7CCSMSUF: Software Engineering and Underlying Technologies for Financial Systems

Kevin Lano

kevin.lano@kcl.ac.uk

Part 5: Trading and Analytics Technologies

kevin.lano@kcl.ac.uk

- Trading technologies: FIX protocol and FIXML
- Data analytics technologies: NoSQL, HBase, Map/Reduce, TensorFlow

Trading technologies: XML and FIXML

- FIX protocol and messages (CF969 slides: Introduction to the FIX Protocol)
- XML (Extensible Markup Language)
- FIXML (Financial Information Exchange ML)

Financial Information eXchange (FIX) protocol

- Open and defacto standard for financial information exchange
- Defines message format and communication model
- Platform independent
- Originated in mid 90's.

Uses of FIX

- Electronic trading, involving exchanges + other market participants
- Fixed income trading
- Streaming multicast of financial data via FAST (FIX Adapted for STreaming)

FIX text format

- Basic text format of a message is sequence of `field=value` bindings, separated by delimiter character (ASCII code 1, the SOH character).
- Fields are integers. Numbers from 0 to 4999 have predefined meanings, numbers from 5000 to 9999 available for application-specific extensions.
- Header segment identifies the FIX version (field 8), message body size (field 9), message type (field 35), etc.
- Body segment contains data of specific transaction, such as account identifier (field 1), price (field 44), etc.
- Finally a message trailer contains a checksum (field 10).

FIX messages

- Message sequence number field records the order of a message within a session (field 34). Enables detection of out-of-order messages, duplicate messages, etc.
- Messages can be administration messages, eg., session establishment and termination messages, resend requests, etc., or messages enacting financial process steps, such as order creation, position report, etc.

XML

- Structured text format used as basis for HTML and many other description and interchange representations
- Nested tree structure of nodes with attributes and subnodes.
- Sometimes written by hand (eg., small web pages or configuration files). More often generated by software.
- Used for SOAP (web service message format) + many other structured data representations.

XML

XML nodes are named by a *tag* identifying what the node represents:

```
<tag1 .... />
```

A node with no subnodes.

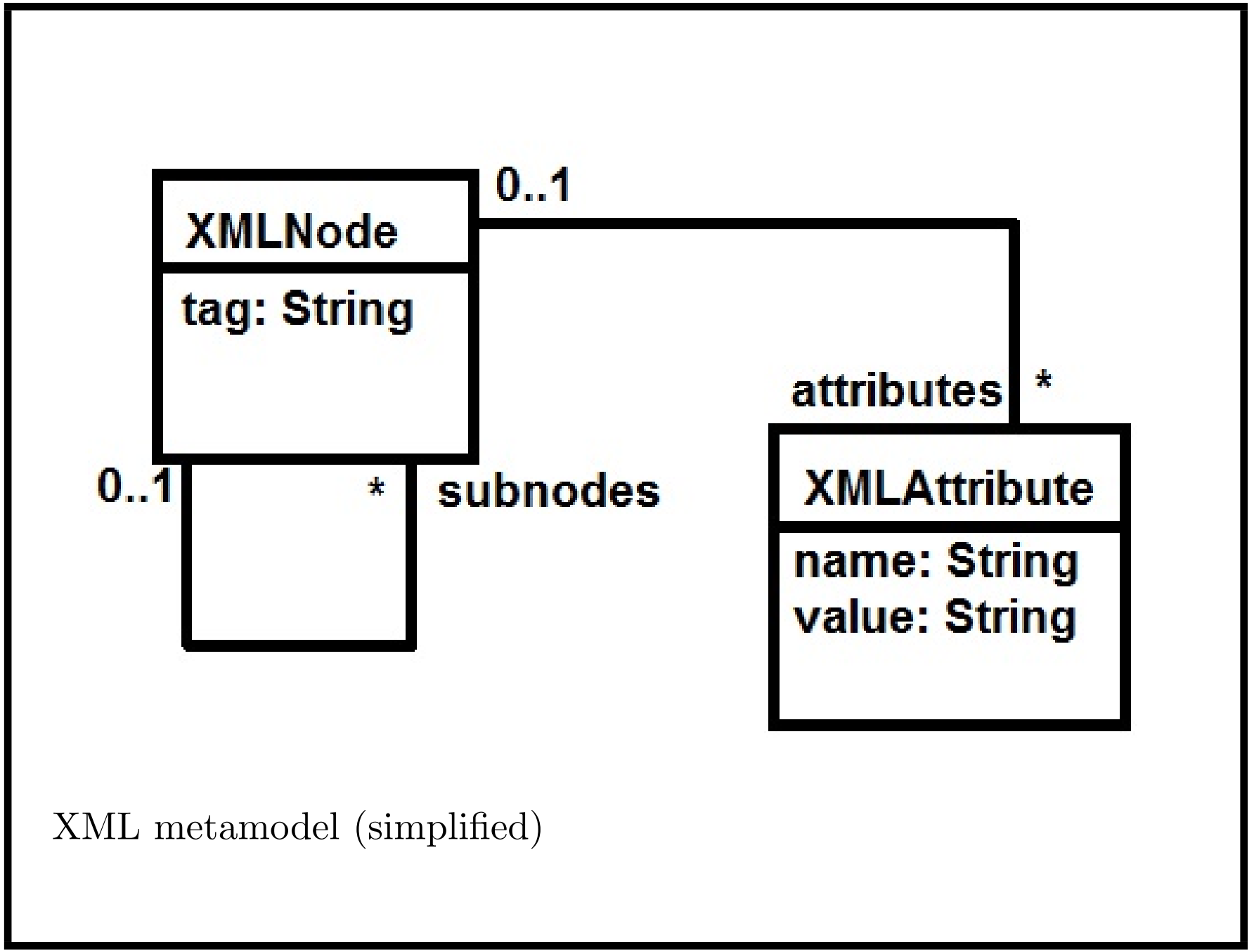
Nodes can have attributes, with values:

```
<tag1 att1="val1" att2="val2" />
```

Nodes may be nested to any depth:

```
<tag1 .... >
  <tag2 ... />
  <tag3 ... />
</tag1>
```

Simple nodes with tags *tag*2 and *tag*3 are nested in composite node with *tag*1.



XML metamodel (simplified)

XML

As an example, the XML data

```
<car make="XJ6" colour="silver" manufacturer="Jaguar">
  <engine capacity="3l" />
  <DVLARecord status="OffRoad" date="20120101"/>
</car>
```

represents a particular car.

Fix Protocol and FIXML

- Financial transactions can be electronically expressed using different formats, for interchange between market participants
- FIX (Financial Information eXchange) protocol provides standardised, non-proprietary message format and communication protocol to exchange financial data
- Messages can be in basic character format or in XML, called FIXML
- FIXML format is defined at http://fixwiki.org/fixwiki/, and http://www.fixtrading.org.
- Produced/consumed by software, eg., trading applications.

FIXML Examples

Simple Order message:

```
<?xml version="1.0" encoding="ASCII"?>
<FIXML>
  <Order ClOrdID="123456" Side="2"
                TransactTm="2001-10-17T09:30:47-05:00" OrdTyp="2"
          Px="93.25" Acct="26522154">
    <Hdr Snt="2001-10-17T09:30:47-05:00"
                PosDup="N" PosRsnd="N" SeqNum="521">
      <Sndr ID="AFUNDMGR"/>
      <Tgt ID="ABROKER"/>
   </Hdr>
   <Instrmt Sym="IBM" ID="459200101" IDSrc="1"/>
   <OrdQty Qty="1000"/>
 </Order>
</FIXML>
```

FIXML Examples

First line indicates version of XML being used. FIXML tag identifes that data is FIXML format.

Defines an *Order* with nested header *Hdr*, instrument *Instrmt* and *OrdQty* subnodes.

The header has nested sender and target subnodes.

FIXML Examples

More complex example of a Position Report message:

```
<?xml version="1.0" encoding="ASCII"?>
<FIXML>
<PosRpt RptID="541386431" Rslt="0"
  BizDt="2003-09-10T00:00:00" Acct="1" AcctTyp="1"
  SetPx="0.00" SetPxTyp="1" PriSetPx="0.00" ReqTyp="0" Ccy="USD">
<Hdr Snt="2001-12-17T09:30:47-05:00" PosDup="N" PosRsnd="N" SeqNum="1002">
<Sndr ID="String" Sub="String" Loc="String"/>
<Tgt ID="String" Sub="String" Loc="String"/>
<OnBhlfOf ID="String" Sub="String" Loc="String"/>
<DlvrTo ID="String" Sub="String" Loc="String"/>
</Hdr>
<Pty ID="OCC" R="21"/>
<Pty ID="99999" R="4"/>
<Pty ID="C" R="38">
<Sub ID="ZZZ" Typ="2"/>
```

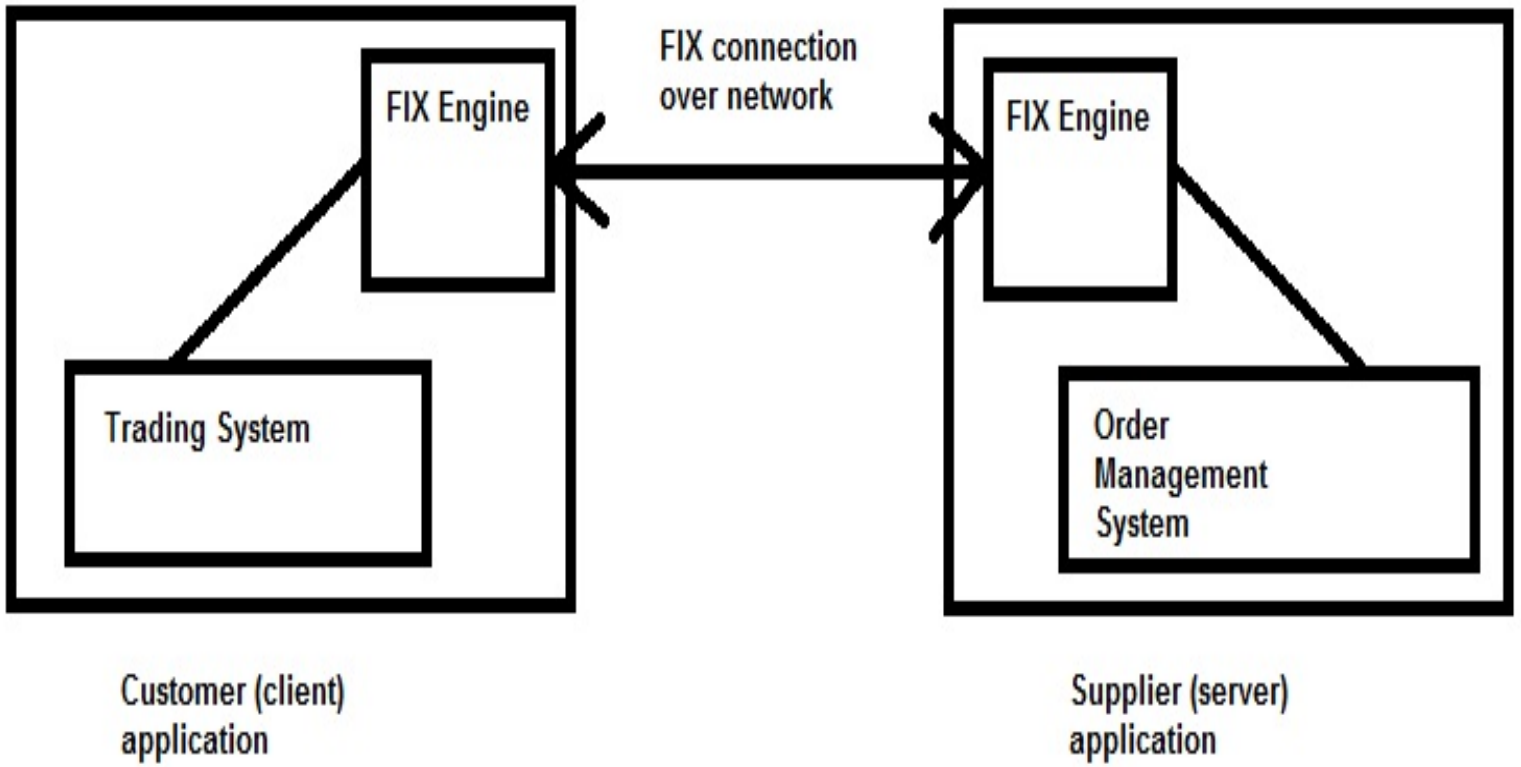
```
</Pty>
<Qty Typ="SOD" Long="35" Short="0"/>
<Qty Typ="FIN" Long="20" Short="10"/>
<Qty Typ="IAS" Long="10"/>
<Amt Typ="FMTM" Amt="0.00"/>
<Instrmt Sym="AOL" ID="KW" IDSrc="J" CFI="OCASPS" MMY="20031122"
 Mat="2003-11-22T00:00:00" Strk="47.50" StrkCcy="USD" Mult="100"/>
</PosRpt>
</FIXML>
```

Multiple subnodes with same tag (Pty and Qty).

Other sample FIXML messages can be found at
http://fixwiki.org/fixwiki/

FIX engines

- A FIX engine is software component which creates and processes FIX messages, either in basic text format, in FIXML or in both.
- Can be called from an application via an API in order to carry out trading commands via FIX protocol.
- There are FIX engines for many different platforms and programming environments.
- Also possible (but time-consuming) to implement your own engine.



Role of FIX engines

FIX engines

- FIX engines act as endpoints of a FIX communication channel between financial applications.
- Manage administration of the communication, enforce the communication protocol including timeouts and error detection and recovery.
- A communication session is initiated by client via a *logon* message to server.
- Server receives the request, and after validation, establishes the connection.
- The client can end a session by sending a *logout* message.
- A special kind of message (heartbeat messages) are sent in both directions at regular intervals in order to maintain the connection. The session may be closed if no message of any kind is received for 2 times the heartbeat interval.

Data Analytics technologies

- NoSQL data storage, HBase (extra slides: Column-oriented databases and HBase)
- MapReduce, Hadoop Big data processing
- TensorFlow machine learning platform.

Data analytics technologies

- NoSQL data storage technologies – Bigtable and HBase
- Column-oriented data models for efficient storage of big data (up to petabytes)
- Bigtable developed at Google for web indexing, Google Earth, Google Finance, etc.

Traditional database properties

- Support CRUD operations (Create, Read, Update, Delete)
- Have ACID (Atomicity, Consistency, Isolation, Durability) properties of database transactions
- Relational databases use *normalised* tables to reduce data redundancy
- SQL query language used to update + search data.

NoSQL databases

- Support CRUD operations
- NoSQL databases relax some ACID properties
- Data may not be normalised – may improve efficiency but results in data duplication. Relies on cheap data storage.
- Instead of SQL, use specialised APIs to read + update data.

NoSQL databases

- Key-value store: Oracle Coherence
- Column-oriented: Bigtable, HBase
- Document databases: MongoDB
- Text-search: Apache Lucene

Column-oriented data model

- Table rows have structure

  `row key, column family 1, ..., column family n`

  Where `row key` is a string, unique for each row, and each `column family` consists of one or more related columns of data cells.

- Rows stored in lexicographical order of keys

- Each cell can store multiple versions of data, each version is timestamped.

Bigtable example: Google Analytics

- Records website statistics (visits per day, etc)
- Row keys of session table are

  `website name + time of session start`

- Thus all session data for visits to one website are stored contiguously. Stored in chronological order.
- Approx 200 TB.

HBase

- Apache HBase (hbase.apache.org) is column-oriented, key-value database
- Each data item addressed by *row key*, *column family*, *column name* as for Bigtable, multiple timestamped versions of items
- Rows stored in lexicographic order of row key
- Choose key so that (i) data likely to be accessed together are close; (ii) data accesses are otherwise distributed across wide range of table.
- Physically, large tables split into disjoint blocks of rows, stored on separate computers.

HBase example: Share data

- Share data (alphavantage.co) with format

  `Symb, date, opening price, high, low, closing price, volume`

  where *Symb* is alphanumeric company symbol, eg, "IBM".
  opening and *closing* prices are the prices at the start and end of the trading day *date*, *volume* is trading volume of shares of *Symb* on *date*.

  date has format day/month/year (2 digits for the day, followed by 2 for month then 4 for year).

- How to ensure that all data for one symbol is stored together, in chronological order?

Recall that "a0" $<$ "a1" in lexicographic order, etc.

HBase example: Share data

Keys must be:

`Symb:year:month:day`

So that row order is:

`IBM:2017:12:28`

`IBM:2017:12:29`

`IBM:2018:01:02`

Successive trading days always stored in successive rows.

HBase example: Share data

- Likely to access nearby dates of one symbol in one process
- Same dates for different shares are distributed across table
- Column family *prices* with 4 columns *opening*, *closing*, *high*, *low*
- Column family *volume* with one column.

HBase Java API

Create a table, add a column family (table, row key and column names are byte arrays):

```
HTableDescriptor table = new HTableDescriptor(
    TableName.valueOf("ShareTable".getBytes()));
table.addFamily(new HColumnDescriptor("prices".getBytes()));
```

Insert a row with key k, column family cf and columns $c1$, $c2$:

```
Put p = new Put(k);
p.addColumn(cf,c1,data1);
p.addColumn(cf,c2,data2);
table.put(p); // or a list of Put's
```

get method obtains single rows/cells. *scan* obtains sets of rows.

```
Get g = new Get(k);
g.addColumn(cf,c1);
```

```
Result r = table.get(g);
if (r.isEmpty())
{ System.err.println(
    "No data for row " + k + " cell " + cf + ":" + c1); }
else
{ System.out.println("" + r.value()); }
```

Scan rn rows starting from k:

```
Scan scan = new Scan();
scan.setStartRow(k);
scan.setCaching(rn);
ResultScanner results = table.getScanner(scan);
for (Result r : results)
{ // do something with r
}
```

Can also filter results in the scan.

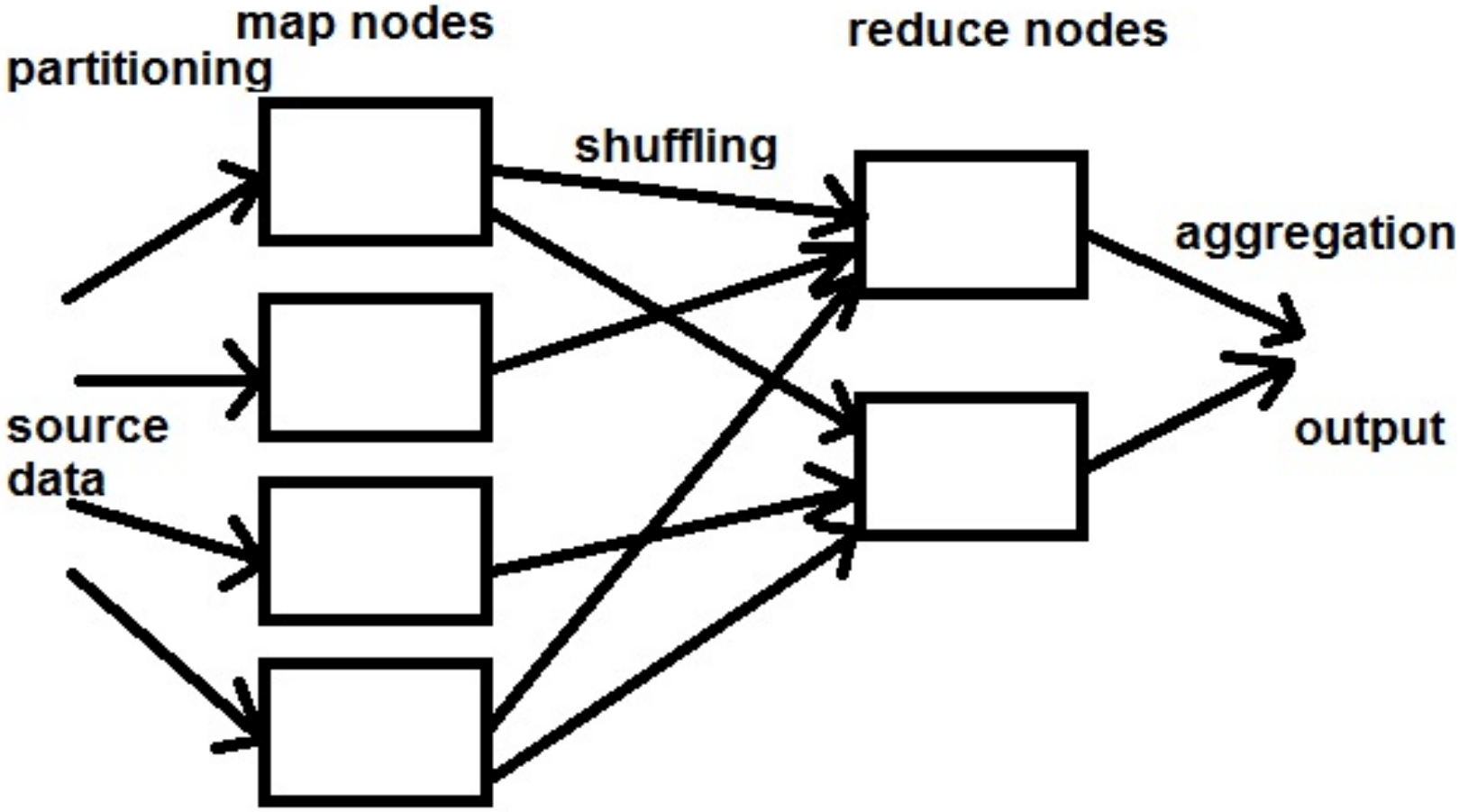
Map/Reduce

- Framework for distributed processing of large datasets
- Partitions dataset into blocks, separately processed by *map* function, possibly on separate computers
- Result sets combined via *reduce* function to produce overall result
- Suitable for analysis of large static datasets (not streams)
- Incorporates reliability via distribution + reassignment of work from failed processors.

Elements of Map/Reduce

A Map/Reduce platform provides:

- Data partitioning and assignment of input for *map* processors
- Partitioning and shuffling of *map* output to *reduce* processors
- Crash recovery.



Map/Reduce processing

Map/Reduce semantics

For particular analysis task, developer writes *map* and *reduce* functions.

- $map : KeyT1 \times ValueT1 \rightarrow Sequence(KeyT2 \times ValueT2)$
  Takes pairs $(k1, v1)$ and produces lists of $(k2, v2)$ pairs.
- $reduce : KeyT2 \times Sequence(ValueT2) \rightarrow Sequence(ValueT3)$
  Takes a $(k2, [w1, ..., wm])$ pair and produces lists $[u1, ..., up]$.

The shuffle step combines *map* results: (k, v) and (k, u) combined to $(k, [v, u])$, etc.

reduce can be performed in stages, if it is *associative*.

Map/Reduce example

Count occurrences of different company symbols in FIXML Order message files m.

- $map(m.name, m.data)$ produces $[(s, 1)]$ if $m.data$ is an Order and $Sym = s$ occurs in m.data, otherwise []
- $reduce(s, counts)$ produces $[(s, \Sigma counts)]$.

Aggregated final result is list of symbols with counts.

reduce is associative: different count sums for same s could be fed into a further *reduce*.

Evaluating general queries using Map/Reduce

$$data \rightarrow select(x \mid P) \rightarrow collect(e) \rightarrow r()$$

Where r is associative.

- Split *data* into partitions based on some $key1$ of elements
- *map* produces e-sequences from x satisfying P
- Allocate to *reduce* based on $key2$ of e-values
- *reduce* applies r.

What operations can r be? *sum*, *prd*, *max*, *min*, others?

Apache Hadoop (hadoop.apache.org)

Open-source framework for distributed storage + processing of Big datasets.

- Hadoop distributed file system (HDFS)
- Hadoop MapReduce implements Map/Reduce using HDFS.

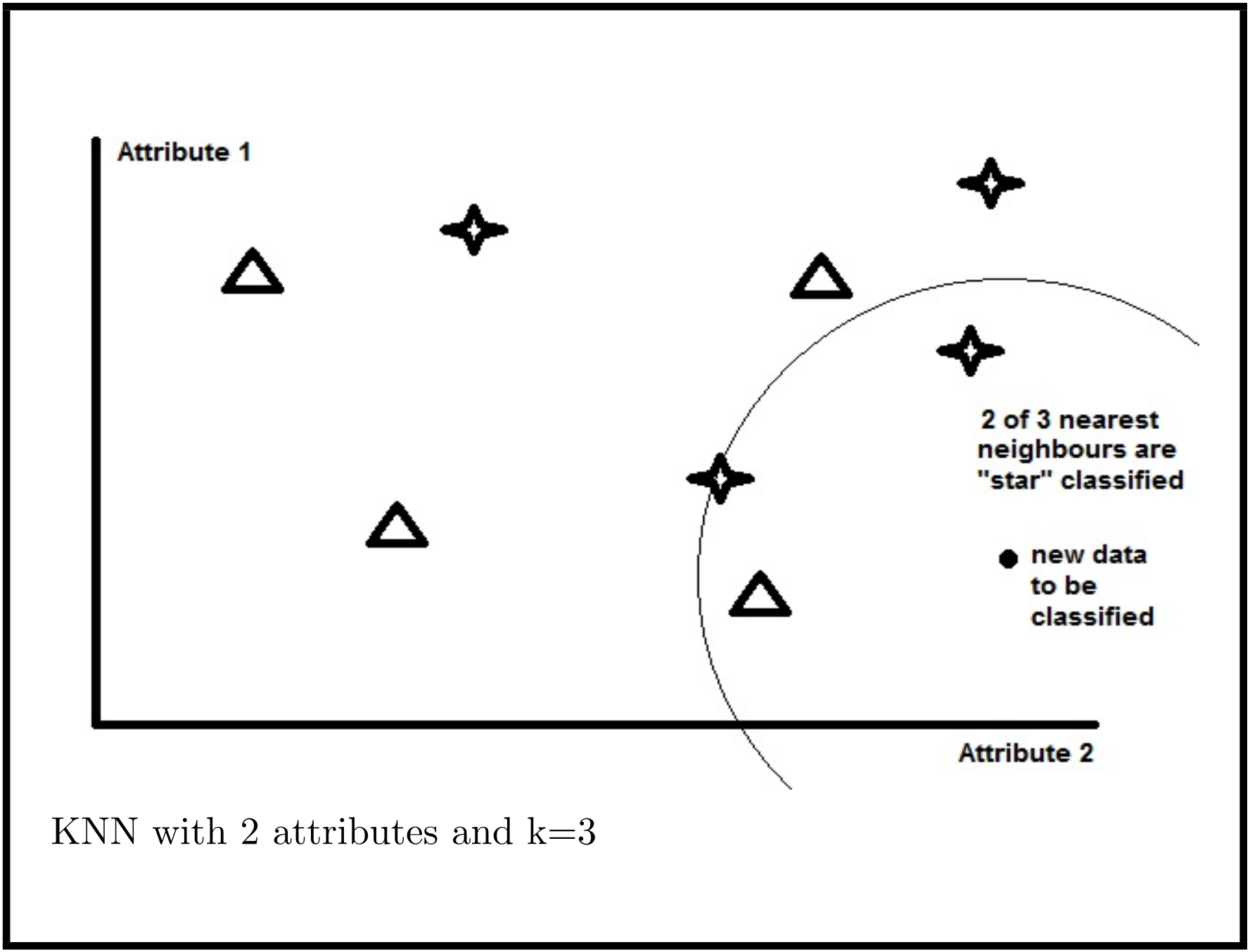
Also supports HBase and other packages.

TensorFlow

- Open-source platform for machine learning
- Supports construction of neural networks for learning data classification rules
- TensorFlow processes data as multi-dimensional arrays/matrices, termed *tensors*
- Training phase builds the classifier using existing data with known classifications
- Could be used for learning trading/investment strategies, etc.

Machine learning algorithms

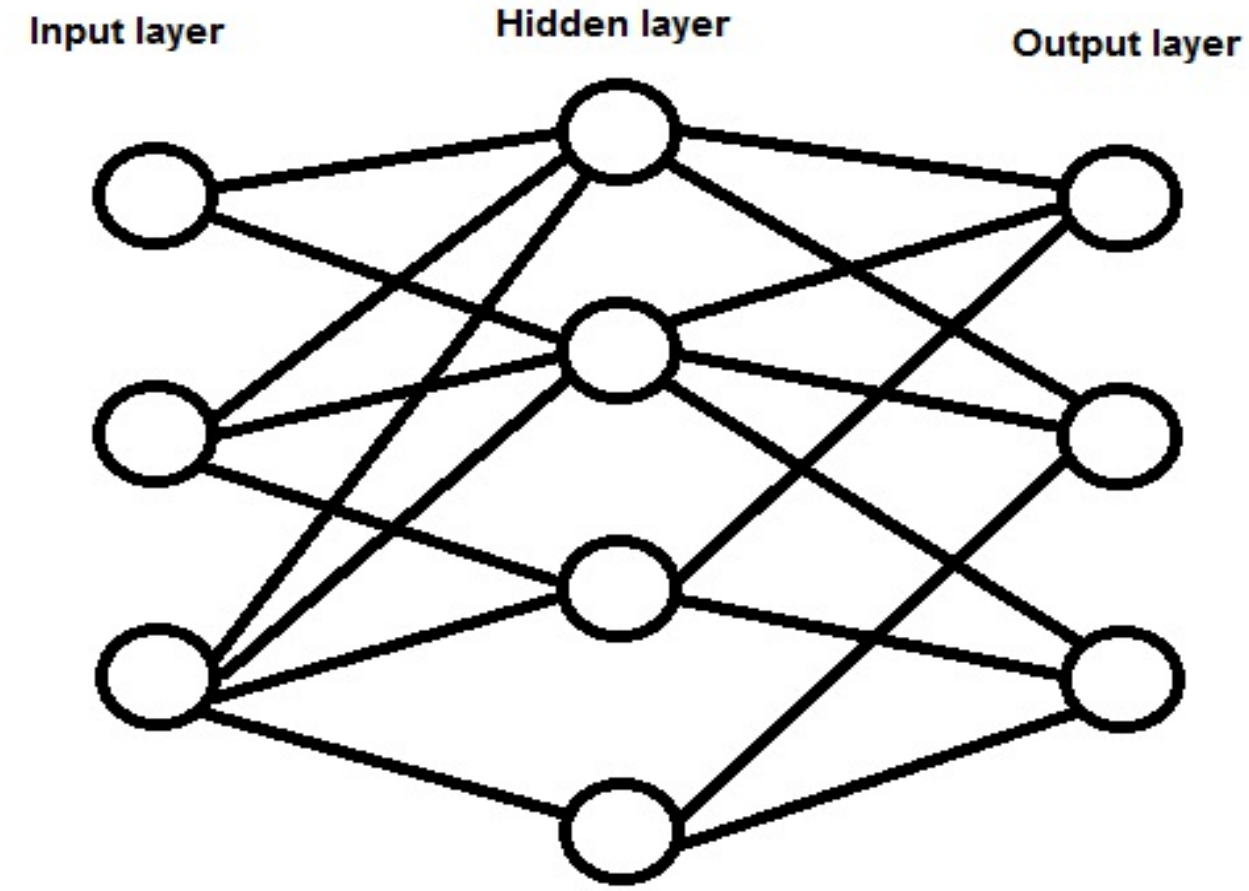
- *K-Nearest Neighbours (KNN)*: data points have N dimensions (attributes) + a label (classification).
- Learning consists of retaining entire training data, represented as vectors.
- New items are classified according to majority label of their k-nearest neighbours.
- Building the model is cheap, but cost of classification of new data depends on N and k, and on distance metric (usually Euclidean).
- Small k is more efficient but generally less accurate.



KNN with 2 attributes and k=3

Machine learning algorithms

- *Multilayer Perceptron (MLP)*: neural networks with input layer, one or more hidden layers, one output layer.
- Hidden layer nodes compute a weighted function of their inputs. Output nodes represent classifications.
- Learning consists of applying training data to network, weights of functions adjust based on known classifications of the data.
- New items are classified according to output node activated when they are input to the network.



MLP with one hidden layer

Machine learning algorithms

- *Recurrent Neural Networks (RNN)* additionally use previous state of nodes as inputs. Can address sequence-prediction problems which MLP cannot.
- *Long Short-Term Memory (LSTM)* can retain state over indefinite periods. Widely applied to AI tasks including speech recognition + automated translation. Appropriate for classifying + making predictions about time-series data.

The Tensorflow-based keras Python library provides MLP and LSTM (https://keras.io).

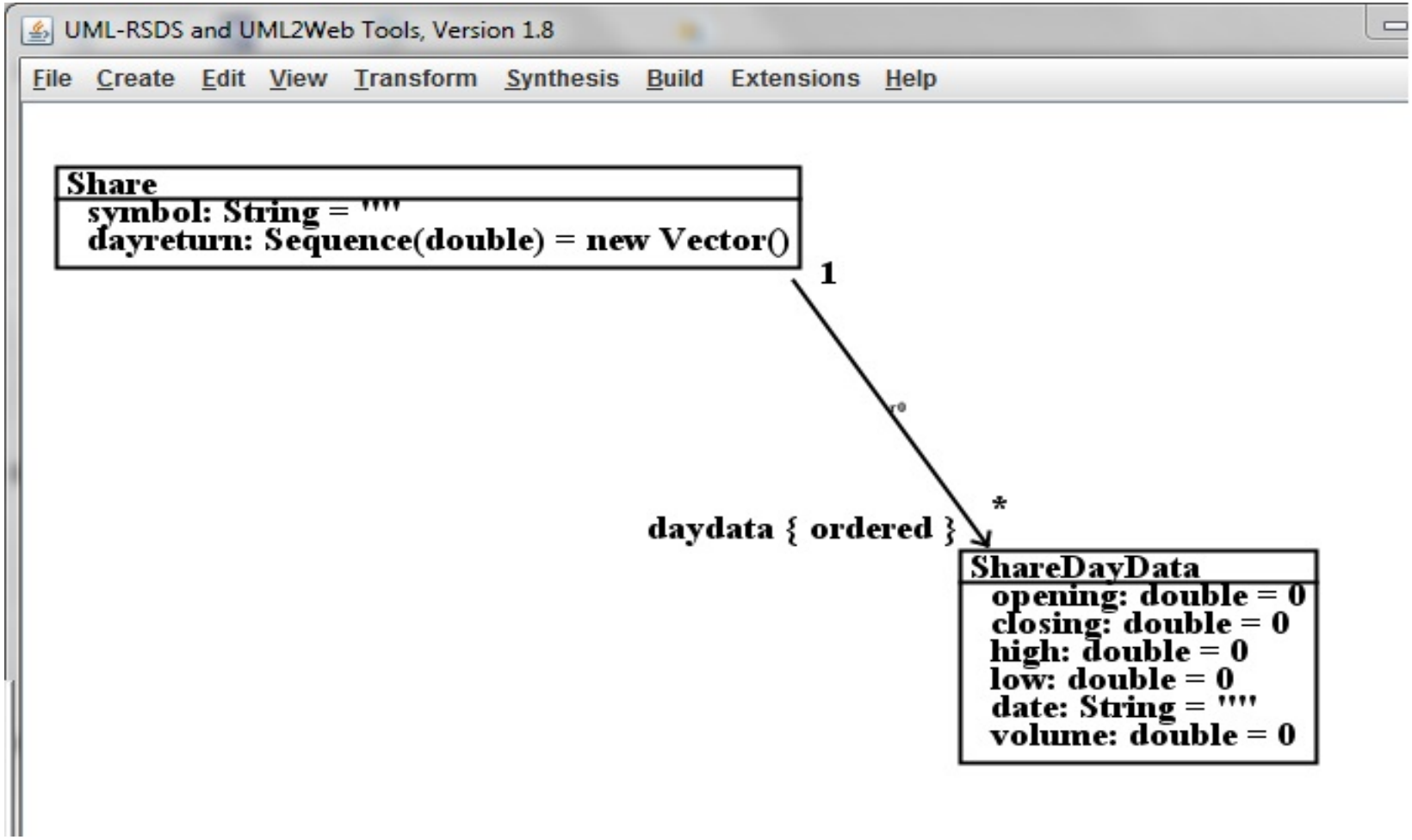
Scikit-learn provides KNN (https://scikit-learn.org).

Case study: Technical analysis of share prices

- Technical analysis involves analysis of share price data over past periods, in order to guide investment decisions.
- Eg., if there is a consistent downward trend in a share price, it is useful to recognise point where trend ceases: this could be opportunity to invest in share.
- Software support in technical analysis involves computation of different technical indicators such as moving averages of share prices.

Technical analysis of share prices

- Technical indicators include *moving averages*
- *SMA (simple moving average)* is average price of share over preceding period (eg., previous 26 trading days).
- As time passes, oldest price in series is replaced by the newest, and average is recalculated.


UML-RSDS and UML2Web Tools, Version 1.8
File Create Edit View Transform Synthesis Build Extensions Help
Share
symbol: String = ""
dayreturn: Sequence(double) = new Vector()
1
*
daydata { ordered }
ShareDayData
opening: double = 0
closing: double = 0
high: double = 0
low: double = 0
date: String = ""
volume: double = 0

Share model

Technical analysis of share prices

26-day average of closing prices is:

$$SMA(26)(n) = (\Sigma_{i=n-25}^{n} daydata[i].closing)/26$$

Recall that sequences are indexed starting from 1.

In OCL, the definition is:

$$SMA(26)(n) = (daydata.subrange(n-25, n) \rightarrow collect(closing) \rightarrow sum())/26$$

for any n from 26 up to $daydata.size$.

Technical analysis of share prices

Simple average treats older information equally to recent information.

An alternative is *exponential moving average* (EMA), which weights more recent information more strongly than older data:

$$EMA(26)(n) = \Sigma_{i=n-25}^{n} daydata[i].closing * \alpha * (1-\alpha)^{(n-i)}$$

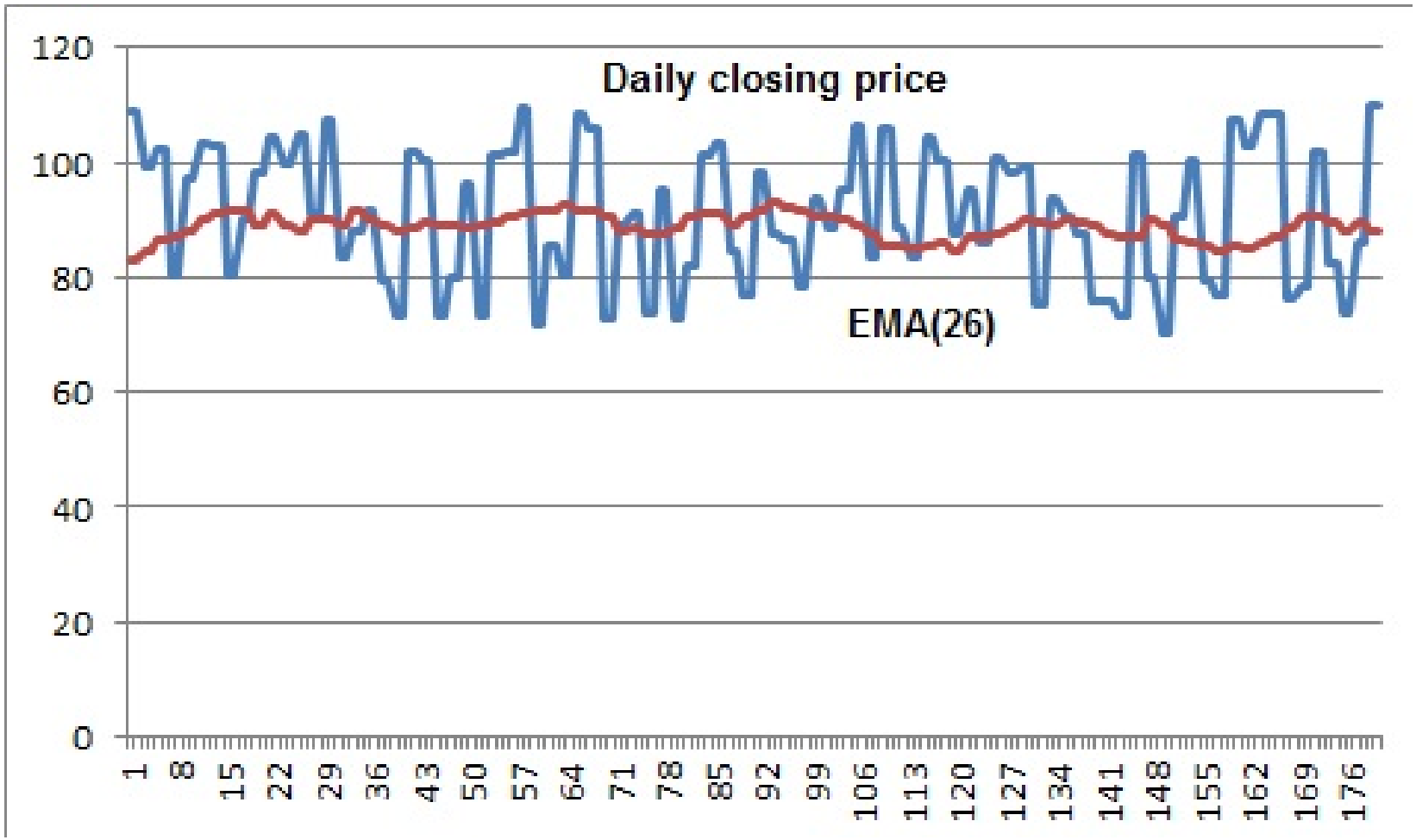
That is:

$$EMA(26)(n) = Integer.Sum(n-25, n, i, \\ daydata[i].closing * \alpha * ((1-\alpha) \rightarrow pow(n-i)))$$

where $n \geq 26$ and $n \leq daydata.size$, for 26-day EMA with decay factor α.

Larger decay factor weights more recent data more heavily.

Here $\alpha = \frac{2}{N+1}$ where N is term of EMA. For $N = 26$, $\alpha = 0.074$ approximately.



Daily price and EMA graph for a share

Technical analysis of share prices

- From EMA, further indicators can be derived.
- *moving average convergence/divergence* (MACD) is difference between a short-term and long-term EMA:

$$MACD(12, 26)(n) \;=\; EMA(12)(n) - EMA(26)(n)$$

- At points when $EMA(12)$ becomes greater than $EMA(26)$, ie., short-term graph crosses long-term from below, is evidence of increasing price trend (or 'bullish' sentiment in market towards the share).
- Corresponds to MACD crossing zero line from negative to positive values.
- Conversely if MACD crosses zero line from positive to negative, can indicate a decreasing price trend and 'bearish' sentiment in market.

Technical analysis of share prices

- Indicators for particular shares can be used for ML
- Eg., for KNN, MLP take 5 most-recent MACD crossovers to predict if price will increase in next day
- Using LSTM, could predict specific next day price based on preceding N days data.

Summary

- We have reviewed important underlying technologies for finance
- FIX protocol enables transfer of financial data and service requests in a standardised format
- Data analytics technologies enable large scale storage and analysis of financial data.