Spark Machine Learning

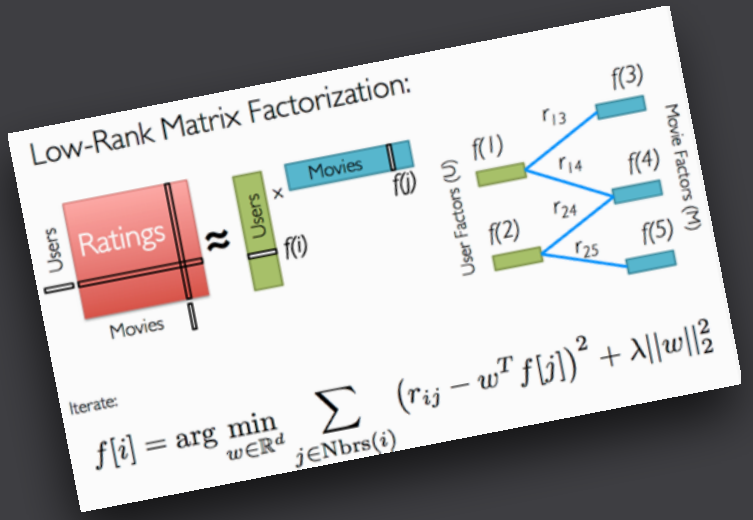
Future Cloud Summer School

Paco Nathan @pacoid

2015-08-08

http://cdn.liber118.com/workshop/fcss_ml.pdf

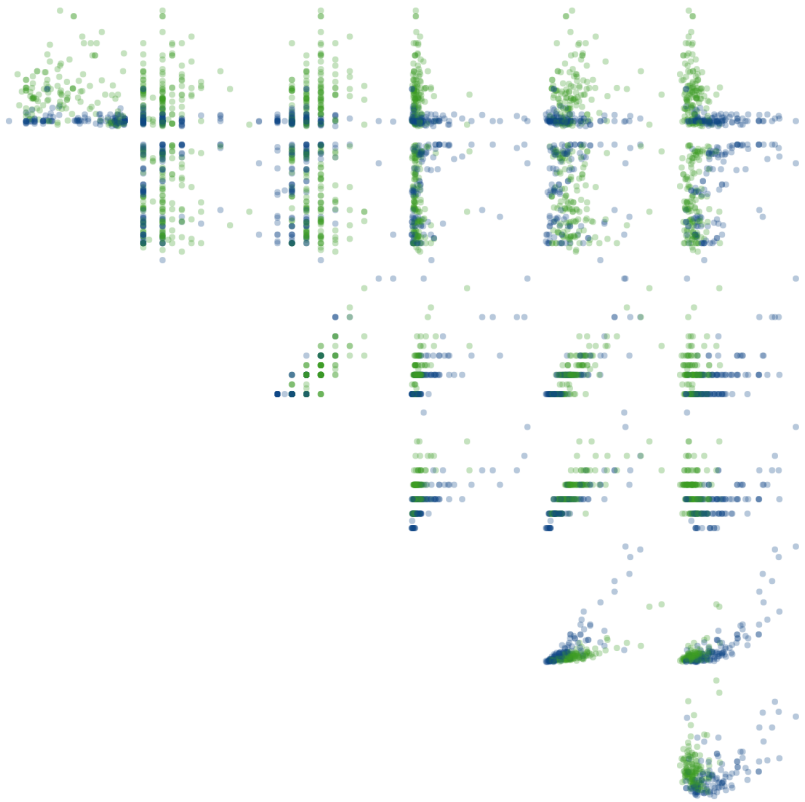
ML Background



A Visual Guide to Machine Learning

Stephanie Yee, Tony Chu

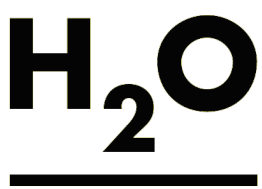
r2d3.us/visual-intro-to-machine-learning-part-1/

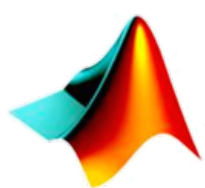


ML: *Background…*

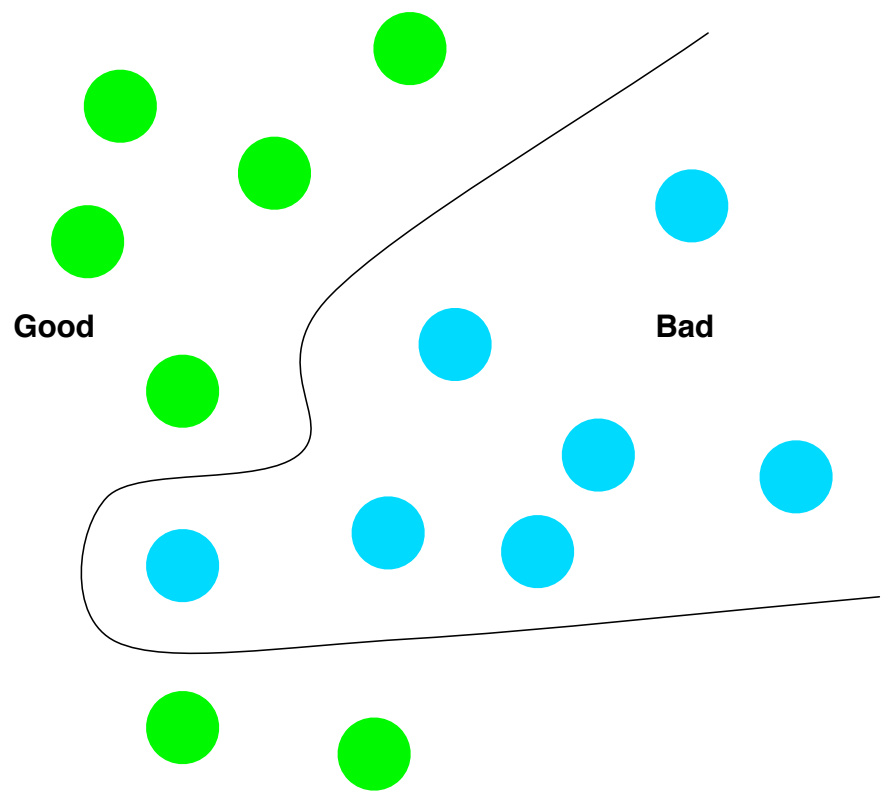
Most of the ML libraries that one encounters today focus on two general kinds of solutions:

- **convex optimization**
- **matrix factorization**





One might think of the convex optimization in this case as a kind of *curve fitting* – generally with some *regularization term* to avoid overfitting, which is not good



ML: *Background…*

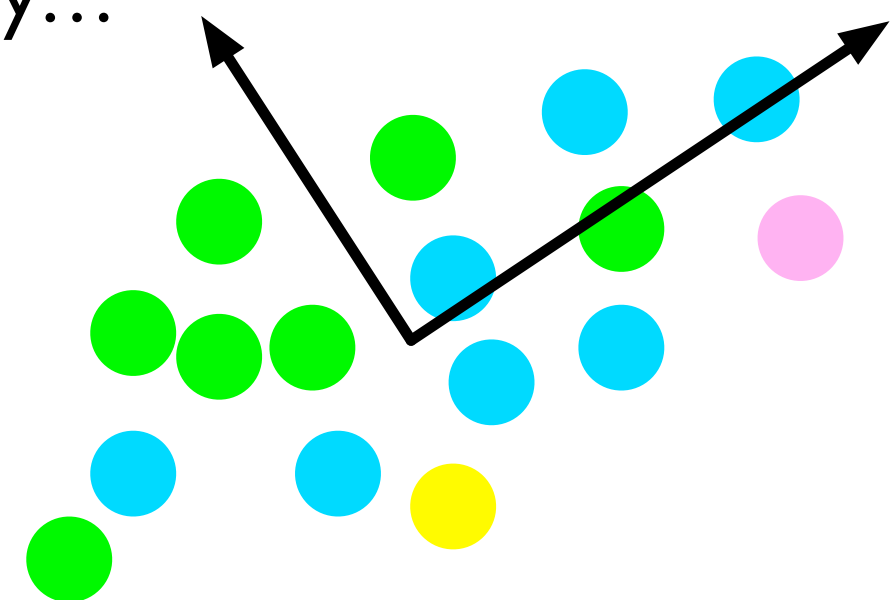
For *supervised learning*, used to create classifiers:

1. categorize the expected data into N classes
2. split a sample of the data into train/test sets
3. use learners to optimize classifiers based on the training set, to label the data into N classes
4. evaluate the classifiers against the test set, measuring error in predicted vs. expected labels

That's great for security problems with simply two classes: *good guys* vs. *bad guys* …

But how do you decide what the classes are for more complex problems in business?

That's where the **matrix factorization** parts come in handy…



For *unsupervised learning*, which is often used to reduce dimension:

1. create a covariance matrix of the data
2. solve for the eigenvectors and eigenvalues of the matrix
3. select the top N eigenvectors, based on diminishing returns for how they explain variance in the data
4. those eigenvectors define your N classes

ML: *Background…*

An excellent overview of ML definitions (up to this point) is given in:

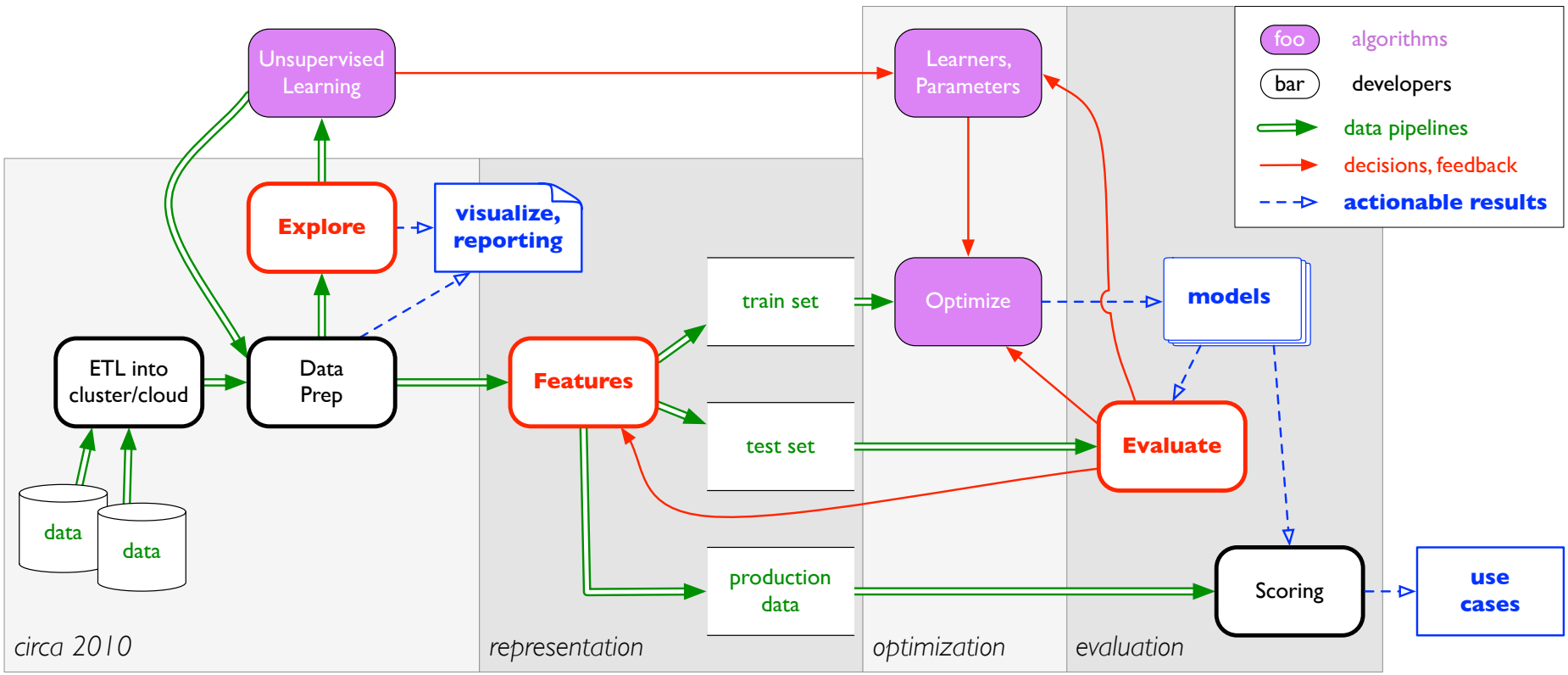
> *A Few Useful Things to Know about Machine Learning*
> **Pedro Domingos**
> CACM 55:10 (Oct 2012)
> http://dl.acm.org/citation.cfm?id=2347755

To wit:

Generalization = Representation + Optimization + Evaluation

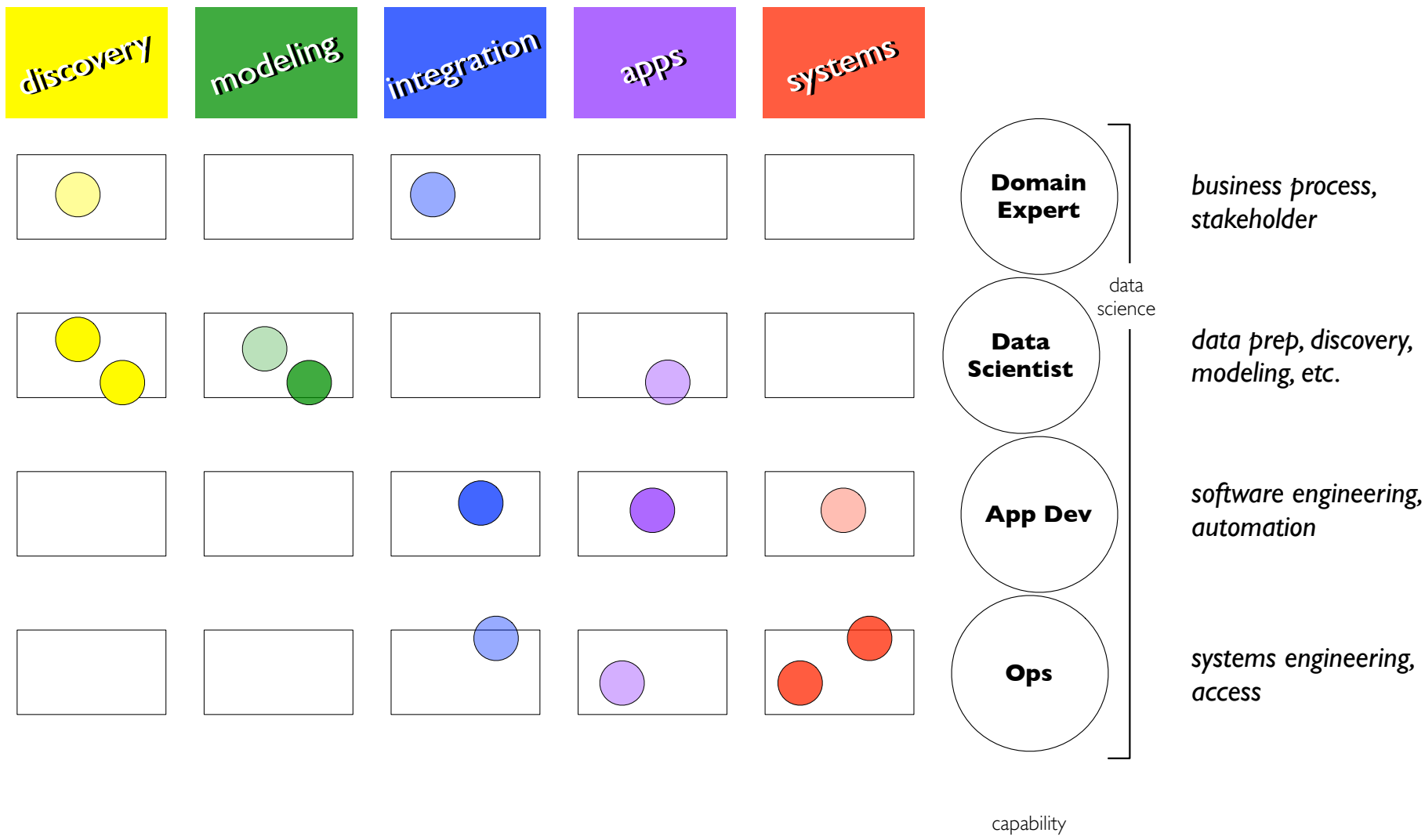
ML: *Workflows*

A generalized ML workflow looks like this…

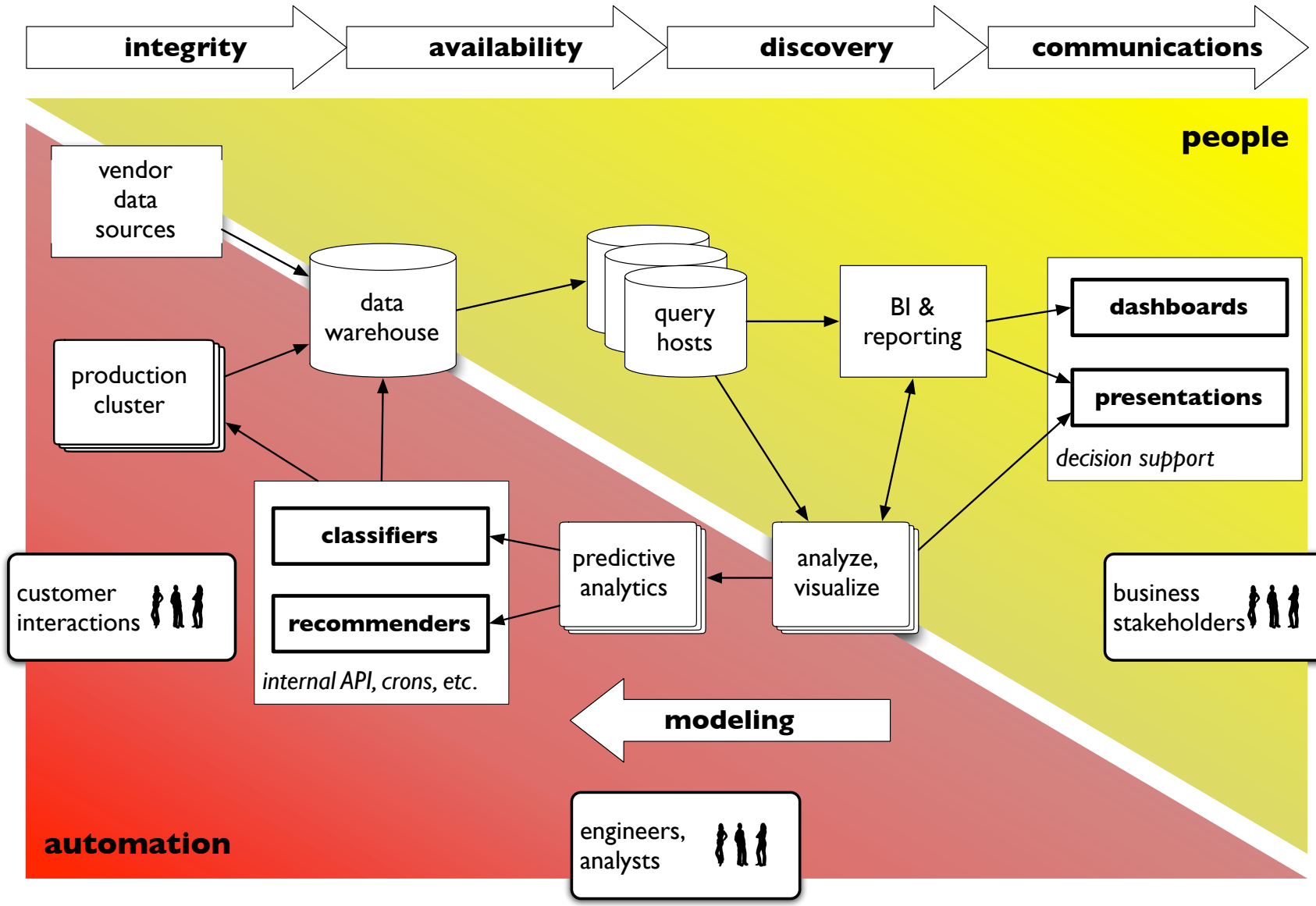


With results shown in **blue**, and the harder parts of this work highlighted in **red**

ML: *Team Composition = Needs x Roles*

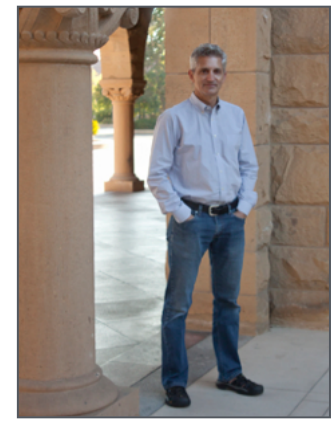


ML: *Organizational Hand-Offs*



Information Systems Laboratory @Stanford published ADMM, optimizing many different ML algorithms using a common formula:

$$\begin{aligned} &\textit{minimize}: && f(x) + g(z) \\ &\textit{subject to}: && Ax + Bz = c \end{aligned}$$



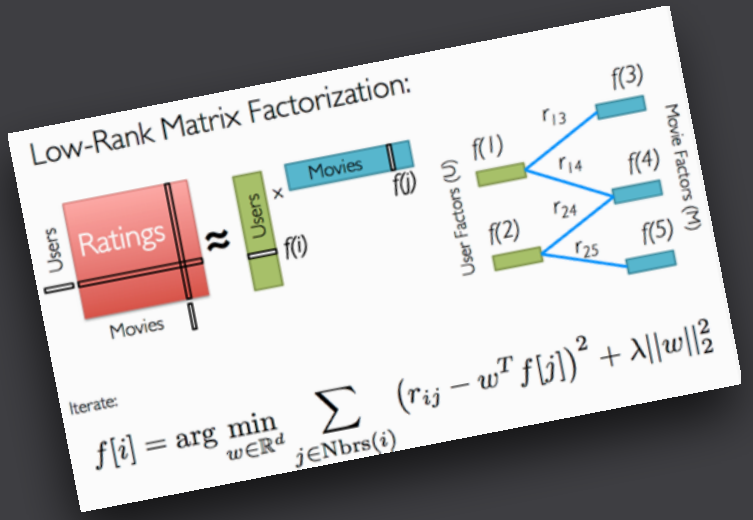
Stephen Boyd
stanford.edu

A *loss function* $f(x)$ and *regularization term* $g(z)$

Alternating Direction Method of Multipliers
S. Boyd, N. Parikh, et al., Stanford (2011)
stanford.edu/~boyd/papers/admm_distr_stats.html

Many such problems can be posed in the framework of convex optimization. Given the significant work on decomposition methods and decentralized algorithms in the optimization community, it is natural to look to parallel optimization algorithms as a mechanism for solving large-scale statistical tasks. This approach also has the benefit that one algorithm could be flexible enough to solve many problems.

MLlib, ML Pipelines, etc.



MLlib: *Recent talks…*

Building, Debugging, and Tuning Spark Machine Learning Pipelines
Joseph Bradley
spark-summit.org/2015/events/practical-machine-learning-pipelines-with-mllib-2/

Scalable Machine Learning (MOOC)
Ameet Talwalkar
edx.org/course/scalable-machine-learning-uc-berkeleyx-cs190-1x

Announcing KeystoneML
Evan Sparks
amplab.cs.berkeley.edu/announcing-keystoneml/

MLlib: *Background…*

Distributing Matrix Computations with Spark MLlib
Reza Zadeh, Databricks
lintool.github.io/SparkTutorial/slides/day3_mllib.pdf

MLlib: Spark's Machine Learning Library
Ameet Talwalkar, Databricks
databricks-training.s3.amazonaws.com/slides/Spark_Summit_MLlib_070214_v2.pdf

Common Patterns and Pitfalls for Implementing Algorithms in Spark
Hossein Falaki, Databricks
lintool.github.io/SparkTutorial/slides/day1_patterns.pdf

Advanced Exercises: MLlib
databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

MLlib: *Background…*

spark.apache.org/docs/latest/mllib-guide.html

Key Points:

- framework vs. library
- *scale*, *parallelism*, *sparsity*
- building blocks for long-term approach

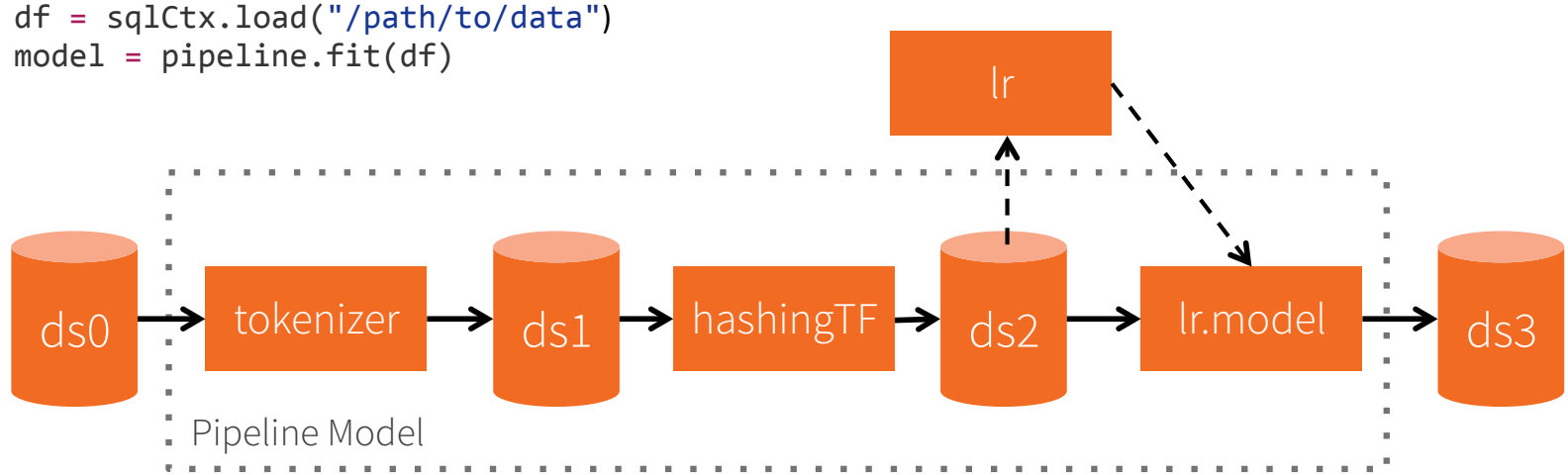
MLlib: *Background…*

Components:

- scalable statistics
- classifiers, regression
- collab filters
- clustering
- matrix factorization
- feature extraction, normalizer
- optimization

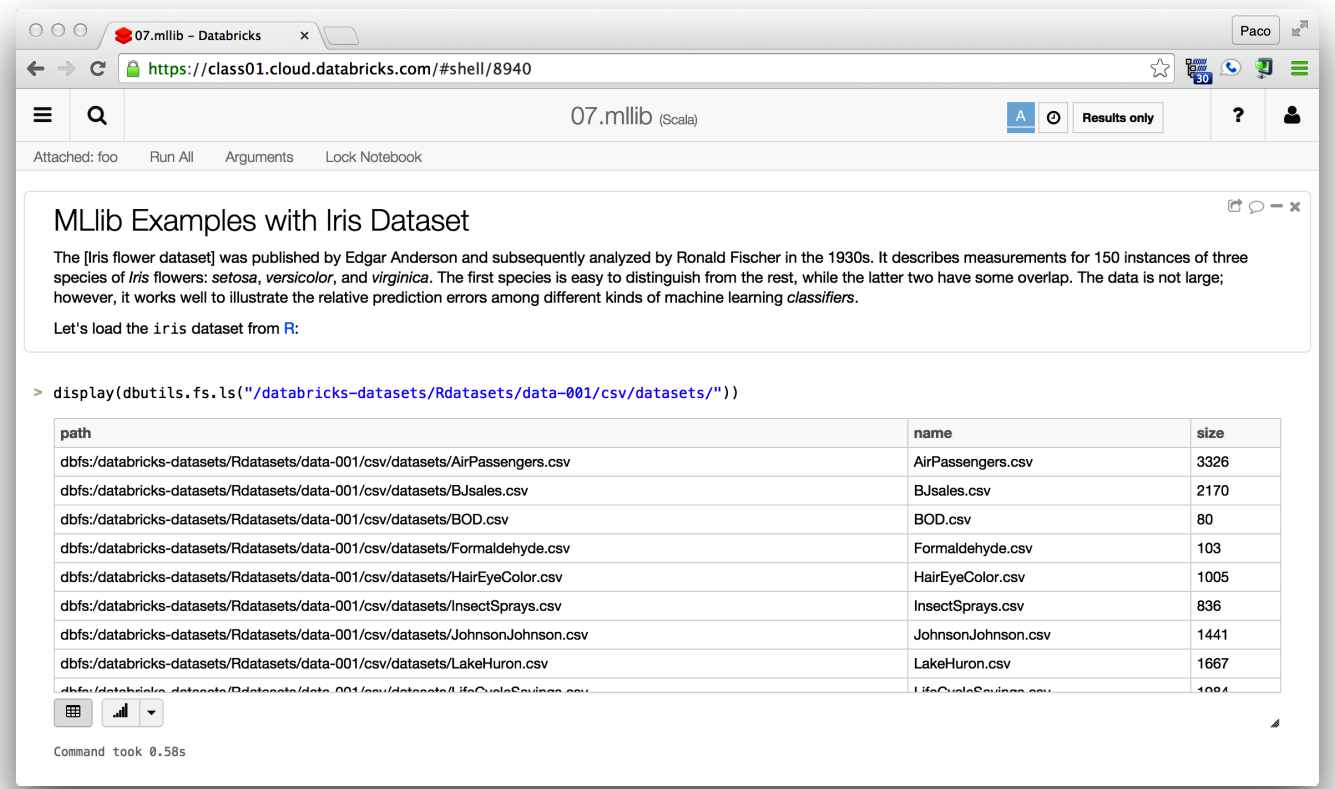
Machine Learning Pipelines

```
tokenizer = Tokenizer(inputCol="text", outputCol="words”)
hashingTF = HashingTF(inputCol="words", outputCol="features”)
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

df = sqlCtx.load("/path/to/data")
model = pipeline.fit(df)
```

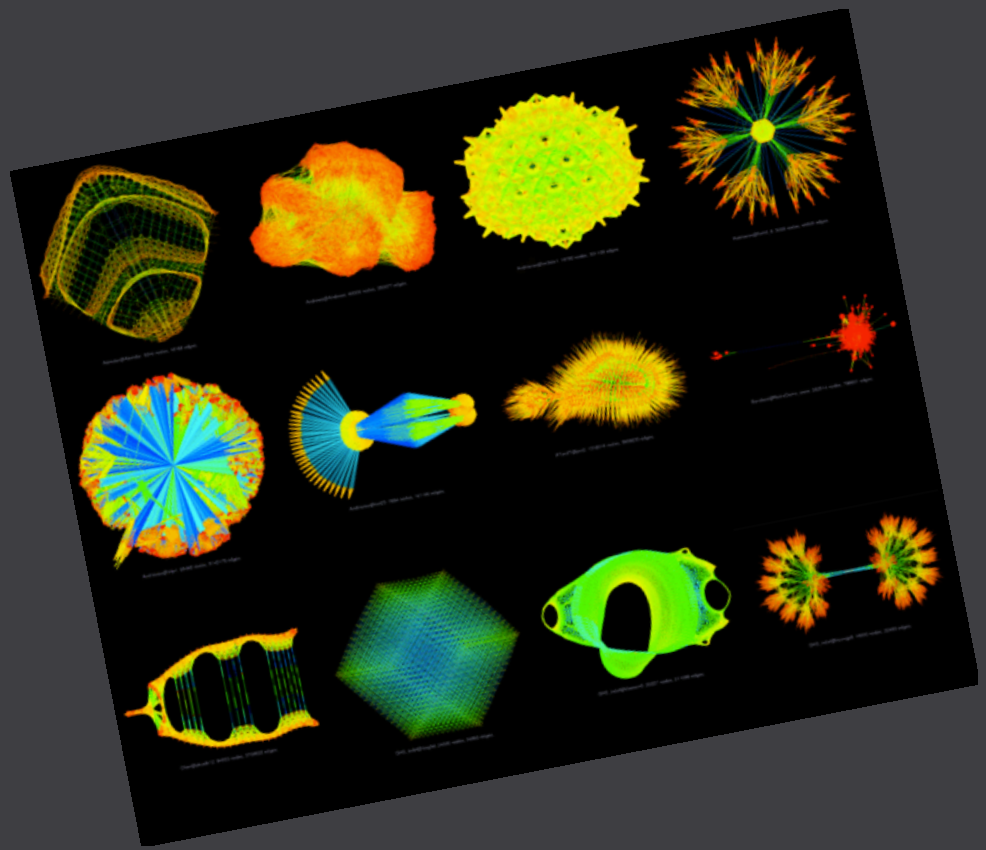


MLlib: *Code Exercise*

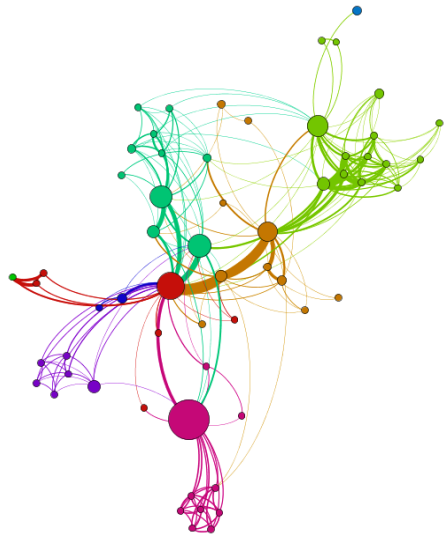
Clone and run `/_SparkCamp/demo_iris_mllib_2` in your folder:



Graph Analytics

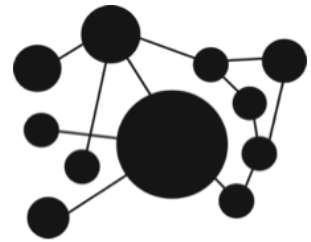


Graph Analytics: *terminology*

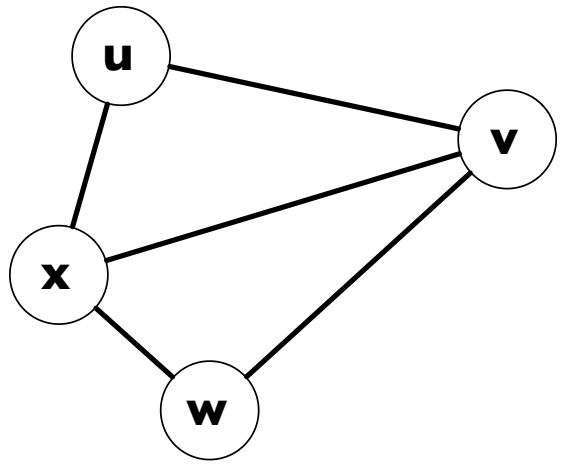


- many real-world problems are often represented as *graphs*
- graphs can generally be converted into *sparse matrices* (bridge to linear algebra)
- *eigenvectors* find the stable points in a system defined by matrices – which may be more efficient to compute
- beyond simpler graphs, complex data may require work with *tensors*

Graph Analytics: *example*



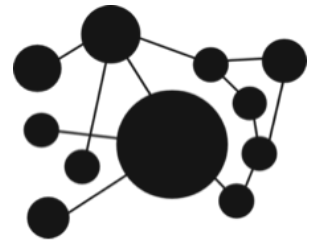
Suppose we have a graph as shown below:



We call **x** a *vertex* (sometimes called a *node*)

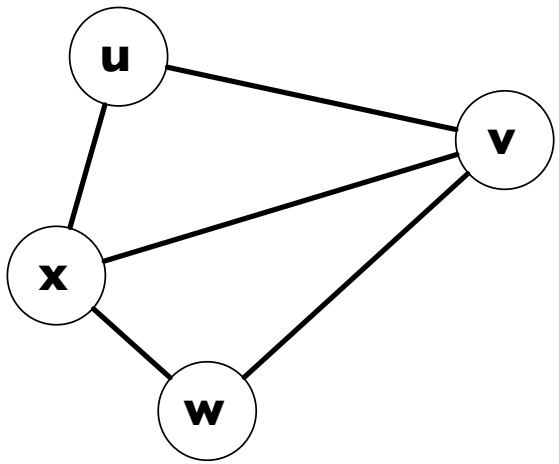
An *edge* (sometimes called an *arc*) is any line connecting two vertices

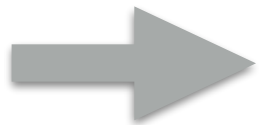
Graph Analytics: *representation*



We can represent this kind of graph as an *adjacency matrix*:

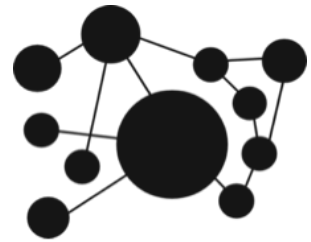
- label the rows and columns based on the vertices
- entries get a 1 if an edge connects the corresponding vertices, or 0 otherwise





	u	v	w	x
u	0	1	0	1
v	1	0	1	1
w	0	1	0	1
x	1	1	1	0

Graph Analytics: *algebraic graph theory*



An *adjacency matrix* always has certain properties:

- it is *symmetric*, i.e., $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$
- it has real eigenvalues

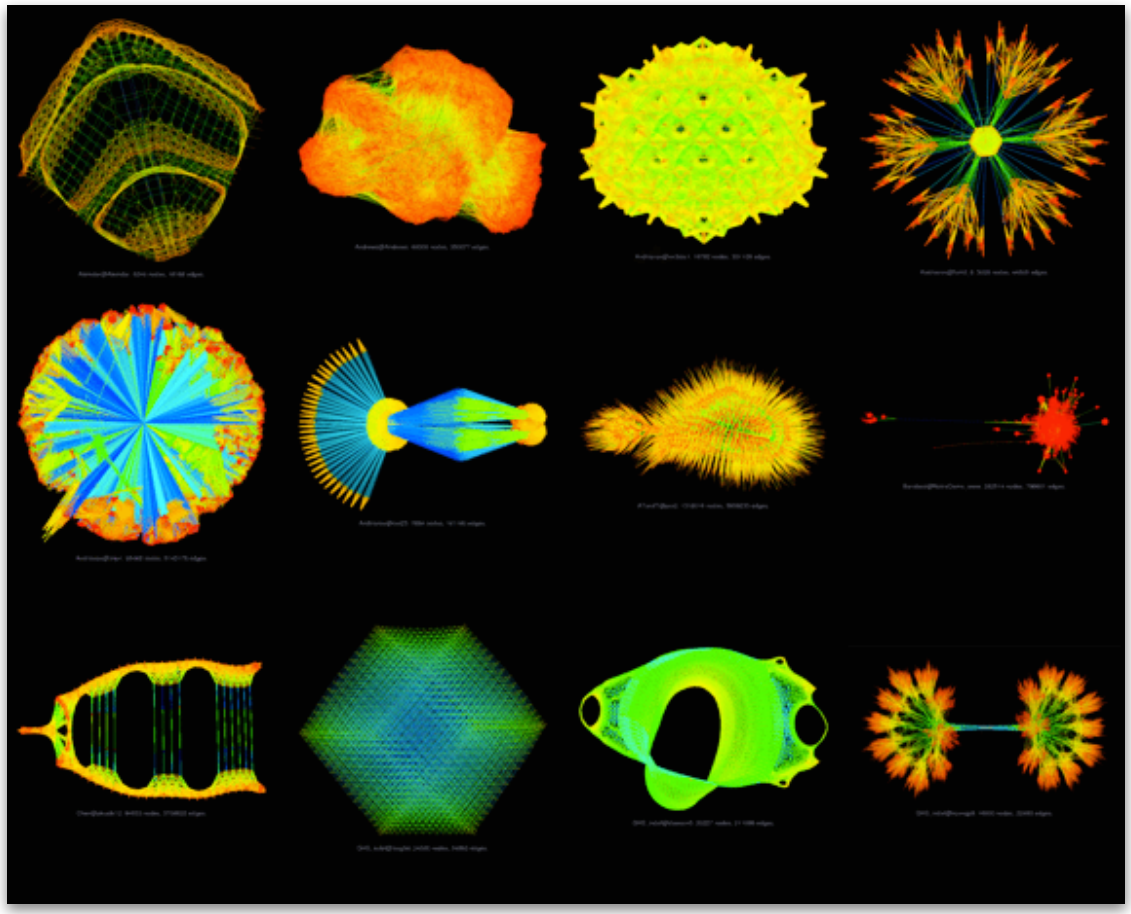
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Therefore *algebraic graph theory* bridges between *linear algebra* and *graph theory*

Graph Analytics: *beauty in sparsity*

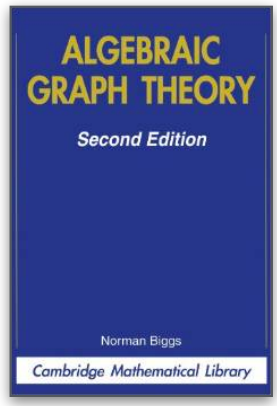
Sparse Matrix Collection… for when you **really** need a wide variety of sparse matrix examples, e.g., to evaluate new ML algorithms

University of Florida Sparse Matrix Collection
cise.ufl.edu/research/sparse/matrices/

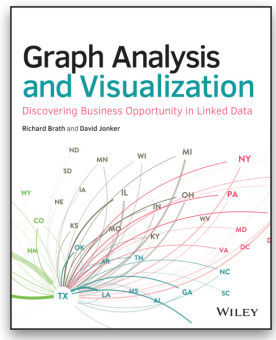


Graph Analytics: *resources*

Algebraic Graph Theory
Norman Biggs
Cambridge (1974)
amazon.com/dp/0521458978



Graph Analysis and Visualization
Richard Brath, David Jonker
Wiley (2015)
shop.oreilly.com/product/9781118845844.do



See also examples in: **Just Enough Math**

Graph Analytics: *tensor solutions emerging*

Although tensor factorization is considered problematic, it may provide more general case solutions, and some work leverages Spark:

The Tensor Renaissance in Data Science

Anima Anandkumar @UC Irvine

radar.oreilly.com/2015/05/the-tensor-renaissance-in-data-science.html

Spacey Random Walks and Higher Order Markov Chains

David Gleich @Purdue

slideshare.net/dgleich/spacey-random-walks-and-higher-order-markov-chains

Graph Analytics:

Although tensor
problematic, it may provide more general case
solutions, and some work leverages Spark:

The Tensor Renaissan

Anima Anandku

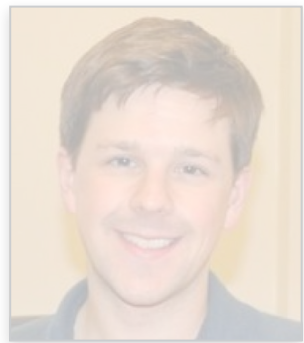
radar.oreilly.com/ r-renaissance-in-da

watch
this space
carefully

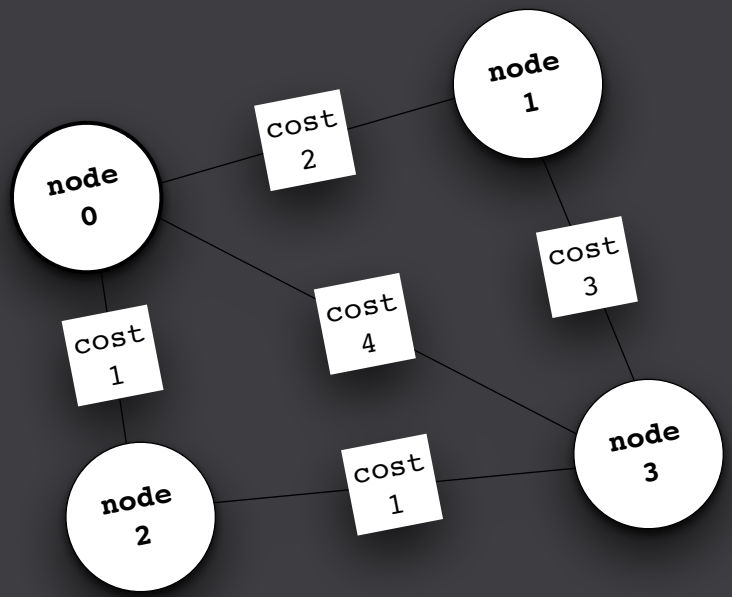
Spacey Random Walk arkov Chains

David Gleich

slideshare.net/dgleich/spacey-random-walks-and-higher-order-markov-chains



GraphX examples



GraphX:

spark.apache.org/docs/latest/graphx-programming-guide.html

Key Points:

- graph-parallel systems
- importance of workflows
- optimizations

GraphX: *Further Reading…*

PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs
J. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin
graphlab.org/files/osdi2012-gonzalez-low-gu-bickson-guestrin.pdf

Pregel: Large-scale graph computing at Google
Grzegorz Czajkowski, et al.
googleresearch.blogspot.com/2009/06/large-scale-graph-computing-at-google.html

GraphX: Unified Graph Analytics on Spark
Ankur Dave, Databricks
databricks-training.s3.amazonaws.com/slides/graphx@sparksummit_2014-07.pdf

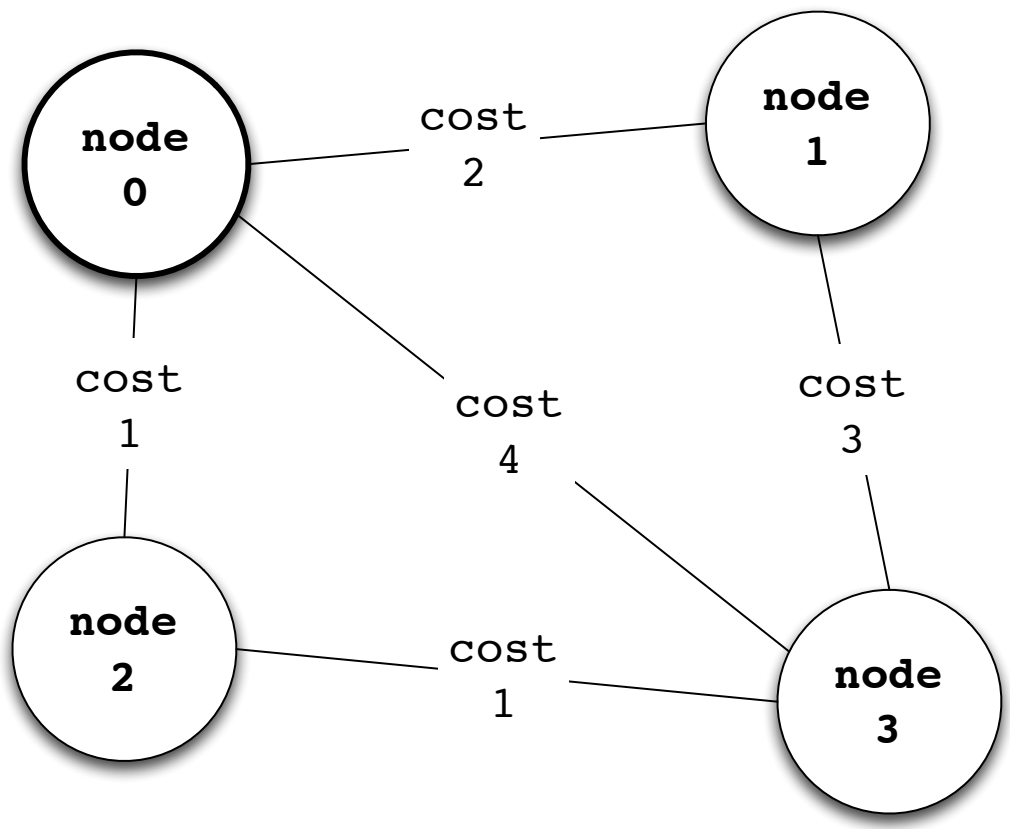
Advanced Exercises: GraphX
databricks-training.s3.amazonaws.com/graph-analytics-with-graphx.html

GraphX: *Example – simple traversals*

```
// http://spark.apache.org/docs/latest/graphx-programming-guide.html

import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

case class Peep(name: String, age: Int)

val nodeArray = Array(
  (1L, Peep("Kim", 23)), (2L, Peep("Pat", 31)),
  (3L, Peep("Chris", 52)), (4L, Peep("Kelly", 39)),
  (5L, Peep("Leslie", 45))
  )
val edgeArray = Array(
  Edge(2L, 1L, 7), Edge(2L, 4L, 2),
  Edge(3L, 2L, 4), Edge(3L, 5L, 3),
  Edge(4L, 1L, 1), Edge(5L, 3L, 9)
  )

val nodeRDD: RDD[(Long, Peep)] = sc.parallelize(nodeArray)
val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
val g: Graph[Peep, Int] = Graph(nodeRDD, edgeRDD)

val results = g.triplets.filter(t => t.attr > 7)

for (triplet <- results.collect) {
  println(s"${triplet.srcAttr.name} loves ${triplet.dstAttr.name}")
}
```

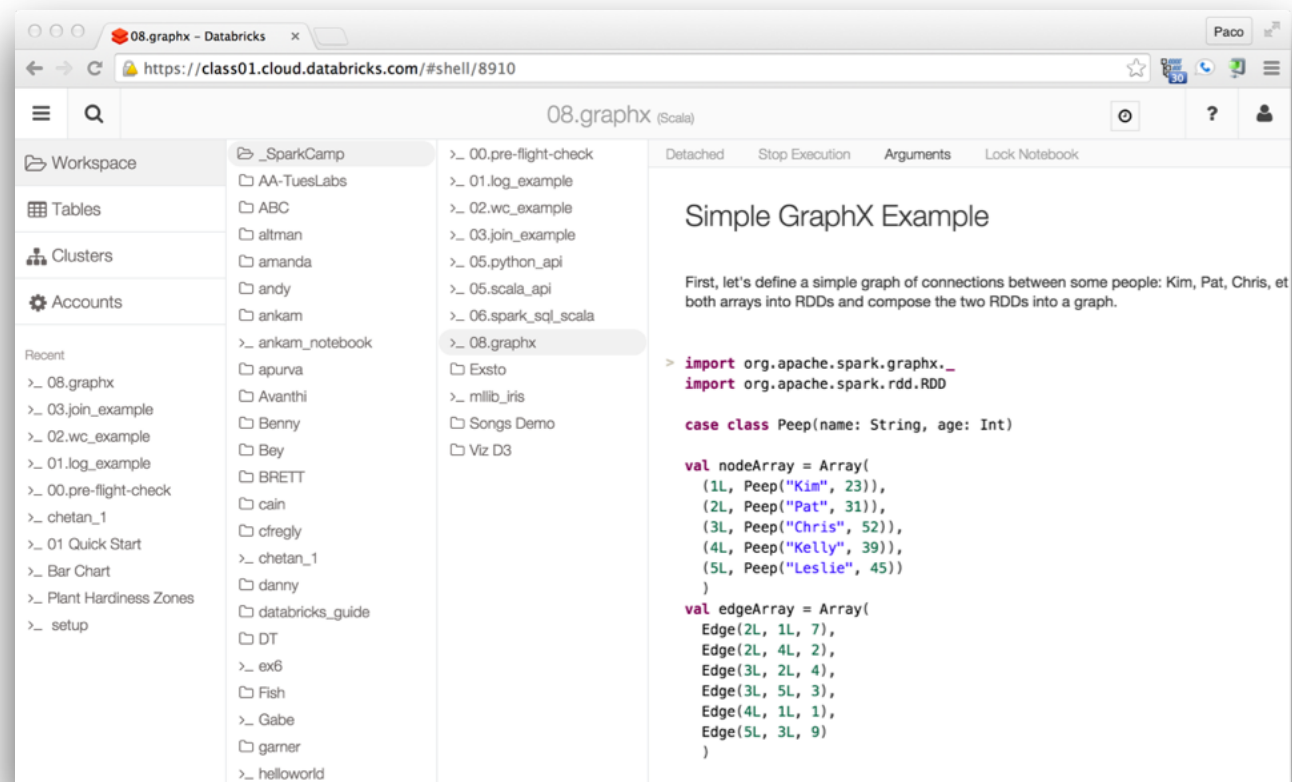
GraphX: *Example – routing problems*

What is the cost to reach **node 0** from any other node in the graph? This is a common use case for graph algorithms, e.g., **Djikstra**



GraphX: *Coding Exercise*

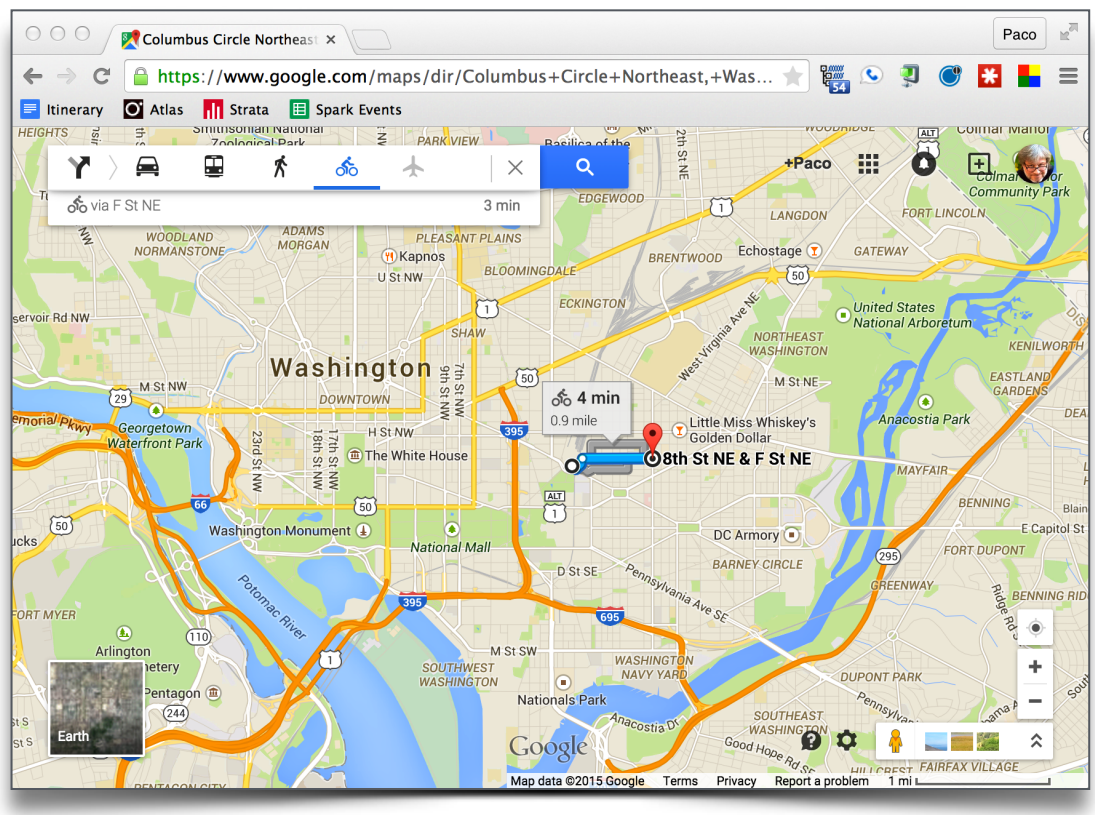
Clone and run `/_SparkCamp/08.graphx` in your folder:



Bikeshare Data Set

Bikeshare: *Data Set*

Clone and run `/_SparkCamp/Bike Share/` in your folder:



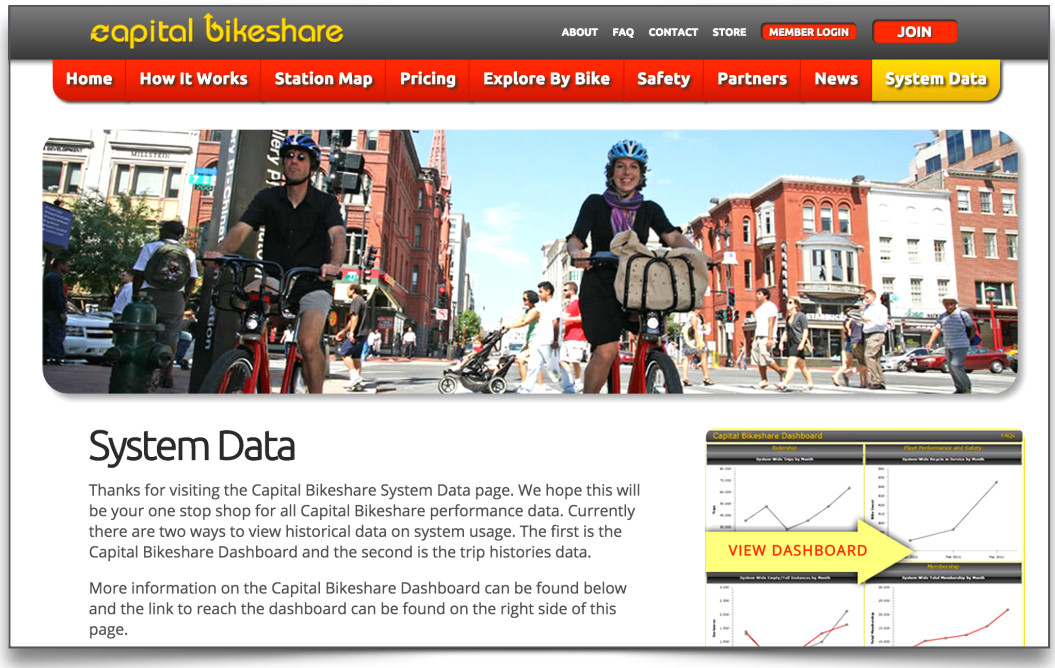
Bikeshare: *Data Set*

For an example data set, we will use *trip history* data from Capital Bikeshare in the Washington DC area:

capitalbikeshare.com/system-data

This data set records bikeshare trips during the first quarter of 2014:

- *trip duration*
- *start date/time*
- *end date/time*
- *start station*
- *end station*
- *bike ID #*
- *member type*



Bikeshare: *Code, etc.*

All of the code shown here is available online as a GitHub public repo:

https://github.com/ceteri/intro_spark

Analytics will focus on a DC bike routes near the following area on a Google map:

http://goo.gl/sAOdSv

Bikeshare: *ETL from downloaded CSV data*

```
// load the bikeshare data set
val raw_trips = sc.textFile("2014-Q1-Trips-History-Data2.csv")
raw_trips.take(4)


def convertDur(dur: String): Long = {
  val dur_regex = """(\d+)h\s(\d+)m\s(\d+)s""".r
  val dur_regex(hour, minute, second) = dur
  (hour.toLong * 3600L) + (minute.toLong * 60L) + second.toLong
}


case class Trip(id: String, dur: Long, s0: String, s1: String, reg: String)


val bike_trips = raw_trips.map(_.split(",")).filter(_(0) != "Duration").
  map(r => Trip(r(5), convertDur(r(0)), r(2), r(4), r(6)))


bike_trips.cache()
```

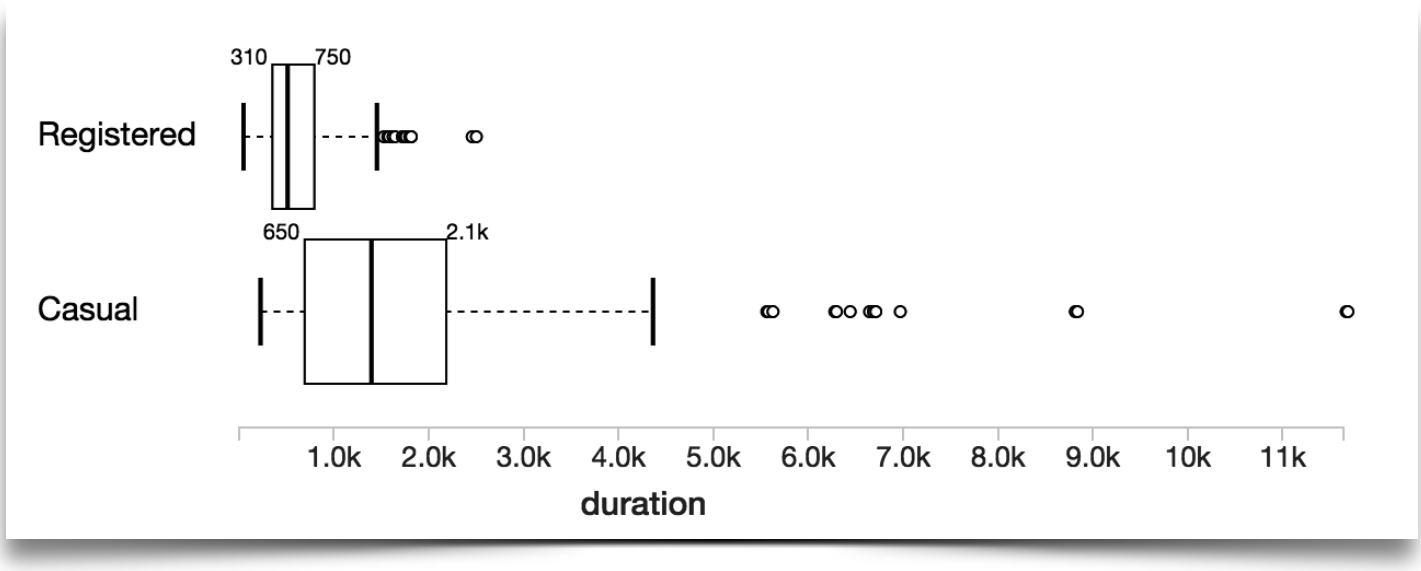
Bikeshare: *Explore the data set with Spark SQL*

```
// use DataFrames to explore the data
val bike_df = bike_trips.toDF()
bike_df.registerTempTable("bikeshare")

sql("SELECT * FROM bikeshare LIMIT 10").show()


val query = """
SELECT COUNT(*) AS num, s1, s0
FROM bikeshare
GROUP BY s0, s1
ORDER BY num DESC LIMIT 10
"""

sql(query).show()


// compare results with Google Maps, 8th NE & F St NE to Columbus Circle
// http://goo.gl/sAOdSv
```

Bikeshare: *What can we model within this data set?*

```
// could this relationship be used to produce classifiers?
bike_df.groupBy("reg").agg(bike_df("reg"), avg(bike_df("dur"))).show()
```

`member type` as a dependent variable,
`duration`, `station0`, `station1` as independent variables



Bikeshare: *Create feature vectors for building classifiers*

```
// featurize the data into vectors using MLlib
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors

val station_map = bike_trips.map(_.s0).union(bike_trips.map(_.s1)).
  distinct().zipWithUniqueId().collectAsMap()

val label_map = Map("Registered" -> 0.0, "Casual" -> 1.0)

var l_bike = bike_trips.map{ t =>
  var s0 = station_map.get(t.s0).getOrElse(0L).toDouble
  var s1 = station_map.get(t.s1).getOrElse(0L).toDouble

  LabeledPoint(label_map(t.reg), Vectors.dense(t.dur, s0, s1))
}


// create a train/test split
val splits = l_bike.randomSplit(Array(0.6, 0.4), seed = 11L)
val train_set = splits(0).cache()
val test_set = splits(1)
val n = test_set.count()
```

Bikeshare: *Build a model using Naïve Bayes*

```
import org.apache.spark.mllib.classification.NaiveBayes

val model = NaiveBayes.train(train_set, lambda = 1.0)

val pred = test_set.map(t => (model.predict(t.features), t.label))
val cm = sc.parallelize(pred.countByValue().toSeq)
val cm_nb = cm.map(x => (x._1, (1.0 * x._2 / n))).sortBy(_._1, true)
cm_nb.foreach(println)
```

Bikeshare: *Build a model using a Decision Tree*

```
import org.apache.spark.mllib.tree.DecisionTree

val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32

val model = DecisionTree.trainClassifier(train_set, numClasses,
 categoricalFeaturesInfo, impurity, maxDepth, maxBins)

val pred = test_set.map(t => (model.predict(t.features), t.label))
val cm = sc.parallelize(pred.countByValue().toSeq)
val cm_dt = cm.map(x => (x._1, (1.0 * x._2 / n))).sortBy(_._1, true)
cm_dt.foreach(println)
```

Bikeshare: *Build a model using Random Forest*

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel

val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 3 // use more in practice
val featureSubsetStrategy = "auto" // let the algorithm choose
val impurity = "gini"
val maxDepth = 4
val maxBins = 32

val model = RandomForest.trainClassifier(train_set, numClasses,
  categoricalFeaturesInfo, numTrees, featureSubsetStrategy, impurity,
  maxDepth, maxBins)

val pred = test_set.map(t => (model.predict(t.features), t.label))
val cm = sc.parallelize(pred.countByValue().toSeq)
val cm_rf = cm.map(x => (x._1, (1.0 * x._2 / n))).sortBy(_._1, true)
cm_rf.foreach(println)
```

Bikeshare: *Compare the models, using their confusion matrices*

```
// compare models
case class TripReport(predicted: Int, observed: Int,
  model: String, frequency: Double)

val part0 = cm_nb.map(x => TripReport(x._1._1.toInt, x._1._2.toInt, "0.NB", x._2))
val part1 = cm_dt.map(x => TripReport(x._1._1.toInt, x._1._2.toInt, "1.DT", x._2))
val part2 = cm_rf.map(x => TripReport(x._1._1.toInt, x._1._2.toInt, "2.RF", x._2))

val cm_df = part0.union(part1).union(part2).toDF()
cm_df.sort("predicted", "observed", "model").show()
```

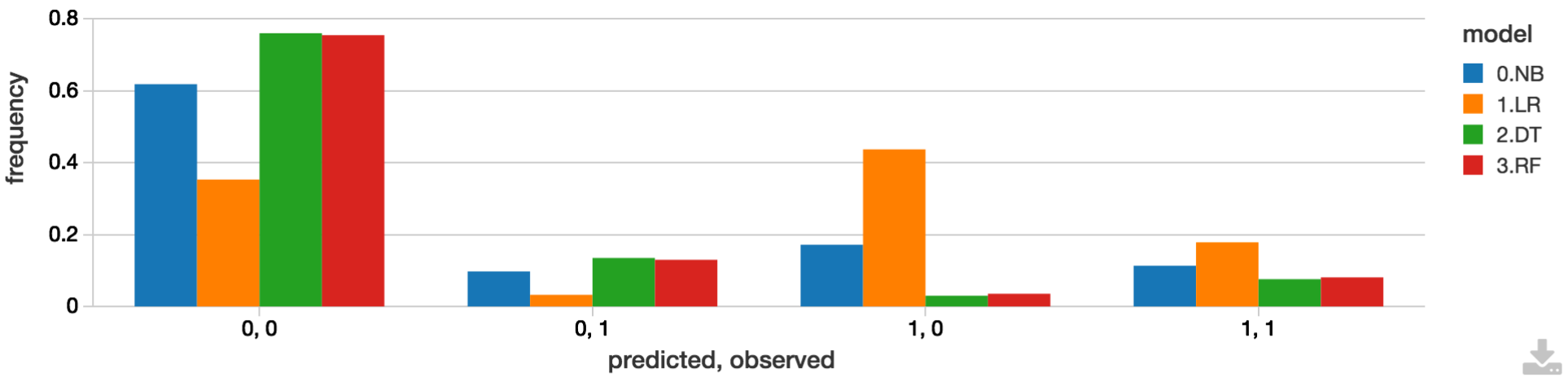
Bikeshare: *Modeling summary*

Naïve Bayes: simple to use (less parameters), produces a highly transparent model

Decision Tree: better predictive power, but more parameters

Random Forest: predictive errors differed, even more parameters – and could have used more trees

Bikeshare: *Modeling summary*



Command took 0.69s

Bikeshare: *What is Parquet?*

Parquet is a columnar format, supported by many different Big Data frameworks

http://parquet.io/

Spark SQL supports read/write of parquet files, automatically preserving the schema of the original data (HUGE benefits)

See also:

Efficient Data Storage for Analytics with Parquet 2.0
Julien Le Dem @Twitter
slideshare.net/julienledem/th-210pledem

Bikeshare: *Store the prepared data using Parquet serialization*

```
case class TripEx(id: String, reg: String, dur: Long,
  s0: String, s1: String, sta0: Long, sta1: Long)

val bike_trips_ex = bike_trips.map{ t =>
  var sta0 = station_map.get(t.s0).getOrElse(0L)
  var sta1 = station_map.get(t.s1).getOrElse(0L)

  TripEx(t.id, t.reg, t.dur, t.s0, t.s1, sta0, sta1)
}.toDF()

bike_trips_ex.take(2)
bike_trips_ex.saveAsParquetFile("bike.parquet")

// compare the relative compression rates on disk
```

Bikeshare: *Load the Parquet data set*

```
val bike_trips = sqlContext.parquetFile("bike.parquet").cache()
bike_trips.registerTempTable("biketrips")
bike_trips.printSchema()

sql("SELECT * FROM biketrips LIMIT 10").show()

val query = """
SELECT
 COUNT(*) AS num, MIN(dur) AS min_dur, MAX(dur) AS max_dur, s0, s1
FROM biketrips
WHERE NOT s0 = s1
GROUP BY s0, s1
ORDER BY num DESC
LIMIT 10
"""

sql(query).show()
// minimum durations between Columbus Circle and 8th & F St NE are ~3 minutes
// as Google Maps predicts http://goo.gl/sAOdSv
```

Bikeshare: *Compose a graph in GraphX*

```
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

// build the node list
val sta0_rdd = bike_trips.map(r =>
 (r(5).asInstanceOf[Long], r(3).asInstanceOf[String]))
val sta1_rdd = bike_trips.map(r =>
 (r(6).asInstanceOf[Long], r(4).asInstanceOf[String]))
val nodeRDD = sta0_rdd.union(sta1_rdd).distinct()
nodeRDD.take(2)

// build the edge list
val edge_kv = bike_trips.map(r => ((r(5), r(6)), r(2))).groupByKey()
edge_kv.take(2)

def median(s: Seq[Long]) = {
  val (lower, upper) = s.sortWith(_ < _).splitAt(s.size / 2)
  if (s.size % 2 == 0) (lower.last + upper.head) / 2.0 else upper.head
}

val edgeRDD = edge_kv.map{ r =>
  val med = median(r._2.map(_.asInstanceOf[Long]).toSeq)
  Edge(r._1._1.asInstanceOf[Long], r._1._2.asInstanceOf[Long], med)
}
edgeRDD.take(2)

val g: Graph[String, Double] = Graph(nodeRDD, edgeRDD)
```

Bikeshare: *PageRank using Pregel in GraphX*

```
val ranks = g.pageRank(0.0001).vertices
ranks.take(10)

case class Rank(id: Long, rank: Double, station: String)

val rank_df = ranks.join(nodeRDD).map(r =>
  Rank(r._1.toLong, r._2._1, r._2._2)).toDF()

rank_df.registerTempTable("ranks")


// which are the most popular bikeshare stations?
sql("SELECT * FROM ranks ORDER BY rank DESC LIMIT 10").show()
```

Bikeshare: *Initialize SSSP to find the shortest path between stations*

```
// find "id" values for the two most popular stations
sql("SELECT * FROM ranks WHERE station LIKE 'Columbus Circle%' ").show()

sql("SELECT * FROM ranks WHERE station LIKE '8th%' ").show()


// initialize for Columbus Circle
val sourceId: VertexId = 190

val initialGraph :
  Graph[(Double, List[VertexId]), Double] = g.mapVertices((id, _) =>
  if (id == sourceId) (0.0, List[VertexId](sourceId))
  else (Double.PositiveInfinity, List[VertexId]()))
```

Bikeshare: *SSSP implementation using Pregel in GraphX*

```
val sssp = initialGraph.pregel((Double.PositiveInfinity, List[VertexId]()),
Int.MaxValue, EdgeDirection.Out)(
  // vertex program
  (id, dist, newDist) => if (dist._1 < newDist._1) dist else newDist,

  // send message
  triplet => {
    if (triplet.srcAttr._1 < triplet.dstAttr._1 - triplet.attr ) {
      Iterator((triplet.dstId, (triplet.srcAttr._1 + triplet.attr ,
triplet.srcAttr._2 :+ triplet.dstId)))
    } else {
      Iterator.empty
    }
  },

  // merge message
  (a, b) => if (a._1 < b._1) a else b)

  println(sssp.vertices.collect.mkString("\n")
)
```

Bikeshare: *Compare results with Google Maps "directions"*

```
sssp.vertices.collect.foreach(println)

// to confirm about the Google Maps estimates
sssp.vertices.filter(_._1 == 274).collect()

225/60.

sql("SELECT * FROM ranks WHERE station LIKE 'Lincoln%' ").show()

sssp.vertices.filter(_._1 == 249).collect()

sql("SELECT * FROM ranks WHERE id = 223 ").show()
```

A Big Picture

A Big Picture...

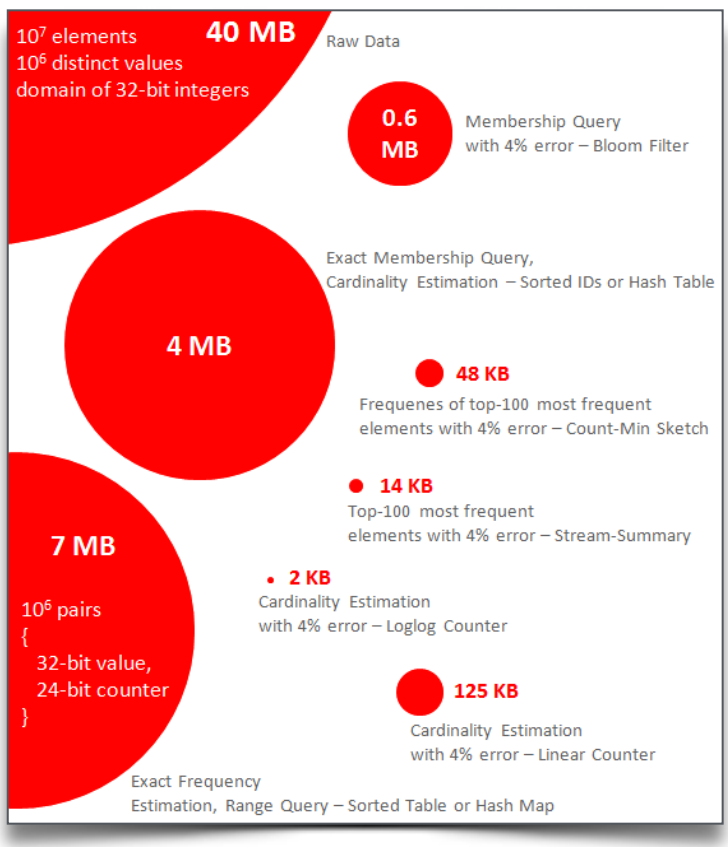
19-20c. statistics emphasized *defensibility* in lieu of *predictability*, based on analytic variance and goodness-of-fit tests

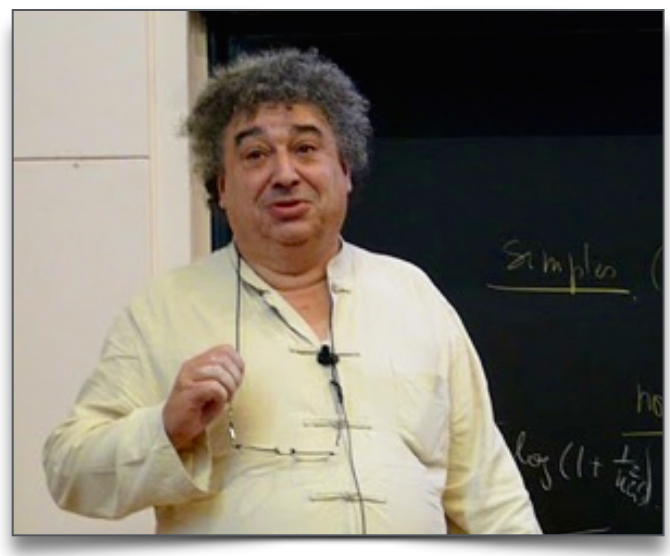
That approach inherently led toward a manner of computational thinking based on **batch windows**

They missed a subtle point…

A Big Picture… The view in the lens has changed

21c. shift towards modeling based on probabilistic approximations: trade bounded errors for greatly reduced resource costs

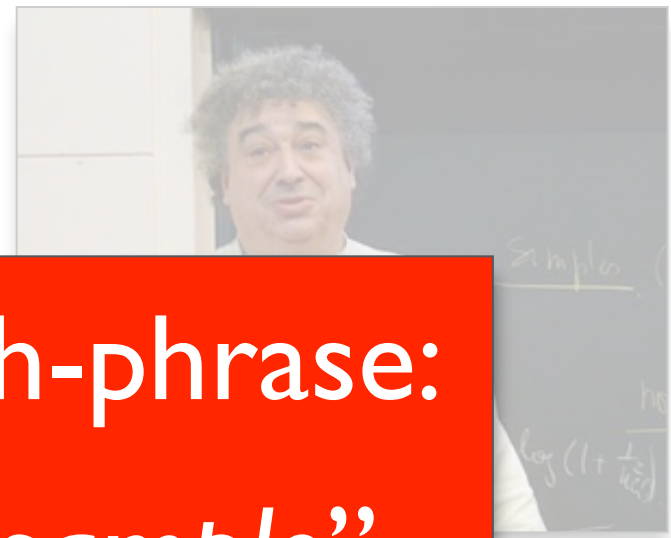




highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/

A Big Picture... The view in the lens has changed

21c. shift towards modeling based on probabil approximations: trade bounded errors for greatly reduced resource costs



Twitter catch-phrase:

"Hash, don't sample"

highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/

Probabilistic Data Structures:

a fascinating and relatively new area, pioneered by relatively few people – e.g., **Philippe Flajolet**

provides *approximation*, with error bounds – in general uses significantly less resources (RAM, CPU, etc.)

many algorithms can be constructed from combinations of read and write *monoids*

aggregate different ranges by composing hashes, instead of repeating full-queries

Probabilistic Data Structures: *Some Examples*

algorithm	*use case*	*example*
Count-Min Sketch	frequency summaries	**code**
HyperLogLog	set cardinality	**code**
Bloom Filter	set membership	
MinHash	set similarity	
DSQ	streaming quantiles	
SkipList	ordered sequence search	

Probabilistic Data Structures: *Performance Bottlenecks*

Add ALL the Things:
Abstract Algebra Meets Analytics

infoq.com/presentations/abstract-algebra-analytics

Avi Bryant, Strange Loop (2013)

Avi Bryant
@avibryant

- *grouping doesn't matter (associativity)*
- *ordering doesn't matter (commutativity)*
- *zeros get ignored*

In other words, while partitioning data at scale is quite difficult, you can let the math allow your code to be flexible at scale

Probabilistic Data Structures: *Performance Bottlenecks*

Algebra for Analytics

speakerdeck.com/johnynek/algebra-for-analytics

Oscar Boykin, *Strata SC* (2014)

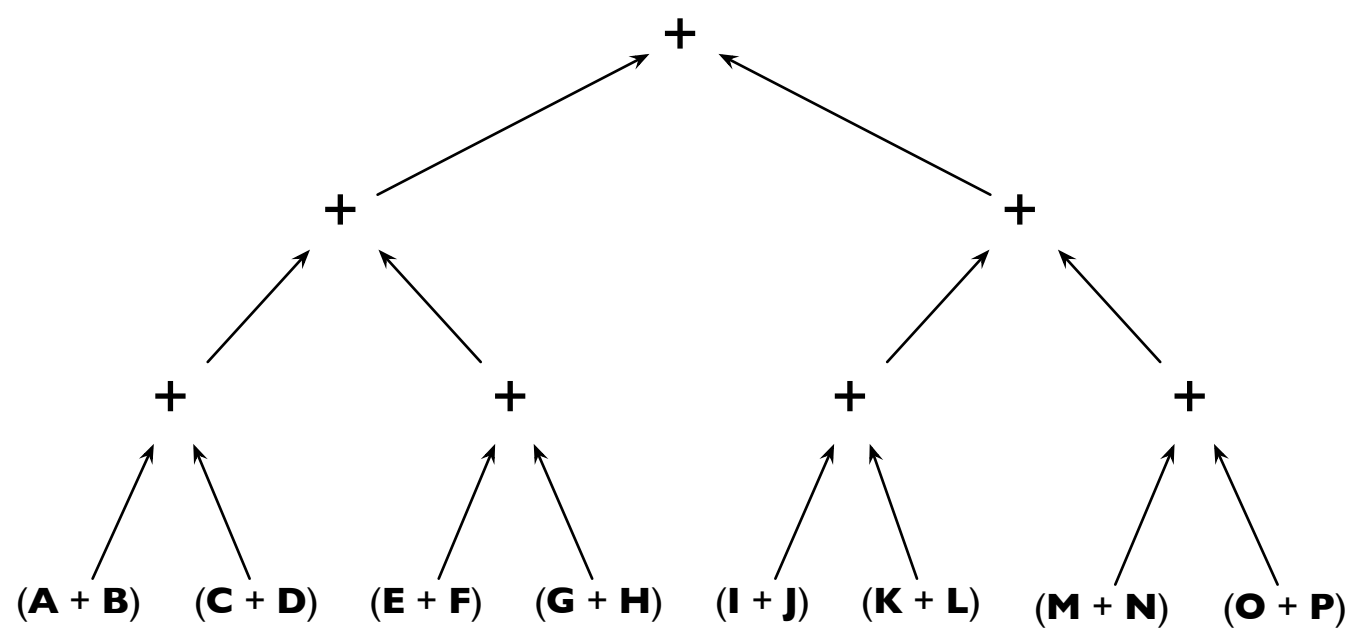
- *"Associativity allows parallelism in reducing"* by letting you put the () where you want
- *"Lack of associativity increases latency exponentially"*

Oscar Boykin
@posco

Algebra for Analytics
Oscar Boykin, *Strata SC* (2014)

A + B + C + D + E + F + G + H + I + J + K + L + M + N + O + P

(**A** + **B**)
+ **C**
+ **D**
+ **E**
+ **F**
+ **G**
+ **H**
+ **I**
+ **J**
+ **K**
+ **L**
+ **M**
+ **N**
+ **O**
+ **P**

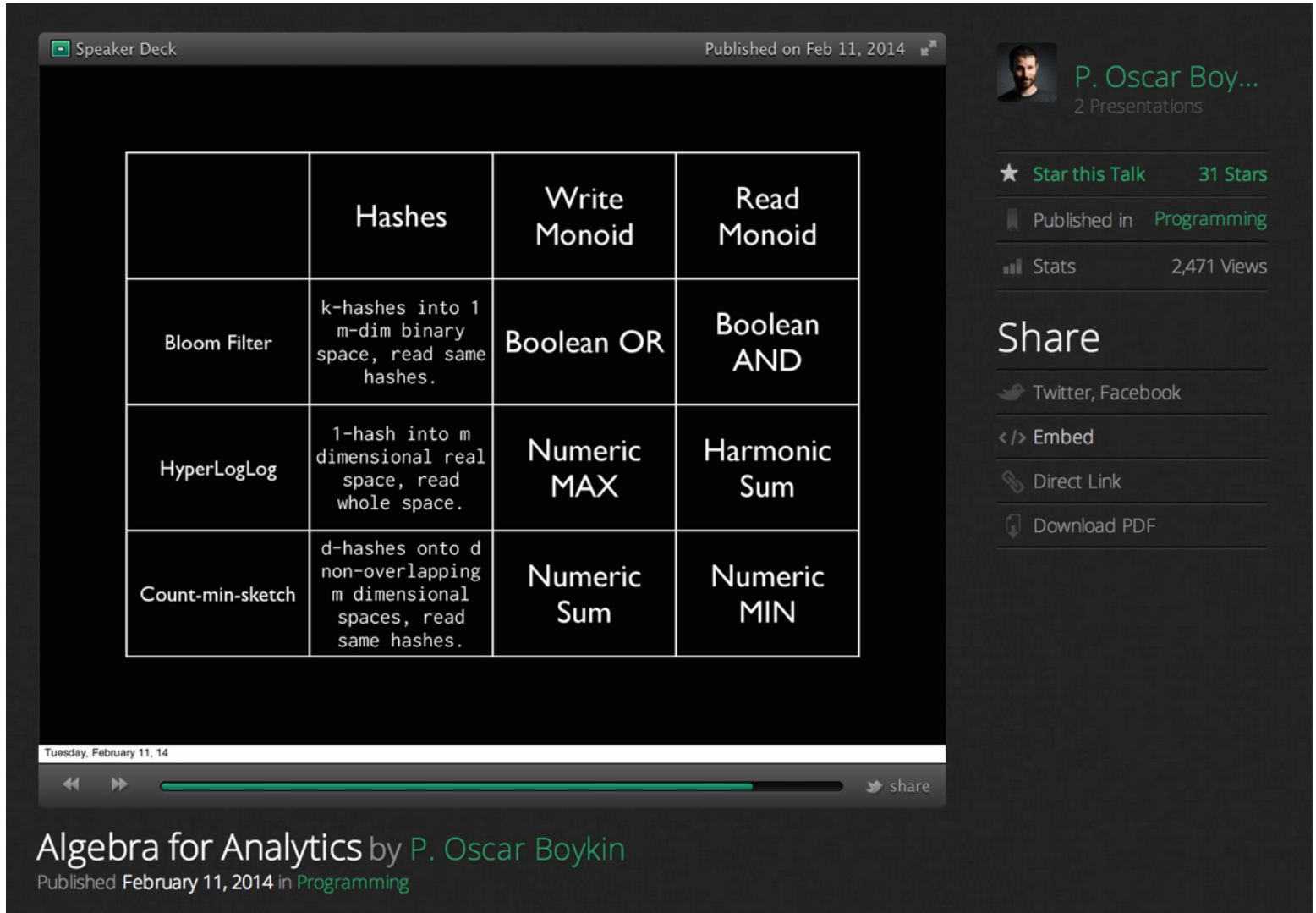


latency = (**N** - 1) = 15

latency = log2(**N**) = 4

Probabilistic Data Structures: *Performance Bottlenecks*

Algebra for Analytics
Oscar Boykin, *Strata SC* (2014)



Abstract Algebra

A *semigroup* is a non-empty set with an associative binary operation. For example, addition of integers:

$$(2 + 3) + 4 = 2 + (3 + 4) = 9$$

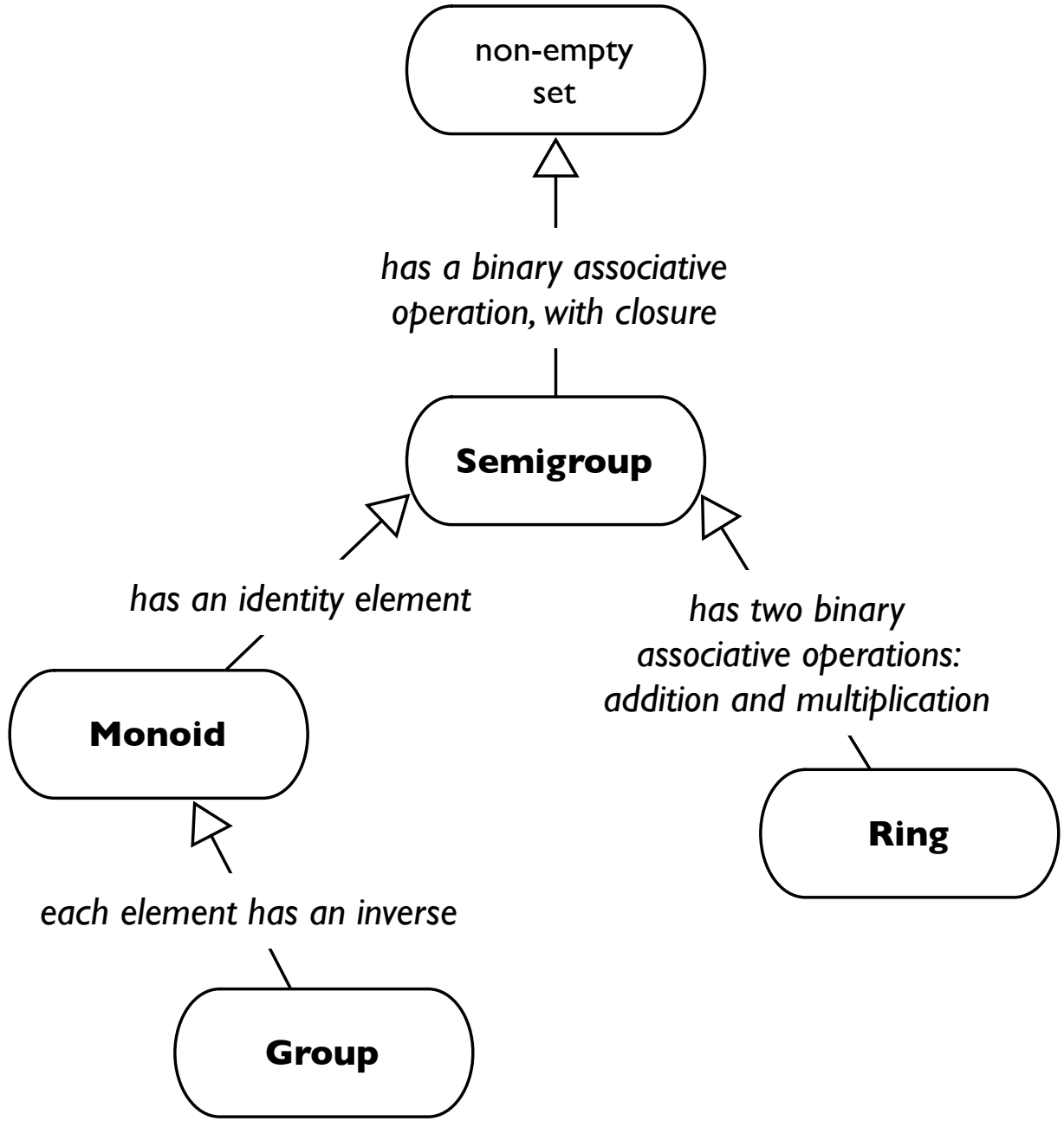
A *monoid* is a semigroup with an *identity element*:

$$2 + 0 = 2$$

That may seem trivial … until you need to *aggregate* billions of complex objects, especially with real-time requirements

Abstract Algebra

Categories:



Probabilistic Data Structures: *Recommended Reading*

Probabilistic Data Structures for Web Analytics and Data Mining
Ilya Katsov (2012-05-01)

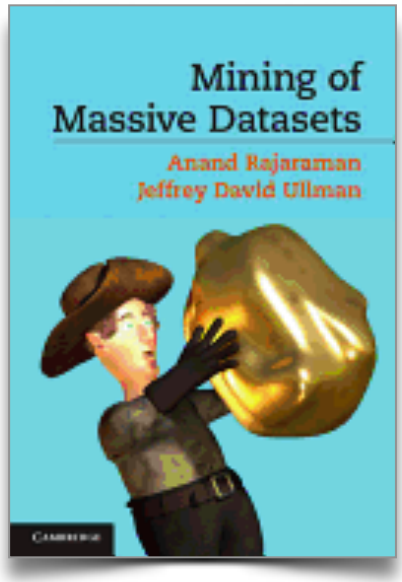
A collection of links for streaming algorithms and data structures
Debasish Ghosh

Aggregate Knowledge blog (now Neustar)
Timon Karnezos, **Matt Curcio**, et al.

Probabilistic Data Structures and Breaking Down Big Sequence Data
C. Titus Brown, O'Reilly (2010-11-10)

Algebird
Avi Bryant, **Oscar Boykin**, et al. Twitter (2012)

Mining of Massive Datasets
Jure Leskovec, **Anand Rajaraman**, **Jeff Ullman**, Cambridge (2011)



Demo: *PySpark Streaming Network Word Count*

```
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(appName="PyStreamNWC", master="local[*]")
ssc = StreamingContext(sc, 5)

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))

counts = lines.flatMap(lambda line: line.split(" ")) \
              .map(lambda word: (word, 1)) \
              .reduceByKey(lambda a, b: a+b)

counts.pprint()

ssc.start()
ssc.awaitTermination()
```

Demo: *PySpark Streaming Network Word Count - Stateful*

```
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

def updateFunc (new_values, last_sum):
    return sum(new_values) + (last_sum or 0)

sc = SparkContext(appName="PyStreamNWC", master="local[*]")
ssc = StreamingContext(sc, 5)
ssc.checkpoint("checkpoint")

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))

counts = lines.flatMap(lambda line: line.split(" ")) \
              .map(lambda word: (word, 1)) \
              .updateStateByKey(updateFunc) \
              .transform(lambda x: x.sortByKey())

counts.pprint()

ssc.start()
ssc.awaitTermination()
```

Further Resources + Q&A

Spark Developer Certification

- go.databricks.com/spark-certified-developer
- defined by Spark experts @Databricks
- assessed by O'Reilly Media
- establishes the bar for Spark expertise

Developer Certification: *Overview*

- 40 multiple-choice questions, 90 minutes
- mostly structured as choices among code blocks
- expect some Python, Java, Scala, SQL
- understand theory of operation
- identify best practices
- recognize code that is more parallel, less memory constrained

Overall, you need to write Spark apps in practice

Developer Certification: *Great Prep…*

Find and study the Spark Summit and Strata + HW talks by:

Vida Ha

Exam prep materials are in production at O'Reilly Media by:

Olivier Girardot

community:

spark.apache.org/community.html

events worldwide: goo.gl/2YqJZK

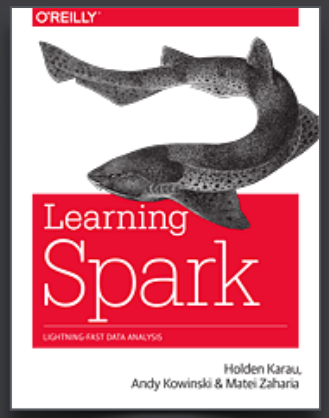
YouTube channel: goo.gl/N5Hx3h

video+preso archives: spark-summit.org

http://spark-summit.org/

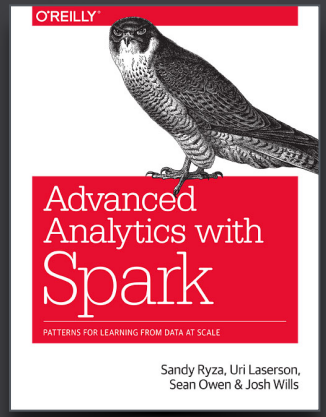
books+videos:

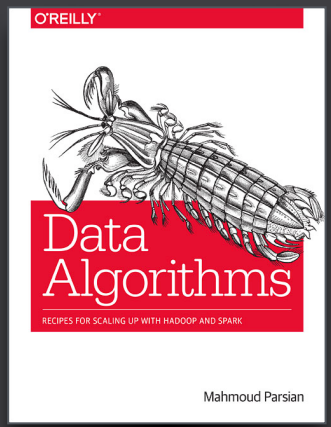
Learning Spark
Holden Karau, Andy Konwinski, Parick Wendell, Matei Zaharia
O'Reilly (2015)
shop.oreilly.com/product/06369200285 12.do



Intro to Apache Spark
Paco Nathan
O'Reilly (2015)
shop.oreilly.com/product/0636920036807.do

Advanced Analytics with Spark
Sandy Ryza, Uri Laserson, Sean Owen, Josh Wills
O'Reilly (2014)
shop.oreilly.com/product/0636920035091.do



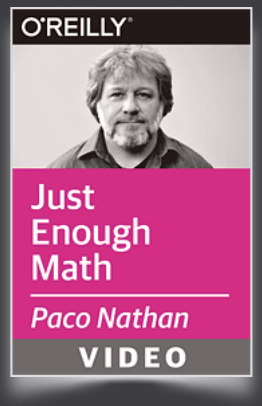


Data Algorithms
Mahmoud Parsian
O'Reilly (2015)
shop.oreilly.com/product/0636920033950.do

presenter:

monthly newsletter for updates,
events, conf summaries, etc.:

liber118.com/pxn/



Just Enough Math
O'Reilly (2014)

justenoughmath.com
preview: youtu.be/TQ58cWgdCpA