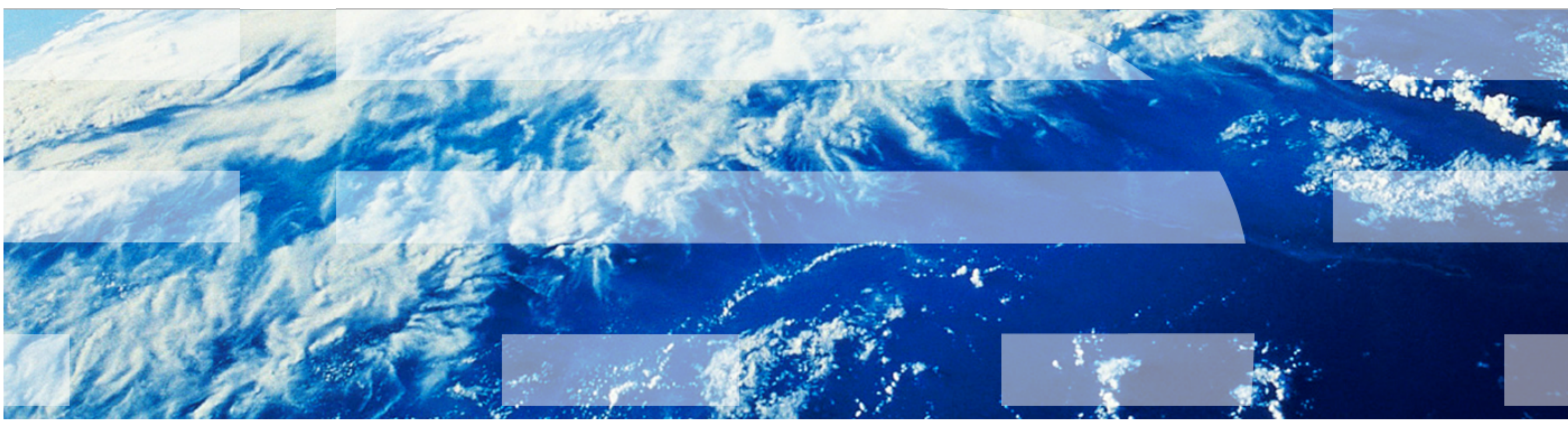
E6895 Advanced Big Data Analytics Lecture 3:

Big Data Foundations (II)

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science



February 8th, 2019

Key Components of Spark MLlib

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- Correlation
- Hypothesis testing

- Correlation: Calculating the correlation between two series of data is a common operation in Statistics
  - Pearson's Correlation
  - Spearman's Correlation

Spark ML Clustering

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- K-means
  - Input Columns
  - Output Columns
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Gaussian Mixture Model (GMM)
  - Input Columns
  - Output Columns

Example: clustering

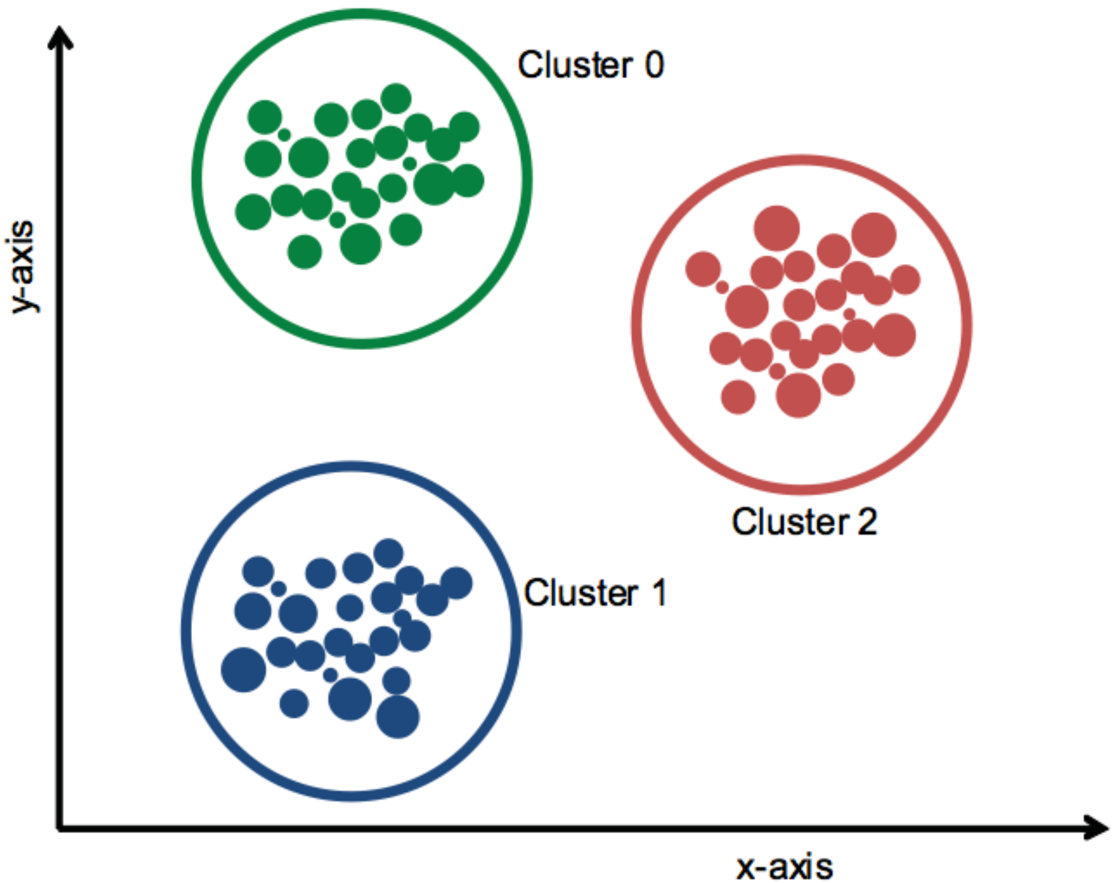
Feature Space

Clustering

Clustering a collection involves three things:

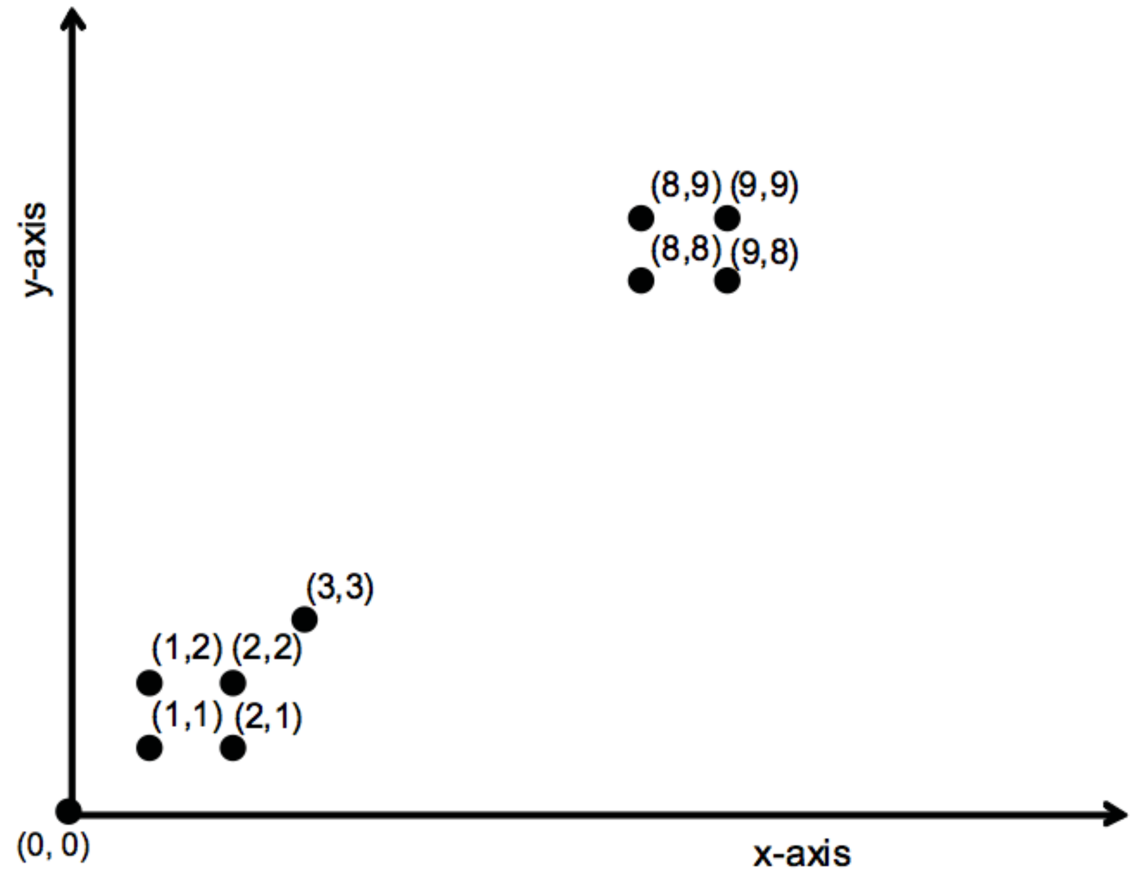
- *An algorithm*—This is the method used to group the books together.
- *A notion of both similarity and dissimilarity*—In the previous discussion, we relied on your assessment of which books belonged in an existing stack and which should start a new one.
- *A stopping condition*—In the library example, this might be the point beyond which books can't be stacked anymore, or when the stacks are already quite dissimilar.

Clustering — on feature plane

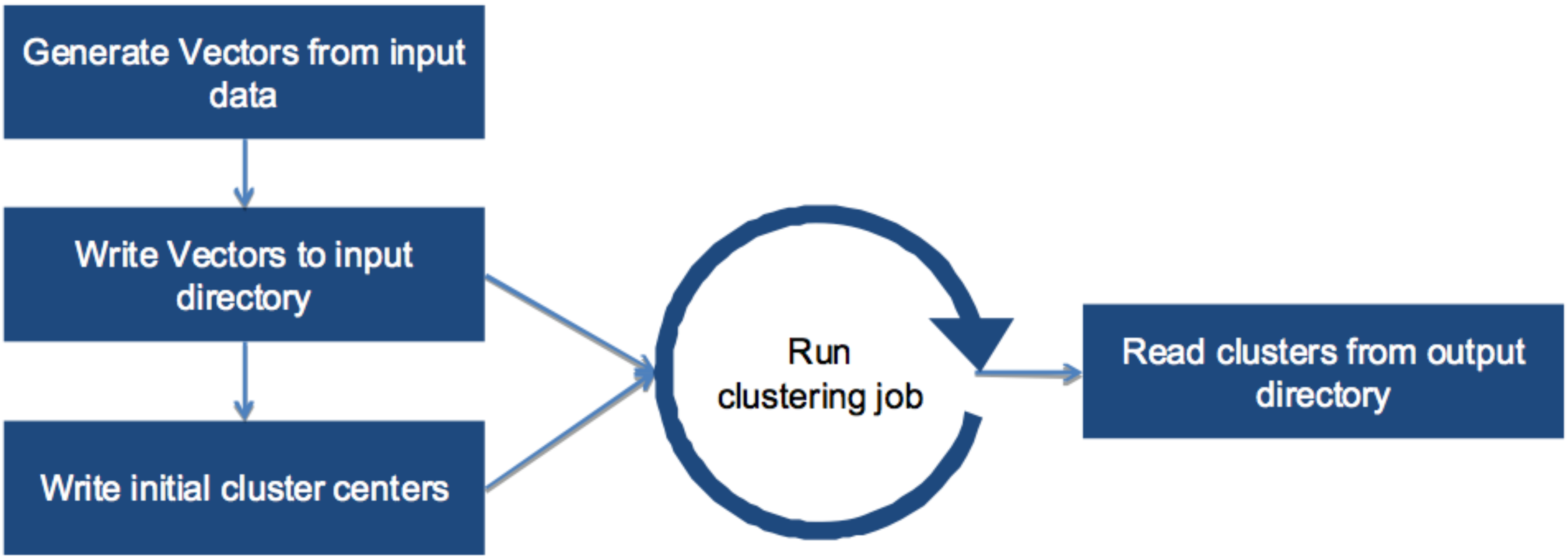


Clustering example

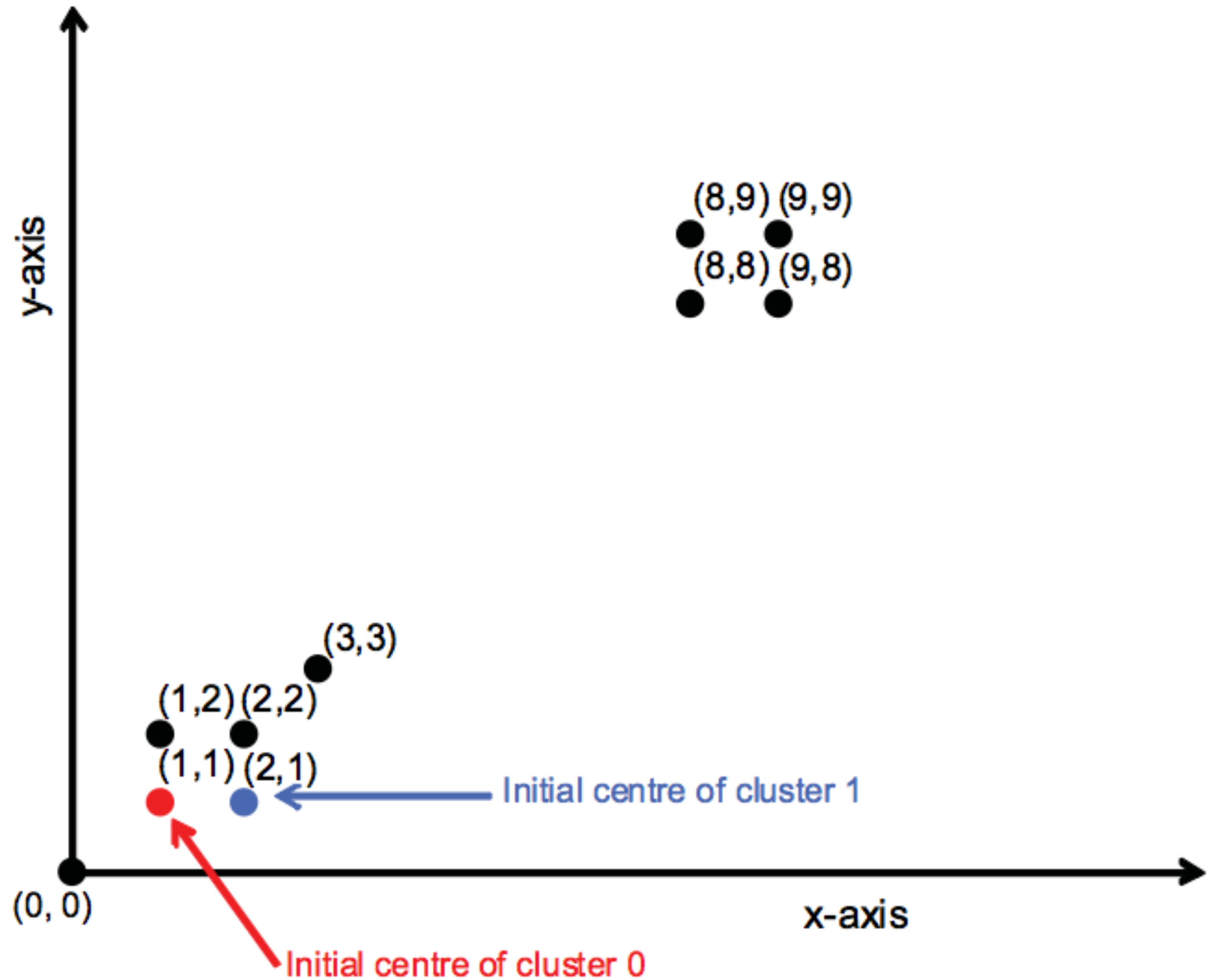
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



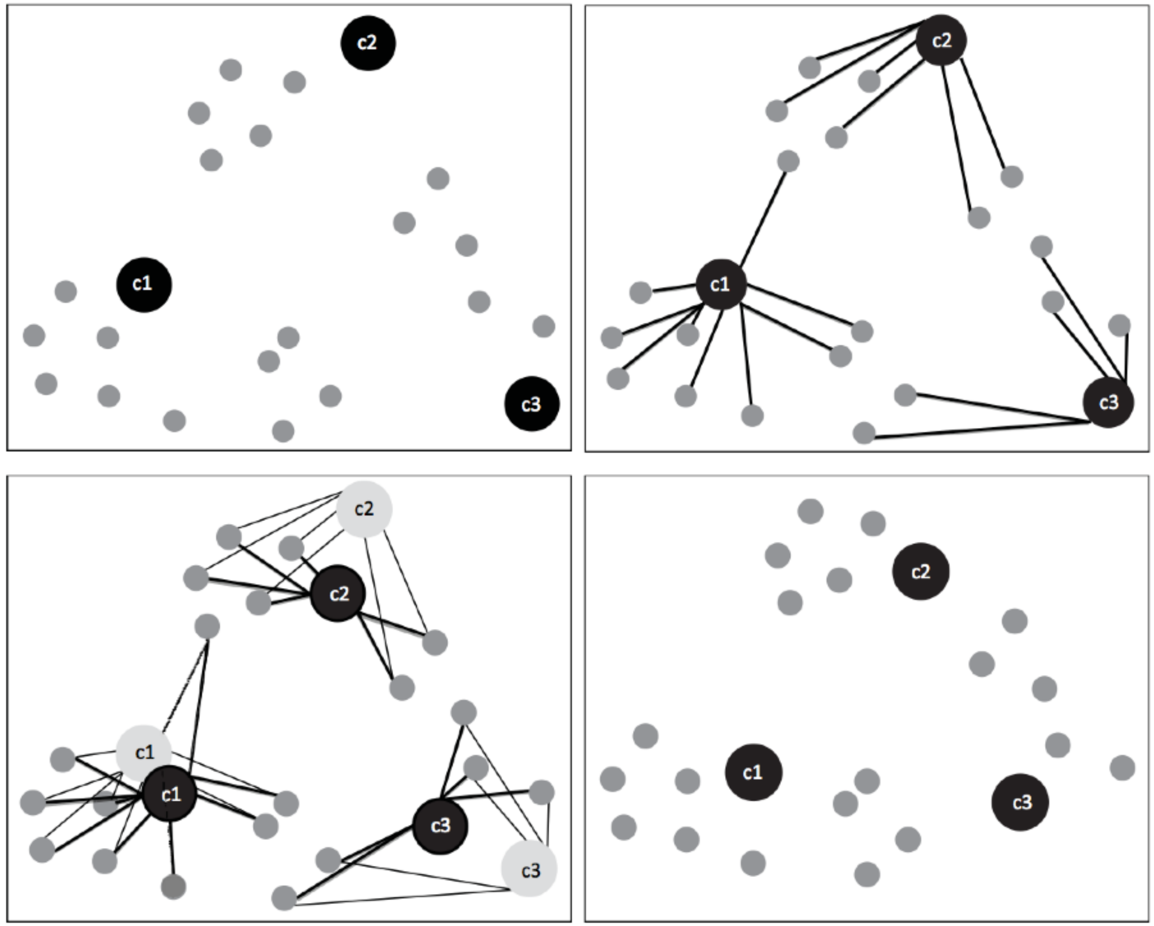
Steps on clustering



Making initial cluster centers



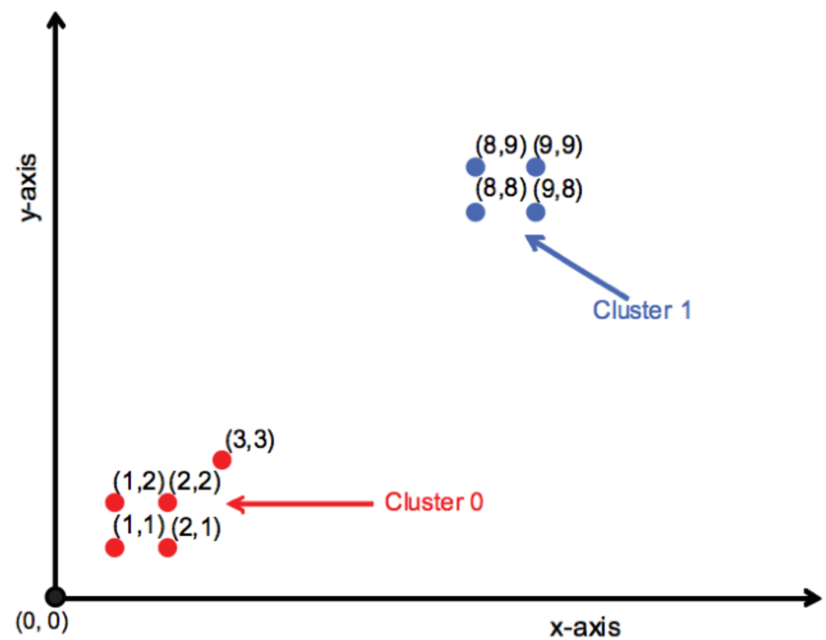
K-mean clustering



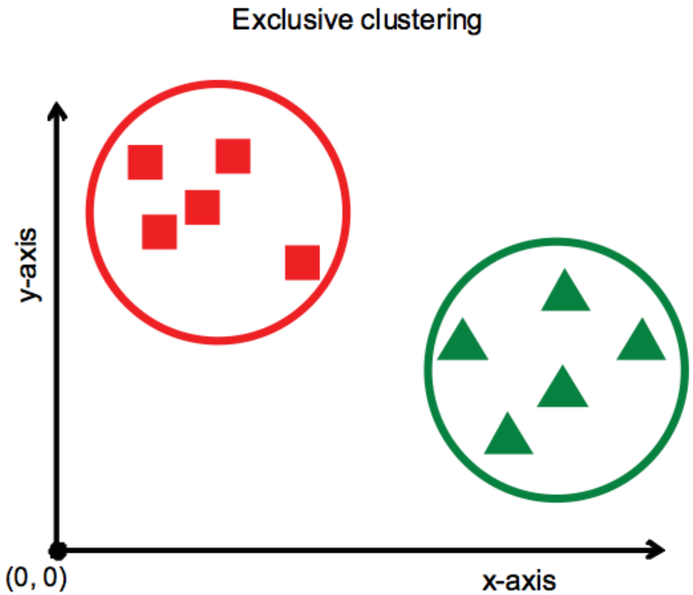
K-means clustering in action. Starting with three random points as centroids (top left), the map stage (top right) assigns each point to the cluster nearest to it. In the reduce stage (bottom left), the associated points are averaged out to produce the new location of the centroid, leaving you with the final configuration (bottom right). After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.

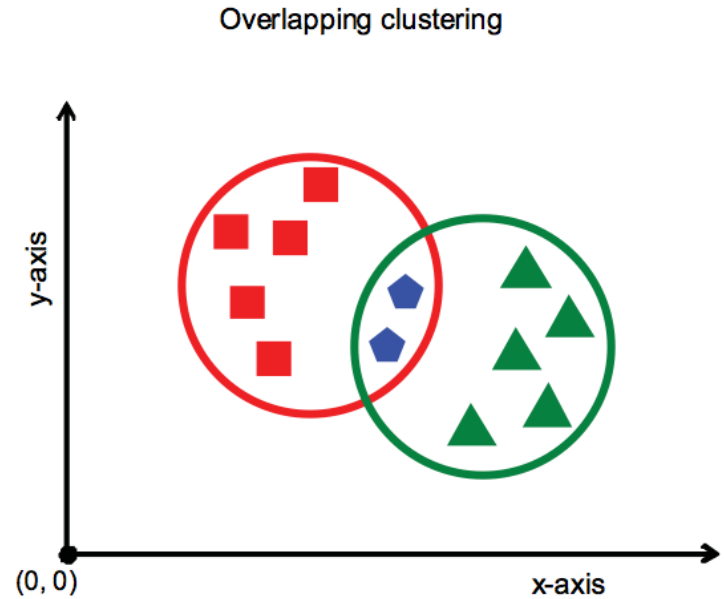
HelloWorld clustering scenario result

```
1.0: [1.000, 1.000] belongs to cluster 0
1.0: [2.000, 1.000] belongs to cluster 0
1.0: [1.000, 2.000] belongs to cluster 0
1.0: [2.000, 2.000] belongs to cluster 0
1.0: [3.000, 3.000] belongs to cluster 0
1.0: [8.000, 8.000] belongs to cluster 1
1.0: [9.000, 8.000] belongs to cluster 1
1.0: [8.000, 9.000] belongs to cluster 1
1.0: [9.000, 9.000] belongs to cluster 1
```

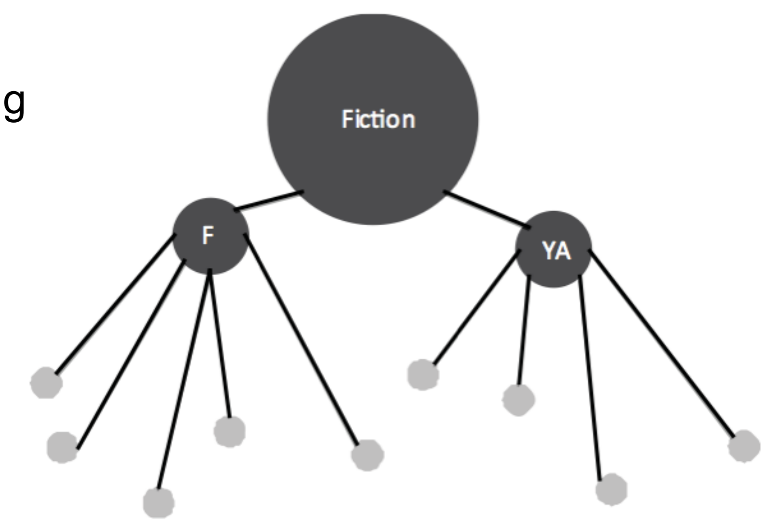


Other clustering algorithms





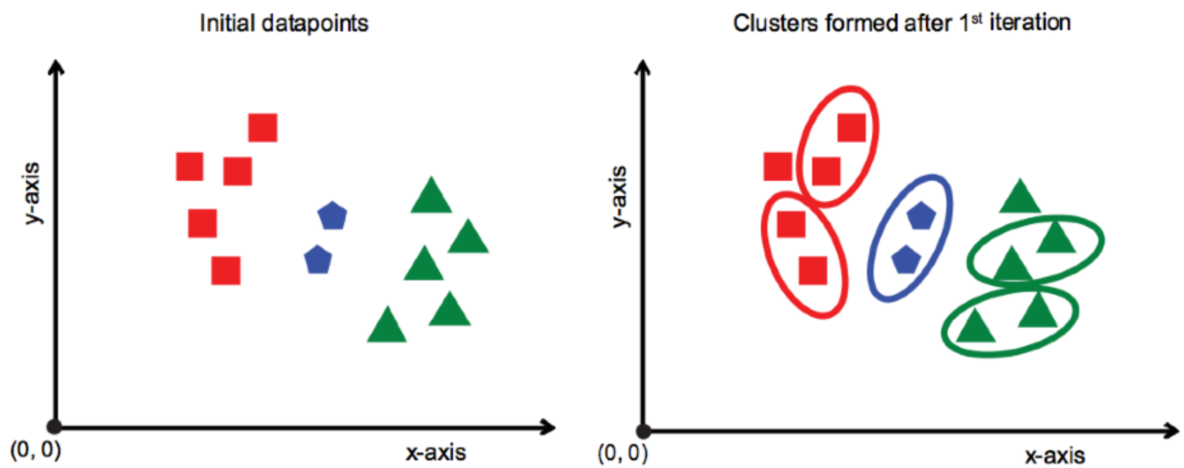
Hierarchical clustering



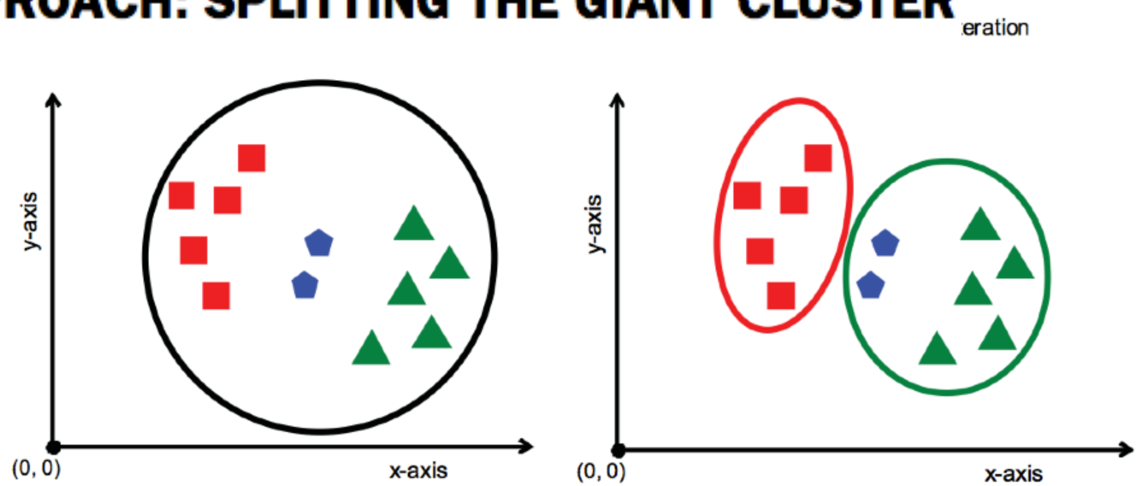
Different clustering approaches

FIXED NUMBER OF CENTERS

BOTTOM-UP APPROACH: FROM POINTS TO CLUSTERS VIA GROUPING



TOP-DOWN APPROACH: SPLITTING THE GIANT CLUSTER



Spark ML Classification and Regression

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- Classification
  - Logistic regression
    - Binomial logistic regression
    - Multinomial logistic regression
  - Decision tree classifier
  - Random forest classifier
  - Gradient-boosted tree classifier
  - Multilayer perceptron classifier
  - Linear Support Vector Machine
  - One-vs-Rest classifier (a.k.a. One-vs-All)
  - Naive Bayes
- Regression
  - Linear regression
  - Generalized linear regression
    - Available families
  - Decision tree regression
  - Random forest regression
  - Gradient-boosted tree regression
  - Survival regression
  - Isotonic regression

Spark ML Classification and Regression

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- Linear methods
- Decision trees
  - Inputs and Outputs
    - Input Columns
    - Output Columns
- Tree Ensembles
  - Random Forests
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)
  - Gradient-Boosted Trees (GBTs)
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)

Classification — definition

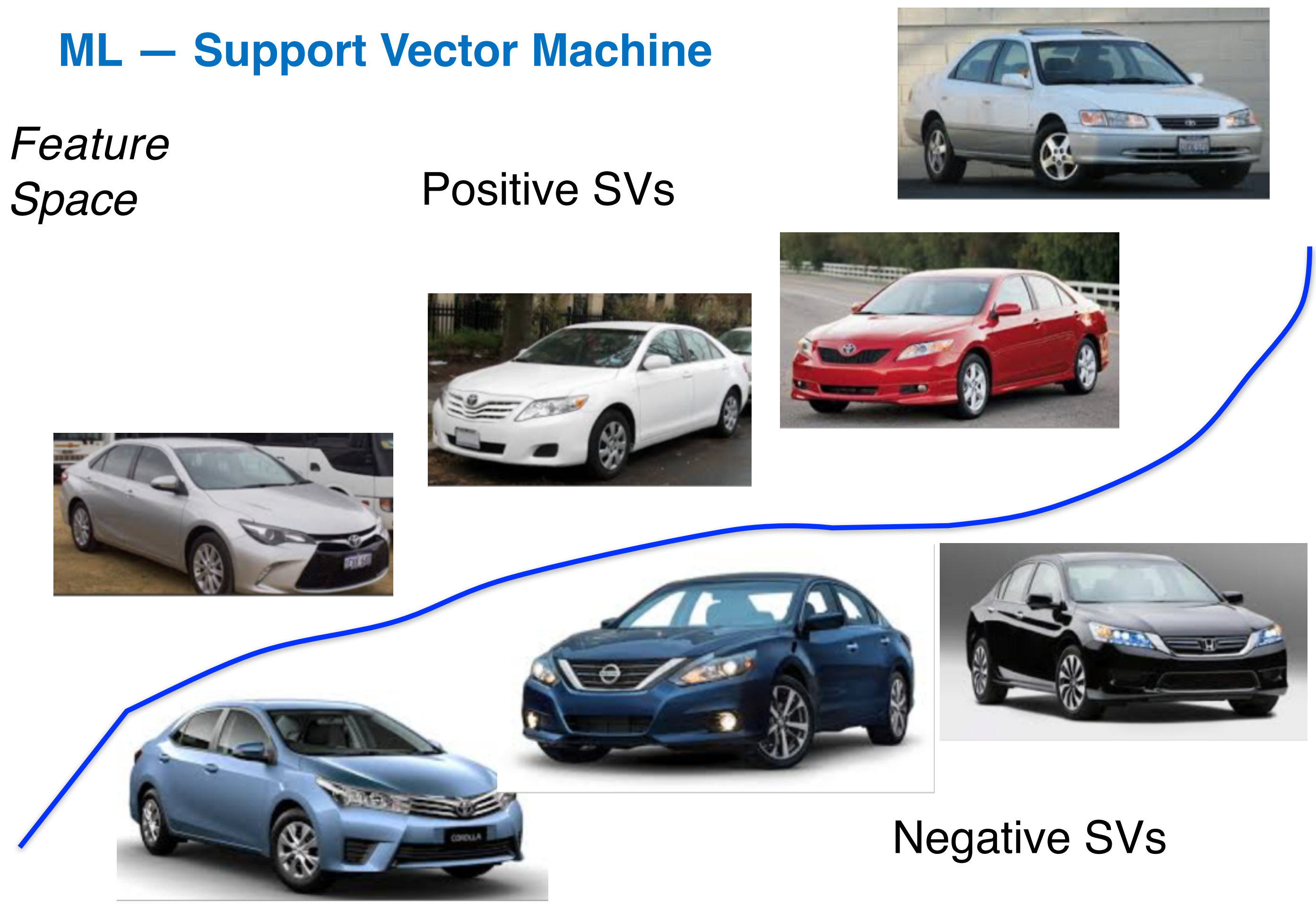
DEFINITION Computer classification systems are a form of machine learning that use learning algorithms to provide a way for computers to make decisions based on experience and, in the process, emulate certain forms of human decision making.

Classification example: using SVM to recognize a Toyota Camry

Non-ML

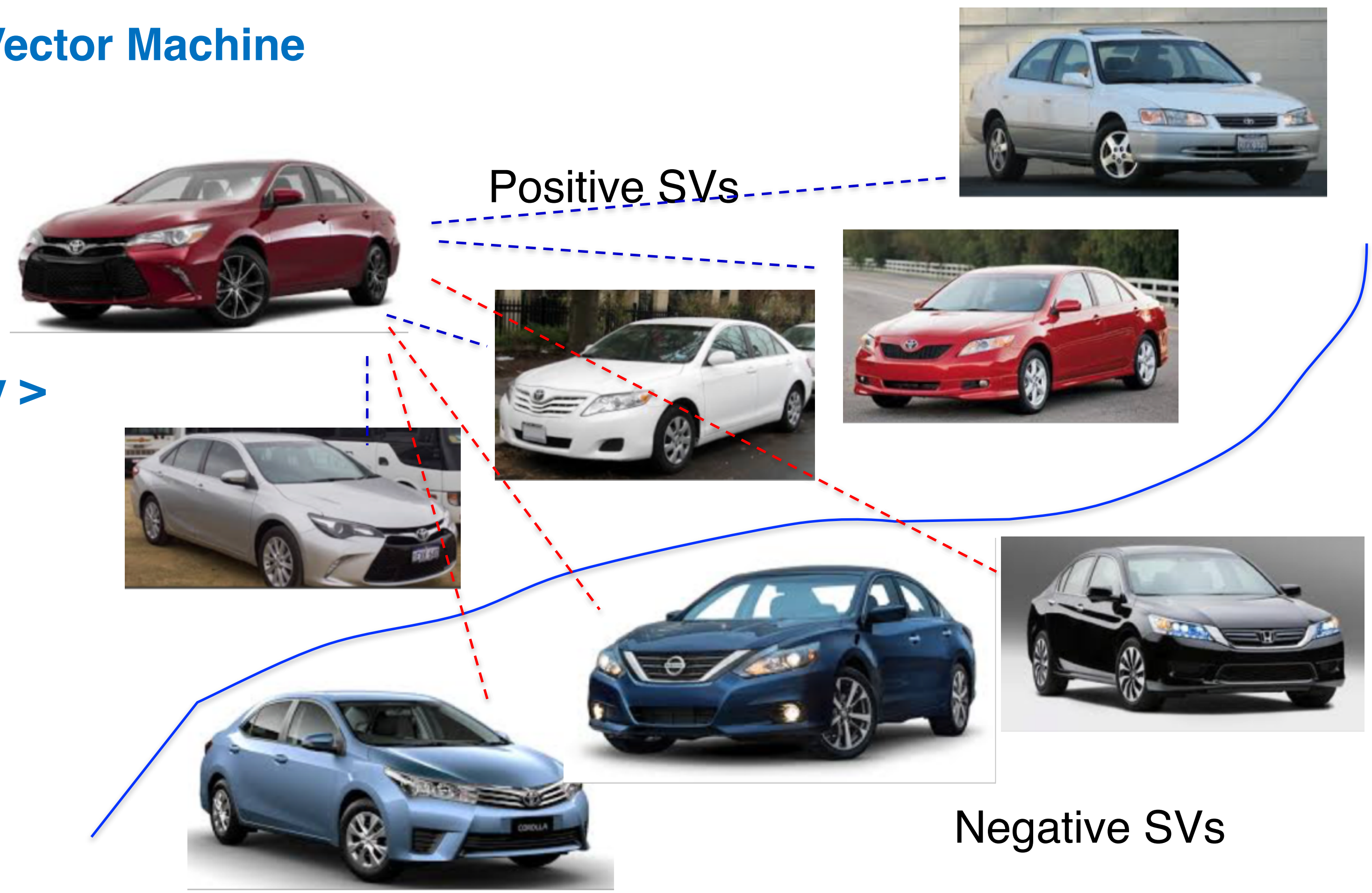
Rule 1.Symbol has something like bull's head
Rule 2.Big black portion in front of car.
Rule 3. …..????

ML — Support Vector Machine



Classification example: using SVM to recognize a Toyota Camry

ML — Support Vector Machine



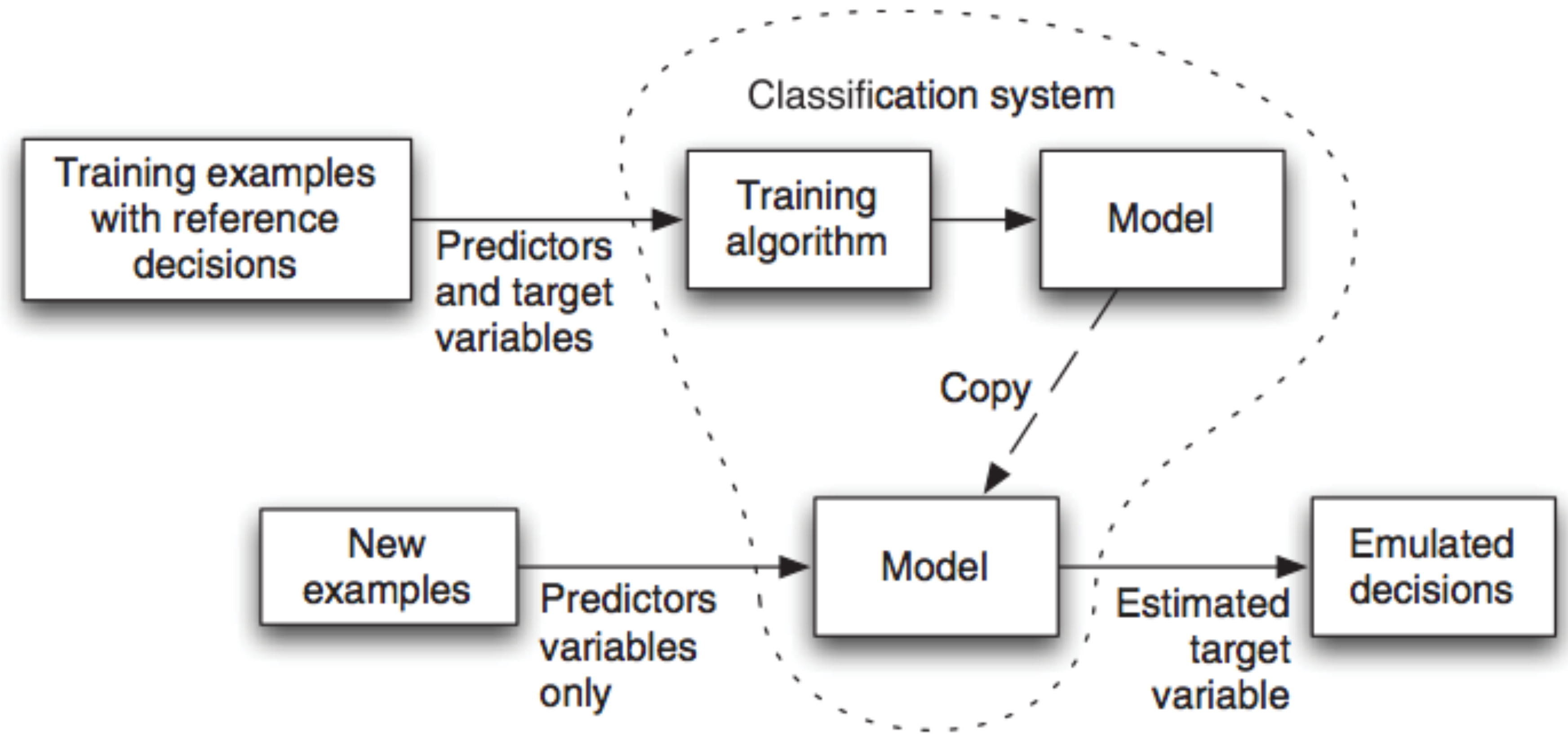
$P_{Camry} >$ 0.95

Feature Space

When to use Big Data System for classification?

System size in number of examples	Choice of classification approach
< 100,000	Traditional, non-Mahout approaches should work very well. Mahout may even be slower for training.
100,000 to 1 million	Mahout begins to be a good choice. The flexible API may make Mahout a preferred choice, even though there is no performance advantage.
1 million to 10 million	Mahout is an excellent choice in this range.
> 10 million	Mahout excels where others fail.

How does a classification system work?



Supervised vs. Unsupervised Learning

Classification algorithms are related to, but still quite different from, clustering algorithms such as the k-means algorithm described in previous chapters. Classification algorithms are a form of supervised learning, as opposed to unsupervised learning, which happens with clustering algorithms. A supervised learning algorithm is one that's given examples that contain the desired value of a target variable. Unsupervised algorithms aren't given the desired answer, but instead must find something plausible on their own.

Supervised and unsupervised learning algorithms can often be usefully combined. A clustering algorithm can be used to create features that can then be used by a learning algorithm, or the output of several classifiers can be used as features by a clustering algorithm. Moreover, clustering systems often build a model that can be used to categorize new data. This clustering system model works much like the model produced by a classification system. The difference lies in what data was used to produce the model. For classification, the training data includes the target variables; for clustering, the training data doesn't include target variables.

Work flow in a typical classification project

Stage	Step
1. Training the model	Define target variable. Collect historical data. Define predictor variables. Select a learning algorithm. Use the learning algorithm to train the model.
2. Evaluating the model	Run test data. Adjust the input (use different predictor variables, different algorithms, or both).
3. Using the model in production	Input new examples to estimate unknown target values. Retrain the model as needed.

Fundamental classification algorithms

Example of fundamental classification algorithms:

- Naive Bayesian
- Complementary Naive Bayesian
- Stochastic Gradient Descent (SDG)
- Random Forest
- Support Vector Machines

Ardi Machine Learning Components

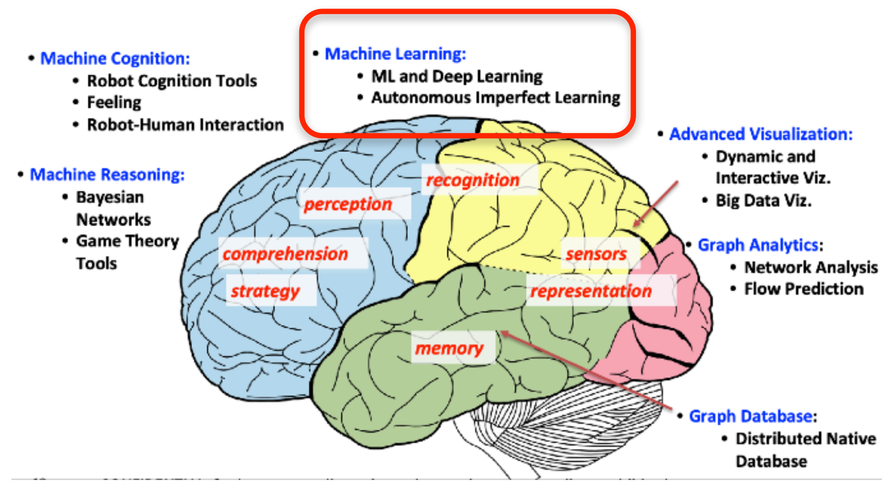
Classification

Support Vector Machine
XGBoost
LightGBM
Random Forest
Decision Tree

Regression

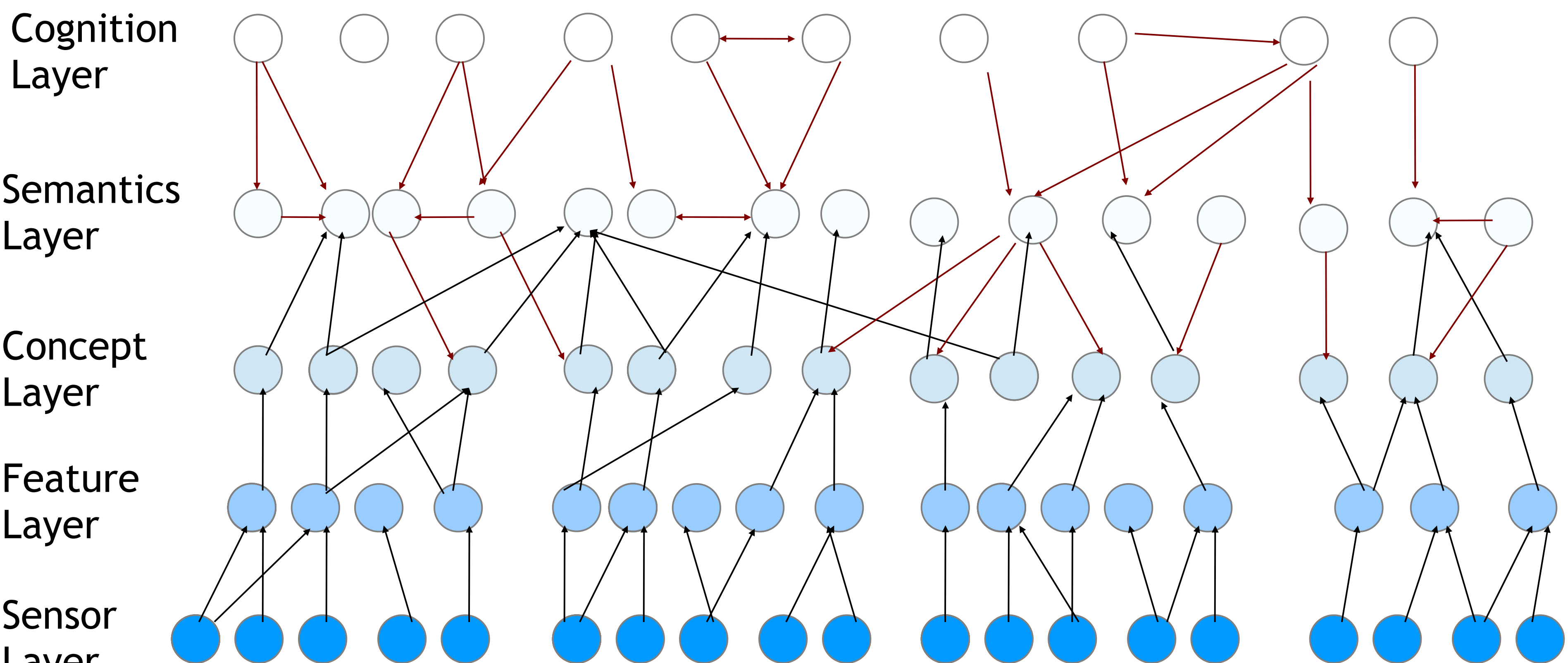
Ordinary Linear Regression
Ridge Regression
Logistic Regression

- **Clustering**
  - K-means
  - Birch
- **Deep Learning**
  - Recurrent Neural Network (RNN)
  - Deep neural network (DNN)
  - Convolutional neural network (CNN)



Example of Multi-Modality Multi-Layer Understanding

- *Structure Learning*
- *Evolutionary Modeling & Prediction*



HR records, Travel records, Badge/Location records, Phone records, Mobile records

Transmitted images, speech content, video content

: observations

: hidden states

Example of Advanced Functions

COLUMBIA UNIVERSITY

Linked Big Data

Most entities are connected. We need a novel AI infrastructure to handle billions connections like our brains.

Machine Reasoning

Human brains differ from animals mainly in the way we can figure out the hidden causes, intentions and motivations.

Advanced Visualization

Visualization is the key to understand analysis -- need to combine data mining, big data analytics, design, rendering, and interaction.

Strategic Thinking

Intelligence is demonstrated on whether we can possibly anticipate other people's behavior and try to find a way a win-win situation.

Autonomous Learning

Continuous Imperfect Learning makes machines to self-learn the environments and keep improving itself, without explicit teachers.

Human Understanding

Understanding people behavior, personality, needs, and values, is key to better serve customers and advise and recommend what to do.

Cognitive Feeling

Machines may evolve into humanlike capabilities such as feeling images, reading body language, and capturing the nuance in speech, etc.

Prediction

We are providing various tools for a better prediction on multi-modality time series that might be affected from various factors.

Spark GraphX

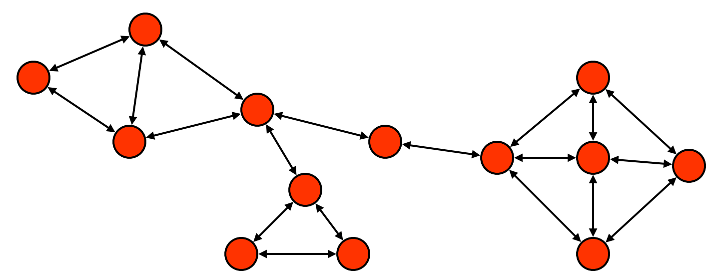
- The Property Graph
    - Example Property Graph
- Graph Operators
    - Summary List of Operators
    - Property Operators
    - Structural Operators
    - Join Operators
    - Neighborhood Aggregation
        - Aggregate Messages (aggregateMessages)
        - Map Reduce Triplets Transition Guide (Legacy)
        - Computing Degree Information
        - Collecting Neighbors
    - Caching and Uncaching
- Pregel API
- Graph Builders
- Vertex and Edge RDDs
    - VertexRDDs
    - EdgeRDDs
- Optimized Representation
- Graph Algorithms
    - PageRank
    - Connected Components
    - Triangle Counting

Graph Analytics

Graph Definitions and Concepts

- A graph:

$$G = (V, E)$$



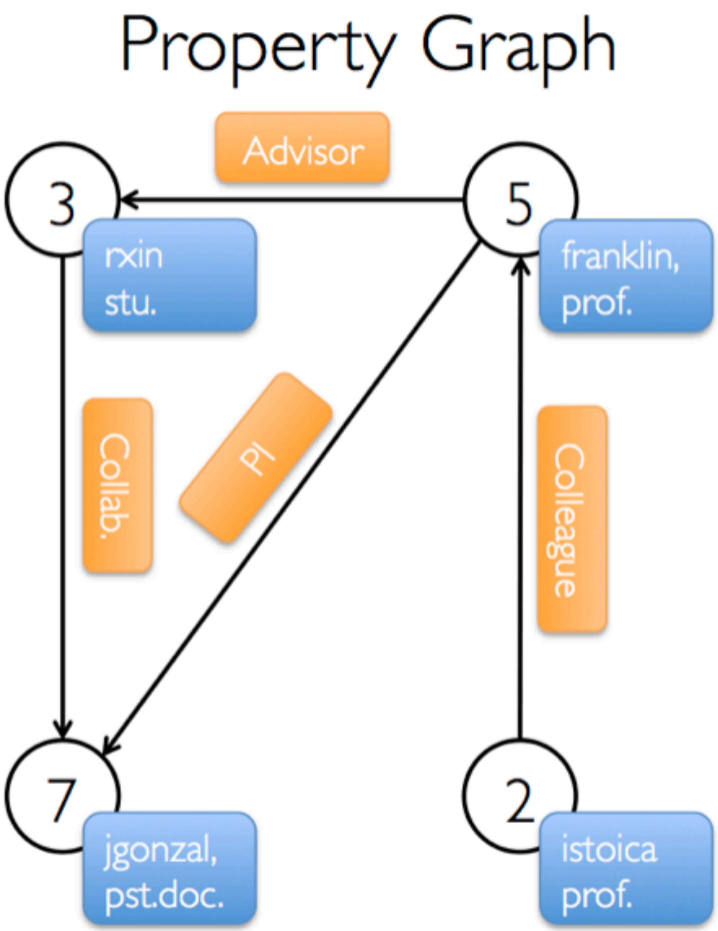
- V = Vertices or Nodes
- E = Edges or Links

- The number of vertices: "Order"

$$N_v = |V|$$

$$N_e = |E|$$

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

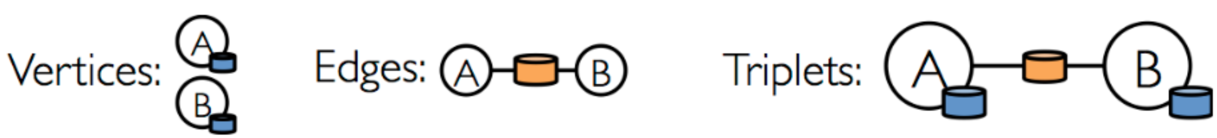
GraphX Graph Operations

```
// Information about the Graph ======
val numEdges: Long
val numVertices: Long
val inDegrees: VertexRDD[Int]
val outDegrees: VertexRDD[Int]
val degrees: VertexRDD[Int]
```

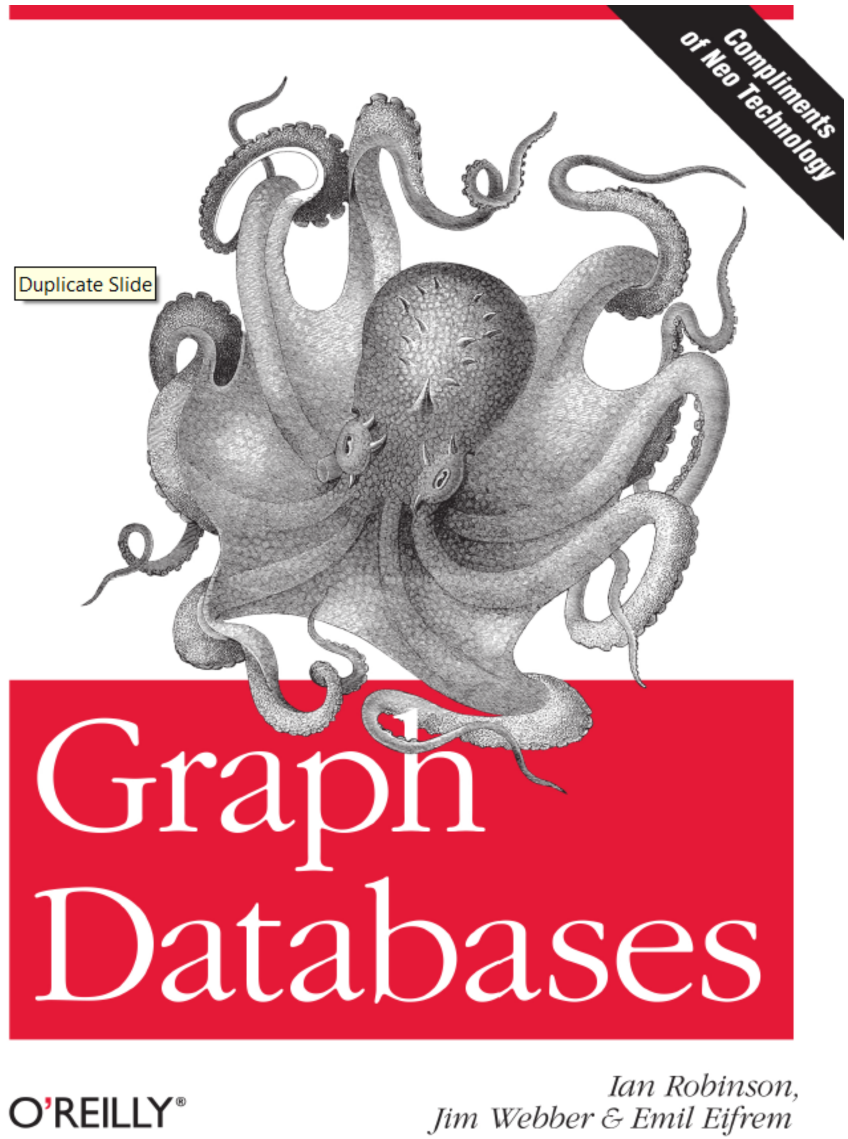
In-degree = 8

Out-degree = 8

```
// Views of the graph as collections ===
val vertices: VertexRDD[VD]
val edges: EdgeRDD[ED]
val triplets: RDD[EdgeTriplet[VD, ED]]
```



Reference



Compliments of Neo Technology
Duplicate Slide
Graph Databases
O'REILLY
Ian Robinson,
Jim Webber & Emil Eifrem

A usual example

User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Figure 2-1. Semantic relationships are hidden in a relational database

Query Example – I

Person

ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend

PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Figure 2-2. Modeling friends and friends-of-friends in a relational database

Asking "who are Bob's friends?" is easy, as shown in Example 2-1.

Example 2-1. Bob's friends

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Query Examples – II & III

Example 2-2. Who is friends with Bob?

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.PersonID = p1.ID
JOIN Person p2
  ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Example 2-3. Alice's friends-of-friends

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Computational intensive

Graph Database Example

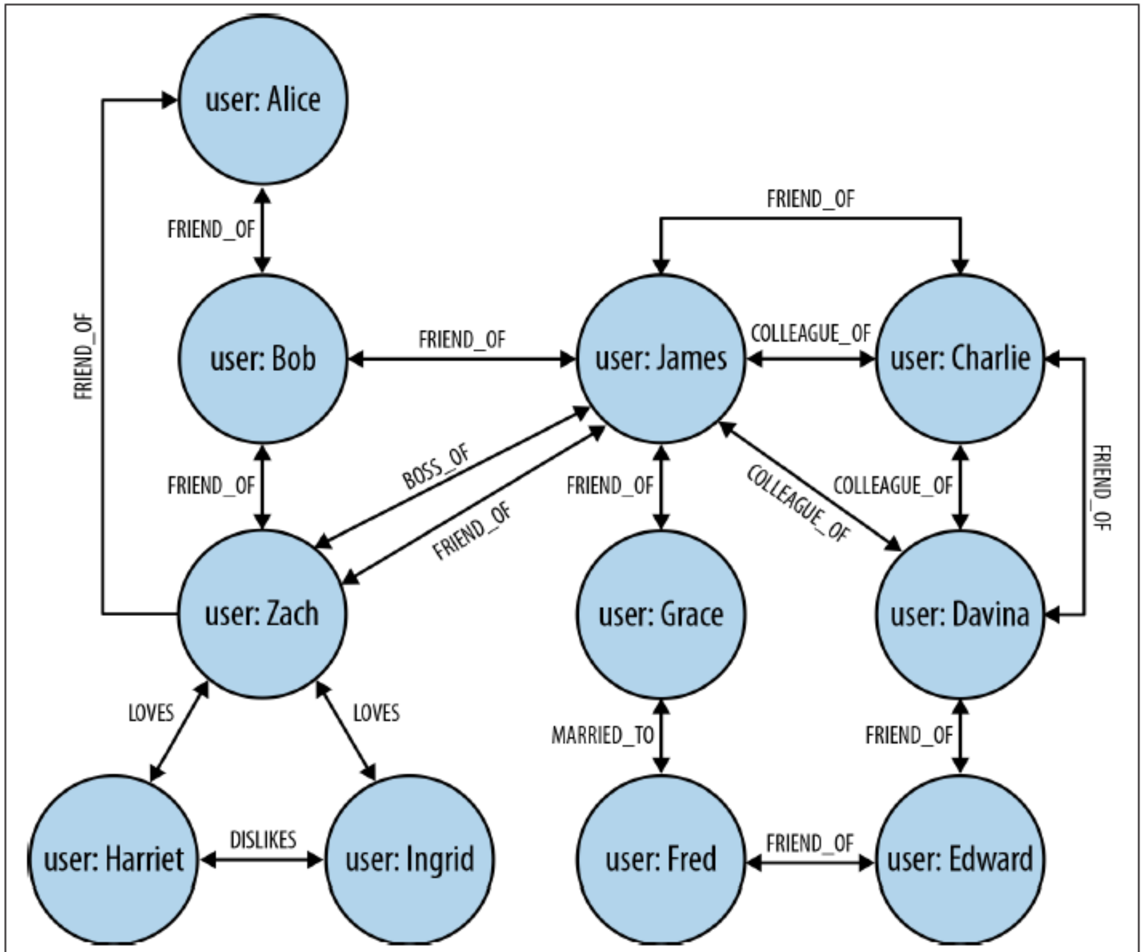


Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

Execution Time in the example of finding extended friends (by Neo4j)

Partner and Vukotic's experiment seeks to find friends-of-friends in a social network, to a maximum depth of five. Given any two persons chosen at random, is there a path that connects them that is at most five relationships long? For a social network containing 1,000,000 people, each with approximately 50 friends, the results strongly suggest that graph databases are the best choice for connected data, as we see in Table 2-1.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Modeling Order History as a Graph

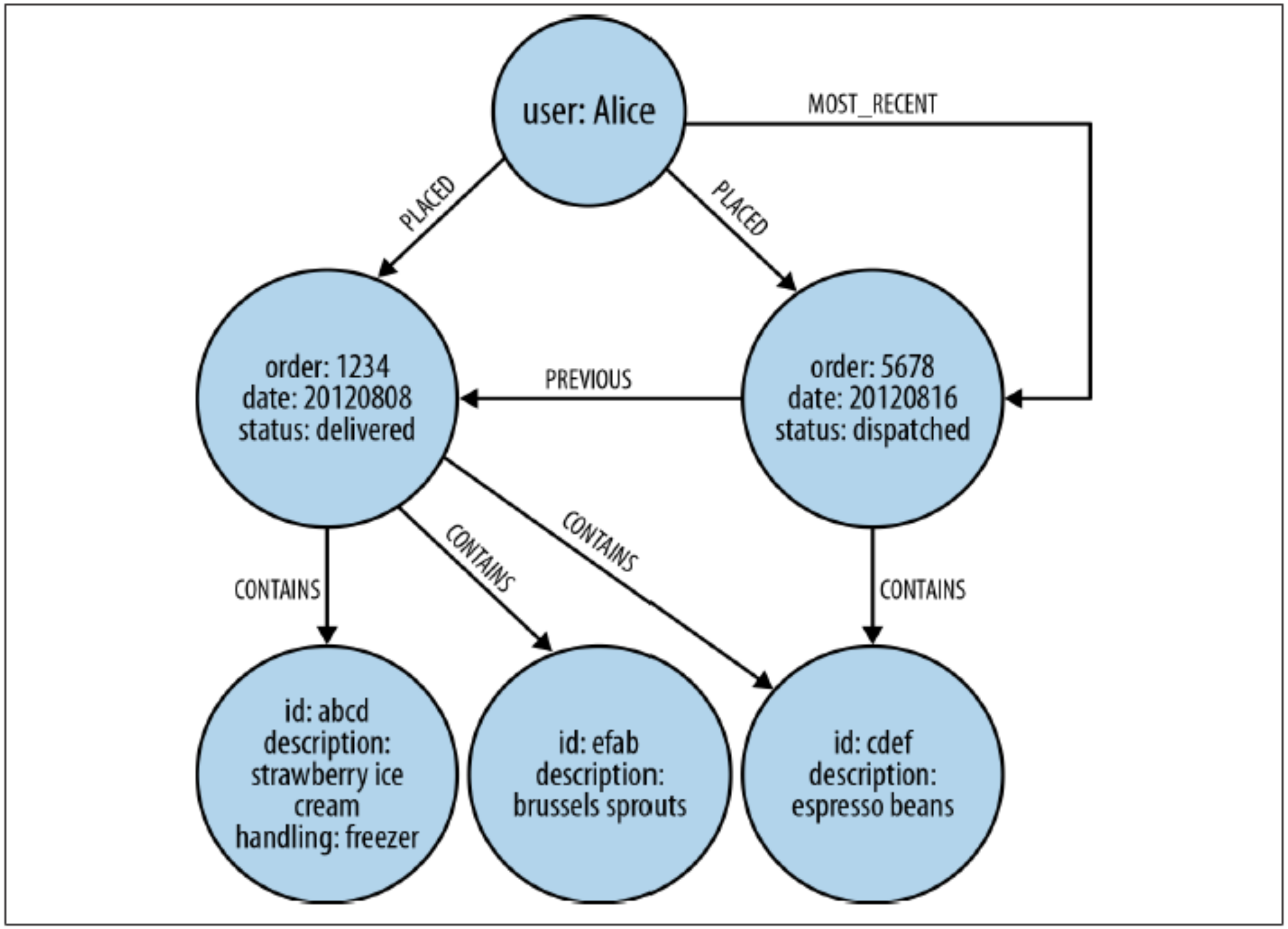


Figure 2-6. Modeling a user's order history in a graph

A query language on Property Graph – Cypher

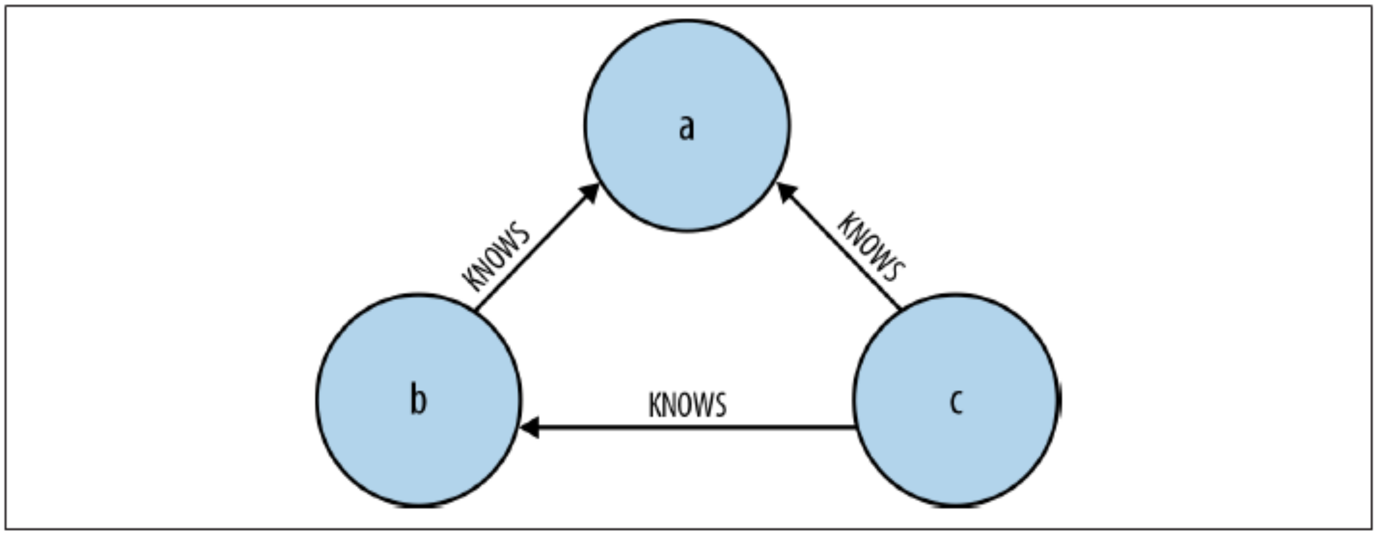


Figure 3-1. A simple graph pattern, expressed using a diagram

This pattern describes three mutual friends. Here's the equivalent ASCII art representation in Cypher:

```
(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
```

Building Application Example – Collaborative Filtering

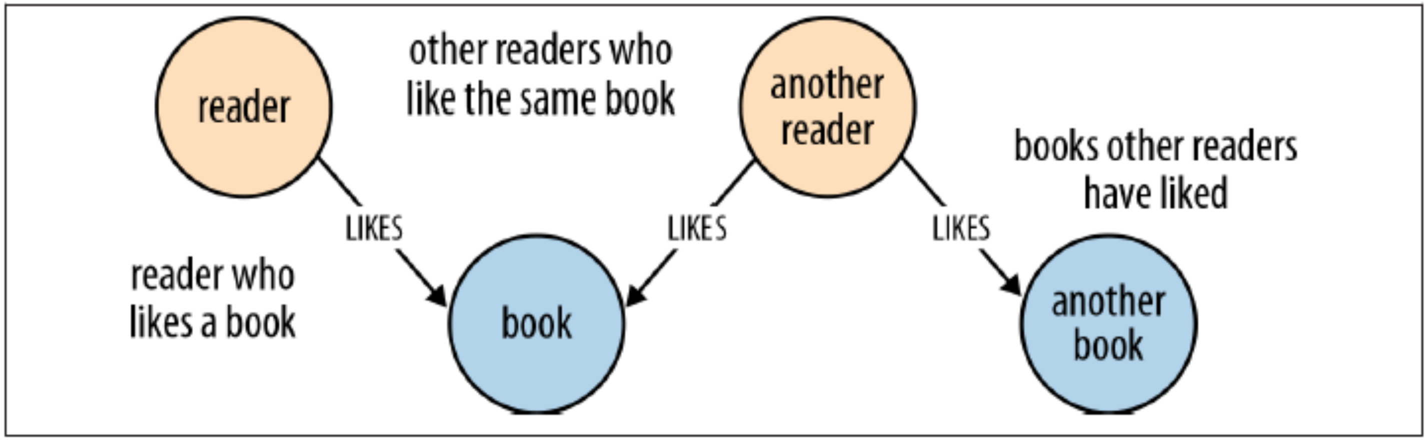


Figure 4-1. Data model for the book reviews user story

Because this data model directly encodes the question presented by the user story, it lends itself to being queried in a way that similarly reflects the structure of the question we want to ask of the data:

```
START reader=node:users(name={readerName})
      book=node:books(isbn={bookISBN})
MATCH reader-[:LIKES]->book<-[:LIKES]-other_readers-[:LIKES]->books
RETURN books.title
```


Example – Email Interaction Graph

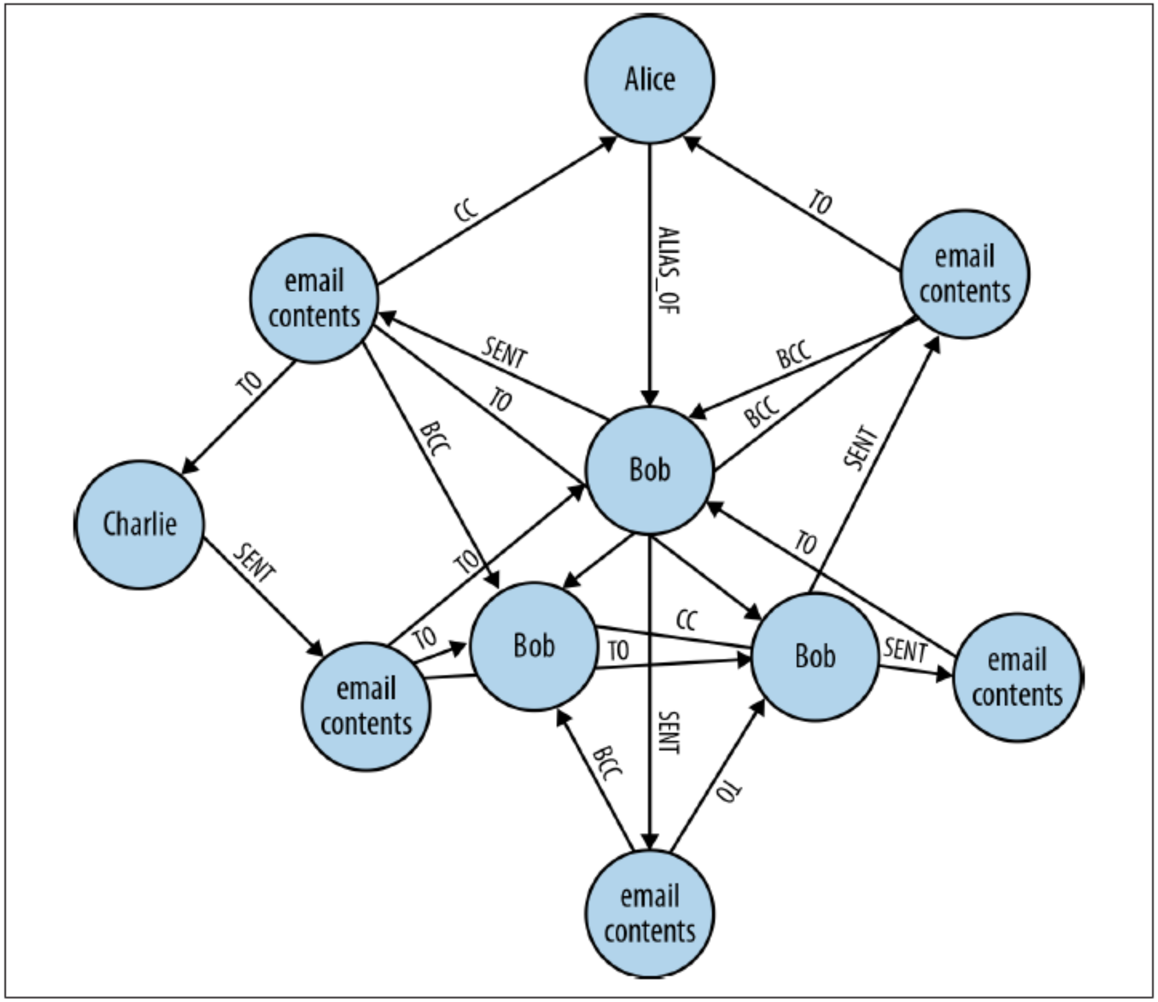


Figure 3-10. A graph of email interactions

```
START bob=node:user(username='Bob')
MATCH (bob)-[:SENT]->(email)-[:CC]->(alias),
      (alias)-[:ALIAS_OF]->(bob)
RETURN email
```

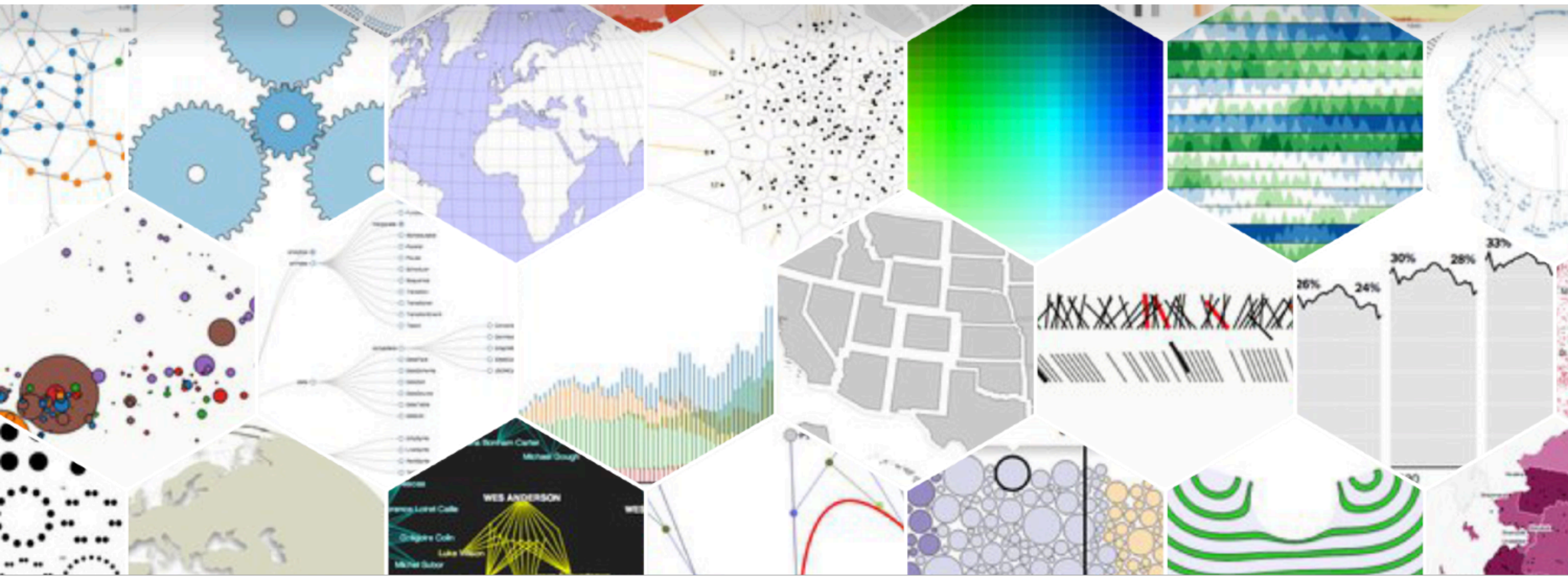
What's this query for?

Data Visualization

D3 Website

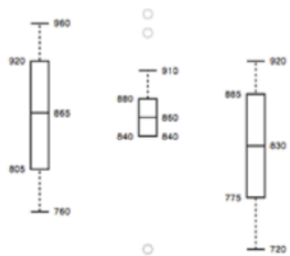
http://d3js.org

D3 Data-Driven Documents



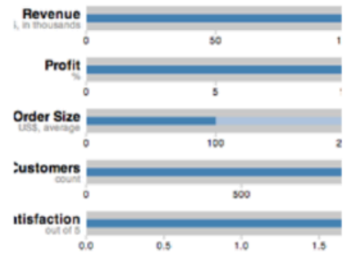
D3 Gallery

Box Plots

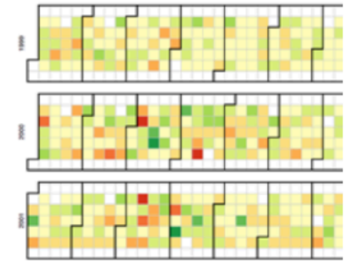


Bubble Chart

Bullet Charts

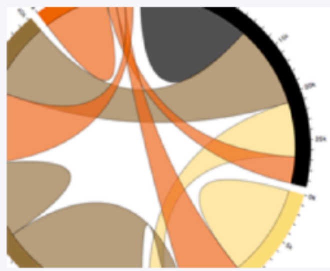


Calendar View

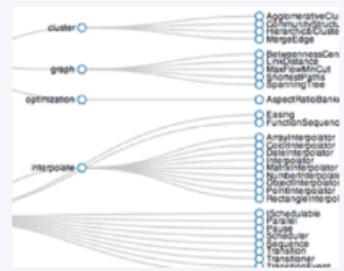


Non-contiguous Cartogram

Chord Diagram

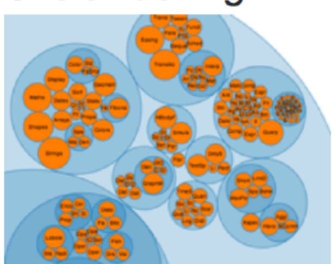


Dendrogram

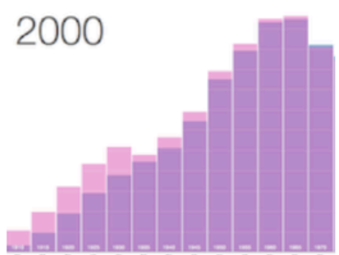


Force-Directed Graph

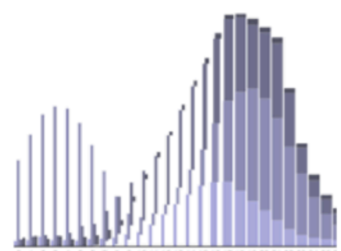
Circle Packing



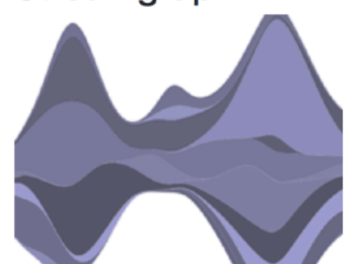
Population Pyramid



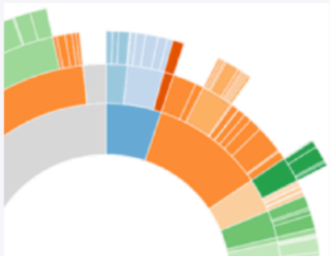
Stacked Bars



Streamgraph

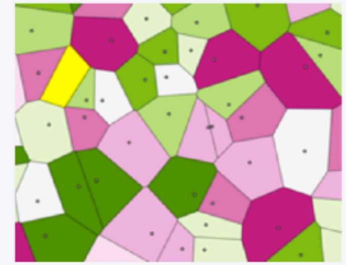


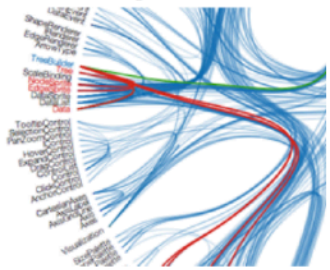
D3 Gallery

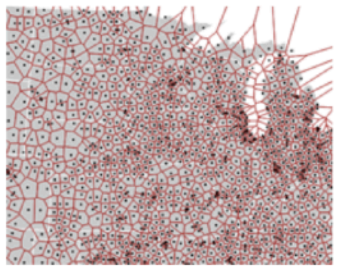
Sunburst


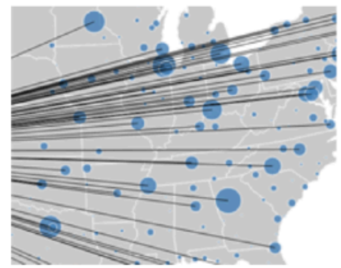
Node-Link Tree

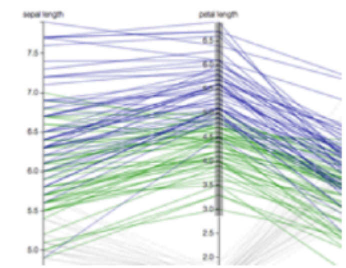

Treemap

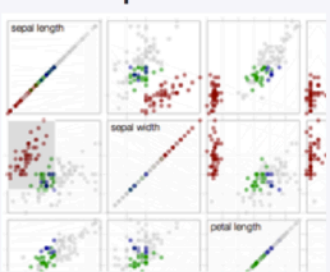

Voronoi Diagram


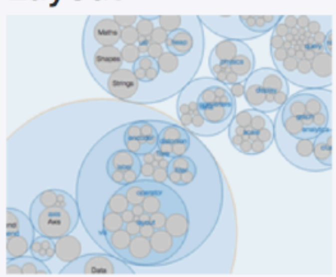
Hierarchical Edge Bundling


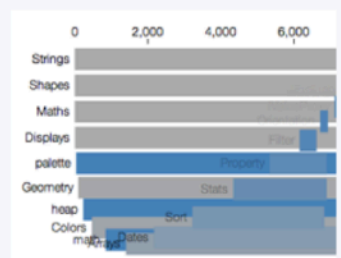
Voronoi Diagram


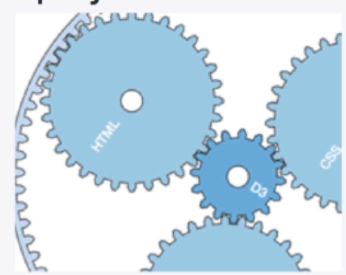
Symbol Map


Parallel Coordinates


Scatterplot Matrix


Zoomable Pack Layout


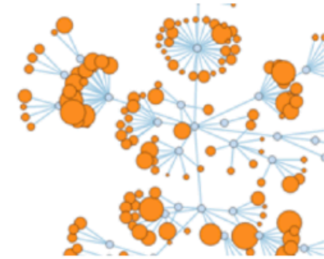
Hierarchical Bars


Epicyclical Gears


D3 Gallery

Collision Detection

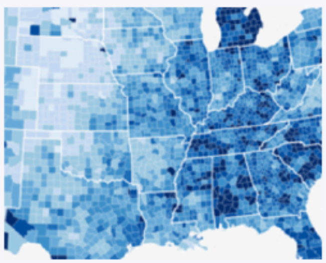
Collapsible Force Layout



Force-Directed States

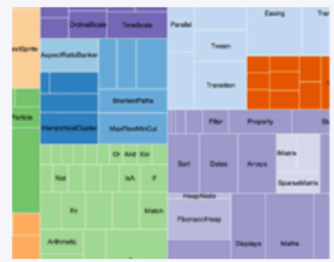
Azimuthal Projections

Choropleth

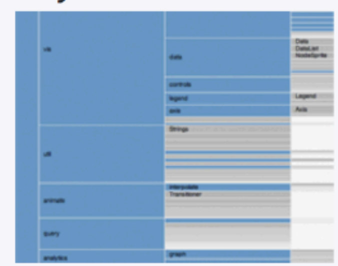


Collapsible Tree Layout

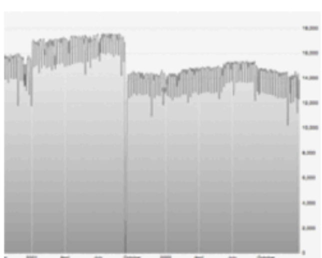
Zoomable Treemap



Zoomable Partition Layout



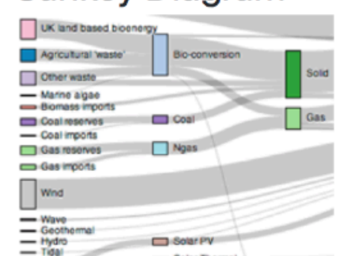
Zoomable Area Chart



Drag and Drop Collapsible Tree Layout

Radial Cluster Layout

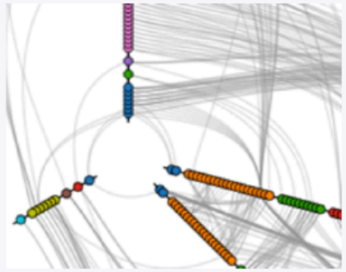
Sankey Diagram



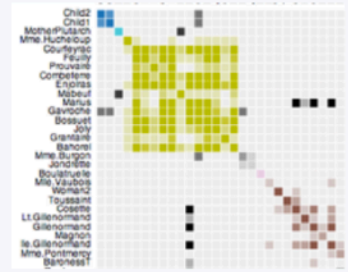
D3 Gallery

Fisheye Distortion

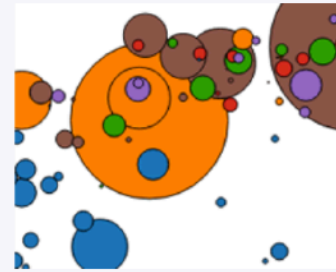
Hive Plot



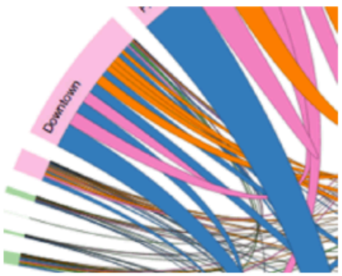
Co-occurrence Matrix



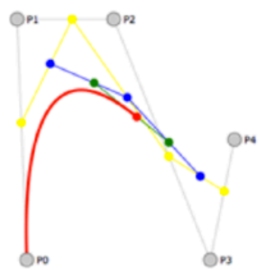
Motion Chart



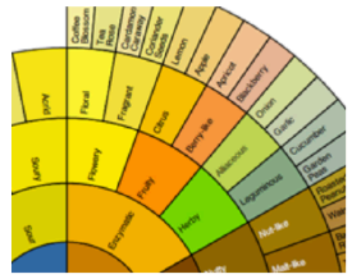
Chord Diagram



Animated Béziers

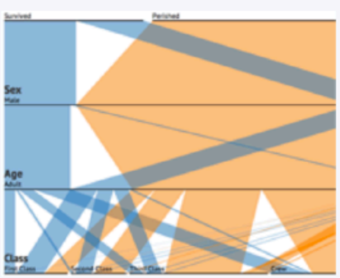


Zoomable Sunburst



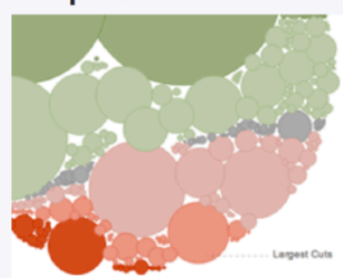
Collatz Graph

Parallel Sets



Word Cloud

Obama's Budget Proposal



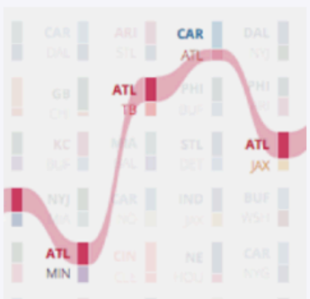
Facebook IPO

D3 Gallery

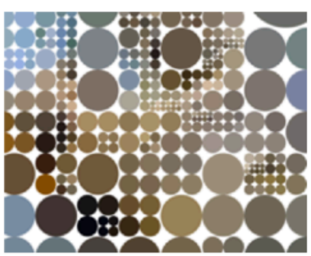
US Trade Deficit

Sequences sunburst

NFL Predictions

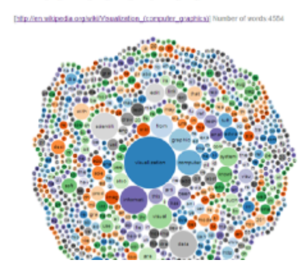


Koalas to the Max

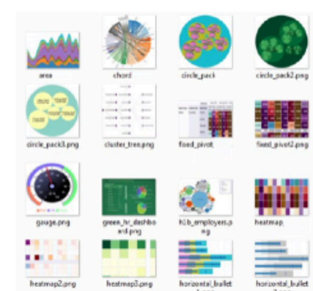


Sankey Creator

Convert any page into bubbles



D3 Builder



Installing

For NPM, `npm install d3`. For Yarn, `yarn add d3`. Otherwise, download the latest release. The released bundle supports AMD, CommonJS, and vanilla environments. Create a custom bundle using Rollup or your preferred bundler. You can also load directly from d3js.org:

```
<script src="https://d3js.org/d3.v5.js"></script>
```

For the minified version:

```
<script src="https://d3js.org/d3.v5.min.js"></script>
```

You can also use the standalone D3 microlibraries. For example, d3-selection:

```
<script src="https://d3js.org/d3-selection.v1.min.js"></script>
```


D3 Selections

In Javascript:

```
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
  var paragraph = paragraphs.item(i);
  paragraph.style.setProperty("color", "white", null);
}
```

D3 employs a declarative approach, operating on arbitrary sets of nodes called *selections*. For example, you can rewrite the above loop as:

```
d3.selectAll("p").style("color", "white");
```

D3 Dynamic Properties

Randomly color paragraphs:

```
d3.selectAll("p").style("color", function() {
  return "hsl(" + Math.random() * 360 + ",100%,50%)";
});
```

Alternate shades of gray for even and odd nodes:

```
d3.selectAll("p").style("color", function(d, i) {
  return i % 2 ? "#fff" : "#eee";
});
```

Randomly color paragraphs:

```
d3.selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
    .style("font-size", function(d) { return d + "px"; });
```

Computed properties often refer to bound data. Data is specified as an array of values, and each value is passed as the first argument (d) to selection functions. With the default join-by-index, the first element in the data array is passed to the first node in the selection, the second element to the second node, and so on. For example, if you bind an array of numbers to paragraph elements, you can use these numbers to compute dynamic font sizes:

D3 Enter

Using D3's *enter* and *exit* selections, you can create new nodes for incoming data and remove outgoing nodes that are no longer needed.

When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection. If there are fewer nodes than data, the extra data elements form the enter selection, which you can instantiate by appending to the `enter` selection. For example:

```
d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
    .text(function(d) { return "I'm number " + d + "!"; });
```

D3 Exit

```
// Update…
var p = d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
    .text(function(d) { return d; });

// Enter…
p.enter().append("p")
    .text(function(d) { return d; });

// Exit…
p.exit().remove();
```

D3 Transformation

For example, to fade the background of the page to black:

```
d3.select("body").transition()
    .style("background-color", "black");
```

Or, to resize circles in a symbol map with a staggered delay:

```
d3.selectAll("circle").transition()
    .duration(750)
    .delay(function(d, i) { return i * 10; })
    .attr("r", function(d) { return Math.sqrt(d * scale); });
```

https://bost.ocks.org/mike/bar/

```
var data = [4, 8, 15, 16, 23, 42];
```

Selecting an Element

Javascript:

```
var div = document.createElement("div");
div.innerHTML = "Hello, world!";
document.body.appendChild(div);
```

D3:

```
var body = d3.select("body");
var div = body.append("div");
div.html("Hello, world!");
```


Chaining Methods

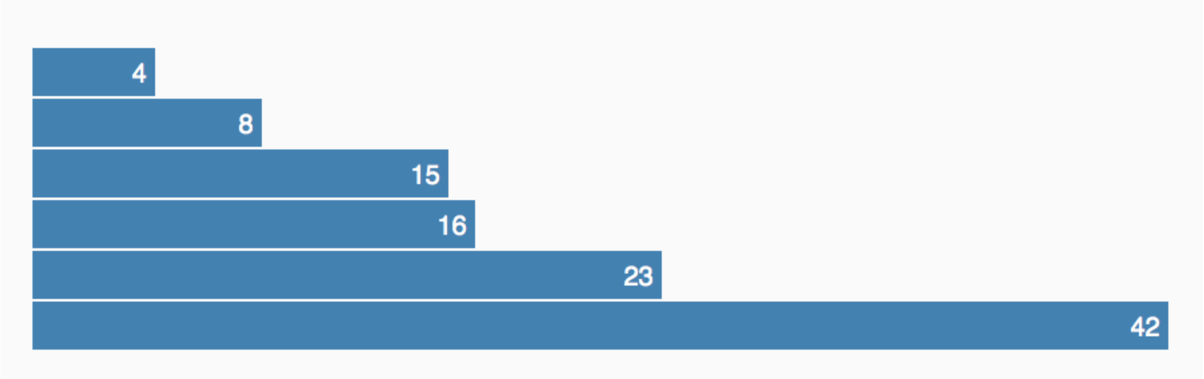
```
var body = d3.select("body");
body.style("color", "black");
body.style("background-color", "white");
```

```
d3.select("body")
    .style("color", "black")
    .style("background-color", "white");
```

```
var section = d3.selectAll("section");

section.append("div")
    .html("First!");

section.append("div")
    .html("Second.");
```

Coding a Chart, Manually

```
<!DOCTYPE html>
<style>

.chart div {
  font: 10px sans-serif;
  background-color: steelblue;
  text-align: right;
  padding: 3px;
  margin: 1px;
  color: white;
}

</style>
<div class="chart">
  <div style="width: 40px;">4</div>
  <div style="width: 80px;">8</div>
  <div style="width: 150px;">15</div>
  <div style="width: 160px;">16</div>
  <div style="width: 230px;">23</div>
  <div style="width: 420px;">42</div>
</div>
```



Coding a Chart, Automatically

```
d3.select(".chart")
  .selectAll("div")
    .data(data)
  .enter().append("div")
    .style("width", function(d) { return d * 10 + "px"; })
    .text(function(d) { return d; });
```

First, we select the chart container using a class selector.

```
var chart = d3.select(".chart");
```

Next we initiate the data join by defining the selection to which we will join data.

```
var bar = chart.selectAll("div");
```

Coding a Chart, Automatically

```
var barUpdate = bar.data(data);
```

```
var barEnter = barUpdate.enter().append("div");
```

```
barEnter.style("width", function(d) { return d * 10 + "px"; });
```

```
barEnter.text(function(d) { return d; });
```

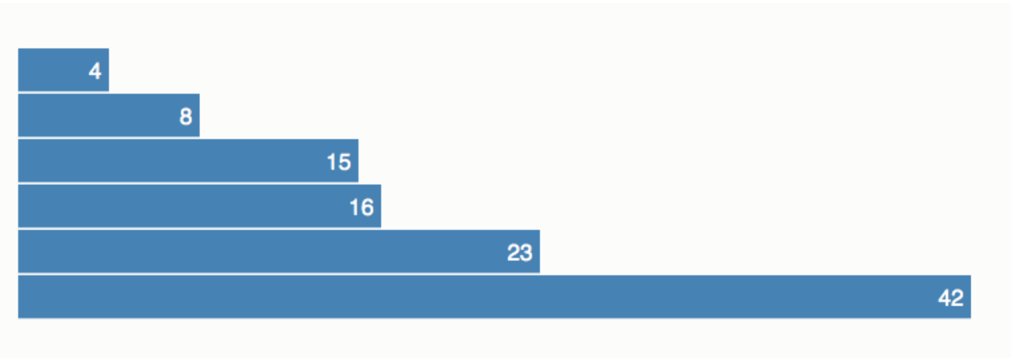
Scaling to Fit

```
var x = d3.scale.linear()
    .domain([0, d3.max(data)])
    .range([0, 420]);
```

D3 Bar Chart Tutorial

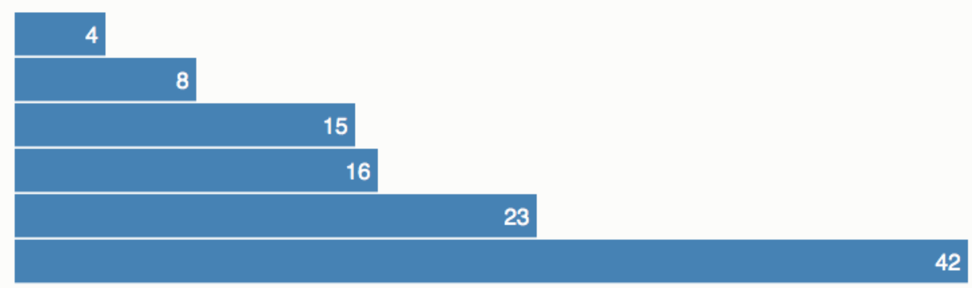
```
<!DOCTYPE html>
<style>

.chart rect {
  fill: steelblue;
}

.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}

</style>
<svg class="chart" width="420" height="120">
  <g transform="translate(0,0)">
    <rect width="40" height="19"></rect>
    <text x="37" y="9.5" dy=".35em">4</text>
  </g>
  <g transform="translate(0,20)">
    <rect width="80" height="19"></rect>
    <text x="77" y="9.5" dy=".35em">8</text>
  </g>
  <g transform="translate(0,40)">
    <rect width="150" height="19"></rect>
    <text x="147" y="9.5" dy=".35em">15</text>
  </g>
  <g transform="translate(0,60)">
    <rect width="160" height="19"></rect>
    <text x="157" y="9.5" dy=".35em">16</text>
  </g>
  <g transform="translate(0,80)">
    <rect width="230" height="19"></rect>
    <text x="227" y="9.5" dy=".35em">23</text>
  </g>
  <g transform="translate(0,100)">
    <rect width="420" height="19"></rect>
    <text x="417" y="9.5" dy=".35em">42</text>
  </g>
</svg>
```

Full code to do it manually



Full code to do it automatically

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

.chart rect {
  fill: steelblue;
}

.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}

</style>
<svg class="chart"></svg>
<script src="//d3js.org/d3.v3.min.js" charset="utf-8"></script>
```



Full code to do it automatically

```
<script>

var data = [4, 8, 15, 16, 23, 42];

var width = 420,
    barHeight = 20;

var x = d3.scale.linear()
    .domain([0, d3.max(data)])
    .range([0, width]);

var chart = d3.select(".chart")
    .attr("width", width)
    .attr("height", barHeight * data.length);

var bar = chart.selectAll("g")
    .data(data)
  .enter().append("g")
    .attr("transform", function(d, i) { return "translate(0," + i * barHeight + ")"; });

bar.append("rect")
    .attr("width", x)
    .attr("height", barHeight - 1);

bar.append("text")
    .attr("x", function(d) { return x(d) - 3; })
    .attr("y", barHeight / 2)
    .attr("dy", ".35em")
    .text(function(d) { return d; });

</script>
```

Load data

```
// 1. Code here runs first, before the download starts.

d3.tsv("data.tsv", function(error, data) {
  // 3. Code here runs last, after the download finishes.
});

// 2. Code here runs second, while the file is downloading.
```

```
name          value
Locke         4
Reyes         8
Ford          15
Jarrah        16
Shephard      23
Kwon          42
```

The equivalent of Javascript code:

```
var data = [
  {name: "Locke",    value:  4},
  {name: "Reyes",    value:  8},
  {name: "Ford",     value: 15},
  {name: "Jarrah",   value: 16},
  {name: "Shephard", value: 23},
  {name: "Kwon",     value: 42}
];
```

D3 Bar Chart Tutorial

```
<!DOCTYPE html>
<meta charset="utf-8">
<style>

.chart rect {
  fill: steelblue;
}

.chart text {
  fill: white;
  font: 10px sans-serif;
  text-anchor: end;
}

</style>
<svg class="chart"></svg>
<script src="//d3js.org/d3.v3.min.js" charset="utf-8"></script>
```

D3 Bar Chart Tutorial

```
<script>

var width = 420,
    barHeight = 20;

var x = d3.scale.linear()
    .range([0, width]);

var chart = d3.select(".chart")
    .attr("width", width);

d3.tsv("data.tsv", type, function(error, data) {
  x.domain([0, d3.max(data, function(d) { return d.value; })]);

  chart.attr("height", barHeight * data.length);

  var bar = chart.selectAll("g")
      .data(data)
    .enter().append("g")
      .attr("transform", function(d, i) { return "translate(0," + i * barHeight + ")"; });

  bar.append("rect")
      .attr("width", function(d) { return x(d.value); })
      .attr("height", barHeight - 1);

  bar.append("text")
      .attr("x", function(d) { return x(d.value) - 3; })
      .attr("y", barHeight / 2)
      .attr("dy", ".35em")
      .text(function(d) { return d.value; });
});

function type(d) {
  d.value = +d.value; // coerce to number
  return d;
}

</script>
```