1])
```

```
array([0,     1,     2,     2,     2,     3,     4,     4,     5,     6])
```

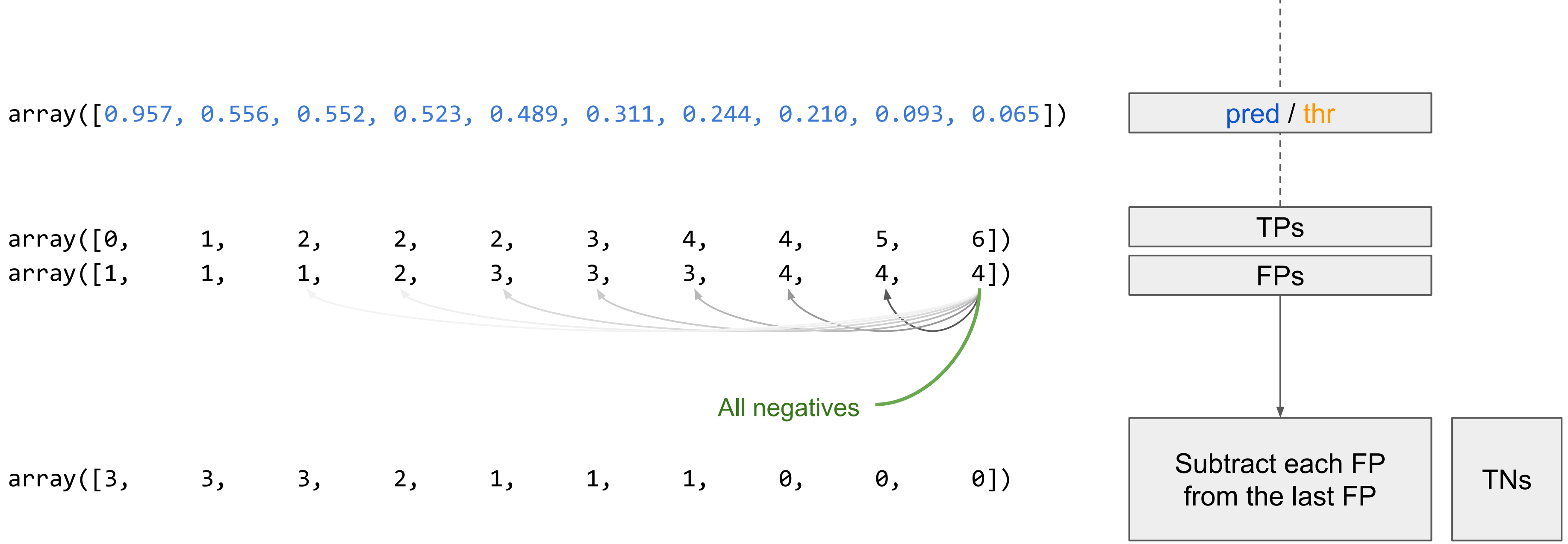



All relevant thresholds
pred / thr
true
Sum true cumulatively
TPs
Subtract TPs to the count of elements
FPs
All positive classifications

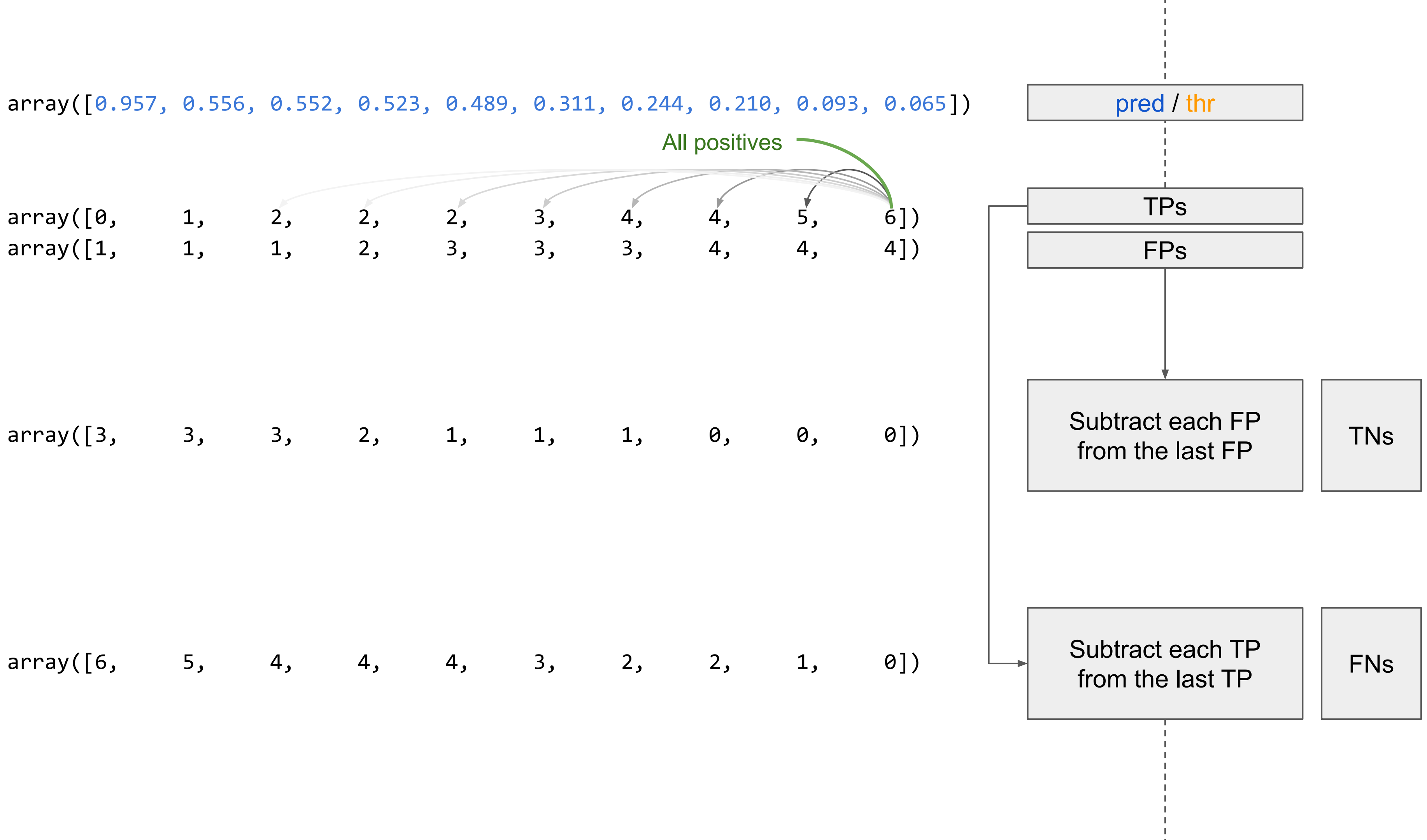
```
array([0.957, 0.556, 0.552, 0.523, 0.489, 0.311, 0.244, 0.210, 0.093, 0.065])
array([0,     1,     1,     0,     0,     1,     1,     0,     1,     1])
```

```
array([0,     1,     2,     2,     2,     3,     4,     4,     5,     6])
```

```
   1-0   2-1   3-2   4-2   5-2   6-3   7-4   8-4   9-5  10-6
array([1,     1,     1,     2,     3,     3,     3,     4,     4,     4])
```

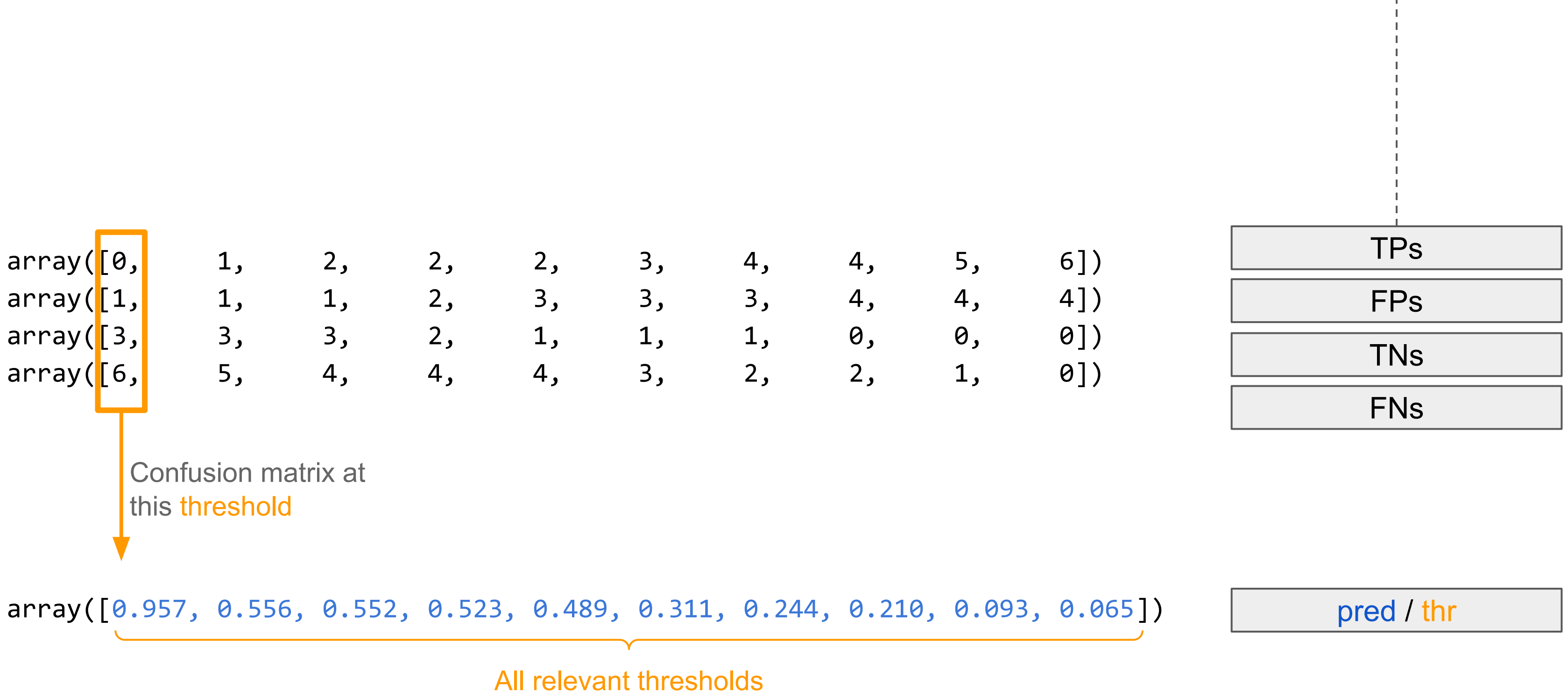


```
array([0.957, 0.556, 0.552, 0.523, 0.489, 0.311, 0.244, 0.210, 0.093, 0.065])


array([0,     1,     2,     2,     2,     3,     4,     4,     5,     6])
array([1,     1,     1,     2,     3,     3,     3,     4,     4,     4])



array([3,     3,     3,     2,     1,     1,     1,     0,     0,     0])
```



```
array([0.957, 0.556, 0.552, 0.523, 0.489, 0.311, 0.244, 0.210, 0.093, 0.065])

array([0,     1,     2,     2,     2,     3,     4,     4,     5,     6])
array([1,     1,     1,     2,     3,     3,     3,     4,     4,     4])



array([3,     3,     3,     2,     1,     1,     1,     0,     0,     0])



array([6,     5,     4,     4,     4,     3,     2,     2,     1,     0])
```

All positives
pred / thr
TPs
FPs
Subtract each FP from the last FP
TNs
Subtract each TP from the last TP
FNs



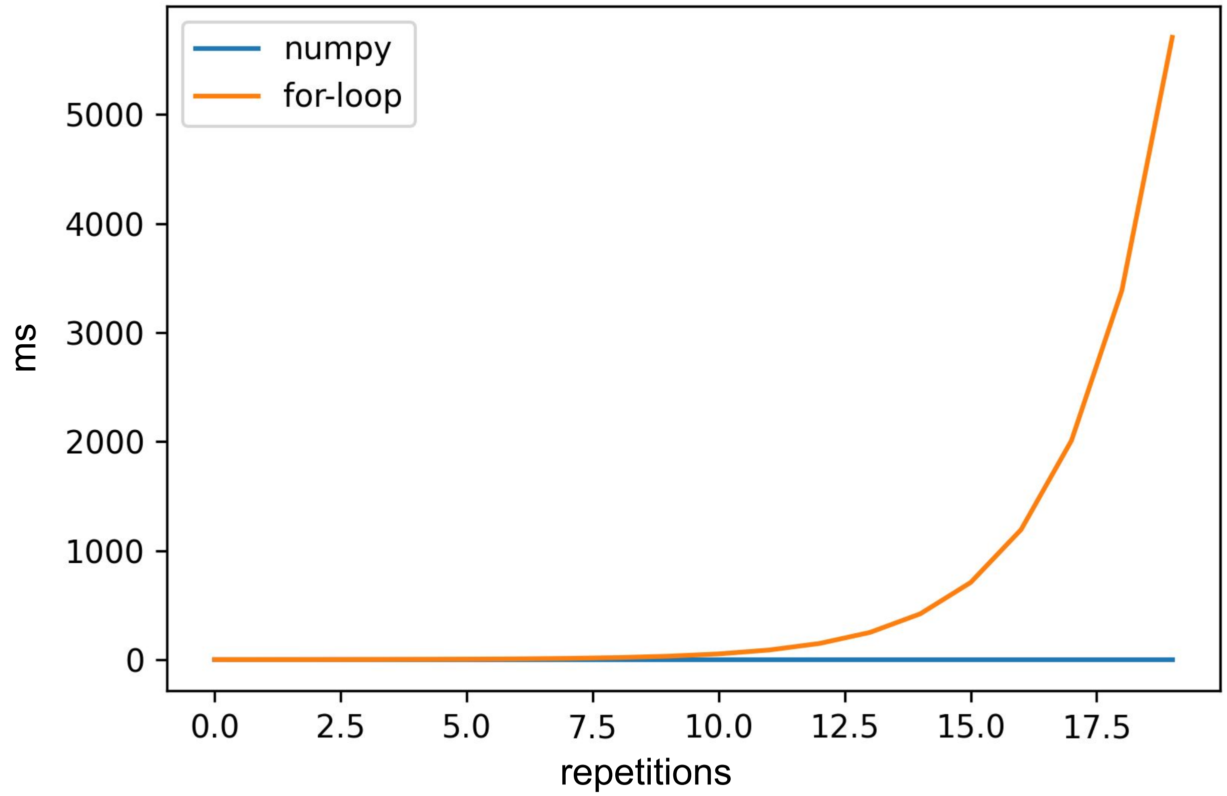
```
array([0,    1,    2,    2,    2,    3,    4,    4,    5,    6])
array([1,    1,    1,    2,    3,    3,    3,    4,    4,    4])
array([3,    3,    3,    2,    1,    1,    1,    0,    0,    0])
array([6,    5,    4,    4,    4,    3,    2,    2,    1,    0])
```

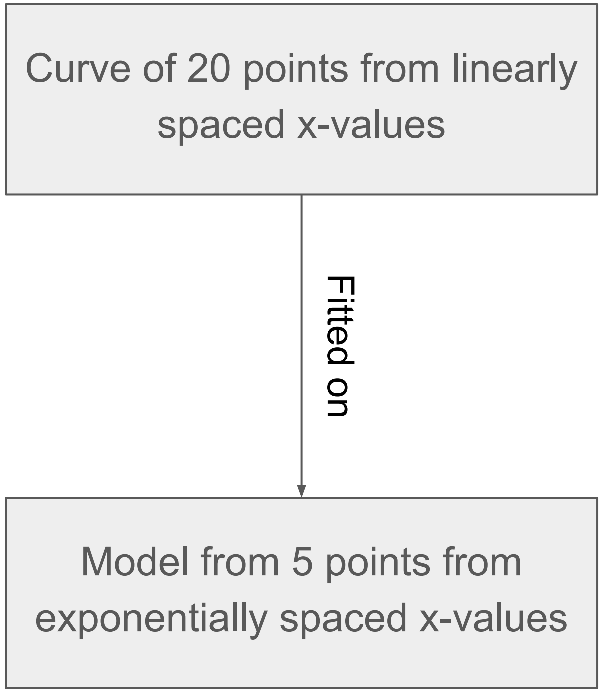
```
array([0.957, 0.556, 0.552, 0.523, 0.489, 0.311, 0.244, 0.210, 0.093, 0.065])
```

Results

Now it’s fast

Efficiency




Curve of 20 points from linearly spaced x-values

Fitted on

Model from 5 points from exponentially spaced x-values

Code complexity

Dimension	Elements in dimension	Cumulative product	Order of magnitude	Execution time
Metrics	10 → 18	18	10^1	<1 second
Proteins	646	~11,500	10^4	<1 second
Thresholds	5 → 1000	~11,500,000	10^7	~1 minute
Predictors	30	~350,000,000	10^8	~1 minute
Bootstrap	1,000	~350,000,000,000	**10^{11}**	**~10 minutes**

Thank you