ML with PyTorch for Developers

Bruno Gonçalves

www.data4sci.com/newsletter

data4sci.substack.com

https://github.com/DataForScience/PyTorch

Events

data4sci.substack.com

LLMs for Data Science

Mar 26, 2025 - 10am-2pm (PST)

NLP with PyTorch

Apr 9, 2025 - 10am-2pm (PST)

Bruno Gonçalves

https://data4sci.com

info@data4sci.com

https://data4sci.com/call

Question

http://github.com/DataForScience/PyTorch/

- What's your job title?
    - Data Scientist
    - Statistician
    - Data Engineer
    - Researcher
    - Business Analyst
    - Software Engineer
    - Other

Question

http://github.com/DataForScience/PyTorch/

- How experienced are you in Python?
    - Beginner (<1 year)
    - Intermediate (1-5 years)
    - Expert (5+ years)

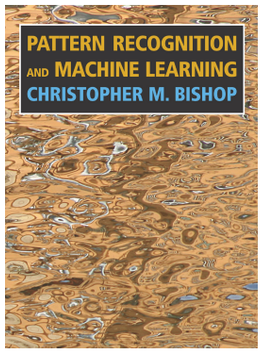
Question

http://github.com/DataForScience/PyTorch/

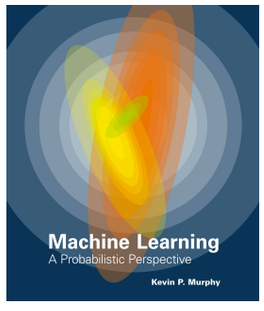
- How did you hear about this webinar?
    - O'Reilly Platform
    - Newsletter
    - data4sci.com Website
    - Previous event
    - Other?

References

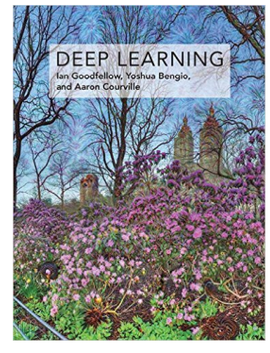
http://github.com/DataForScience/PyTorch/



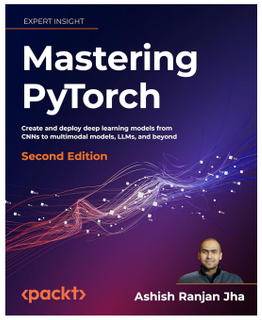
https://amzn.to/2AcDHmX



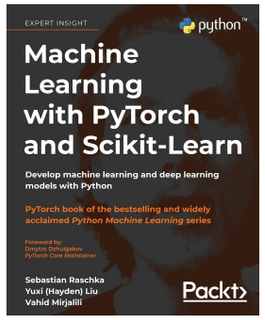
https://amzn.to/3gZZPBu



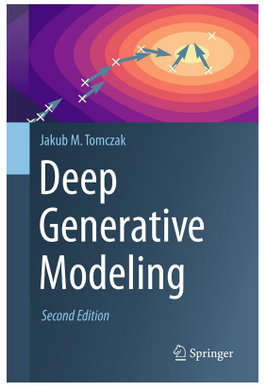
https://amzn.to/2BGr0RL



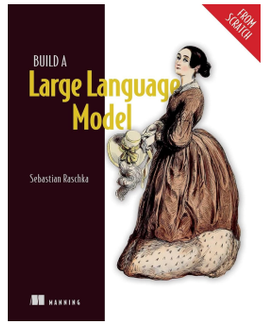
https://amzn.to/4btVVgI



https://amzn.to/41wW1Qu



https://amzn.to/4koG7zY

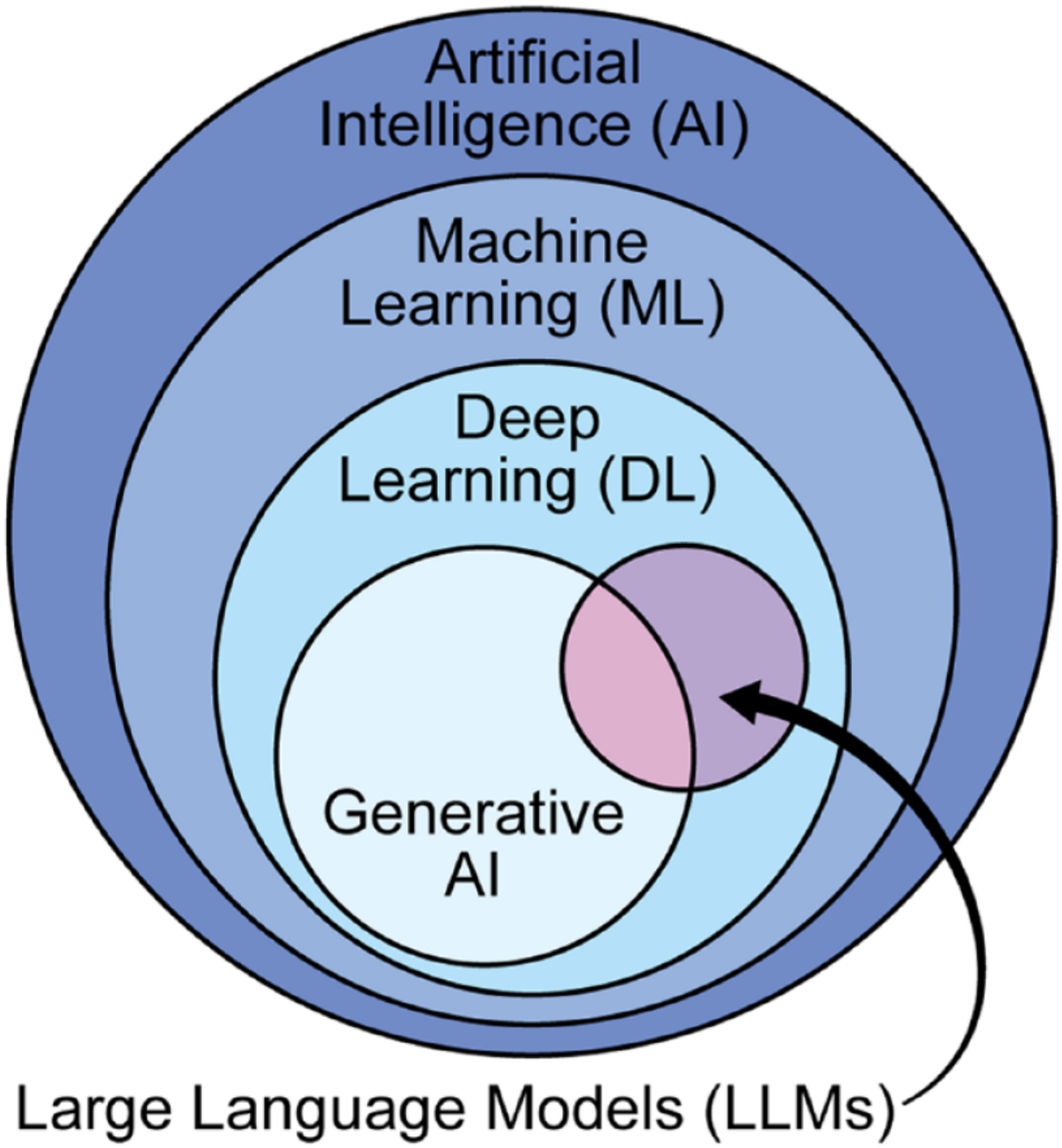


https://amzn.to/3F69ffh

Table of Contents

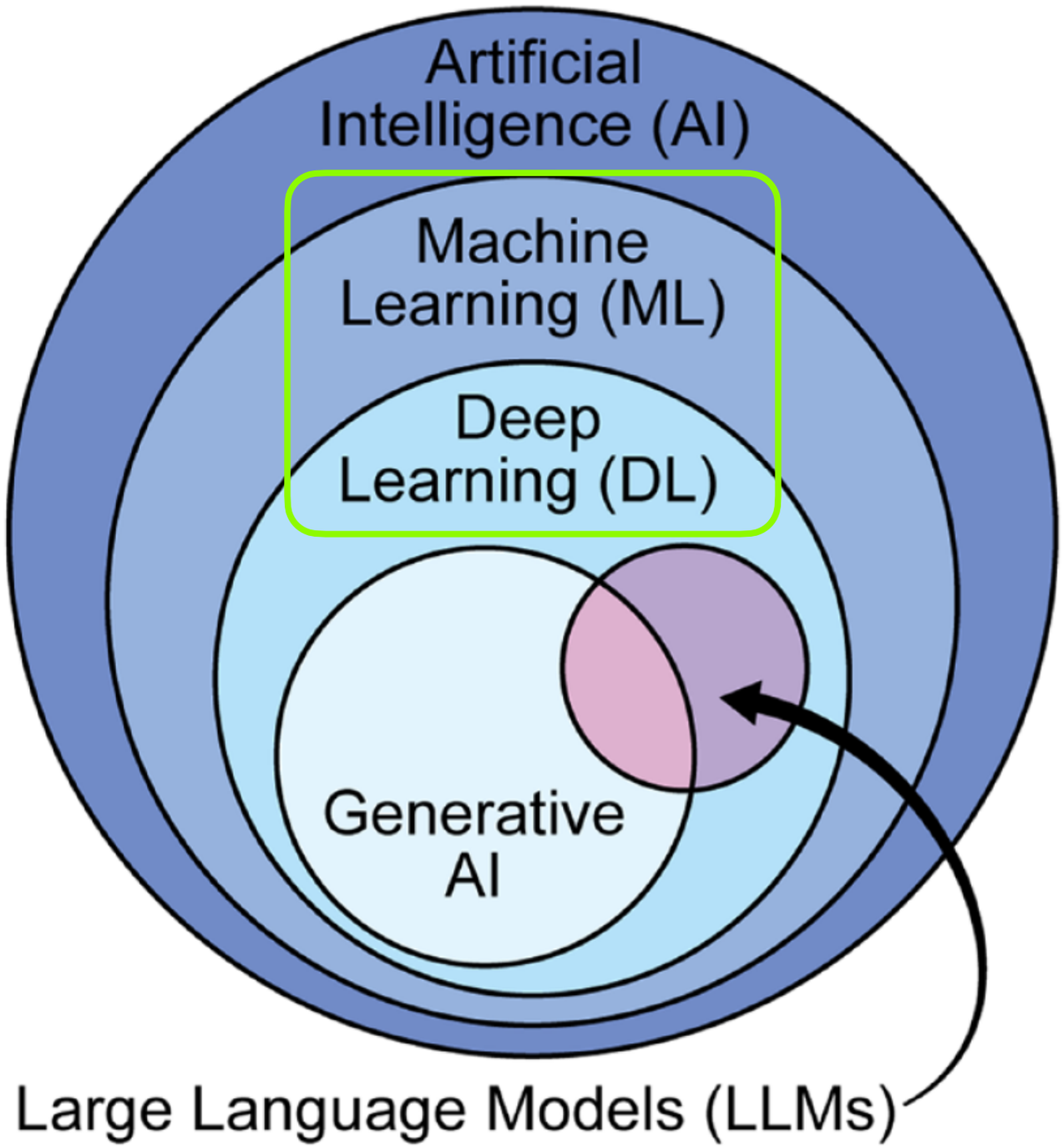
1. Machine Learning Overview

What are we talking about?



- Artificial Intelligence (AI): A field of CS focused on creating intelligent agents.
- Machine Learning (ML): Subset of AI focused on developing algorithms that can learn from data.
- Deep Learning (DL): Neural networks with many layers
- Generative Models: Models that can generate new data based on learned patterns
- Large Language Models (LLMs): Statistical models used to predict words in natural language

What are we talking about?



- Artificial Intelligence (AI): A field of CS focused on creating intelligent agents.
- Machine Learning (ML): Subset of AI focused on developing algorithms that can learn from data.
- Deep Learning (DL): Neural networks with many layers
- Generative Models: Models that can generate new data based on learned patterns
- Large Language Models (LLMs): Statistical models used to predict words in natural language

Basic Principles of Machine Learning

- Data is the Foundation
    - Large, diverse, and relevant datasets typically lead to better-performing models
- Learning is Pattern Recognition
    - “Learning” means approximating the relationships or distributions within data in order to perform well on new, unseen data.
- Model Fitting and Parameter Optimization
    - Algorithms try to optimize a “loss function” by adjusting model parameters to minimize the loss.
- Bias vs Variance
    - Balancing how well the model fits known and unknown data
- Evaluation and Iteration
    - Model development is an iteratively process

3 Types of Machine Learning

- Unsupervised Learning
  - Autonomously learn an good representation of the dataset
  - Find clusters in input

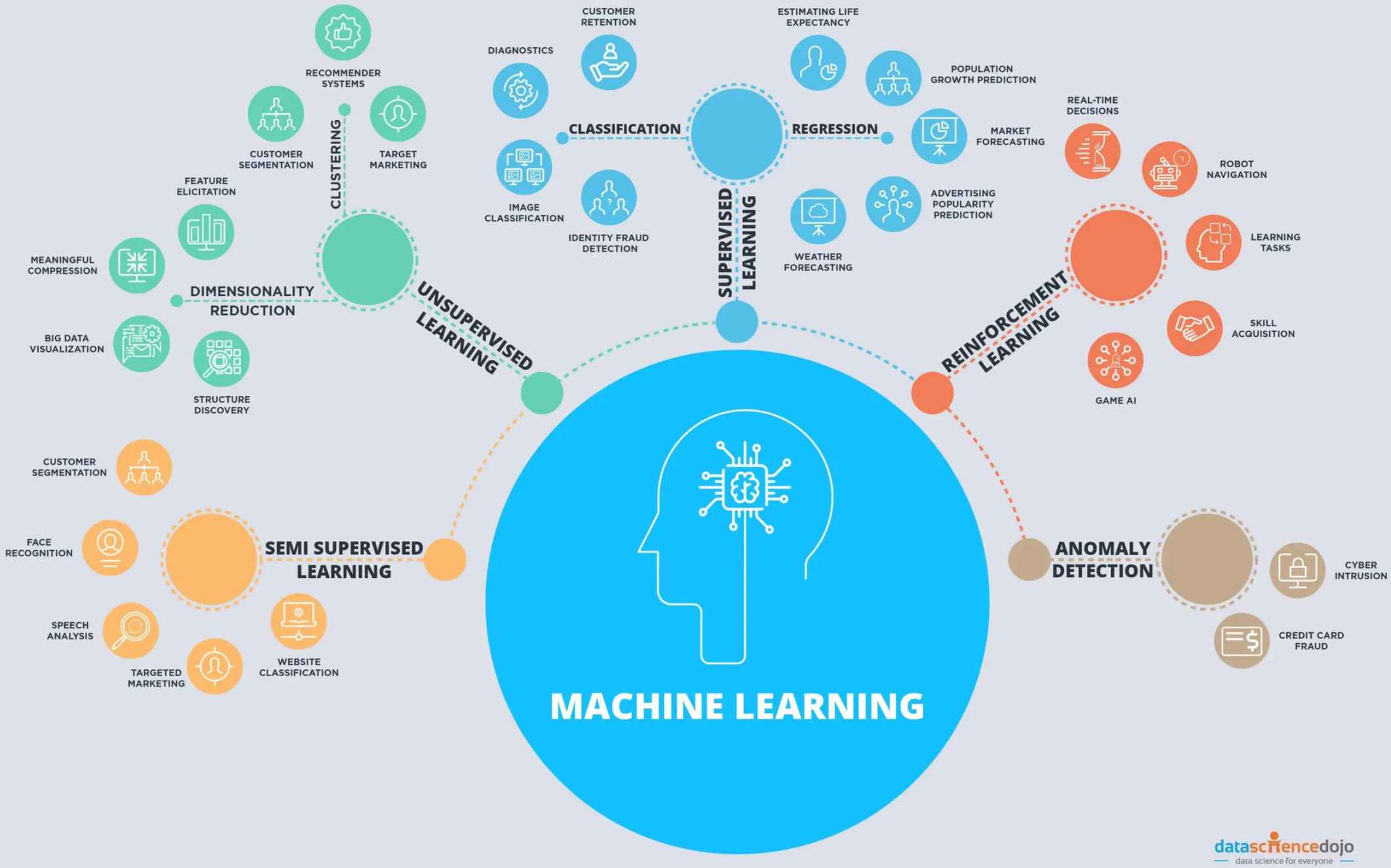
3 Types of Machine Learning

- Unsupervised Learning
  - Autonomously learn an good representation of the dataset
  - Find clusters in input
- Supervised Learning
  - Predict output given input
  - Training set of known inputs and outputs is provided

3 Types of Machine Learning

- Unsupervised Learning
  - Autonomously learn an good representation of the dataset
  - Find clusters in input
- Supervised Learning
  - Predict output given input
  - Training set of known inputs and outputs is provided
- Reinforcement Learning
  - Learn sequence of actions to maximize payoff
  - Discount factor for delayed rewards

3 Types of Problems



3 Types of Machine Learning

- Unsupervised Learning
  - Autonomously learn an good representation of the dataset
  - Find clusters in input
- Supervised Learning
  - Predict output given input
  - Training set of known inputs and outputs is provided
- ~~Reinforcement Learning~~
  - Learn sequence of actions to maximize payoff
  - Discount factor for delayed rewards

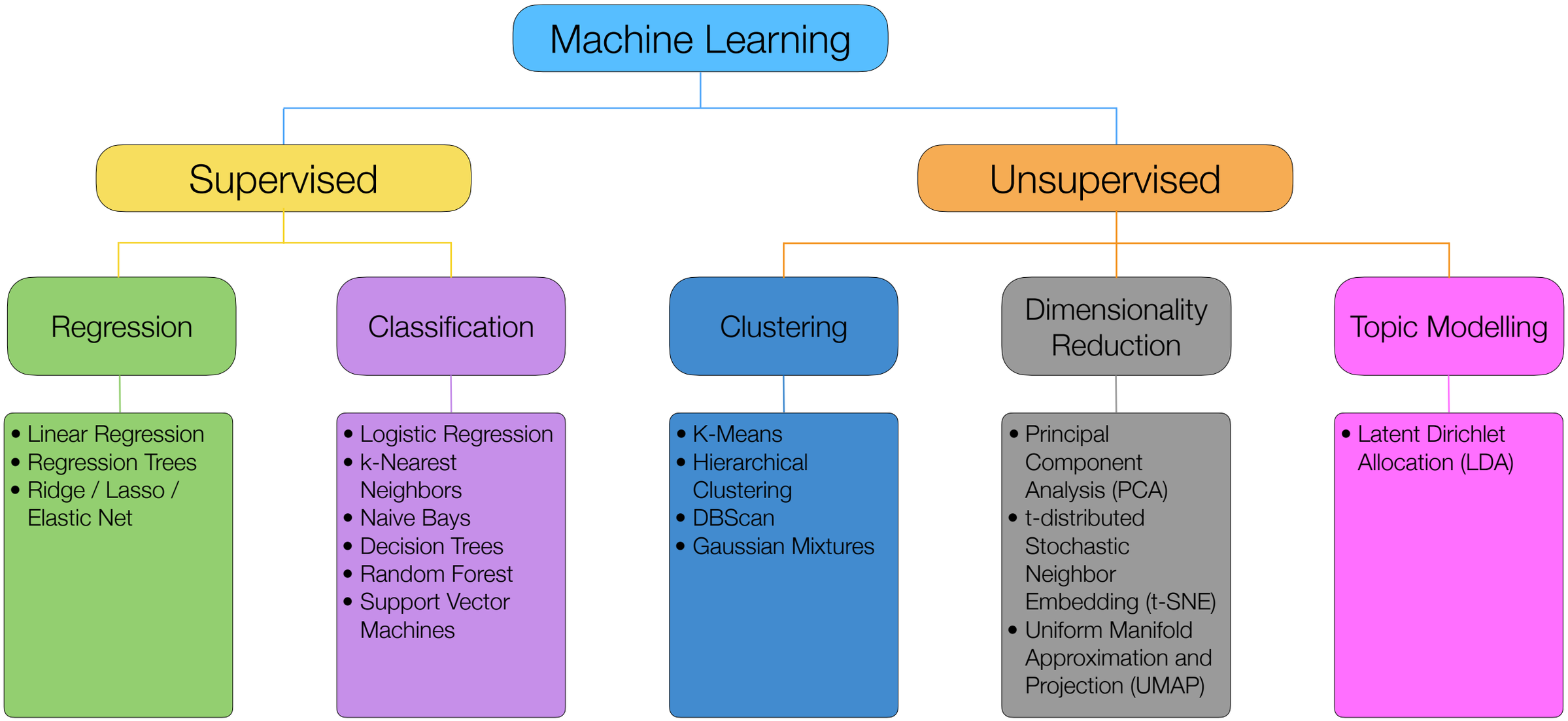
Unsupervised Learning

- Unlabeled Data: useful for discovering hidden structures in raw datasets.
- Groups similar data points (clustering) and finds structure in data
- Two main types of problems:
    - Dimensionality Reduction: Find the most relevant features or combinations of features
    - Clustering: Group similar instances together
- Common in customer segmentation, fraud detection, recommendation systems, anomaly detection, and image compression.
- No clear way to evaluate model accuracy or choose the right number of clusters k without labels

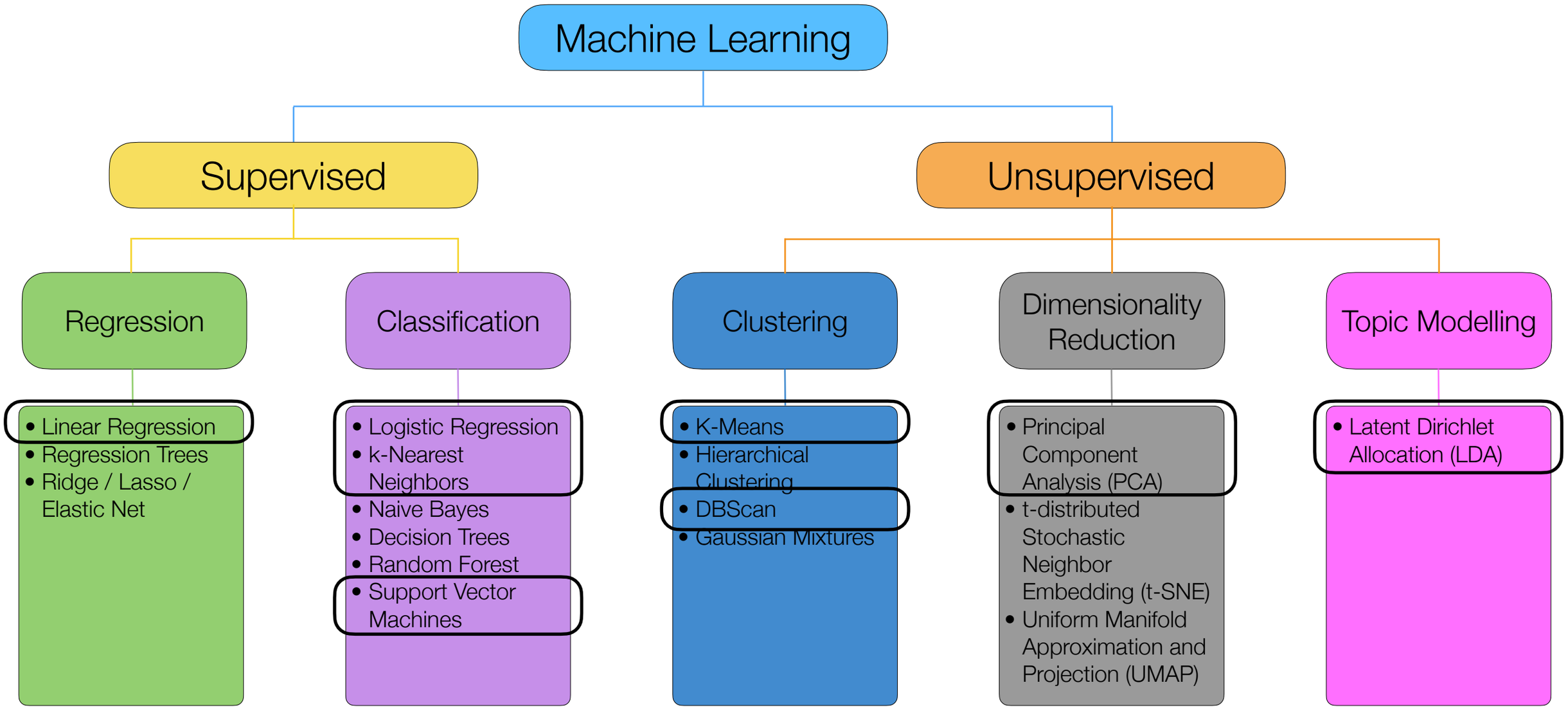
Supervised Learning

- Supervised learning depends on feature-label pairs
- Predicts continuous or discrete values
- Two main types of problems:
    - Classification: Assigning labels to instances.
    - Regression: Predicts numerical values.
- Common in fraud detection, medical diagnosis, speech recognition, spam filtering, stock price prediction, and recommendation systems.
- Data is split into Training and Testing datasets
- Model performance is measured using:
    - Accuracy, Precision, Recall, F1-score (for Classification).
    - Mean Squared Error (MSE), R^2 Score (for Regression).
- Computationally Expensive for Large Datasets

Machine Learning Overview



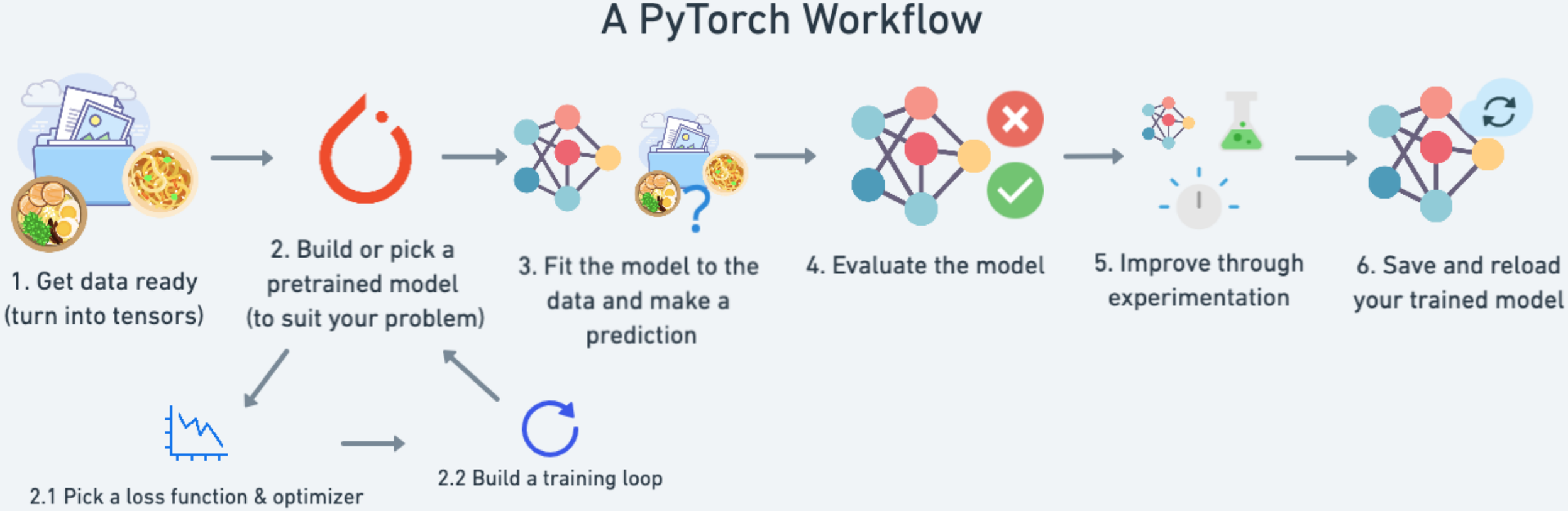
Machine Learning Overview



Machine Learning Pipelines

- ML pipelines are automated sequence of steps that build, train, and evaluate a machine learning model efficiently.

- MLOps integrates pipelines into Continuous Integration/Continuous Deployment workflows, ensuring automated retraining, versioning, reproducibility, and scalable deployment in production systems.



Machine Learning as Optimization

- (Machine) Learning can (often) be thought of as an optimization problem.

- Optimization Problems have 3 distinct pieces:

    - The constraints

    - The function to optimize

    - The optimization algorithm.

Machine Learning as Optimization

- (Machine) Learning can (often) be thought of as an optimization problem.

- Optimization Problems have 3 distinct pieces:

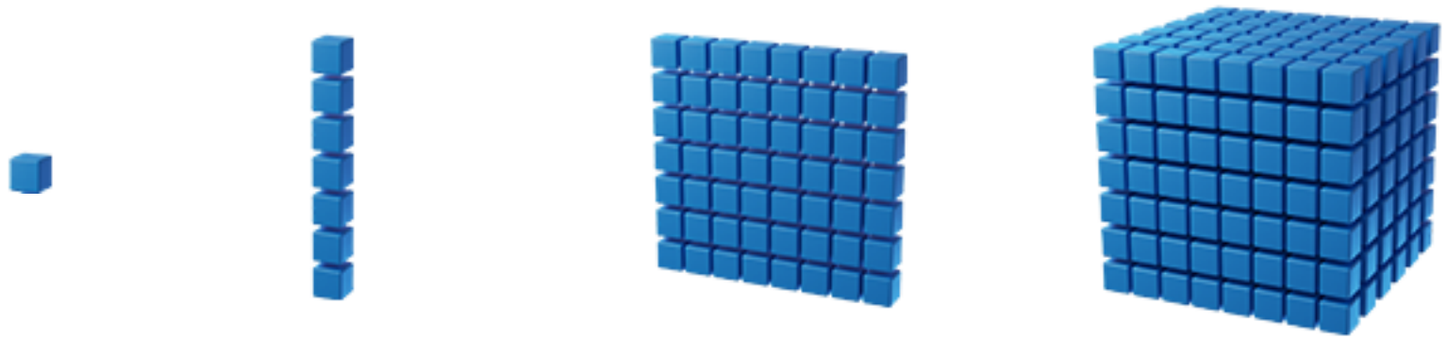
  - The constraints — Problem Representation
  - The function to optimize — Prediction Error
  - The optimization algorithm. — Gradient Descent

PyTorch Overview

- Developed by Facebook's AI Research (FAIR) team and initially released in January 2017.
- High level structure allowing for intuitive model definition, especially for complex architectures
- Robust Tensor data structure and automatic differentiation mechanism to compute gradients with minimal overhead
- Seamless support for a wide range of specialized hardware (GPUs, MPS, etc)
- Ready-to-use building blocks that can be chained together to create models ranging from simple linear regression to state of the art deep learning architectures
- Support for distributed and multi-GPU Training
- Large community contributes new models, tutorials, and extensions regularly

Tensors

- Tensors are a mathematical generalization of scalars, vectors, and matrices to arbitrary dimensions:



0D Tensor (scalar) | 1D Tensor (vector) | 2D Tensor (matrix) | 3D Tensor

- 0D tensor (rank-0): a single number like 5
- 1D tensor (rank-1): an array of numbers [1, 2, 3]
- 2D tensor (rank-2): a matrix of numbers [[1, 2], [3, 4]]
- Higher-dimensional tensors continue this pattern with 3D, 4D, and beyond.

Tensors

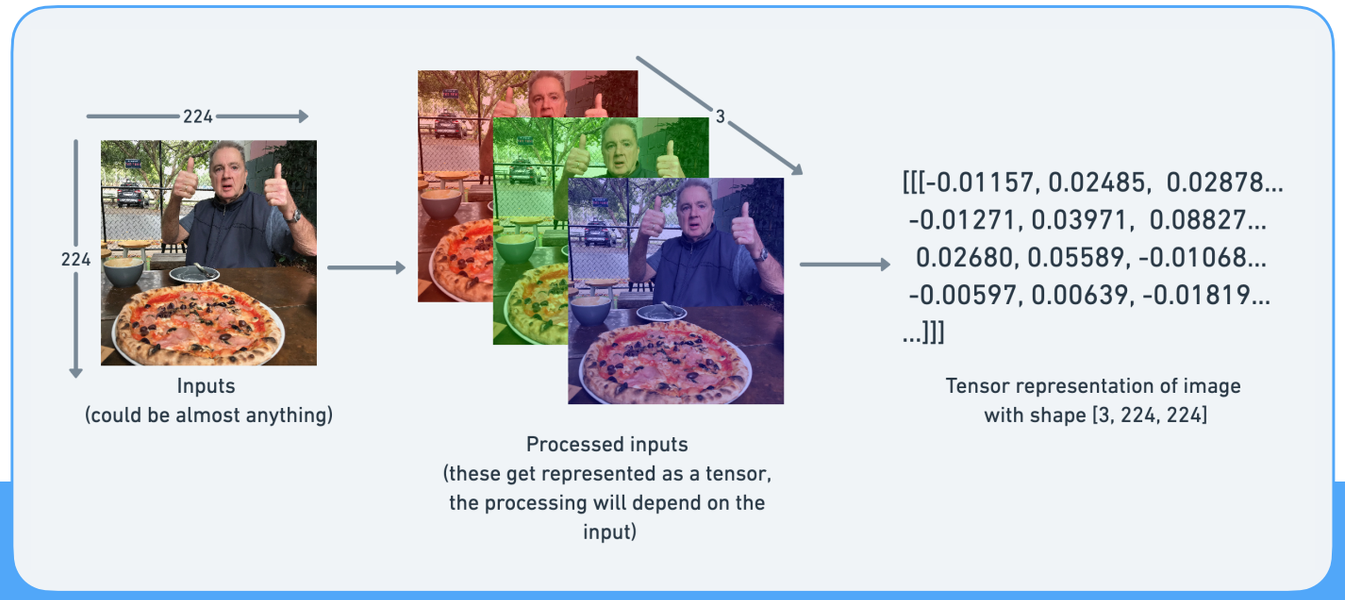
https://github.com/mrdbourke/pytorch-deep-learning/blob/main/00_pytorch_fundamentals.ipynb

- In PyTorch, tensor is the fundamental primary data structure: multidimensional arrays that can be efficiently manipulated on GPUs.
- Tensors are a unified way of describing complicated systems.
- Tensors in practice:
  - Timeseries (1D tensor: timesteps)
  - Tabular data (2D tensor: rows x columns)
  - Images (3D tensors: height × width × color channels)
  - Video (4D tensors: frames × height × width × channels)
  - Textual data (embedding dimensions × sequence length × batch size)

Tensors

https://github.com/mrdbourke/pytorch-deep-learning/blob/main/00_pytorch_fundamentals.ipynb

- In PyTorch, tensor is the fundamental primary data structure: multidimensional arrays that can be efficiently manipulated on GPUs.
- Tensors are a unified way of describing complicated systems.
- Tensors in practice:
  - Timeseries (1D tensor: timesteps)
  - Tabular data (2D tensor: rows x columns)
  - Images (3D tensors: height × width × color channels)
  - Video (4D tensors: frames × height × width × channels)
  - Textual data (embedding dimensions × sequence length × batch size)



Tensors in PyTorch

https://github.com/mrdbourke/pytorch-deep-learning/blob/main/00_pytorch_fundamentals.ipynb

- Tensors are the fundamental data structure, conceptually similar to NumPy arrays
- Features:
    - Automatic Differentiation: allow PyTorch to automatically compute gradients with respect to the operations performed
    - GPU acceleration: Tensors can reside on either CPU or GPU.
        - Wide range of data types: boolean, integer, floating point, and complex in 8, 16, 32, 64, and 128 bit formats
- Extensive support for mathematical and linear algebra operations

Hardware Acceleration

https://pytorch.org/docs/stable/backends.html

- PyTorch is capable of leveraging a wide range of specialized hardware:
  - Nvidia GPUs - **device** = "cuda"
  - AMD GPUs - **device** = "cuda" 🤓
  - Apple Silicon - **device** = "mps"
- Tensors can be moved between devices with a simple **.to**(device) call
- Leverage multi-core CPUs, vectorized instructions (via libraries like MKL or OpenBLAS), and efficient threading for faster CPU-based computations.
- Utilities like DataParallel and DistributedDataParallel allow PyTorch to scale models across multiple GPUs, in or or more machines, improving throughput and training speed.

Advanced Features

- PyTorch comes complete with high-level frameworks, to simplify developing complex algorithms:
  - torch.nn module: Provides building blocks (layers, loss functions) for neural network construction.
  - torch.optim: A suite of optimization algorithms (e.g., SGD, Adam) that can be coupled with PyTorch Tensors and autograd for easy training loops.
- Specialized packages have been developed that leverage PyTorch's tensors under the hood:
  - torchvision - popular datasets, model architectures, and common image transformations for computer vision.
  - torchtext - Text utilities, models, transforms, and datasets for PyTorch (deprecated)
  - torchaudio - Building Blocks for Audio and Speech Processing.

Machine Learning Overview

http://github.com/DataForScience/PyTorch/

2. Unsupervised Learning

Use Cases

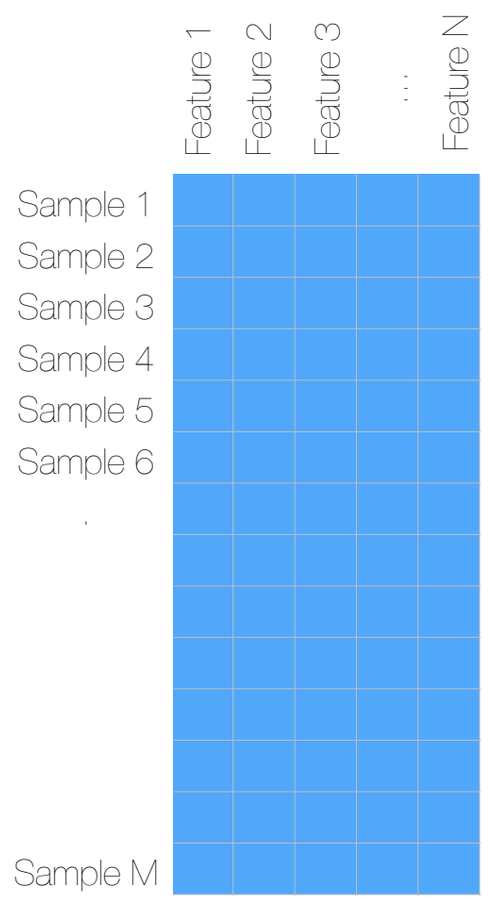
- Visualization
- Data exploration
- Pattern recognition
- Dimensionality reduction
- Clustering
- etc

Use Cases

- Visualization
- Data exploration
- Pattern recognition
- Dimensionality reduction - PCA
- Clustering - K-Means, DBSCAN
- etc

Data Preparation

- Dataset formatted as an MxN matrix of M instances and N features
- Each column corresponds to a different dimension in the feature space
- Each row represents a different sample or instance
- Unsupervised Learning can
  - Reduce the number of features (columns)
  - Group instances (rows) by how similar they are

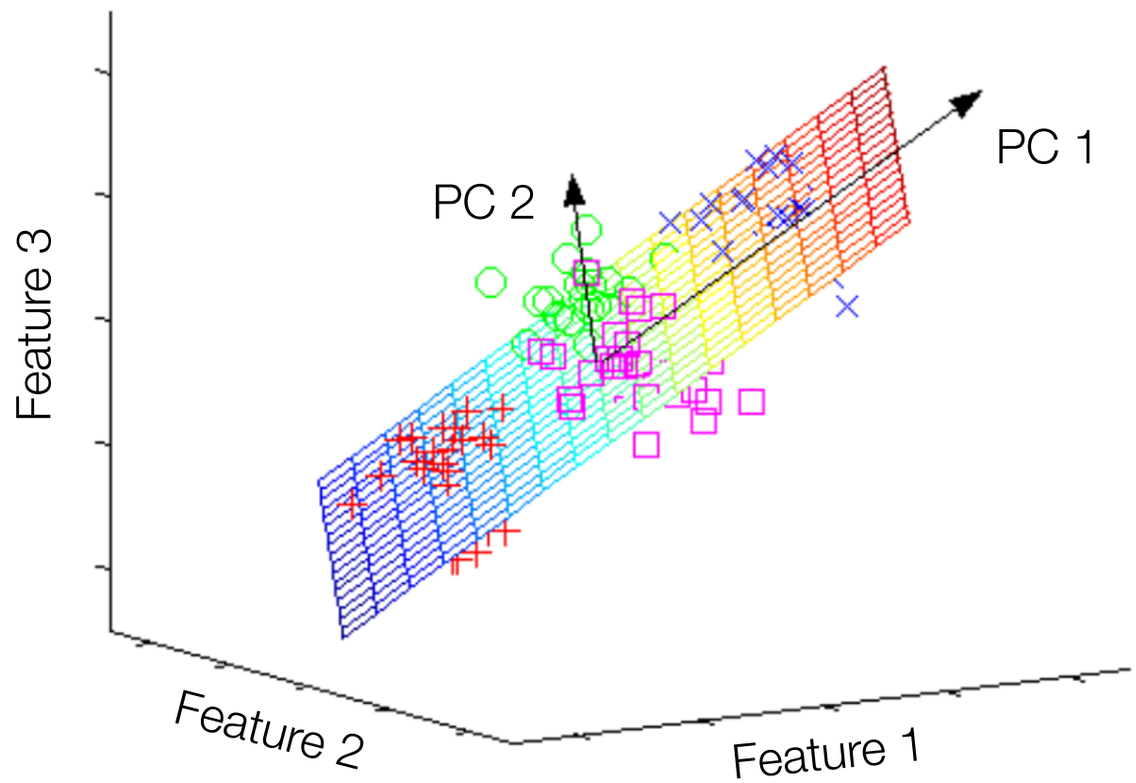


Principal Component Analysis (PCA)

J. of Ed. Psy., 24, 417 (1933)

- Finds the directions of maximum variance of the dataset
- Useful for dimensionality reduction
- Often used as preprocessing of the dataset

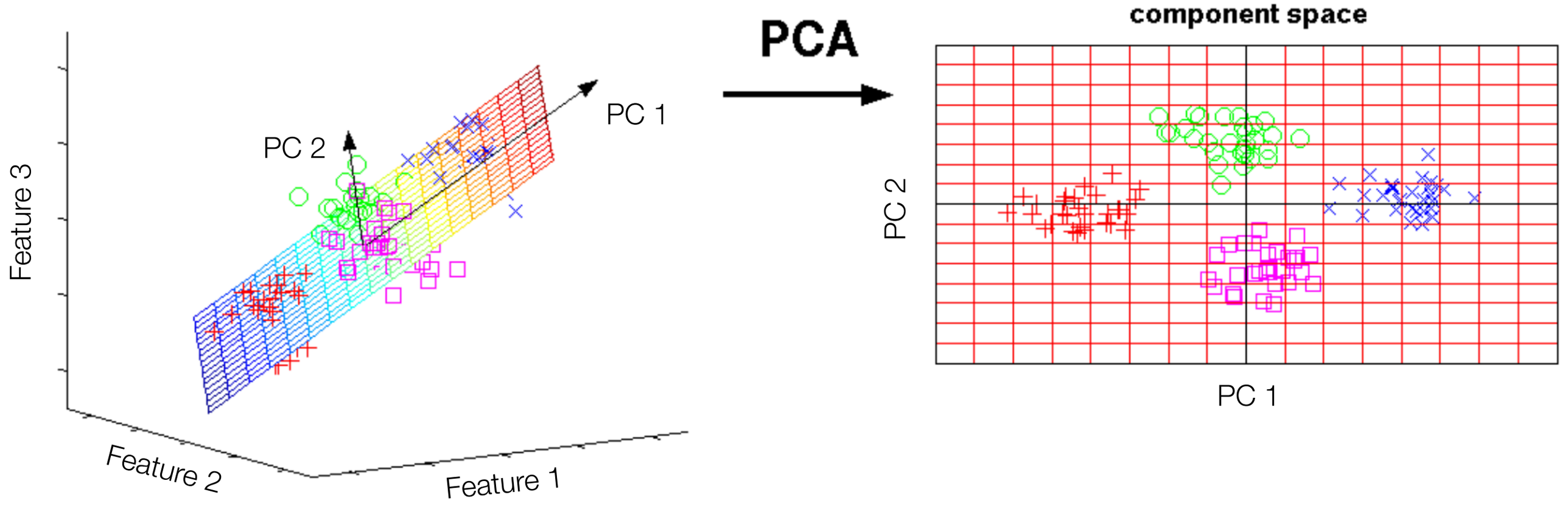
Original Data Space



Principal Component Analysis (PCA)

J. of Ed. Psy., 24, 417 (1933)

- Finds the directions of maximum variance of the dataset
- Useful for dimensionality reduction
- Often used as preprocessing of the dataset



Principal Component Analysis (PCA)

J. of Ed. Psy., 24, 417 (1933)

- The Principal Component projection, T, of a matrix A is:

$$T = AW$$

- W is the eigenvector matrix of:

$$AA^T$$

- And corresponds to the right singular vectors of A obtained by Singular Value Decomposition (SVD):

$$A = U\Sigma W^T$$

Generalization of Eigenvalue/Eigenvector decomposition for non-square matrices.

- So we can write:

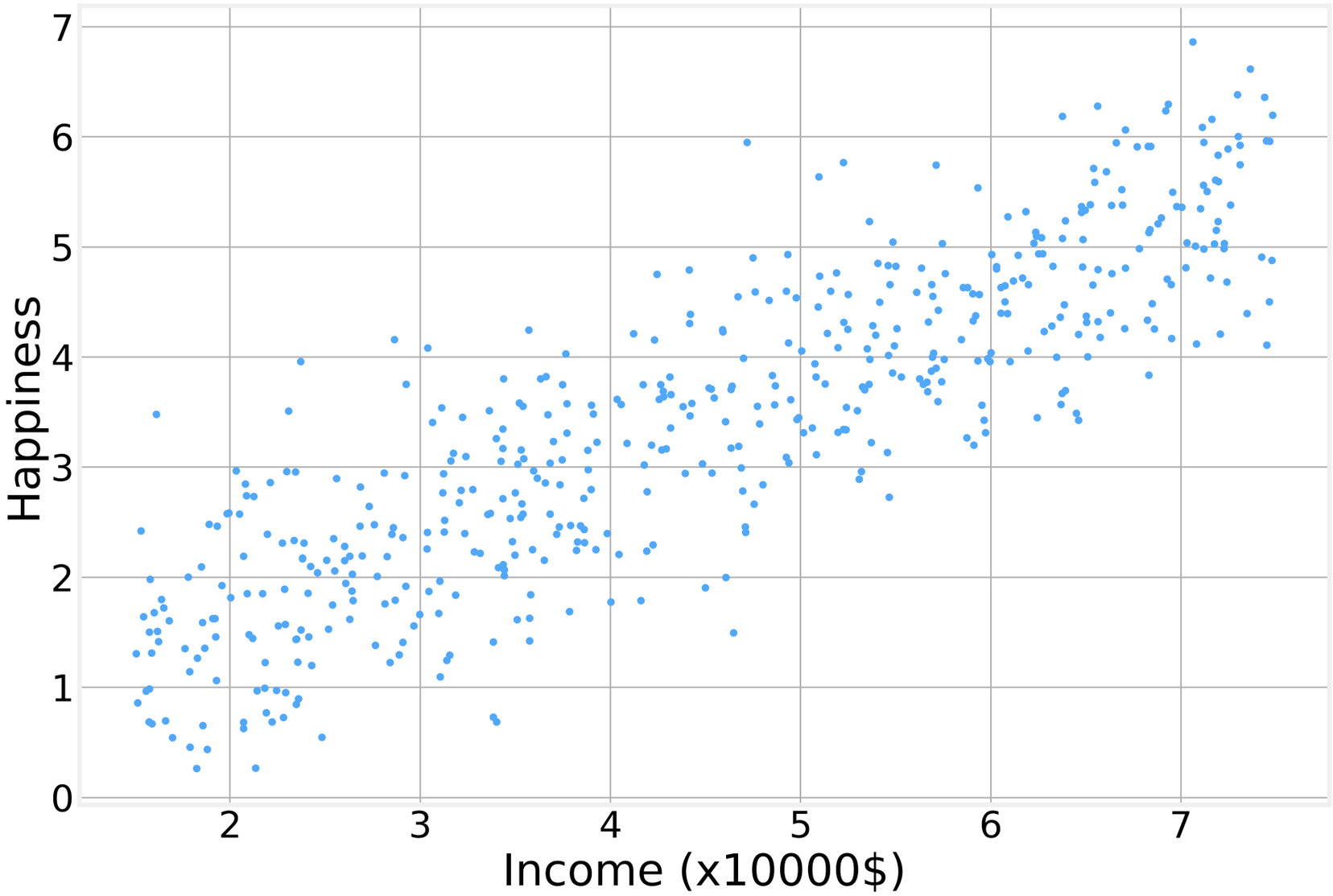
$$T = U\Sigma W^T W \equiv U\Sigma$$

- Showing that the Principle Component projection corresponds to the left singular vectors of A scaled by the respective singular values

- Columns of T are ordered in order of decreasing variance.

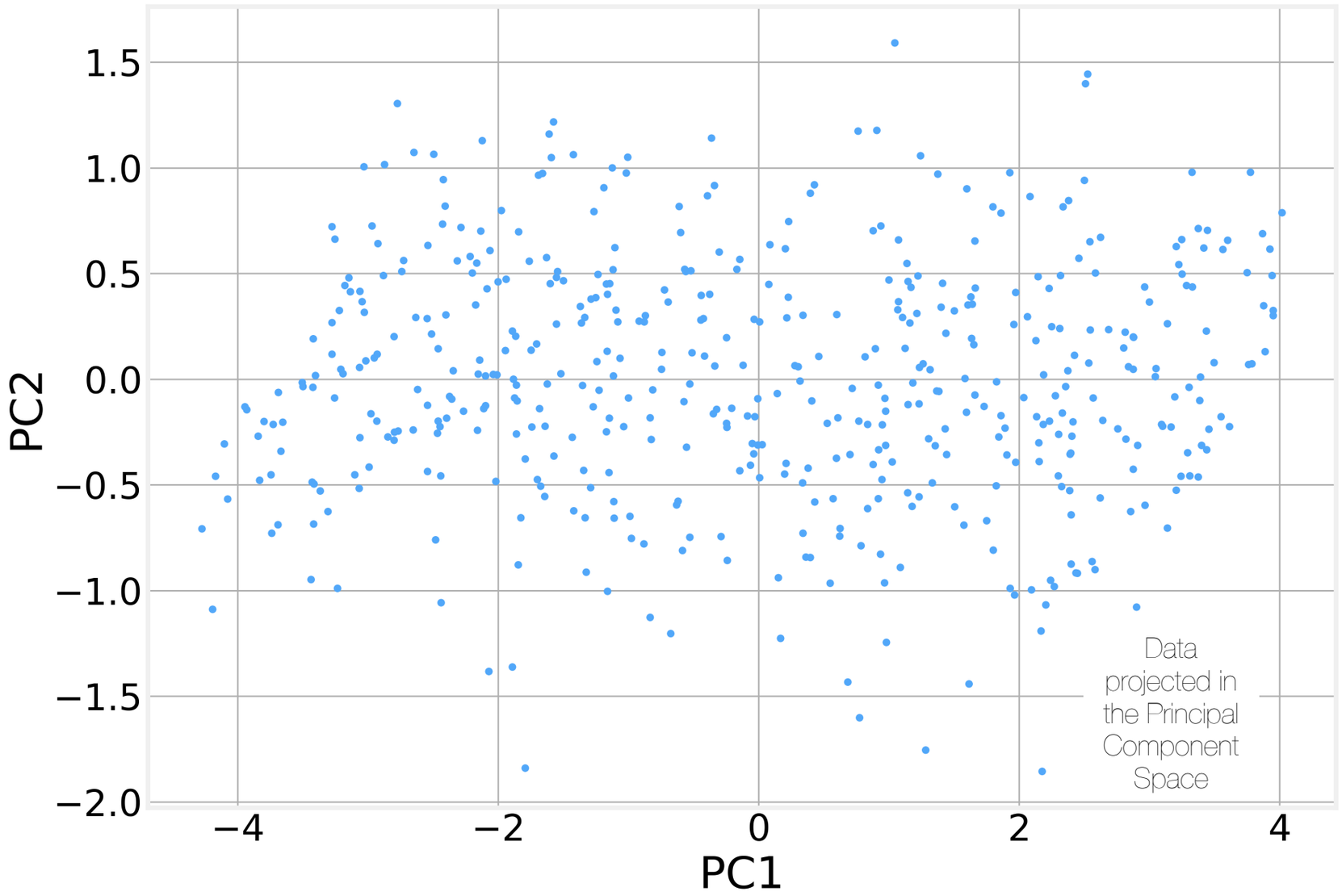
Principal Component Analysis (PCA)

https://www.scribbr.com/wp-content/uploads//2020/02/income.data_.zip



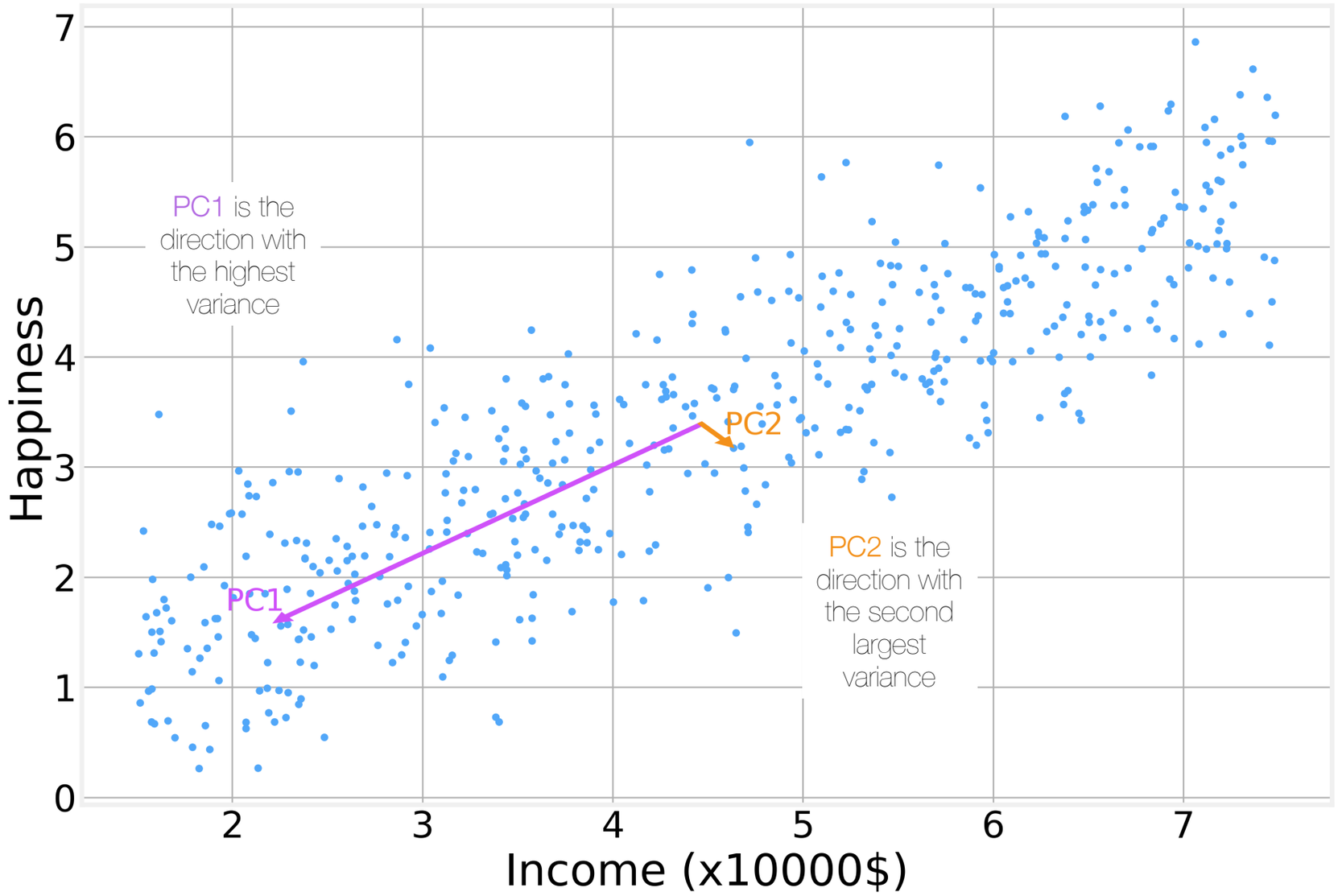
Principal Component Analysis (PCA)

https://www.scribbr.com/wp-content/uploads//2020/02/income.data_.zip



Principal Component Analysis (PCA)

https://www.scribbr.com/wp-content/uploads//2020/02/income.data_.zip



K-Means

Bull. Acad. Polon. Sci. 4, 801 (1957)

- Choose k randomly chosen points to be the centroid of each cluster
- Assign each point to belong the cluster whose centroid is closest
- Recompute the centroid positions (mean cluster position)
- Repeat until convergence (centroids stop moving)

K-Means: Convergence

Bull. Acad. Polon. Sci. 4, 801 (1957)

- How to quantify the "quality" of the solution found at each iteration, n?

- Measure the "Inertia", the square intra-cluster distance:

$$I_n = \sum_{i=0}^{N} \| x_i - \mu_i \|^2$$

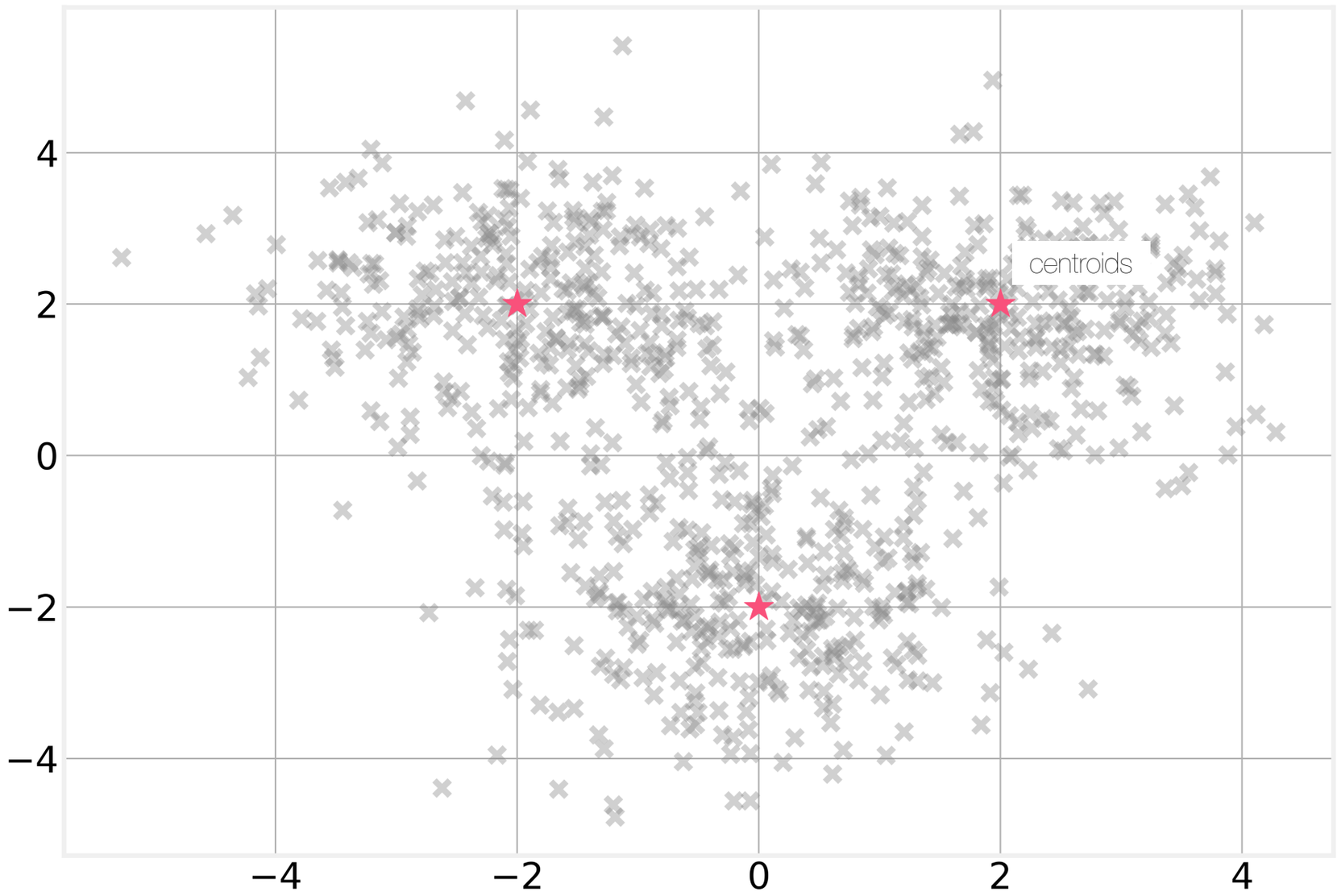
- where μ_i are the coordinates of the centroid of the cluster to which x_i is assigned.

- Smaller values are better

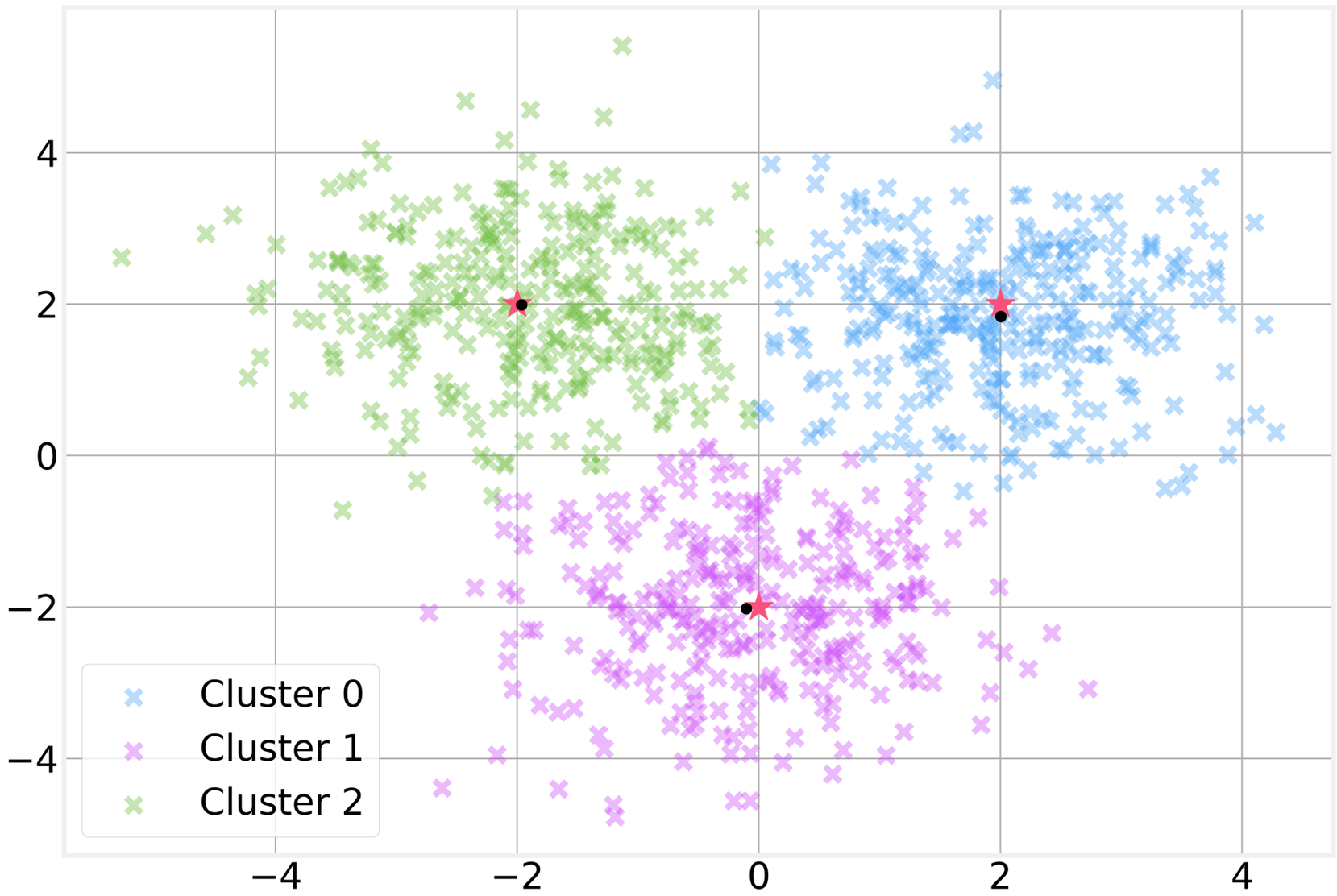
- Can stop when the relative variation is smaller than some value

$$\frac{I_{n+1} - I_n}{I_n} < \epsilon$$

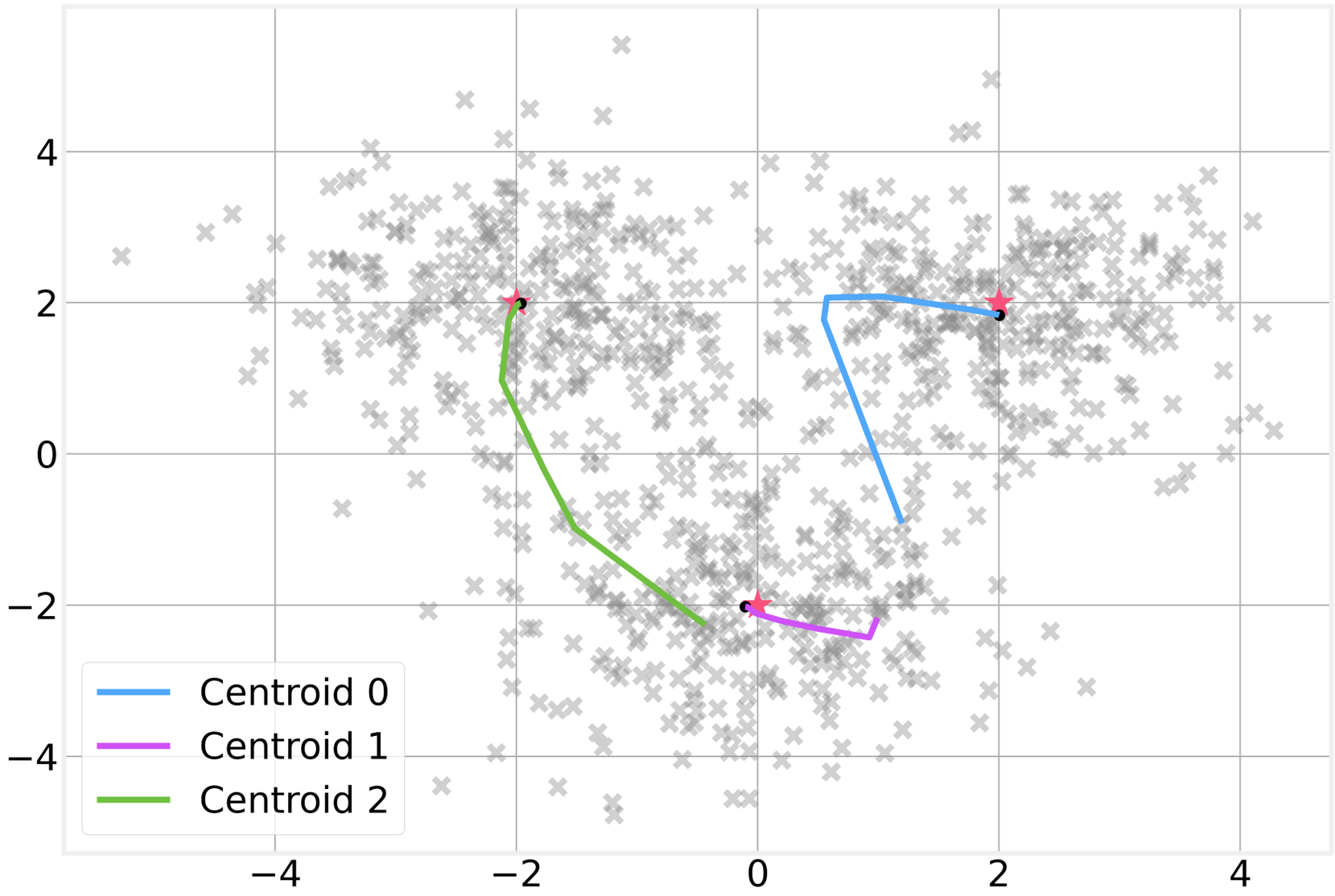
K-Means



K-Means

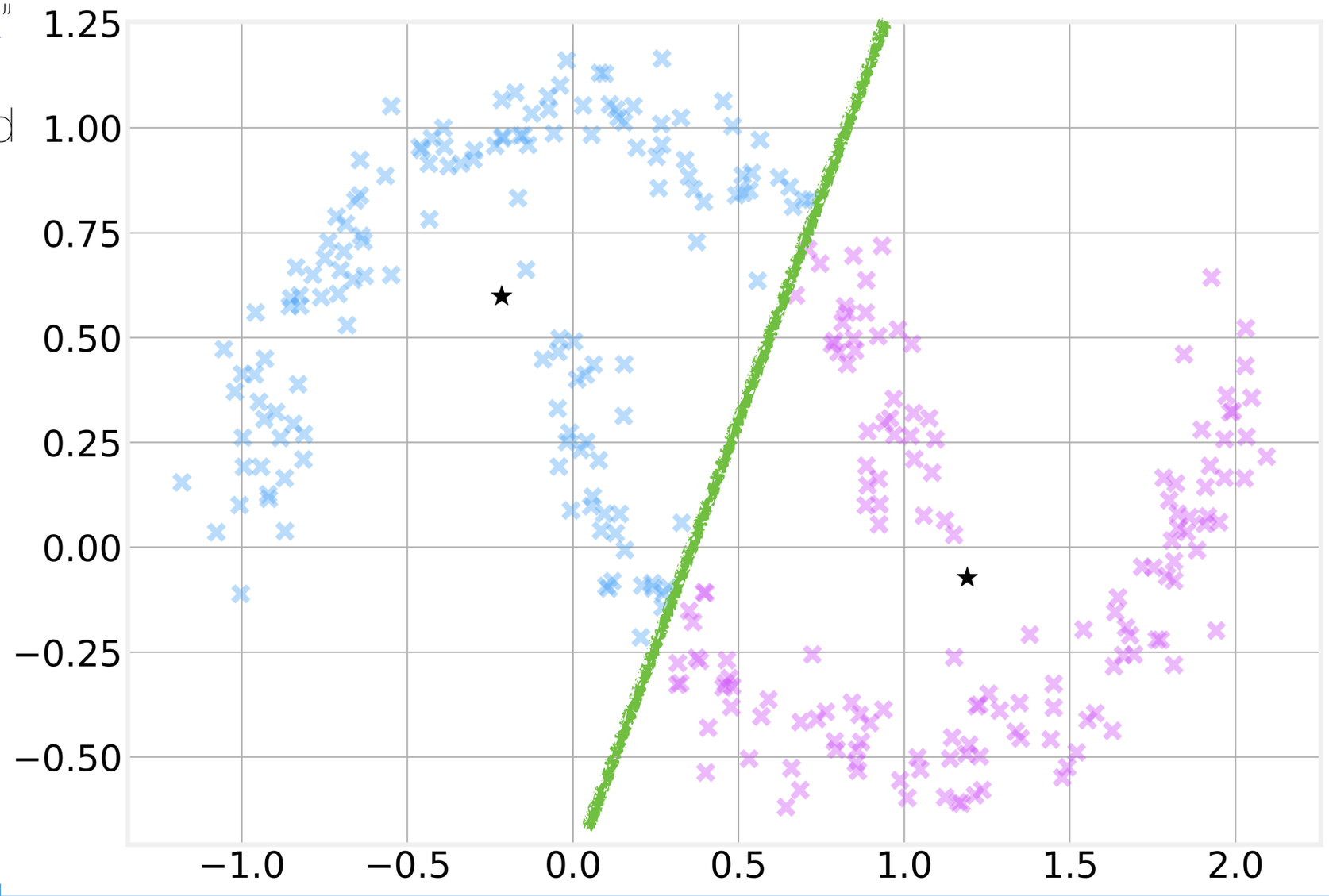


K-Means



K-Means: Limitations

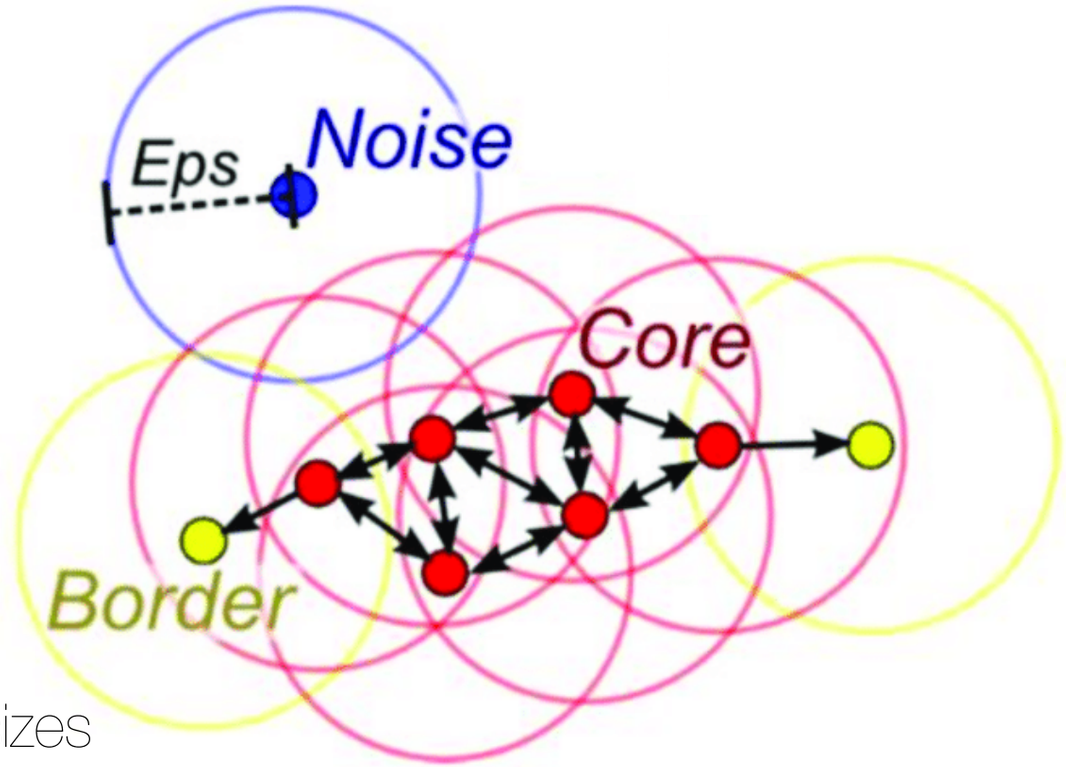
- No guarantees about Finding the "Best" solution
- Each run will find different solution
- No clear way to determine "k"
- Clusters must be similar sized
- Boundaries are always linear



DBSCAN

KDD'96, 226 (1996)

- DBSCAN is a clustering algorithm that overcomes some of the limitations of k-means
- DBSCAN groups points based on their density: points that are close (less than ϵ distance apart) from each other get grouped into the same cluster.
- Three types of points:
  - Core points: A point with at least **MinPts** neighbors within ϵ distance.
  - Border points: A point within ϵ distance of a core point but has fewer than **MinPts** neighbors.
  - Noise: Otherwise
- Advantages:
  - No need to specify the number of clusters
  - Naturally handles clusters of different shapes and sizes



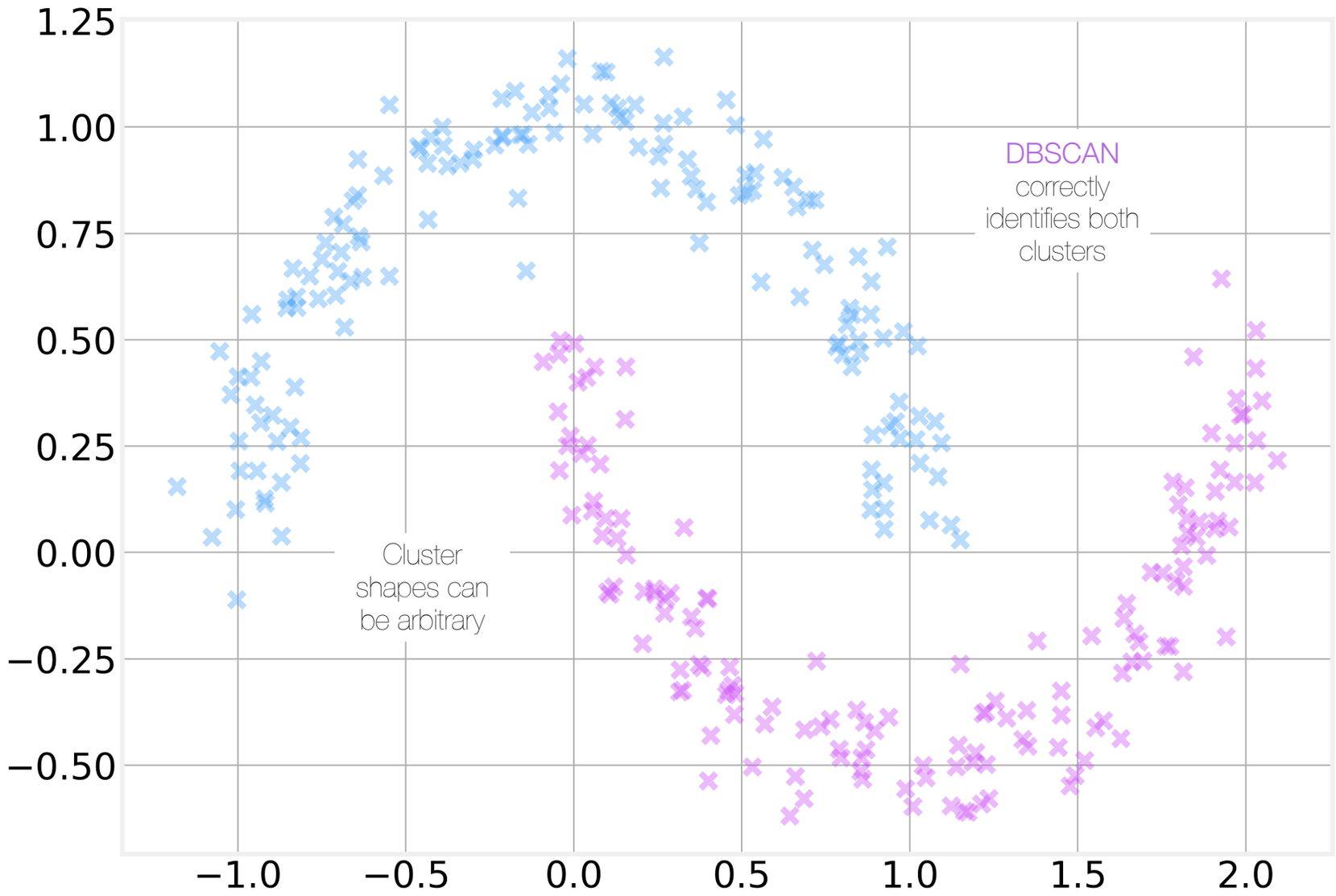
DBSCAN

KDD'96, 226 (1996)

- Algorithm:
  - Select a point at random
  - Find all points within ϵ distance. If there are at least $MinPts$ form a new cluster
  - Expand the cluster by finding all points within ϵ distance.
    - Repeat until all points are either part of a cluster or "noise"
- Points within a cluster are labeled using an integer ≥ 0, while outliers (or noise points) are marked with -1

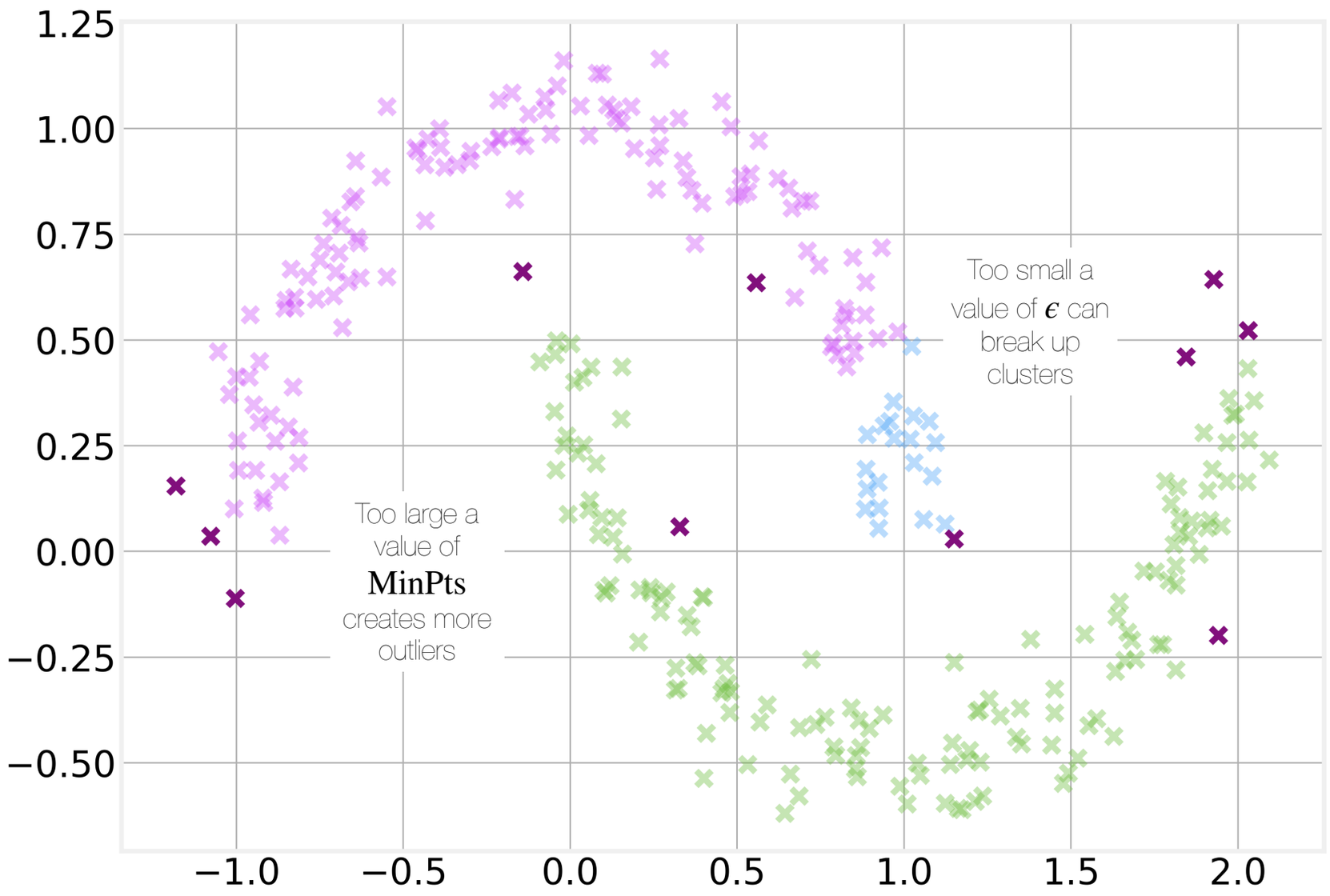
DBSCAN

KDD'96, 226 (1996)



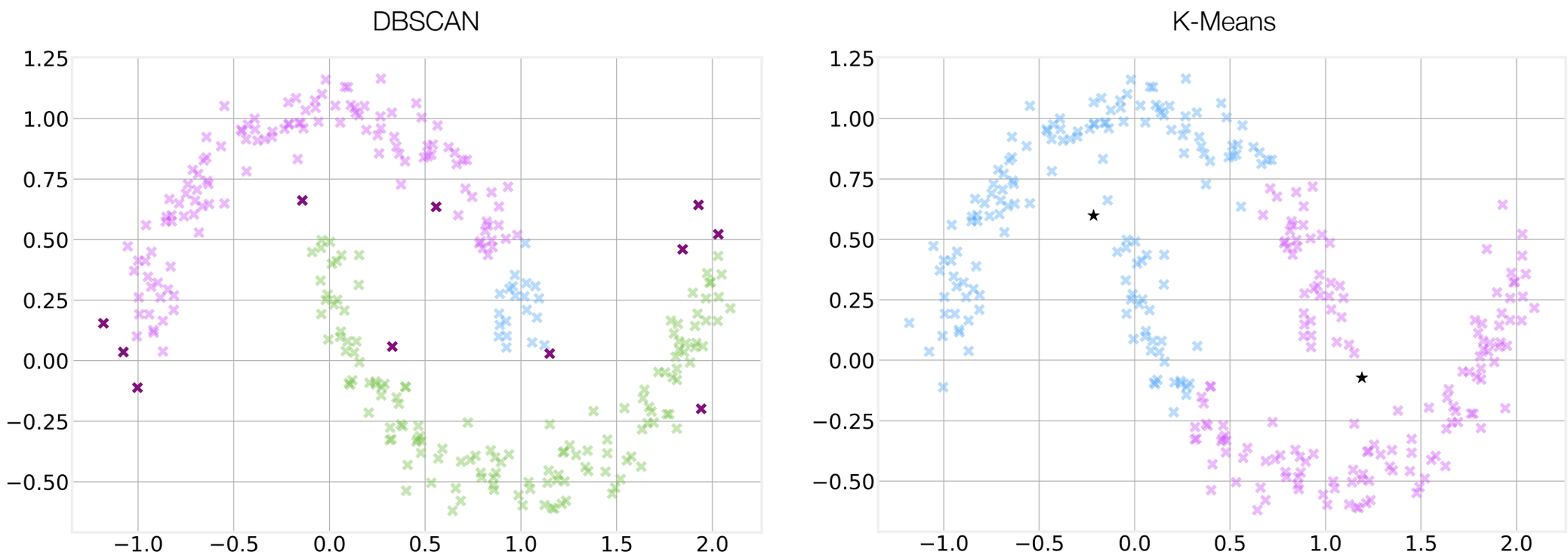
DBSCAN

KDD'96, 226 (1996)



DBSCAN

KDD'96, 226 (1996)



Feature	DBSCAN	K-Means
Requires K?	❌ No	✅ Yes
Handles noise?	✅ Yes	❌ No
Cluster shapes	✅ Any shape	❌ Round clusters only
Works with large datasets?	⚠️ Slower than K-Means for large datasets	✅ Fast for large datasets

Unsupervised Learning

http://github.com/DataForScience/PyTorch/

3. Supervised Learning

Use Cases

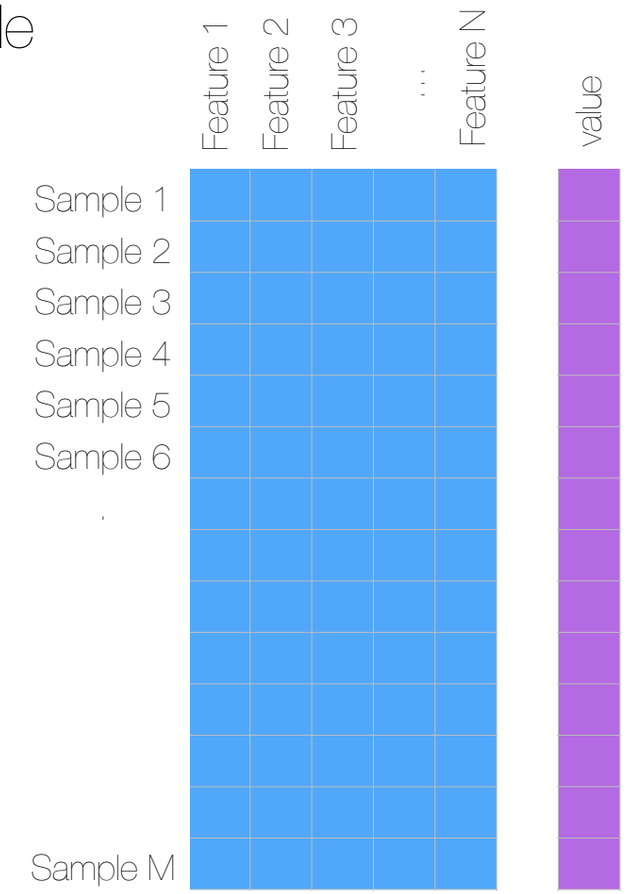
- Regression
- Classification (Binary and Multiclass)
- Object Recognition in Images
- Recommender Systems
- Spam Detection
- Sentiment Analysis
- Medical Diagnosis
- etc

Use Cases

- Regression - Linear Regression
- Classification (Binary and Multiclass) - Logistic Regression
- Object Recognition in Images
- Recommender Systems - K-Nearest Neighbors
- Spam Detection
- Sentiment Analysis - Support Vector Machines
- Medical Diagnosis
- etc

Data Preparation

- Supervised Learning requires labeled data: input features come with known output values.
- The goal is to learn a function that maps inputs to outputs so that the model can make predictions on new, unseen data.
- Dataset formatted as an MxN matrix of M samples and N features
- Each sample corresponds to a specific value of the target variable
- Supervised Learning can:
  - Predict the value of a function at previously unseen points
  - Determine the class that an instance belongs to

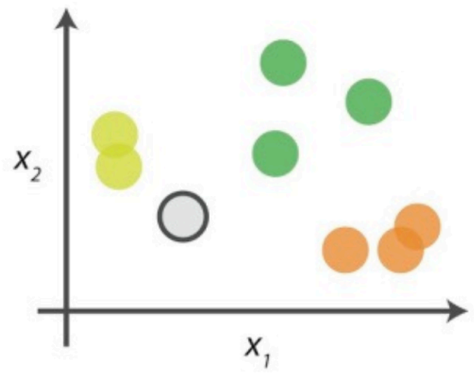


K-Nearest Neighbors

- Perhaps the simplest of supervised learning algorithms
- Effectively memorizes all previously seen data
- Intuitively takes advantage of natural data clustering (density).
- Define that the class of any datapoint is given by the plurality of it's k nearest neighbors
- Many variations using:
    - different distance metrics,
    - weighting procedures,
    - etc…

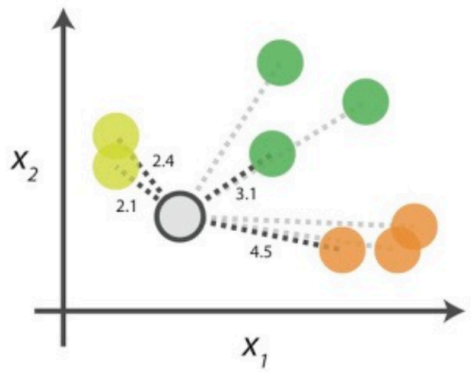
K-Nearest Neighbors

0. Look at the data



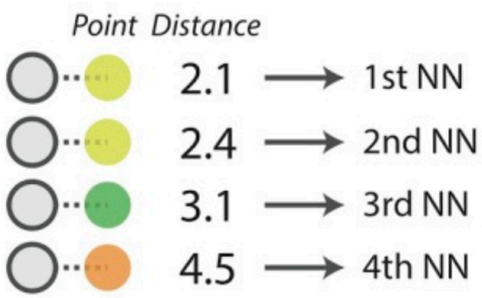
Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

1. Calculate distances



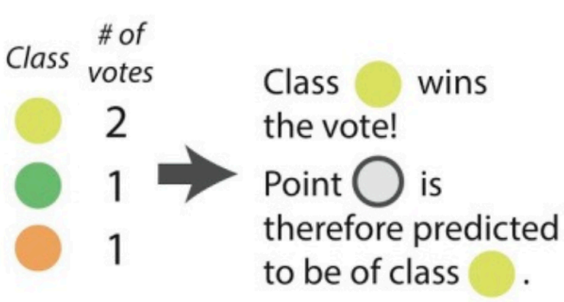
Start by calculating the distances between the grey point and all other points.

2. Find neighbours



Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

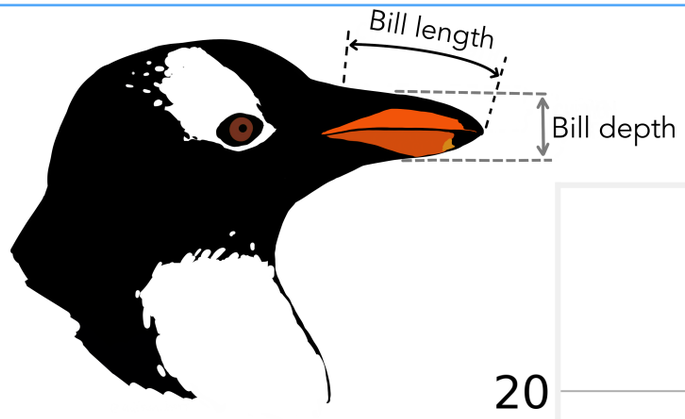
3. Vote on labels

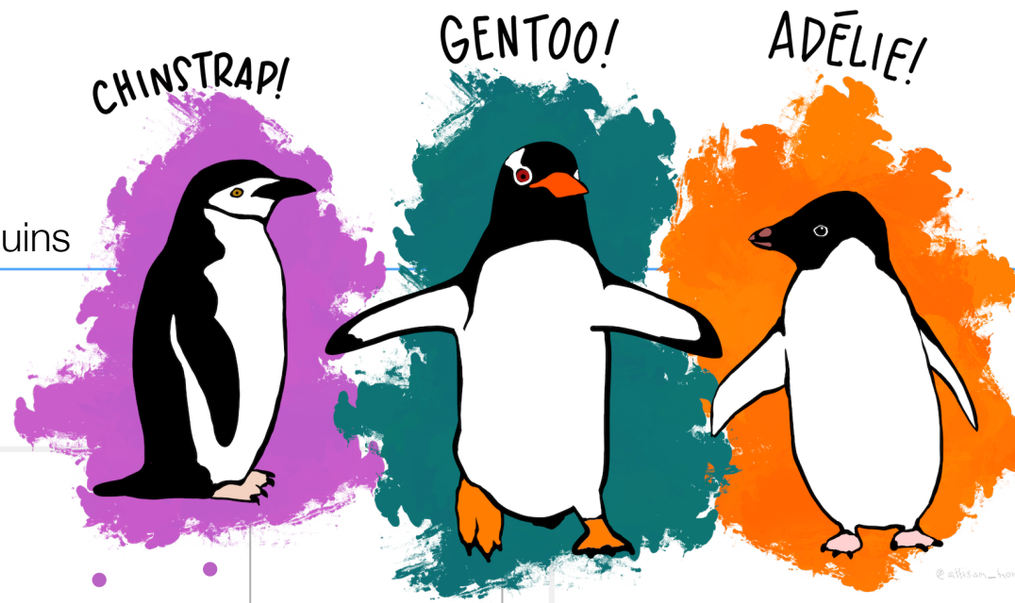


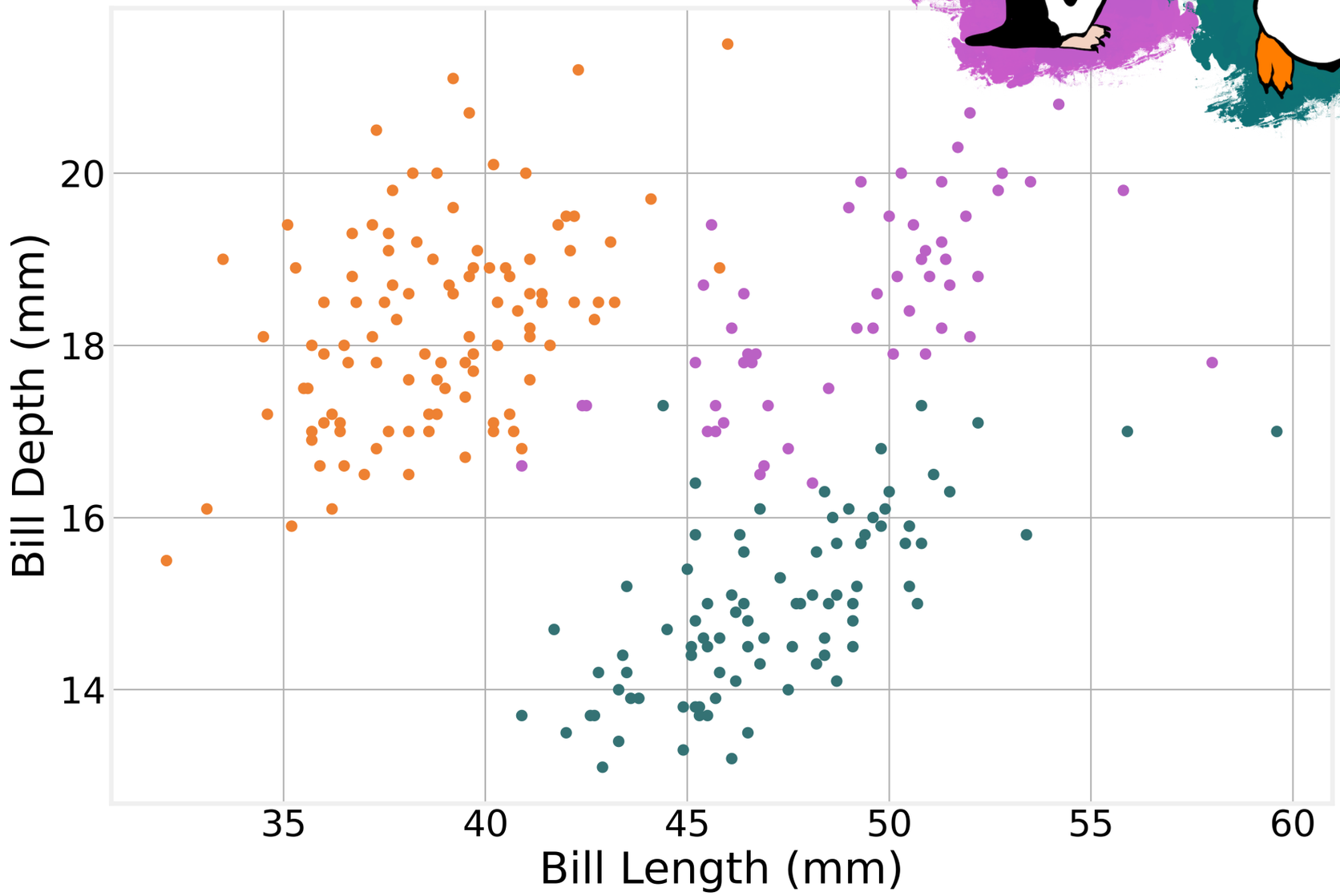
Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

Penguin Dataset

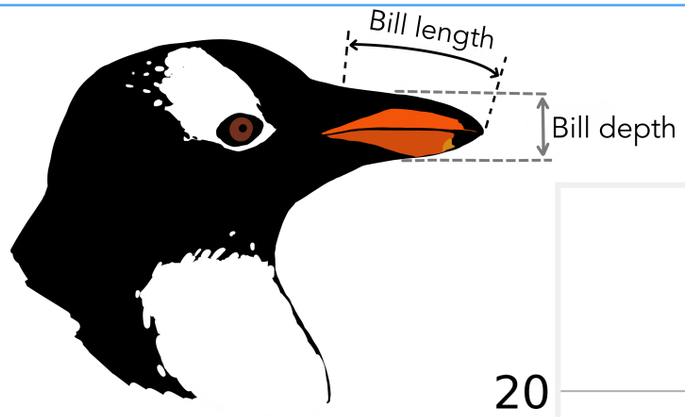
https://github.com/mcnakhaee/palmerpenguins

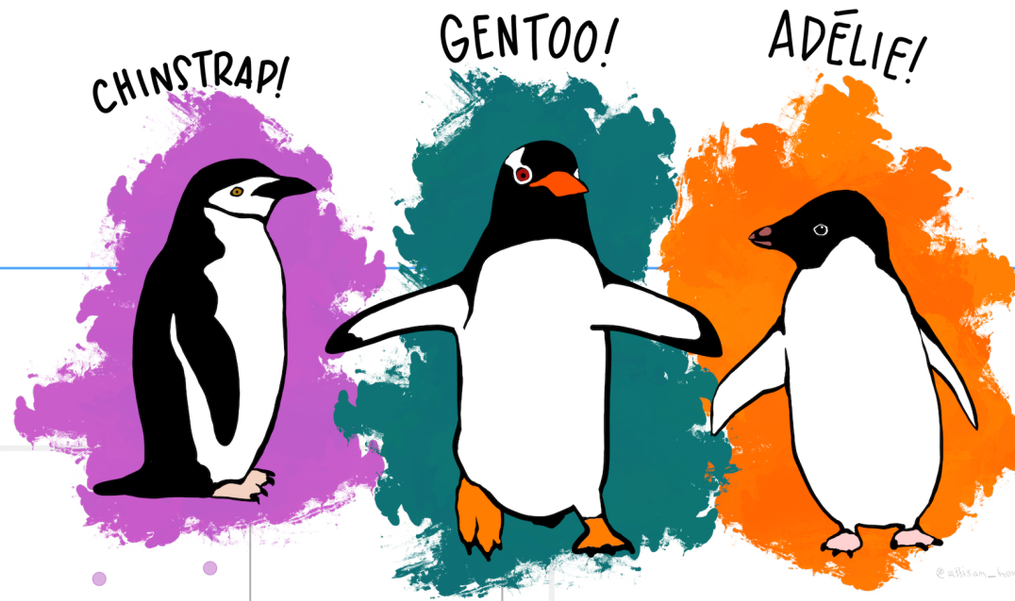


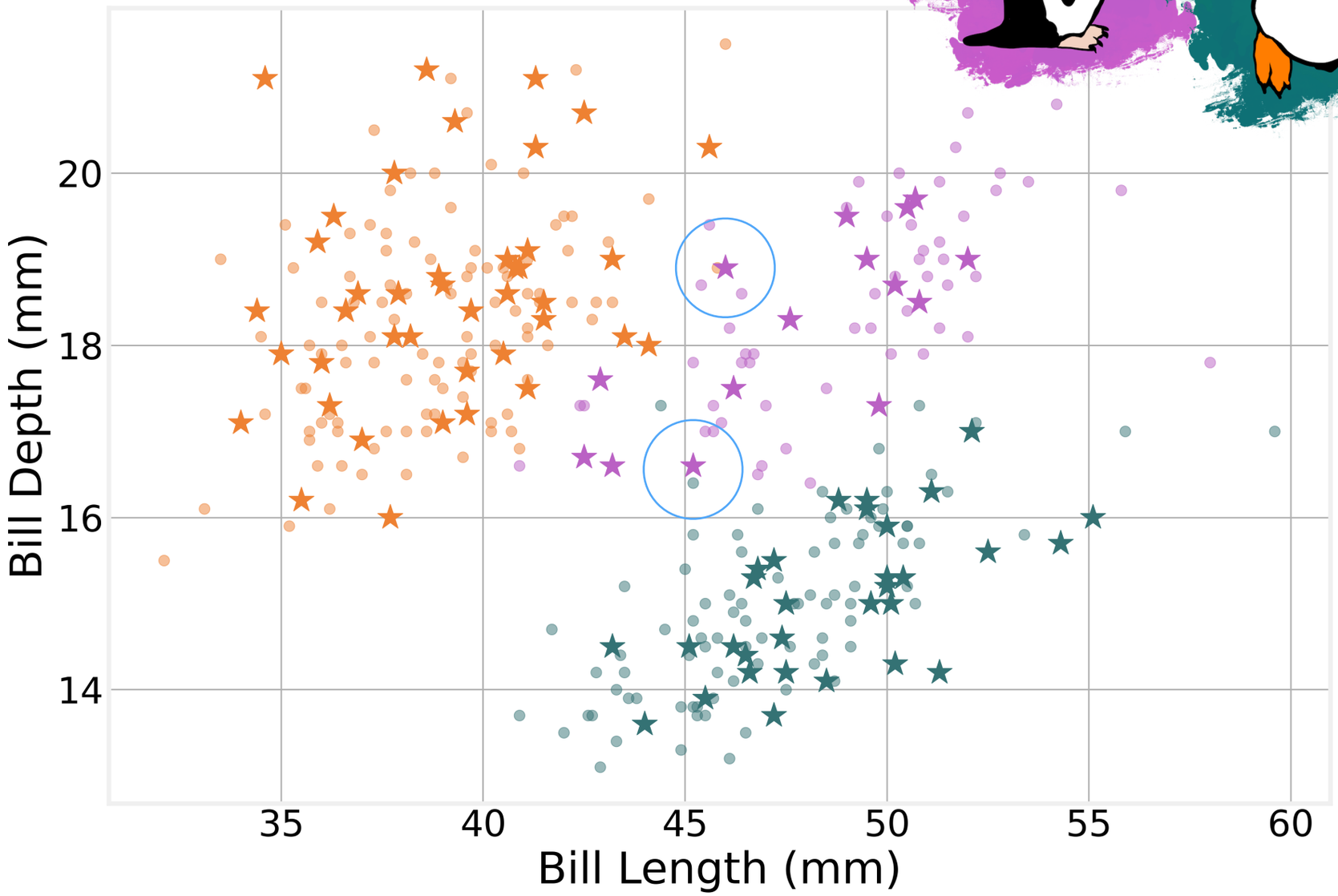




K-Nearest Neighbors







Linear Regression

$$\vec{x_i} = (x_0, x_1, \cdots, x_n)^T$$

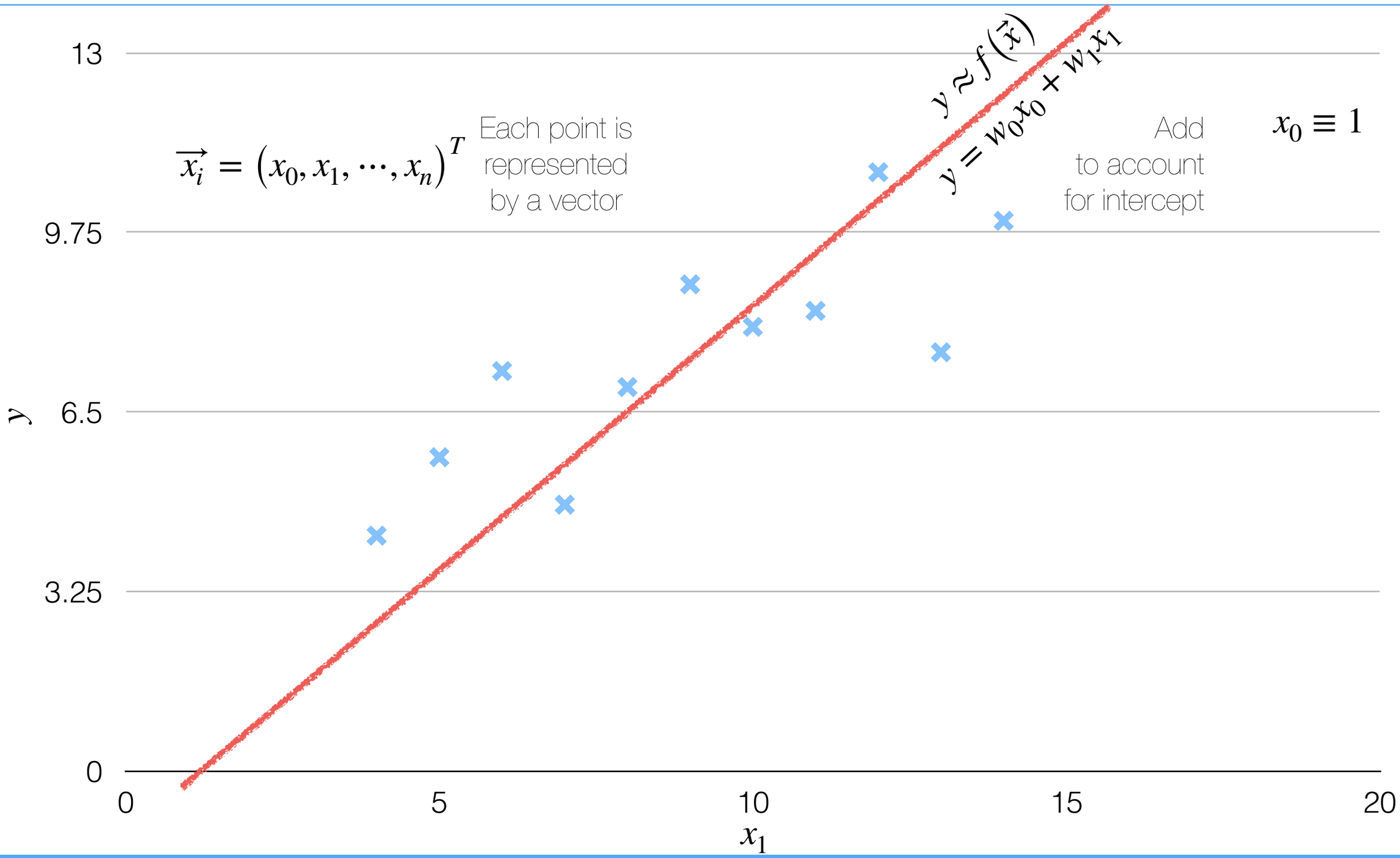
Each point is represented by a vector

$$y \approx f(\vec{x})$$

$$y = w_0 x_0 + w_1 x_1$$

Add to account for intercept

$$x_0 \equiv 1$$



Linear Regression

- We are assuming that our functional dependence is of the form:

$$f\left(\vec{x}\right) = w_0 + w_1 x_1 + \cdots + w_n x_n \equiv X\overrightarrow{w}$$

- In other words, at each step, our hypothesis is:

$$h_w\left(X\right) = X\overrightarrow{w} \equiv \hat{y}$$

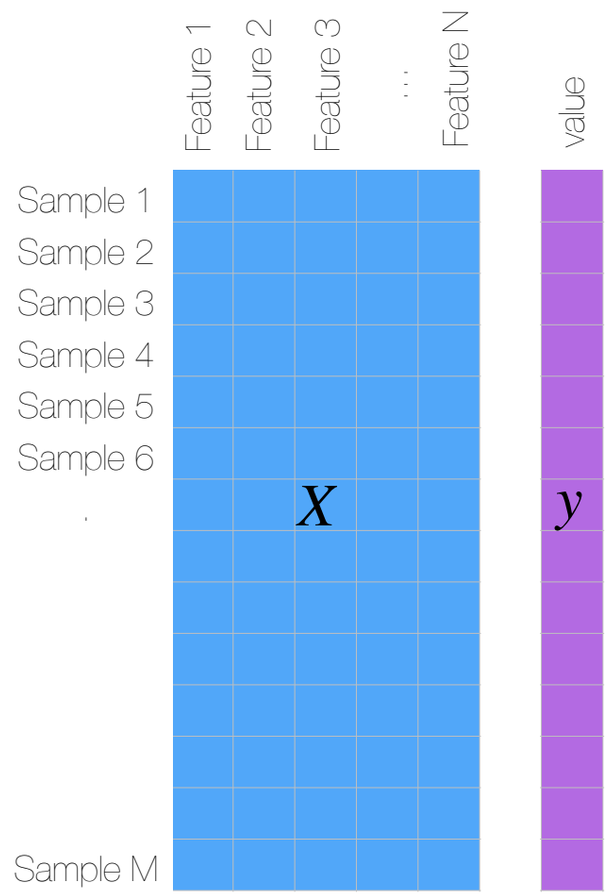
and it imposes a Constraint on the solutions that can be found.

- We quantify our far our hypothesis is from the correct value using an Error/Loss Function:

$$J_w\left(X,\vec{y}\right) = \frac{1}{2m}\sum_i \left[h_w\left(x^{(i)}\right) - y^{(i)}\right]^2$$

or, vectorially:

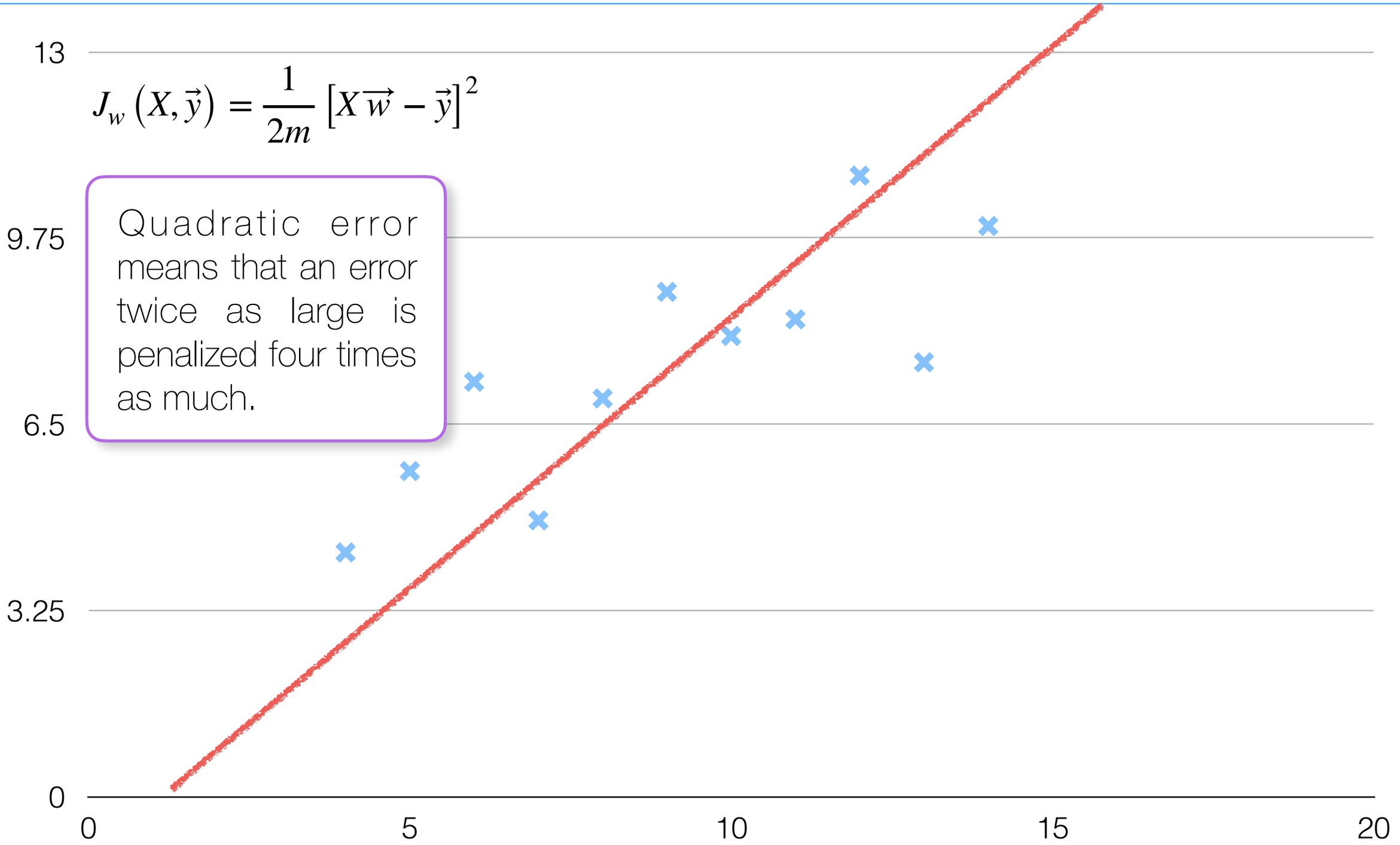
$$J_w\left(X,\vec{y}\right) = \frac{1}{2m}\left[X\overrightarrow{w} - \vec{y}\right]^2$$



Geometric Interpretation

$$J_w\left(X,\vec{y}\right)=\frac{1}{2m}\left[X\vec{w}-\vec{y}\right]^2$$

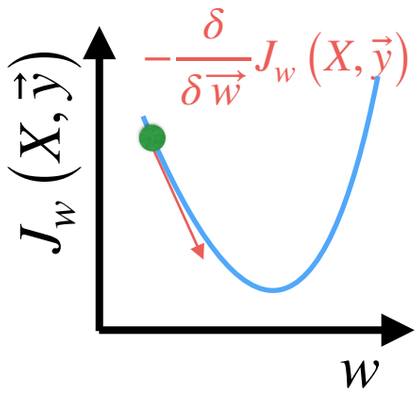
Quadratic error means that an error twice as large is penalized four times as much.

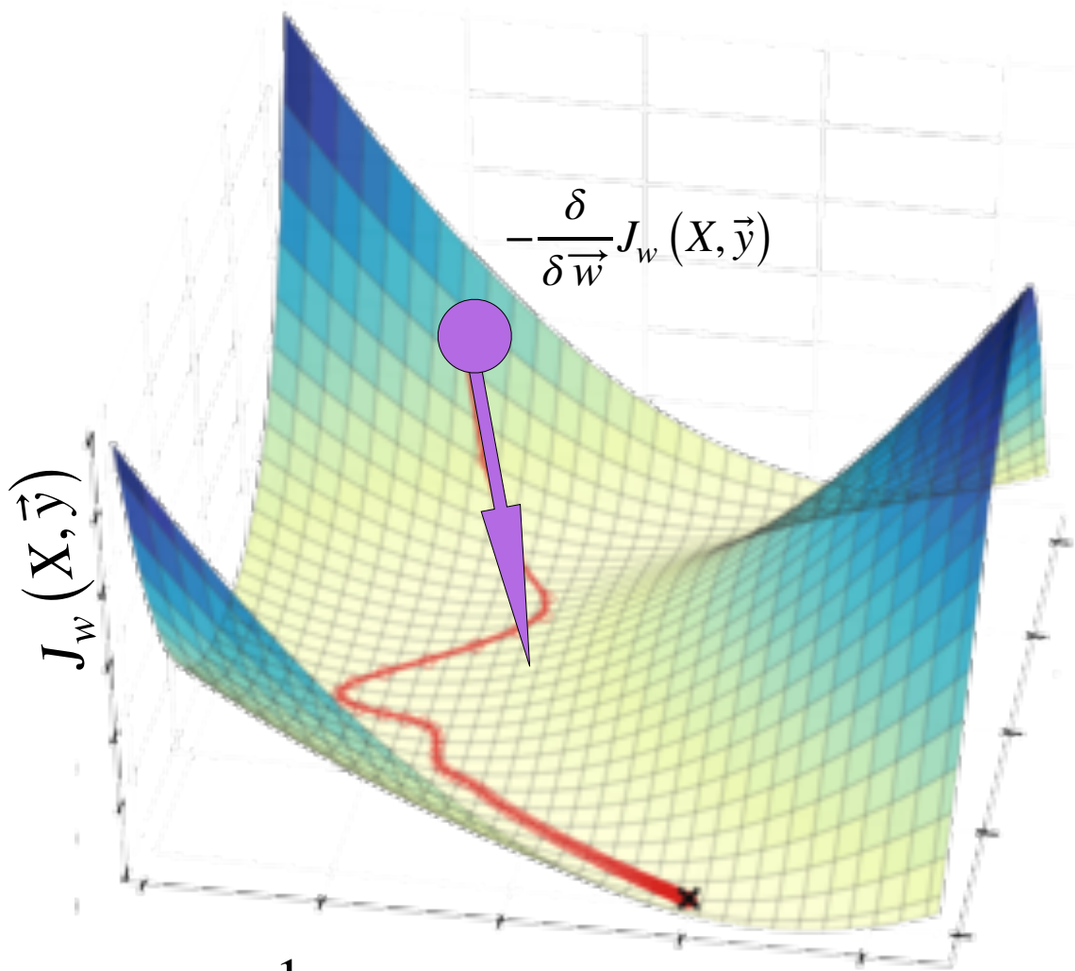


Gradient Descent

- Goal: Find the minimum of $J_w\left(X,\vec{y}\right)$ by varying the components of $\vec{w}$

- Intuition: Follow the slope of the error function until convergence





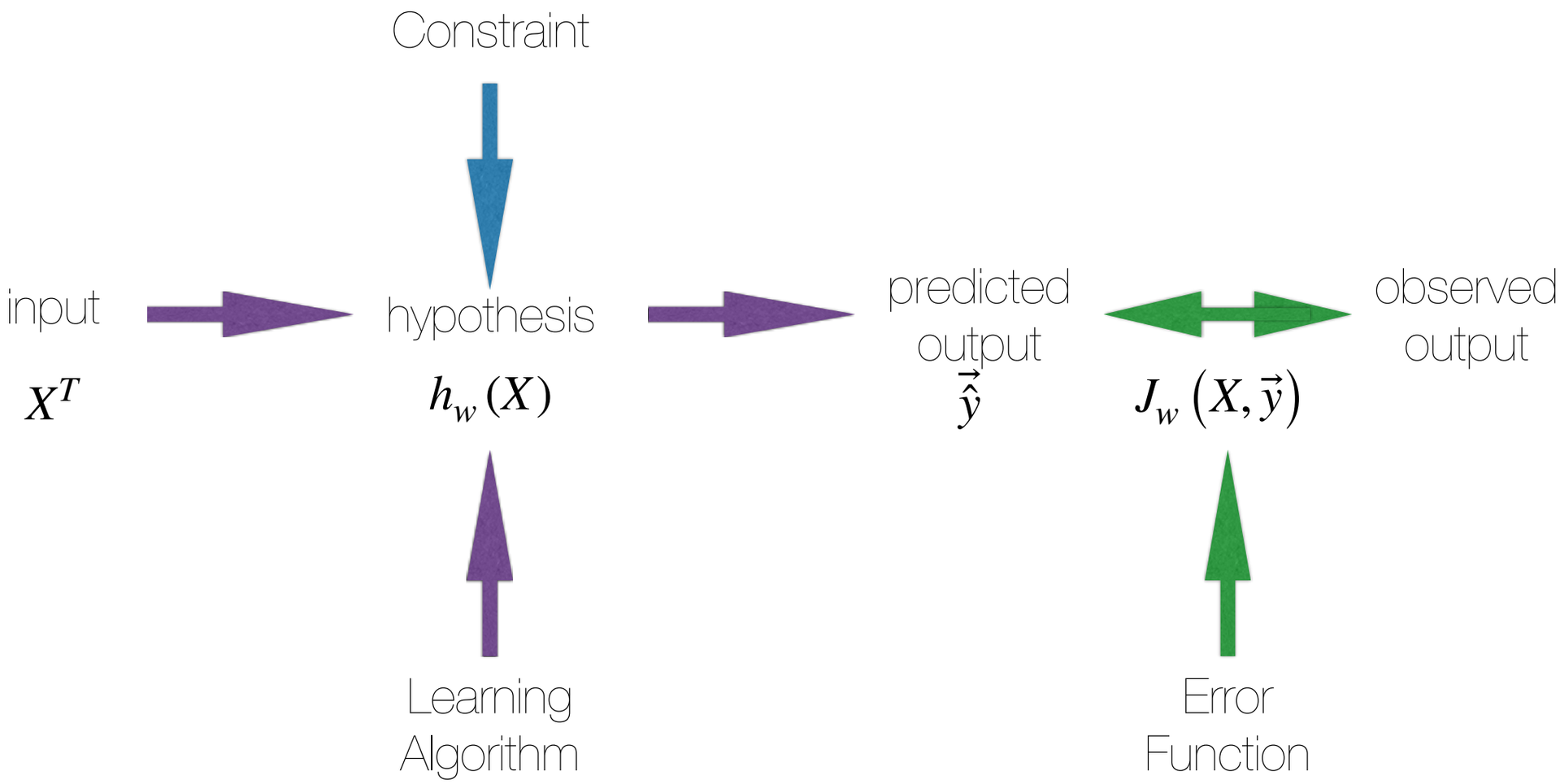
- Algorithm:

  - Guess $\vec{w}^{(0)}$ (initial values of the parameters)
  - Update until "convergence":

step size

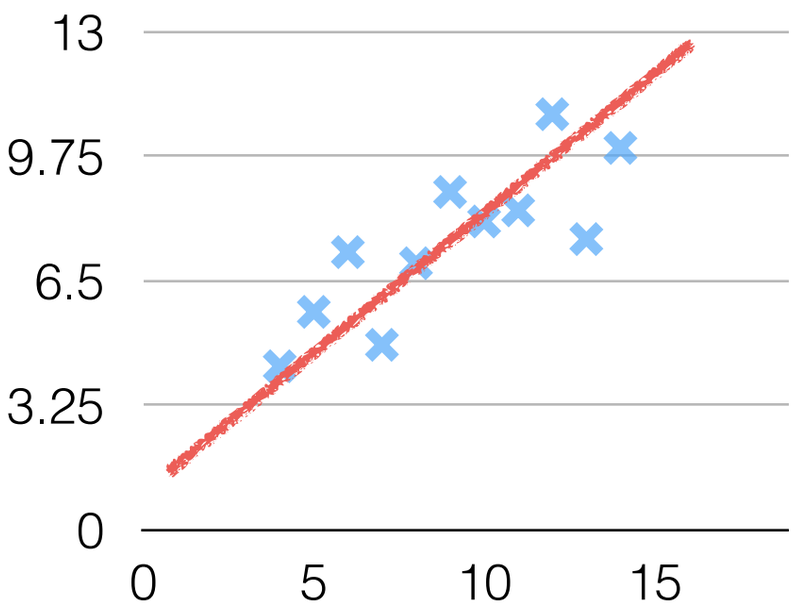
$$w_j = w_j - \alpha \frac{\delta}{\delta w_j} J_w\left(X,\vec{y}\right) \qquad \frac{\delta}{\delta w_j} J_w\left(X,\vec{y}\right) = \frac{1}{m} X^T \cdot \left(h_w(X) - \vec{y}\right)$$

Learning Procedure



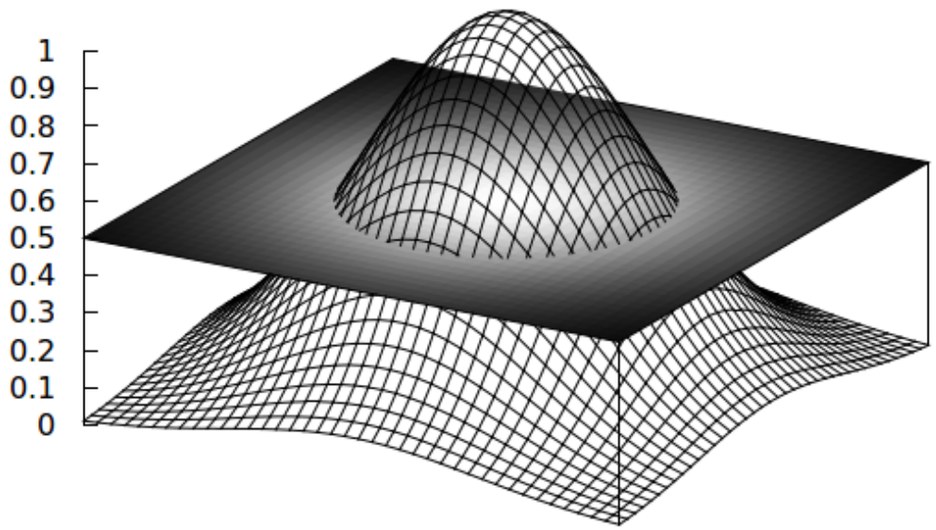
Geometric Interpretation

2D



$$y = w_0 + w_1 x_1$$

3D



$$y = w_0 + w_1 x_1 + w_2 x_2$$

nD

$$y = X\vec{w}$$

Add to account for intercept

$$x_0 \equiv 1$$

Finds the hyperplane that splits the points in two such that the errors on each side balance out

Logistic Regression (Classification)

- Not actually Regression, but rather Classification

- Predict the probability of instance belonging to the given class:

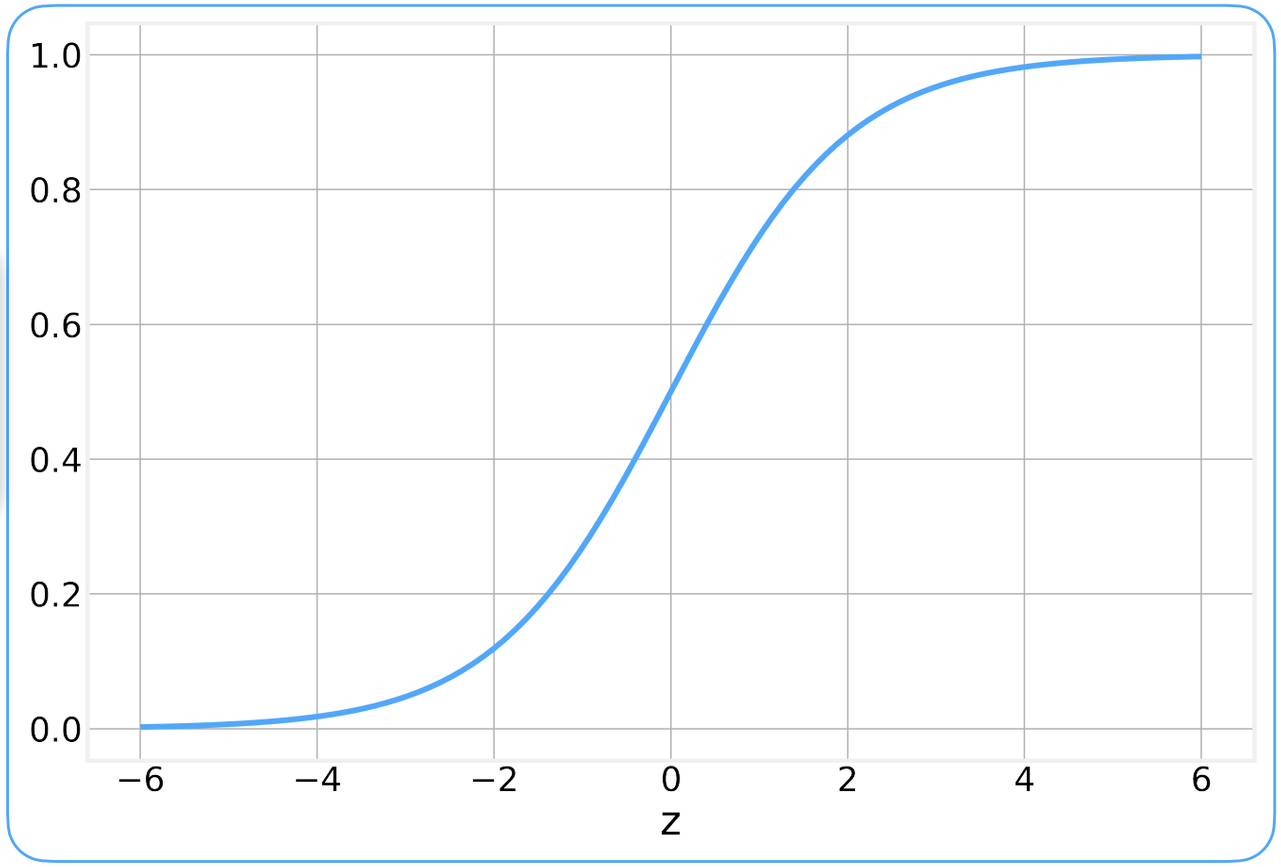
$$h_w(X) \in [0,1]$$

- Use Sigmoid/Logistic function to map weighted inputs to $[0,1]$

$$h_w(X) = \phi\left(X\vec{w}\right)$$

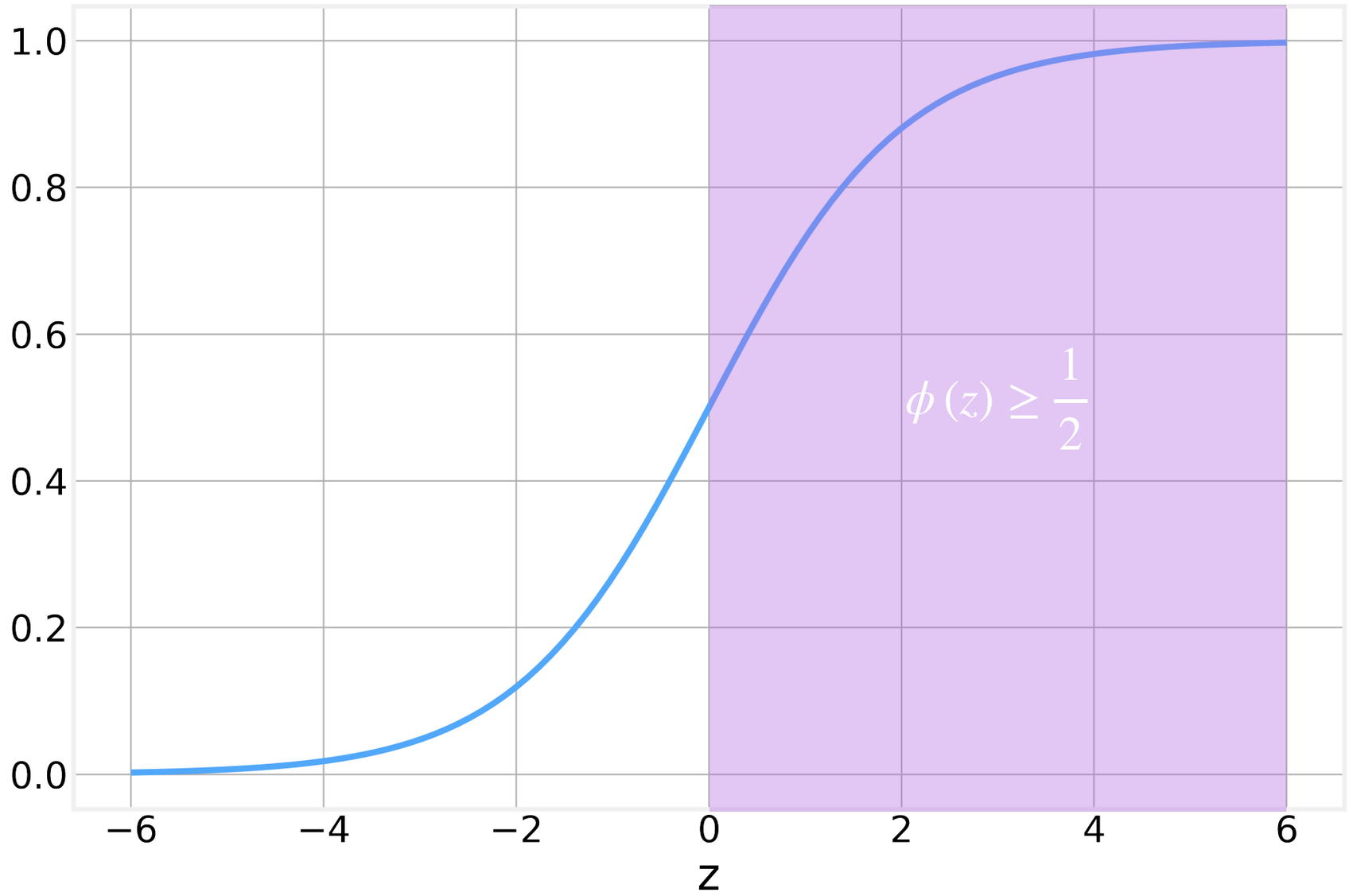
1 - part of the class
0 - otherwise

z encapsulates all the parameters and input values

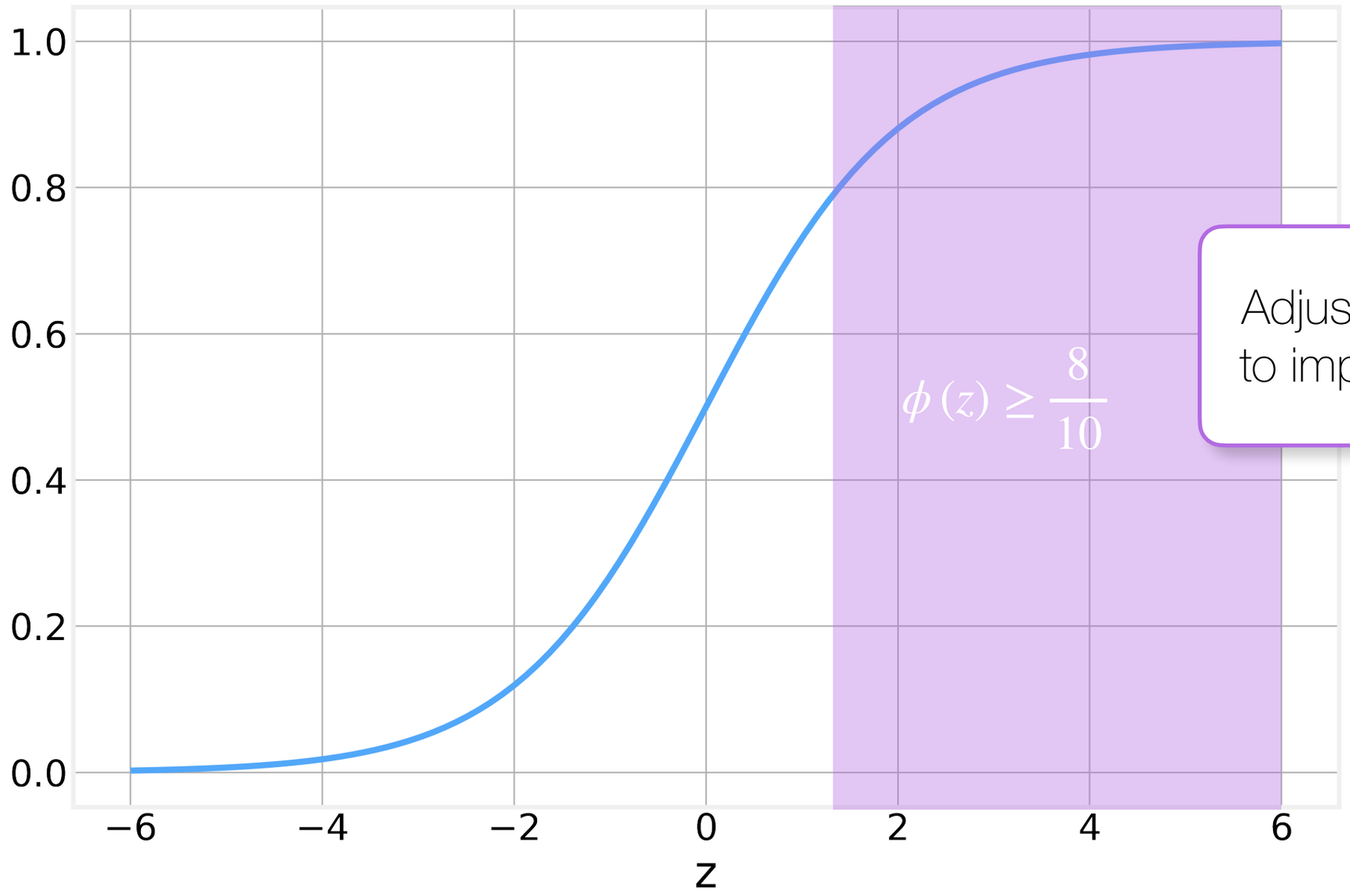


Geometric Interpretation

maximize the value of z for members of the class



Geometric Interpretation



Logistic Regression

- Error Function - Cross Entropy

$$J_w\left(X,\vec{y}\right) = -\frac{1}{m}\left[y^T \log\left(h_w\left(X\right)\right) + \left(1-y\right)^T \log\left(1-h_w\left(X\right)\right)\right]$$

measures the "distance" between two probability distributions

$$h_w\left(X\right) = \frac{1}{1+e^{-X\vec{w}}}$$

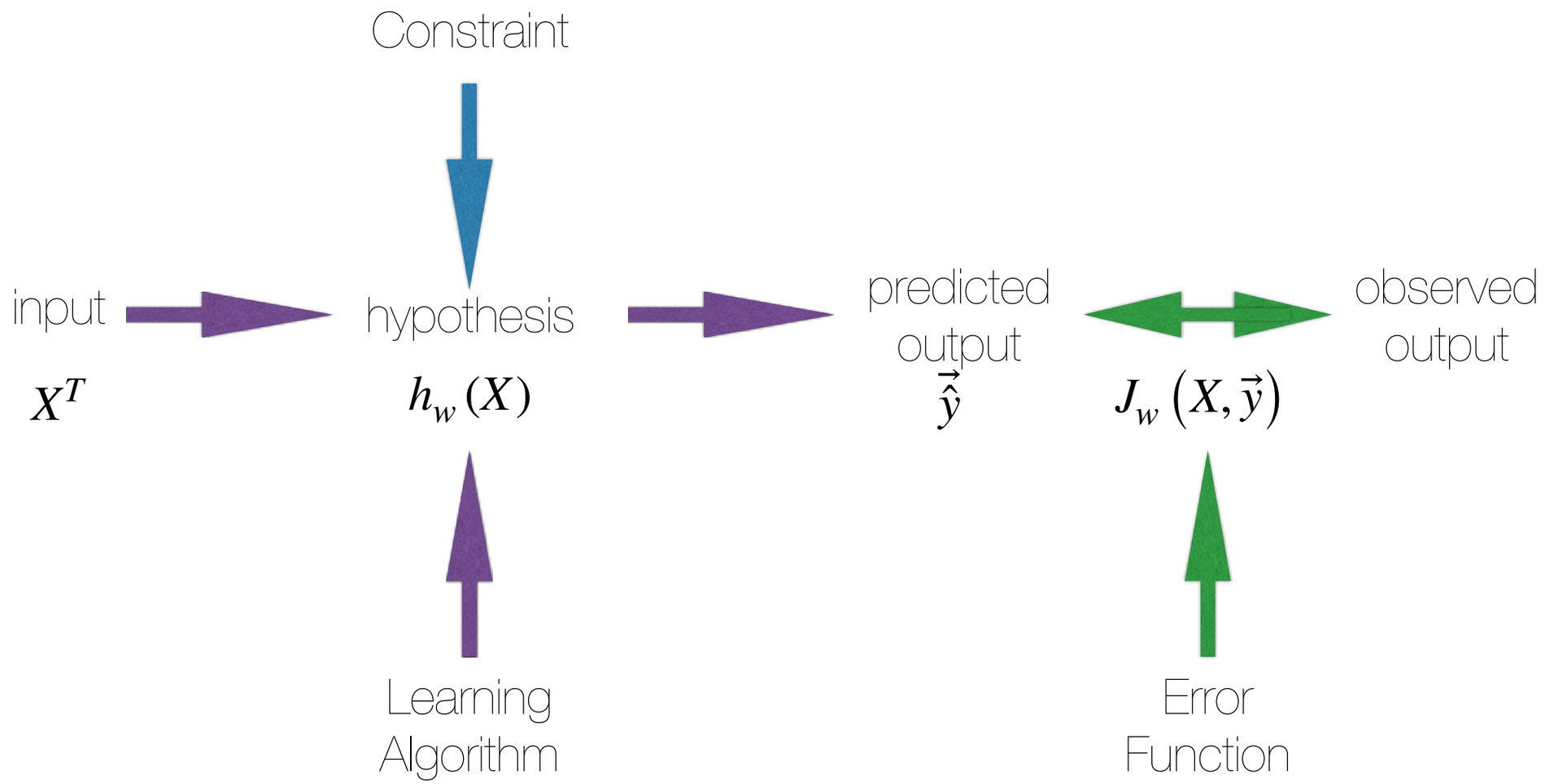
- Effectively treating the labels as probabilities (an instance with label=1 has Probability 1 of belonging to the class).

- Gradient - same as Logistic Regression

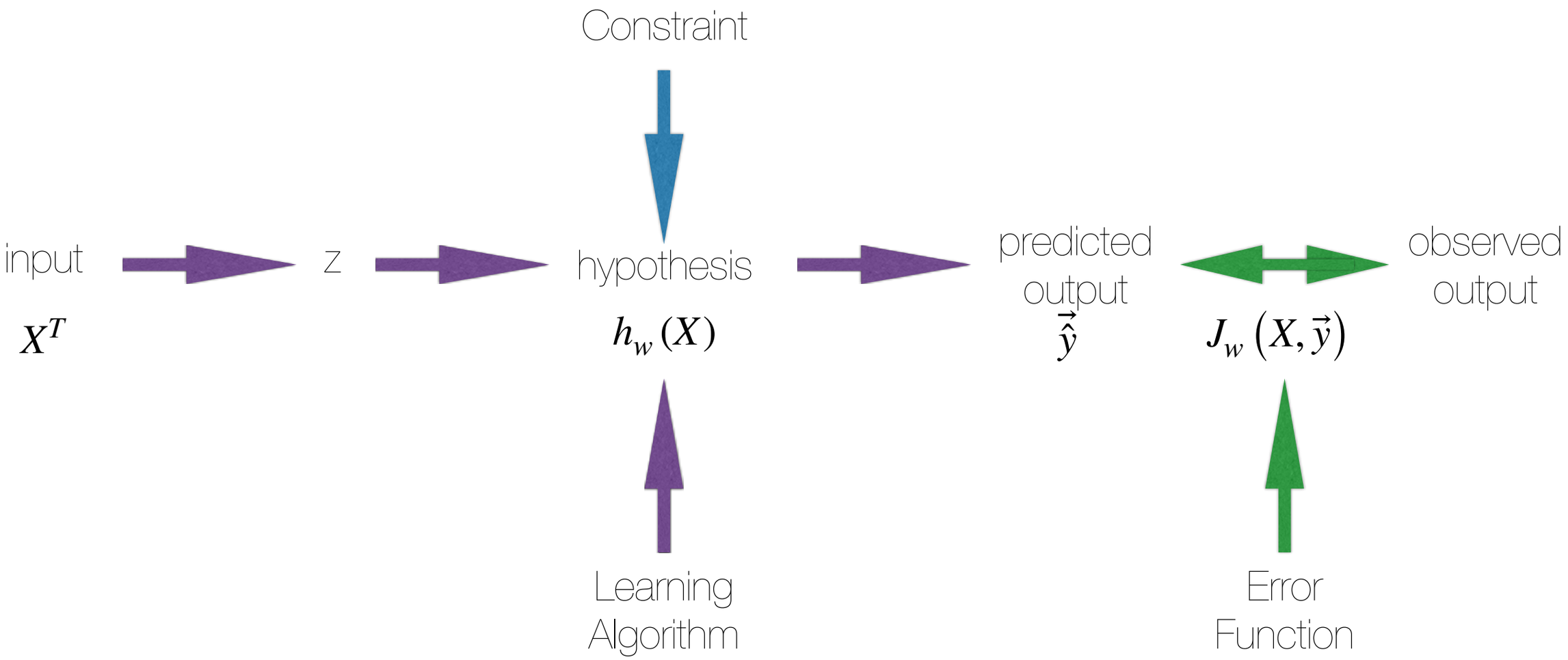
$$w_j = w_j - \alpha\frac{\delta}{\delta w_j}J_w\left(X,\vec{y}\right)$$

$$\frac{\delta}{\delta w_j}J_w\left(X,\vec{y}\right) = \frac{1}{m}X^T\cdot\left(h_w\left(X\right)-\vec{y}\right)$$

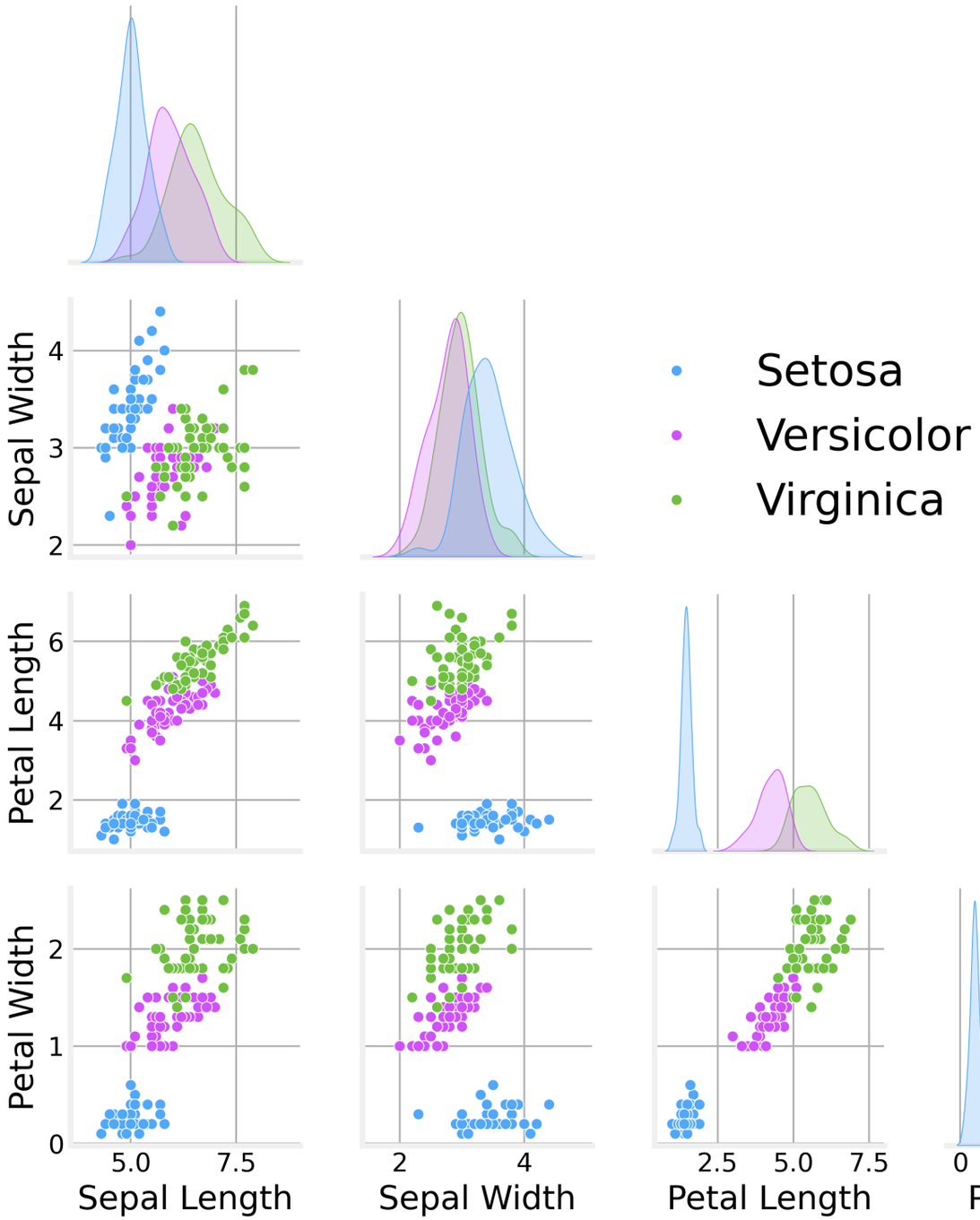
Learning Procedure



Learning Procedure



Iris dataset

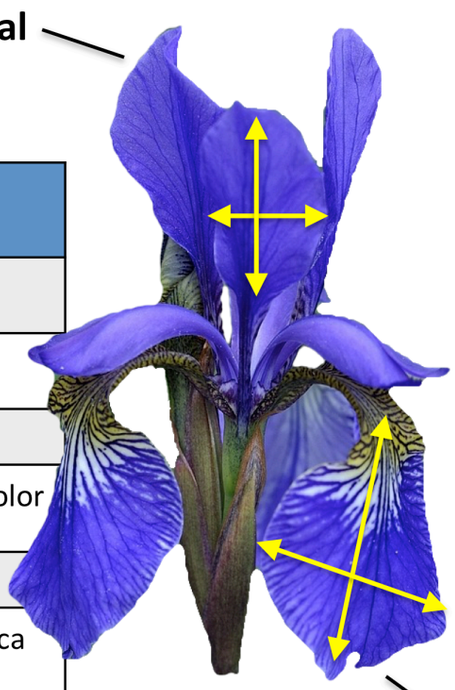


Samples
(instances, observations)

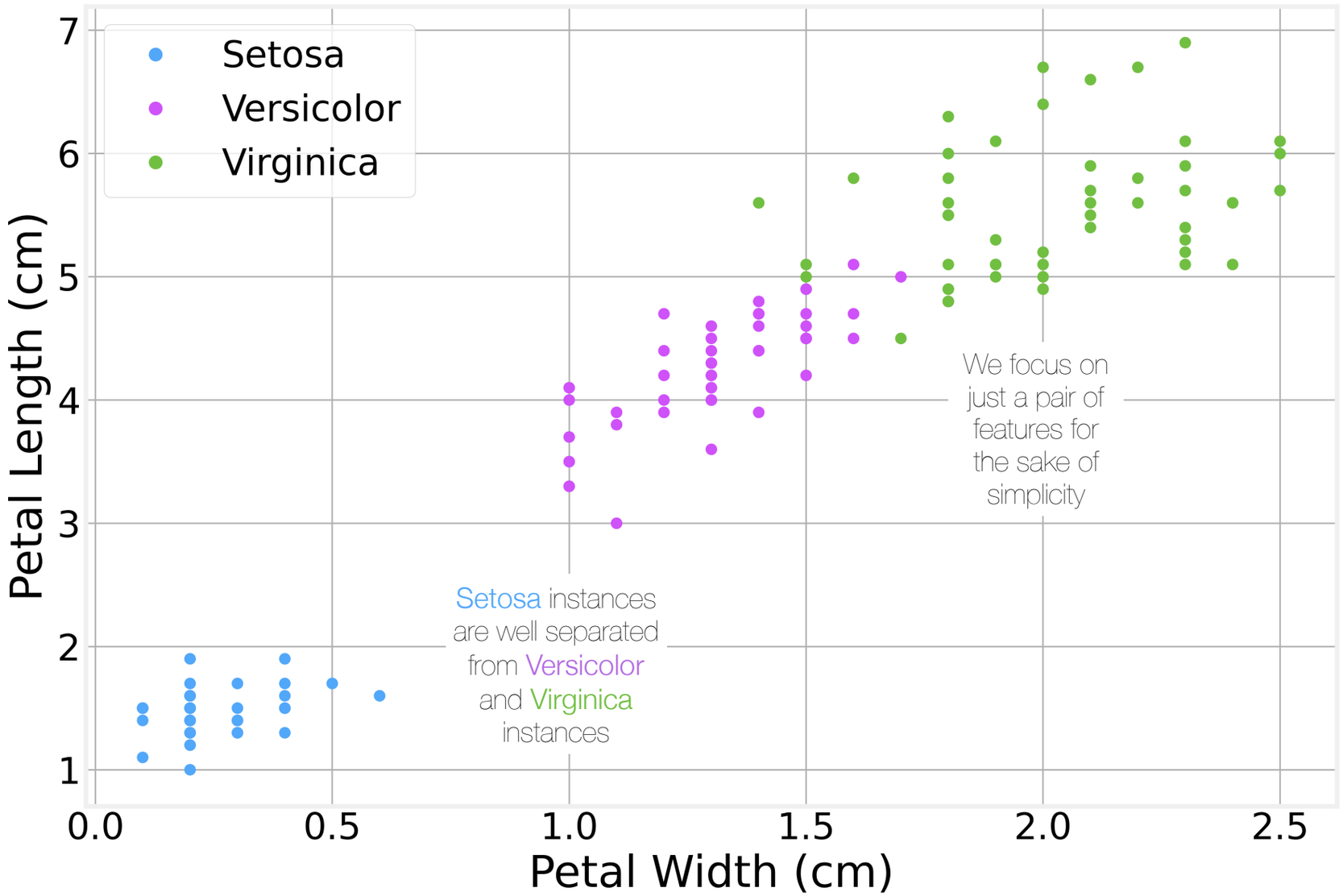
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Features
(attributes, measurements, dimensions)

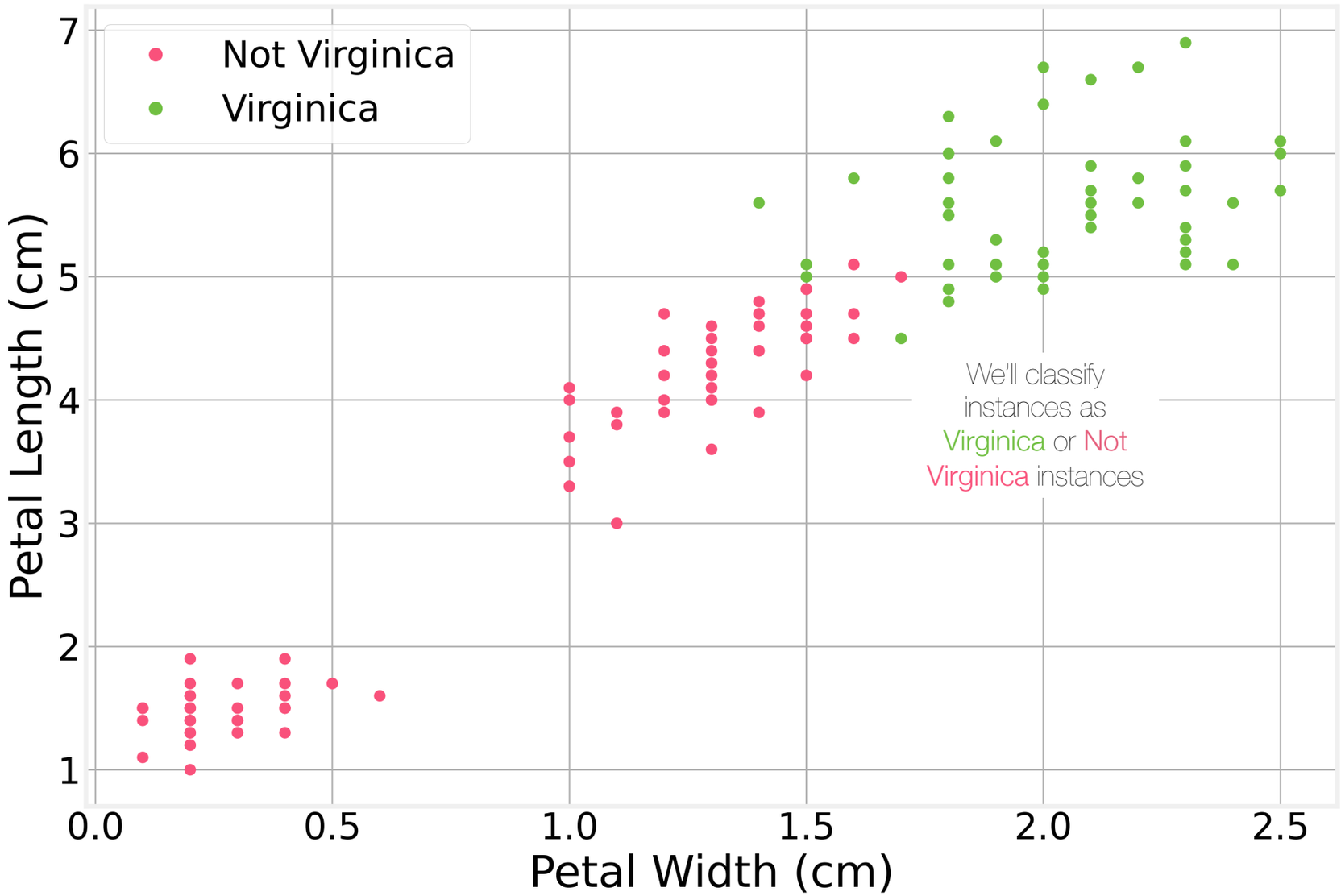
Class labels
(targets)



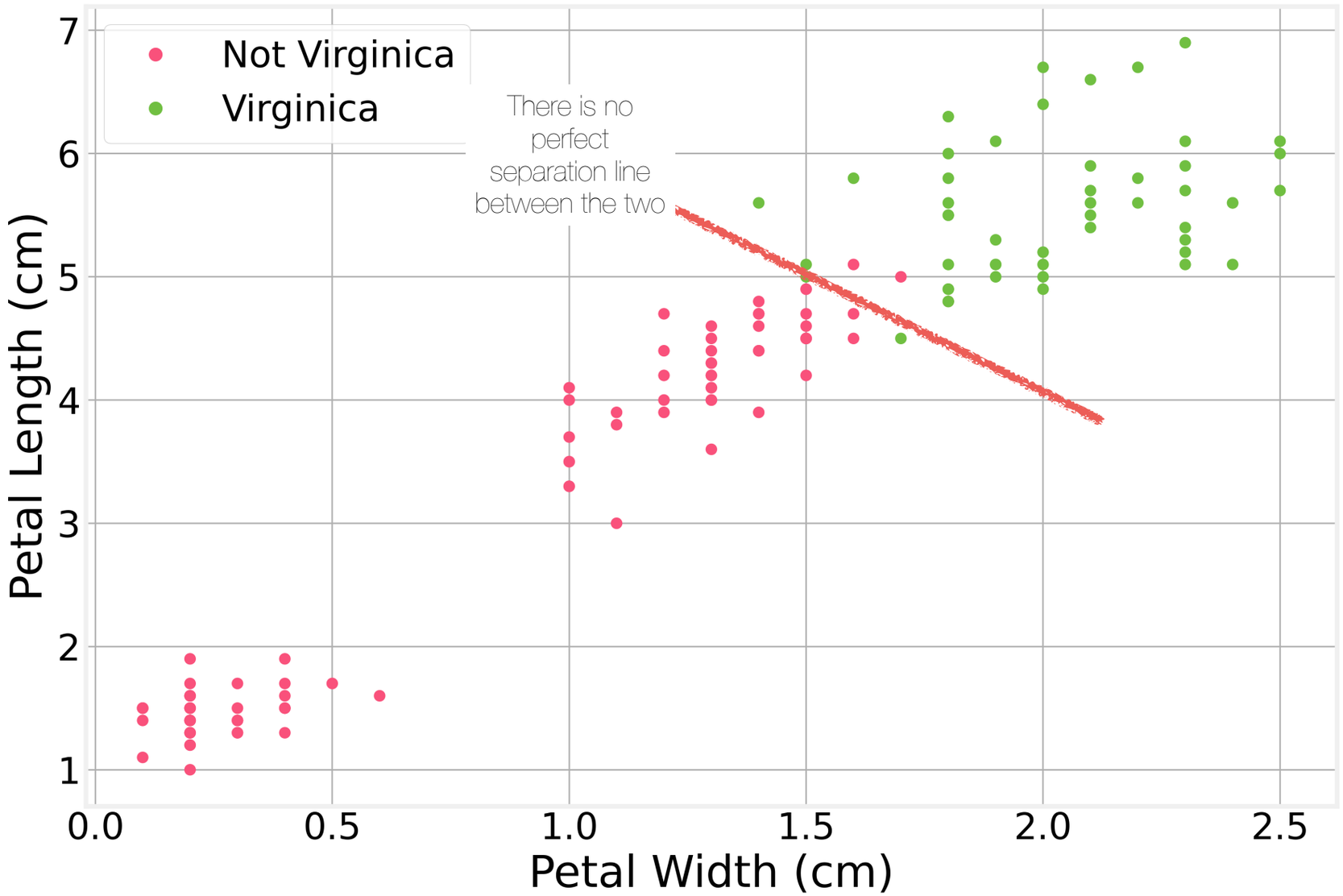
Logistic Regression



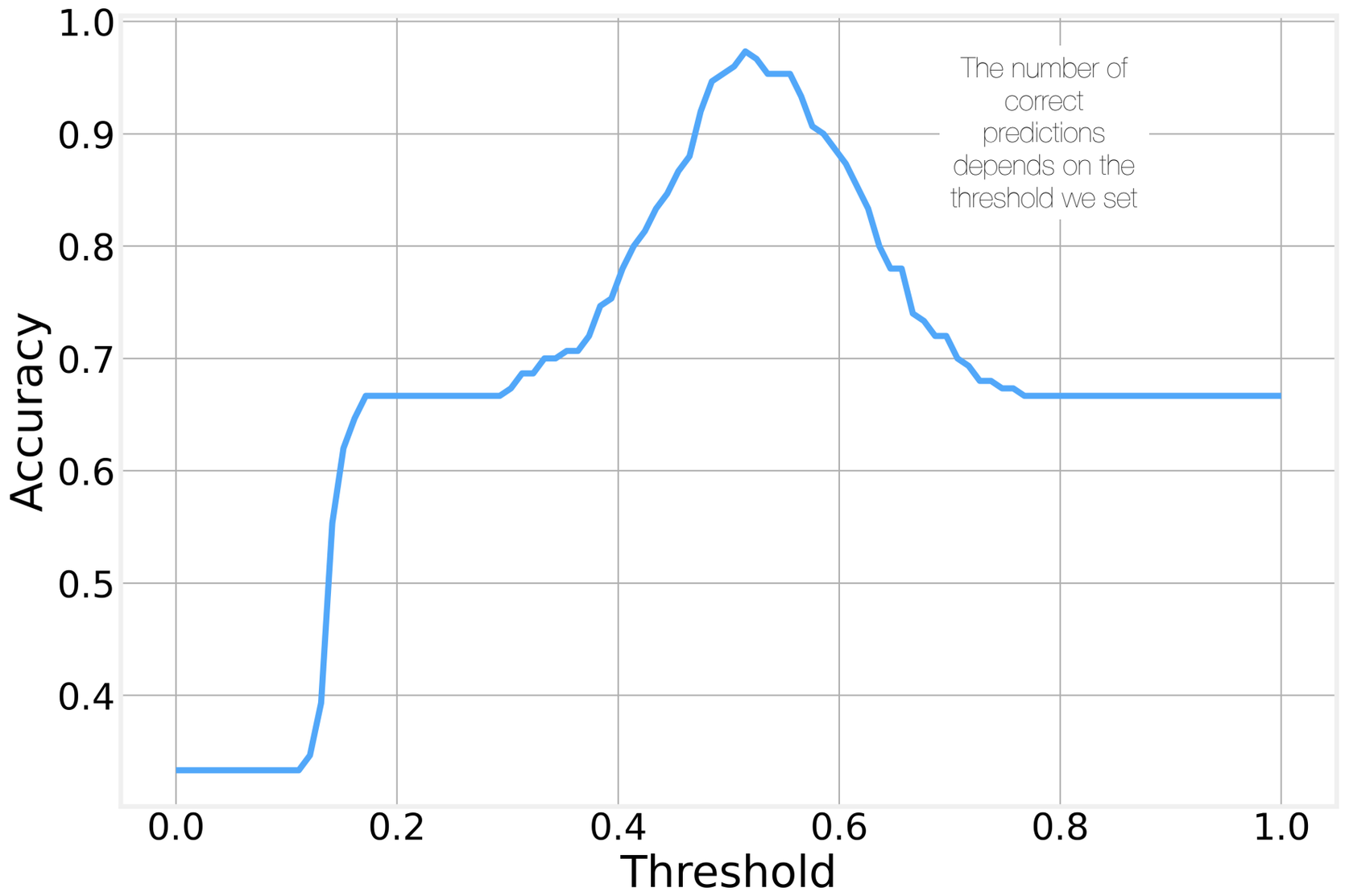
Logistic Regression



Logistic Regression



Logistic Regression

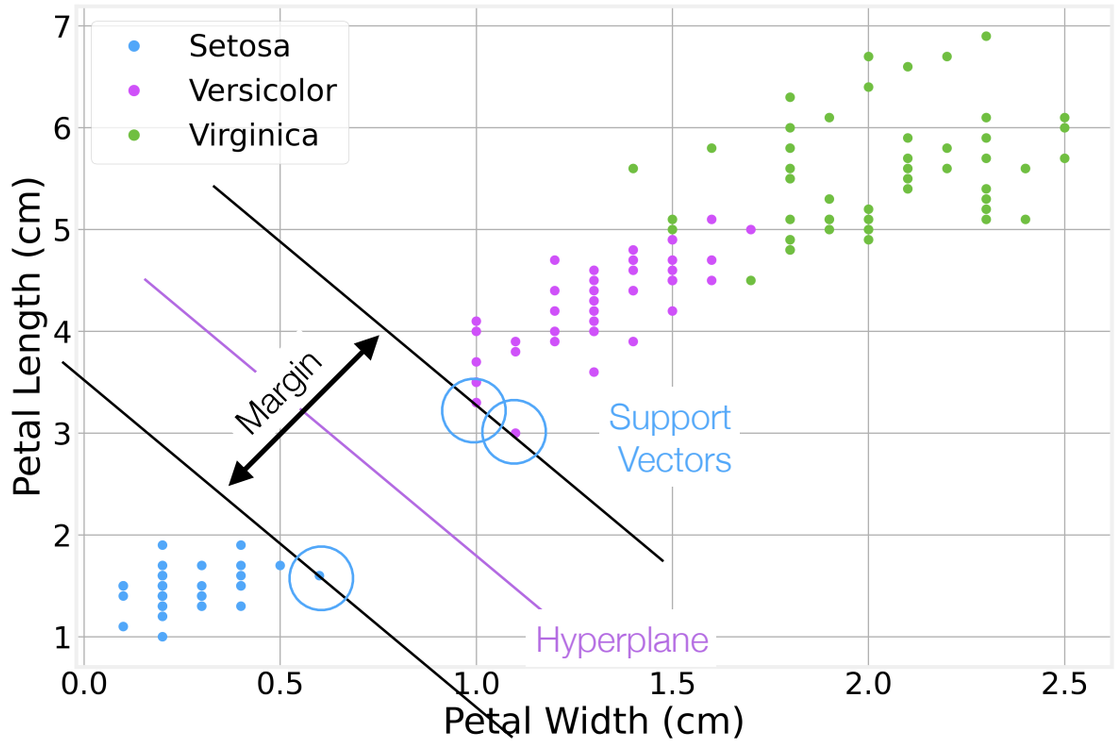


Support Vector Machines

- Find the optimal hyperplane that best separates different classes.
- The hyperplane (or decision boundary) should be as far as possible from any data points
- Closest data points are called support vectors.
- Distance between the hyperplane and the nearest support vectors is called the margin.
- SVM finds the hyperplane that maximizes the margin.

Support Vector Machines

- Find the optimal hyperplane that best separates different classes.
- The hyperplane (or decision boundary) should be as far as possible from any data points
- Closest data points are called support vectors.
- Distance between the hyperplane and the nearest support vectors is called the margin.
- SVM finds the hyperplane that maximizes the margin.



Support Vector Machines

- A hyperplane in N dimensions is defined by:

$$\vec{w} \cdot \vec{x} = 0$$

meaning that it is defined by all the points, $\vec{x}$ perpendicular to a given (weight) vector $\vec{w}$.

- The margin is defined as:

$$\frac{1}{\| \vec{w} \|}$$

which we can maximize by minimizing:

$$\frac{1}{2} \| \vec{w} \|^2$$

- Subject to the constraint that all points are correctly classified:

$$y_i \left(\vec{w} \cdot \vec{x_i} \right) \geq 1$$

The convention in SVMs is that $y_i \in \{-1,1\}$

Support Vector Machines

- Slack variables, $\xi_i \geq 0$, allow for some points to be misclassified. Using ξ_i, then the loss function is:

$$\frac{1}{2} \parallel \vec{w} \parallel^2 + C \sum_i^n \xi_i$$

- And the constraint becomes:

$$y_i \left(\vec{w} \cdot \vec{x_i} \right) \geq 1 - \xi_i$$

- Model predictions are given by:

$$y_i = \text{sign} \left(\vec{w} \cdot \vec{x}_i \right)$$

Practical Considerations

- So far we have looked at very idealized cases. Reality is never this simple!

- In practice, many details have to be considered:

    - Data normalization

    - Overfitting

    - Hyperparameters

    - Bias, Variance tradeoffs

    - etc…

Data Normalization

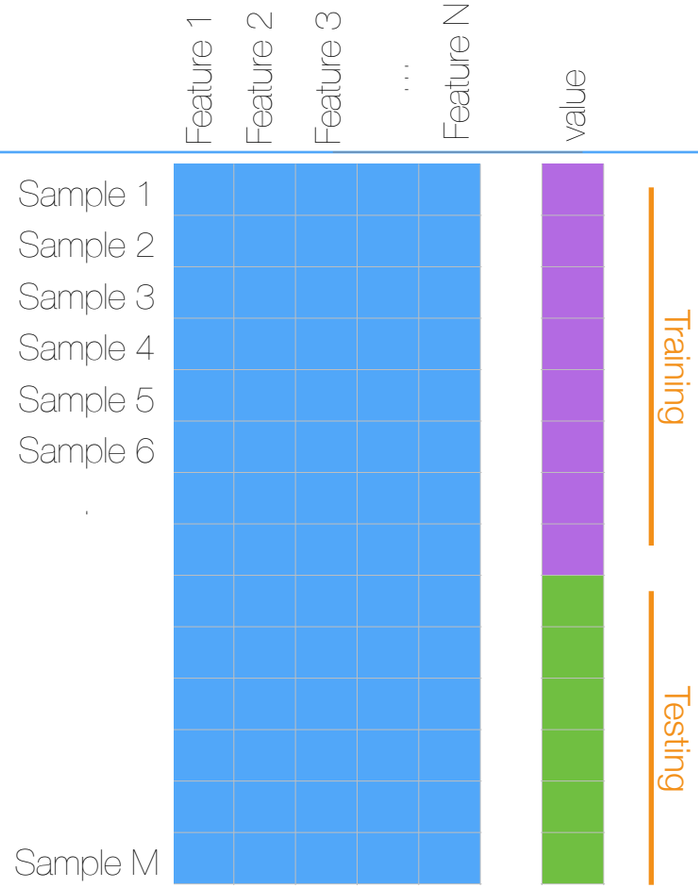
- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some

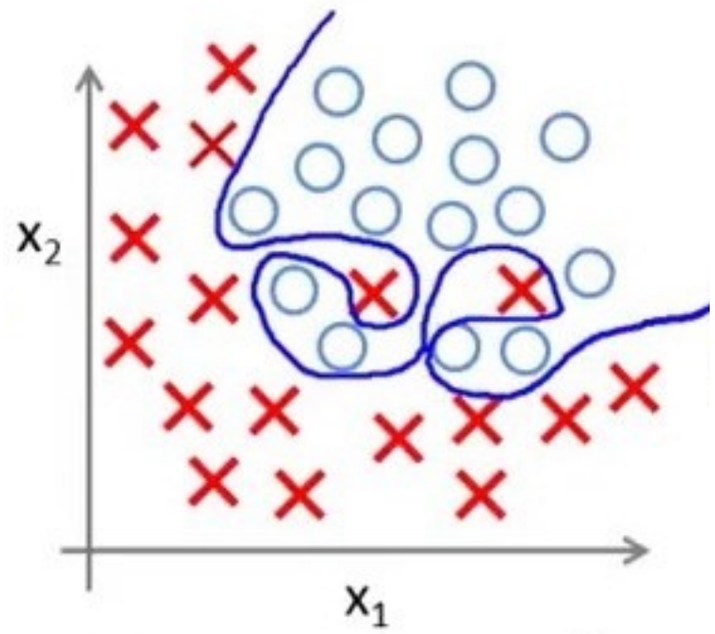
Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some

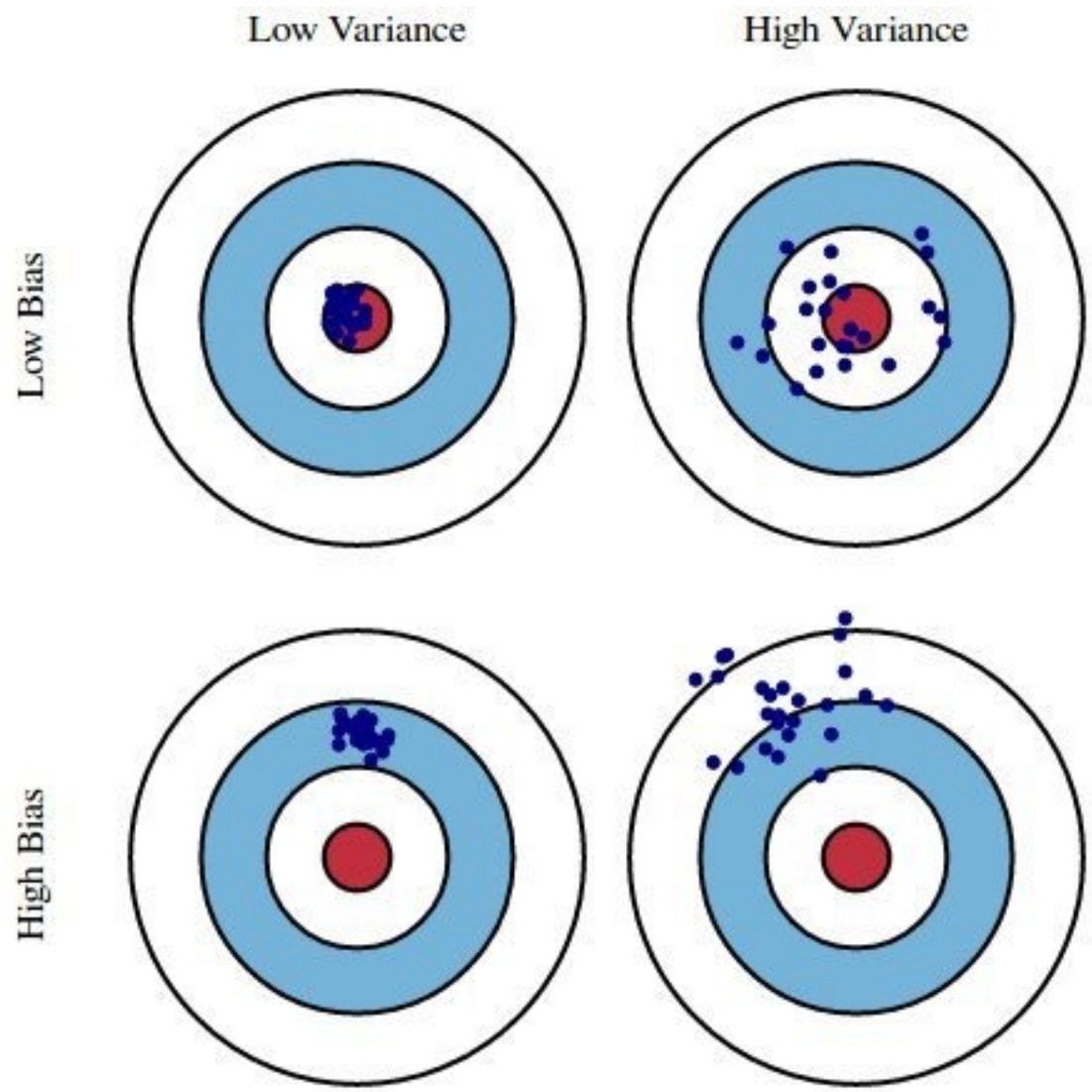
Supervised Learning - Overfitting

- "Learning the noise"
- "Memorization" instead of "generalization"
- How can we prevent it?
  - Split dataset into two subsets: Training and Testing
  - Train model using only the Training dataset and evaluate results in the previously unseen Testing dataset.
- Different heuristics on how to split:
  - Single split
  - k-fold cross validation: split dataset in k parts, train in k-1 and evaluate in 1, repeat k times and average results.

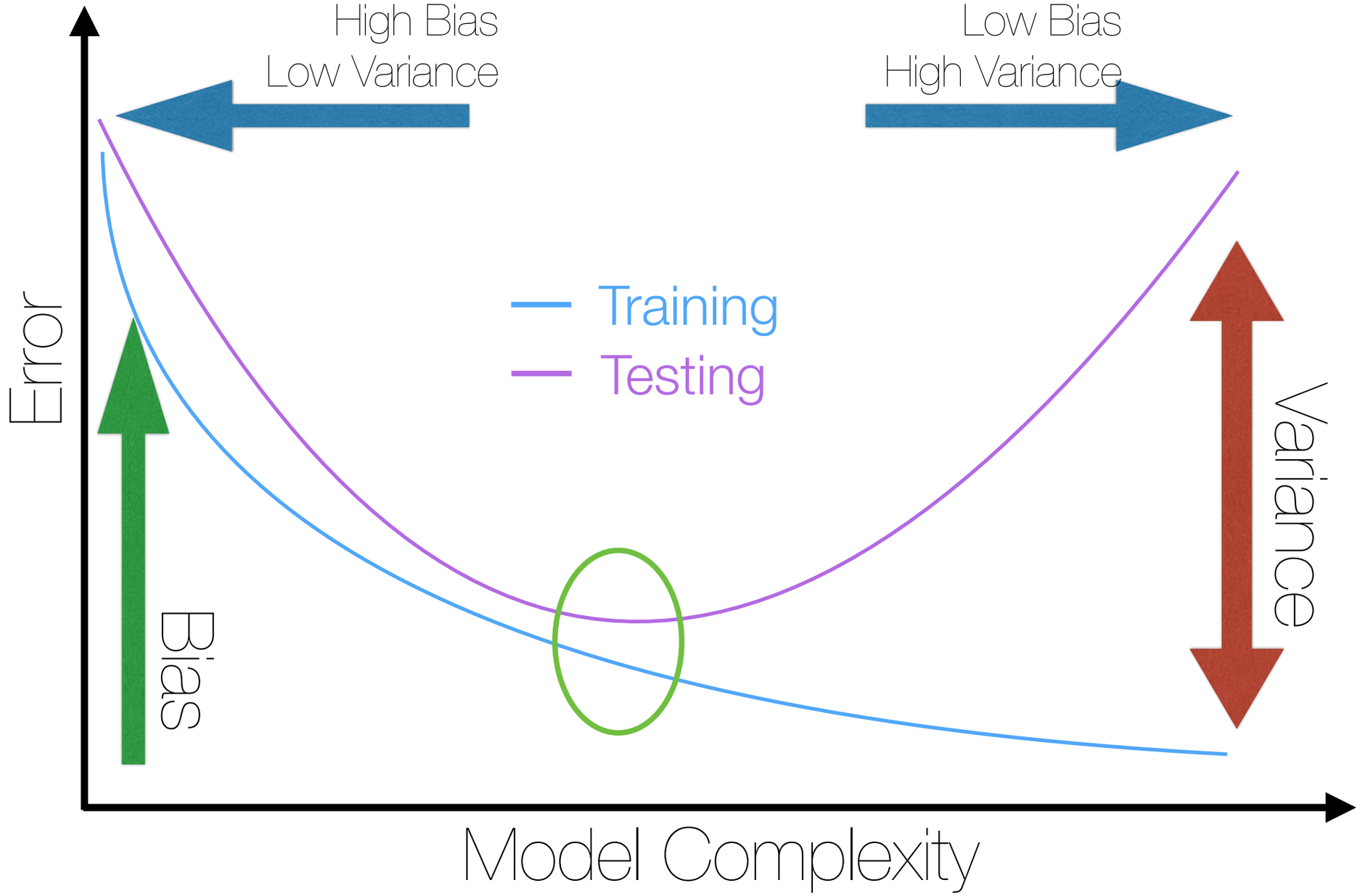




Bias-Variance Tradeoff



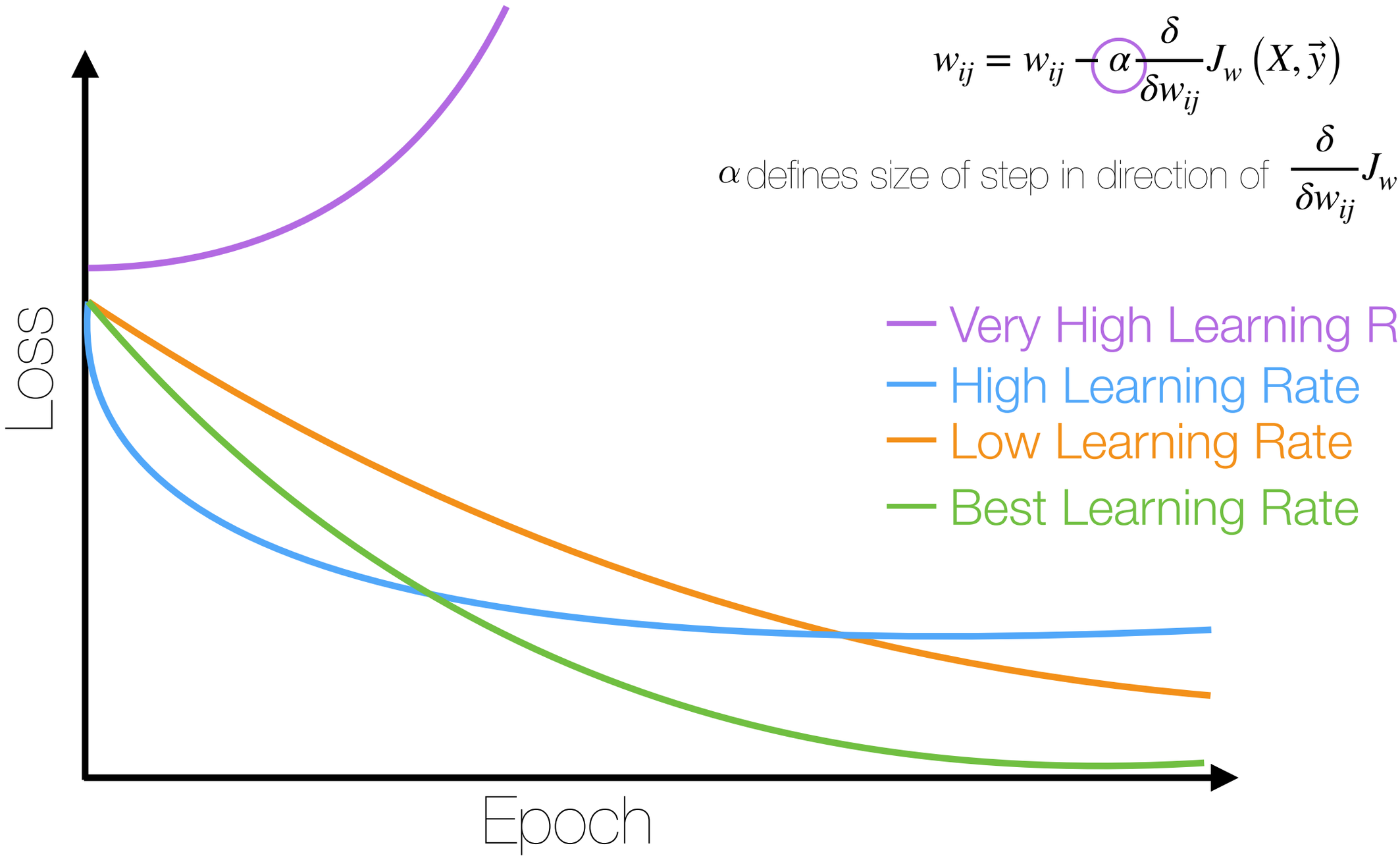
Bias-Variance Tradeoff



Learning Rate

$$w_{ij} = w_{ij} - \alpha \frac{\delta}{\delta w_{ij}} J_w\left(X, \vec{y}\right)$$

α defines size of step in direction of $\frac{\delta}{\delta w_{ij}} J_w\left(X, \vec{y}\right)$



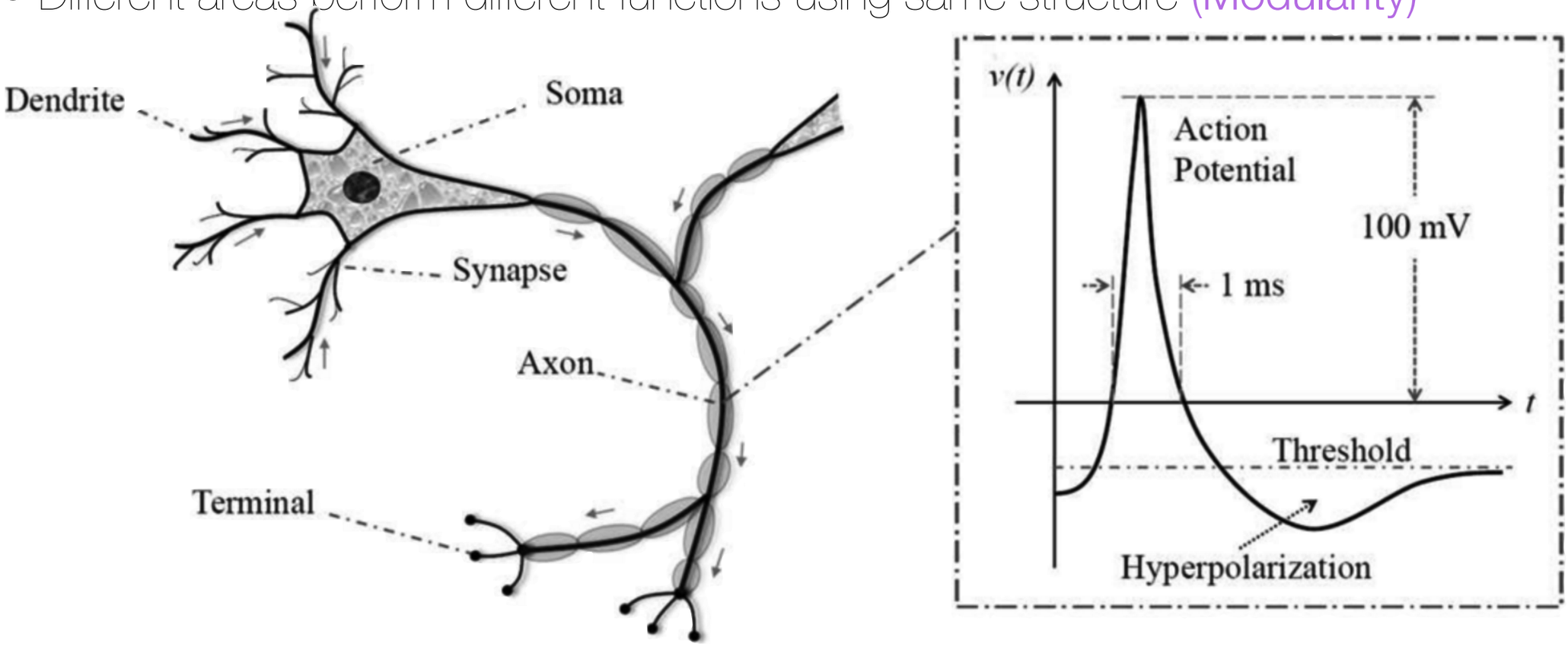
Supervised Learning

http://github.com/DataForScience/PyTorch/

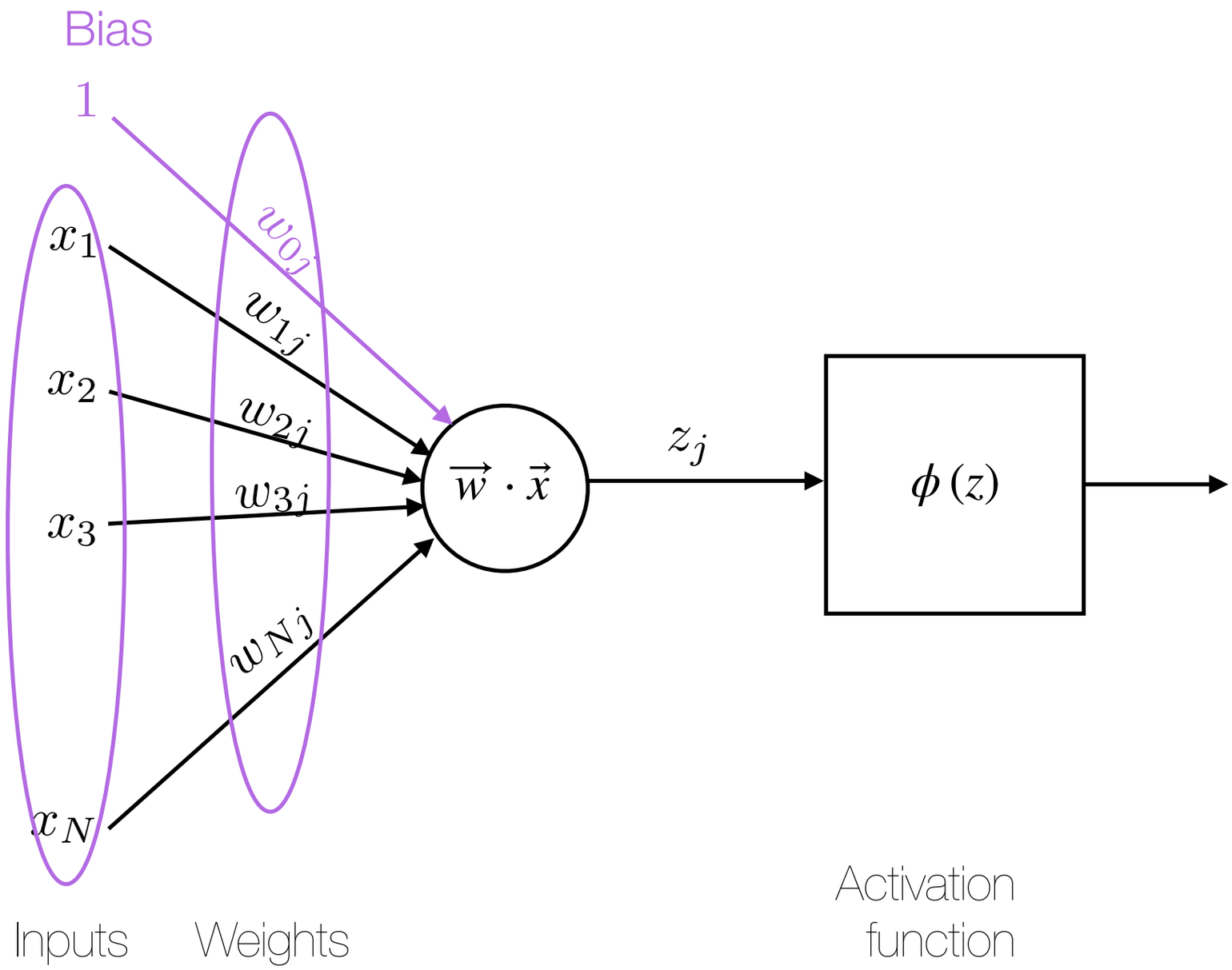
4. Neural Networks

How the Brain "Works" (Cartoon version)

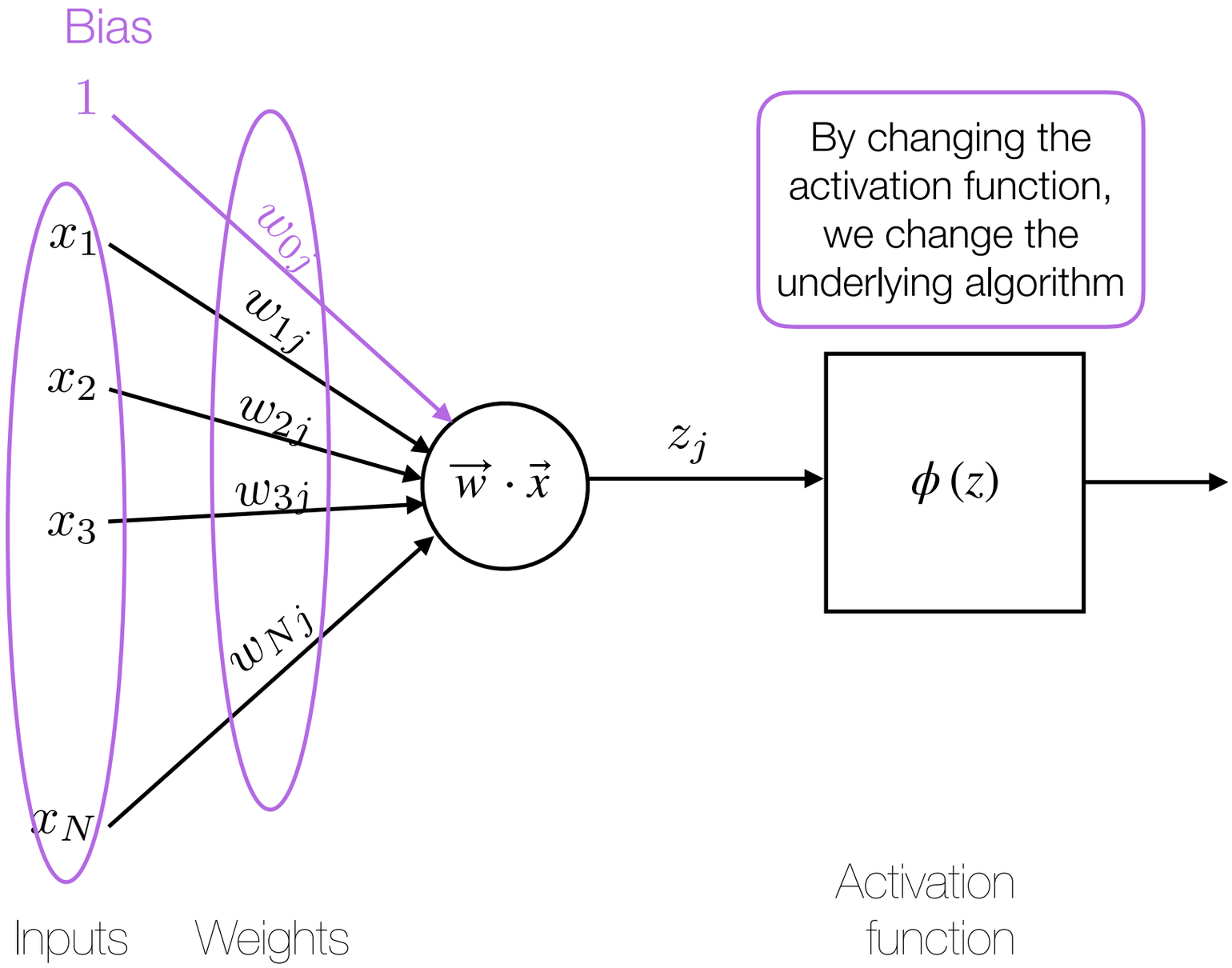
- Each neuron receives input from other neurons
- 10^{11} neurons, each with with 10^{4} weights
- Weights can be positive or negative
- Weights adapt during the learning process
- "neurons that fire together wire together" (Hebb)
- Different areas perform different functions using same structure (Modularity)



Artificial Neuron

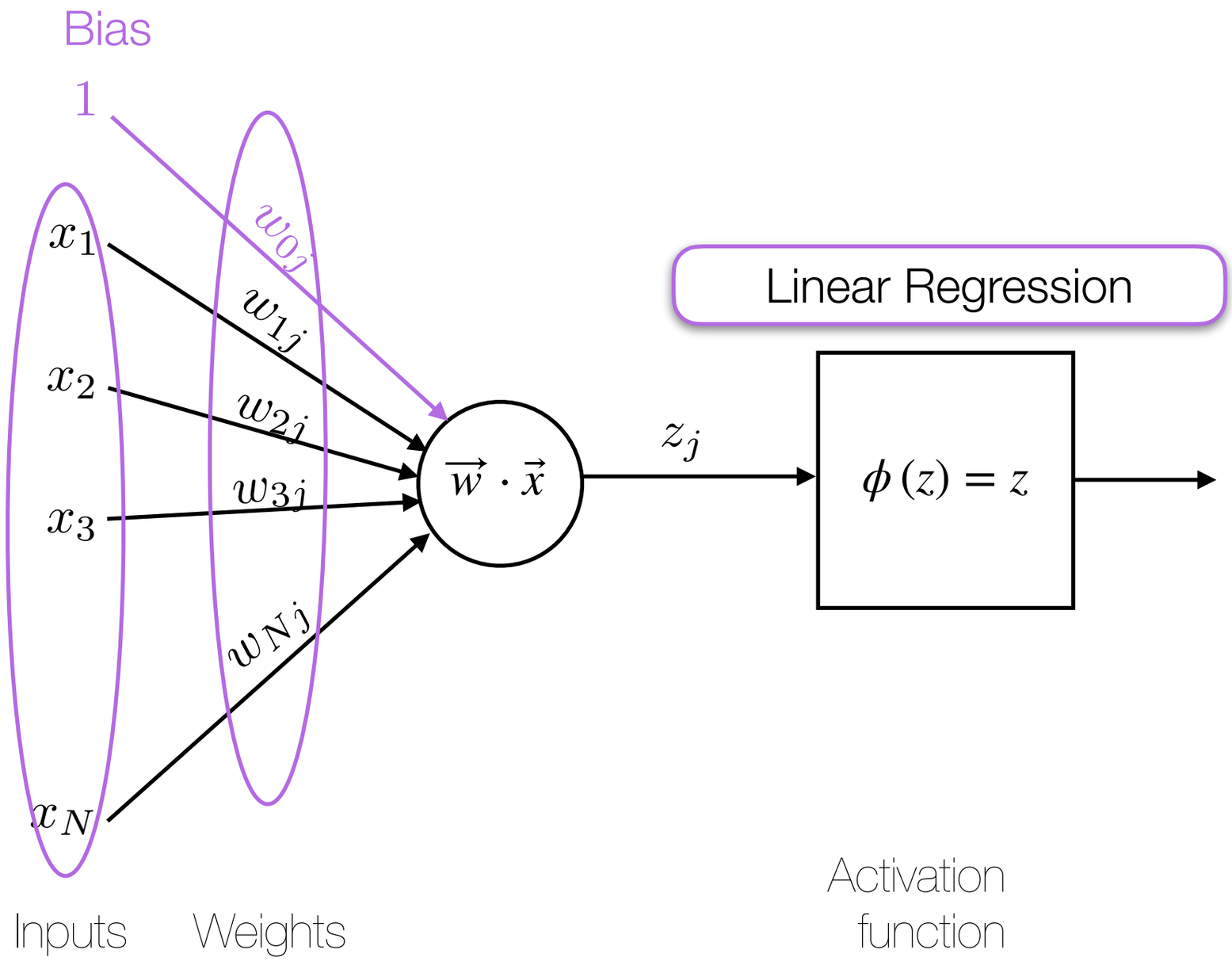


Artificial Neuron

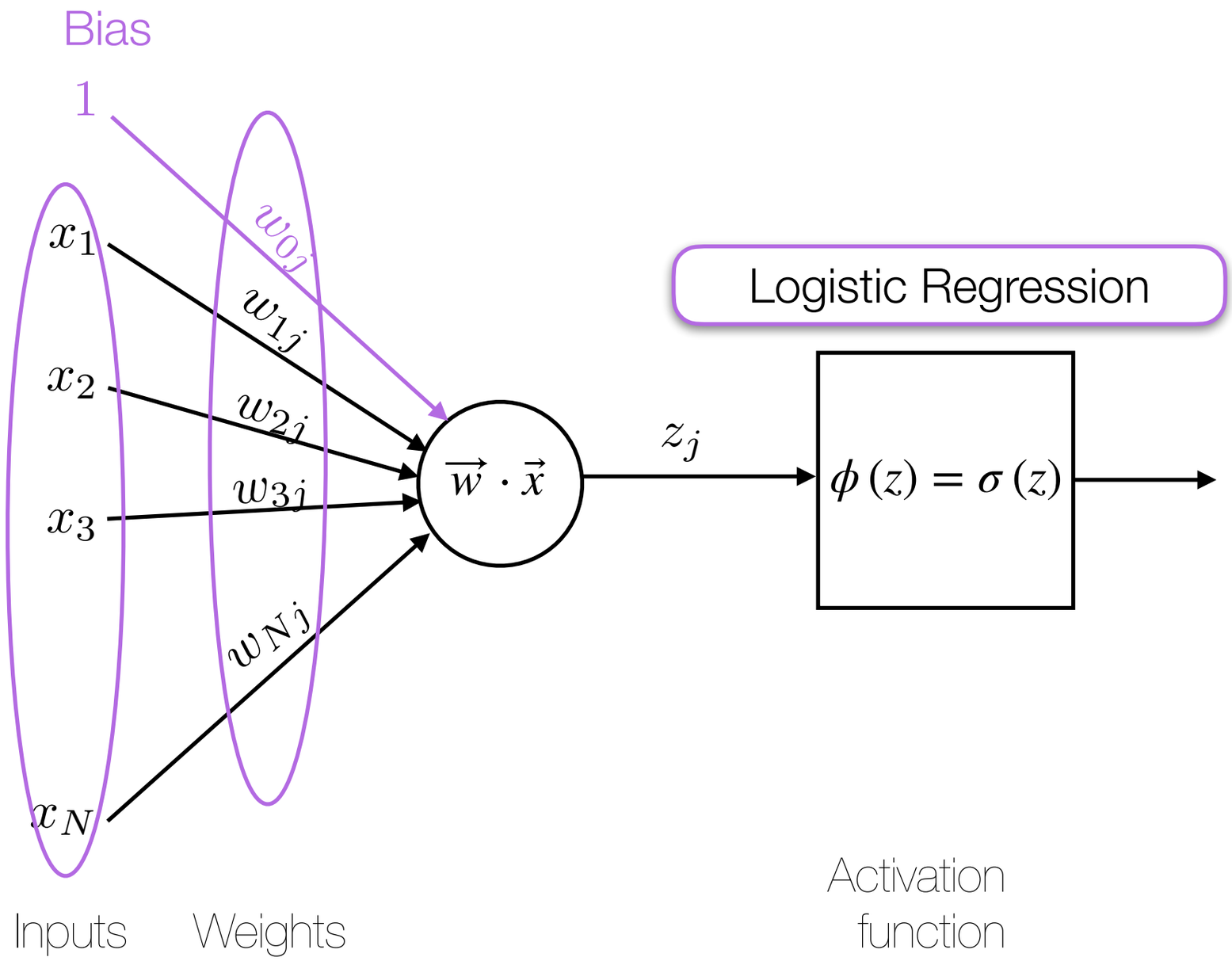


By changing the activation function, we change the underlying algorithm

Artificial Neuron



Artificial Neuron

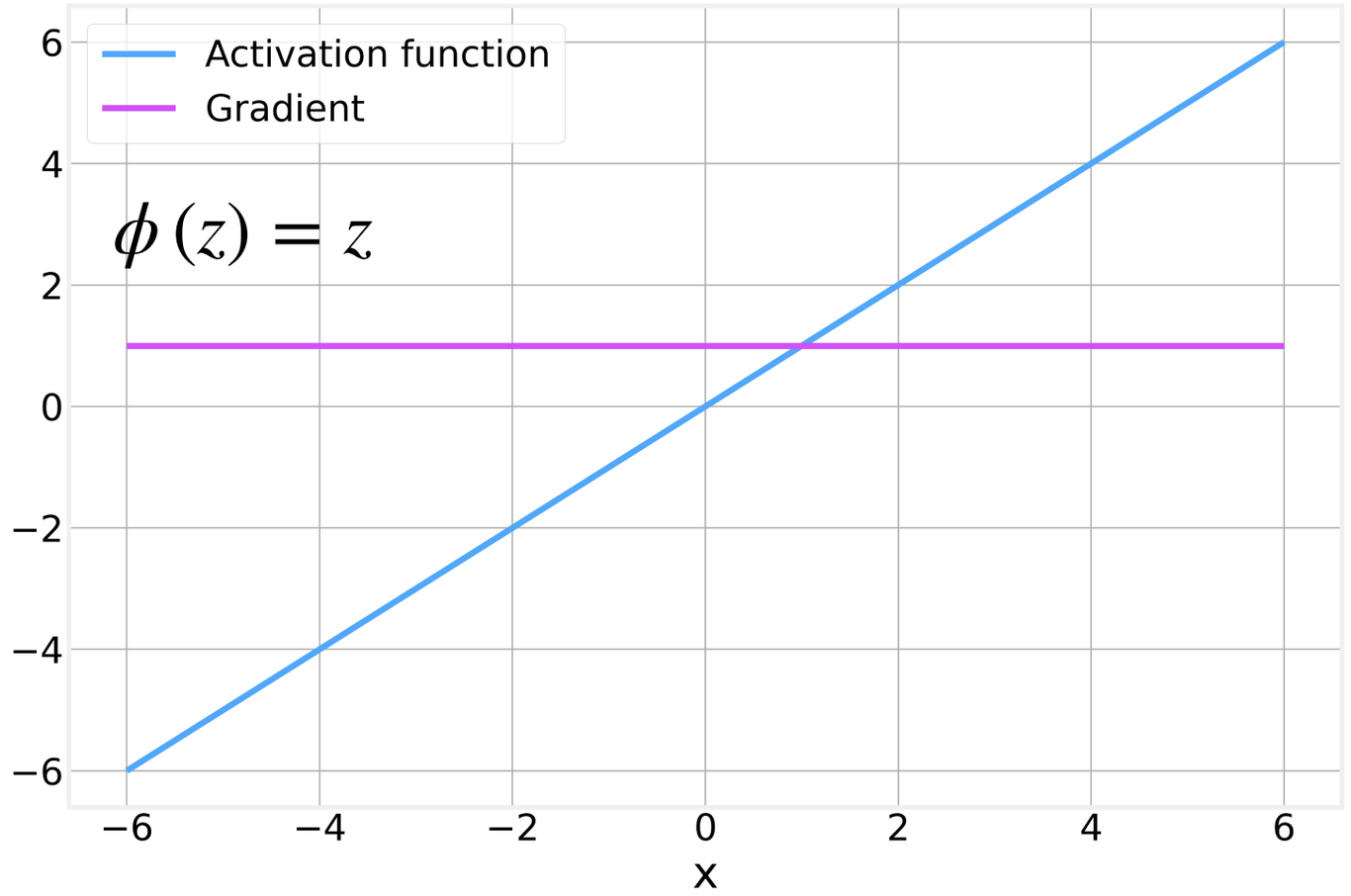


Activation Function

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data

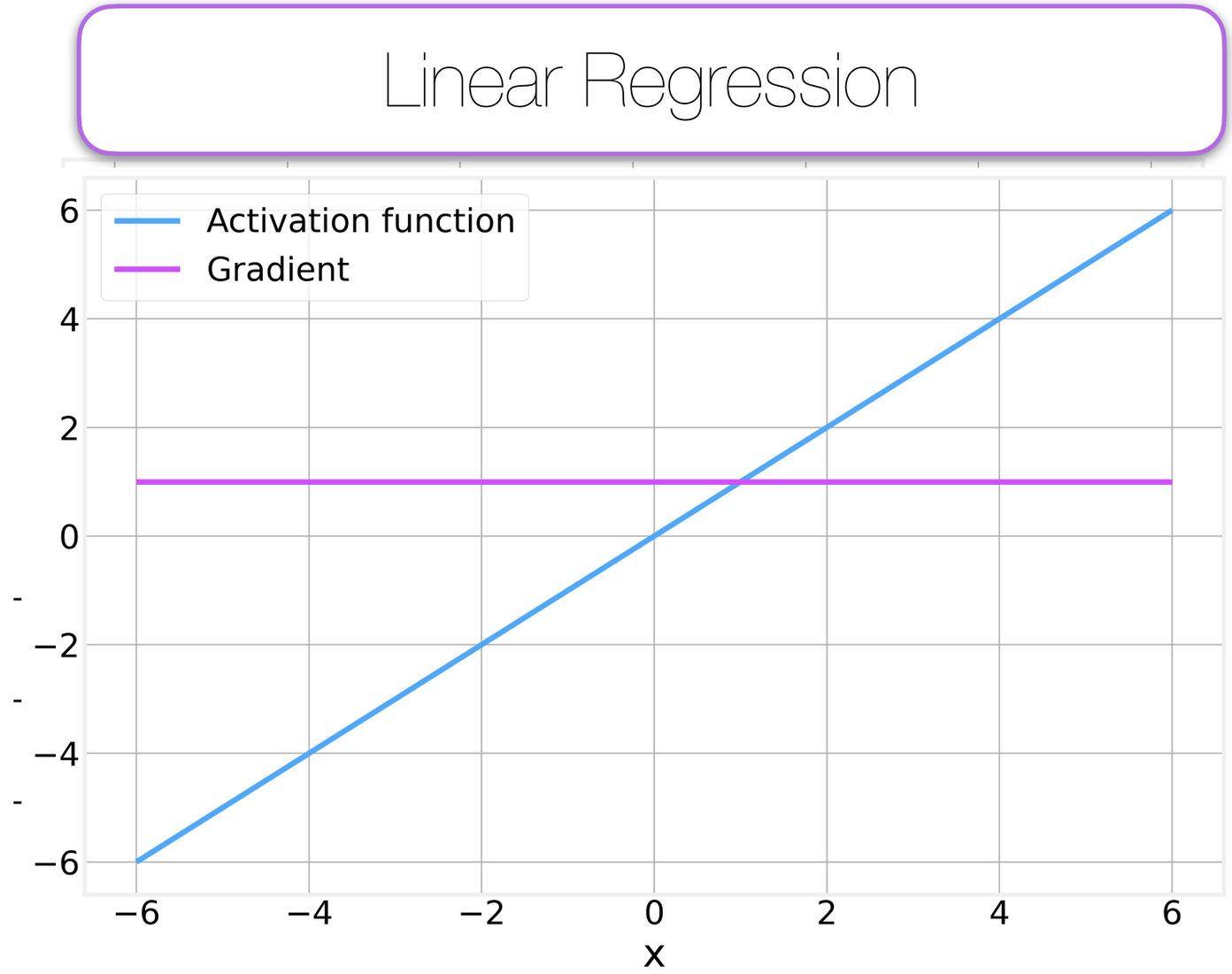
Activation Function - Linear

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- The simplest



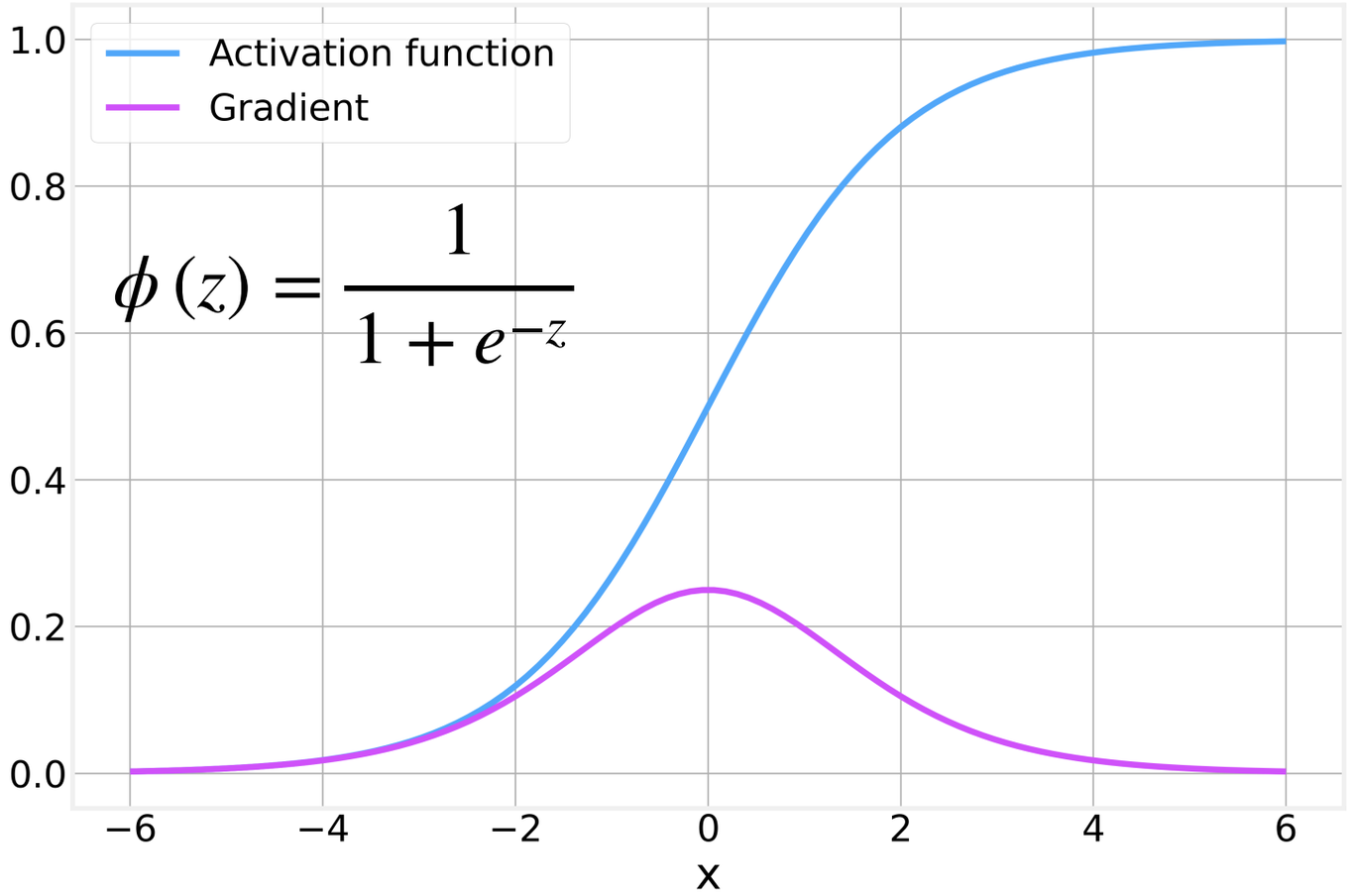
Activation Function - Linear

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- The simplest



Activation Function - Sigmoid

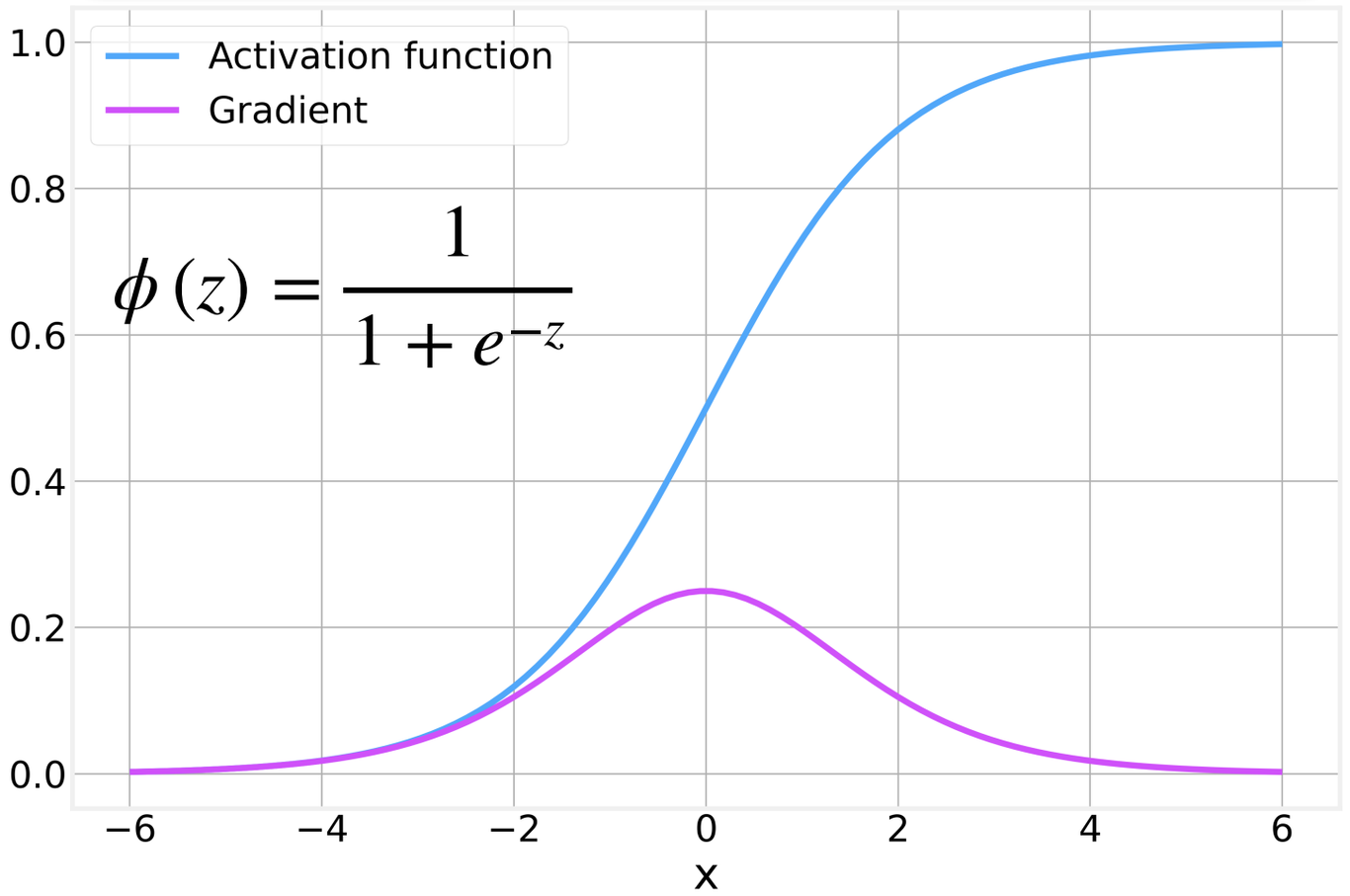
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Perhaps the most common



Activation Function - Sigmoid

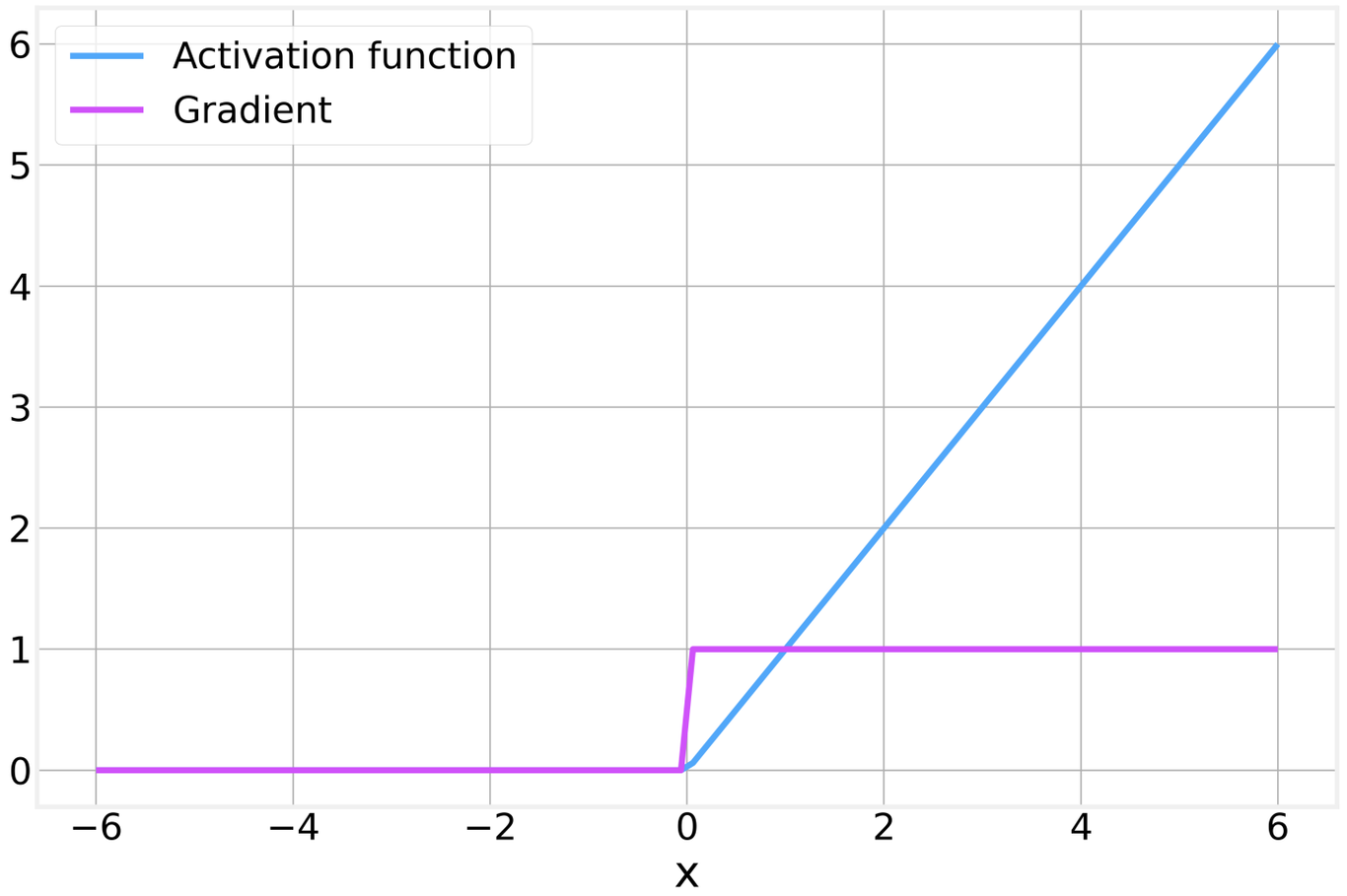
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Perhaps the most common

Logistic Regression



Activation Function - ReLu

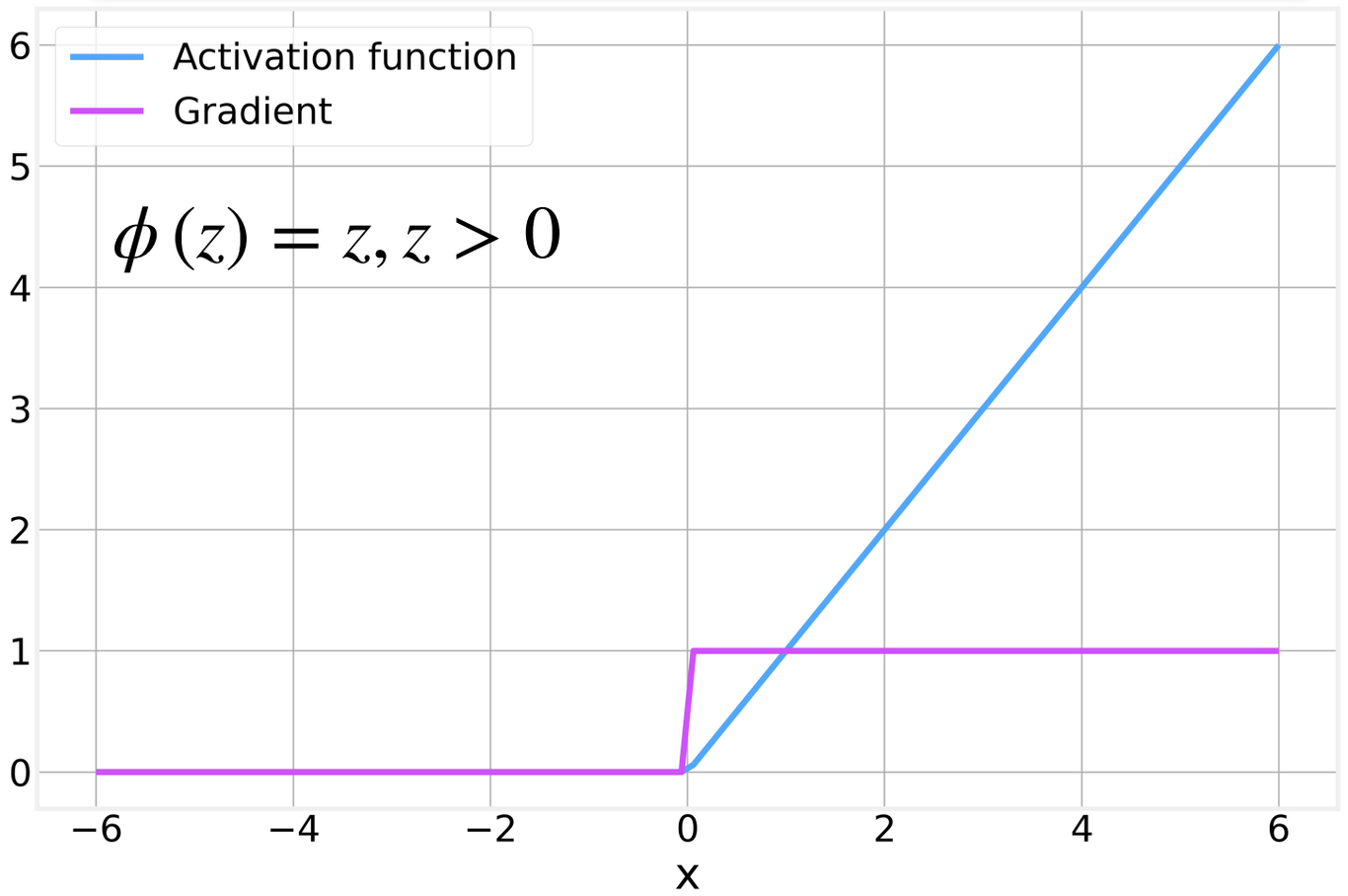
- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Results in faster learning than with sigmoid



Activation Function - ReLu

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Results in faster learning than with sigmoid

Stepwise Regression



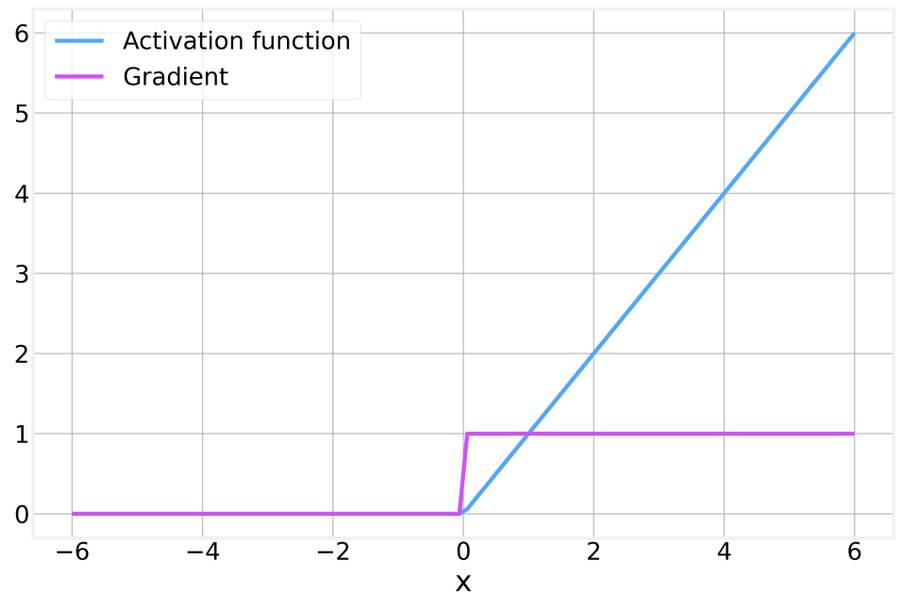
Stepwise Regression

https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline

- Multivariate Adaptive Regression Spline (MARS) is the best known example
- Fit curves using a linear combination of:

$$\hat{f}(x) = \sum_i c_i B_i(x)$$

- The basis functions $B_i(x)$ can be:
  - Constant
  - "Hinge" functions of the form: $max(0, x - b)$ and $max(0, b - x)$
  - Products of hinges

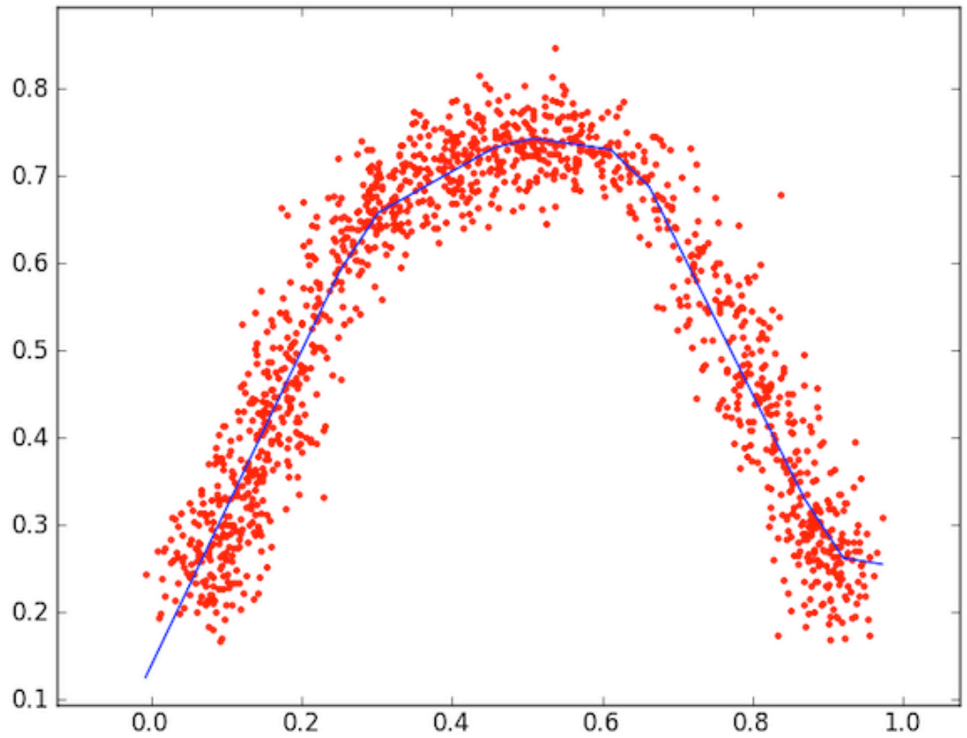


Stepwise Regression

https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline

- Multivariate Adaptive Regression Spline (MARS) is the best known example
- Fit curves using a linear combination of: $\hat{f}(x) = \sum_i c_i B_i(x)$

$$
\begin{aligned}
y(x) &= 1.013 \\
&+ 1.198\ max(0, x - 0.485) \\
&- 1.803\ max(0, 0.485 - x) \\
&- 1.321\ max(0, x - 0.283) \\
&- 1.609\ max(0, x - 0.640) \\
&+ 1.591\ max(0, x - 0.907)
\end{aligned}
$$

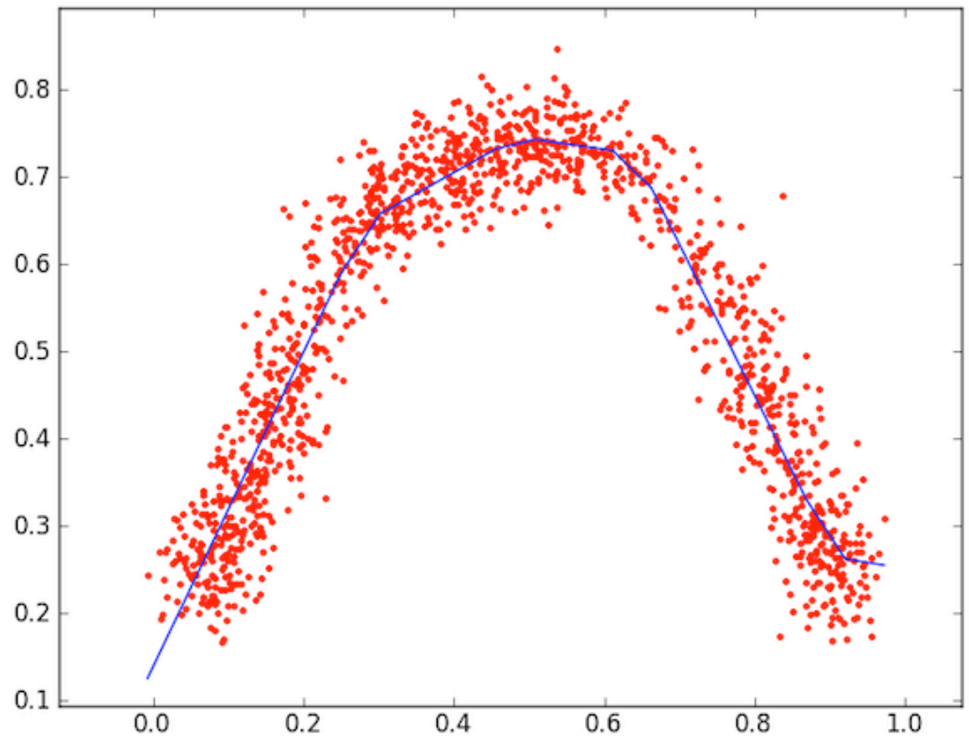


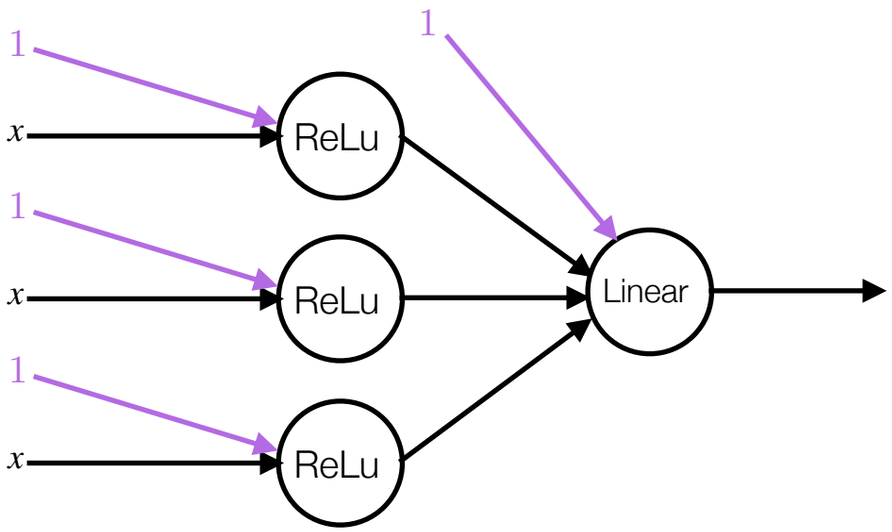
Stepwise Regression

https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_spline

- Multivariate Adaptive Regression Spline (MARS) is the best known example
- Fit curves using a linear combination of: $\hat{f}(x) = \sum_i c_i B_i(x)$

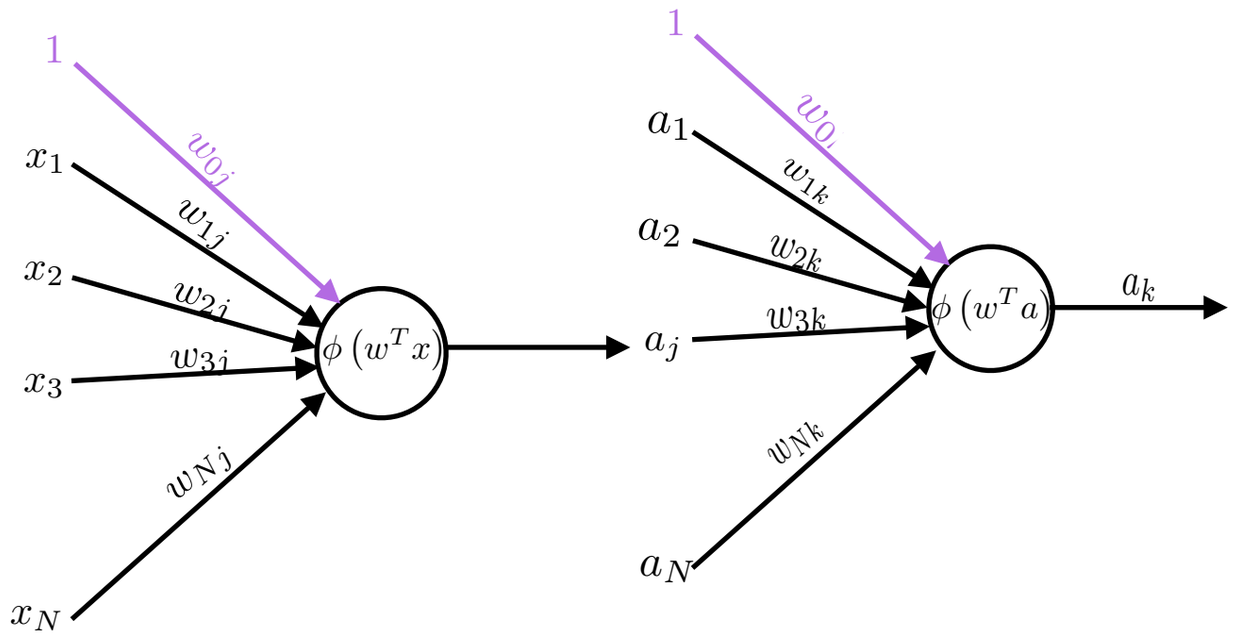
$$
\begin{aligned}
y(x) &= 1.013 \\
&+ 1.198\ max(0, x - 0.485) \\
&- 1.803\ max(0, 0.485 - x) \\
&- 1.321\ max(0, x - 0.283) \\
&- 1.609\ max(0, x - 0.640) \\
&+ 1.591\ max(0, x - 0.907)
\end{aligned}
$$





Forward Propagation

- The output of a perceptron is determined by a sequence of steps:
  - obtain the inputs
  - multiply the inputs by the respective weights
  - calculate output using the activation function
- To create a multi-layer perceptron, you can simply use the output of one layer as the input to the next one.



- But how can we propagate back the errors and update the weights?

Loss Functions

- For learning to occur, we must quantify how far off we are from the desired output. There are two common ways of doing this:

  - Quadratic error function:

$$E = \frac{1}{N} \sum_n |y_n - a_n|^2$$

  - Cross Entropy

$$J = -\frac{1}{N} \sum_n \left[y_n^T \log a_n + (1 - y_n)^T \log (1 - a_n) \right]$$

The Cross Entropy is complementary to sigmoid activation in the output layer and improves its stability.

Regularization

- Helps keep weights relatively small by adding a penalization to the cost function.

- Two common choices:

$$\hat{J}_w(X) = J_w(X) + \lambda \sum_{ij} \left| w_{ij} \right| \qquad \text{"Lasso"}$$

$$\hat{J}_w(X) = J_w(X) + \lambda \sum_{ij} w_{ij}^2 \qquad \text{L2}$$

- Lasso helps with feature selection by driving less important weights to zero

Backward Propagation of Errors (BackProp)

- BackProp operates in two phases:
    - Forward propagate the inputs and calculate the deltas
    - Update the weights
- The error at the output is a weighted average difference between predicted output and the observed one.
- For inner layers there is no “real output”!

BackProp

- Let $\delta^{(l)}$ be the error at each of the total L layers:

- Then:

$$\delta^{(L)} = h_w(X) - y$$

- And for every other layer, in reverse order:

$$\delta^{(l)} = W^{(l)T}\delta^{(l+1)} .* \phi^{\dagger}\left(z^{(l)}\right)$$

- Until:

$$\delta^{(1)} \equiv 0$$

as there's no error on the input layer.

- And finally:

$$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}$$

$$\frac{\partial}{\partial w_{ij}^{(l)}} J_w\left(X, \vec{y}\right) = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda w_{ij}^{(l)}$$

torch.nn

- The torch.nn module is PyTorch's flexible and modular Neural Networks library. It provides a wide range of tools to build and train neural networks.

- nn.Module is the base class (and workhorse) for all NN modules in PyTorch. It provides a standard way to define models, layers, encapsulate parameters, define forward passes, and manage submodules. Models are created by subclassing nn.Module.

- Models must, at a minimum, override the __init__() and forward() methods of nn.Module

```python
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        # Define layers here

    def forward(self, x):
        # Define forward pass here
        return x
```

- torch.nn provides a range of pre-defined layers and loss functions to simplify the development of neural network models

torch.nn

- Common Layers:
  - nn.Linear - Fully connected layer
  - nn.Conv2d - 2D convolutional layer
  - nn.ReLU - Rectified Linear Unit activation function
  - nn.MaxPool2d - 2D max pooling layer
  - nn.BatchNorm2d - 2D batch normalization layer
- Loss Functions
  - nn.CrossEntropyLoss - Cross-entropy loss for classification tasks
  - nn.MSELoss - Mean squared error loss for regression tasks
  - nn.BCELoss - Binary cross-entropy loss for binary classification tasks

torch.optim

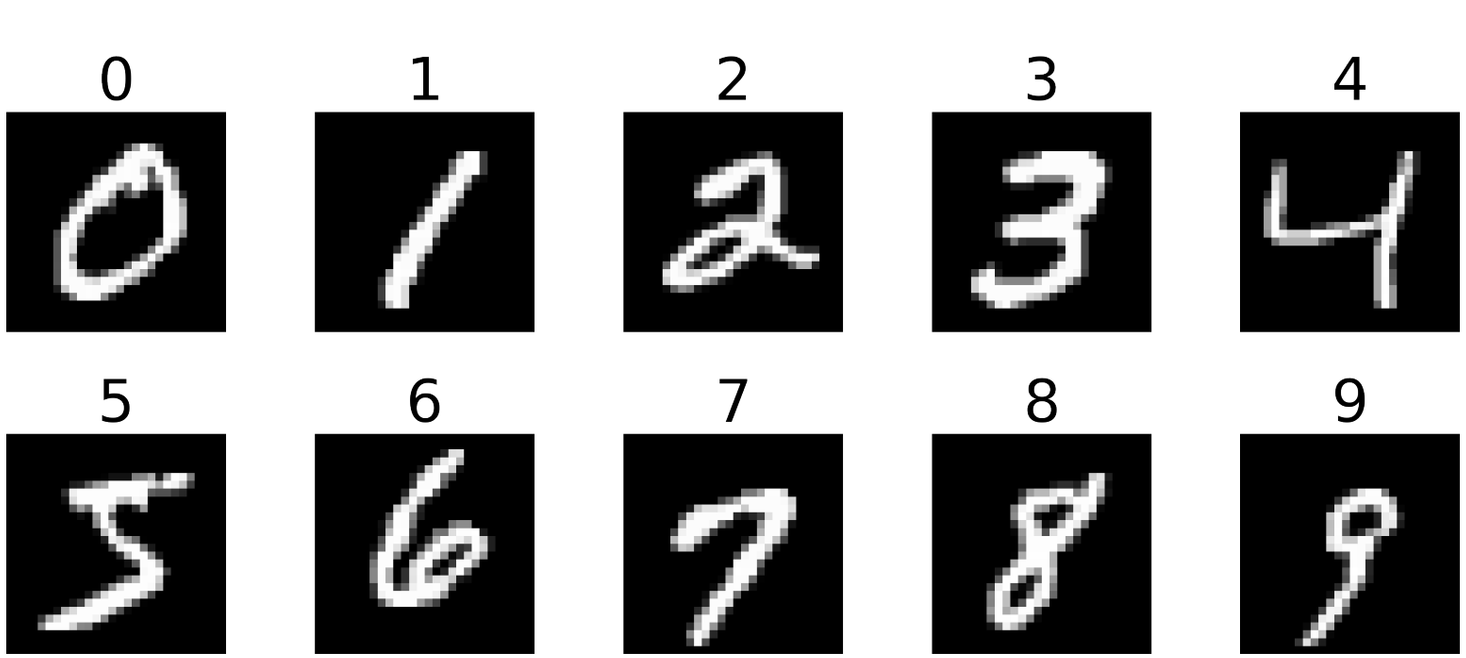
- Optimizers are essential for training neural networks.
- PyTorch provides several optimizers in the torch.optim module, such as:
  - optim.SGD - Stochastic Gradient Descent
  - optim.Adam - Adaptive Moment Estimation
  - optim.RMSprop - Root Mean Square Propagation

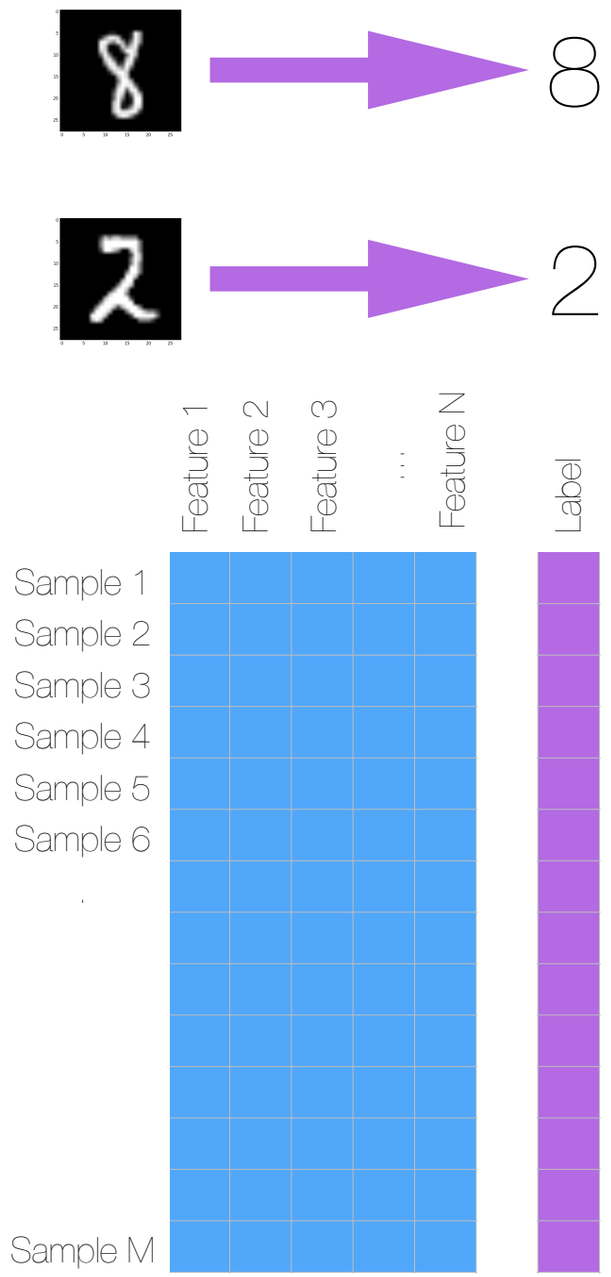
A practical example - MNIST

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond





yann.lecun.com/exdb/mnist/

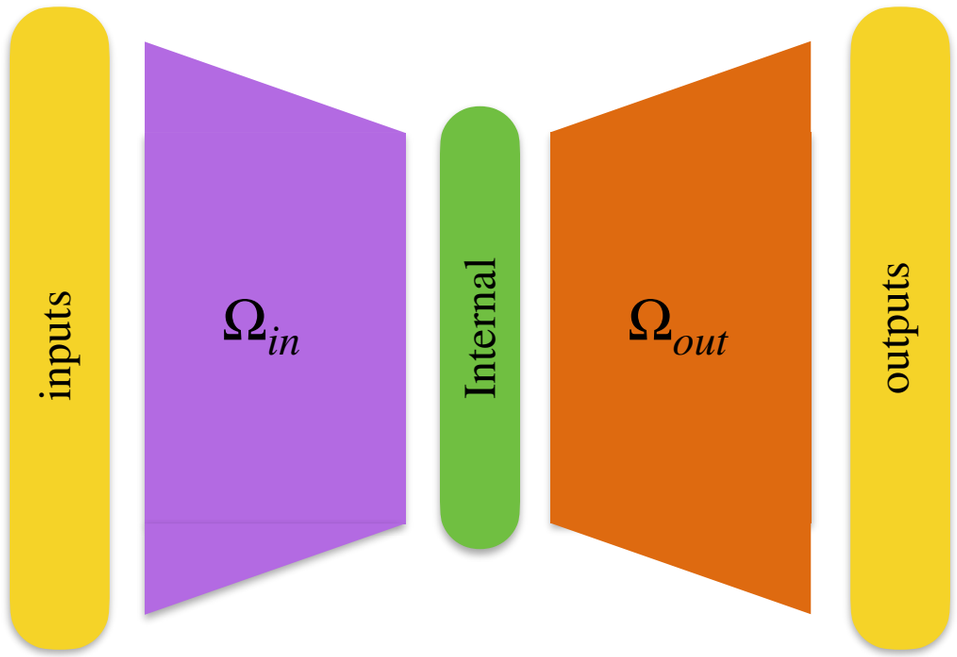
Neural Networks

http://github.com/DataForScience/PyTorch/

5. Deep Learning Applications

Encoder-Decoder

- Auto-Encoders use the same values for both inputs and outputs
- The Internal/hidden layer(s) have a smaller number of units than the input
- The fundamental idea is that the Network needs to learn an internal representation of its inputs that is smaller but from which it is still possible to reconstruct the input.

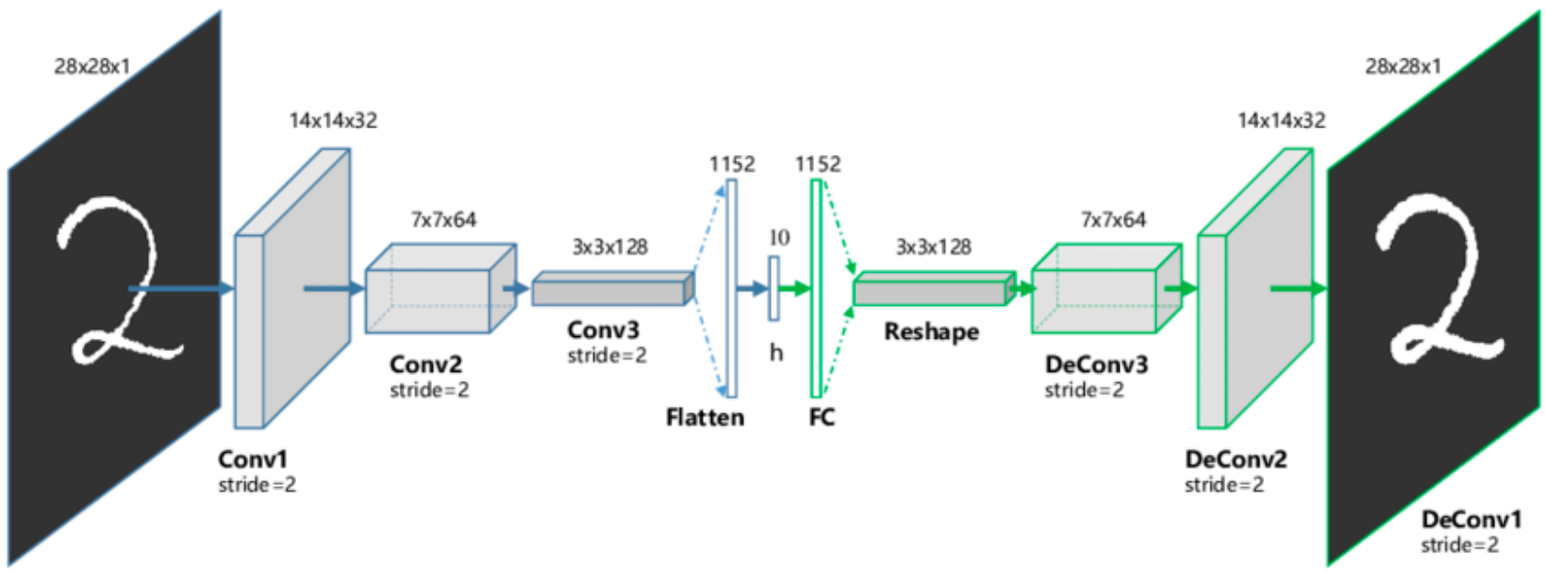


- Think of it as "zipping" and "unzipping" the input values. 

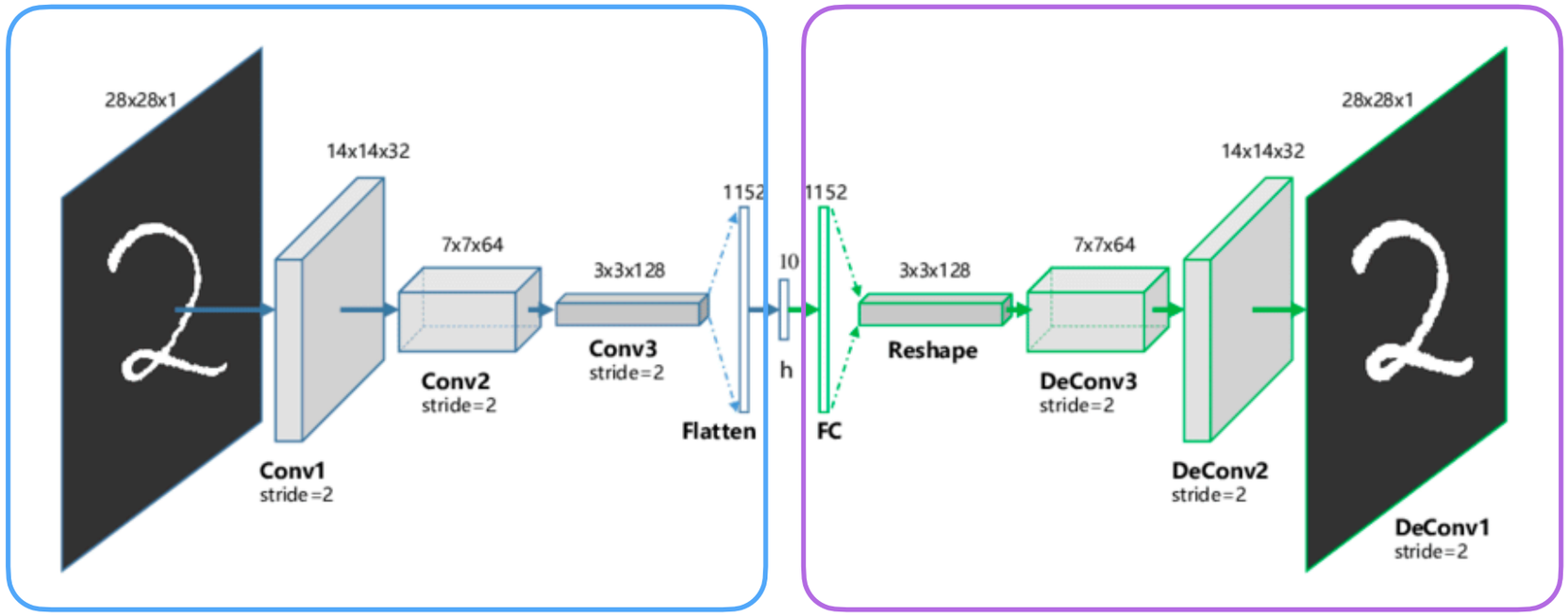
Auto-Encoders

https://www.researchgate.net/figure/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there_fig1_320658590

- After training, the parts of the network that generate the internal representation can be used as inputs to the Networks
- This is similar to what we did when we reused the word embeddings generated by training a word2vec network
- Auto-encoders can be arbitrarily complex, including many layers between the input and the internal representation (or Code) and are often used in Image Processing to generate efficient representations of complex images



Encoder/Decoder Architecture



Encoder

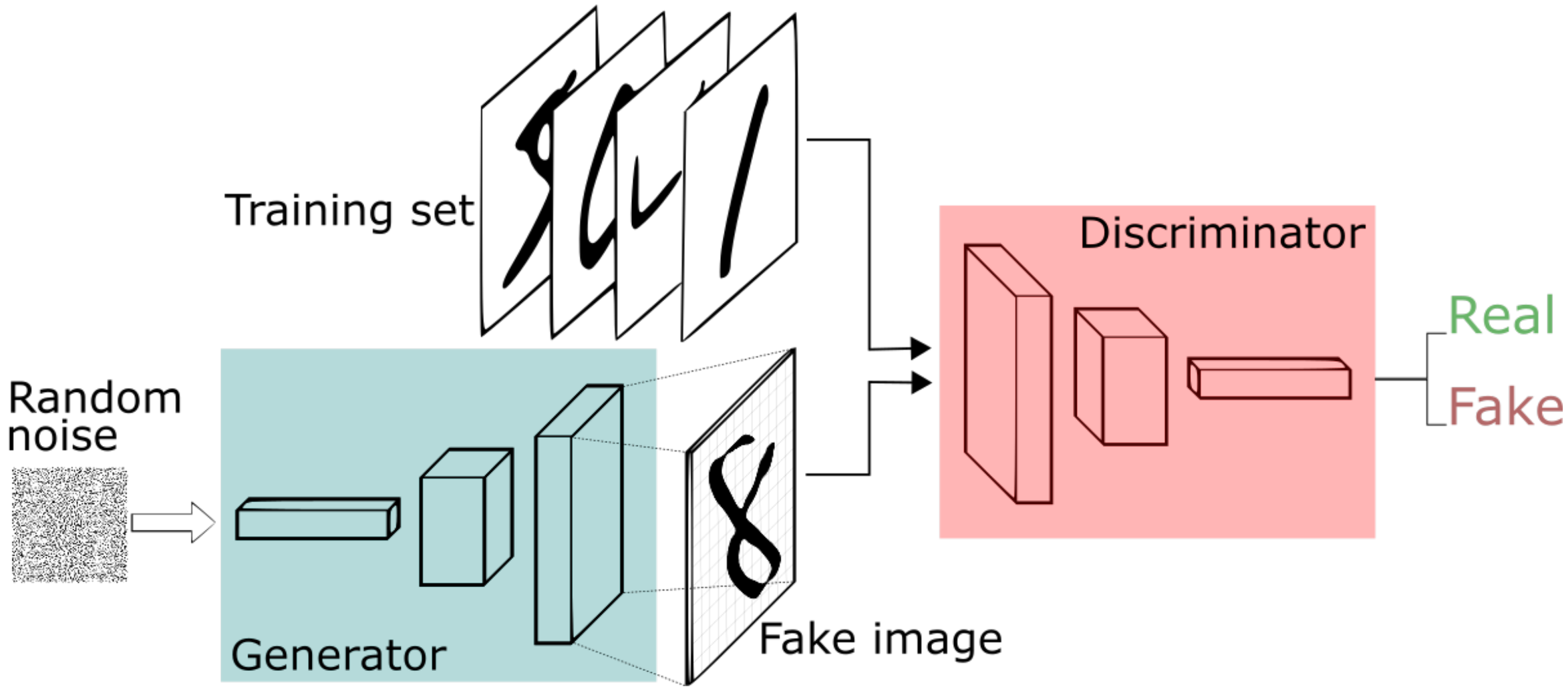
Input -> Internal Representation

Decoder

Internal Representation -> Output

Generative Adversarial Networks

https://sthalles.github.io/intro-to-gans/



Generative Adversarial Networks

- Generative Adversarial Networks (GANs) consist of two neural networks: a Generator and a Discriminator.
- Generator creates fake data samples, while the Discriminator evaluates their authenticity.
- Both networks are trained simultaneously in a zero-sum game framework.
- The Generator learns to produce data indistinguishable from real data.
- The Discriminator learns to distinguish real from fake data samples.
- GANs can generate high-quality synthetic data, such as images, text, and audio.
- Useful in various applications, such as image synthesis, data augmentation, and style transfer.

Graph Neural Networks

https://lightning.ai/docs/pytorch/stable/notebooks/course_UvA-DL/06-graph-neural-networks.html

- Graph Neural Networks (GNNs) are designed to work directly with graph-structured data.
- GNNs leverage the relationships and interactions between nodes in a graph to improve learning and prediction tasks.
- Particularly useful for tasks like node classification, link prediction, etc where data naturally represented as a graph:
    - Social networks
    - Molecular structures
    - Knowledge graphs
- GNNs use message passing mechanisms to aggregate information from neighboring nodes and update node representations.

Graph Neural Networks

https://lightning.ai/docs/pytorch/stable/notebooks/course_UvA-DL/06-graph-neural-networks.html

- Key components of GNNs include nodes (vertices), edges (connections), and node/edge features.

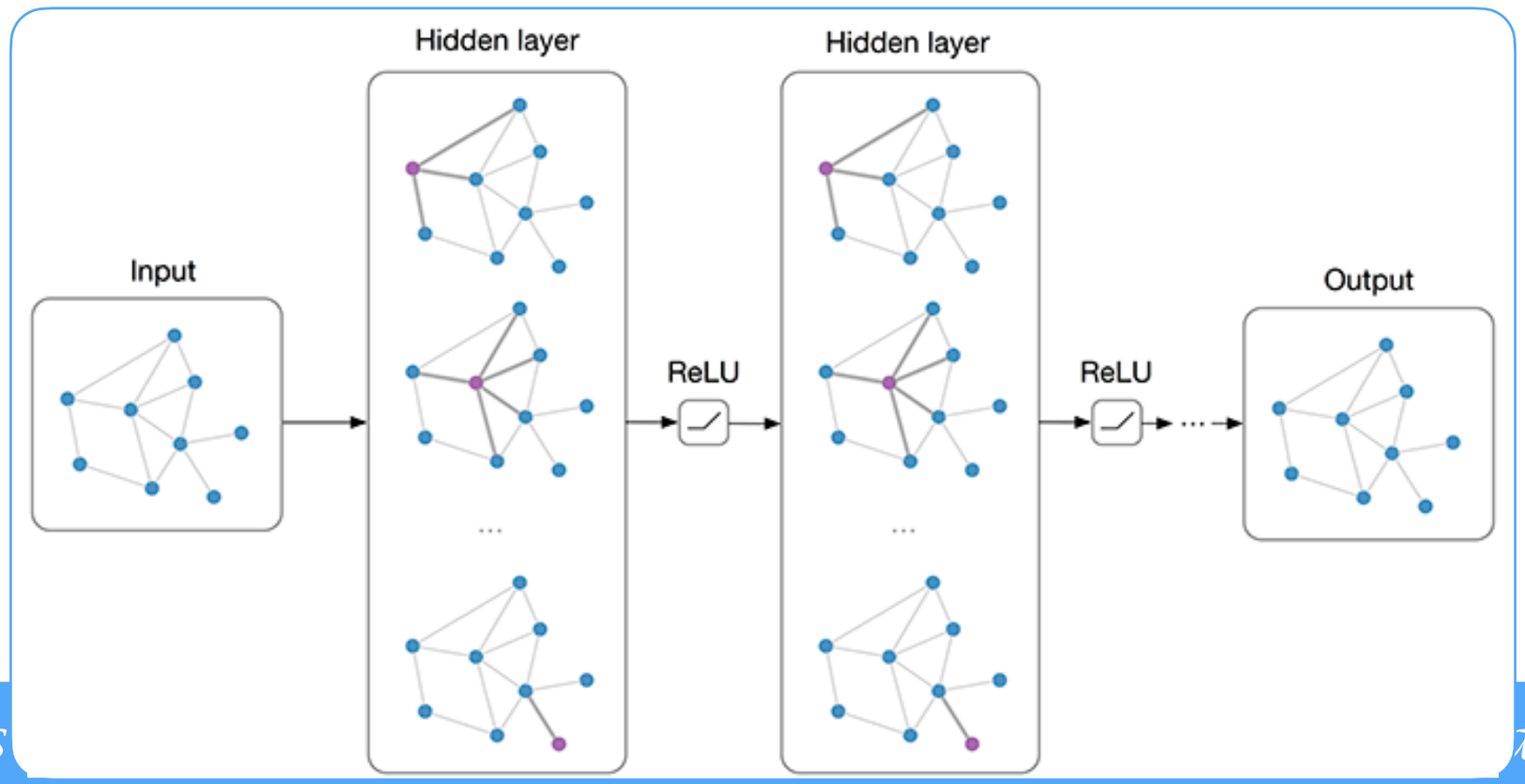
- Can handle homogeneous graphs (single kind of nodes and edges) and heterogeneous graphs (multiple types of nodes and edges).

- Training GNNs typically involves supervised learning, but unsupervised and semi-supervised approaches are also used.

Graph Neural Networks

https://lightning.ai/docs/pytorch/stable/notebooks/course_UvA-DL/06-graph-neural-networks.html

- Training algorithm:
  - Propagate node features through the GNN layers.
  - Aggregate information from neighboring nodes using message-passing mechanisms.
  - Apply non-linear activation functions after each layer.
  - Compute the loss based on the model's predictions and the ground truth labels.
  - Update the GNN parameters using an optimization algorithm like gradient descent.



Deep Learning Applications

http://github.com/DataForScience/PyTorch/

Events

data4sci.substack.com

LLMs for Data Science

Mar 26, 2025 - 10am-2pm (PST)

NLP with PyTorch

Apr 9, 2025 - 10am-2pm (PST)

Bruno Gonçalves

https://data4sci.com

info@data4sci.com

https://data4sci.com/call

Python Data Visualization: Create impactful visuals, animations and dashboards

https://bit.ly/DataViz_LL

Begin

Complete this course and earn a badge!

6h 36m • 11 sections

Course Outline

Sneak Peek

The Sneak Peek program provides early access to Pearson video products and is exclusively available to subscribers. Content for titles in this program is made available throughout the development cycle, so products may not be complete, edited, or finalized, including video post-production editing.

Information visualization, as David McCandless aptly puts it, is a form of "knowledge compression." Our highly evolved visual processing system enables us to efficiently handle vast amounts of information. Visualization's power lies in its ability to encode data intuitively, making complex data accessible. As data grows in volume and complexity, the importance of effective visualization increases. This video explores how the human visual cortex processes colors and shapes and how we can utilize these mechanisms for effective visualization using Pythons powerful visualization libraries.

Starting with pandas and Matplotlib, two core Python libraries, we learn about the basics of Python data pre-processing and visualization before moving on to more advanced packages. Seaborn, built on top of Matplotlib, simplifies common tasks and enhances productivity. Interactive visualizations using Bokeh and Plotly are also explored. We'll use Jupyter notebooks to craft our visualizations.

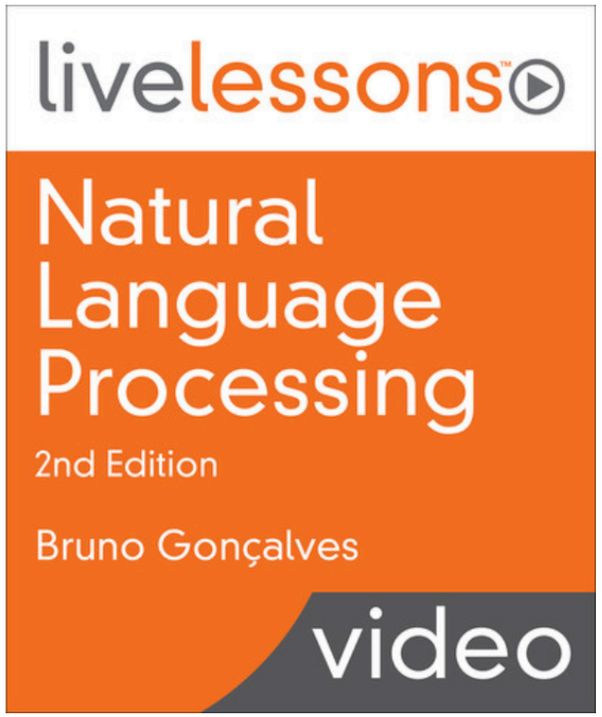
VIDEO

Natural Language Processing, 2nd Edition

Write the first review

By Bruno Goncalves

https://bit.ly/NLP_LL



TIME TO COMPLETE:
5h 23m

TOPICS:
Natural Language Processing

PUBLISHED BY:
Addison-Wesley Professional

PUBLICATION DATE:
October 2021

Start

5 Hours of Video Instruction

Overview

Natural Language Processing LiveLessons covers thefundamentals of Natural Language Processing in a simple and intuitive way,empowering you to add NLP to your toolkit. Using the powerful NLTK package, itgradually moves from the basics of text representation, cleaning, topicdetection, regular expressions, and sentiment analysis before moving on to theKeras deep learning framework to explore more advanced topics such as textclassification and sequence-to-sequence models. After successfully completingthese lessons you'll be equipped with a fundamental and practical understandingof state-of-the-art Natural Language Processing tools and algorithms.

VIDEO

Times Series Analysis for Everyone

★★★★★ 1 review

By Bruno Goncalves

https://bit.ly/Timeseries_LL

TIME TO COMPLETE:
6h

TOPICS:
Time Series

PUBLISHED BY:
Pearson

PUBLICATION DATE:
November 2021

Start

6 Hours of Video Instruction

The perfect introduction to time-based analytics

Overview

Times Series Analysis for Everyone LiveLessons covers the fundamental tools and techniques for the analysis of time series data. These lessons introduce you to the basic concepts, ideas, and algorithms necessary to develop your own time series applications in a step-by-step, intuitive fashion. The lessons follow a gradual progression, from the more specific to the more abstract, taking you from the very basics to some of the most recent and sophisticated algorithms by leveraging the statsmodels, arch, and Keras state-of-the-art models.

END