Machine Learning Foundations
Data Structures & Algorithms

Computer Science
Techniques for Developing the
Most Efficient Data Models

Jon Krohn, Ph.D.

jonkrohn.com/talks

github.com/jonkrohn/ML-foundations

Machine Learning Foundations
Data Structures & Algorithms

Slides: jonkrohn.com/talks

Code: github.com/jonkrohn/ML-foundations

Stay in Touch:

jonkrohn.com to sign up for email newsletter

linkedin.com/in/jonkrohn

jonkrohn.com/youtube

twitter.com/JonKrohnLearns

The Pomodoro Technique

Rounds of:

- 25 minutes of work
- with 5 minute breaks

Questions best handled at breaks, so save questions until then.

When people ask questions that have already been answered, do me a favor and let them know, politely providing response if appropriate.

Except during breaks, I recommend attending to this lecture only as topics are not discrete: Later material builds on earlier material.

POLL

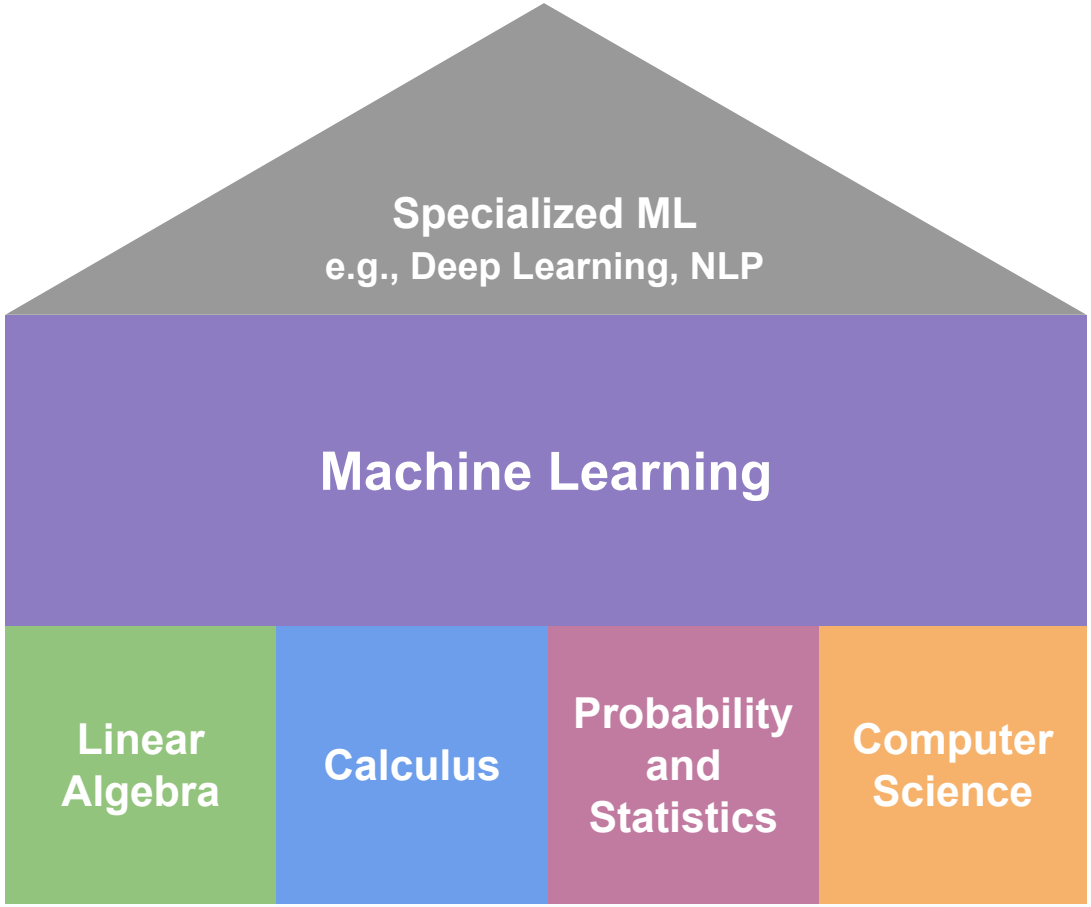
What is your level of familiarity with data structures and algorithms?

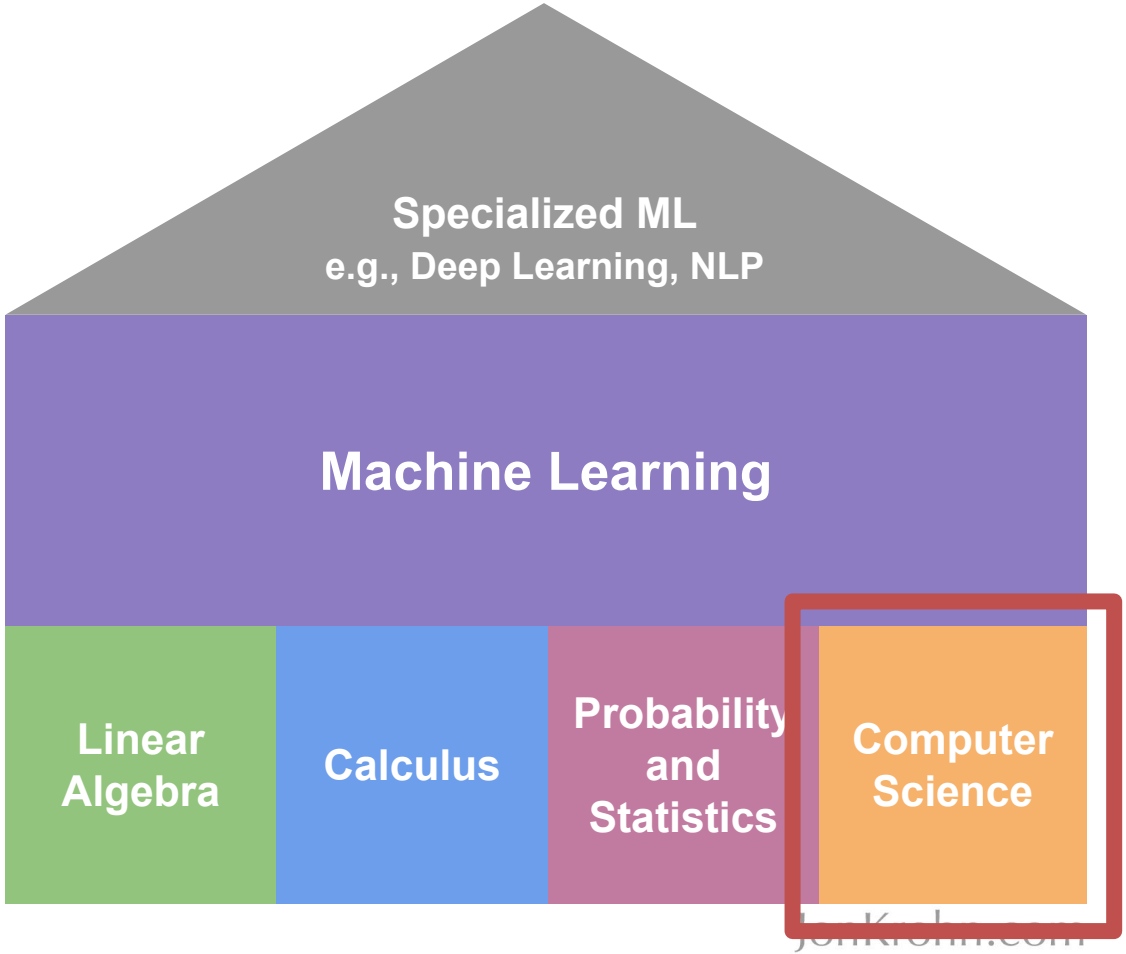
- Little to no formal exposure
- Some understanding of the theory
- Deep understanding of the theory

POLL

What is your level of familiarity with Machine Learning?

- Little to no exposure, or exposure to theory only
- Experience applying machine learning with code
- Experience applying machine learning with code and some understanding of the underlying theory
- Experience applying machine learning with code and strong understanding of the underlying theory


Specialized ML
e.g., Deep Learning, NLP
Machine Learning
Linear Algebra
Calculus
Probability and Statistics
Computer Science
JonKrohn.com


Specialized ML
e.g., Deep Learning, NLP
Machine Learning
Linear Algebra
Calculus
Probability and Statistics
Computer Science

ML Foundations Series

Algorithms & Data Structures **builds upon** and is **foundational for**:

1. Intro to Linear Algebra *(arrays/tensors)*
2. Linear Algebra II: Matrix Operations
3. Calculus I: Limits & Derivatives *(graphs)*
4. Calculus II: Partial Derivatives & Integrals *(ROC AUC)*
5. Probability & Information Theory *(random sampling)*
6. Intro to Statistics
7. **Algorithms & Data Structures**
8. Optimization

Data Structures & Algorithms

1. Intro to Data Structures and Algorithms
2. Lists and Dictionaries
3. Trees and Graphs

Data Structures & Algorithms

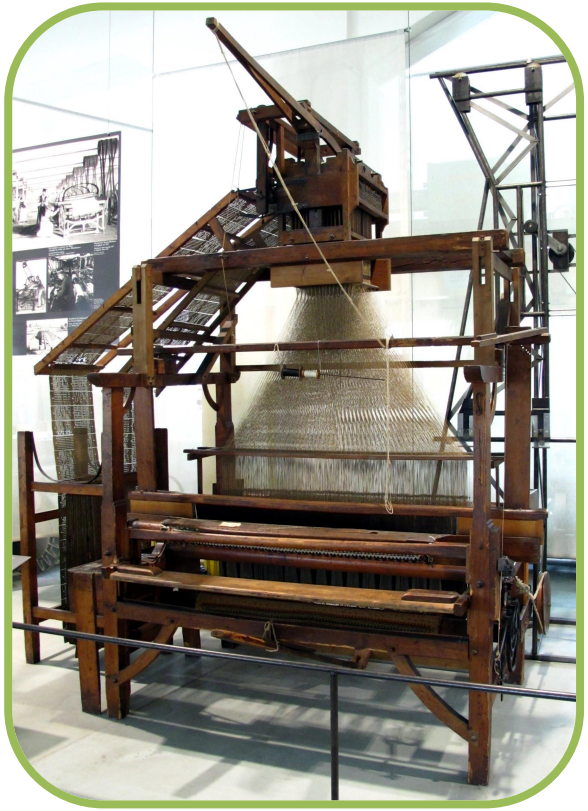
1. **Intro to Data Structures and Algorithms**
2. Lists and Dictionaries
3. Trees and Graphs

Segment 1: Intro to DSA

- A Brief History of Data
- A Brief History of Algorithms
- Applications of DSA to ML
- “Big O” Notation for Time and Space Complexity

Brief History of Data

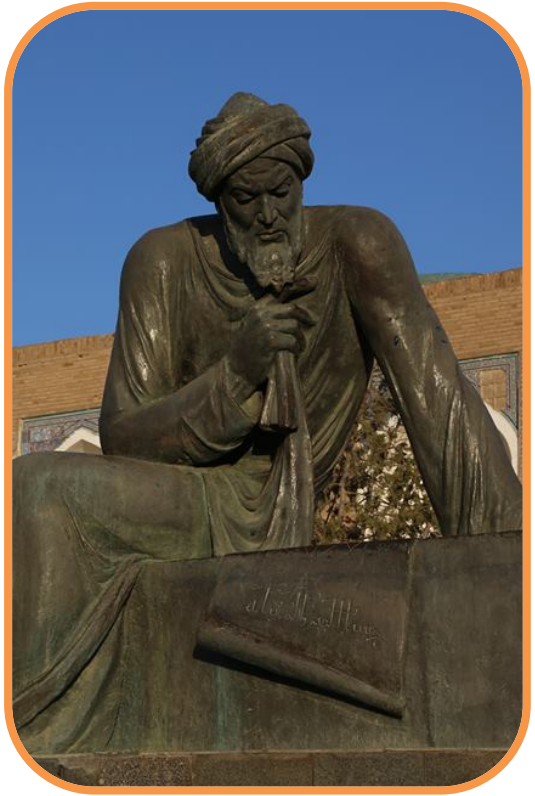
- **Data** are units of information
  - 18th c. from Latin *datum*, “something given”
- 4b-year-old **RNA** (Earth is 4.5b) → DNA
- 7500 BCE **Mesopotamia**: clay accounting tokens
- 3400-3100 BCE **Sumeria**: written clay records
- Punch cards / punched tape
  - Since 1725: **used in mechanical looms**
  - 1950s to 1970s: common in computing
- 1946: “Data” term in computing
- 1954: “Data processing” term
- Today: hard disk drives, optical discs, mag. tape
- **Data structures**: lists, dictionaries, objects, etc.



commons.wikimedia.org/wiki/File:DMM_29263ab_Jacquardwebstuhl.jpg

Brief History of Algorithms

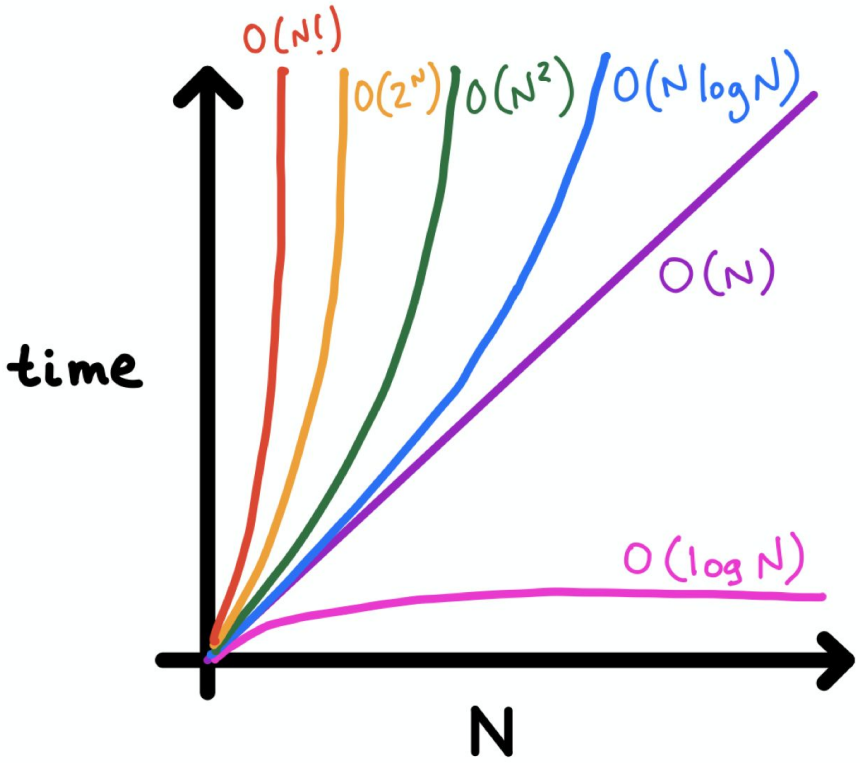
- Unambiguous instructions for solving problem or performing computation
- From initial state (e.g., input) go thru finite well-defined states ‘til final state (e.g., with output)
- Can incorporate random probabilities so not necessarily deterministic, e.g., SGD, MCMC
- Arithmetic algos (e.g., for division) are ancient:
  - Babylonians (2500 BCE), Egyptians (1500 BCE)
  - Greeks’ sieve of Eratosthenes, Euclidean algo
- 9th c. CE: Arab cryptographic algos
  - Word from **Muḥammad ibn Mūsā al-Khwārizmī**
- 1930s: Modern concept formalized
  - Gödel, Turing, many others



Applications of DSA to ML

- Ensure you're using the correct data structure for the situation
- Be thoughtful about time/space complexity considerations
  - Model training
  - Model deployment in production
  - Software 2.0: ML/DL algos typically have constant time/space complexity, assuming fixed batch size
- Conceptualize and implement ML models as graphs
  - TensorFlow
  - PyTorch
  - "Knowledge graphs"

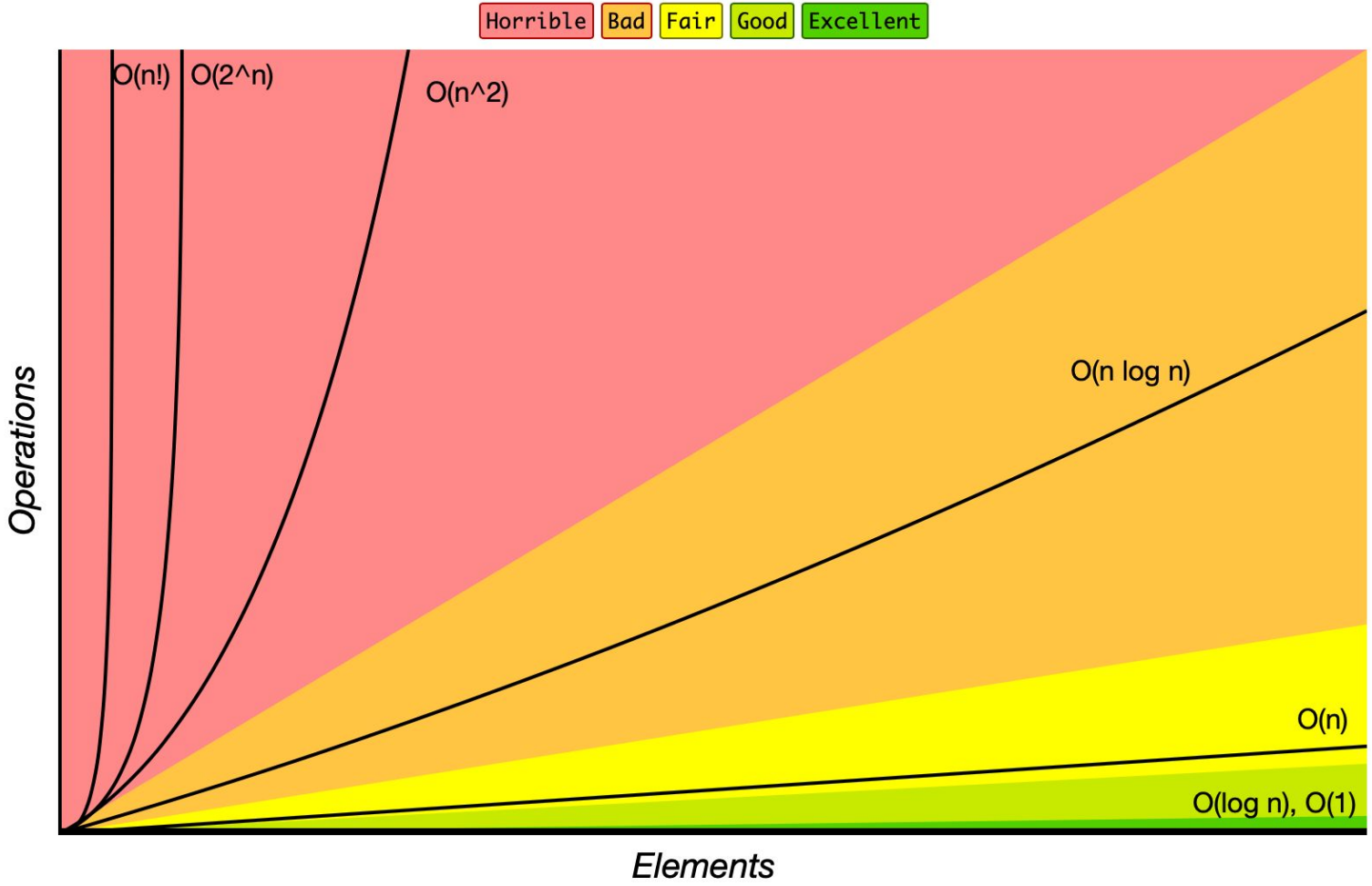
“Big O” (Order of Complexity) Notation



Hands-on code demo: `7-algos-and-data-structures.ipynb`

Common Runtimes

Big O	Nickname	Example(s)
O(1)	constant	first item in list; is list length odd or even
O(n)	linear	search linearly through list; print every item in list
O(n^2)	polynomial	loop nested within loop; bubble sort
O(log n)	logarithmic	binary search
O(2^n)	exponential	count combinations of list elements
O(n log n)		mergesort; quick sort
O(n!)	factorial	generate all permutations of a list; simple Traveling Salesman solution



bigocheatsheet.com

Best vs Worst Case

- Variance in Big O possible based on data
- E.g., for quick sort:
  - Best case is $O(n)$ if all items are equal
  - Worst case is $O(n^2)$ if "pivot" is always largest item
  - Expected case (a.k.a. average case) is $O(n \log n)$

Further Big O Detail

- Given n, can then select appropriate algo based on Big O
- Can equally be applied to space complexity
- Drop constants: $O(2n) \rightarrow O(n)$
- Retain only dominant term: $O(n^2 + n) \rightarrow O(n^2)$
- May involve multiple variables, e.g.: $O(n^2 + m)$, $O(nm)$

Exercise: Estimate the Google Colab runtime of the `element_multiplier()` method with a list of length of a million.

Data Structures & Algorithms

1. Intro to Data Structures and Algorithms
2. **Lists and Dictionaries**
3. Trees and Graphs

Segment 2: Lists and Dictionaries

- List-Based Data Structures
- Searching and Sorting
- Set-Based Data Structures
- Hashing

Lists

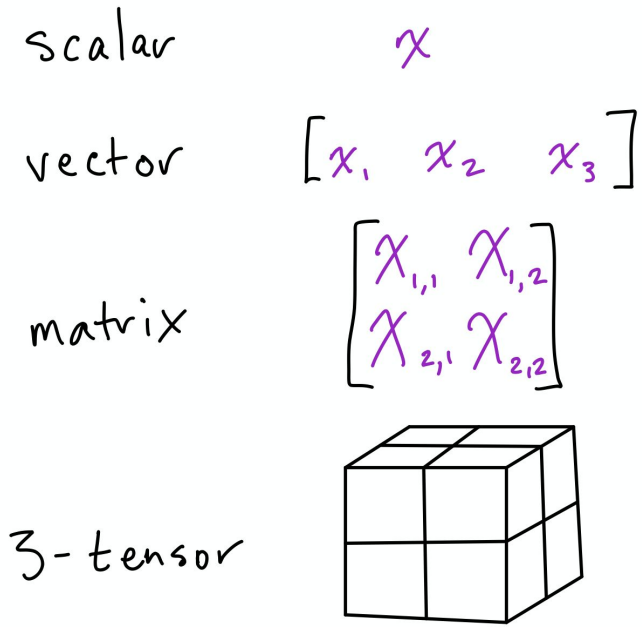
- An *ordered* **collection** of items
- Can specify item by its index
- List-based data structures:
  - Arrays
  - Linked lists
  - Stacks
  - Queues
  - Deques

Hands-on code demo

unsplash.com/photos/RLw-UC03Gwc

Arrays

- **Lists** we can apply math operations on
- Used extensively in *ML Foundations*
- Most important data structure in ML; commonly used for:
  - Tensors
  - Holding data for training models (e.g., input from HDF5, CSV)
  - Preprocessing data
  - Receiving model outputs
  - Calculating summary metrics
  - Plotting
- Typically, all elements are same type
  - Most often numeric in ML



Hands-on code demo

Linked Lists

- Similar to (vector) array, but elements aren't indexed
  - Instead, elements in linked lists (called *nodes* or *units*) store which element in list is next (nodes are "linked")
- Makes adding or removing elements anywhere in the list computationally efficient (O(1))
  - Note, however, that accessing elements is O(n) in worst case
- Not a built-in Python data structure
- Not especially common in ML; could be used for parsing lists with undefined length then converted to array before input into ML model

unsplash.com/photos/PDxYfXVlK2M

Doubly-Linked Lists

- Nodes store which is next *and which precedes it*
- Enables traversal in either direction
- Same time complexities as singly-linked

unsplash.com/photos/PDxYfXVlK2M

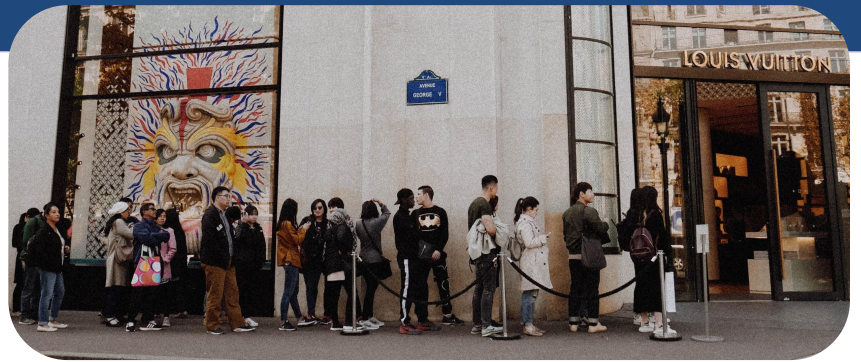
Stacks

- Behave like real-world stacks
  - I.e., LIFO: "*Last in, first out*"
- E.g., imagine stacking a deck of cards:
  - Put (*push*) one card on table
  - Stack (*push*) another on top, then push a third
  - To access cards, top card is retrieved (*pop*-ped) first
  - Last to pop is card placed on table first
- Push and pop are both O(1)
- Useful anytime most recent items will be needed before earlier ones
  - E.g., social media newsfeed
- Rare to use directly in ML
  - Could be used under hood to implement programming languages

Hands-on code demo

unsplash.com/photos/IEISYENbXp8

Queues



- Behave like real-world queues (“lines” in U.S./Canada)
  - I.e., FIFO: “*First in, first out*”
- **Head**: first/oldest element
- **Tail**: last/newest
- *Enqueue*: add element to tail
- *Dequeue*: remove element from head
- *Peek*: access head element but leave it in queue

unsplash.com/photos/SsBI9pweAeA

Deques

- *Double-ended queue*
- Can enqueue or dequeue *from either end*
- In other words, a generalized:
  - stack (enqueue and dequeue from same end)
  - *or* queue (enqueue and dequeue from opposing ends)

Hands-on code demo:

jonkrohn.com/DQN

unsplash.com/photos/sLtmudkaL9M

Searching & Sorting

- Will not typically write search/sort algos in ML
- Exemplary to demo:
  - Time complexity
  - Space complexity
  - Typical time/space tradeoff
- Helpful for any ML pipelines, incl.:
  - Data capture
  - Preprocessing
  - Training
  - Deployment

Hands-on code demo: `7-algos-and-data-structures.ipynb`

unsplash.com/photos/18dsFU_VpZI

Maps

- Have *key-value structure*
  - **Key** (name) indicates where **value** is
  - ...like real-world map shows where place is
- A.k.a. **dictionary** (e.g., in Python)
  - ...like how name shows where definition is
- Used frequently in NLP: integer index of words

Hands-on code demo:

jonkrohn.com/film-classifier

unsplash.com/photos/1UDjq8s8cy0 unsplash.com/photos/TrhLCn1abMU

Sets

- *Unordered* collection of items
  - Akin to having a bag of stuff
- No elements can be duplicated
- **Maps** are a **set**-type data structure
  - The keys are a formal set
    - *No duplicate keys*

Hands-on code demo: `7-algos-and-data-structures.ipynb`

unsplash.com/photos/oC66vXsqnc8

Hashing

- **Hash functions** enable set searching in constant time, O(1) !!!
- This contrasts starkly with:
  - List-based structures (or unhashed sets), where search is linear time O(*n*)
  - Or, at best, O(log *n*) with binary search of pre-sorted list
- *Well, that's crazy pants… how does it work?*

unsplash.com/photos/a31khKMmRd0

Hashing

- Ultimate trick is that *indexed* data structures (e.g., with *sequential* integers) have O(1) time search/retrieval
- So, what we need is a way to convert dictionary values into an index…
- *This* is what hash functions do:
  - Input some value
  - Output a *hash value*
    - Used as indices of *hash table*

Hands-on code demo

unsplash.com/photos/a31khKMmRd0

Hashing in ML

- Integer-string maps in NLP
- Efficiently loading large training data set
- Any production data access
  - Use instead of arrays
  - E.g., cached, pre-computed features

unsplash.com/photos/a31khKMmRd0

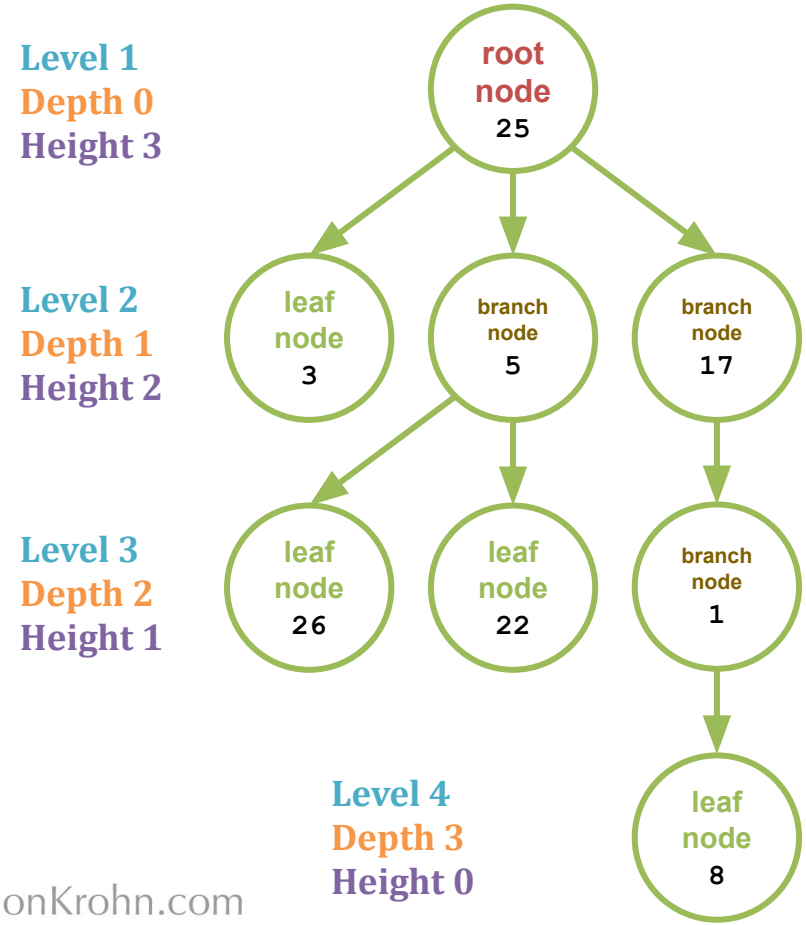
Data Structures & Algorithms

1. Intro to Data Structures and Algorithms
2. Lists and Dictionaries
3. **Trees and Graphs**

Segment 3: Trees and Graphs

- Trees
  - General Terminology
  - Decision Trees
  - Random Forests
  - Gradient-Boosting (XGBoost)
- Graphs
  - General Terminology
  - Directed vs Undirected Graphs
  - Directed Acyclic Graphs
- Resources for Further Study

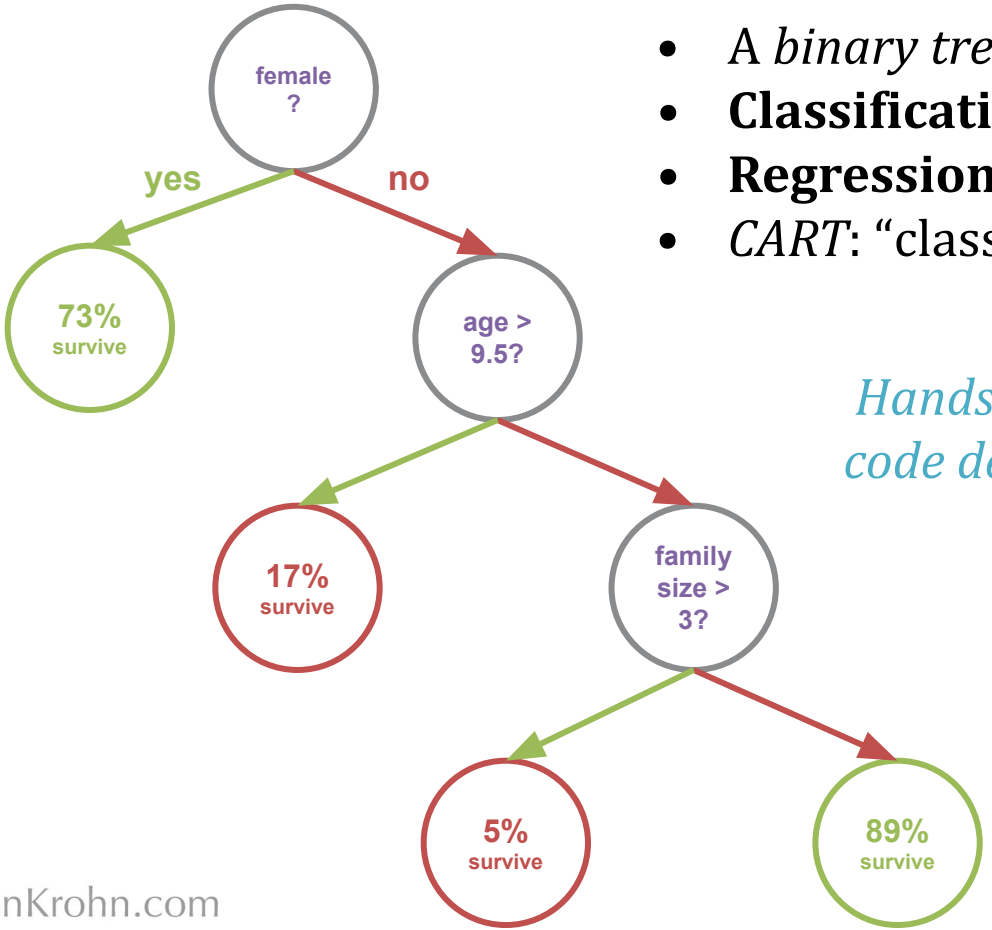
Trees



- Extension of linked list
- Start at *root node*
- Add *branch* (aka *internal*) *nodes*
- All nodes must be connected
- Each *child* node has one *parent*
- Childless nodes: *leaf* or *external*
- No cyclic connections
- Tree "height" = n_{levels} - 1 (i.e., root)

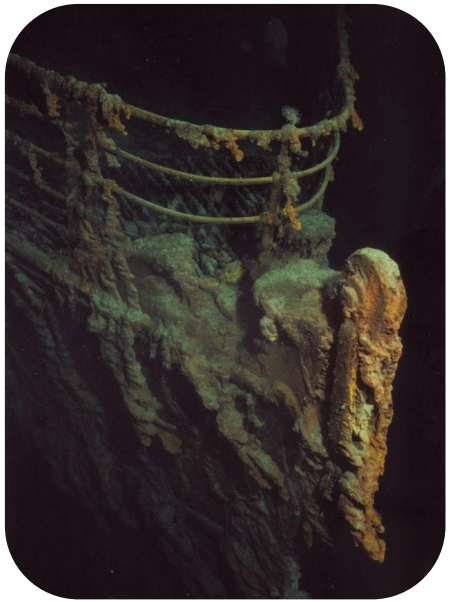
unsplash.com/photos/tGTVxeOr_Rs

Decision Trees



- A *binary tree* ("max. 2 child nodes per parent")
- **Classification tree**: predict categorical outcome
- **Regression tree**: predict continuous outcome
- *CART*: "classification and regression trees"

Hands-on code demo



unsplash.com/photos/hMwl23e44_Q

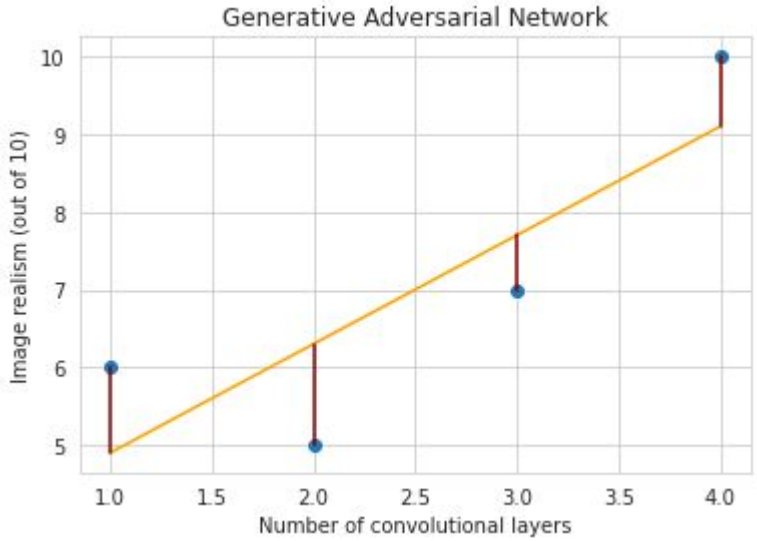
Random Forests

- Ensembles tend to outperform any single model
  - Less prone to overfitting relative to single decision tree
- Repeat this stochastic process, e.g., 100 times:
  - Use sampling with replacement to "bootstrap aggregate" (*bag*) a "new" sample of training data
  - Fit a decision tree; when splitting each node:
    - Use all of the features or,
    - Randomly subset the feature set
- Averaging diverse trees together tends to:
  - Reduce variance
  - Cancel out errors

Hands-on code demo

unsplash.com/photos/sp-p7uuT0tw

Gradient-Boosted Decision Trees



- Also an ensemble method
- However, trees are *not* independent
- 1st tree is a regular ol' decision tree
- 2nd tree trained on 1st's residuals (errors)
- 3rd tree trained on 2nd's residuals
- …
- Repeat until no more improvements
- Alongside deep learning, a common competition-winning approach

Hands-on code demo

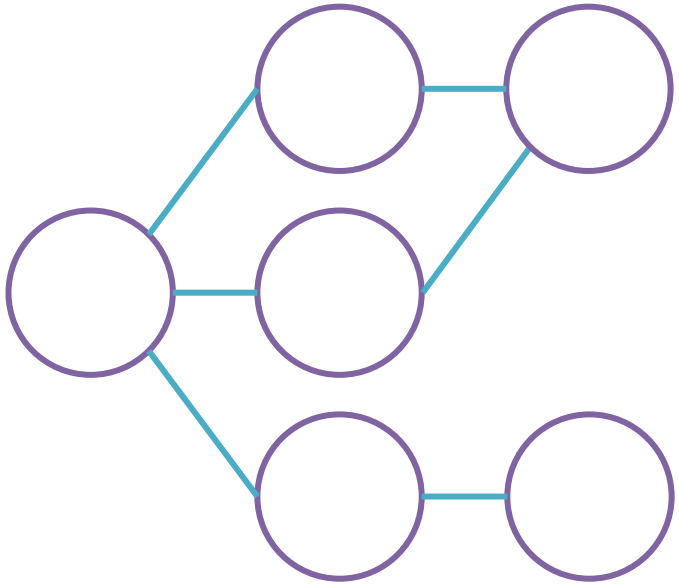
Other Tree Concepts

(Largely irrelevant to ML, so a quick skim only)

- More considerations during search relative to 1D data structure
  - Generally search has O(n) time complexity
- Likewise, data deletion or insertion have extra considerations, e.g.:
  - Handling children after deleting parent node
- **Binary search tree** (BST):
  - Binary tree wherein larger values are to the right of smaller ones
  - Typically much quicker O(log n) search
    - Except in O(n) worst case (i.e., *unbalanced*, a high level:node ratio)
  - "Self-balancing" BSTs automatically structured for O(log n) traversal
    - Used for ***k*-nearest neighbor** clustering algos
- **Heaps**: in "max heaps", parent has value > childrens'; opposite in min heap
  - ~Reduced time complexity, e.g., average O(n/2) search for binary heap

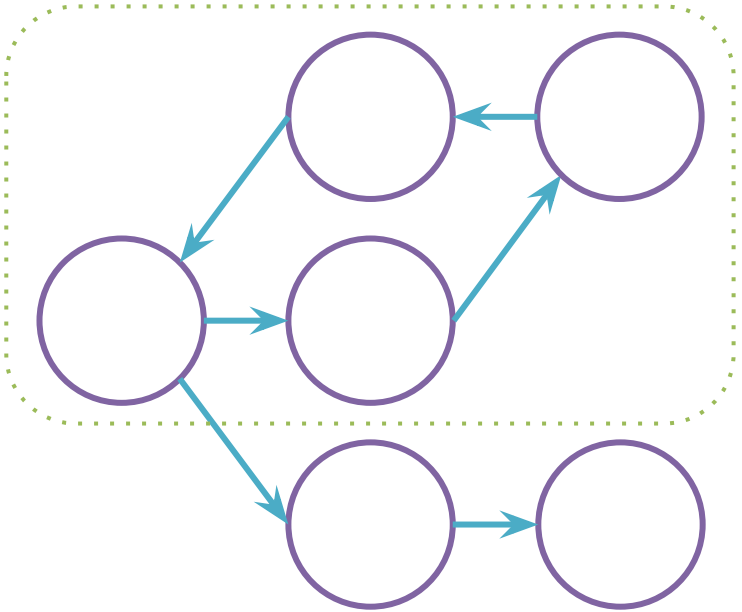
Graphs (a.k.a. Networks)

- **Nodes** (or *vertices*) are connected by **edges**
- Edges aren't necessarily "directed"
- Nodes typically store data, but edges can too
  - E.g., social network: Jane and Sal have met 25 times
- No general limitation on # of edges per node
- Generally, no particular root note
- Trees, however, are a specific type of graph



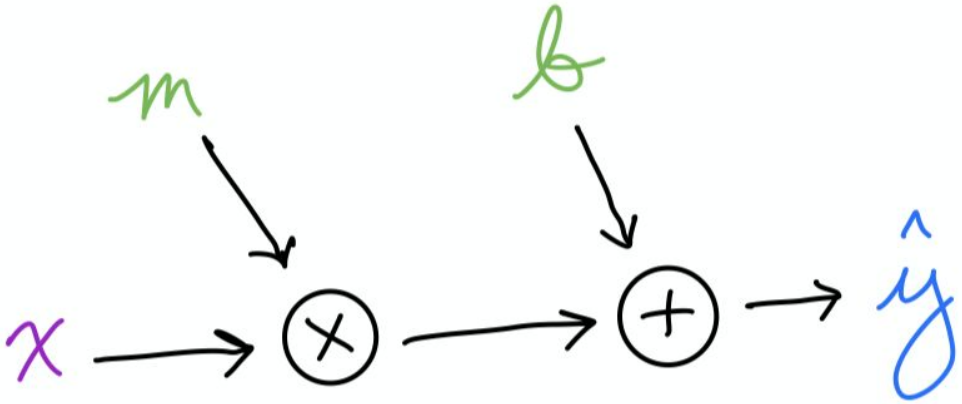
Directed Graphs

- **Edges** have direction
- Can convey temporal sequence through nodes
  - E.g., specifying a calculation or algo
- Can have **cycles**
  - ...which can lead to infinite loops



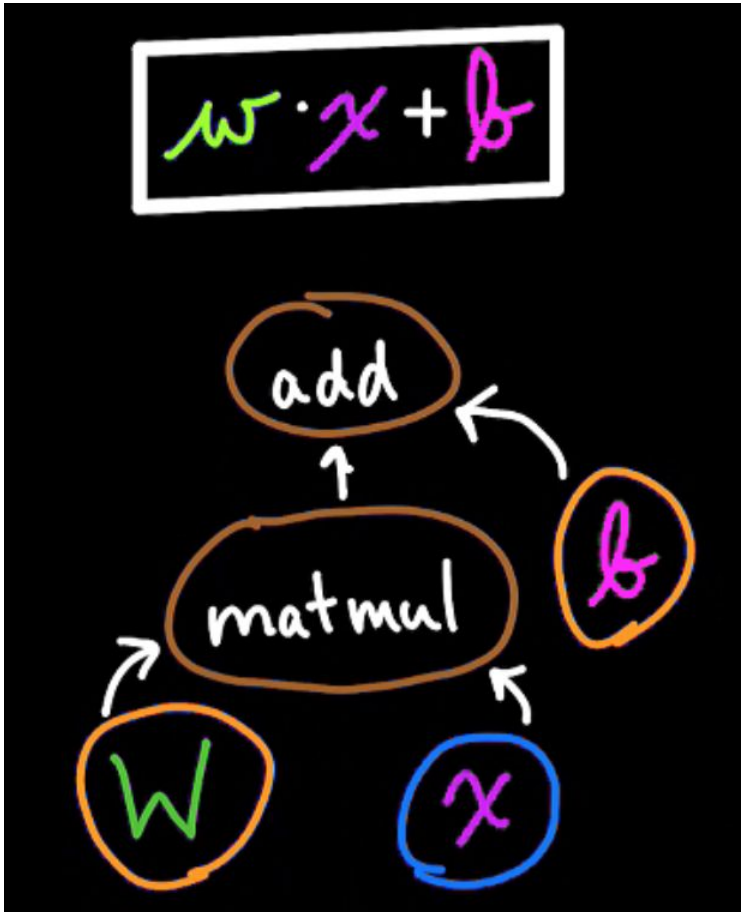
Directed *Acyclic* Graphs (DAGs)

- Directed graph with no cycles
- Line equation $y = mx + b$ as DAG (from *Calc I* and *Calc II*)
  - Nodes are input, output, parameters, or operations
  - Edges are tensors (but non-operation nodes can be too)



Hands-on code demo: `single-point-regression-gradient.ipynb`

Dense Neuron Layer DAG



- Operation
- Placeholder tensor input
- Variable tensor input

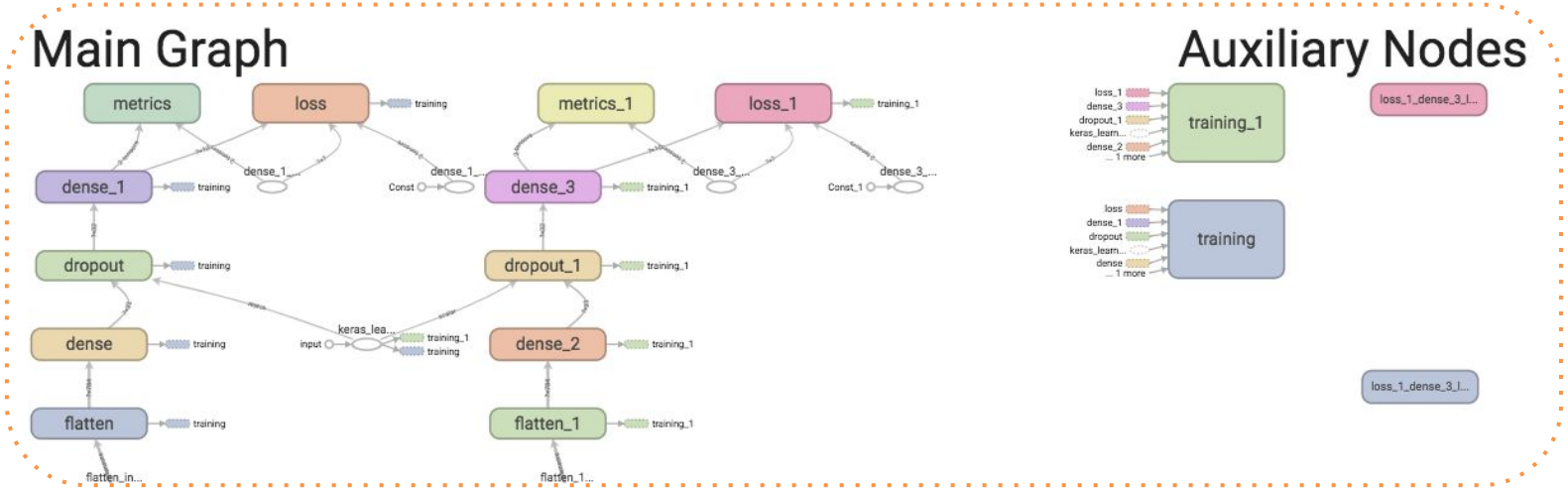
Hands-on code demos:

`jonkrohn.com/neurTF1`

`jonkrohn.com/deepTF1`

TensorBoard

- Compatible with TensorFlow: tensorflow.org/tensorboard
- ...and PyTorch: pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html
- Detail on Graphs: tensorflow.org/tensorboard/graphs



Other Graph Concepts

(Largely irrelevant to ML, so a quick skim only)

Connectivity

- A graph can have subgraphs that are unconnected
- Two graphs can have the same number of nodes
  - One with more edges has relatively greater connectivity

Graph traversal

- *Depth-first search*: follow a given path as far as it will go
- *Breadth-first search*: priority is traversing adjacent nodes

Bonus: What about Pandas??

- Critical data structures in Python ML
- Not a traditional CS data structures
- **DataFrame**
  - 2D matrix array
  - Columns of independent types
- **Series**: 1D vector array
- Lots of:
  - Convenient data methods
  - Optimization

unsplash.com/photos/c9mIIOlNzj4

Resources for Further Study

- Udacity *Data Structures & Algorithms in Python* course
  - Free
  - Clear, concise videos
  - Python exercises with interactive testing
- Sedgewick & Wayne (2011) *Algorithms* (4th ed.)
  - Free, standalone website to accompany textbook
    - Wonderful visualizations of theory
    - Note: Accompanying code's in Java
- Bhargava (2016) *Grokking Algorithms*
- Classic DSA computer science problems:
  - Shortest Path
  - Traveling Salesman
  - Knapsack

Next (Final) Subject: *Optimization*

- Ties the preceding seven subjects together
- How we descend gradients of cost w.r.t. model parameters
  - Granular, matrix-level operations
  - Quickly, abstractly with auto-differentiation libraries, e.g.:
    - PyTorch
    - TensorFlow
- “Fancy” optimizers common in deep learning, e.g.:
  - RMSProp
  - Adam
  - Nadam

POLL *with Multiple Answers Possible*

What other topics interest you most?

- Linear Algebra
- Calculus
- Probability Theory
- Statistics
- More Computer Science (e.g., algorithms, data structures)
- Machine Learning Basics
- Advanced Machine Learning, incl. Deep Learning
- Something Else

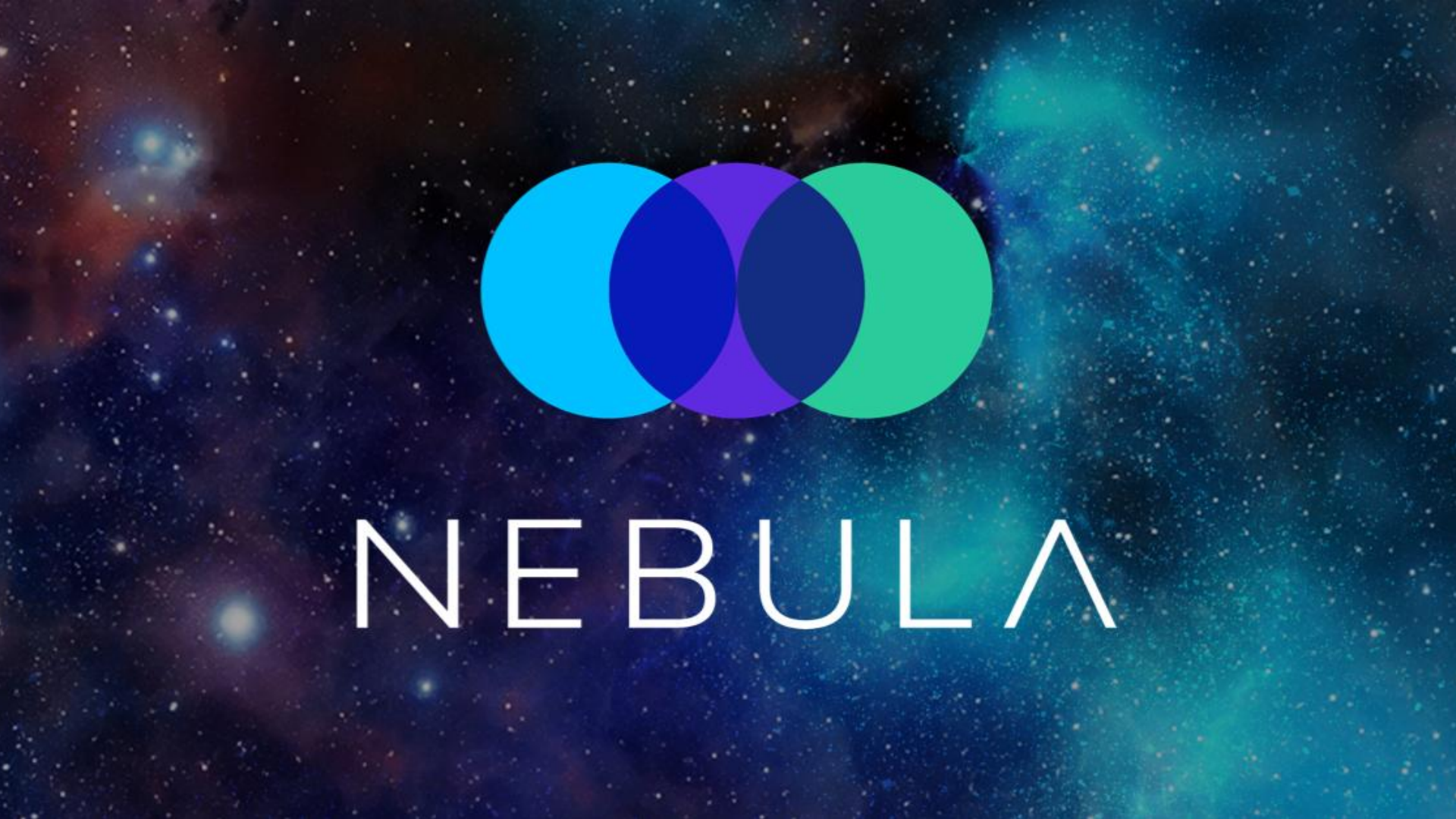
Stay in Touch

jonkrohn.com to sign up for email newsletter

linkedin.com/in/jonkrohn

youtube.com/c/JonKrohnLearns

twitter.com/JonKrohnLearns


NEBULA

PLACEHOLDER
FOR:

5-Minute Timer

PLACEHOLDER
FOR:

10-Minute Timer

PLACEHOLDER
FOR:

15-Minute Timer