STATISTICAL TOOLS TO SOLVE **COMMON** DEVELOPMENT **PROBLEMS**

@alblurrow

If the only tool you have is a hammer, you treat everything as if it were a nail

"Abraham Maslow"

Photo: https://flic.kr/p/96VdSv

for loops
if/else
hash tables
lists

map, reduce, zip

CALCULATE UNIQUE ELEMENTS

```
func unique(ids []string) int {

    table := make(map[string]bool)

    for id := range ids {
        table[id] = true
    }

    return len(table)
}
```

WHAT IF?

Many many elements

A stream

Different time windows

The process is killed

Distributed

HOW ACCURATE DOES IT NEED TO BE?

HYPERLOGLOG

For BIG data sets (> 10^6)

Constant memory usage (few KiB)

Error < 2 %

HOW LIKELY

Get 2 heads in a row?

2^2 states. 1 in 4

Get 10 heads in a row?

2^{10} states. 1 in 1024

SAME IDEA

Instead of counting heads, we count streaks of zeros

ID: 1234567890 ➪ 1

ID: 9876543200 ➪ 2

Likelihood of seeing:

Ending in 0: 1 out of 10^1 ➪ Around 10 IDs

Ending in 00: 1 out of 10^2 ➪ Around 100 IDs

PUTTING IT ALL TOGETHER

Hash function

It uses buckets

Average of buckets

BENEFITS

Low and constant memory

Easy to store

Stream of data

Can be combined

- Distributed
- Different time windows

Many implementations already available

LET'S SEE IT

NOT THE ONLY ONE

More approximate algorithms:

Quantiles: t-digest

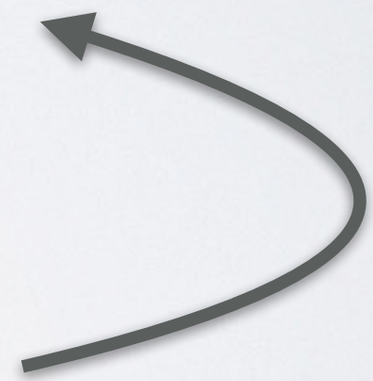
Histograms

Top-k: heavy hitters

RESOURCE ALLOCATION

Initially simple

Product Owner adds new rule



if user has X do XX else YY

if/else nightmare

A PROBABILIST APPROACH

$$P(A \wedge B) = P(A) \times P(B \mid A)$$

A PROBABILIST APPROACH

For all resources:

Calculate: `Score(resource, user)`

Sort

Choose the top X

WHATS THE SCORE?

For each rule: function to return [0,1]

Multiply all of them

EXAMPLES

```
func money(user, resource) float64 {
    return 1 - (Abs(resource.Value - user.Money) / MAX_MONEY)
}

func userTypeA(user, resource) float64 {
    if user.Type == TypeA {
        return 1
    }
    return 0
}

func increaseWithAge(user, resource) float64 {

    daysInService := user.Time.Sub(Now()).Days()

    return Log2( daysInService / 3 + 0.5 )  // 1 in 13 days

}
```

BENEFITS

Simpler code

Changing rules is easy

Adapts automatically

ADAPTING

```
func reactions(user, resource) float64 {

    return resource.Consumptions / resource.Impressions
}

func value(user, resource) float64 {

    return resource.Value
}
```

WE NEED SOME DATA

If it reacts to some real data

We need that data

Divide the allocations:

10% is completely random

90% Uses the algorithm

BUZZ WORDS!

Epsilon (**ϵ**) greedy

Machine learning

Reinforced learning

Photo: https://flic.kr/p/47Cfvn

SHOW ME THE CODE

CHOOSE THE RIGHT TOOL

You don't always need 100% accuracy

Probability and statistics are helpful

Learn the basics, not the theorems