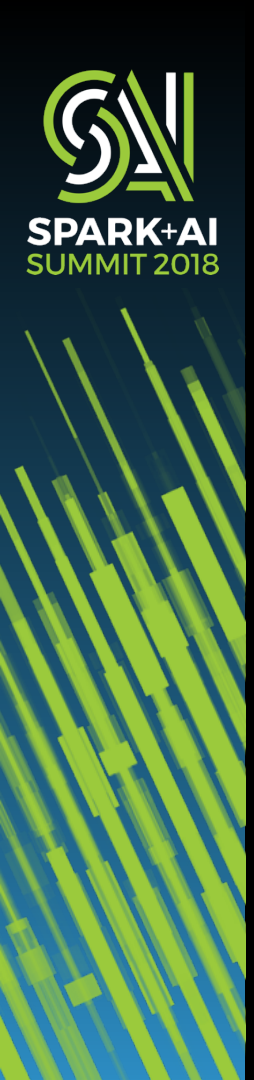


Predictive Maintenance at the Dutch Railways

Air leakage detection in train breaking pipes

Ivo Everts, GoDataDriven (Amsterdam, NL)

#DSSAIS16

GoDataDriven

proudly part of Xebia Group

Tünde Alkemade
Ivo Everts
Jelte Hoekstra
Giovanni Lanzani
Janelle Zoutkamp

Stefan Hendriks
Wouter Hordijk
Inge Kalsbeek
Wan-Jui Lee
Inka Locht
Kee Man
Nick Oosterhof
Margot Peters
Cyriana Roelofs
Mattijs Suurland

GoDataDriven

proudly part of Xebia Group

Tünde Alkemade
Ivo Everts
Jelte Hoekstra
Giovanni Lanzani
Janelle Zoutkamp

Stefan Hendriks
Wouter Hordijk
Inge Kalsbeek
Wan-Jui Lee
Inka Locht
Kee Man
Nick Oosterhof
Margot Peters
Cyriana Roelofs
Mattijs Suurland

Published paper in June 2017
'Contextual Air Leakage Detection in Train Braking Pipes'
@ 'Advances in Artificial Intelligence: From Theory to Practice'

GoDataDriven

proudly part of Xebia Group

Tünde Alkemade
Ivo Everts
Jelte Hoekstra
Giovanni Lanzani
Janelle Zoutkamp

Stefan Hendriks
Wouter Hordijk
Inge Kalsbeek
Wan-Jui Lee
Inka Locht
Kee Man
Nick Oosterhof
Margot Peters
Cyriana Roelofs
Mattijs Suurland

Published paper in June 2017
'Contextual Air Leakage Detection in Train Braking Pipes'
@ 'Advances in Artificial Intelligence: From Theory to Practice'

Implemented the paper using PySpark; teamwork for data delivery, proper productionizing, way of working

GoDataDriven

proudly part of Xebia Group

Tünde Alkemade
Ivo Everts
Jelte Hoekstra
Giovanni Lanzani
Janelle Zoutkamp

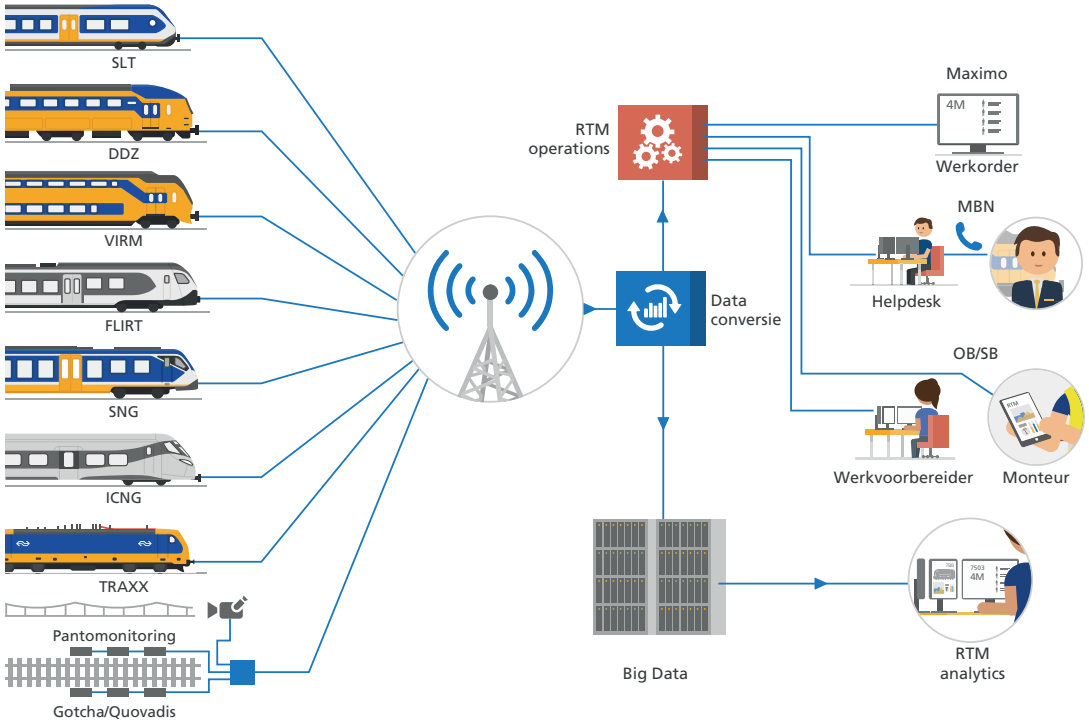
Consulting data scientist @ GoDataDriven
MSc in AI; PhD in Computer Vision
Did lot of scientific programming
Now mostly Python, Hadoop, Spark, Keras

Stefan Hendriks
Wouter Hordijk
Inge Kalsbeek
Wan-Jui Lee
Inka Locht
Kee Man
Nick Oosterhof
Margot Peters
Cyriana Roelofs
Mattijs Suurland

Published paper in June 2017
'Contextual Air Leakage Detection in Train Braking Pipes'
@ 'Advances in Artificial Intelligence: From Theory to Practice'

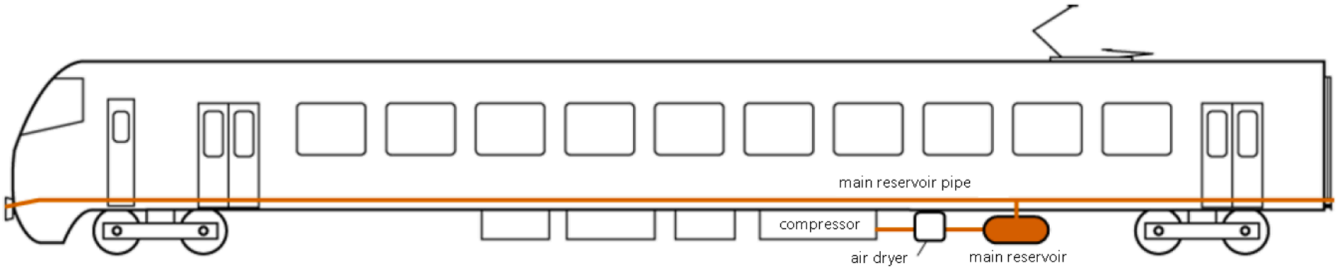
Implemented the paper using PySpark; teamwork for data delivery, proper productionizing, way of working

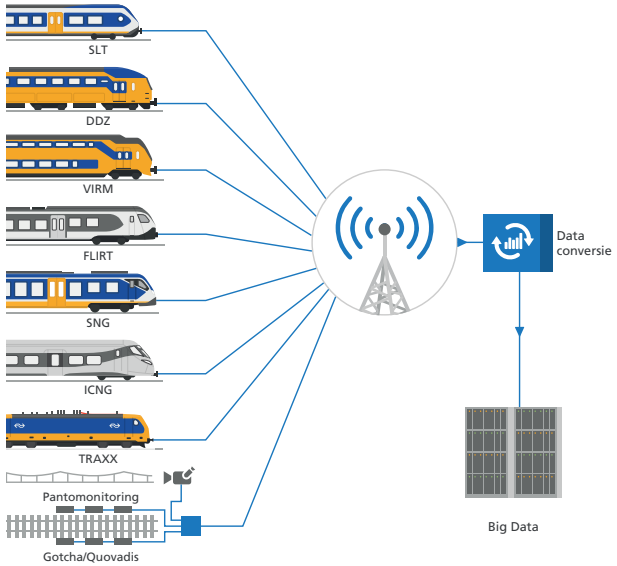
Big Data @ NS (Dutch Railways)



Outline

- Data ingestion
- Air leakage detection
- Other use cases





Collecting data for ***R**eal **T**ime **M**onitoring*

Data ingestion

Data ingestion

Real time data

~20GB per day; 22B records

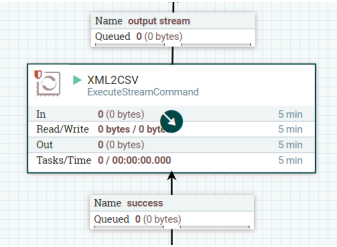
Historic data

22B records

Data ingestion

Real time data

~20GB per day; 22B records



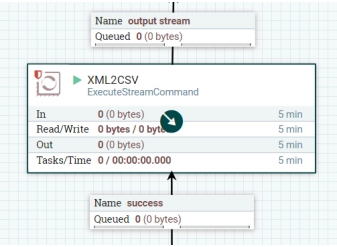
Historic data

22B records

Data ingestion

Real time data

~20GB per day; 22B records



Property	Value
Command Arguments	rewrite-xml.py;${hiveTable};${origin}
Command Path	**python**
Ignore STDIN	false

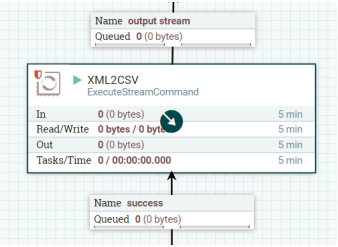
Historic data

22B records

Data ingestion

Real time data

~20GB per day; 22B records



Property	Value
Command Arguments	rewrite-xml.py;${hiveTable};${origin}
Command Path	python
Ignore STDIN	false

```
# fetch args
table_name = sys.argv[1]
origin = sys.argv[2]

# read xml string from stdin
xml_string = ''.join(sys.stdin.readlines())

# parse 'm and print to stdout
data = xml2table(xml_string, table_name, origin, start_dt, end_dt)
print(data.to_csv(encoding='utf-8', header=False, index=False,
```

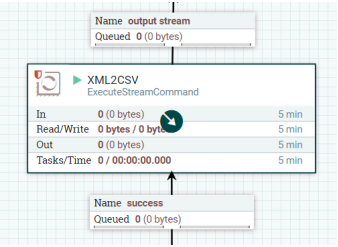
Historic data

22B records

Data ingestion

Real time data

~20GB per day; 22B records



Property	Value
Command Arguments	rewrite-xml.py;${hiveTable};${origin}
Command Path	python
Ignore STDIN	false

```
# fetch args
table_name = sys.argv[1]
origin = sys.argv[2]

# read xml string from stdin
xml_string = ''.join(sys.stdin.readlines())

# parse 'm and print to stdout
data = xml2table(xml_string, table_name, origin, start_dt, end_dt)
print(data.to_csv(encoding='utf-8', header=False, index=False,
```

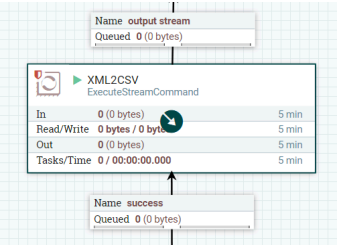
Historic data

22B records

Data ingestion

Real time data

~20GB per day; 22B records



Property	Value
Command Arguments	rewrite-xml.py;${hiveTable};${origin}
Command Path	python
Ignore STDIN	false

```
# fetch args
table_name = sys.argv[1]
origin = sys.argv[2]

# read xml string from stdin
xml_string = ''.join(sys.stdin.readlines())

# parse 'm and print to stdout
data = xml2table(xml_string, table_name, origin, start_dt, end_dt)
print(data.to_csv(encoding='utf-8', header=False, index=False,
```

Historic data

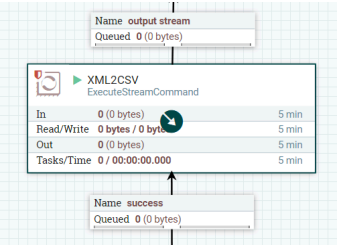
22B records

```
def get_history_df(hdfs_binary_files, output_table_name):
    rdd = (hdfs_binary_files
           .map(unzip_xml)
           .flatMap(lambda x: xml2rows(x, output_table_name)))
    return None if rdd.isEmpty() else rdd.toDF()

...
...

# read binary files and parse 'm
hdfs_binary_files = (sc.binaryFiles('{}*'.format(file_group))
                     .persist(StorageLevel.MEMORY_AND_DISK_SER)
                     .repartition(num_executors))
sdf = get_history_df(hdfs_binary_files, args.output_table_name)
```

Data ingestion

Real time data

~20GB per day; 22B records



Property	Value
Command Arguments	rewrite-xml.py;${hiveTable};${origin}
Command Path	python
Ignore STDIN	false

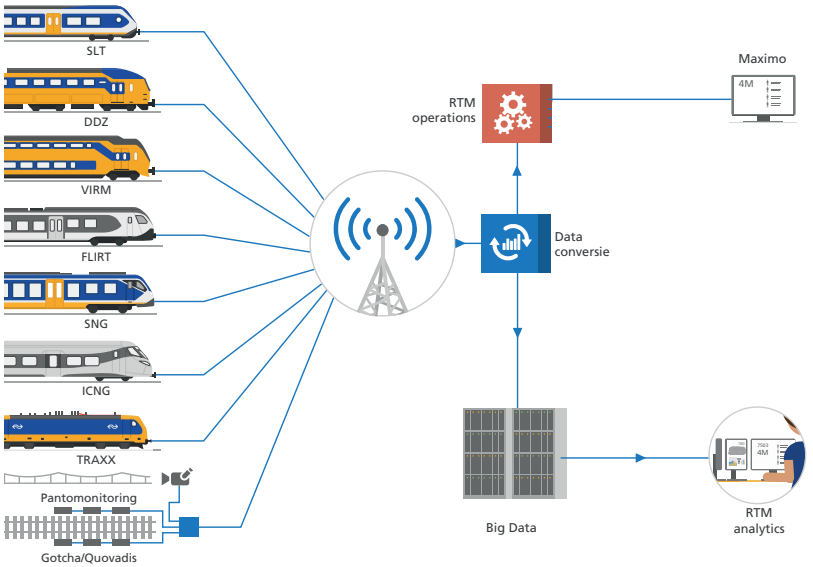
```
# fetch args
table_name = sys.argv[1]
origin = sys.argv[2]

# read xml string from stdin
xml_string = ''.join(sys.stdin.readlines())

# parse 'm and print to stdout
data = xml2table(xml_string, table_name, origin, start_dt, end_dt)
print(data.to_csv(encoding='utf-8', header=False, index=False,
```

Historic data

22B records

```
def get_history_df(hdfs_binary_files, output_table_name):
    rdd = (hdfs_binary_files
           .map(unzip_xml)
           .flatMap(lambda x: xml2rows(x, output_table_name)))
    return None if rdd.isEmpty() else rdd.toDF()

...
...

# read binary files and parse 'm
hdfs_binary_files = (sc.binaryFiles('{}*'.format(file_group))
                     .persist(StorageLevel.MEMORY_AND_DISK_SER)
                     .repartition(num_executors))
sdf = get_history_df(hdfs_binary_files, args.output_table_name)
```

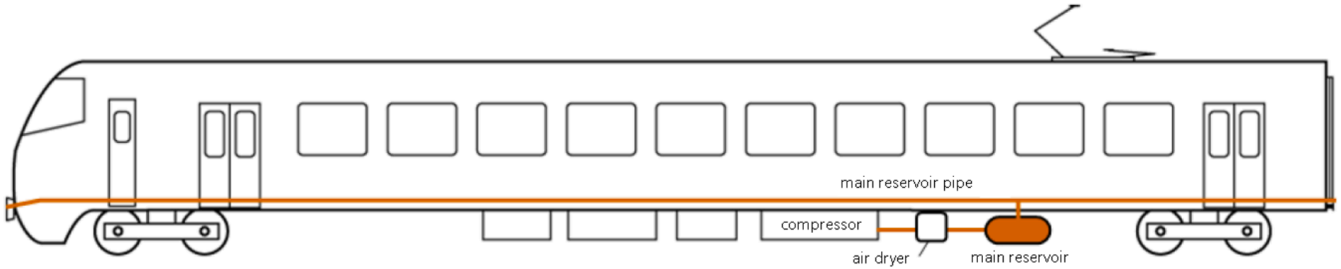
>> Better suited for large datasets
>> Easier to (re)partition and append
>> Preference for a unified approach



One of the cases using RTM data

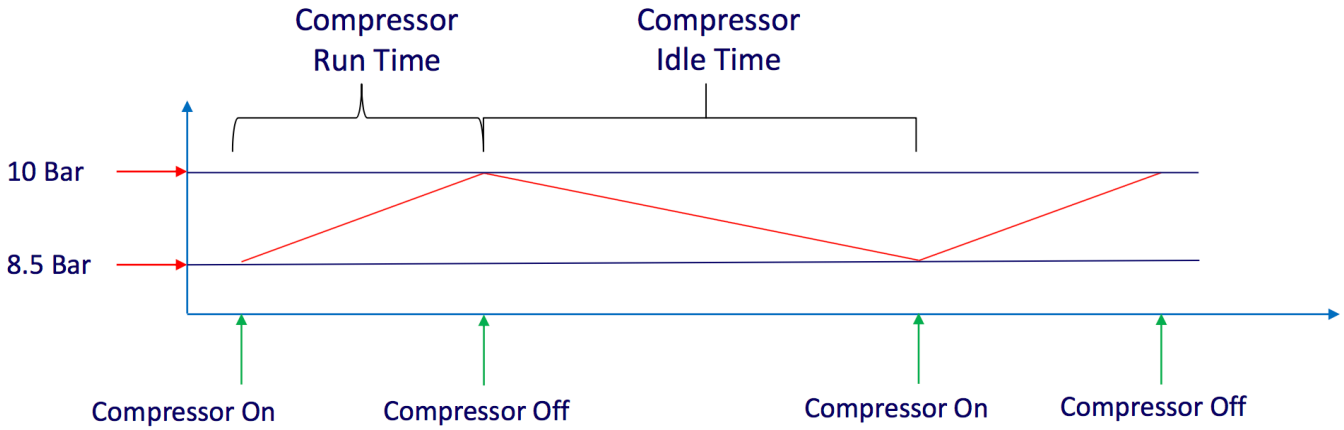
Air leakage detection

Air leakage detection



- **Data** compressor logs
- **Detection** classification
- **Diagnosis** clustering
- **Prognosis** regression
- **Operation** humans

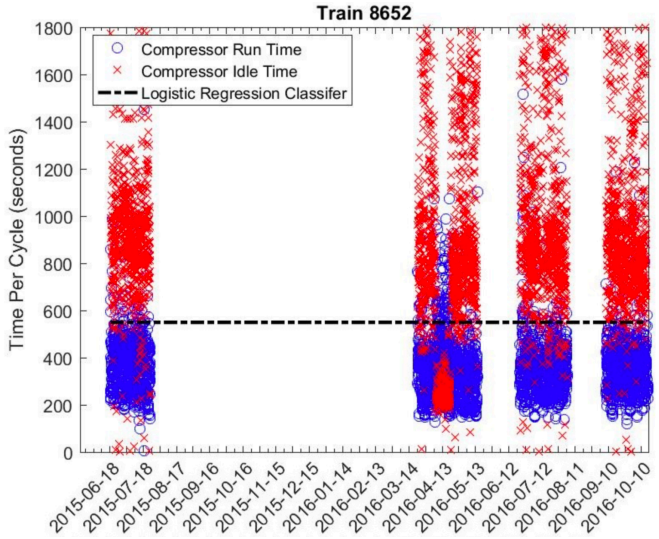
Air leakage detection



We extract median compressor run- and idle- times per hour

- Data
  compressor logs
- Detection
  classification
- Diagnosis
  clustering
- Prognosis
  regression
- Operation
  humans

Air leakage detection

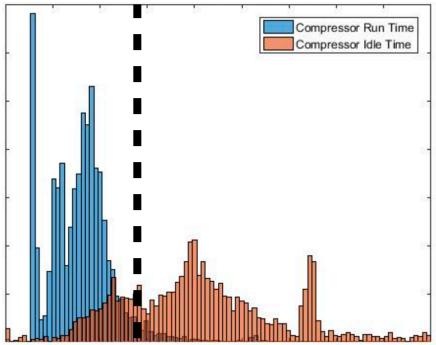


- **Data** compressor logs
- **Detection** classification
- **Diagnosis** clustering
- **Prognosis** regression
- **Operation** humans

Air leakage detection

One model per train =>
compressor durations collected with a `groupBy()`
and modelled in a UDF using scikit-learn
to obtain the per-class probabilities:

$$P(Y = 0|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{m} w_i x_i)}$$



Data
compressor logs

Detection
classification

Diagnosis
clustering

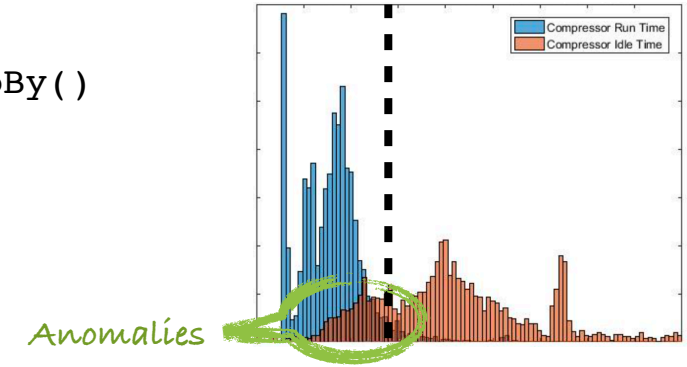
Prognosis
regression

Operation
humans

Air leakage detection

One model per train =>
compressor durations collected with a `groupBy()`
and modelled in a UDF using scikit-learn
to obtain the per-class probabilities:

$$P(Y=0|X) = \frac{1}{1+\exp(w_0 + \sum_{i=1}^{m} w_i x_i)}$$



Data
compressor logs

Detection
classification

Diagnosis
clustering

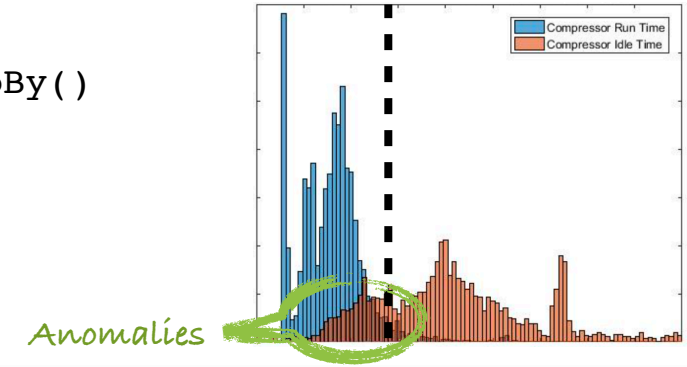
Prognosis
regression

Operation
humans

Air leakage detection

One model per train =>
compressor durations collected with a `groupBy()`
and modelled in a UDF using scikit-learn
to obtain the per-class probabilities:

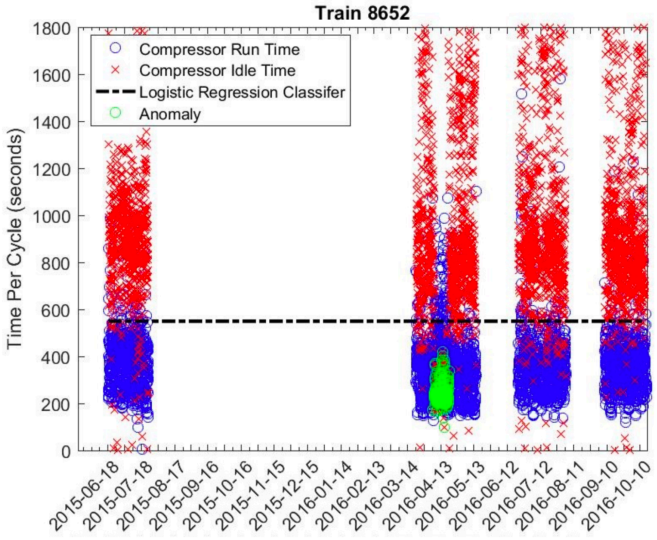
$$P(Y = 0|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{m} w_i x_i)}$$




```
# group data per train and collect data LOL
train_data = (df.groupBy('rolling_stock_number')
                .agg(F.collect_list('median_duration').alias('median_duration'),
                     F.collect_list('label').alias('label')))

# define UDF and fit model
logistic_regression = F.udf(lambda X, y: sklearn_wrapper(X, y), ArrayType(FloatType()))
model_data = (train_data.withColumn('logistic_model_parameters',
                                    logistic_regression(train_data.median_duration,
                                                        train_data.label)))
```

- Data
  compressor logs
- Detection
  classification
- Diagnosis
  clustering
- Prognosis
  regression
- Operation
  humans

Air leakage detection

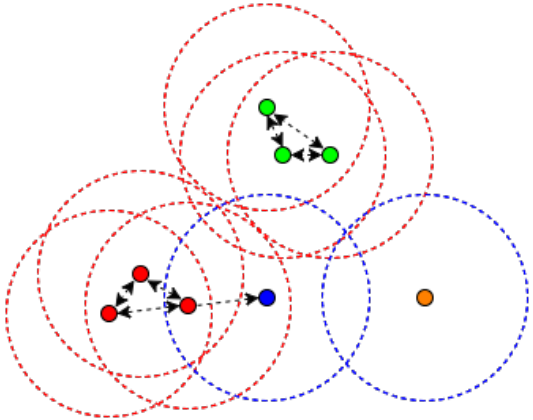


- **Data** — compressor logs
- **Detection** — classification
- **Diagnosis** — clustering
- **Prognosis** — regression
- **Operation** — humans

Air leakage detection

Air leakage is defined as a cluster of anomalies, robust against outliers. For this we use a 'dynamic' variation of dbscan clustering, in a UDF, where the minimal cluster size is dependent on the logistic function:

$$\beta = 2 * minPts * \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{m} w_i x_i)}$$



Data
compressor logs

Detection
classification

Diagnosis
clustering

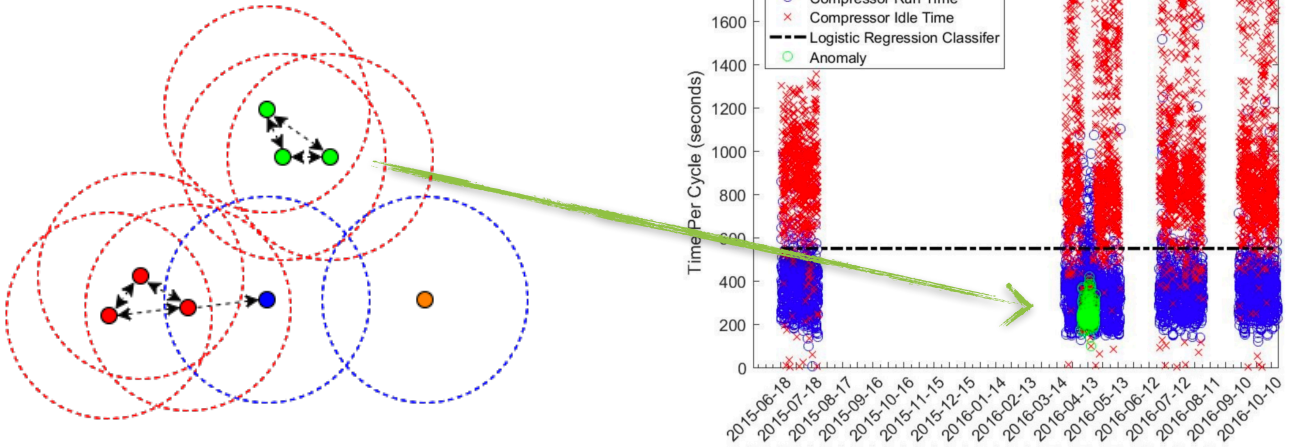
Prognosis
regression

Operation
humans

Air leakage detection

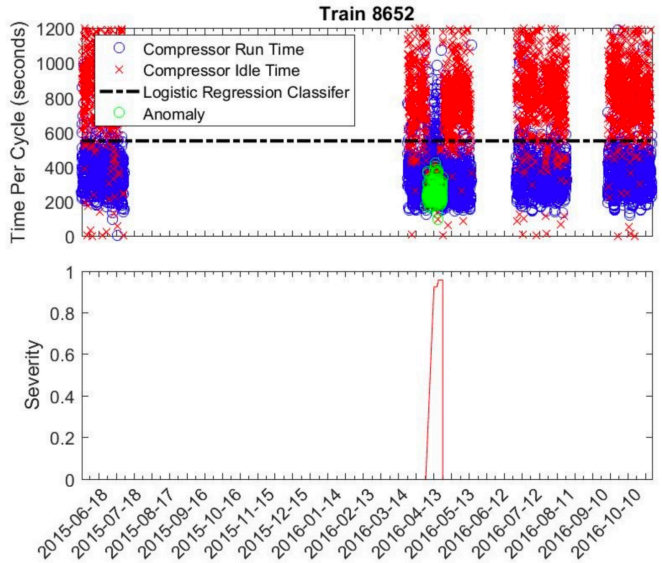
Air leakage is defined as a cluster of anomalies, robust against outliers. For this we use a 'dynamic' variation of dbscan clustering, in a UDF, where the minimal cluster size is dependent on the logistic function:

$$\beta = 2 * minPts * \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{m} w_i x_i)}$$



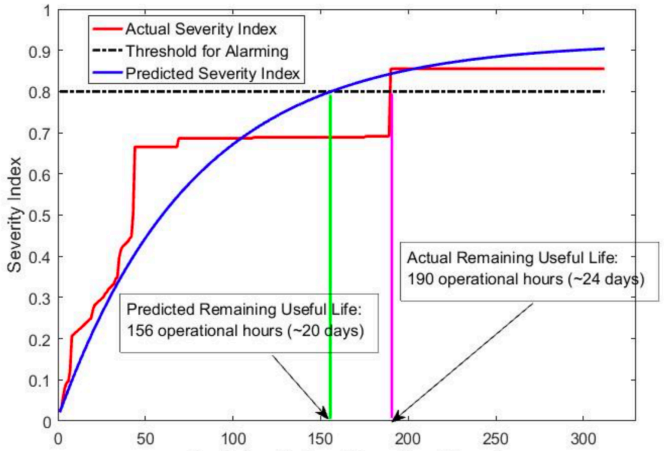
- Data: compressor logs
- Detection: classification
- Diagnosis: clustering
- Prognosis: regression
- Operation: humans

Air leakage detection



- **Data** compressor logs
- **Detection** classification
- **Diagnosis** clustering
- **Prognosis** regression
- **Operation** humans

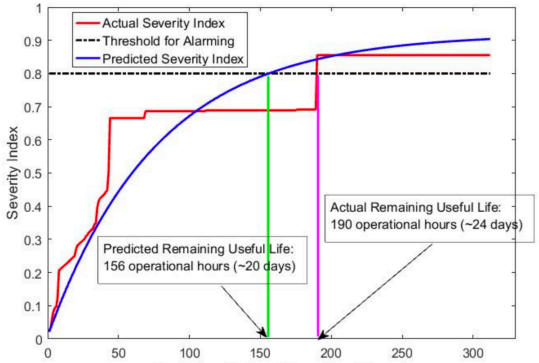
Air leakage detection



- **Data** compressor logs
- **Detection** classification
- **Diagnosis** clustering
- **Prognosis** regression
- **Operation** humans

Air leakage detection

In the current implementation
we train one regressor for all trains,
so a pure Spark implementation is possible



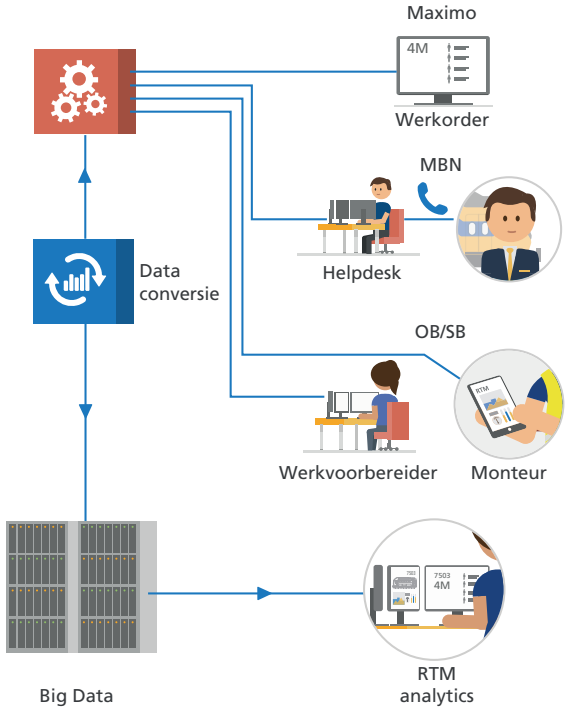

```
# import relevant classes from sparkml
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
...
# prepare and run linear regression
assembler = VectorAssembler(inputCols=x_cols, outputCol='features')
input = assembler.transform(train_data).select(['label', 'features'])
output = LinearRegression().setSolver('l-bfgs').fit(input)
```

- Data
  compressor logs
- Detection
  classification
- Diagnosis
  clustering
- Prognosis
  regression
- Operation
  humans

Air leakage detection

Oh no, humans in the loop!

- **>> Prognosis is difficult to explain and does not perform well enough**
- **>> Diagnosis does perform pretty good (~85% true positives; ~15% solved on time)**
- **>> Dedicated staff for bridging between algorithm and organisation**
- **>> We are observing recent air leakages that would have been prevented, so we're getting there**



Data
compressor logs

Detection
classification

Diagnosis
clustering

Prognosis
regression

Operation
humans

Air leakage detection

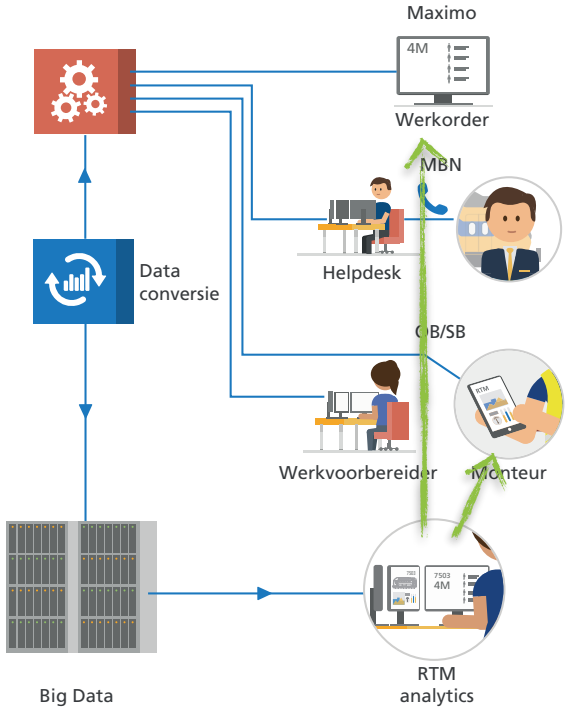
Oh no, humans in the loop!

>> **Prognosis is difficult to explain and does not perform well enough**

>> **Diagnosis does perform pretty good (~85% true positives; ~15% solved on time)**

>> **Dedicated staff for bridging between algorithm and organisation**

>> **We are observing recent air leakages that would have been prevented, so we're getting there**



Data
compressor logs

Detection
classification

Diagnosis
clustering

Prognosis
regression

Operation
humans

More use cases

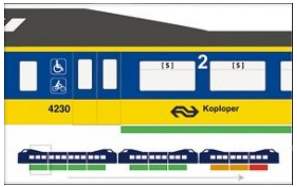
>> Hot wheels: failing axle bearings

>> Crowd control: passengers distribution

>> Slippery tracks: traction detection

>> Water usage: when you got to go you got to go

>> ...and many more to come



SPARK+AI SUMMIT 2018

Predictive Maintenance at the Dutch Railways

Air leakage detection in train breaking pipes

Ivo Everts | ivoeverts@godatadriven.com | @ivoeverts

#DSSAIS16