Analyzing Time Series Data with Spark

Sandy Ryza


1888
1889
1890
1891
1892

What’s time series data?

What’s time series data?

What makes it unique?

What’s time series data?

What makes it unique?

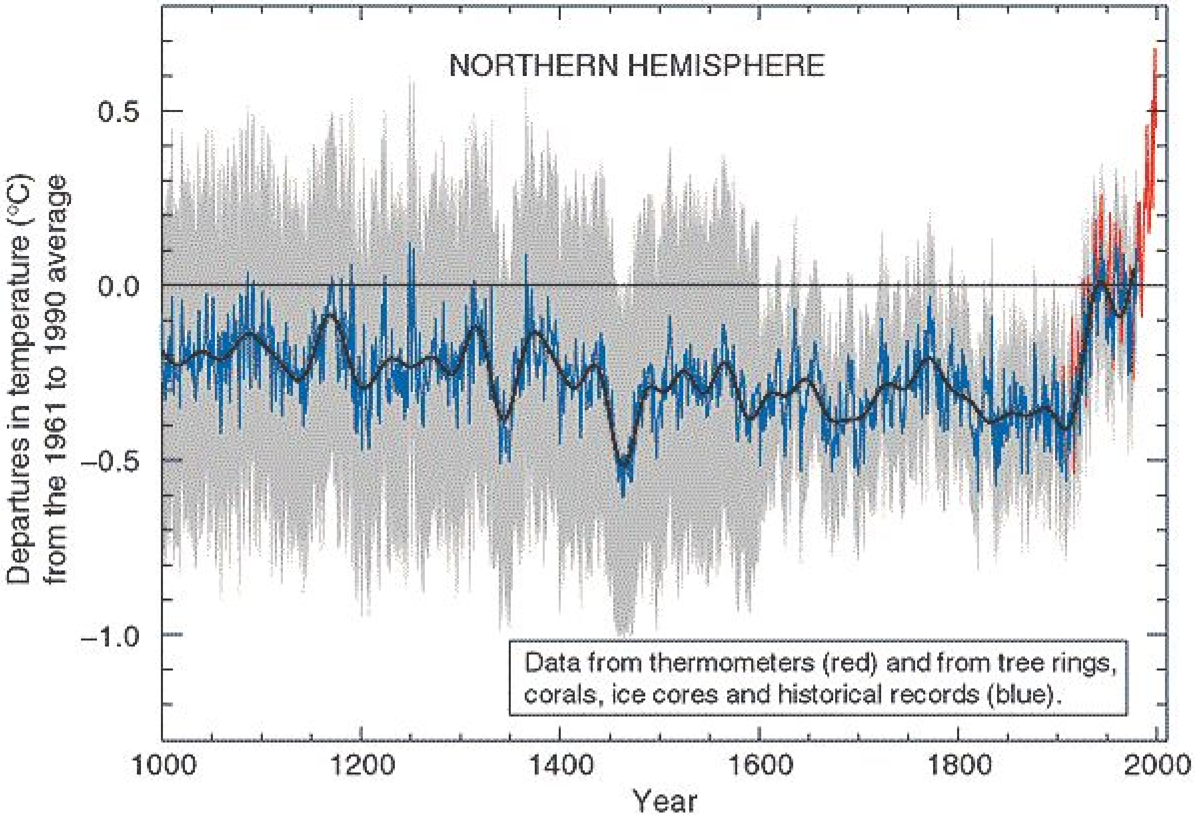
What do we do with it?

What’s time series data?

What makes it unique?

What do we do with it?

How do we deal with lots of it?


NORTHERN HEMISPHERE
Departures in temperature (°C) from the 1961 to 1990 average
0.5
0.0
−0.5
−1.0
1000
1200
1400
1600
1800
2000
Year
Data from thermometers (red) and from tree rings, corals, ice cores and historical records (blue).

Univariate

Time	**Observation**
4/10/1990 23:54:12	4.5
4/10/1990 23:54:13	5.5
4/10/1990 23:54:14	6.6
4/10/1990 23:54:15	7.8
4/10/1990 23:54:16	3.3

Multivariate

Time	Something	Something Else
4/10/1990 23:54:12	4.5	100.4
4/10/1990 23:54:13	5.5	101.3
4/10/1990 23:54:14	6.6	450.2
4/10/1990 23:54:15	7.8	600
4/10/1990 23:54:16	3.3	2000

What’s time series data?

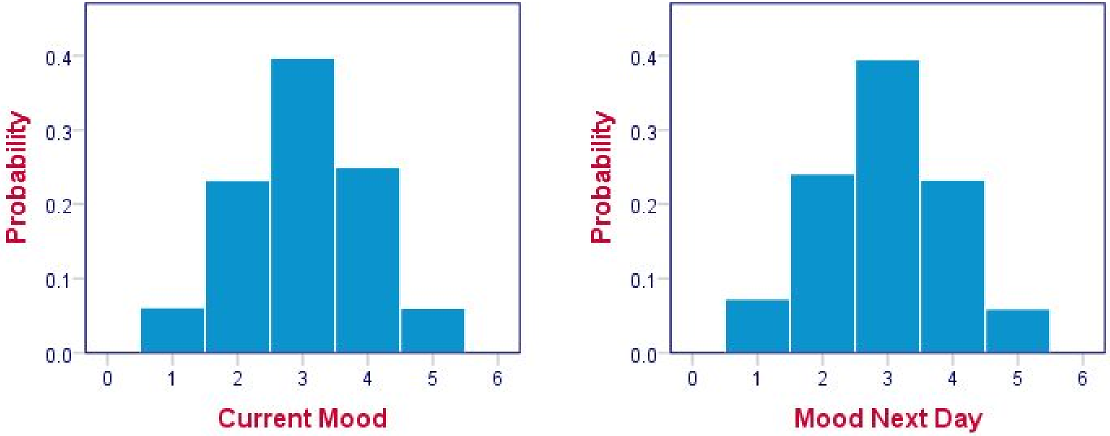
What makes it unique?

What do we do with it?

How do we deal with lots of it?

Basic Assumption of Most of Statistics

- Data is i.i.d (independent and identically distributed)



Time Series Dependencies

- What happens now depends on the past, but not the future
- What happens now depends on the recent past more than the distant past
- What happens now depends on the season
    - More traffic on weekdays
- What happens now depends on an absolute moment in time
    - Hurricane Sandy

Human time


HEY GIRL,
Time IS A SOCIAL CONSTRUCT, BUT EVERYONE LIKES TO CUDDLE.
memegenerator.net

Human time

```
vec = datestr(busdays('1/2/01','1/9/01','weekly'))
```

```
vec =

05-Jan-2001

12-Jan-2001
```

What’s time series data?

What makes it unique?

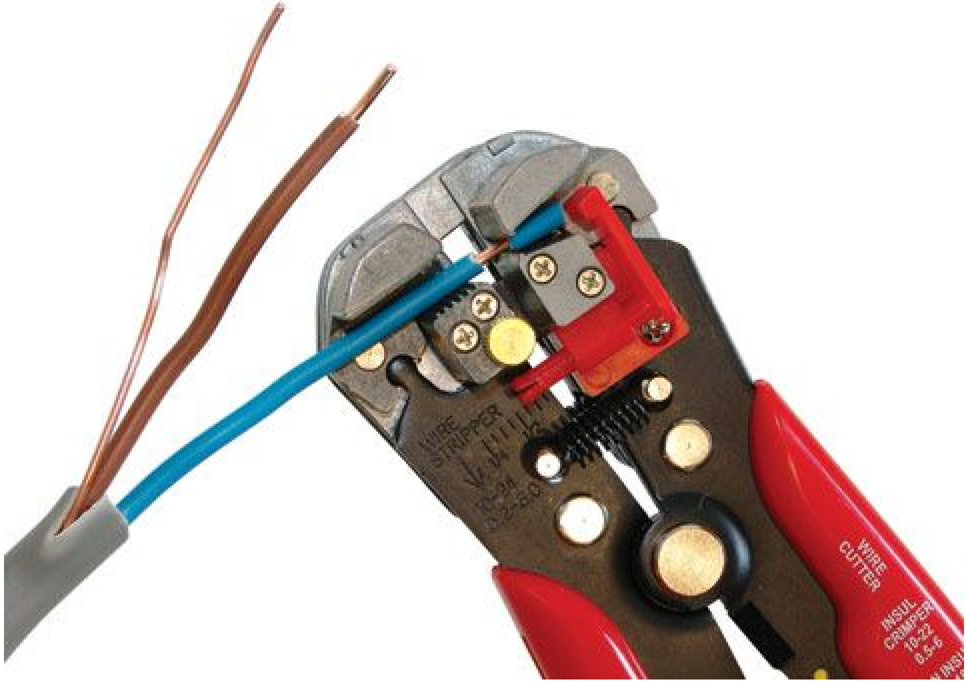
What do we do with it?

How do we deal with lots of it?

Forecasting

Das Orakel zu Delphi.

Stripping Out Time Dependencies



Detecting Anomalies

What’s time series data?

What makes it unique?

What do we do with it?

How do we deal with lots of it?

Existing Packages

- Matlab
  - Econometrics toolbox
- Python
  - Pandas
- R
  - zoo, xts
- SAS
  - ETS

Window Functions

```
SELECT buyerid, saletime, qtysold,
LAG(qtysold,1) OVER (order by buyerid, saletime) AS prev_qtysold
FROM sales WHERE buyerid = 3 ORDER BY buyerid, saletime;
```

```
buyerid  |       saletime        | qtysold | prev_qtysold
---------+---------------------+---------+--------------
       3 | 2008-01-16 01:06:09 |       1 |
       3 | 2008-01-28 02:10:01 |       1 |            1
       3 | 2008-03-12 10:39:53 |       1 |            1
       3 | 2008-03-13 02:56:07 |       1 |            1
       3 | 2008-03-29 08:21:39 |       2 |            1
       3 | 2008-04-27 02:39:01 |       1 |            2
```

Window Functions

```
windowSpec = \
  Window
    .partitionBy(df['category']) \
    .orderBy(df['revenue'].desc()) \
    .rangeBetween(-sys.maxsize, sys.maxsize)

dataFrame = sqlContext.table("productRevenue")

revenue_difference = \
  (func.max(dataFrame['revenue']).over(windowSpec) - dataFrame['revenue'])

dataFrame.select(
  dataFrame['product'],
  dataFrame['category'],
  dataFrame['revenue'],
  revenue_difference.alias("revenue_difference"))
```

Window Functions

- Serial dependencies:
  - LAG
- Absolute position:
  - RANK

The “Time Series for Spark” Project

https://github.com/cloudera/spark-timeseries

Goal: Provide a natural API for manipulating large scale time series data.

Goal: Provide statistical routines for modeling large scale time series data.

Scala

Java

python

How do we lay it out?
* Within a machine
* Across machines

“Observations”

Timestamp	Key	Value
2015-04-10	A	2.0
2015-04-11	A	3.0
2015-04-10	B	4.5
2015-04-11	B	1.5
2015-04-10	C	6.0

“Instants”

Timestamp	A	B	C
2015-04-10	2.0	4.5	6.0
2015-04-11	3.0	1.5	NaN

“Time Series”

DateTimeIndex: [2015-04-10, 2015-04-11]	
Key	**Series**
A	[2.0, 3.0]
B	[4.5, 1.5]
C	[6.0, NaN]

Time series in real world applications

- Time series are not that big!

- But there might be lots of them
  - Payments for every mortgage in America
  - Metrics from every robot in a factory
  - Investment portfolio with hundreds of thousands of derivatives

TimeSeriesRDD

Stores a collection of univariate time series with a conformed time index

```
rdd: RDD[String, Vector[Double]]

index: DateTimeIndex
```

TimeSeriesRDD

	5:00 PM	6:00 PM	7:00 PM	8:00 PM	9:00 PM	10:00 PM
GOOG	$523		$524	$600	$574	$400
AAPL	$384	$384	$385	$385	$378	$345
YHOO	$40	$60			$70	$80
MSFT	$134	$138	$175	$178	$123	$184
ORCL	$23	$30	$35	$45	$38	

TimeSeriesRDD distributed partitioning

	5:00 PM	6:00 PM	7:00 PM	8:00 PM	9:00 PM	10:00 PM
GOOG	$523		$524	$600	$574	$400
AAPL	$384	$384	$385	$385	$378	$345
YHOO	$40	$60			$70	$80
MSFT	$134	$138	$175	$178	$123	$184
ORCL	$23	$30	$35	$45	$38	

```scala
val tsRdd: TimeSeriesRDD = ...

// Find a sub-slice between two dates
val subslice = tsRdd.slice(
    ZonedDateTime.parse("2015-4-10", ISO_DATE),
    ZonedDateTime.parse("2015-4-14", ISO_DATE))

// Fill in missing values based on linear interpolation
val filled = subslice.fill("linear")

// Use an AR(1) model to remove serial correlations
val residuals = filled.mapSeries(series =>
    ar(series, 1).removeTimeDependentEffects(series))
```

DateTimeIndex

```
val dtIndex = DateTimeIndex.uniform(
  ZonedDateTime.parse("2015-4-10", ISO_DATE),
  ZonedDateTime.parse("2015-5-14", ISO_DATE),
  2 businessDays) // wowza that's some syntactic sugar

dtIndex.dateTimeAtLoc(5)
```

Time Series Models

ARIMA:

```scala
val modelsRdd = tsRdd.map { ts =>
  ARIMA.autoFit(ts)
}
```

GARCH:

```scala
val modelsRdd = tsRdd.map { ts =>
  GARCH.fitModel(ts)
}
```

Stats

```
val mostAutocorrelated = tsRdd.map { ts =>
  (TimeSeriesStatisticalTests.dwtest(ts), ts)
}.max
```

Roadmap: Core

- ~~RDDs~~ -> Datasets
- Full windowing
  - tumbling, sliding, sessions
- Full resampling
  - up and down
- Looong series
  - Across partitions

Roadmap: Stats

- Vector autoregression
- Regression ARIMA
- ARIMAX
- Full Holt-Winters
- Anomaly detection
    - Standard models