Building Machine Learning Models in Python with scikit-learn

PROCESSING DATA WITH SCIKIT-LEARN

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Understanding different types of ML algorithms and use cases

Working with numerical and categorical data

Standardization of numerical data input into an ML model

Working with text, representing text data in numerical form

Representing pixel intensities and extracting features from images

Prerequisites and Course Outline

Beginner course on building ML models using scikit-learn

Comfortable with Python programming

Prerequisite Courses

Python: Getting Started

Python Fundamentals

Advanced Python

Related Courses

How to Think About Machine Learning Algorithms

Understanding Machine Learning with Python

Understanding the Foundations of TensorFlow

Software and Skills

Be very comfortable programming in Python (Python 3)

Be comfortable working with Jupyter notebooks

Understand some basics of machine learning

Course Outline

Processing data

- Data preparation, representing text as numbers, representing images as matrices

Building specialized regression models

- Lasso and Ridge regression, Support Vector Regression

Building SVM and gradient boosting models

- Support Vector Machines for text and image classification, Gradient Boosting for regression

Clustering and dimensionality reduction

- Mean-shift clustering, Principal Components Analysis

Understanding Machine Learning

A machine learning algorithm is an algorithm that is able to learn from data

Machine Learning

Work with a huge maze of data

Find patterns

Make intelligent decisions

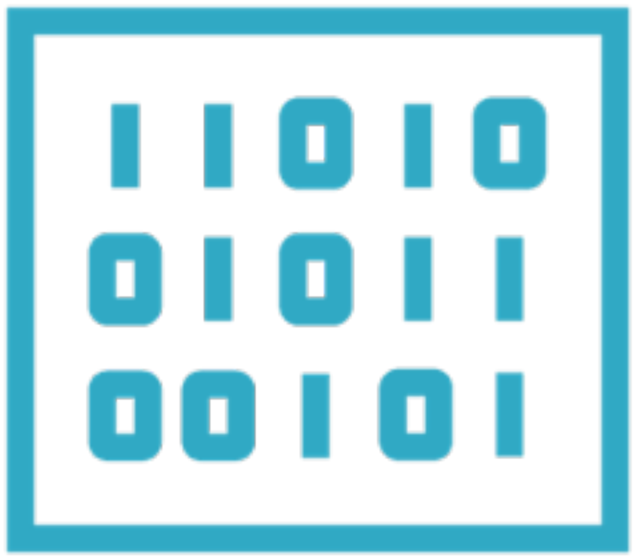
Machine Learning

Emails on a server

Spam or Ham?

Trash or Inbox

Machine Learning



Images represented as pixels

Identify edges, colors, shapes

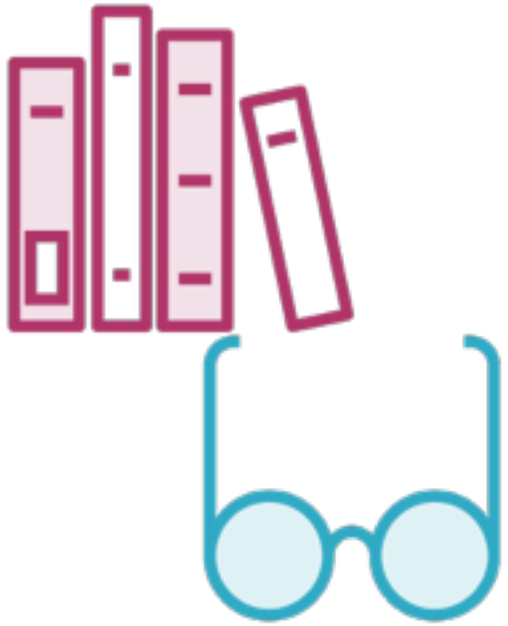
A photo of a little girl

Types of Machine Learning Problems

Classification

Regression

Clustering



Rule-extraction

Types of Machine Learning Problems

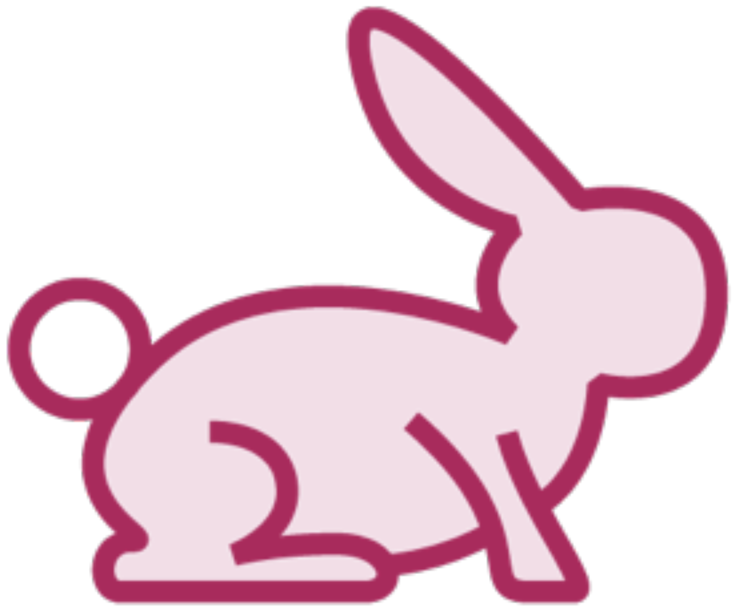
Classification

Regression

Clustering

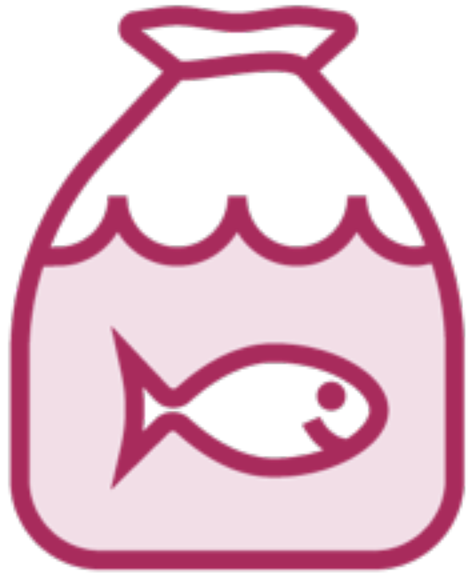
Rule-extraction

Whales: Fish or Mammals?



Mammals

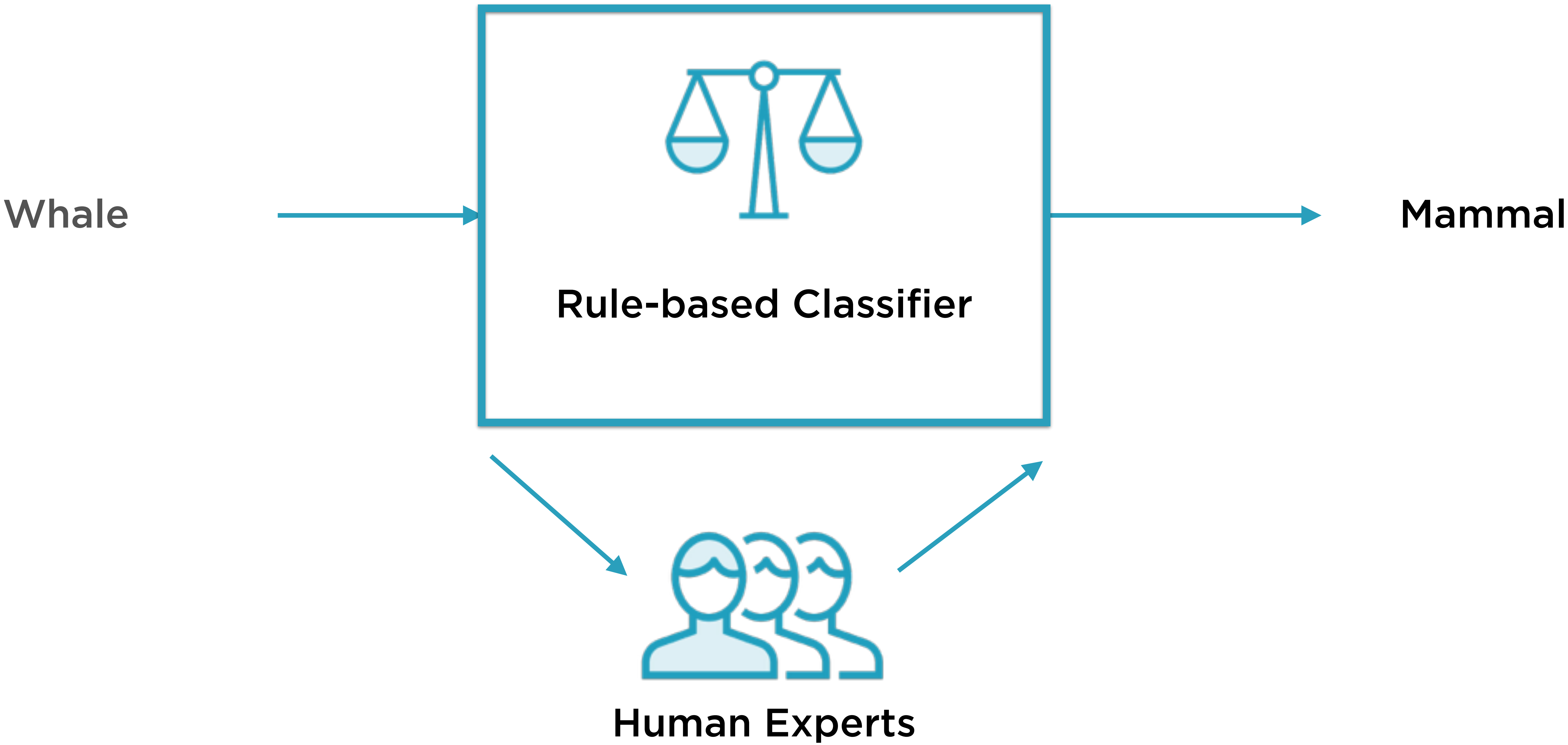
Members of the infraorder *Cetacea*



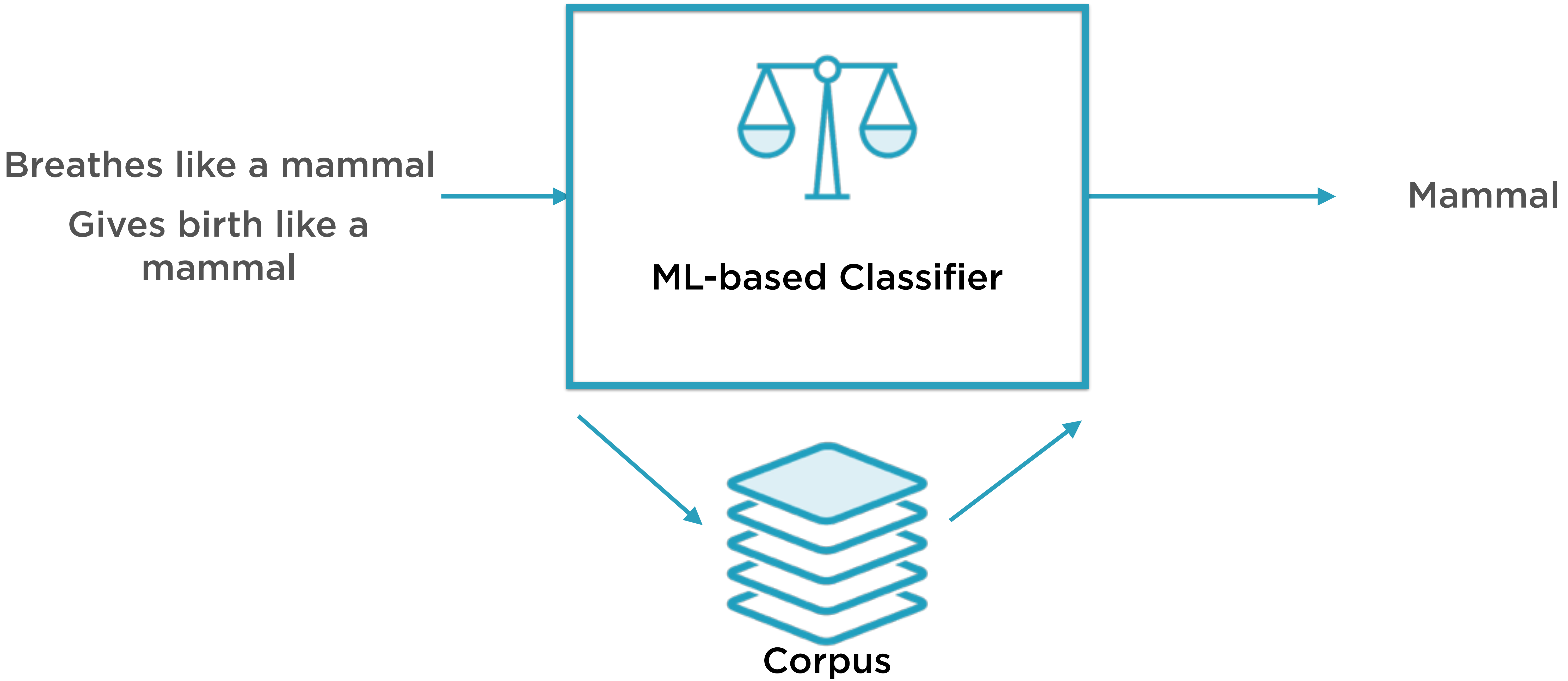
Fish

Look like fish, swim like fish, move with fish

Rule-based Binary Classifier



ML-based Binary Classifier



ML-based Binary Classifier

Corpus

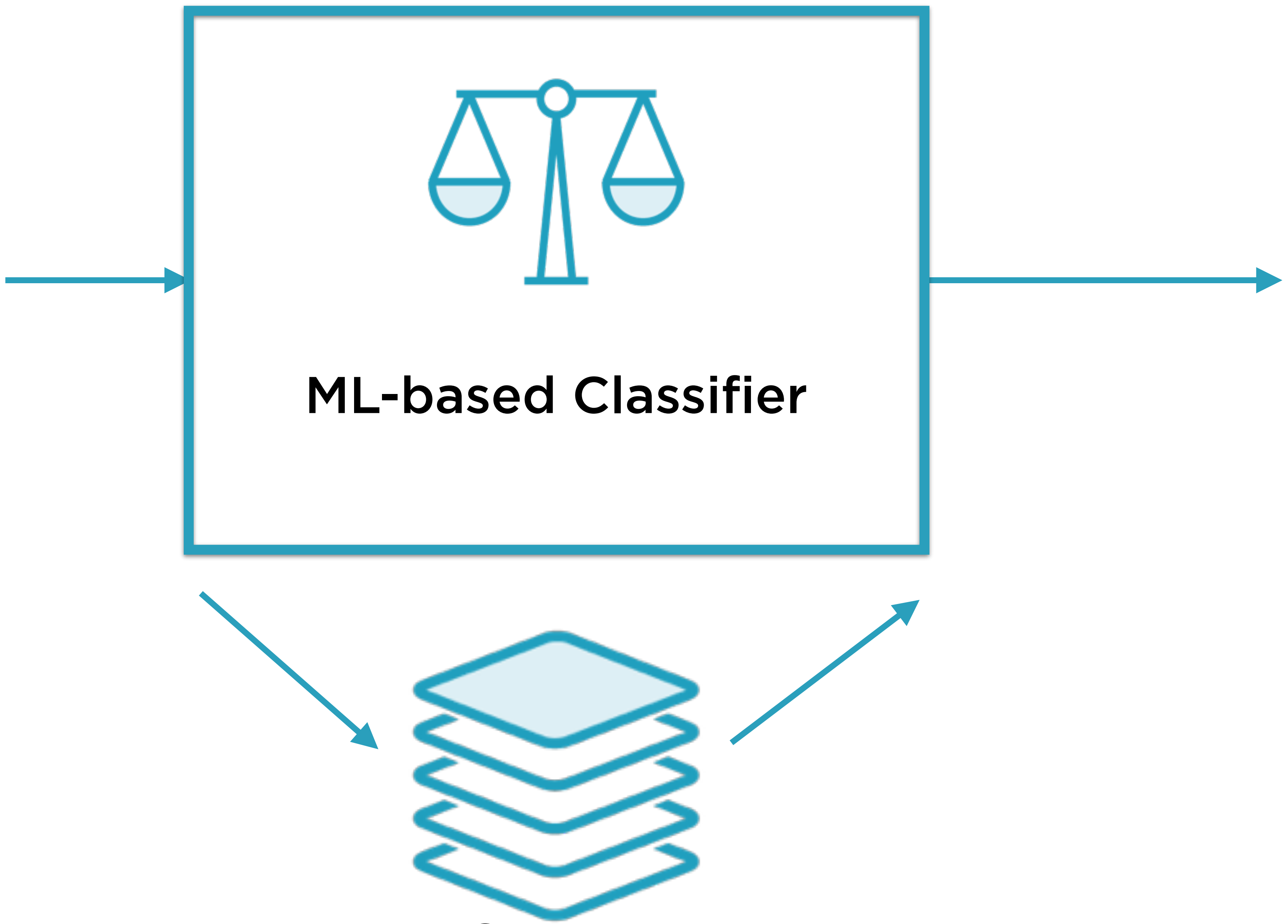
Classification Algorithm

ML-based Classifier

ML-based Binary Classifier

Breathes like a mammal

Gives birth like a mammal



Mammal

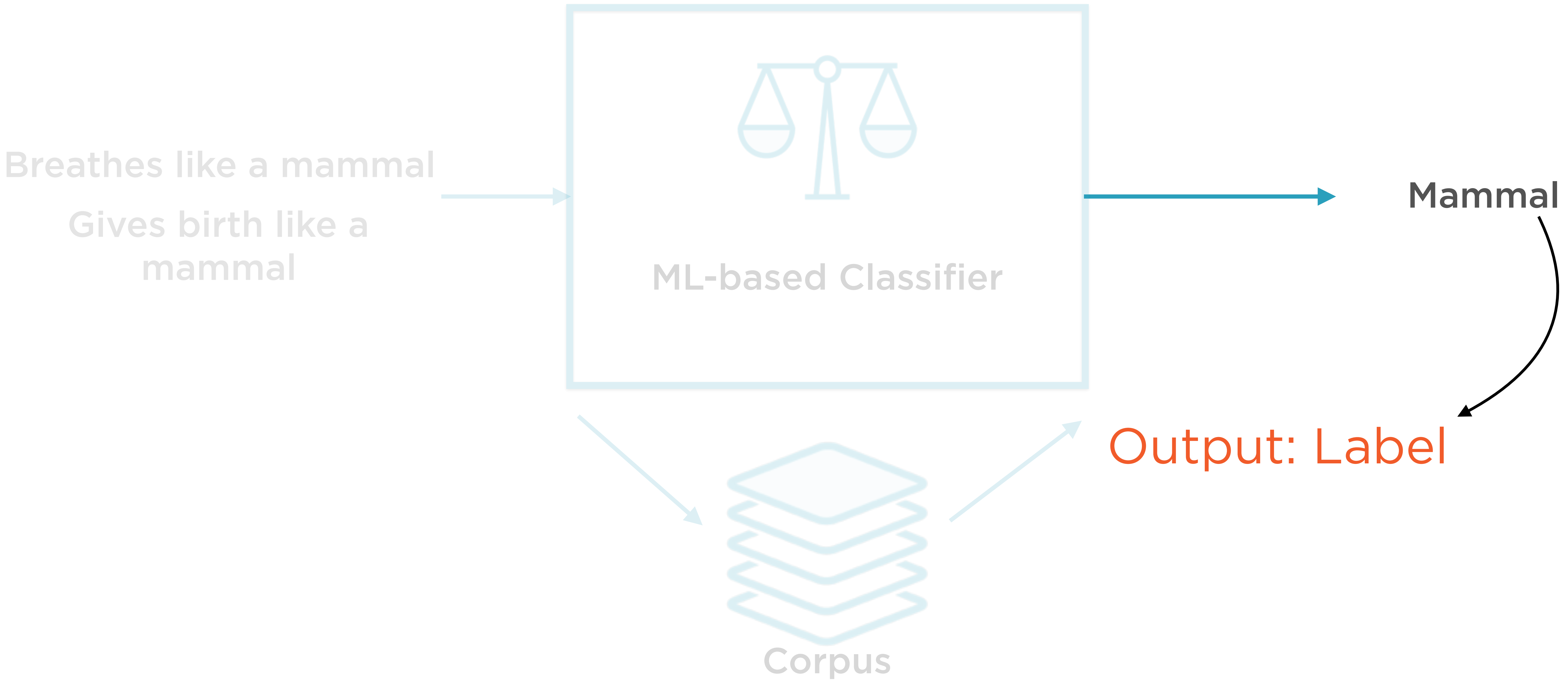
ML-based Binary Classifier

Breathes like a mammal

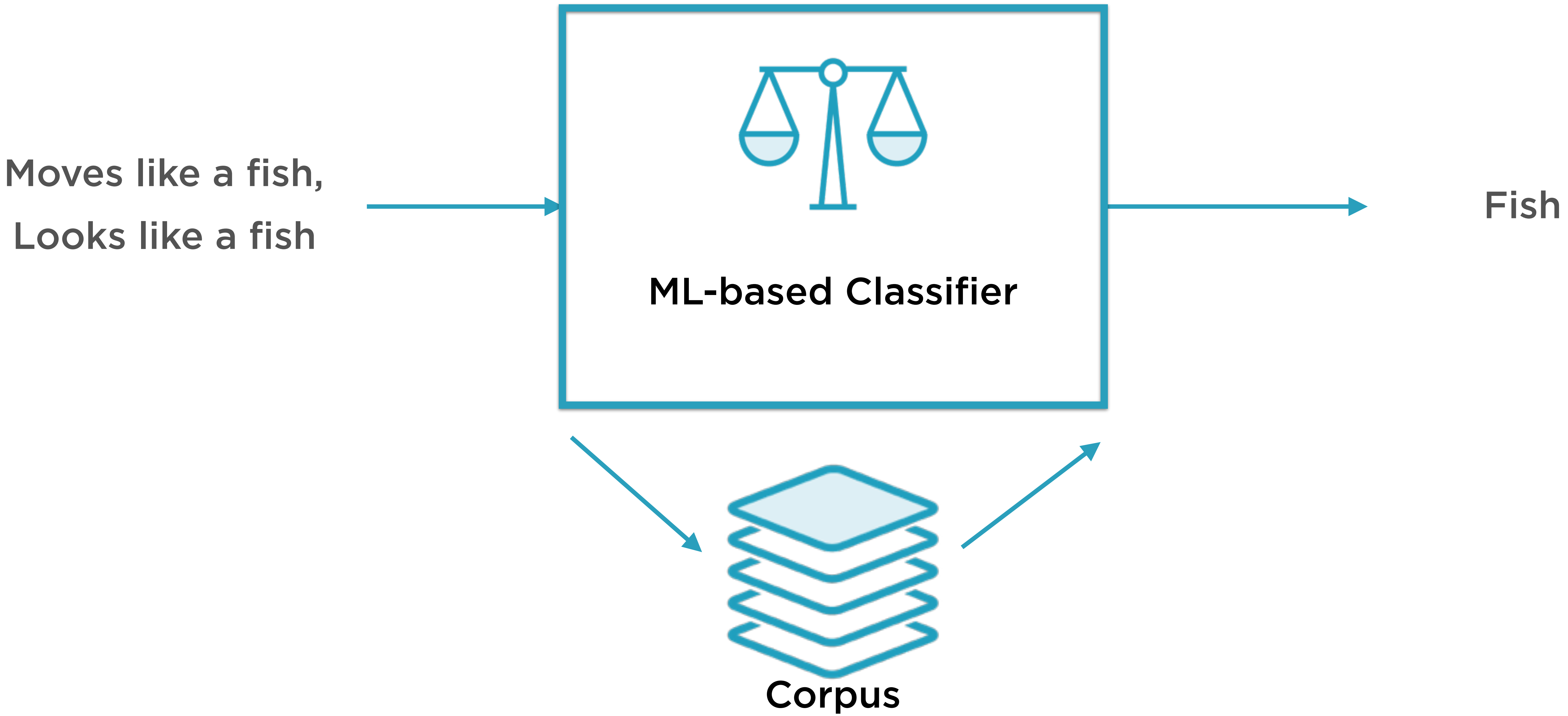
Gives birth like a mammal

Input: Feature Vector

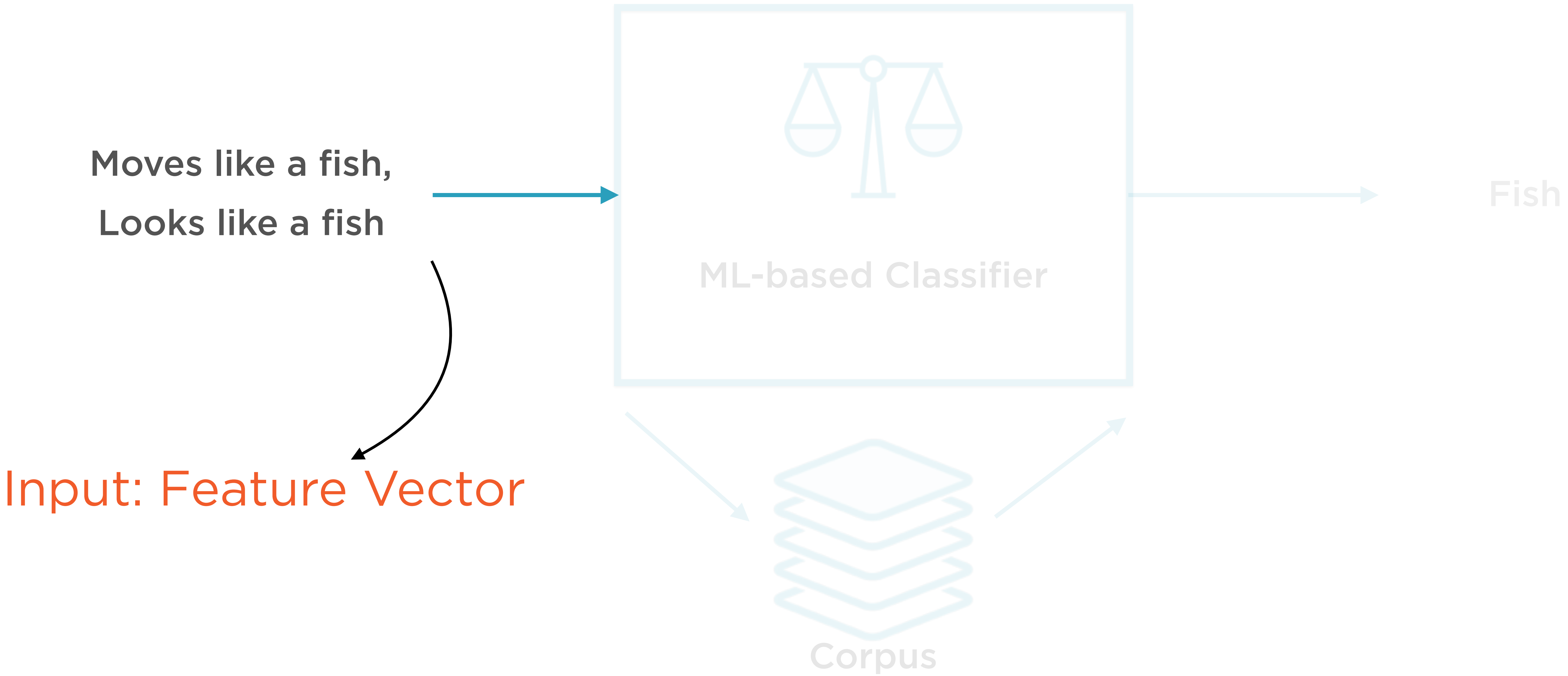
ML-based Binary Classifier



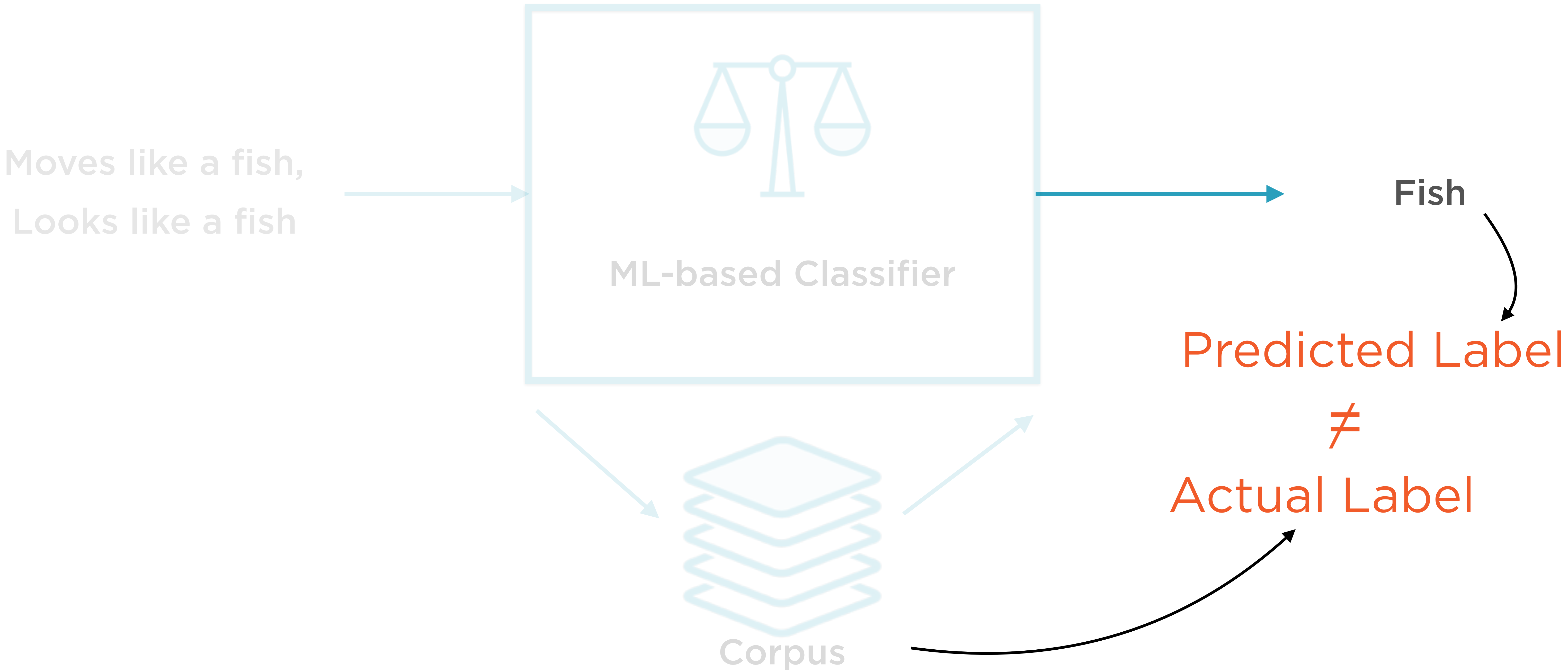
ML-based Binary Classifier



ML-based Binary Classifier



ML-based Binary Classifier



“Traditional” ML-based systems still rely on experts to decide what features to pay attention to

Traditional ML Models

Regression models: Linear, Lasso, Ridge, SVR

Classification models: Naive Bayes, SVMs, Decision trees

“Representation” ML-based systems figure out by themselves what features to pay attention to

Representation ML Models

Deep learning models such as neural networks

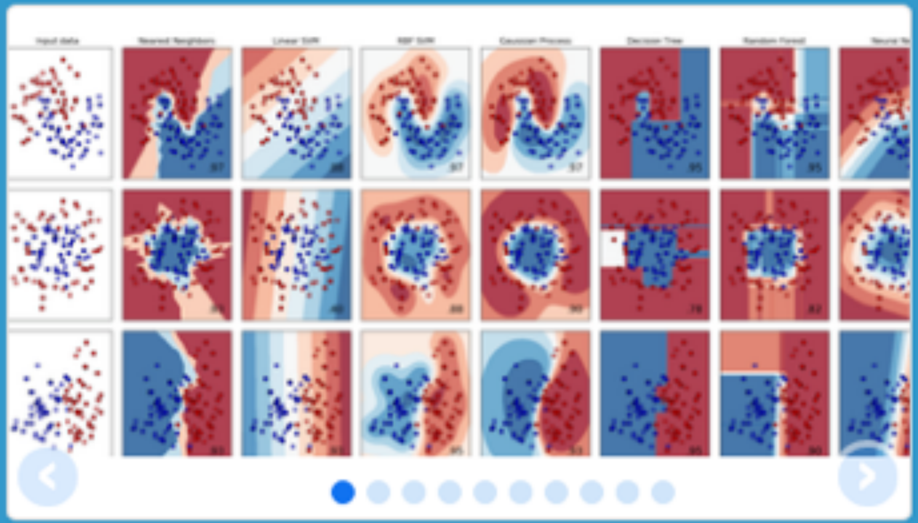
scikit-learn - a popular, open source, Python library

Classification, regression, clustering, dimensionality reduction algorithms scikit-learn: machine learning

scikit-learn.org/stable/

Home Installation Documentation Examples

Google Custom Search Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

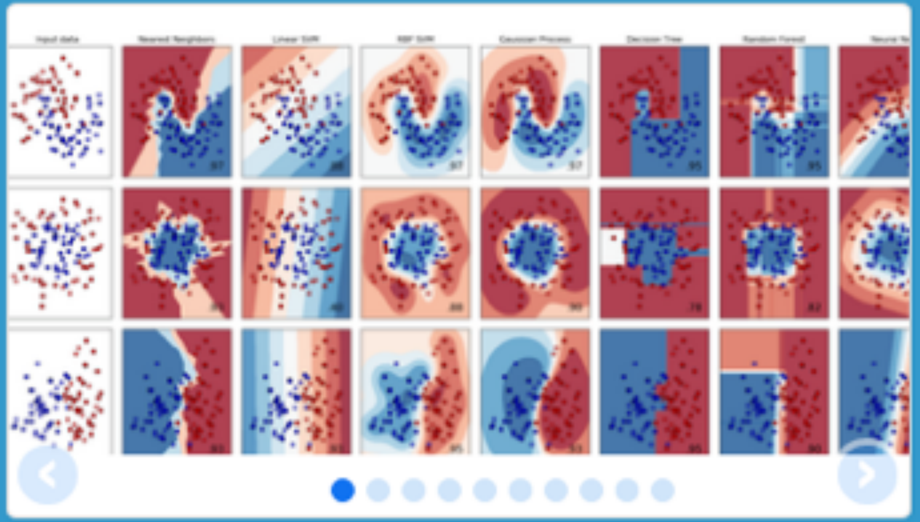
Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

scikit-learn: machine learning

scikit-learn.org/stable/

Home | Installation | Documentation | Examples

Google Custom Search | Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.

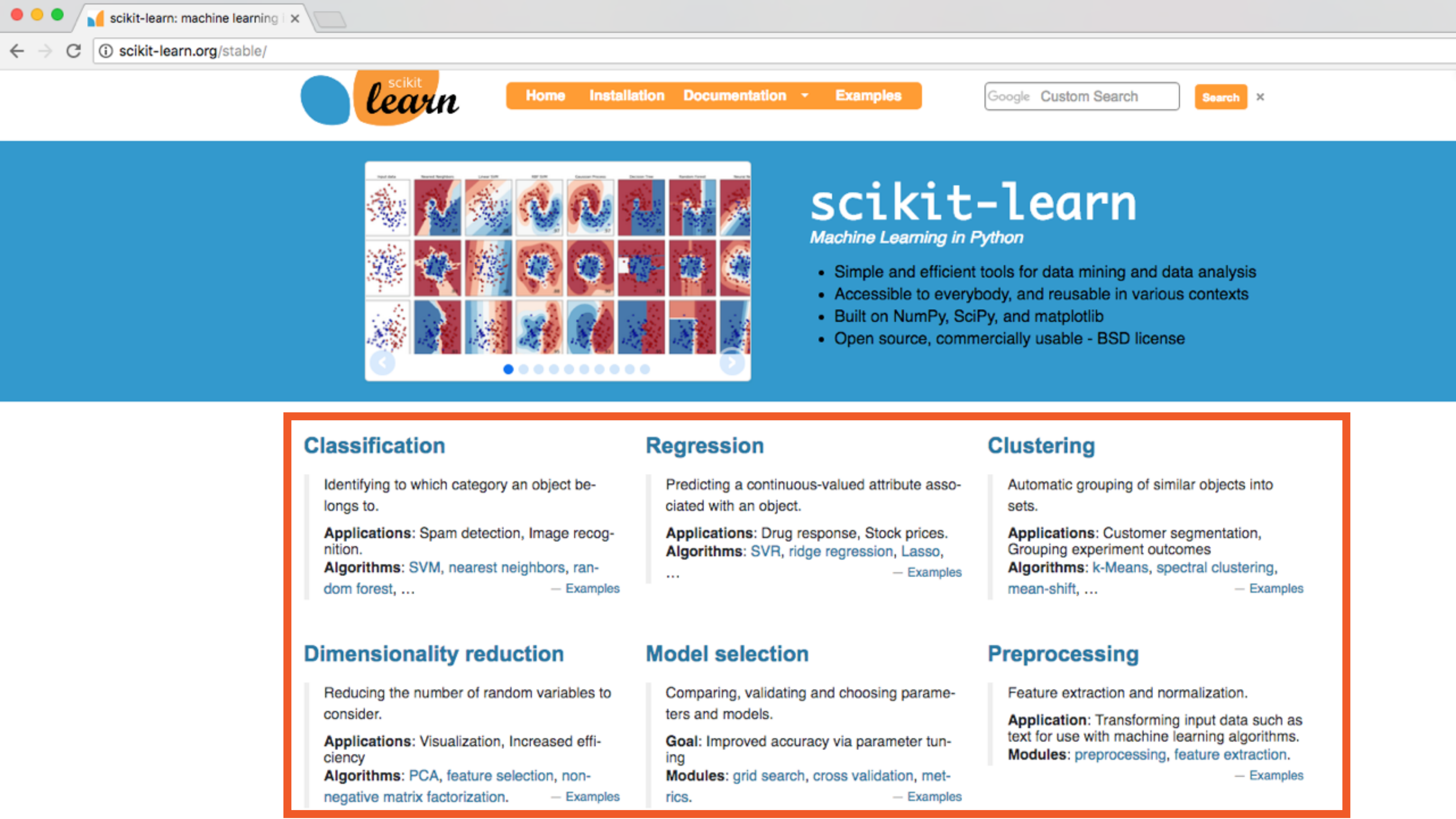
— Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

— Examples


scikit-learn: machine learning
scikit-learn.org/stable/
scikit learn
Home
Installation
Documentation
Examples
Google Custom Search
Search
scikit-learn
Machine Learning in Python
Simple and efficient tools for data mining and data analysis
Accessible to everybody, and reusable in various contexts
Built on NumPy, SciPy, and matplotlib
Open source, commercially usable - BSD license
Classification
Identifying to which category an object belongs to.
Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...
— Examples
Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...
— Examples
Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...
— Examples
Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.
— Examples
Model selection
Comparing, validating and choosing parameters and models.
Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.
— Examples
Preprocessing
Feature extraction and normalization.
Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.
— Examples

Supervised and Unsupervised Learning

Types of ML Algorithms

Supervised

Labels associated with the training data is used to correct the algorithm

Unsupervised

The model has to be set up right to learn structure in the data

Types of ML Algorithms

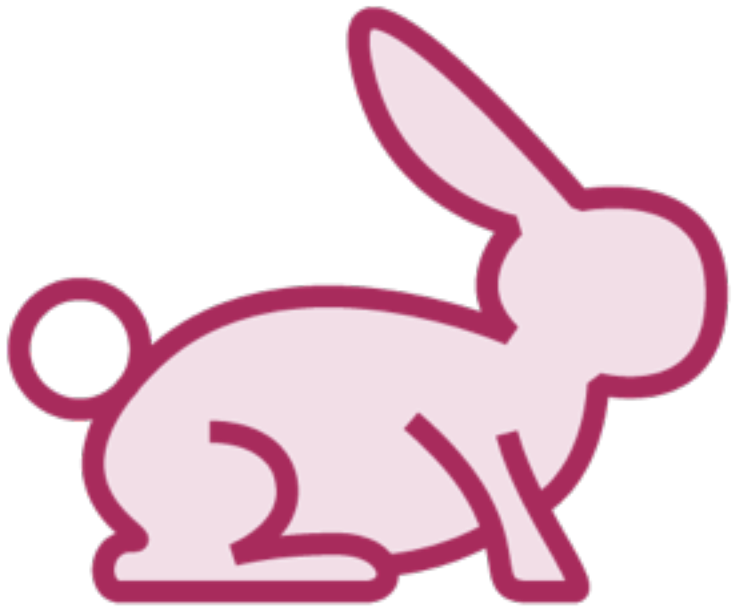
Supervised

Labels associated with the training data is used to correct the algorithm

Unsupervised

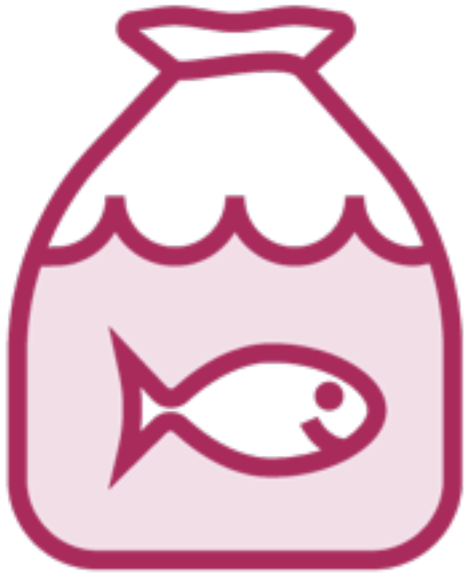
The model has to be set up right to learn structure in the data

Whales: Fish or Mammals?



Mammals

Members of the infraorder *Cetacea*



Fish

Look like fish, swim like fish, move with fish

Whales: Fish or Mammals?

ML-based Classifier

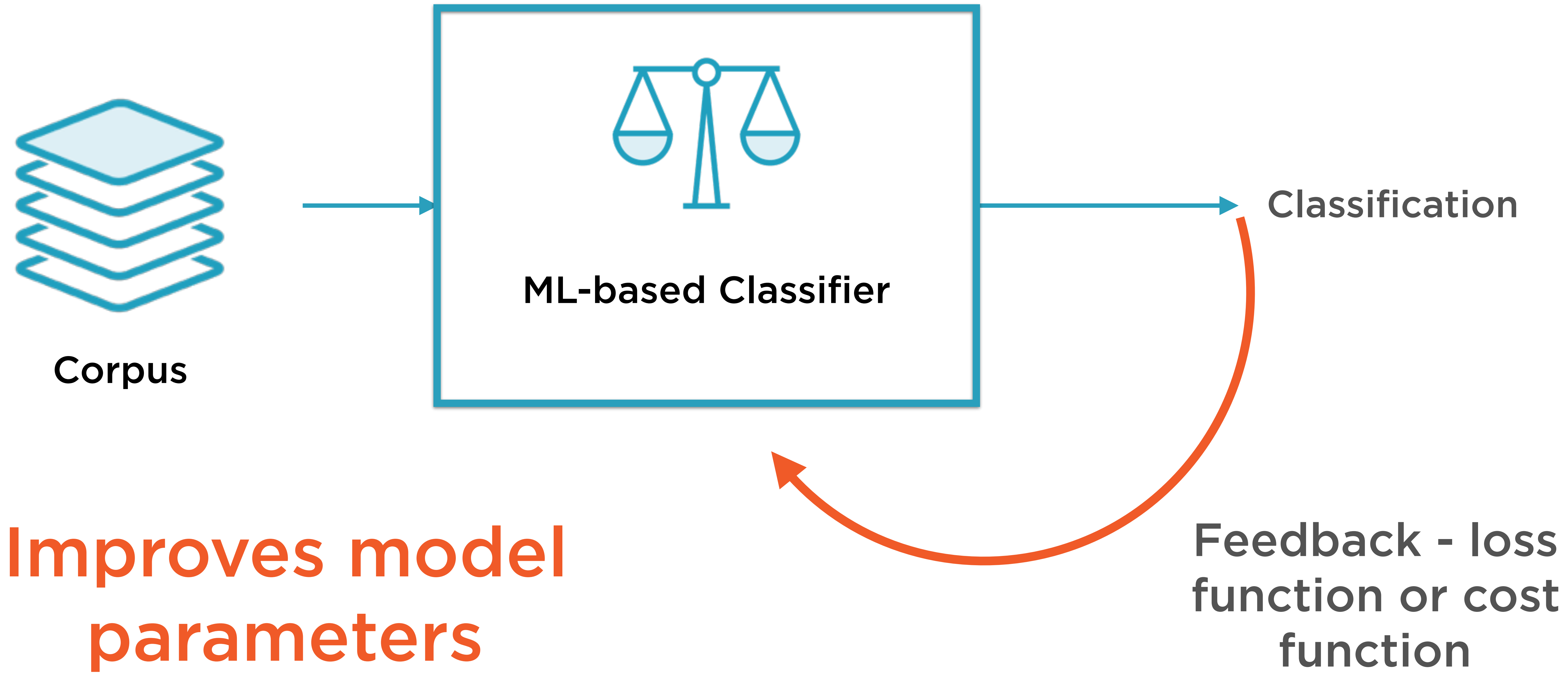
Training

Feed in a large corpus of data classified correctly

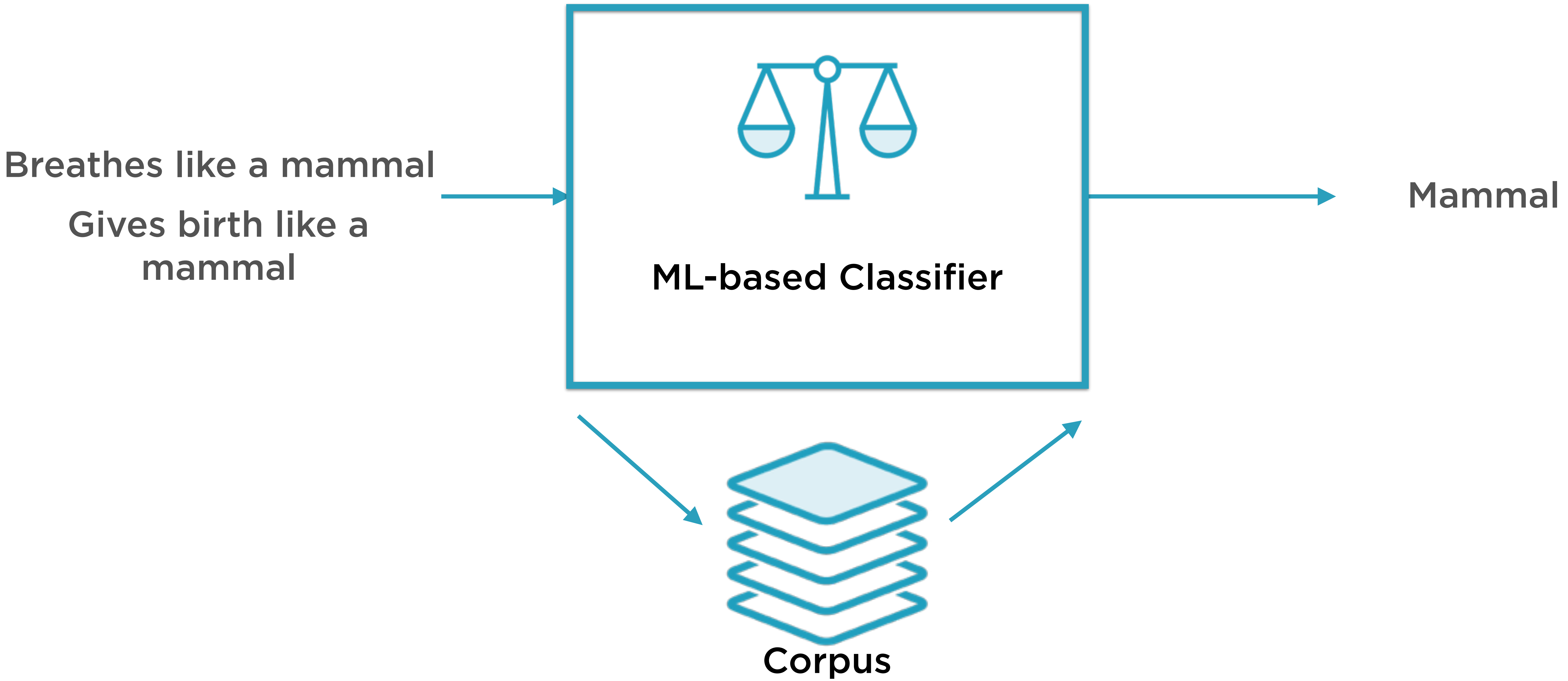
Prediction

Use it to classify new instances which it has not seen before

Training the ML-based Classifier



ML-based Binary Classifier



x Variables

The attributes that the ML algorithm focuses on are called features

Each data point is a list - or vector - of such features

Thus, the input into an ML algorithm is a feature vector

Feature vectors are usually called the x variables

y Variables

The attributes that the ML algorithm tries to predict are called labels

Types of labels

- categorical (classification)
- continuous (regression)

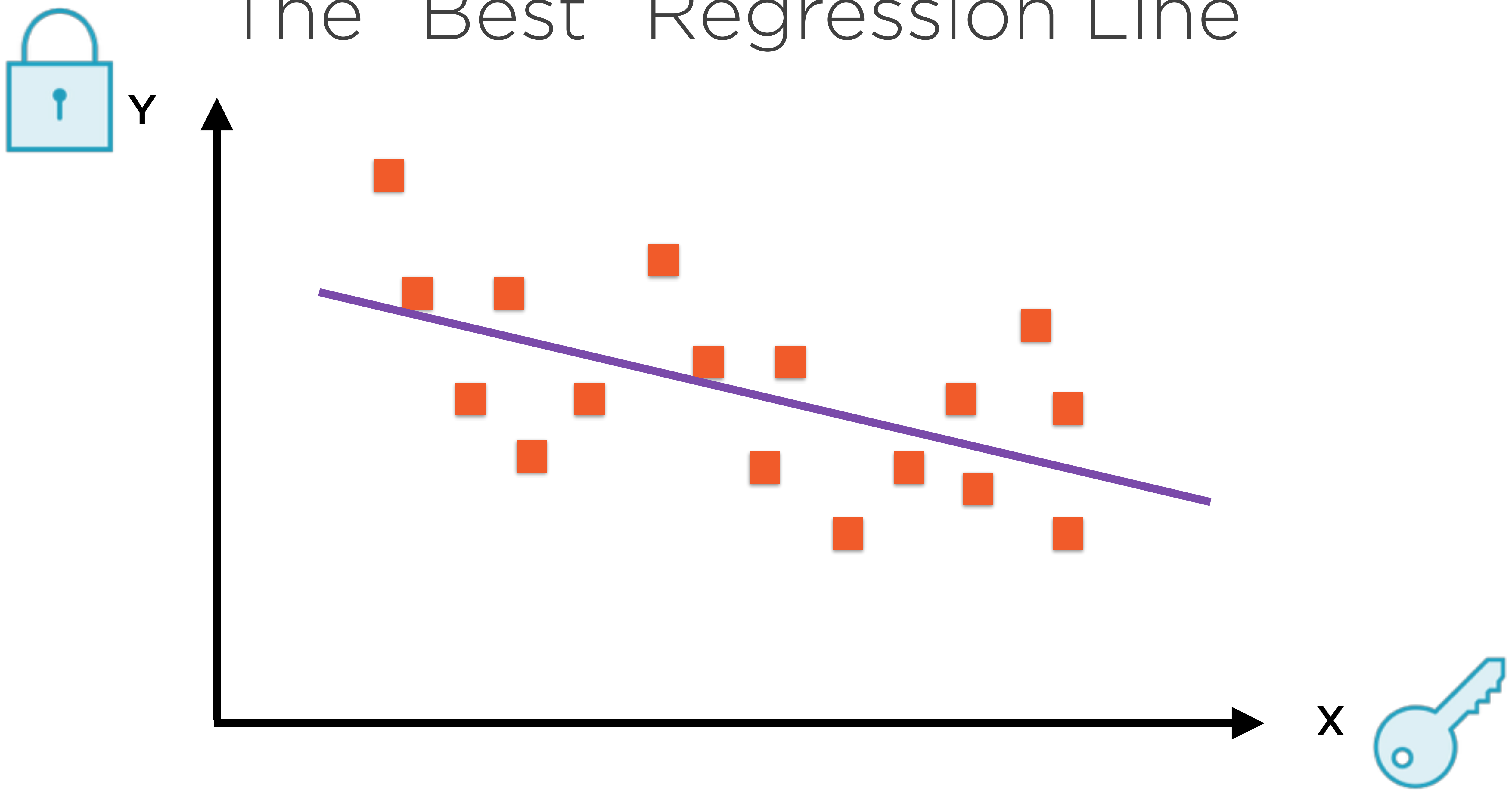
Labels are usually called the y variables

```
y = f(x)
```

Supervised Machine Learning

Most machine learning algorithms seek to “learn” the function f that links the features and the labels

The “Best” Regression Line



Linear Regression involves finding the “best fit” line via a training process

```
y = Wx + b
```

$$f(x) = Wx + b$$

Linear regression specifies, up-front, that the function f is linear

```
def doSomethingReallyComplicated(x1,x2…):
  …
  …
  …
  return complicatedResult
```

f(x) = doSomethingReallyComplicated(x)

ML algorithms such as neural network can "learn" (reverse-engineer) pretty much anything given the right training data

Types of ML Algorithms

Supervised

Labels associated with the training data is used to correct the algorithm

Unsupervised

The model has to be set up right to learn structure in the data

Unsupervised Learning does not have:

- y variables
- a labeled corpus

Supervised Learning

Input variable x and output variable y

Learn the mapping function $y = f(x)$

Approximate the mapping function so for new values of x we can predict y

Use existing dataset to correct our mapping function approximation

Unsupervised Learning

Only have input data x – no output data

Model the underlying structure to learn more about data

Algorithms self discover the patterns and structure in the data

Unsupervised ML Algorithms

Clustering

Identify patterns in data items e.g. K-means clustering

Dimensionality reduction

Identify significant factors that drive data e.g. PCA

Continuous and Categorical Data

Continuous and Categorical Variables

Continuous

Can take an infinite set of values (height, weight, income...)

Categorical

Can take a finite set of values (Male/ Female, Day of week...)

Categorical variables that can take just two values are called binary variables

Continuous and Categorical Variables

Continuous

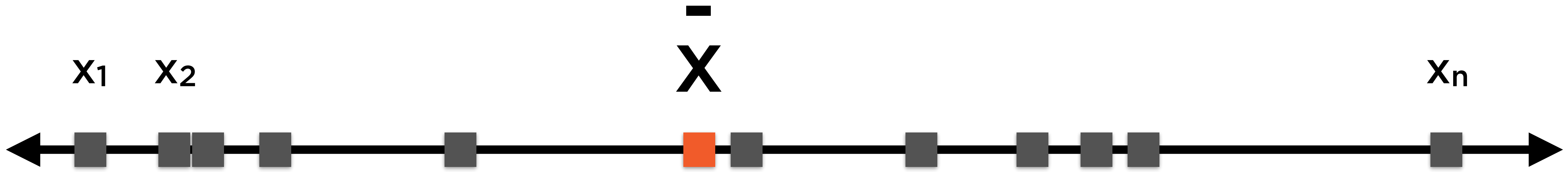
Can take an infinite set of values (height, weight, income...)

Categorical

Can take a finite set of values (Male/ Female, Day of week...)

Standardizing Data: Mean and Variance

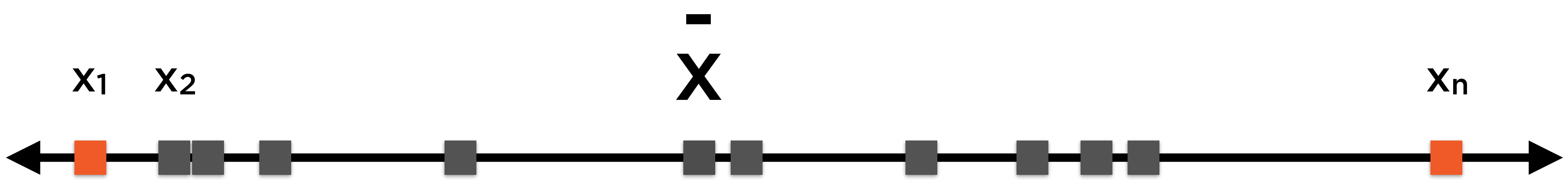
Mean as Headline



The mean, or average, is the one number that best represents all of these data points

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

Variation Is Important Too

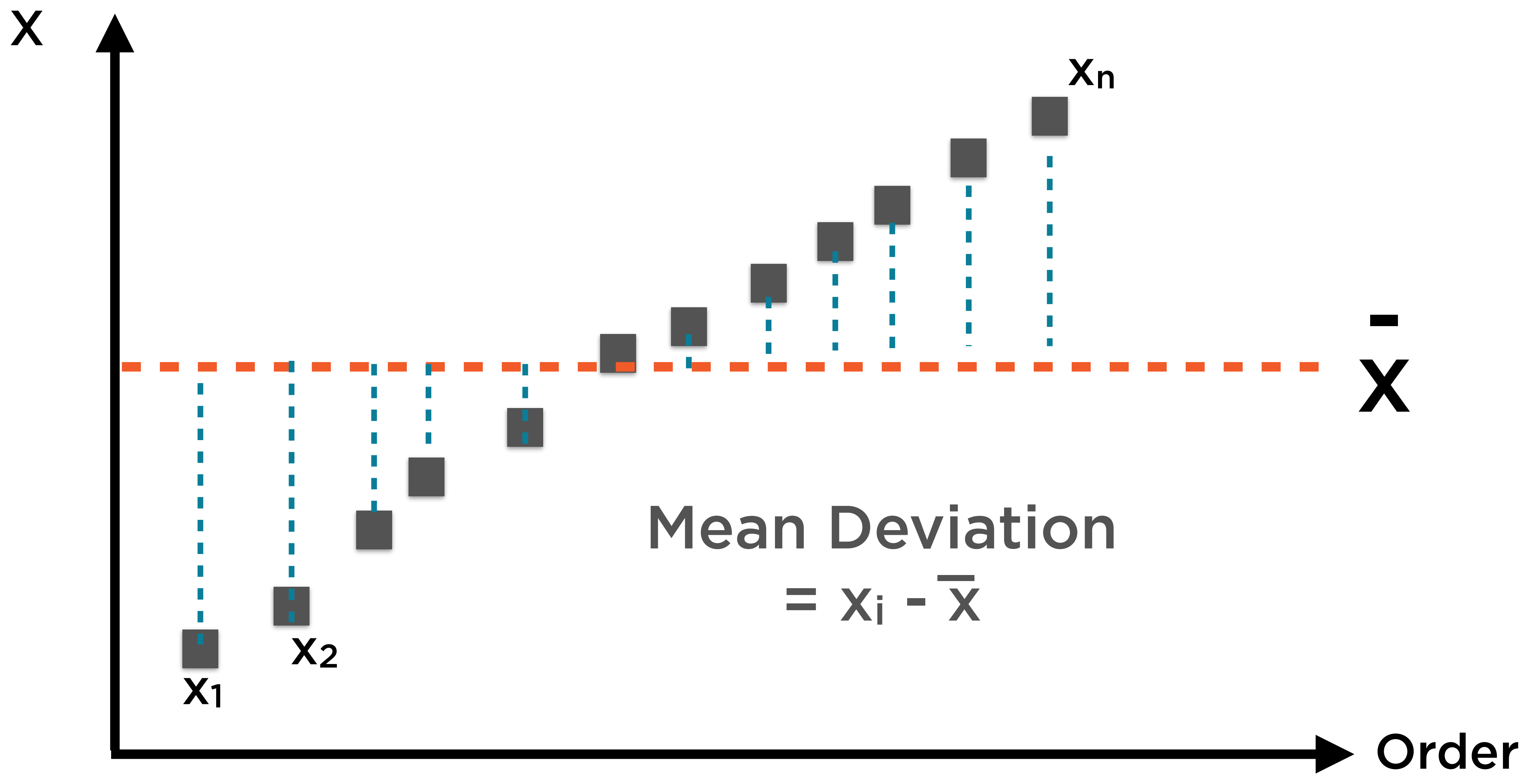


"Do the numbers jump around?"

$$\text{Range} = X_{max} - X_{min}$$

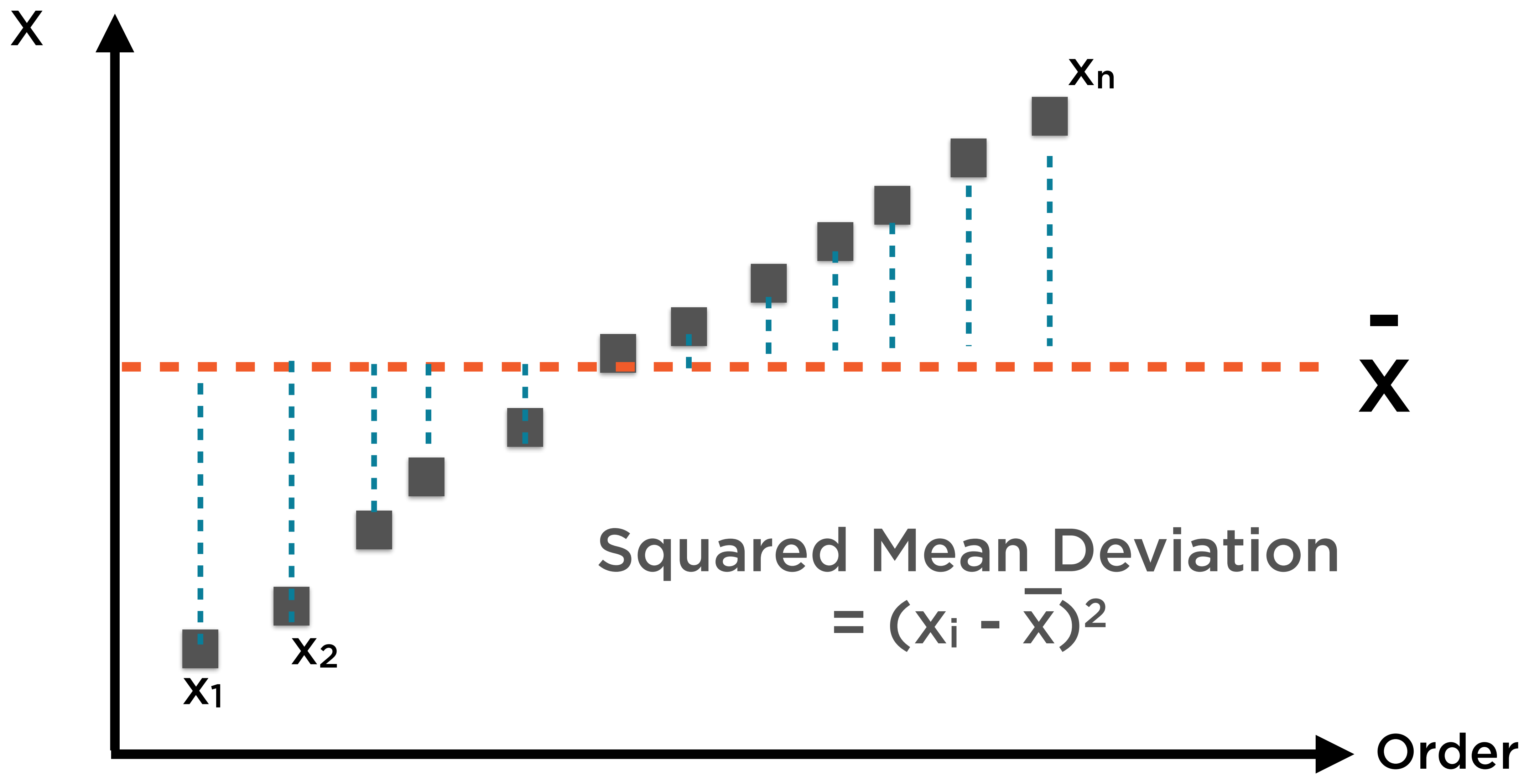
The range ignores the mean, and is swayed by outliers - that's where variance comes in

Variance as Asterisk



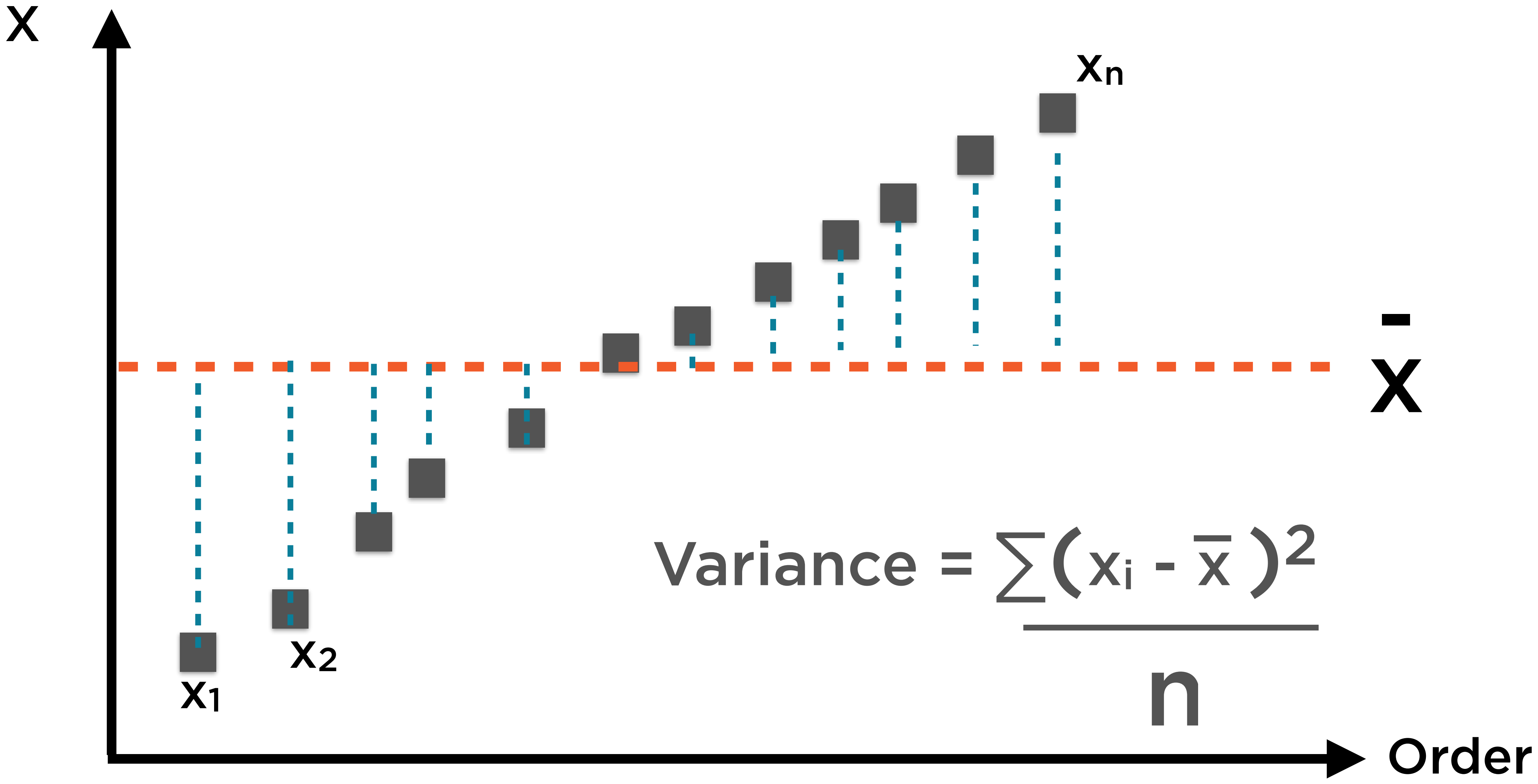
Variance is the second-most important number to summarize this set of data points

Variance as Asterisk



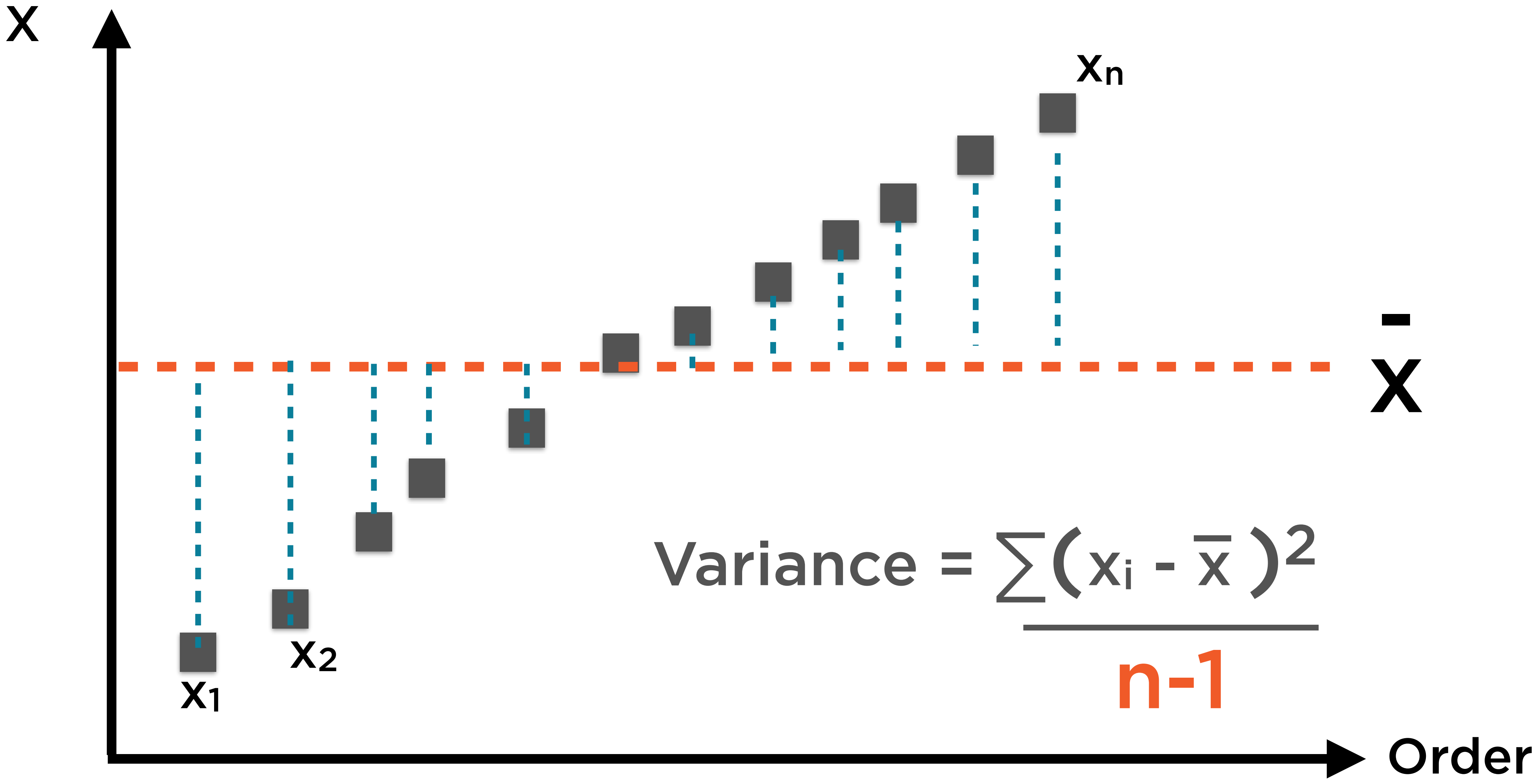
Variance is the second-most important number to summarize this set of data points

Variance as Asterisk



Variance is the second-most important number to summarize this set of data points

Variance as Asterisk



$$\text{Variance} = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

We can improve our estimate of the variance by tweaking the denominator - this is called Bessel's Correction

Mean and Variance

Mean and variance succinctly summarize a set of numbers

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$\text{Variance} = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

Variance and Standard Deviation

Standard deviation is the square root of variance

$$\text{Variance} = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

$$\text{Std Dev} = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

Standardizing Data

$$\begin{bmatrix} \boxed{\begin{matrix} X_{11} \\ X_{21} \\ X_{31} \\ \cdots \\ X_{n1} \end{matrix}} & \cdots & \boxed{\begin{matrix} X_{1k} \\ X_{2k} \\ X_{3k} \\ \cdots \\ X_{nk} \end{matrix}} \end{bmatrix}$$

$$\begin{matrix} \text{avg}(X_1) & \cdots & \text{avg}(X_k) \\ \text{stdev}(X_1) & \cdots & \text{stdev}(X_k) \end{matrix}$$

Standardizing Data

$$\begin{bmatrix} \dfrac{x_{11} - \mathrm{avg}(X_1)}{\mathrm{stdev}(X_1)} & \cdots & \dfrac{x_{1k} - \mathrm{avg}(X_k)}{\mathrm{stdev}(X_k)} \\ \cdots & & \cdots \\ \dfrac{x_{n1} - \mathrm{avg}(X_1)}{\mathrm{stdev}(X_1)} & & \dfrac{x_{nk} - \mathrm{avg}(X_k)}{\mathrm{stdev}(X_k)} \end{bmatrix}$$

Each column of the standardized data has mean 0 and variance 1

Standardized Data

Many techniques work best on standardized data

Standardization prevents some (high-variance) data series from dominating

Examples:

- Principal Components Analysis
- Lasso/Ridge Regression

Continuous and Categorical Variables

Continuous

Can take an infinite set of values (height, weight, income...)

Categorical

Can take a finite set of values (Male/ Female, Day of week...)

Categorical Data

Continuous data can be ordered, categorical data can not

ML algorithms only operate on numbers

Categorical data need to be encoded as numbers

Numerical encodings of categorical data should never be ordered

Categorical Data

- Continuous data can be ordered, categorical data can not
- ML algorithms only operate on numbers
- Categorical data need to be encoded as numbers
- Numerical encodings of categorical data should never be ordered

One-hot Encoding

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

One-hot Encoding

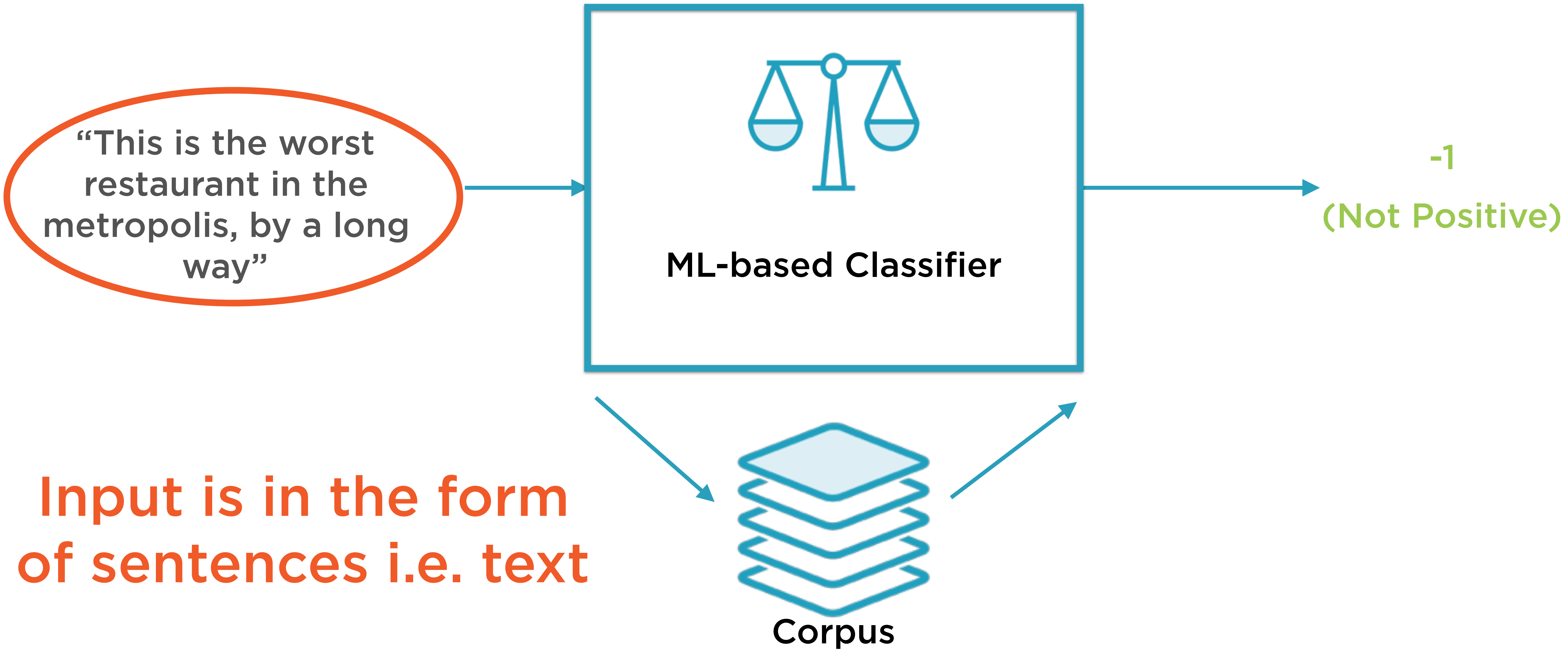
	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Monday	0	1	0	0	0	0	0
Thursday	0	0	0	0	1	0	0
Saturday	0	0	0	0	0	0	1

Demo

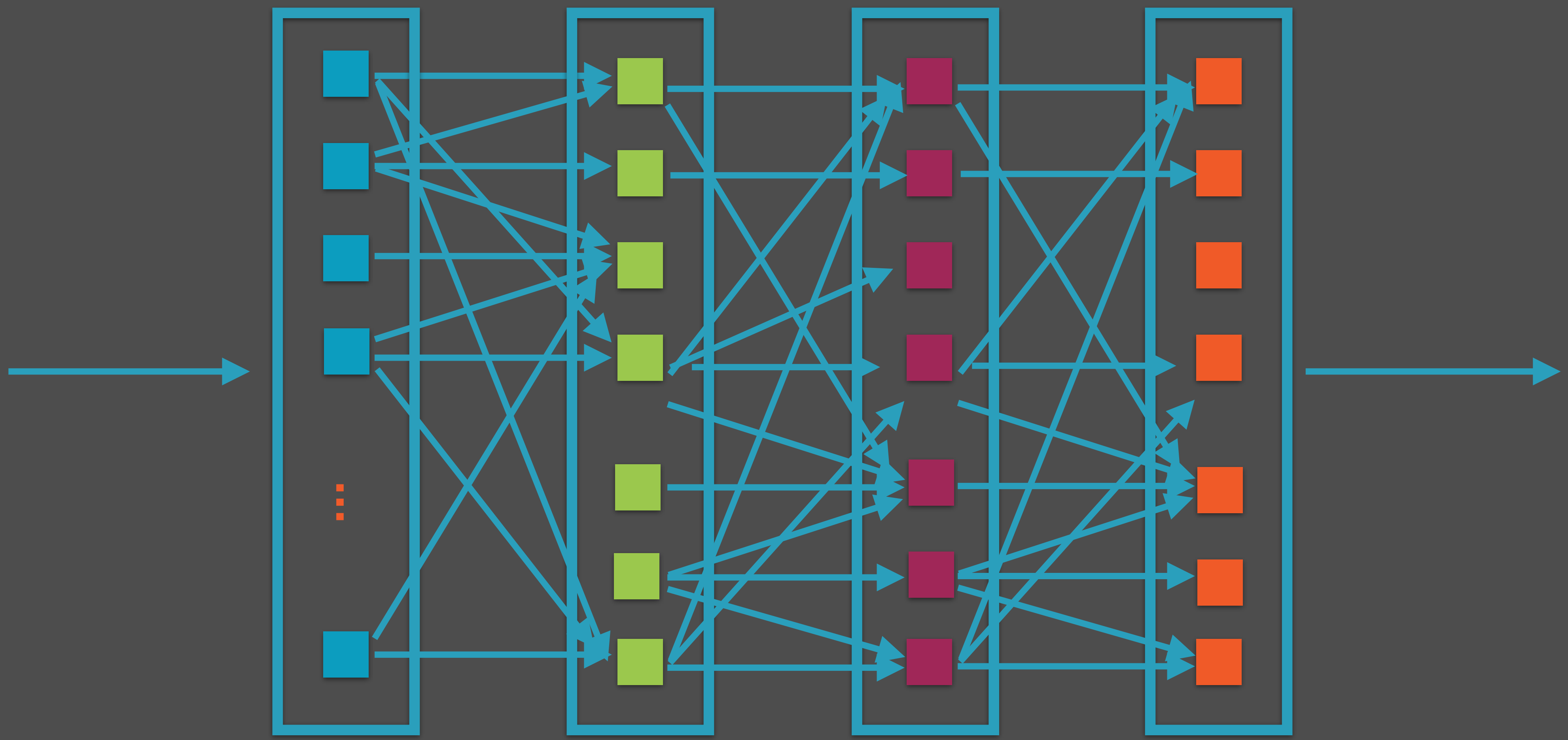
Working with numeric and categorical data

Encoding Text Data in Numeric Form

Sentiment Analysis Using Neural Networks



Input is in the form of sentences i.e. text



Neural networks only process **numeric input**, they don’t work with plain text

```
d = “This is not the worst restaurant in the metropolis,
not by a long way”
```

Document as Word Sequence

Model a document as an ordered sequence of words

```
d = “This is not the worst restaurant in the metropolis,
not by a long way”

(“This”, “is”,”not”,”the”,”worst”,”restaurant”,”in”,”the”,
“metropolis”, “not”,”by”,”a”,”long”,”way”)
```

Document as Word Sequence

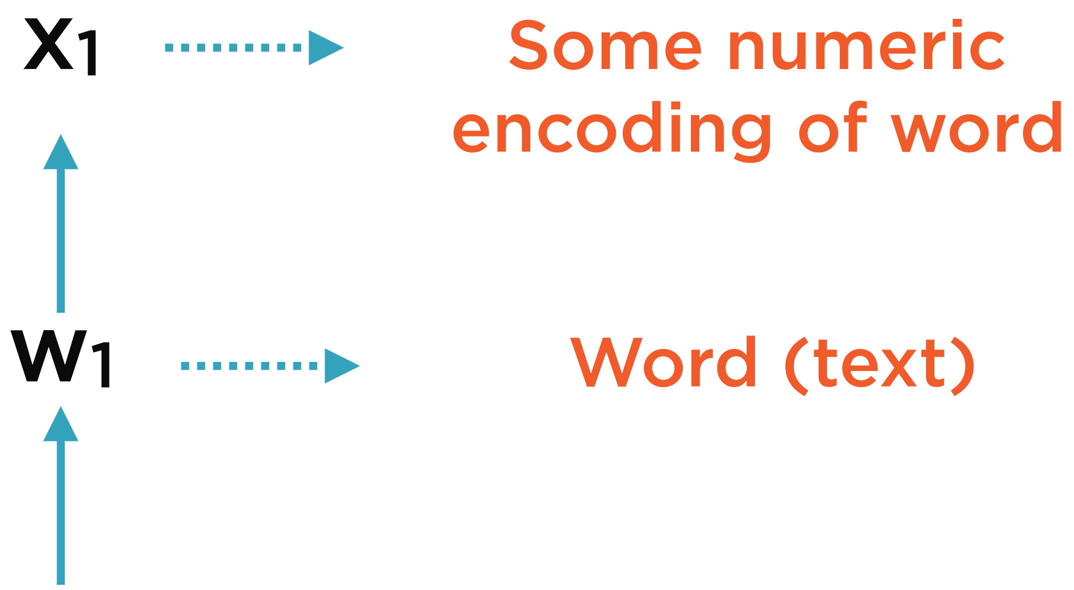
Tokenise document into individual words

x_0 → Some numeric encoding of word

w_0 → Word (text)

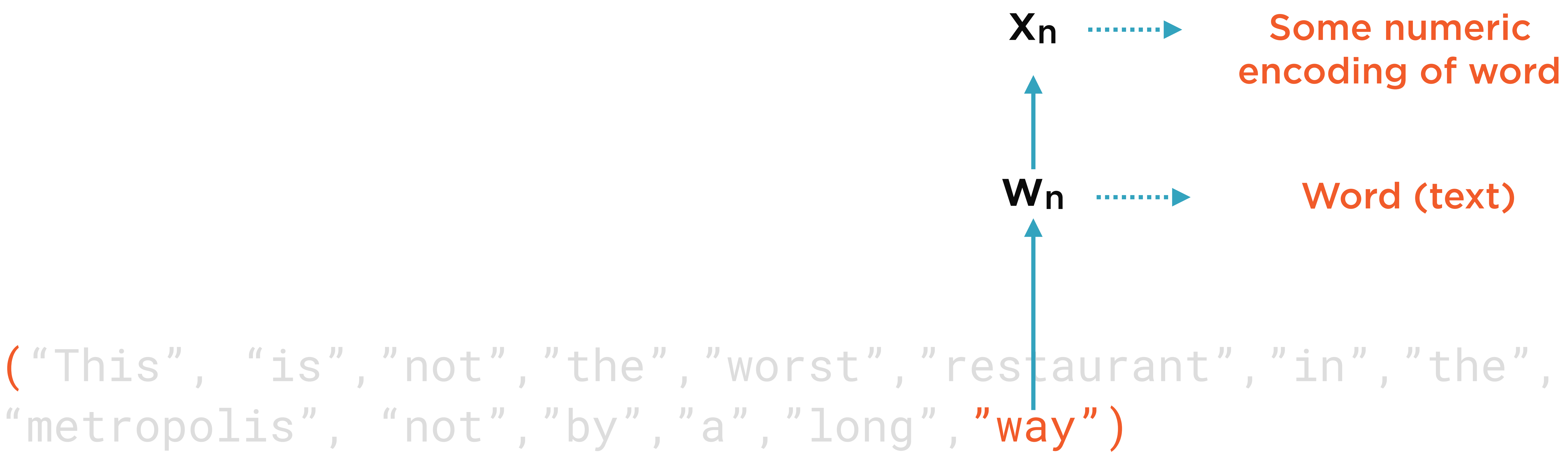
("This", "is","not","the","worst","restaurant","in","the", "metropolis", "not","by","a","long","way")

Represent Each Word as a Number



(“This”, “is”,”not”,”the”,”worst”,”restaurant”,”in”,”the”, “metropolis”, “not”,”by”,”a”,”long”,”way”)

Represent Each Word as a Number



Represent Each Word as a Number

$$d = [x_0, x_1, \dots x_n]$$

Document as Tensor

Represent each word as numeric data, aggregate into tensor

```
xi = [?]
```

The Big Question

How best can words be represented as numeric data?

```
d = [[?], [?], … [?]]
```

The Big Question

How best can words be represented as numeric data?

Word Embeddings

One-hot

Frequency-based

Prediction-based

Word Embeddings

One-hot

Frequency-based

Prediction-based

Word Embeddings

Numerical representations of text which capture meanings and semantic relationships

Not covered in this course

Word Embeddings

One-hot

Frequency-based

Prediction-based

Documents and Corpus

Reviews

Reviews
Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

D = Entire corpus

d_i = One document in corpus

One-hot Encoding

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

All Words

amazing
worst
movie
ever
two
thumbs
up
Part
was
bad
3
the
there
with
greats

Create a set of all words (all across the corpus)

One-hot Encoding

	Amazing!	Worst movie ever	Two thumbs up
amazing	1	0	0
worst	0	1	1
movie	0	1	1
ever	0	1	1
two	0	0	1
thumbs	0	0	1
up	0	0	1
Part	0	0	0
was	0	0	0
bad	0	0	0
3	0	0	0
the	0	0	0
there	0	0	0
with	0	0	0
greats	0	0	0

Express each review as a tuple of 1,0 elements

One-hot Encoding

	Amazing!	Worst movie ever	Two thumbs up
amazing	1	0	0
worst	0	1	1
movie	0	1	1
ever	0	1	1
two	0	0	1
thumbs	0	0	1
up	0	0	1
Part	0	0	0
was	0	0	0
bad	0	0	0
3	0	0	0
the	0	0	0
there	0	0	0
with	0	0	0
greats	0	0	0

Express each review as a tuple of 1,0 elements

One-hot Encoding

	Amazing!	Worst movie ever	Two thumbs up
amazing	1	0	0
worst	0	1	1
movie	0	1	1
ever	0	1	1
two	0	0	1
thumbs	0	0	1
up	0	0	1
Part	0	0	0
was	0	0	0
bad	0	0	0
3	0	0	0
the	0	0	0
there	0	0	0
with	0	0	0
greats	0	0	0

Express each review as a tuple of 1,0 elements

Flaws of One-hot Encoding

Large vocabulary - enormous feature vectors

Unordered - Lost all context

Binary - Lost frequency information

One-hot encoding does NOT capture any semantic information or relationship between words

Frequency-based Embedding

Word Embeddings

One-hot

Frequency-based

Prediction-based

Word Embeddings

One-hot

Frequency-based

Prediction-based

Frequency-based Embeddings

Count

TF-IDF

Frequency-based Embeddings

Count

TF-IDF

Capture how often a word occurs in a document i.e. the **counts** or the **frequency**

```
d1 = "The movie was bad"

d2 = "The actors were bad, sets were bad"
```

Document as Word Sequence

Model a document as an ordered sequence of words

```
d1 = “The movie was bad”
(“The”, “movie”,”was”,”bad”)

d2 = “The actors were bad, sets were bad”
(“The”, “actors”,”were”,”bad”, “sets”, “were”, bad)
```

Document as Word Sequence

Tokenize the document into words

Count Vector Encoding

Reviews
The movie was bad
The actors were bad, sets were bad

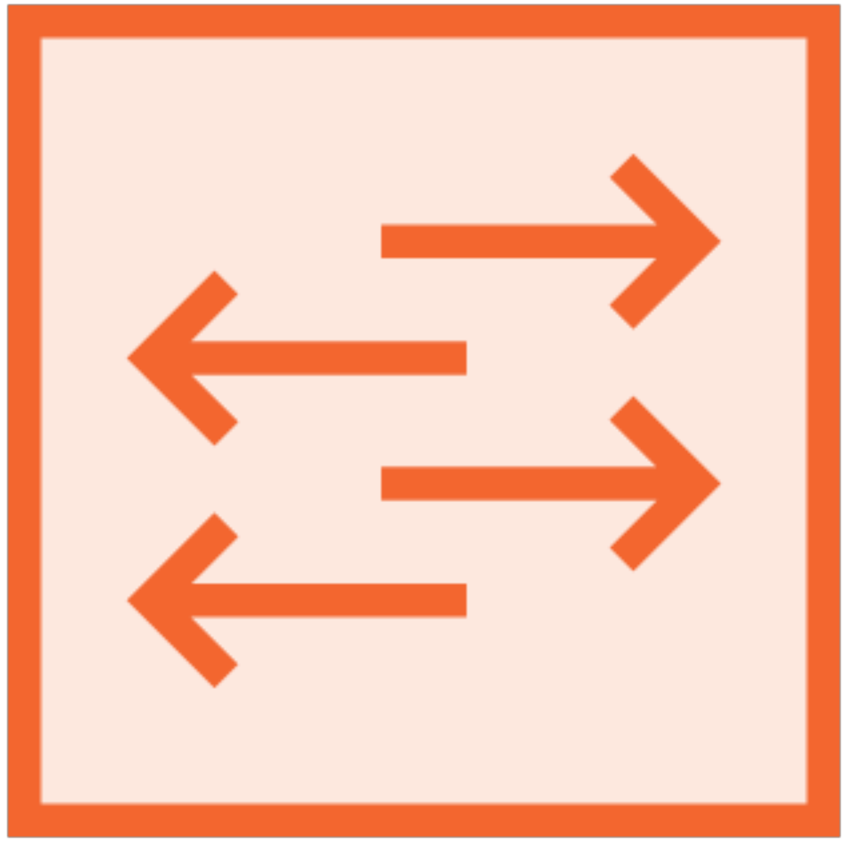
All Words
the
movie
was
bad
actors
were
sets

Create a set of all words (all across the corpus)

Count Vector Encoding

	d1: The movie was bad	d2: The actors were bad, sets were bad
the	1	1
movie	1	1
was	1	1
bad	1	2
actors	0	1
were	0	1
sets	0	1

Express each review as a frequency of the words which appear in that review



Sparse Vectors

Large vocabulary - enormous feature vectors

Alternative: Choose only the top N words based on frequency

Flaws of Count Vectors

Large vocabulary - enormous feature vectors

Unordered - lost all context

Semantics and word relationships lost

Flaws of Count Vectors

Large vocabulary - enormous feature vectors

Unordered - lost all context

Semantics and word relationships lost

Hash words to buckets to have a fixed vocabulary size

Choose enough buckets so that collisions are rare

Hash Encoding

	d1: The movie was bad	d2: The actors were bad, sets were bad
the	1	1
movie	1	0
was	1	0
bad	1	2
$\boxed{\text{actors}}$	0	1
were	0	2
$\boxed{\text{sets}}$	0	1

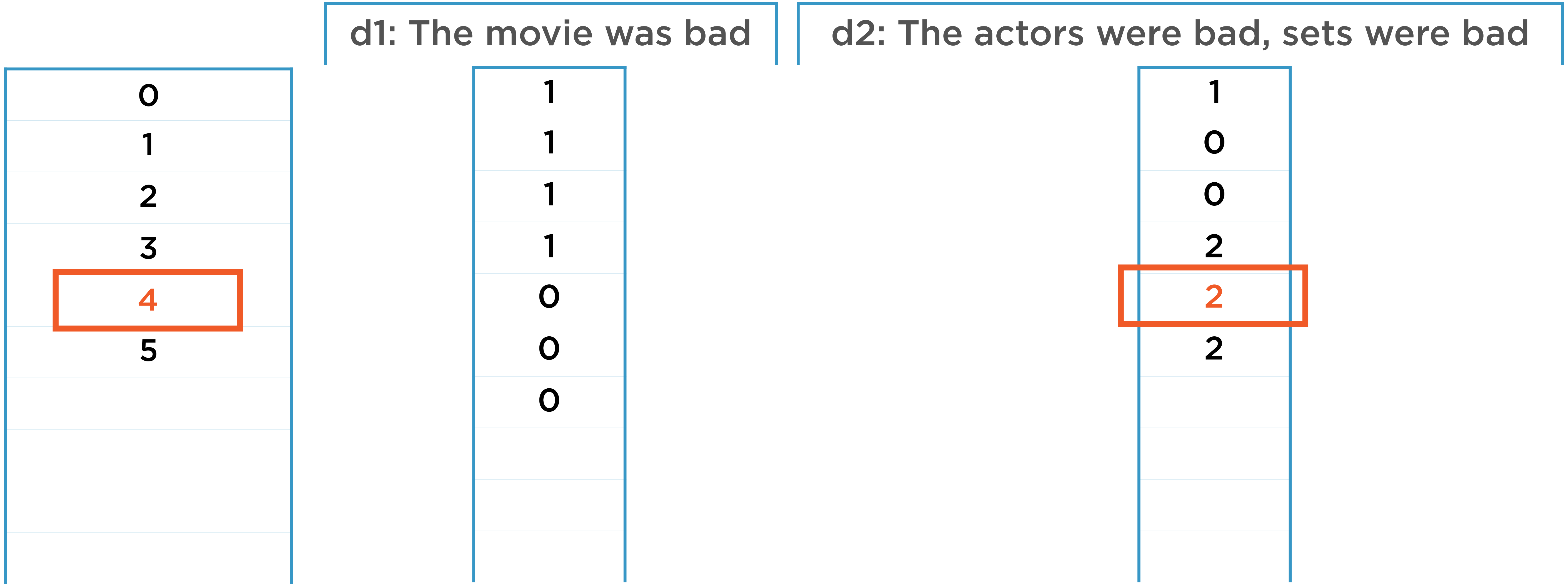
Suppose the words “actors” and “sets” hashed to the same bucket (represented by an integer)

Hash Encoding

	d1: The movie was bad	d2: The actors were bad, sets were bad
0	1	1
1	1	0
2	1	0
3	1	2
4	0	1
5	0	2
4	0	1

Suppose the words "actors" and "sets" hashed to the **same** bucket (represented by an integer)

Hash Encoding



Suppose the words “actors” and “sets” hashed to the **same** bucket (represented by an integer)

Frequency-based Embeddings

Count

TF-IDF

Captures how often a word occurs in a **document** as well as the **entire corpus**

x_0 → Some numeric encoding of word

w_0 → Word (text)

("This", "is","not","the","worst","restaurant","in","the", "metropolis", "not","by","a","long","way")

Document as Word Sequence

Tokenise document into words

$$d = [x_0, x_1, \ldots x_n]$$

Document as Tensor

Represent each word as numeric data, aggregate into tensor

$$x_i = tf(w_i) \times idf(w_i)$$

Tf-Idf

Tf = Term Frequency; Idf = Inverse Document Frequency

Tf-Idf

Frequently in a single document

Might be important

Frequently in the corpus

Probably a common word like “a”, “an”, “the”

Documents and Corpus

Reviews

Reviews
Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

D = Entire corpus
d_i = One document in corpus

$$x_{i,j} = tf(w_i, d_j) \times idf(w_i, D)$$

Tf-Idf

Encoding of word i in document j depends on word, document and also on entire corpus

$$x_{i,j} = tf(w_i, d_j) \times idf(w_i, D)$$

Tf = Term Frequency

Measure of how frequently word i occurs in document j

$$x_{i,j} = tf(w_i,\ d_j)\ \times\ idf(w_i,\ D)$$

Idf = Inverse Document Frequency

Measure of how infrequently word i occurs in corpus D

$$x_{i,j} = tf(w_i, d_j) \times idf(w_i, D)$$

Tf-Idf

High weight for word i in document j if word occurs a lot in this document, but rarely elsewhere

Evaluating Tf-Idf

Important advantages

- Feature vector much more tractable in size
- Frequency and relevance captured

One big drawback

- Context still not captured

Demo

Representing text data in numerical form

- CountVectorizer
- TfidfVectorizer
- HashingVectorizer

```
vectorizer.fit(<data>)
```

Generate Unique IDs for Words in Corpus

Every word in the corpus is given a unique integer ID

```
vectorizer.transform(<data>)
```

Assign the Generated IDs to Corpus

The word IDs generated using fit() are now applied to the corpus passed in to transform

```
vectorizer.fit_and_transform(<data>)
```

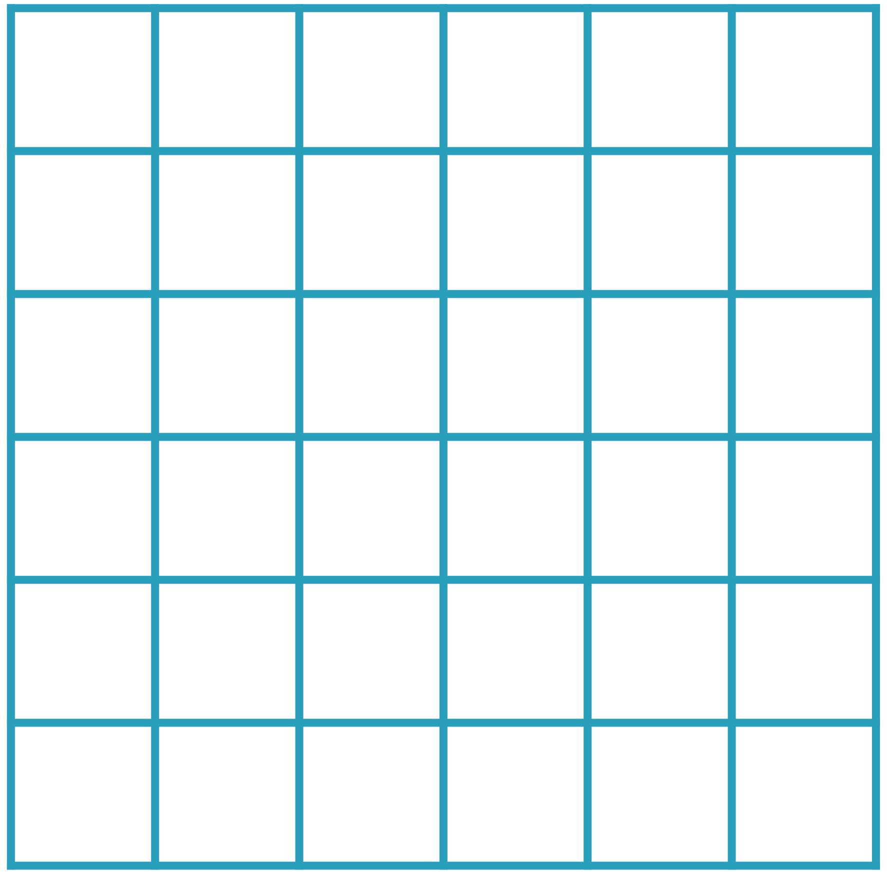
Generate and Assign Unique Word IDs

If the ID generation and assignment is on the same corpus, this is the method to use

Working with Images

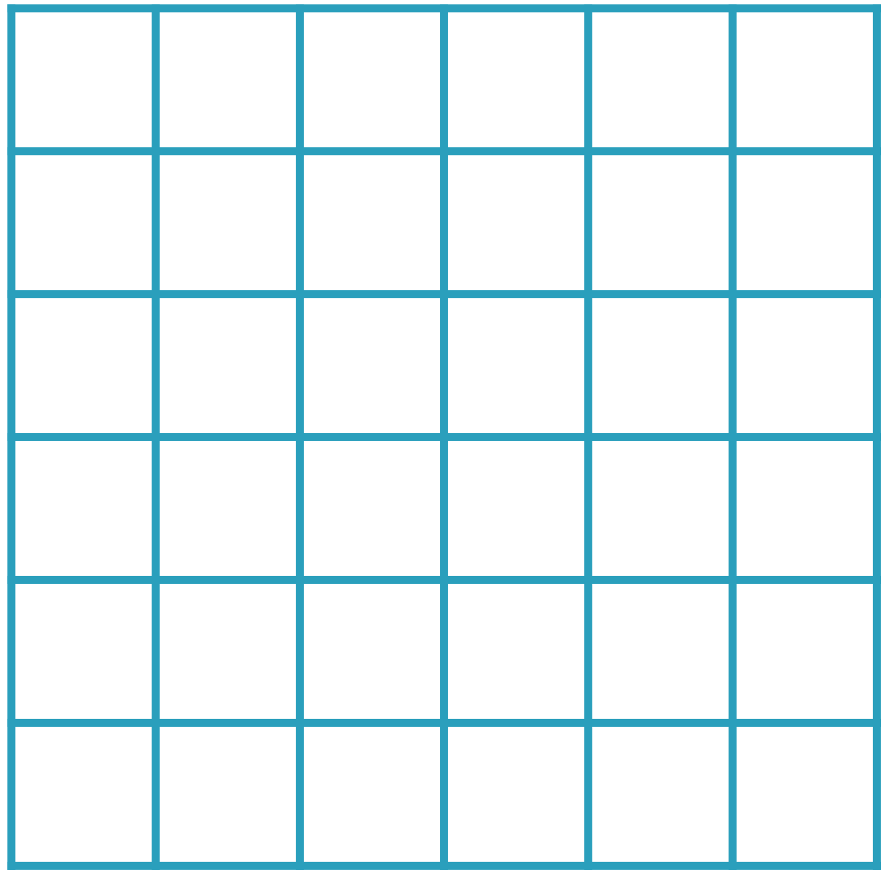
Image as a Matrix

Image as a Matrix



Each pixel holds a value based on the type of image

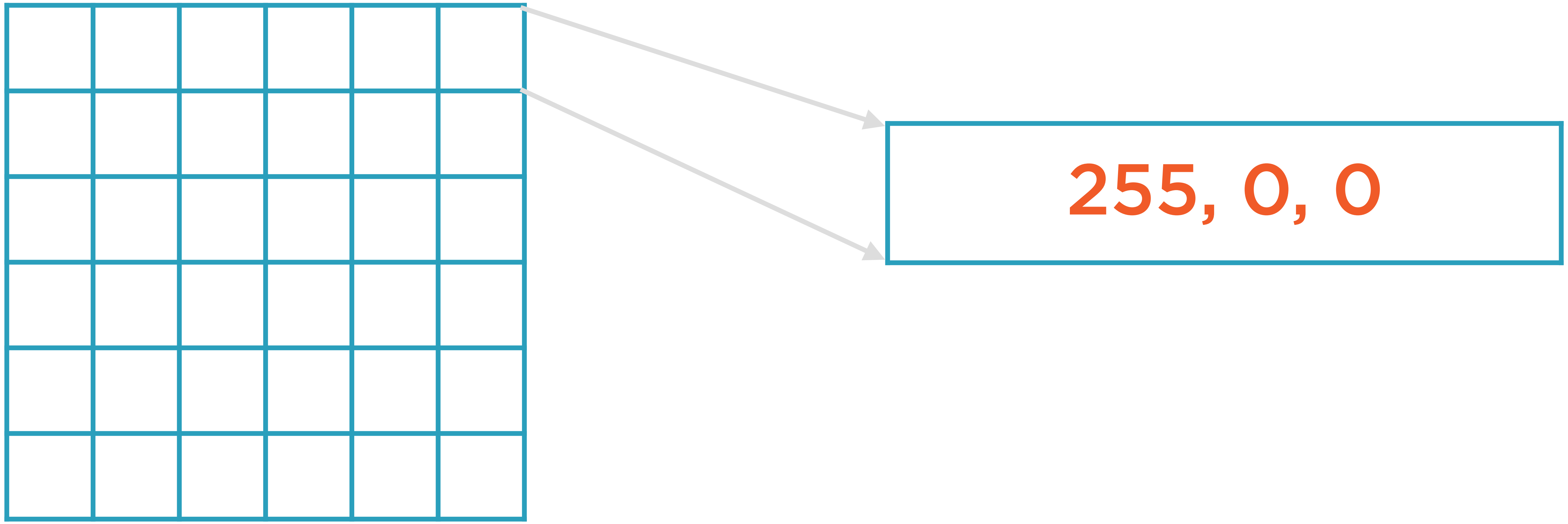
RGB Images



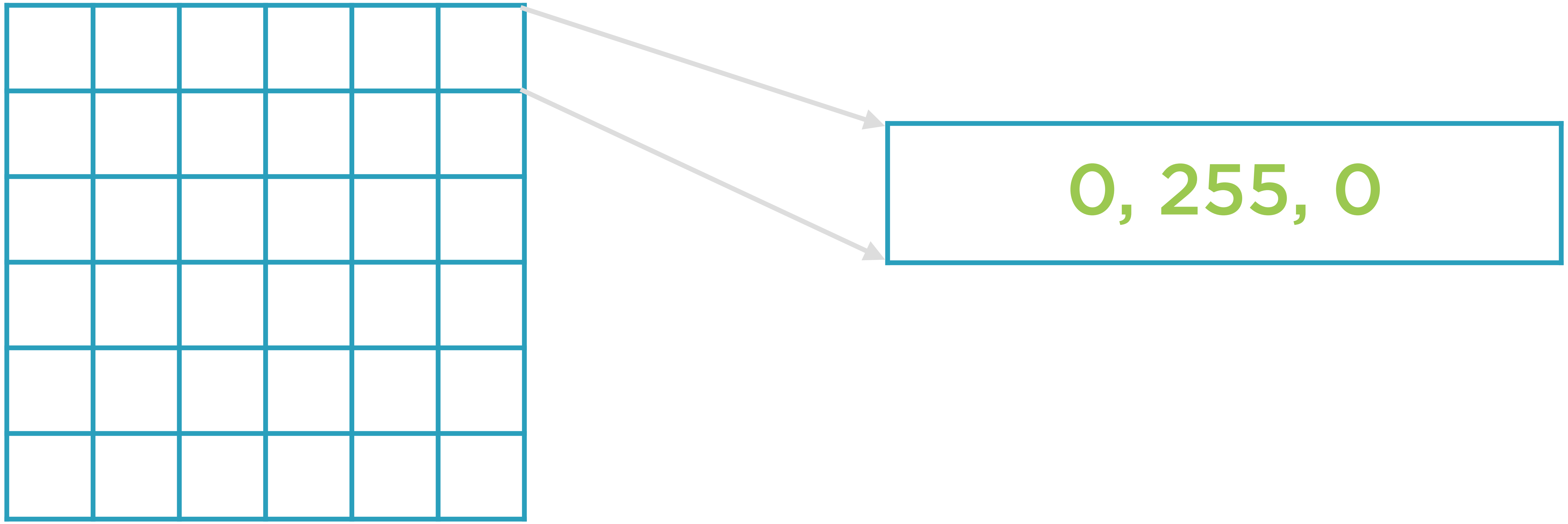
RGB values are for color images

R, G, B: 0-255

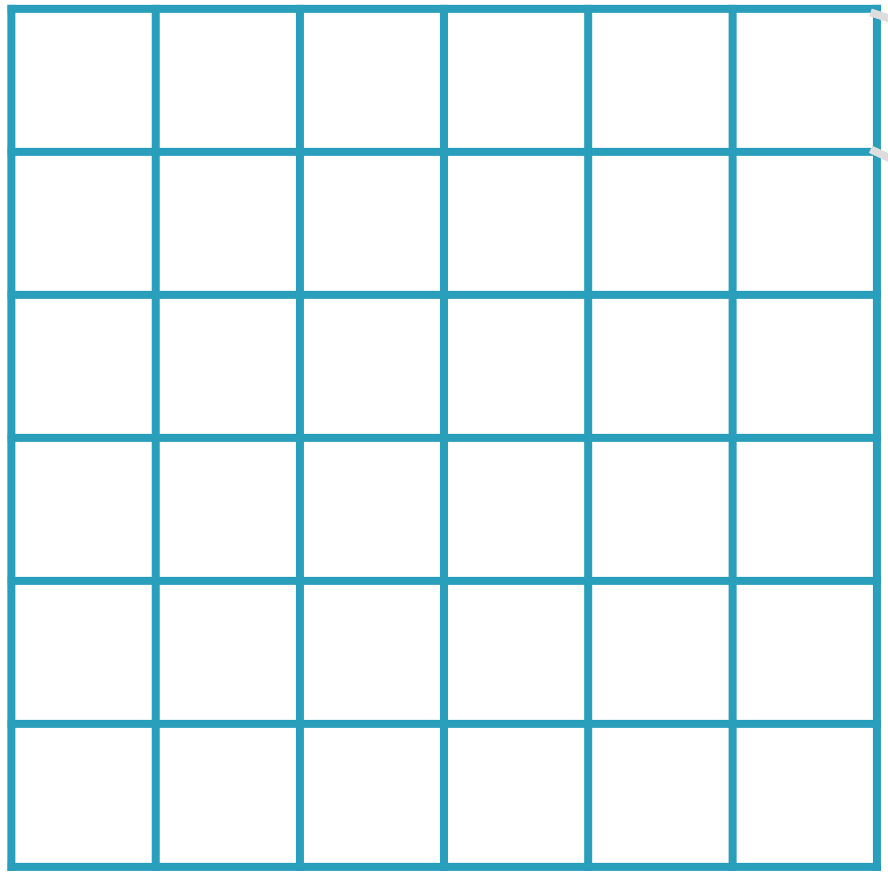
RGB Images

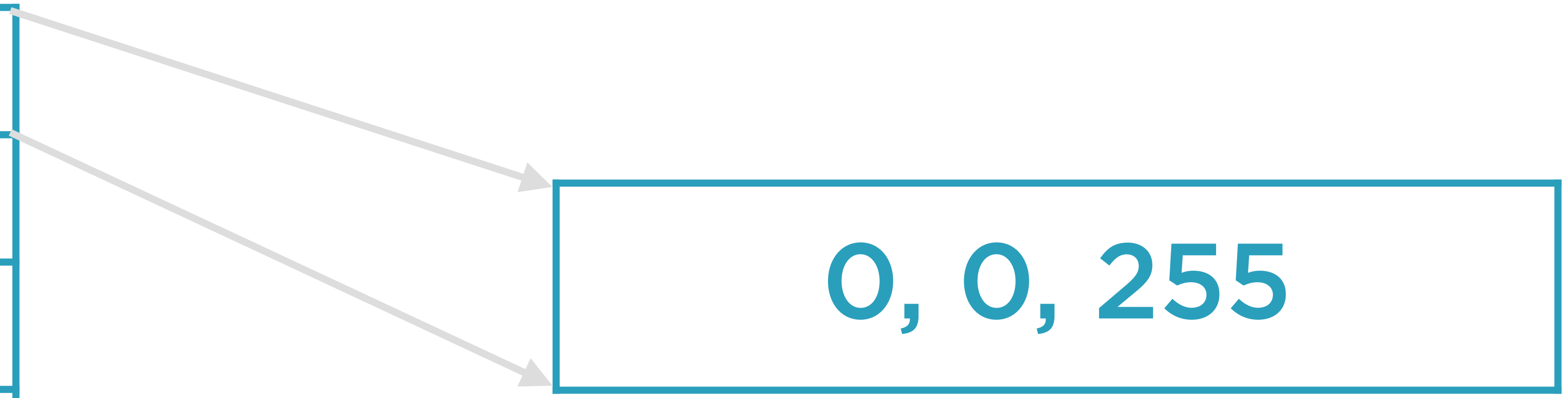


RGB Images



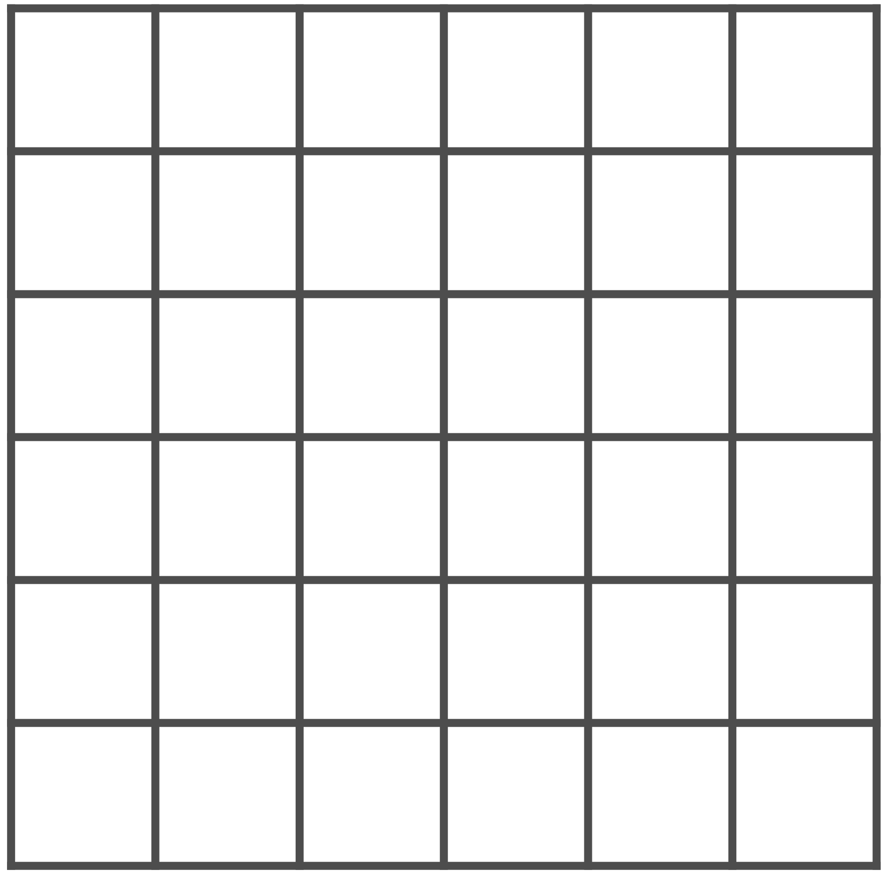
RGB Images



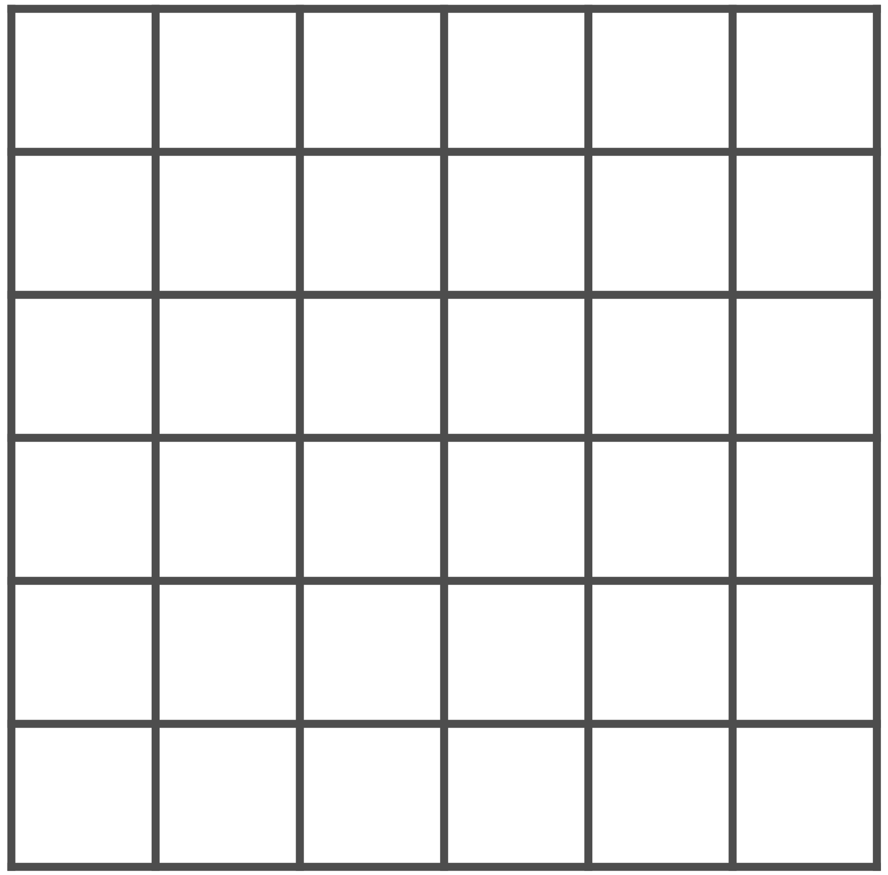


3 values to represent color, 3 channels

Grayscale Images



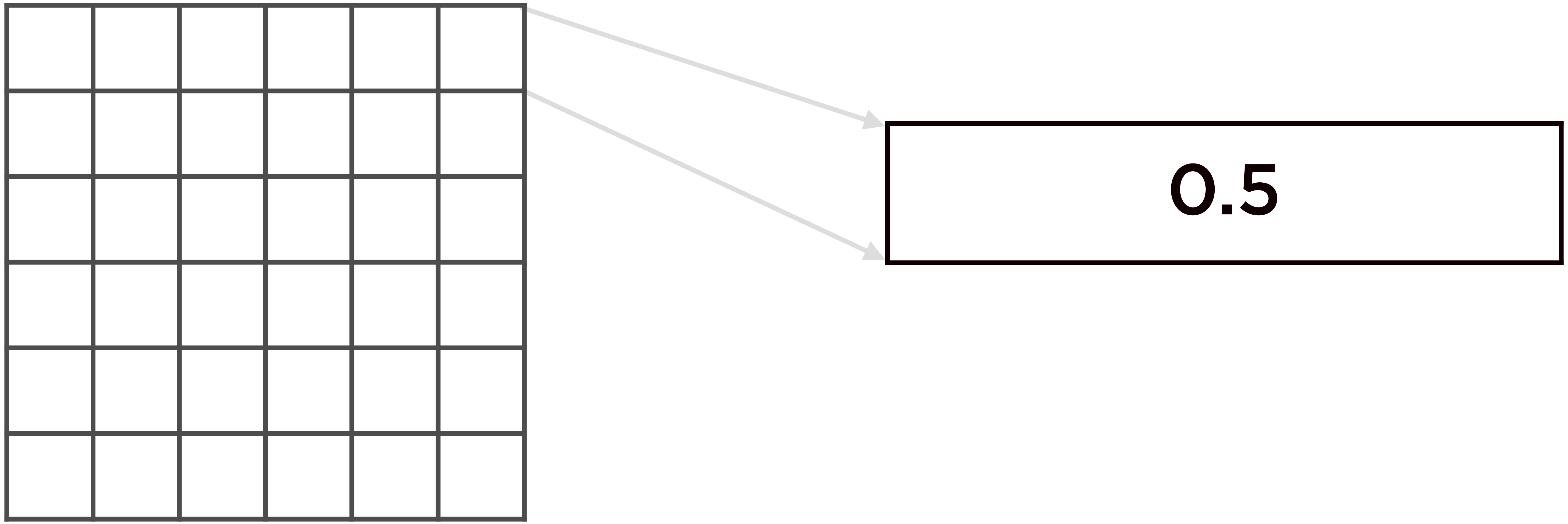
Grayscale Images



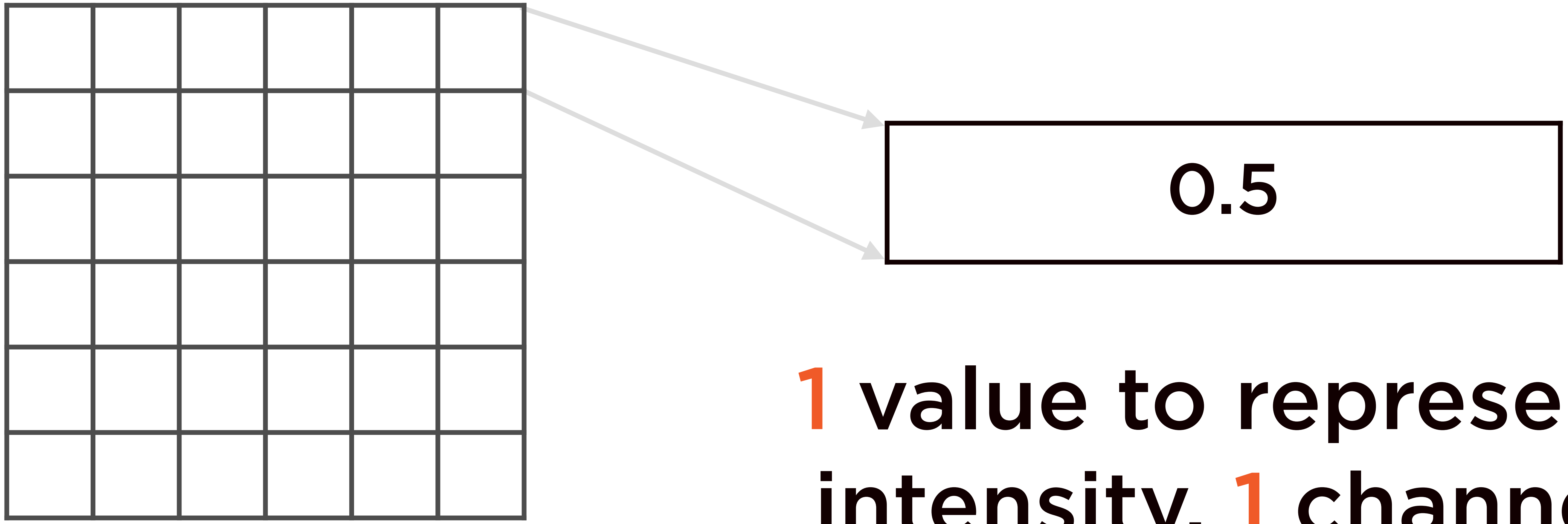
Each pixel represents only intensity information

0.0 - 1.0

Grayscale Images



Grayscale Images

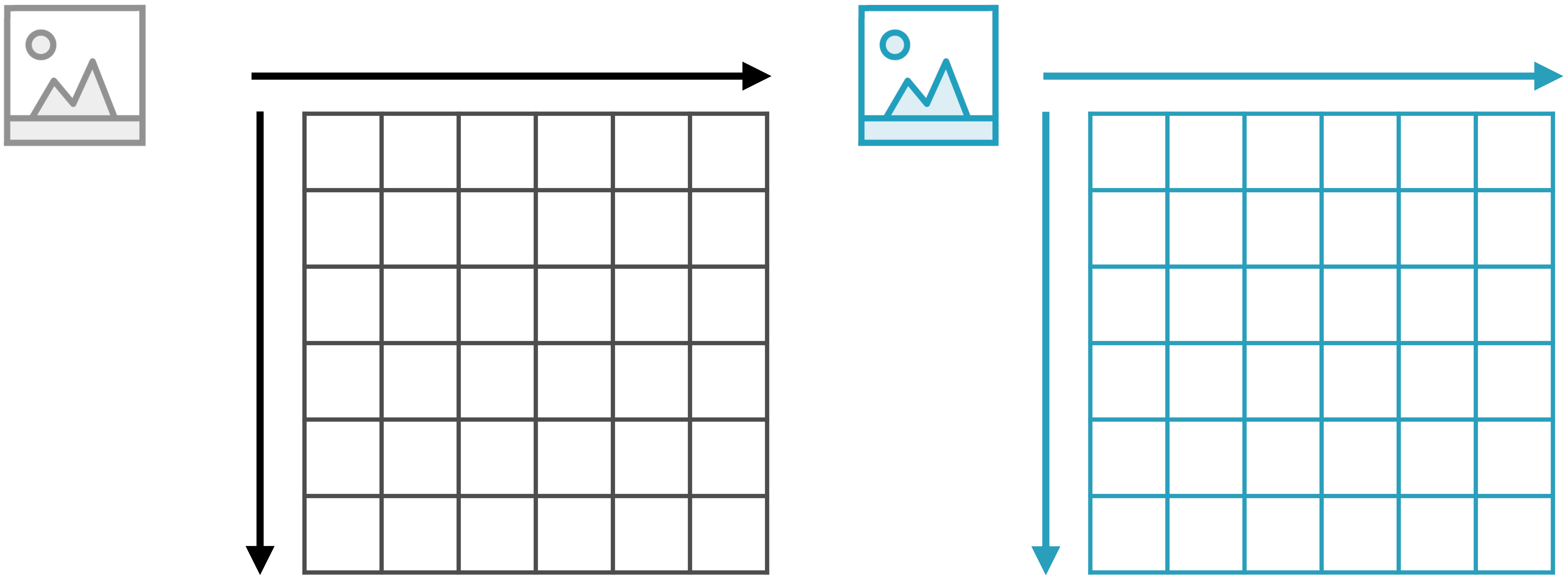


1 value to represent intensity, 1 channel

Image as a Matrix

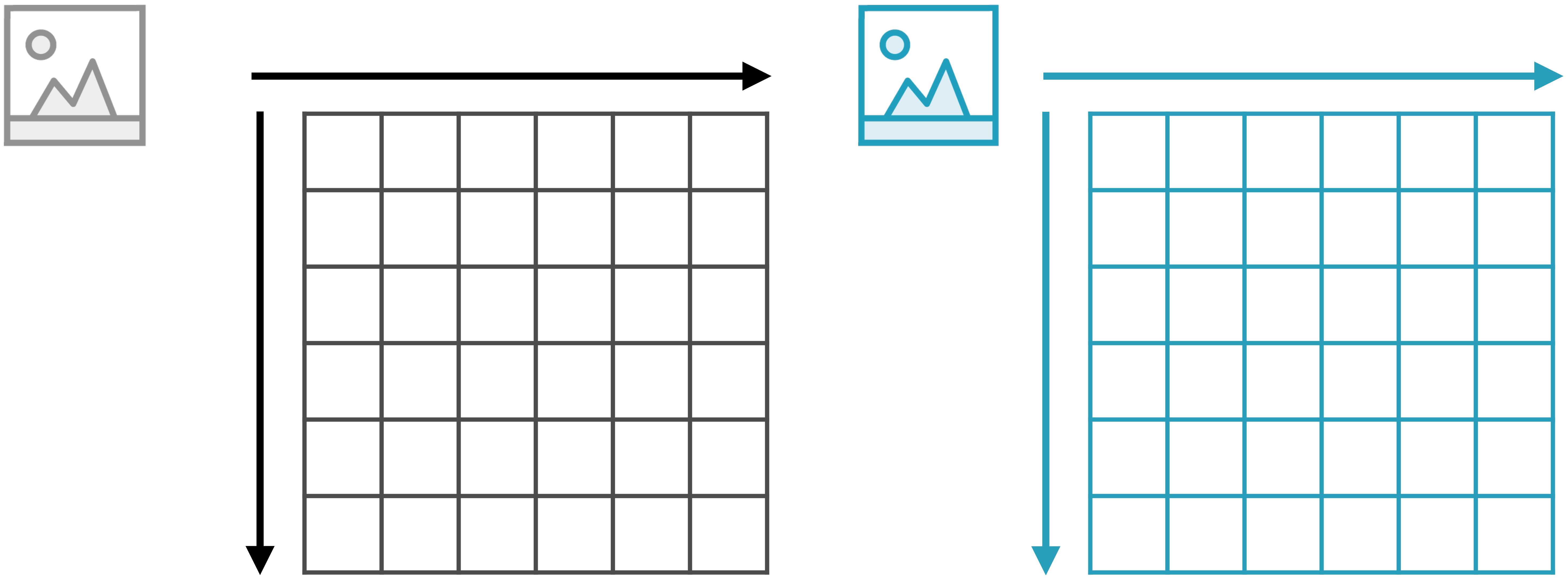
Single channel and multi-channel images

Image as a Matrix



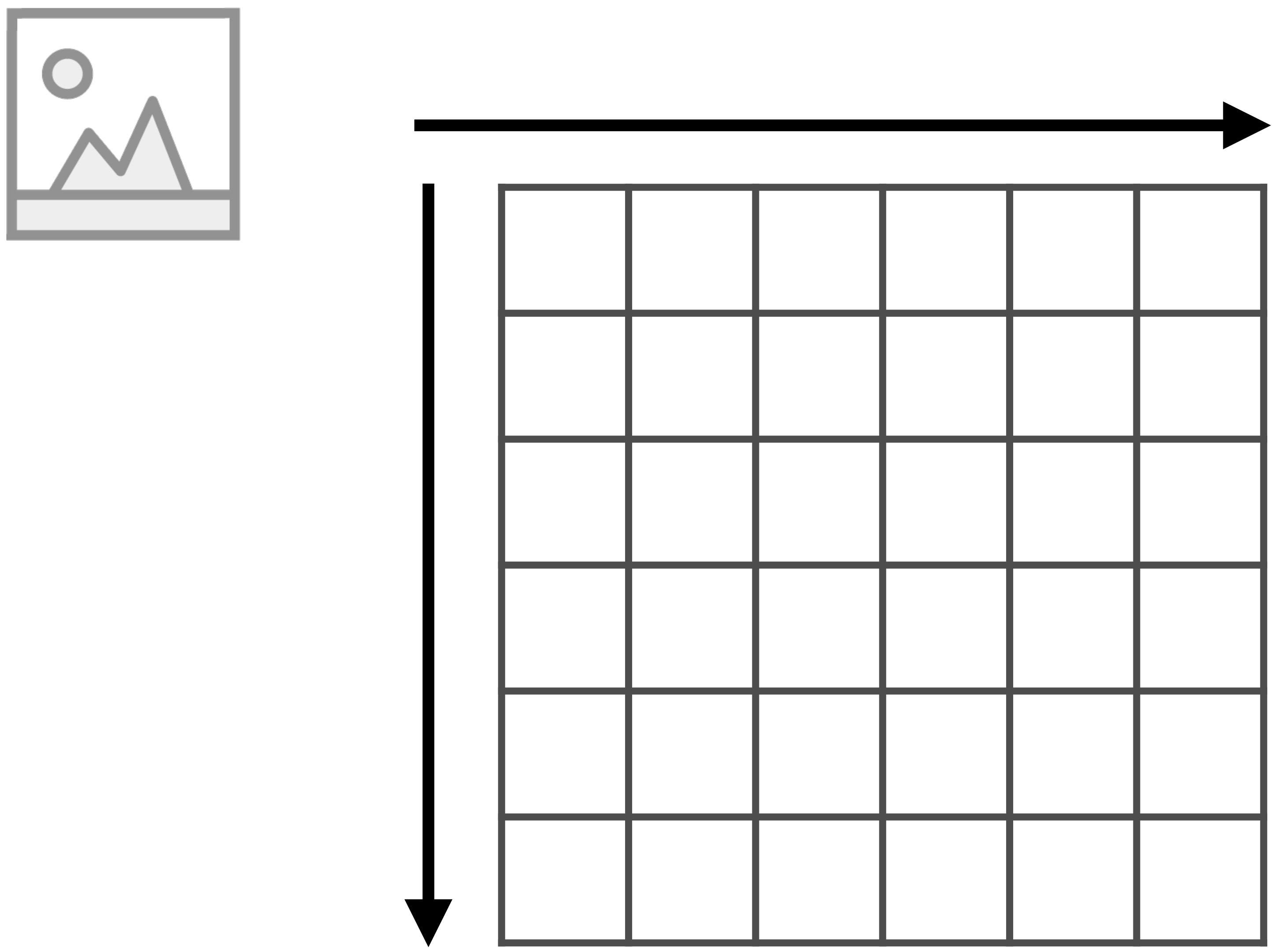
Images can be represented by a 3-D matrix

Image as a Matrix

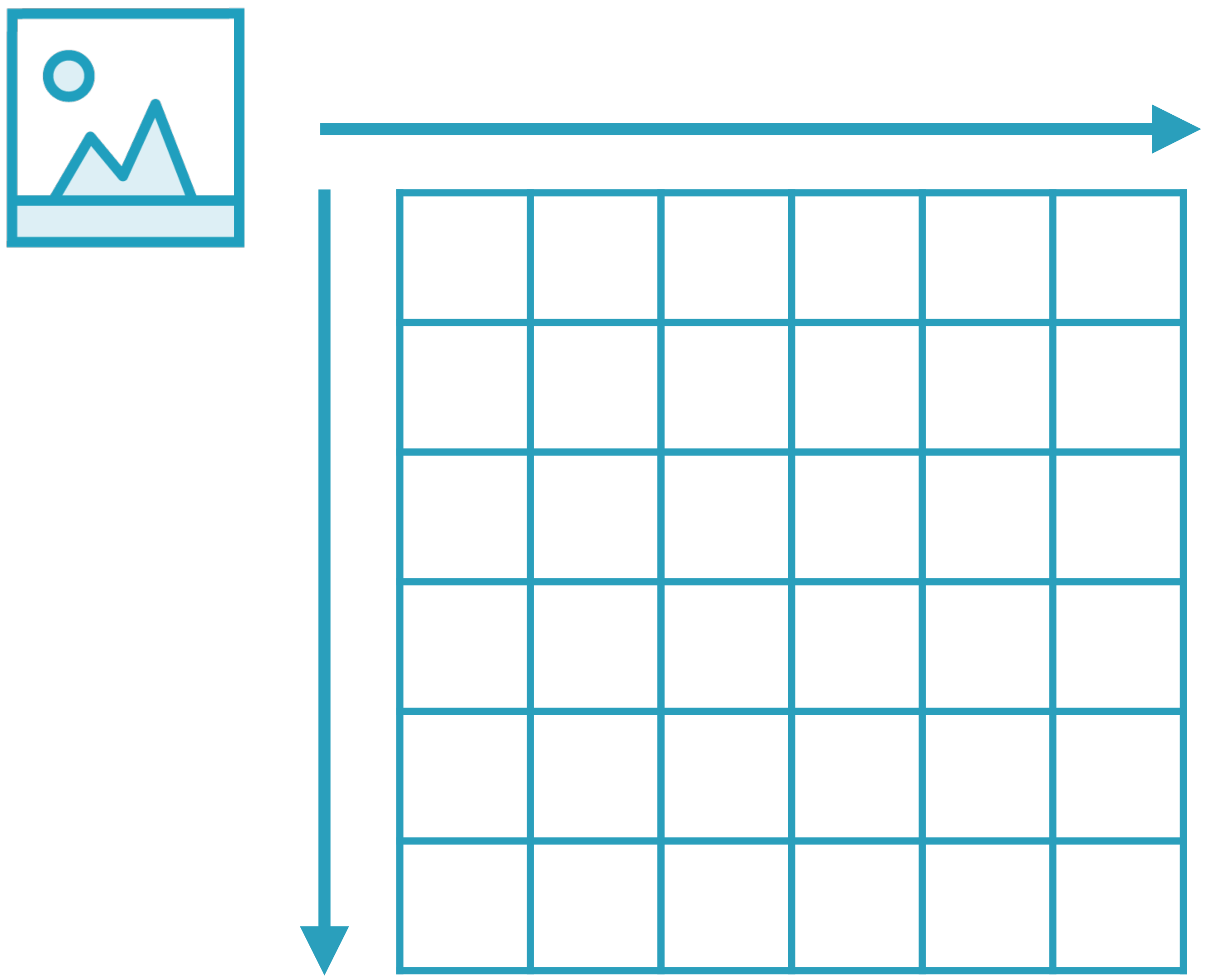


The number of channels specifies the number of elements in the 3rd dimension

Image as a Matrix



$(6, 6, \boxed{1})$



$(6, 6, \boxed{3})$

List of Images

A list of images can be represented as a 4D matrix

List of Images

The images should all be the same size

List of Images

$$(10, 6, 6, \boxed{3})$$

The number of channels

List of Images

(10, 6, 6, 3)

The height and width of each image in the list

List of Images

$(\boxed{10,}\ 6, 6, 3)$

The number of images

scikit-image is a collection of algorithms for image processing

Not covered in this course

Demo

Image feature extraction for color and grayscale images

Use the OpenCV library for image processing

Summary

Understanding different types of ML algorithms and use cases

Working with numerical and categorical data

Using mean and variance to standardize numeric data

One-hot representation of categorical data

Word encodings using counts, TF/IDF and hashing

Extracting features from color and grayscale images