

Analyzing log data with Apache Spark

William Benton
Red Hat Emerging Technology

BACKGROUND

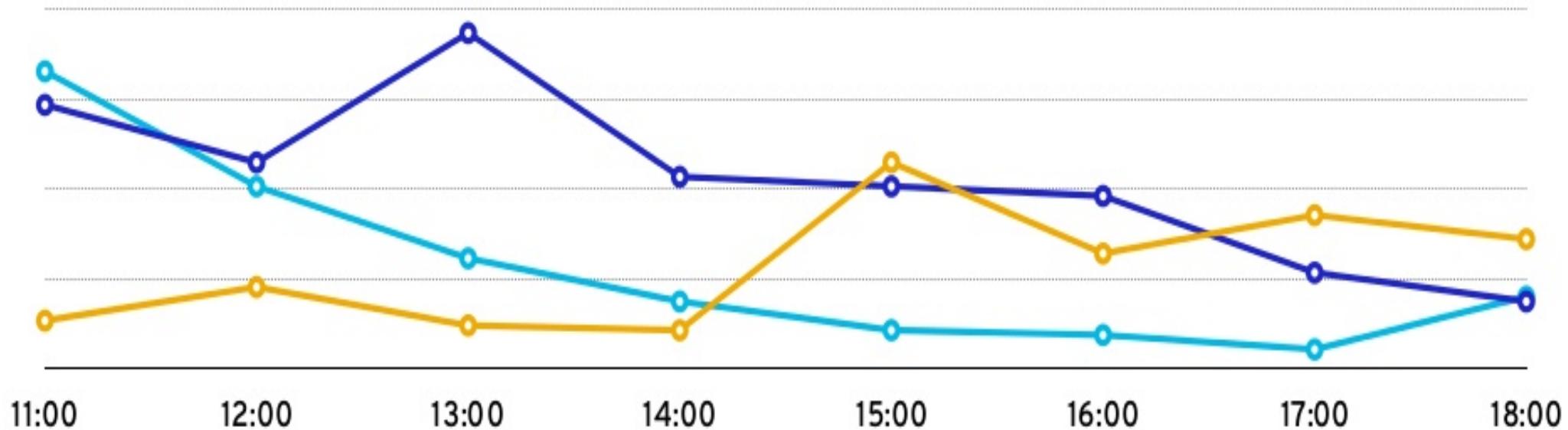
Challenges of log data

Challenges of log data

```
SELECT hostname, DATEPART(HH, timestamp) AS hour, COUNT(msg)  
FROM LOGS WHERE level='CRIT' AND msg LIKE '%failure%'  
GROUP BY hostname, hour
```

Challenges of log data

```
SELECT hostname, DATEPART(HH, timestamp) AS hour, COUNT(msg)  
FROM LOGS WHERE level='CRIT' AND msg LIKE '%failure%'  
GROUP BY hostname, hour
```



Challenges of log data

postgres

INFO

INFO

WARN

CRIT

DEBUG

INFO

httpd

GET

GET

GET

POST

GET (404)

syslog

WARN

INFO

WARN

INFO

INFO

INFO

(ca. 2000)

Challenges of log data

postgres

INFO	WARN	CRIT	INFO
------	------	------	------

Cassandra

INFO	CRIT	INFO	INFO
------	------	------	------

httpd

GET	GET	GET	POST
-----	-----	-----	------

nginx

GET	POST	PUT	POST
-----	------	-----	------

syslog

INFO	INFO	INFO	WARN
------	------	------	------

Rails

INFO	INFO	WARN	INFO
------	------	------	------

CouchDB

INFO	INFO	CRIT	INFO
------	------	------	------

redis

INFO	CRIT	INFO	INFO
------	------	------	------

httpd

GET	GET (404)	POST	
-----	-----------	------	--

httpd

PUT (500)	GET	PUT	
-----------	-----	-----	--

Django

INFO	INFO	INFO	WARN
------	------	------	------

syslog

INFO	WARN	INFO	INFO
------	------	------	------

haproxy

INFO	INFO	WARN	DEBUG	CRIT
------	------	------	-------	------

k8s

WARN	WARN	INFO	INFO	INFO
------	------	------	------	------

(ca. 2016)

Challenges of log data



How many services are generating logs in your datacenter today?

DATA INGEST

Collecting log data

collecting



normalizing



analysis

Ingesting live log
data via rsyslog,
logstash, fluentd

Reconciling log
record metadata
across sources

cache warehoused
data as Parquet files
on Gluster volume
local to Spark cluster

Collecting log data

warehousing

Storing normalized
records in ES indices

analysis

cache warehoused
data as Parquet files
on Gluster volume
local to Spark cluster

Collecting log data

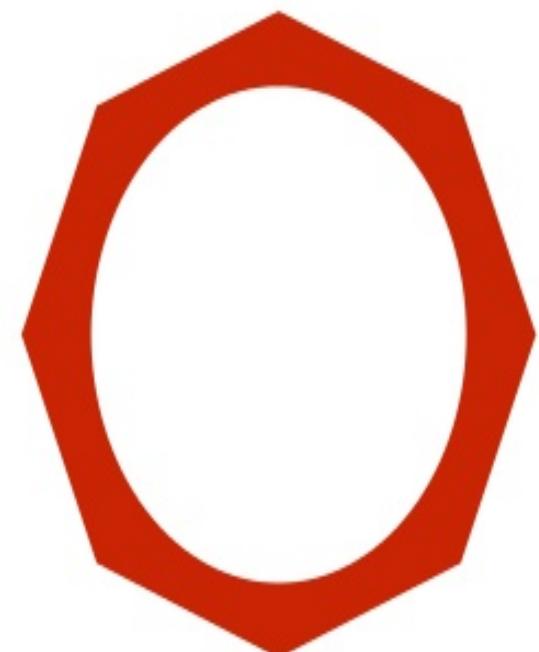
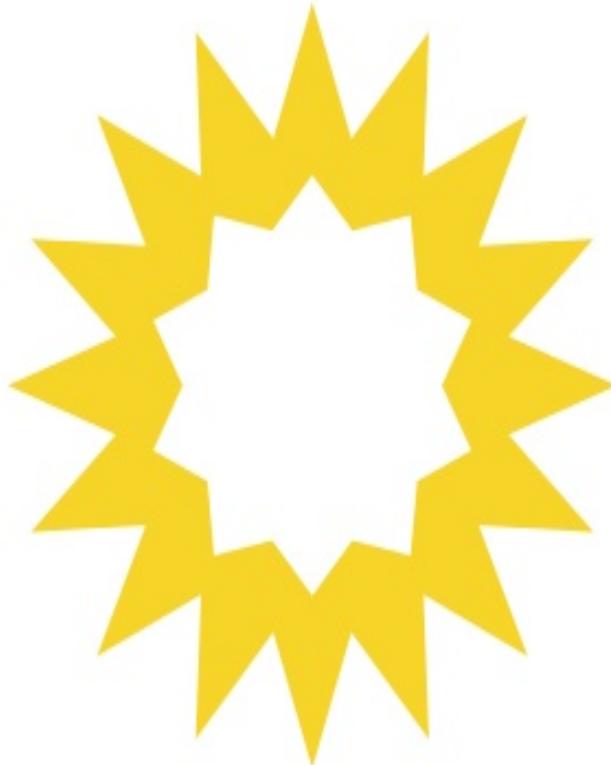
warehousing

Storing normalized
records in ES indices

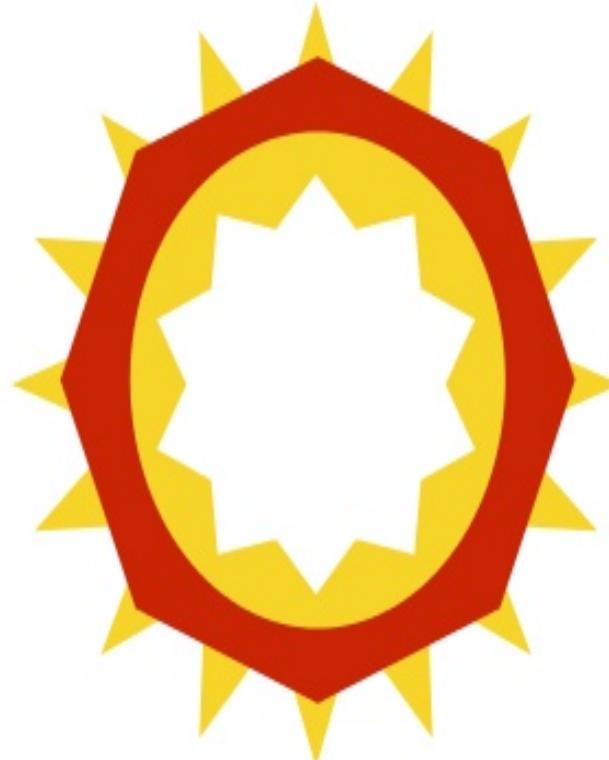
analysis

cache warehoused
data as Parquet files
on Gluster volume
local to Spark cluster

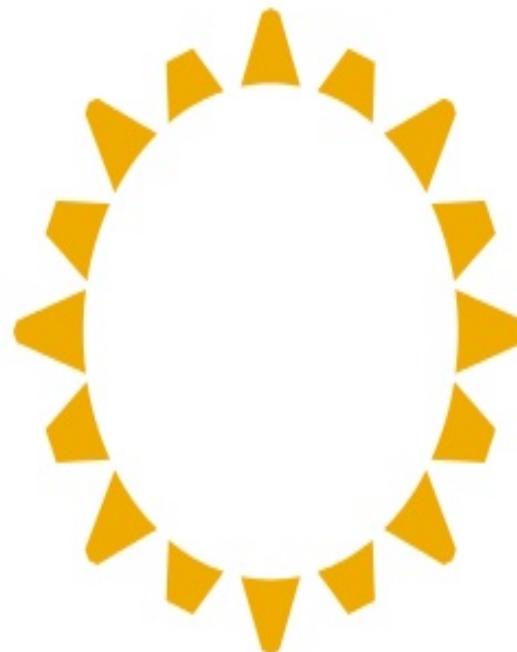
Schema mediation



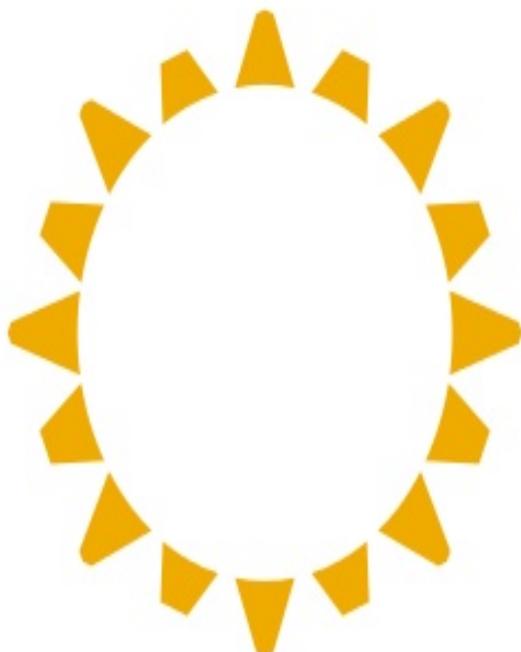
Schema mediation



Schema mediation



Schema mediation



timestamp, level, host, IP
addresses, message, &c.

rsyslog-style metadata, like
app name, facility, &c.

Exploring structured data

```
logs
  .select("level").distinct
  .map { case Row(s: String) => s }
  .collect
debug, notice, emerg,
err, warning, crit, info,
severe, alert
```

```
logs
  .groupBy($"level", $"rsyslog.app_name")
  .agg(count("level").as("total"))
  .orderBy($"total".desc)
  .show
info      kubelet      17933574
info      kube-proxy   10961117
err       journal     6867921
info      systemd     5184475
...
```

Exploring structured data

```
logs
  .select("level").distinct
  .as[String].collect
                                debug, notice, emerg,
                                err, warning, crit, info,
                                severe, alert
```

```
logs
  .groupBy($"level", $"rsyslog.app_name")
  .agg(count("level").as("total"))
  .orderBy($"total".desc)
  .show
```

level	app_name	count
info	kubelet	17933574
info	kube-proxy	10961117
err	journal	6867921
info	systemd	5184475
...		

Exploring structured data

```
logs
  .select("level").distinct
  .as[String].collect
    debug, notice, emerg,
    err, warning, crit, info,
    severe, alert
```

This class must be declared outside the REPL!

```
logs
  .groupBy($"level", $"rsyslog.app_name")
  .agg(count("level").as("total"))
  .orderBy($"total".desc)
  .show
```

	info	kubelet	17933574
	info	kube-proxy	10961117
	err	journal	6867921
	info	systemd	5184475
		...	

FEATURE ENGINEERING

From log records to vectors

What does it mean for two sets of categorical features to be similar?

red	-> 000	pancakes	-> 10000
green	-> 010	waffles	-> 01000
blue	-> 100	aeblikiver	-> 00100
orange	-> 001	omelets	-> 00001
		bacon	-> 00000
		hash browns	-> 00010

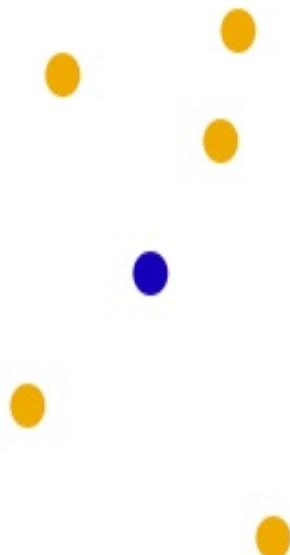
From log records to vectors

What does it mean for two sets of categorical features to be similar?

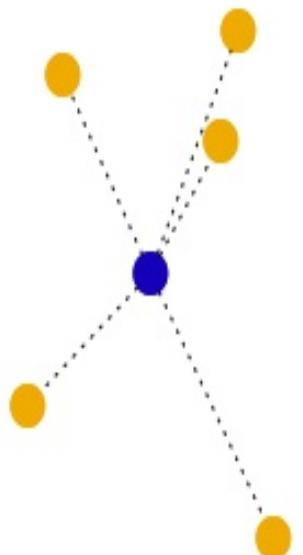
red	-> 000	pancakes	-> 10000
green	-> 010	waffles	-> 01000
blue	-> 100	aeblikiver	-> 00100
orange	-> 001	omelets	-> 00001
		bacon	-> 00000
		hash browns	-> 00010

red pancakes -> 00010000
orange waffles -> 00101000

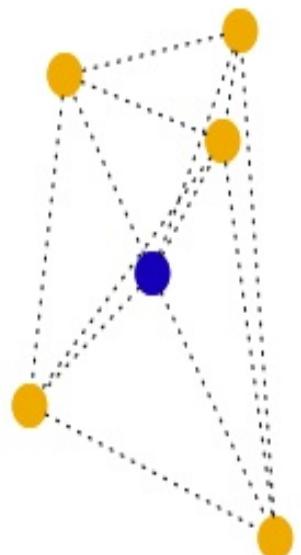
Similarity and distance



Similarity and distance

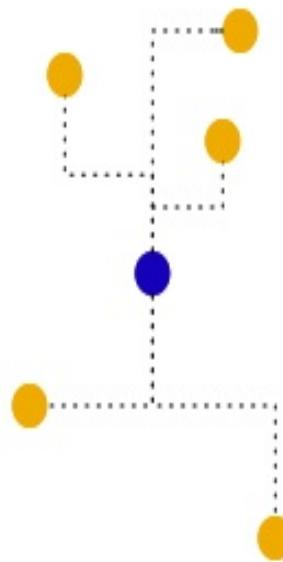


Similarity and distance



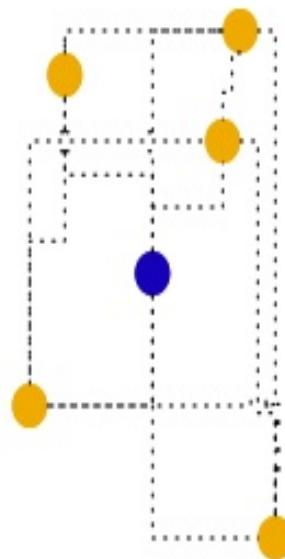
$$\sqrt{(q - p) \cdot (q - p)}$$

Similarity and distance



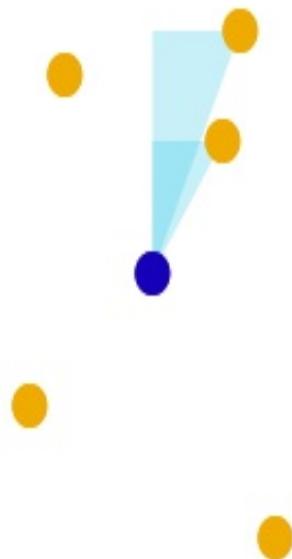
$$\sum_{i=1}^n |p_i - q_i|$$

Similarity and distance



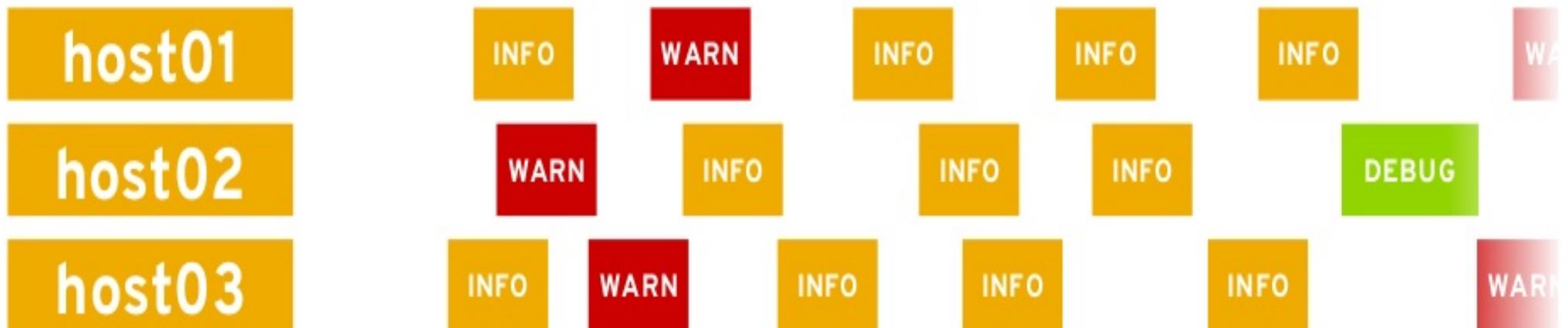
$$\sum_{i=1}^n |p_i - q_i|$$

Similarity and distance

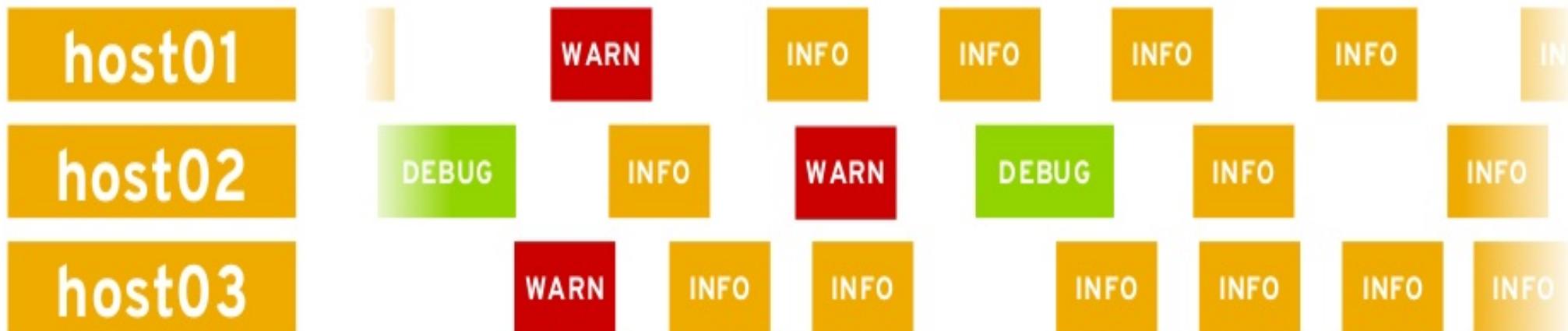


$$\frac{\mathbf{P} \cdot \mathbf{q}}{\|\mathbf{P}\| \|\mathbf{q}\|}$$

Other interesting features



Other interesting features



Other interesting features



Other interesting features

- ★★★: Great food, great service, a must-visit!
- ★★★★: Our whole table got gastroenteritis.
- ★: This place is so wonderful that it has ruined all other tacos for me and my family.

Other interesting features

INFO: Everything is great! Just checking in to let you know I'm OK.

Other interesting features

INFO: Everything is great! Just checking in to let you know I'm OK.
CRIT: No requests in last hour; suspending running app containers.

Other interesting features

INFO: Everything is great! Just checking in to let you know I'm OK.
CRIT: No requests in last hour; suspending running app containers.
INFO: Phoenix datacenter is on fire; may not rise from ashes.

Other interesting features

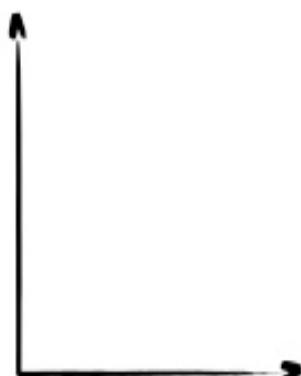
INFO: Everything is great! Just checking in to let you know I'm OK.
CRIT: No requests in last hour; suspending running app containers.
INFO: Phoenix datacenter is on fire; may not rise from ashes.

See <https://links.freevariable.com/nlp-logs/> for more!

**VISUALIZING STRUCTURE
and FINDING OUTLIERS**

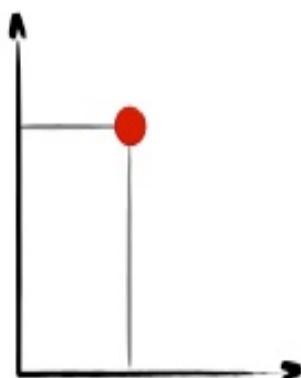
Multidimensional data

Multidimensional data



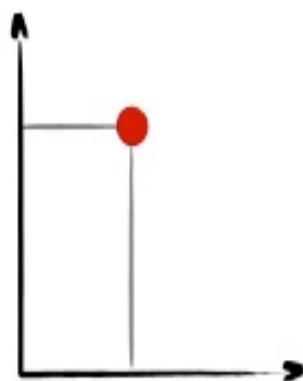
[4,7]

Multidimensional data

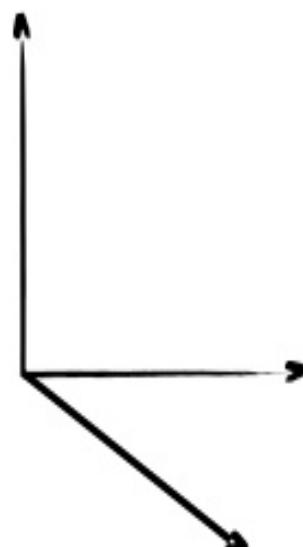


[4,7]

Multidimensional data

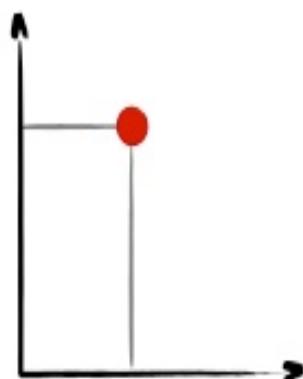


[4,7]

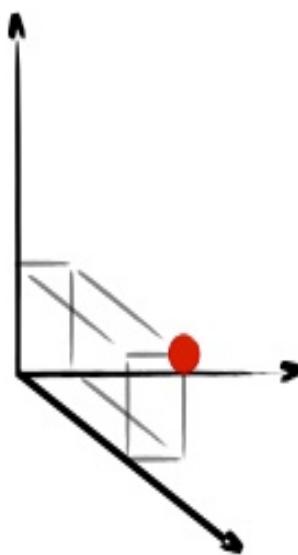


[2,3,5]

Multidimensional data

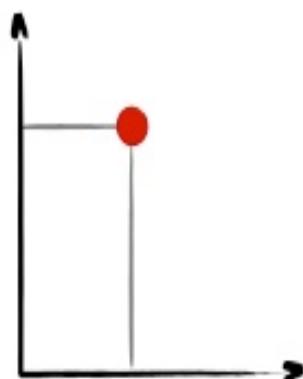


[4,7]

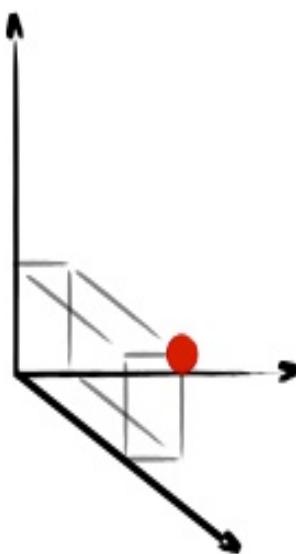


[2,3,5]

Multidimensional data



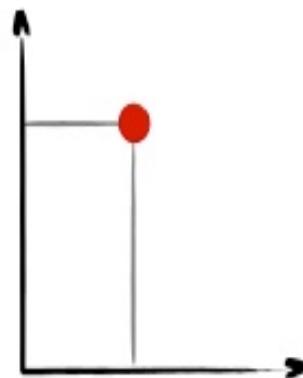
[4,7]



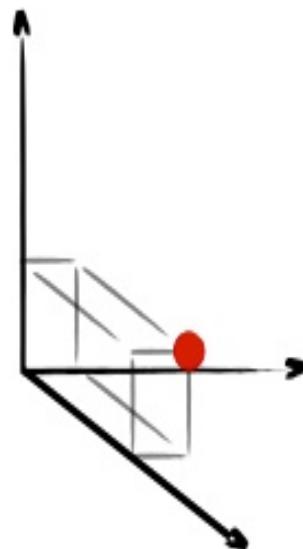
[2,3,5]

[7,1,6,5,12,
8,9,2,2,4,
7,11,6,1,5]

Multidimensional data



[4,7]



[2,3,5]

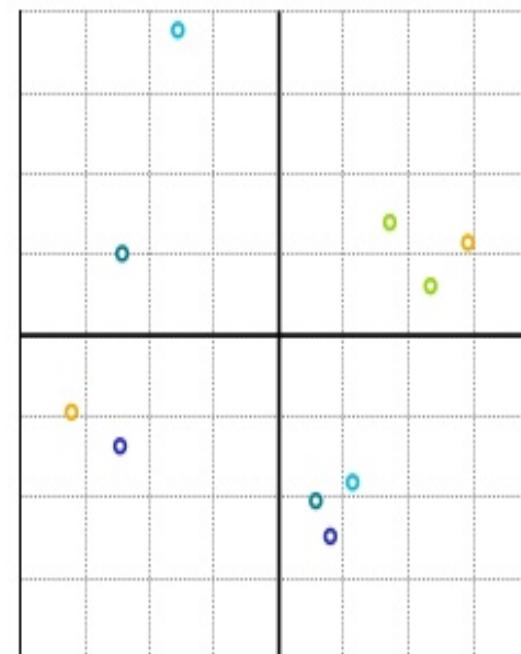


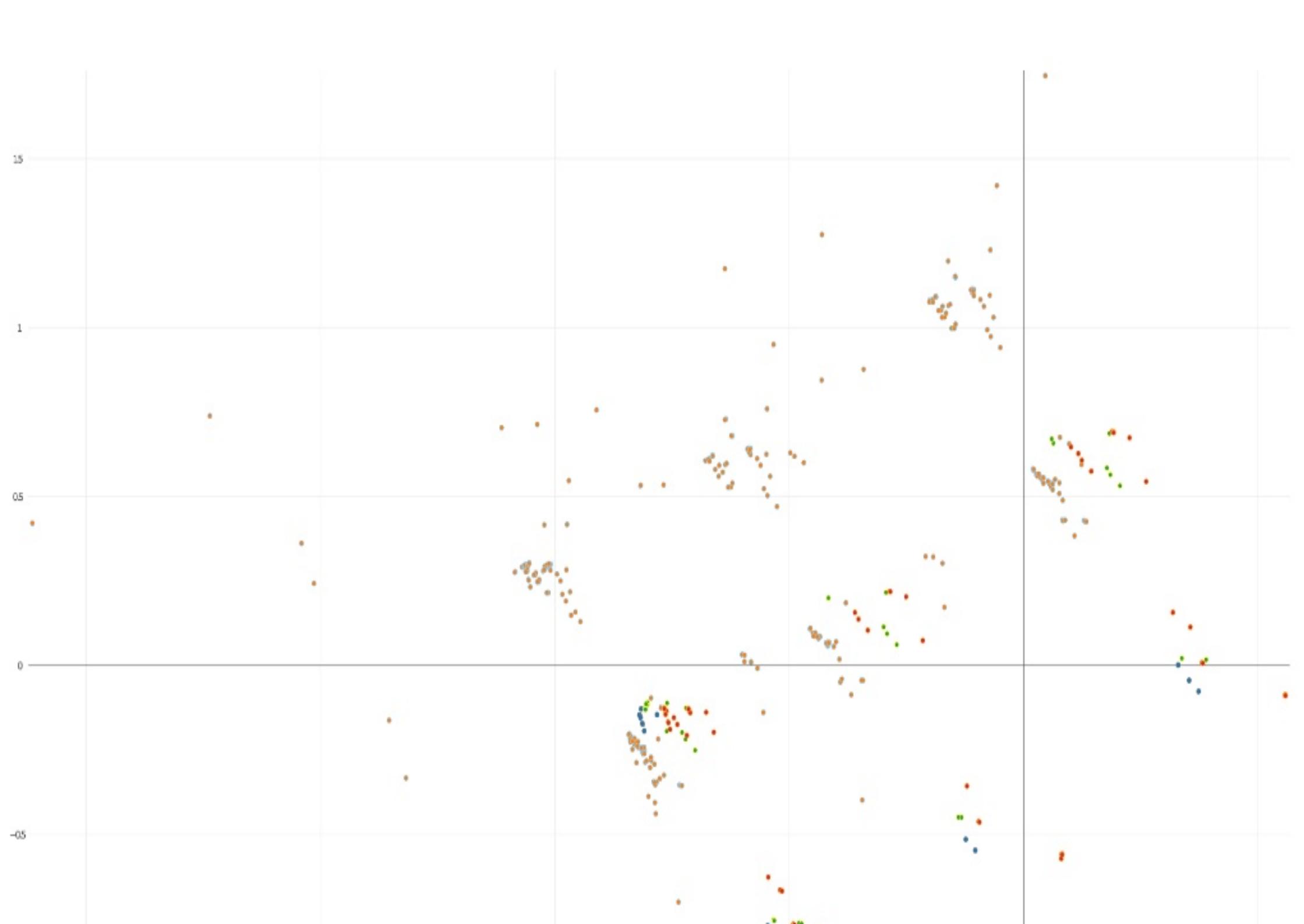
[7,1,6,5,12,
8,9,2,2,4,
7,11,6,1,5]

A linear approach: PCA

A linear approach: PCA

0	0	0	1	1	0	1	0	1	0
0	0	1	0	0	0	1	1	0	0
1	0	1	1	0	1	0	0	0	0
0	0	0	0	0	0	1	1	0	1
0	1	0	0	1	0	0	1	0	0
1	0	0	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0	0	0
0	1	0	0	0	1	0	0	1	1
0	0	0	0	1	0	0	1	0	1
1	1	0	0	0	0	0	0	0	1

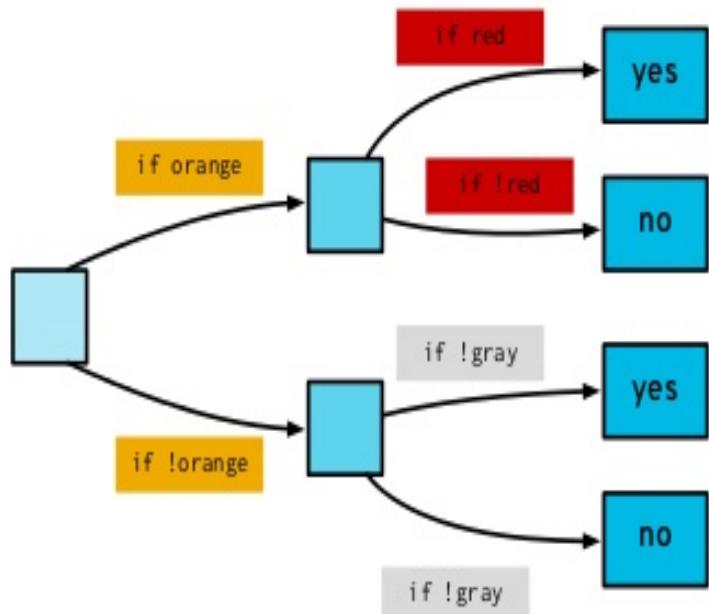




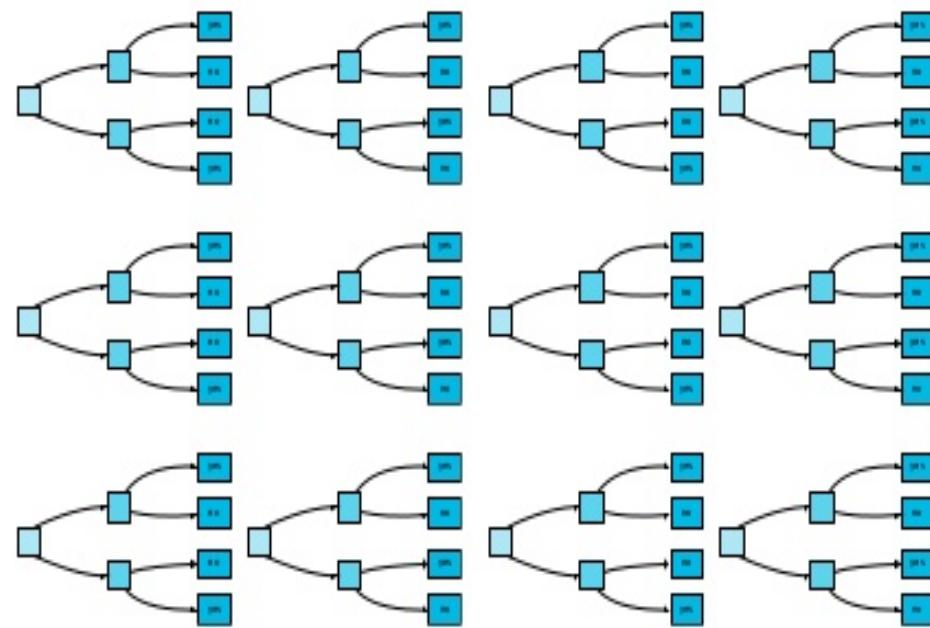
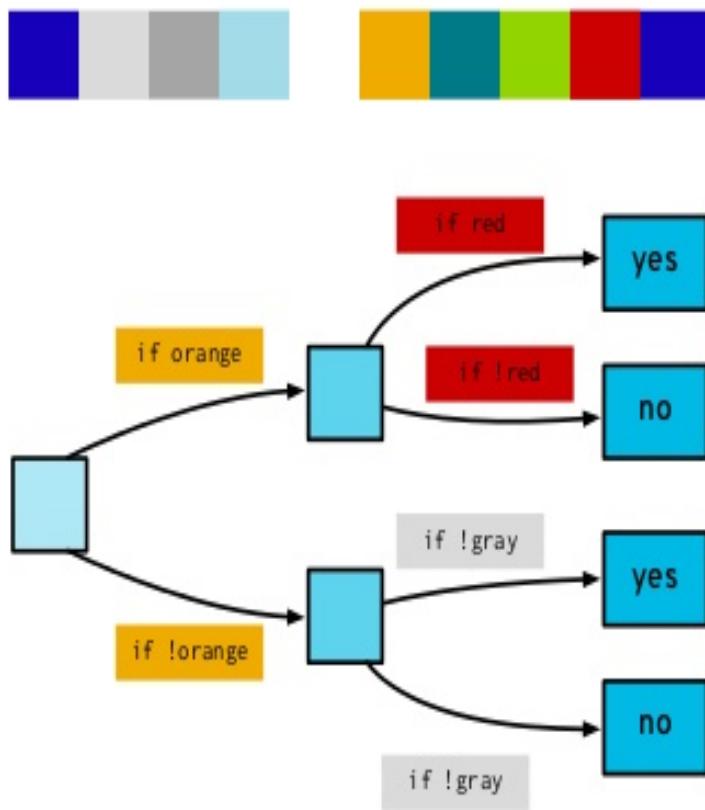
Tree-based approaches



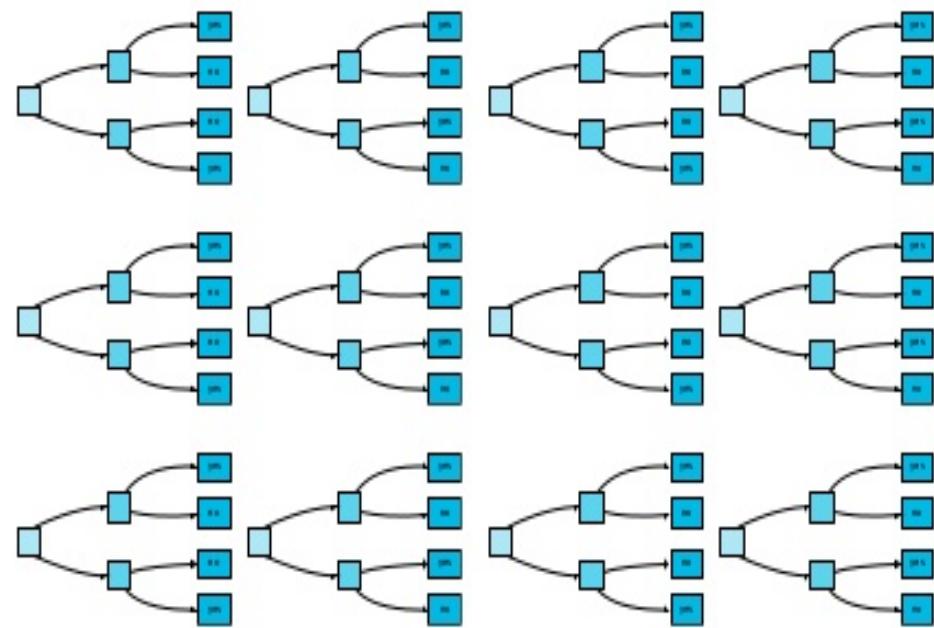
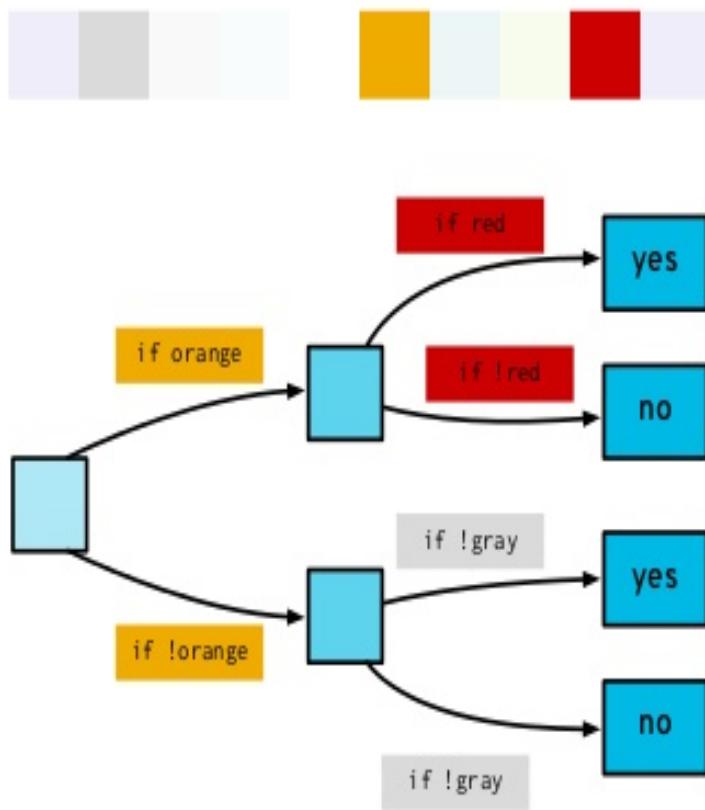
Tree-based approaches



Tree-based approaches



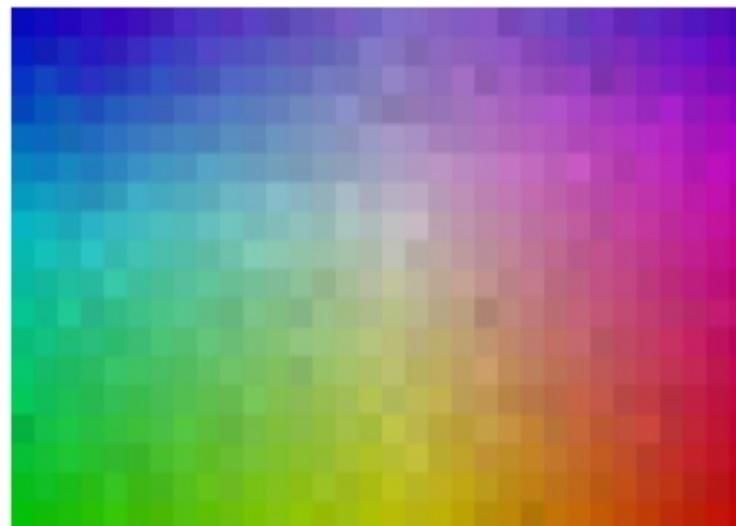
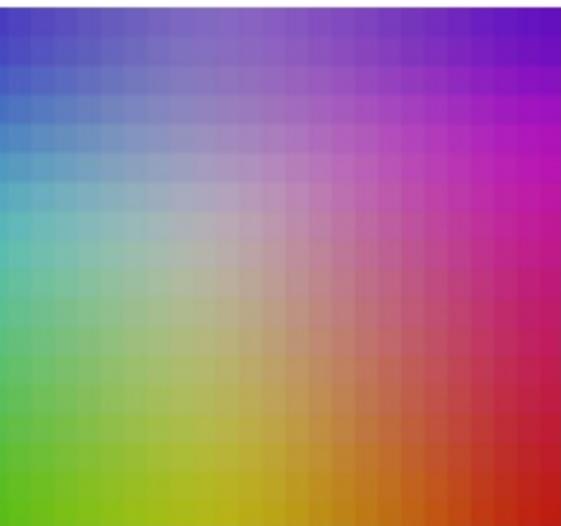
Tree-based approaches



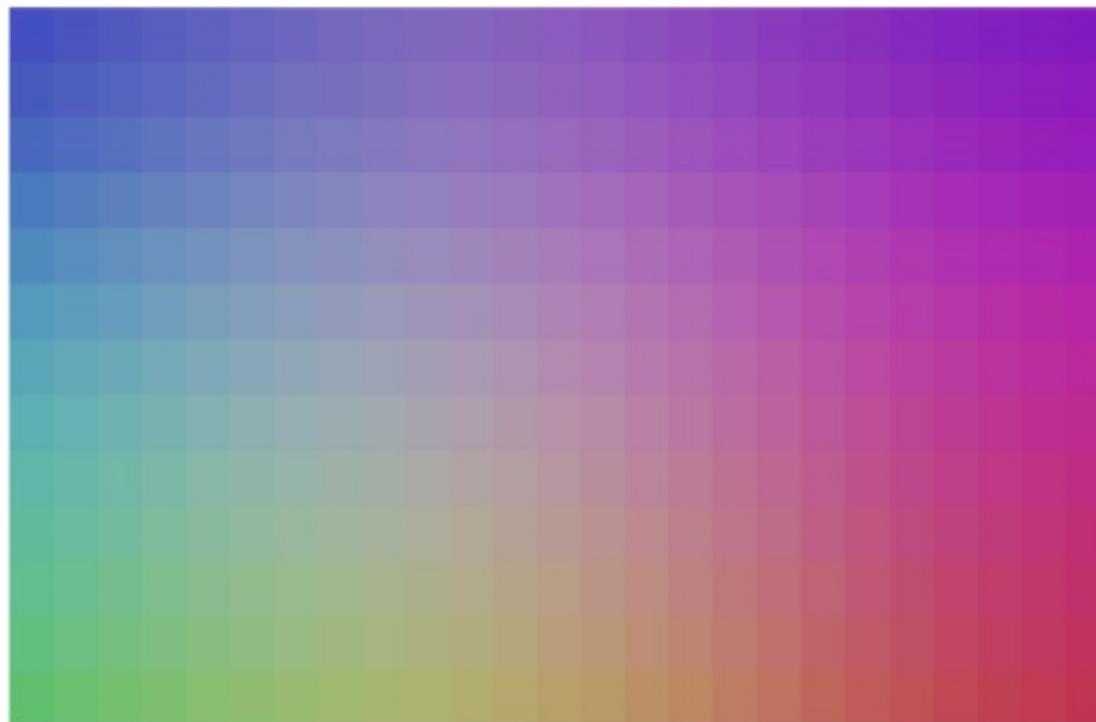
Self-organizing maps



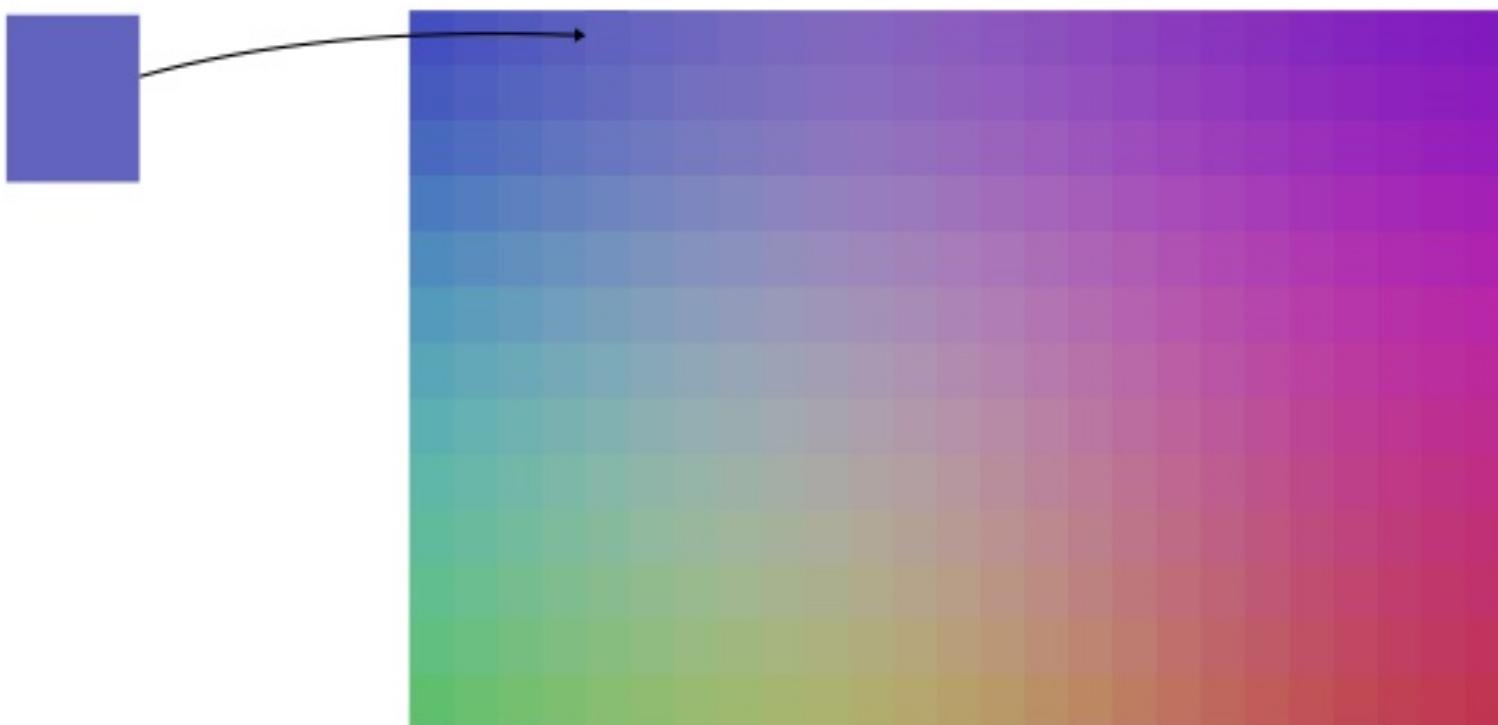
Self-organizing maps



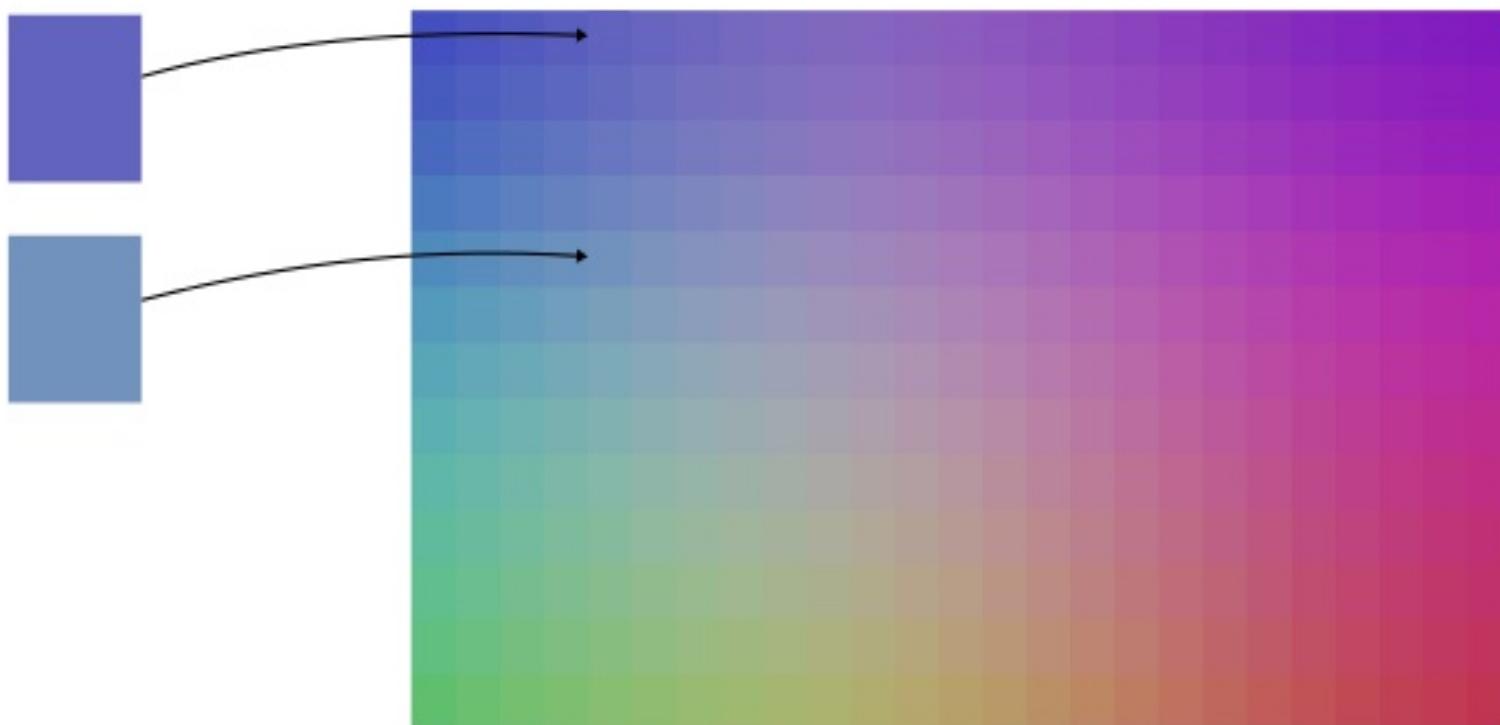
Finding outliers with SOMs



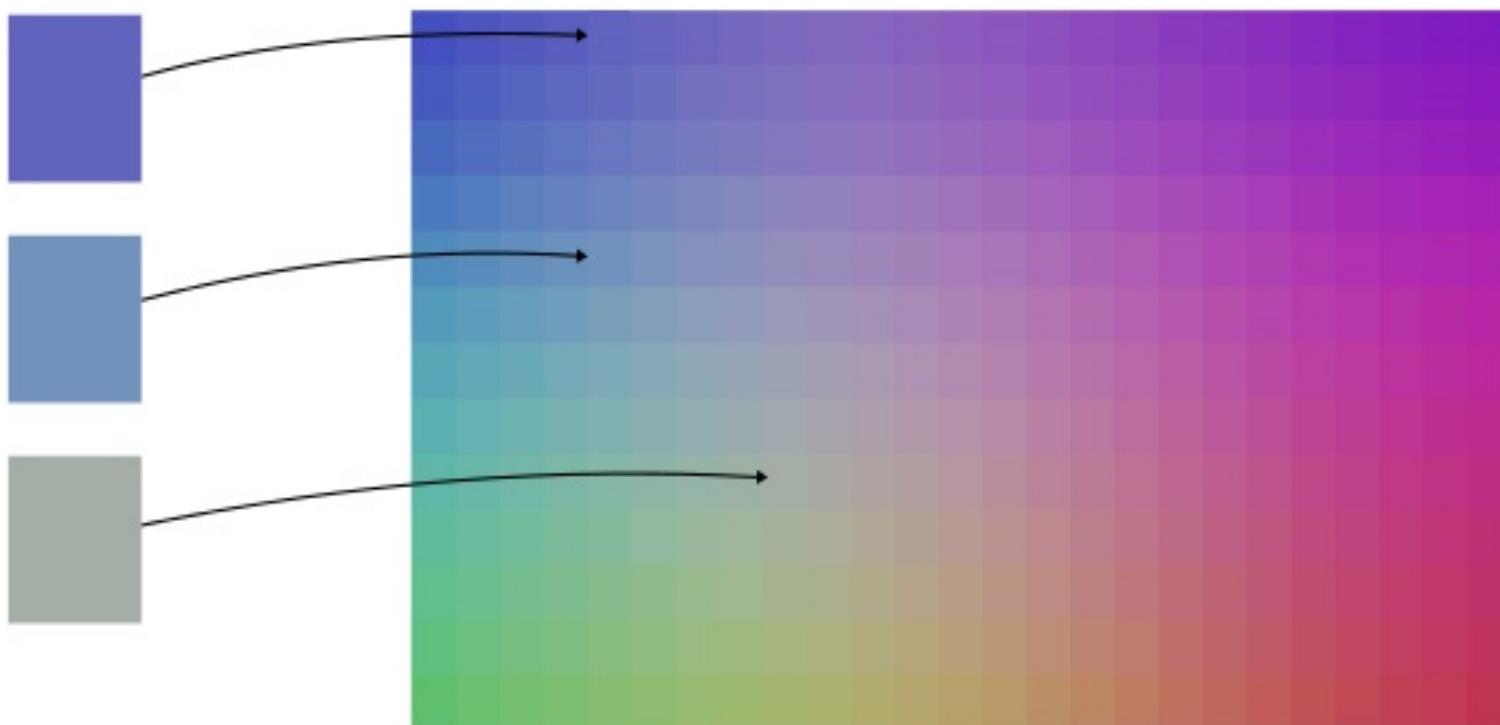
Finding outliers with SOMs



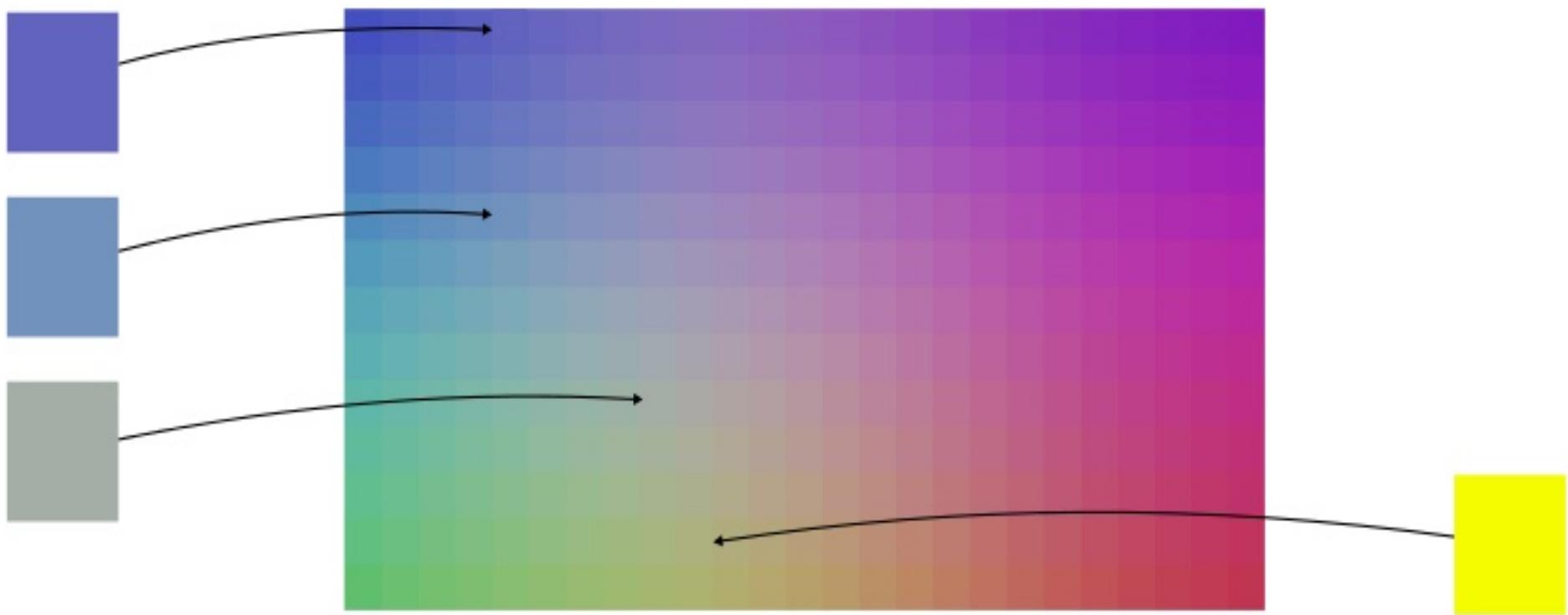
Finding outliers with SOMs



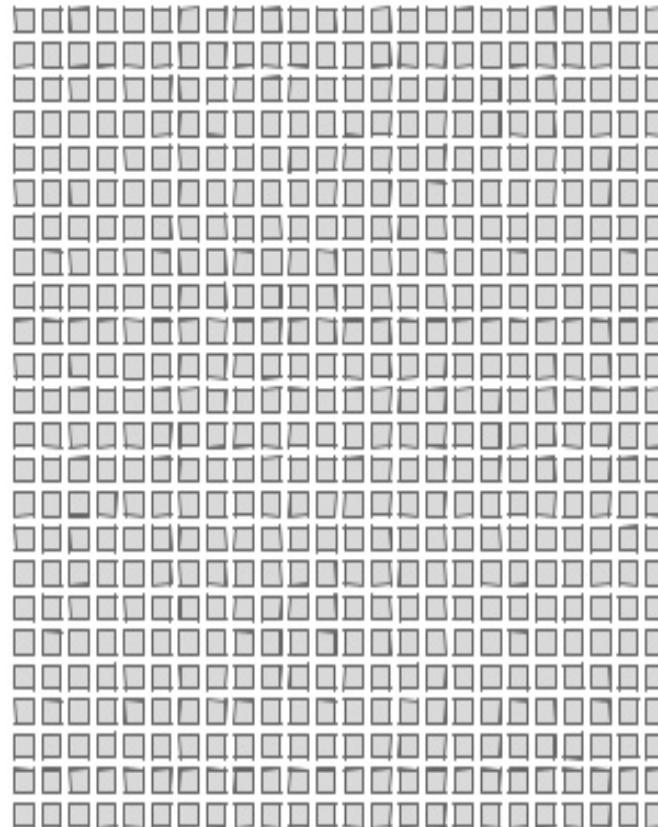
Finding outliers with SOMs



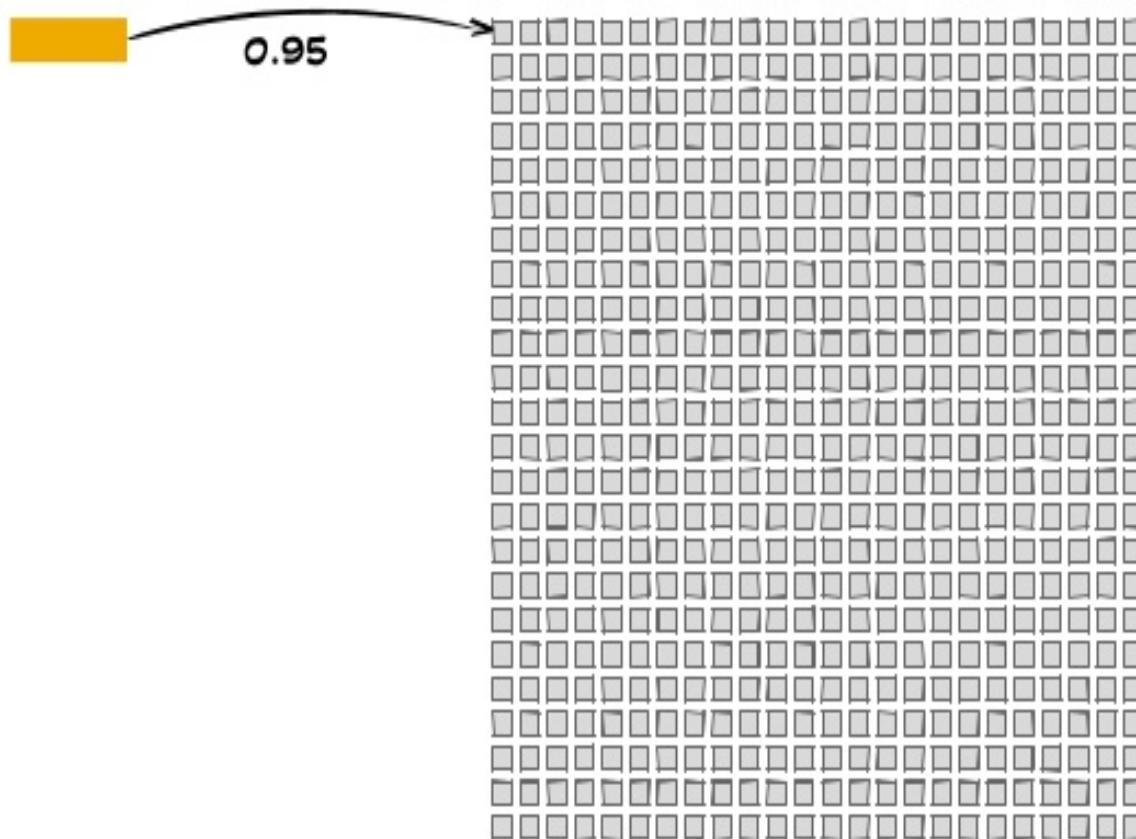
Finding outliers with SOMs



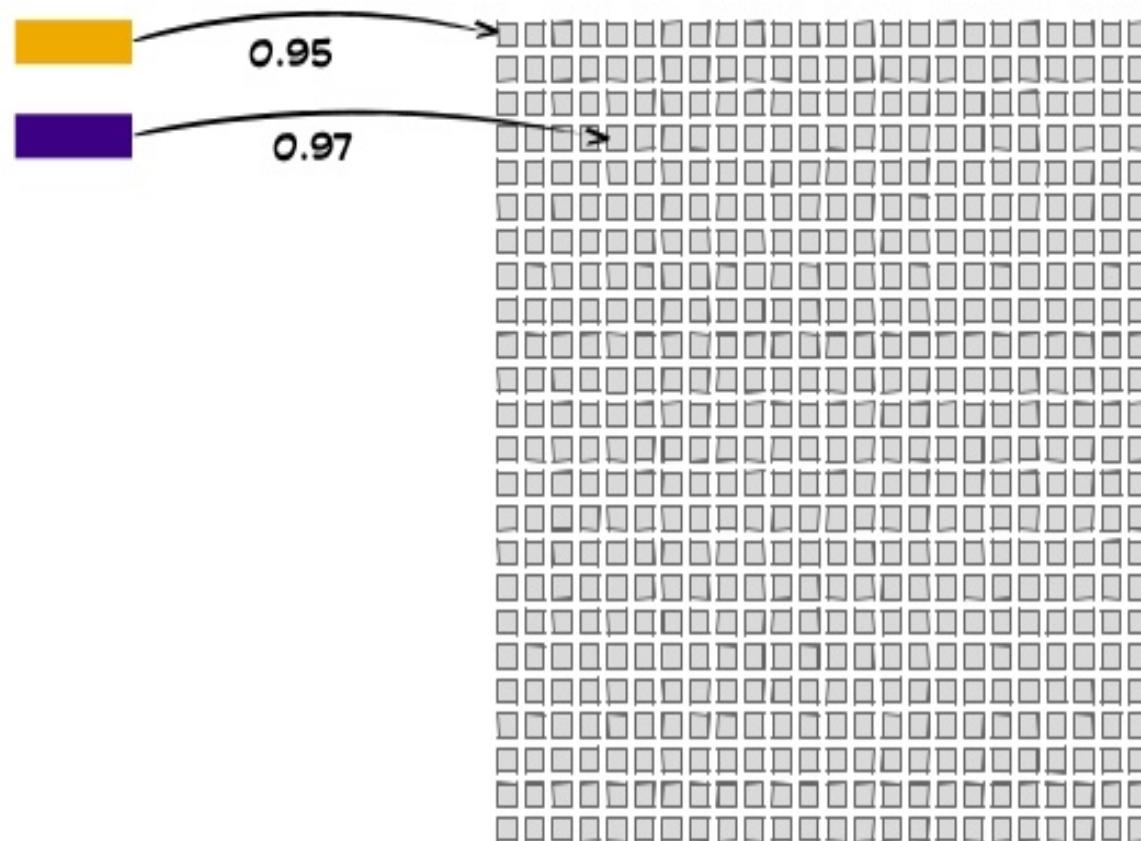
Outliers in log data



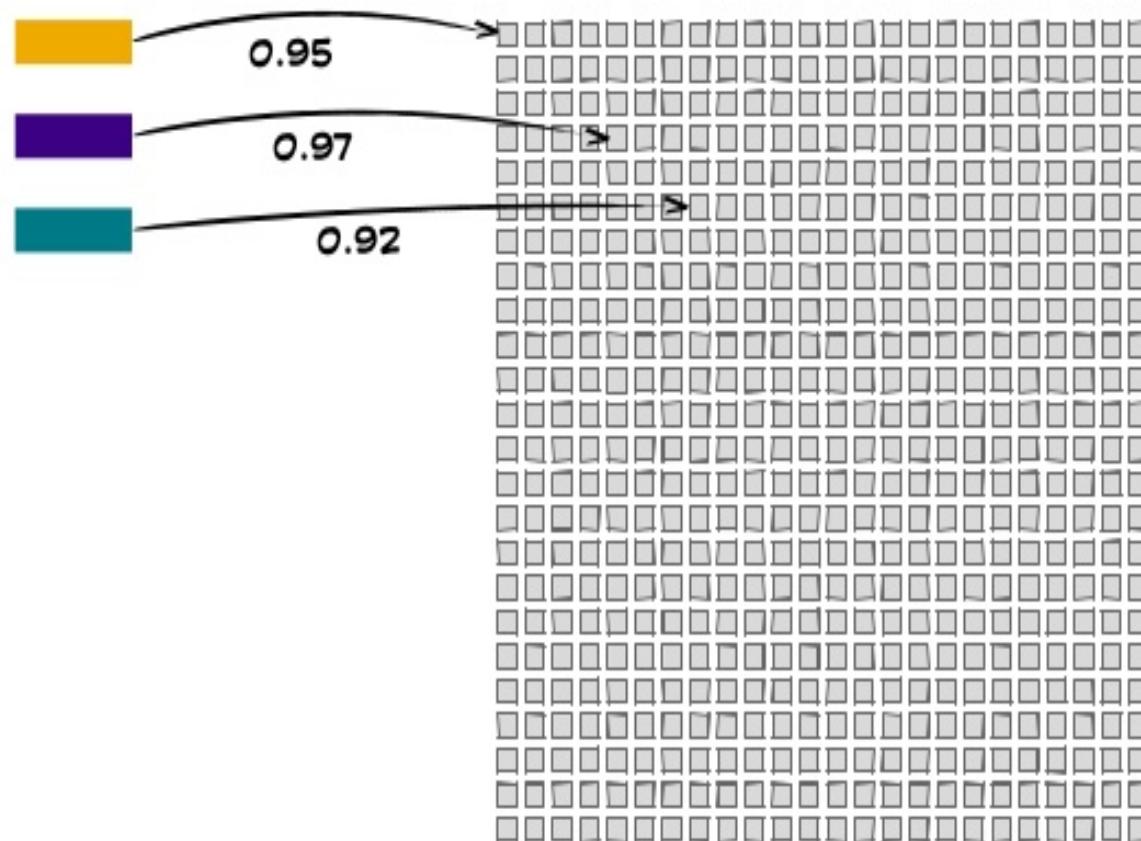
Outliers in log data



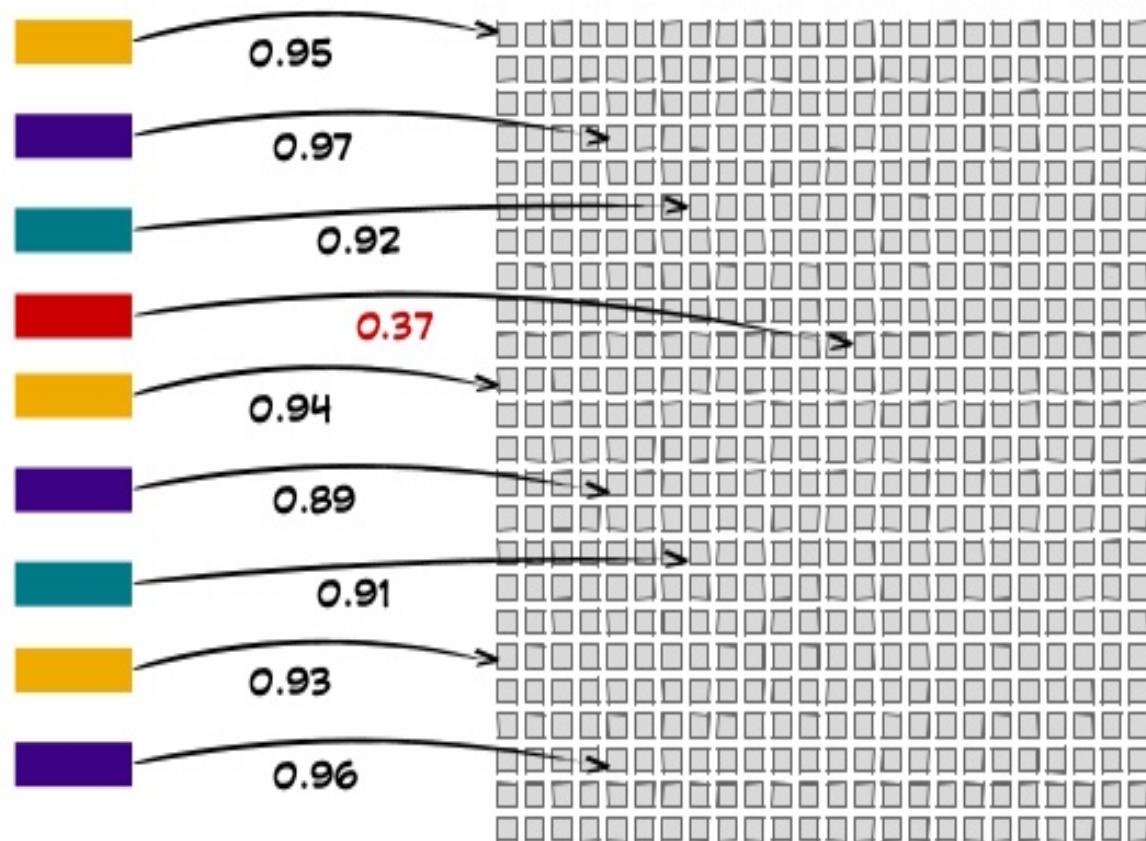
Outliers in log data



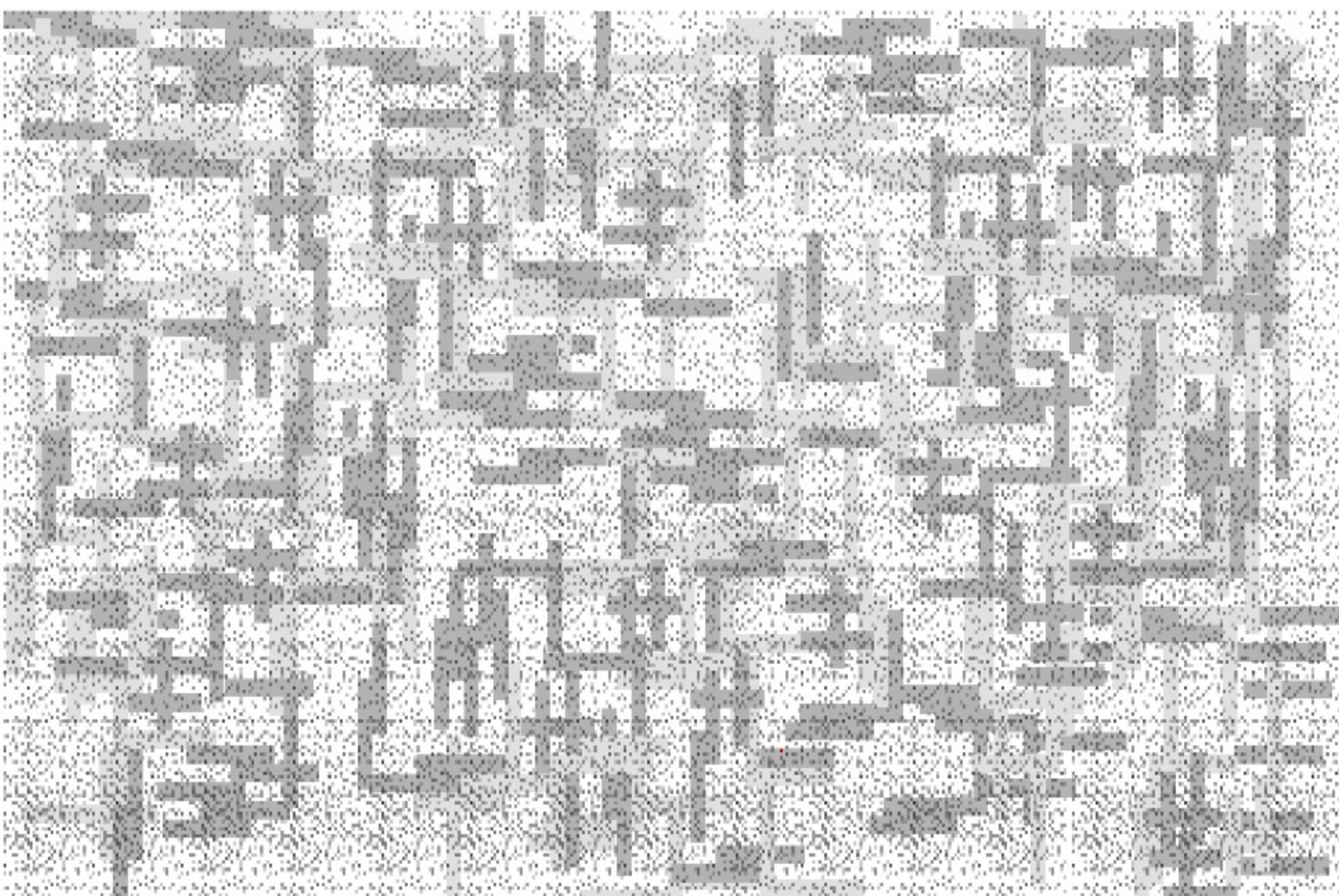
Outliers in log data



Outliers in log data

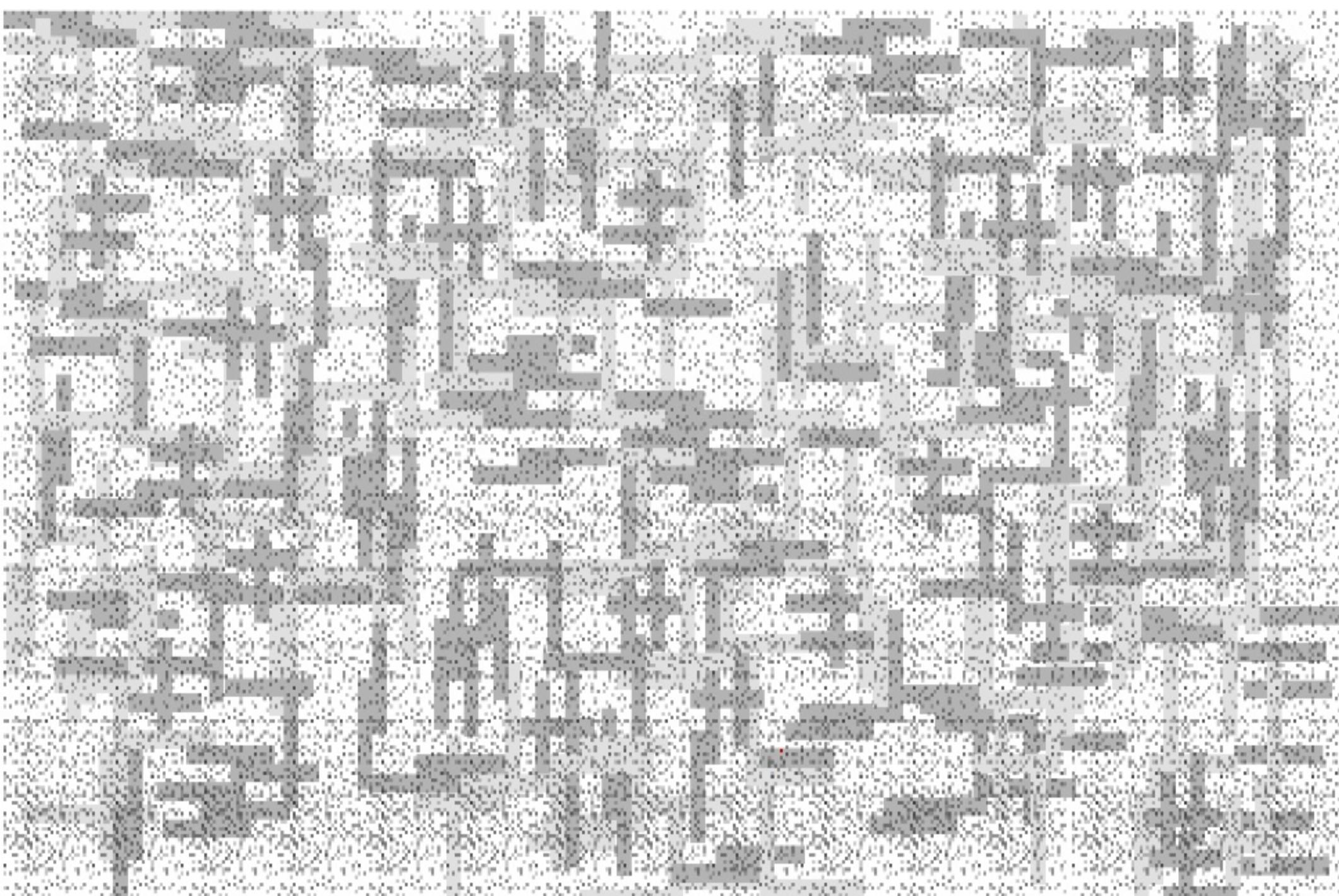


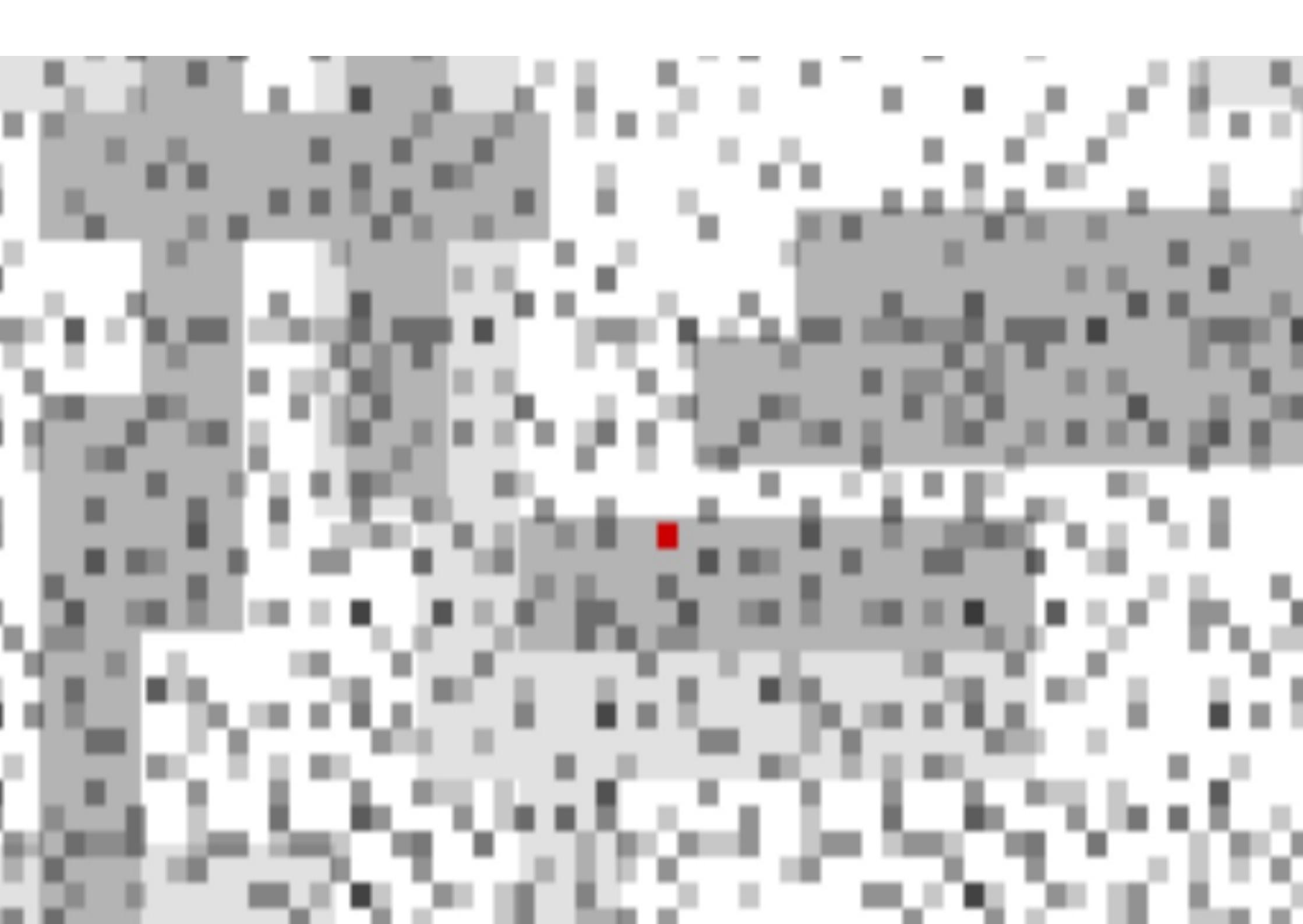
An **outlier** is any record whose best match was at least 4σ below the mean.





**Out of 310 million log
records, we identified
0.0012% as outliers.**





Thirty most extreme outliers

10 Can not communicate with power supply 2.

9 Power supply 2 failed.

8 Power supply redundancy is lost.

1 Drive A is removed.

1 Can not communicate with power supply 1.

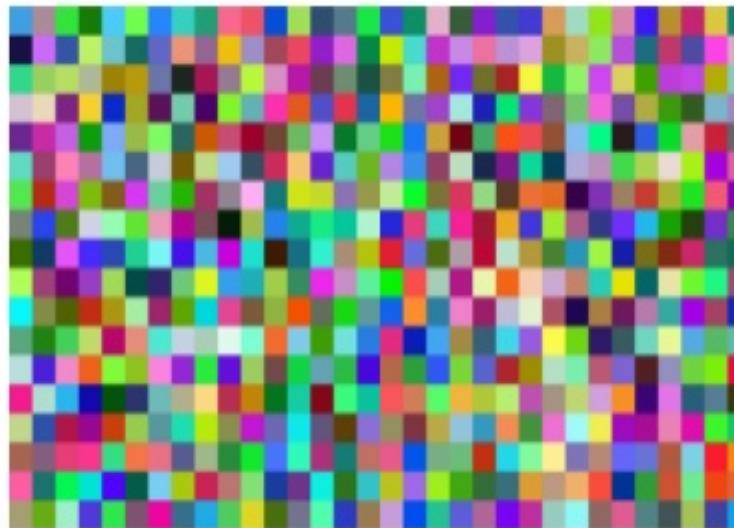
1 Power supply 1 failed.

SOM TRAINING in SPARK

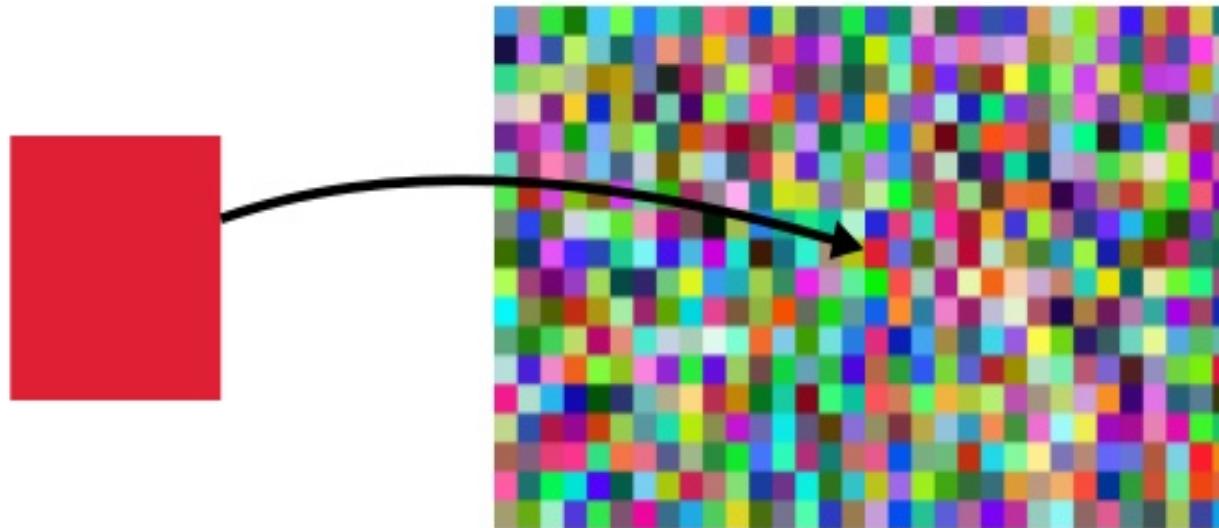
On-line SOM training



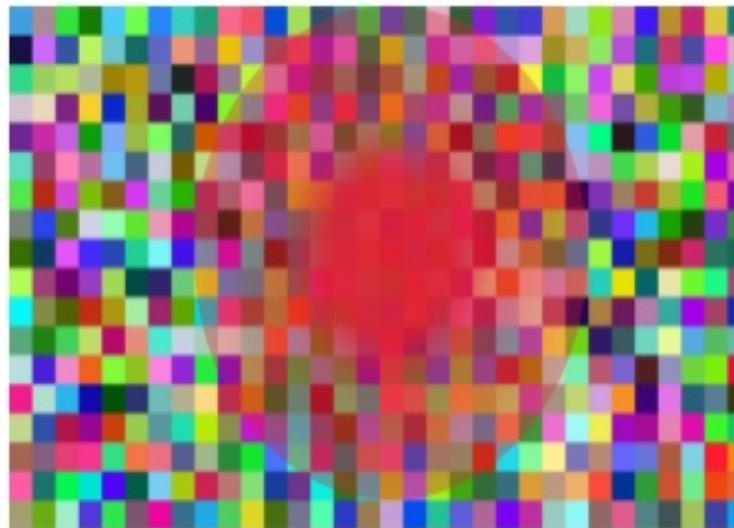
On-line SOM training



On-line SOM training



On-line SOM training



On-line SOM training

```
while t < iterations:  
    for ex in examples:  
        t = t + 1  
        if t == iterations:  
            break  
        bestMatch = closest(somt, ex)  
        for (unit, wt) in neighborhood(bestMatch, sigma(t)):  
            somt+1[unit] = somt[unit] + ex * alpha(t) * wt
```

On-line SOM training

```
while t < iterations:  
    for ex in examples:  
        t = t + 1  
        if t == iterations:  
            break  
        bestMatch = closest(somt, ex)  
        for (unit, wt) in neighborhood(bestMatch, sigma(t)):  
            somt+1[unit] = somt[unit] + ex * alpha(t) * wt
```

at each step, we update each unit by
adding its value from the previous step...

On-line SOM training

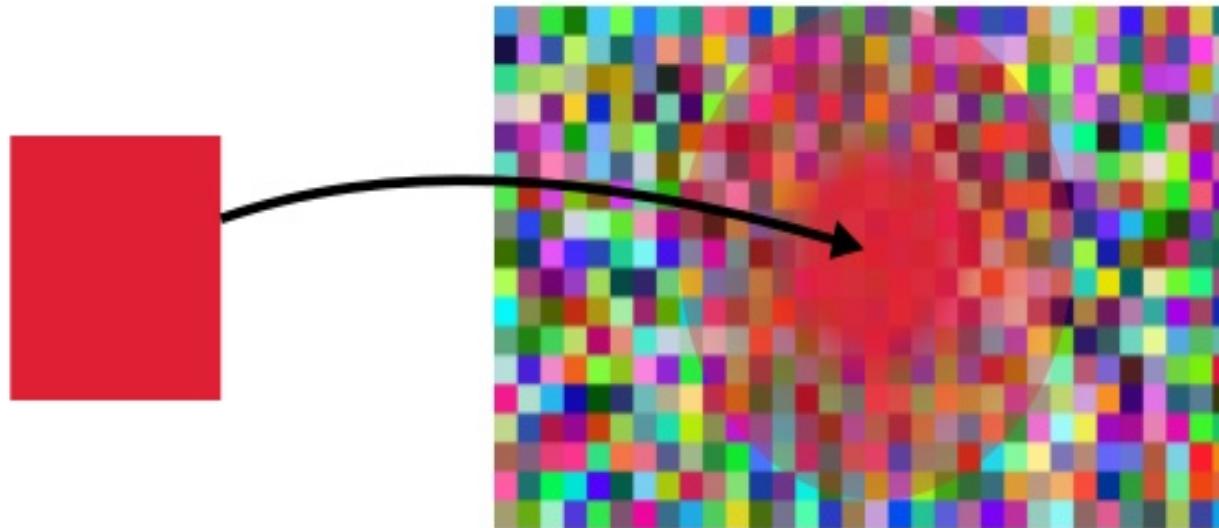
```
while t < iterations:  
    for ex in examples:  
        t = t + 1  
        if t == iterations:  
            break  
        bestMatch = closest(somt, ex)  
        for (unit, wt) in neighborhood(bestMatch, sigma(t)):  
            somt+1[unit] = somt[unit] + ex * alpha(t) * wt
```

to the example that we considered...

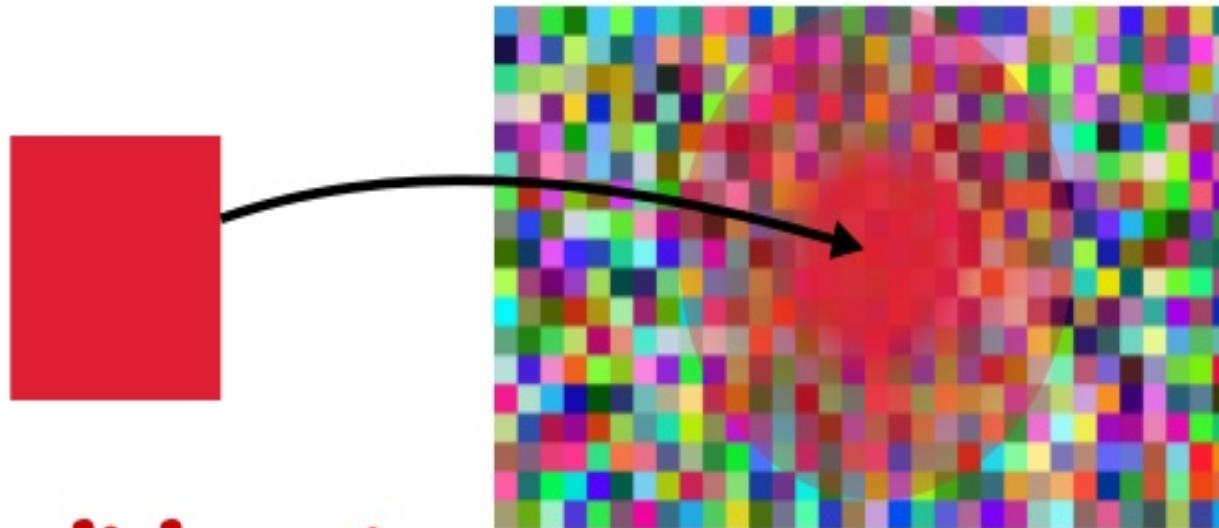
On-line SOM training

```
while t < iterations:  
    for ex in examples:  
        t = t + 1  
        if t == iterations:  
            break  
        bestMatch = closest(somt, ex)  
        for (unit, wt) in neighborhood(bestMatch, sigma(t)):  
            somt+1[unit] = somt[unit] + ex * alpha(t) * wt  
                                         scaled by a learning factor and the  
                                         distance from this unit to its best match
```

On-line SOM training



On-line SOM training



not parallel

sensitive to
example order

sensitive to
learning rate

Batch SOM training

```
for t in (1 to iterations):
    state = newState()
    for ex in examples:
        bestMatch = closest(somt-1, ex)
        hood = neighborhood(bestMatch, sigma(t))
        state.matches += ex * hood
        state.hoods += hood
    somt = newSOM(state.matches / state.hoods)
```

Batch SOM training

```
for t in (1 to iterations):
    state = newState()
    for ex in examples:
        bestMatch = closest(somt-1, ex)
        hood = neighborhood(bestMatch, sigma(t))
        state.matches += ex * hood
        state.hoods += hood
    somt = newSOM(state.matches / state.hoods)
```

update the state of every cell in the neighborhood
of the best matching unit, weighting by distance

Batch SOM training

```
for t in (1 to iterations):
    state = newState()
    for ex in examples:
        bestMatch = closest(somt-1, ex)
        hood = neighborhood(bestMatch, sigma(t))
        state.matches += ex * hood
        state.hoods += hood
    somt = newSOM(state.matches / state.hoods)
    keep track of the distance weights
    we've seen for a weighted average
```

Batch SOM training

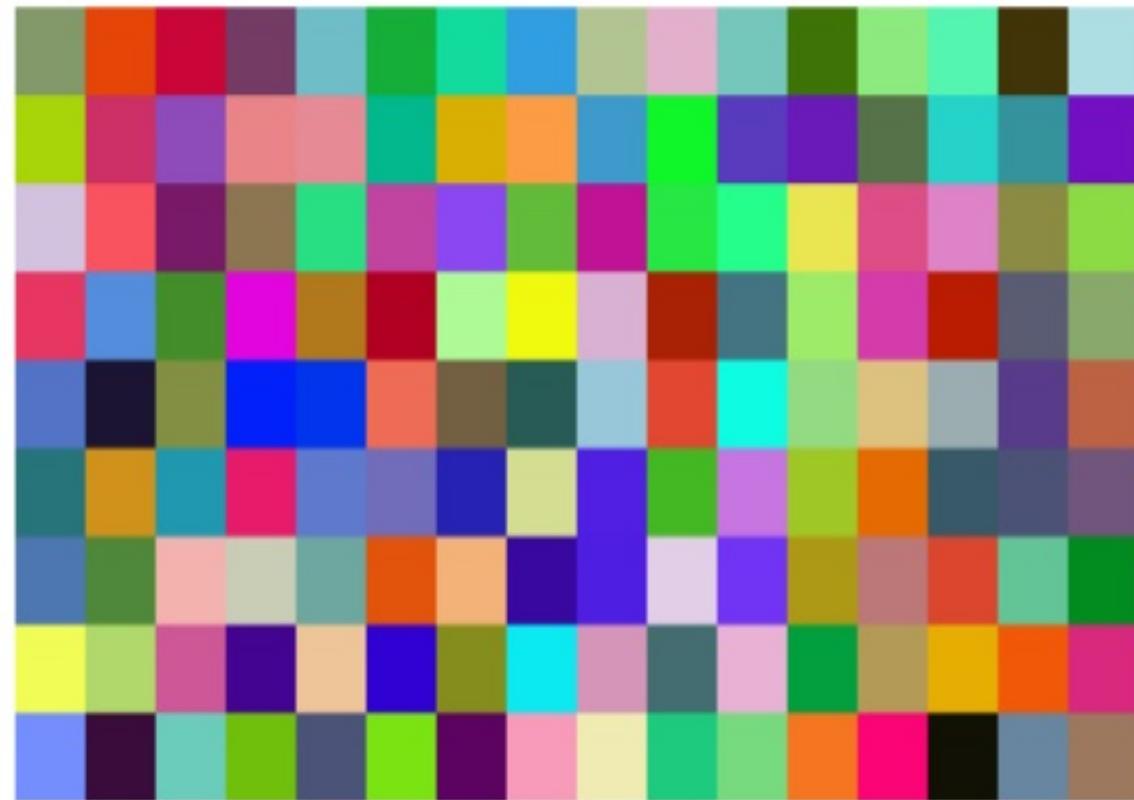
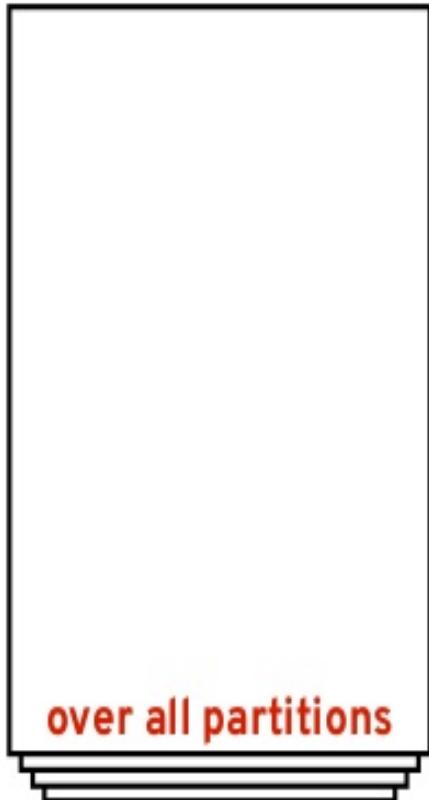
```
for t in (1 to iterations):
    state = newState()
    for ex in examples:
        bestMatch = closest(somt-1, ex)
        hood = neighborhood(bestMatch, sigma(t))
        state.matches += ex * hood
        state.hoods += hood
    somt = newSOM(state.matches / state.hoods)
```

since we can easily merge multiple states, we

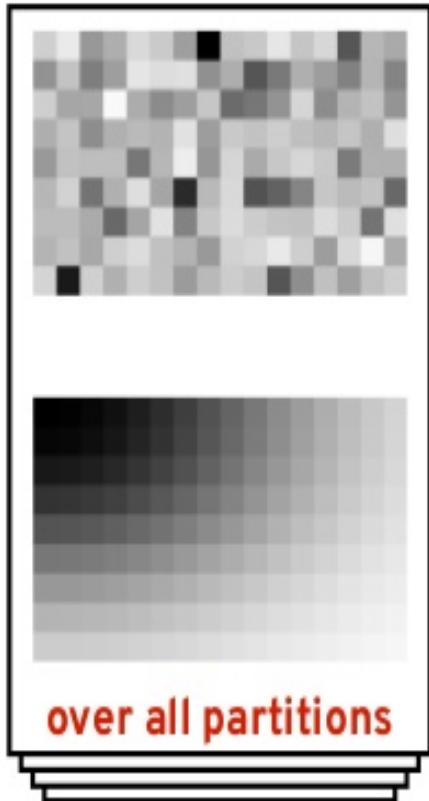
Batch SOM training



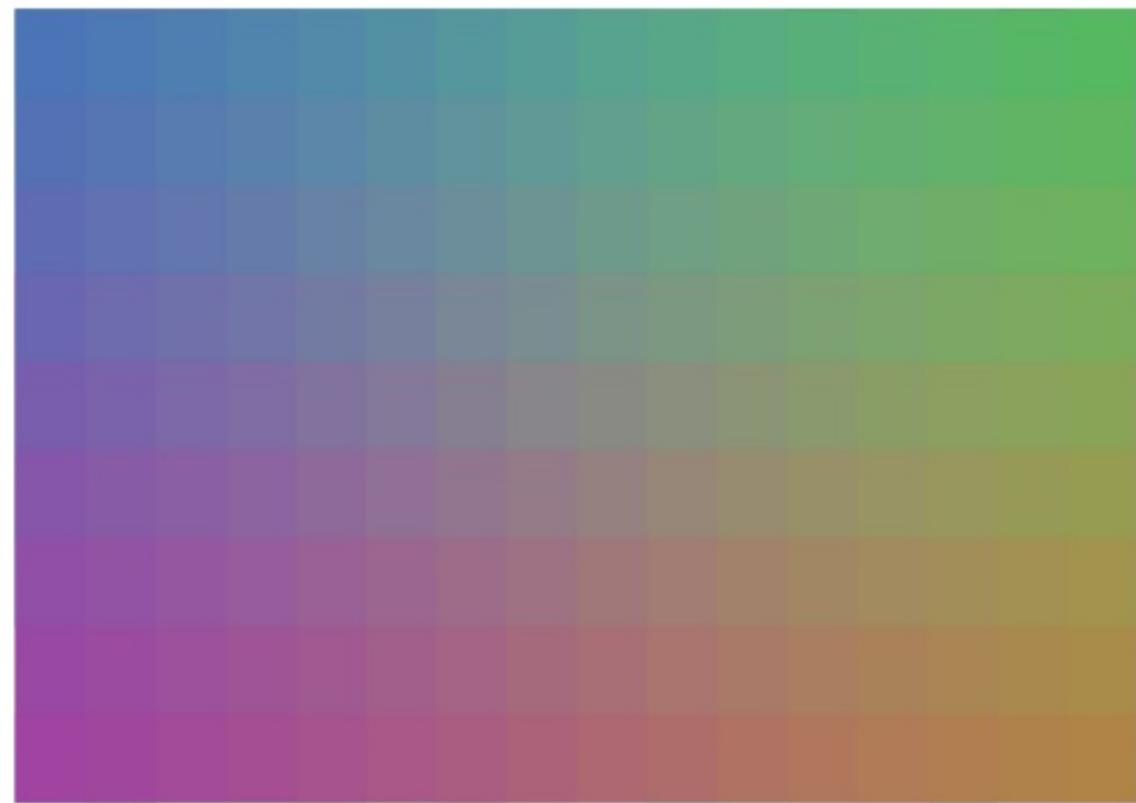
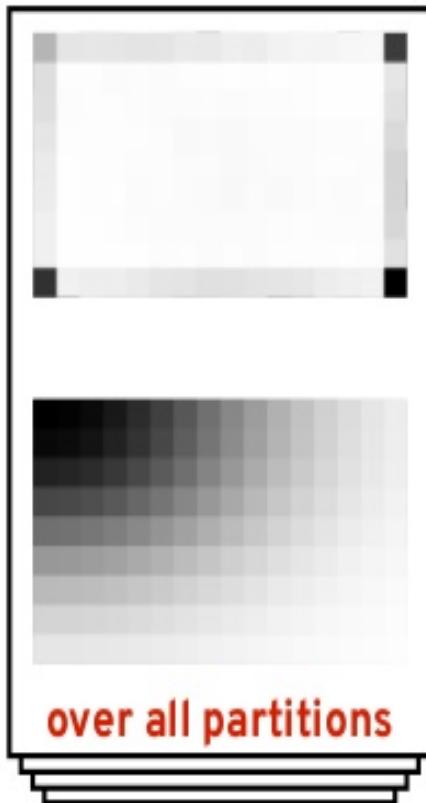
Batch SOM training



Batch SOM training



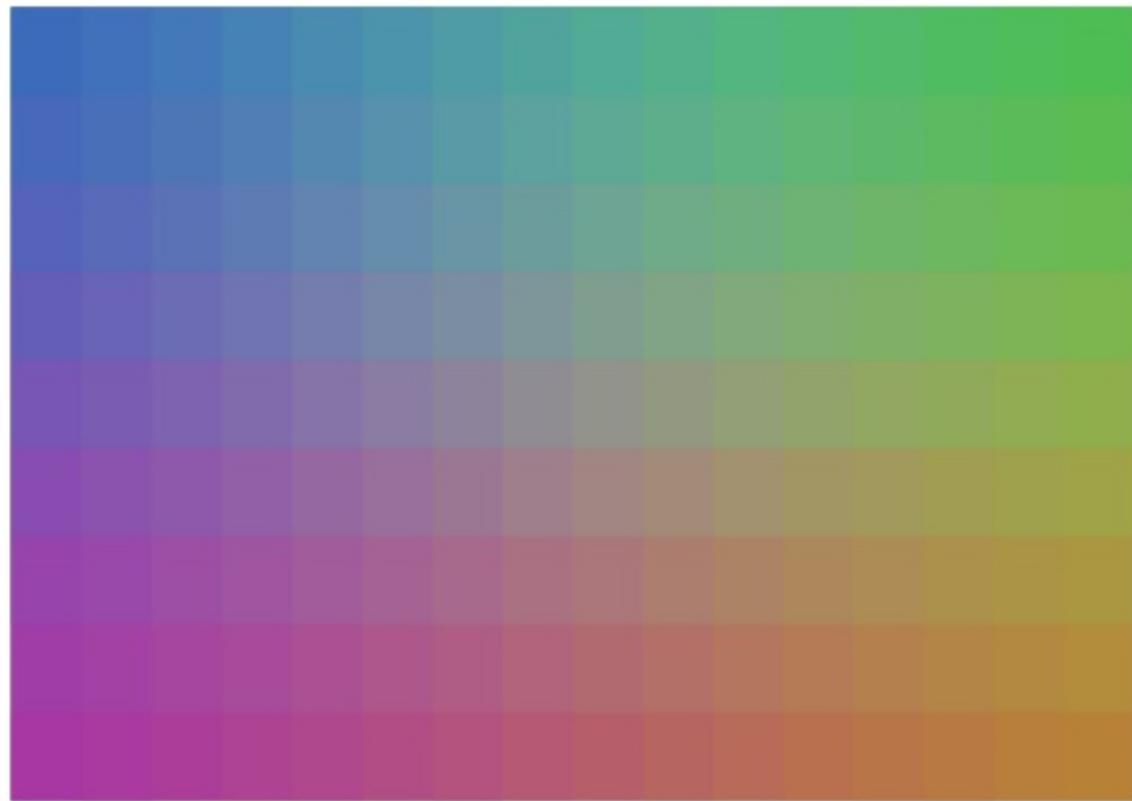
Batch SOM training



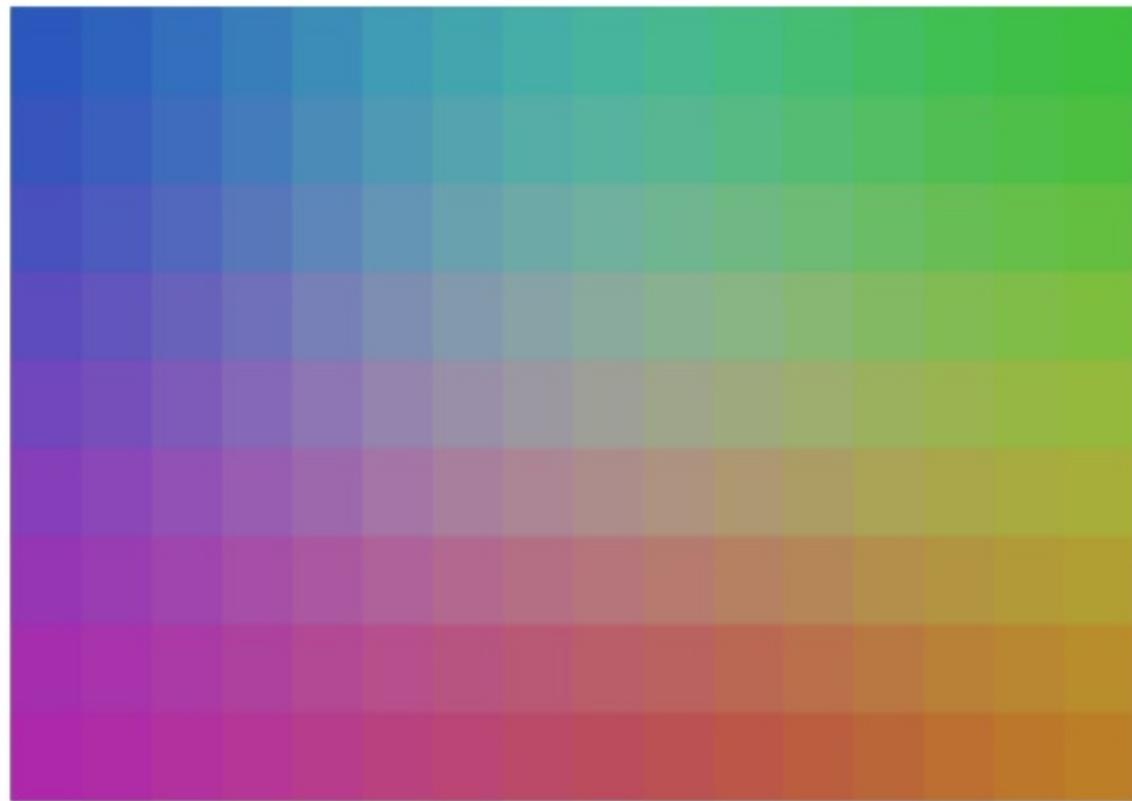
Batch SOM training



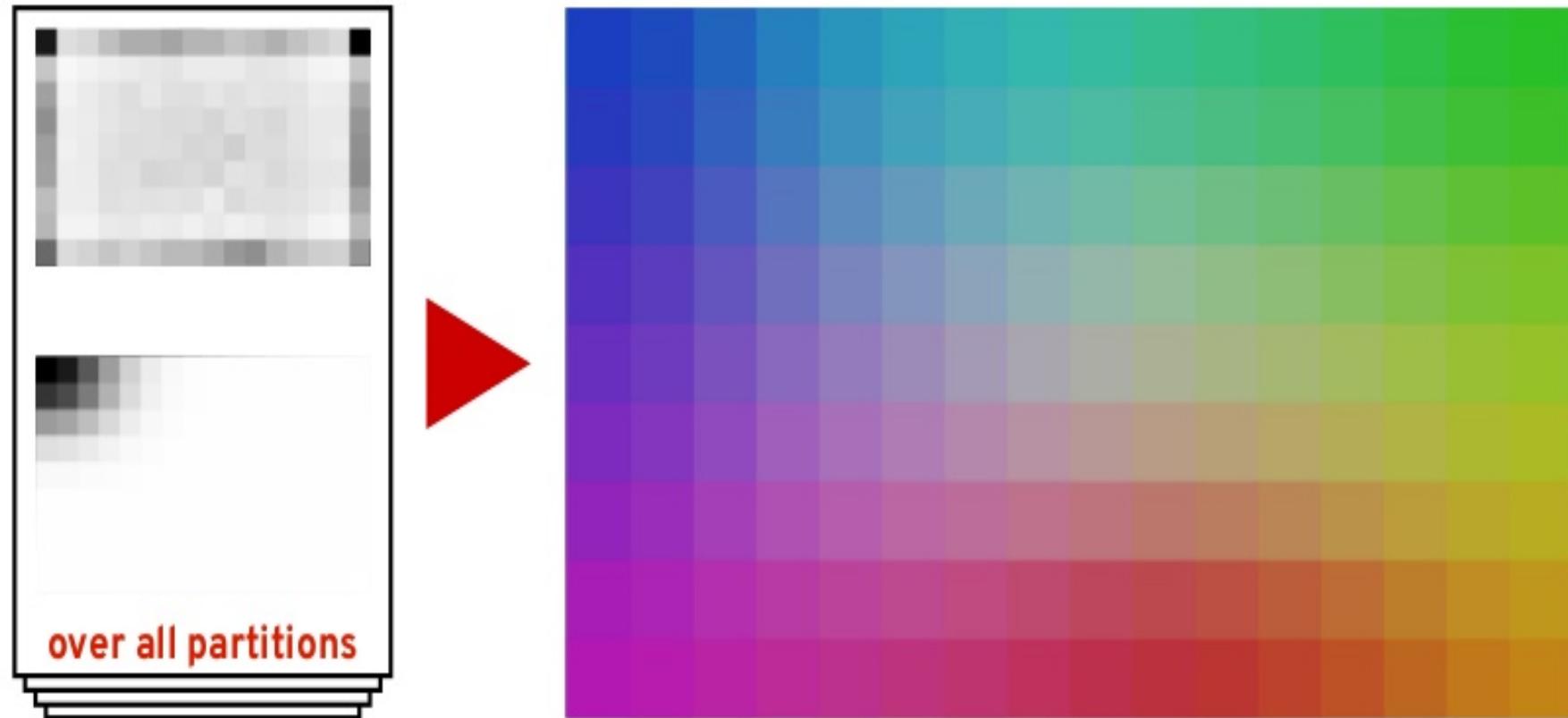
over all partitions



Batch SOM training



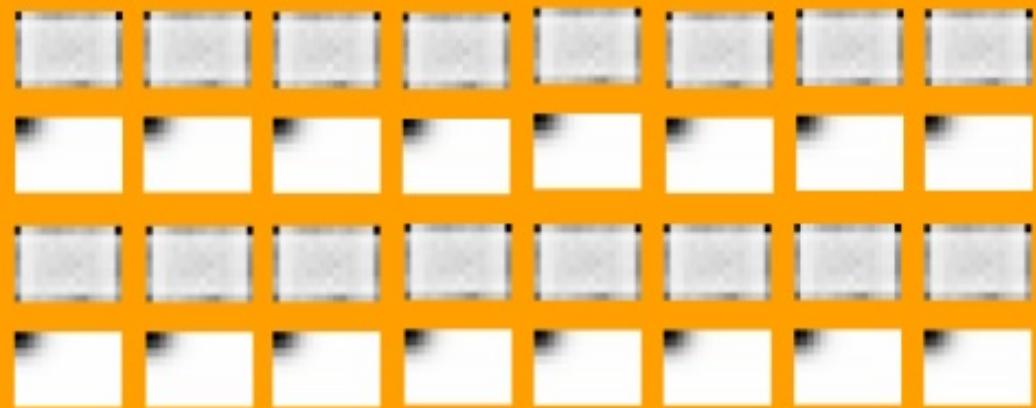
Batch SOM training



driver (using aggregate)

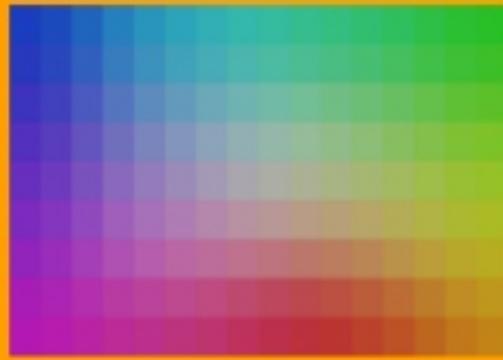


workers



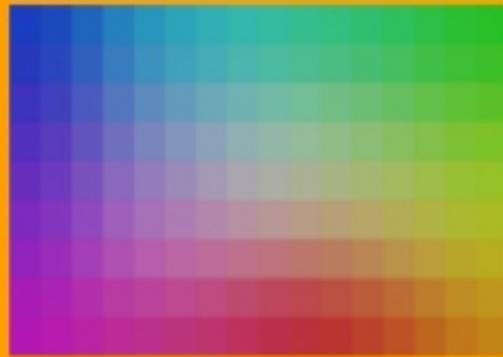
driver (using aggregate)

workers



driver (using aggregate)

workers

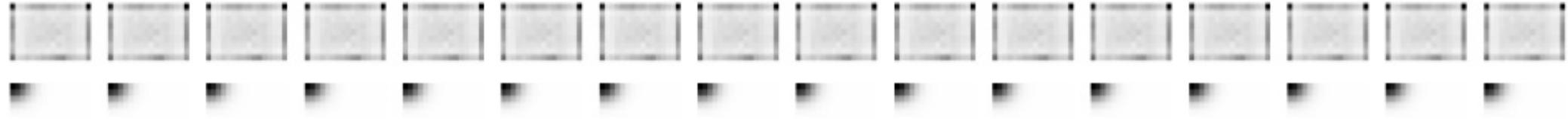


driver (using aggregate)

What if you have a 3 mb model and 2,048 partitions?

workers

driver (using treeAggregate)



workers

driver (using treeAggregate)

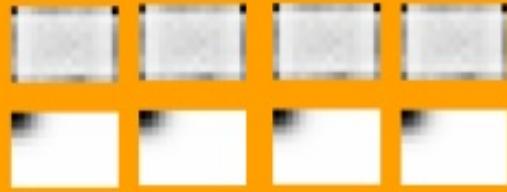


workers

driver (using treeAggregate)

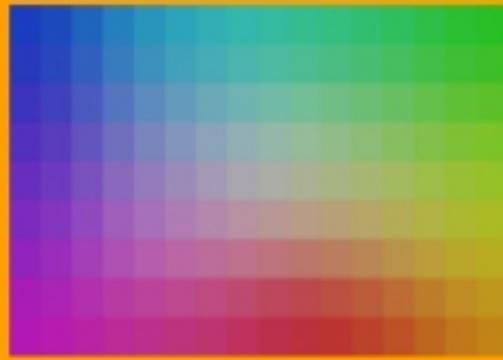


workers



driver (using treeAggregate)

workers



driver (using treeAggregate)

workers

SHARING MODELS BEYOND SPARK

Sharing models

```
class Model(private var entries: breeze.linalg.DenseVector[Double],  
            /* ... lots of (possibly) mutable state ... */ )  
  implements java.io.Serializable {  
  
  // lots of implementation details here  
  
}
```

Sharing models

```
case class FrozenModel(entries: Array[Double], /* ... */) { }

class Model(private var entries: breeze.linalg.DenseVector[Double],
            /* ... lots of (possibly) mutable state ... */)
    implements java.io.Serializable {

    // lots of implementation details here

}
```

Sharing models

```
case class FrozenModel(entries: Array[Double], /* ... */) { }

class Model(private var entries: breeze.linalg.DenseVector[Double],
            /* ... lots of (possibly) mutable state ... */)
    implements java.io.Serializable {

    // lots of implementation details here

    def freeze: FrozenModel = // ...
}

object Model {
    def thaw(im: FrozenModel): Model = // ...
}
```

Sharing models

```
import org.json4s.jackson.Serialization
import org.json4s.jackson.Serialization.{read=>jread, write=>jwrite}
implicit val formats = Serialization.formats(NoTypeHints)

def toJson(m: Model): String = {
    jwrite(som.freeze)
}

def fromJson(json: String): Try[Model] = {
    Try({
        Model.thaw(jread[FrozenModel](json))
    })
}
```

Sharing models

```
import org.json4s.jackson.Serialization
import org.json4s.jackson.Serialization.{read=>jread, write=>jwrite}
implicit val formats = Serialization.formats(NoTypeHints)

def toJson(m: Model): String = {
    jwrite(som.freeze)
}

def fromJson(json: String): Try[Model] = {
    Try({
        Model.thaw(jread[FrozenModel](json))
    })
}
```

Also consider how you'll share feature encoders and other parts of your learning pipeline!

PRACTICAL MATTERS

Spark and ElasticSearch

Data locality is an issue and caching is even more important than when running from local storage.

If your data are write-once, consider exporting ES indices to Parquet files and analyzing those instead.

Structured queries in Spark

Always program defensively: mediate schemas,
explicitly convert null values, etc.

**Use the Dataset API whenever possible to minimize
boilerplate and benefit from query planning without
(entirely) forsaking type safety.**

Memory and partitioning

Large JVM heaps can lead to appalling GC pauses and executor timeouts.

Use multiple JVMs or off-heap storage (in Spark 2.0!)

Tree aggregation can save you both memory and execution time by partially aggregating at worker nodes.

Interoperability

Avoid brittle or language-specific model serializers
when sharing models with non-Spark environments.

JSON is imperfect but ubiquitous. However, json4s
will serialize case classes for free!

See also [SPARK-13944](#), merged recently into 2.0.

Feature engineering

Favor feature engineering effort over complex or novel learning algorithms.

Prefer approaches that train interpretable models.

Design your feature engineering pipeline so you can translate feature vectors back to factor values.

THANKS!

@willb • willb@redhat.com
<https://chapeau.freevariable.com>