Industry 4.0 and Design: Introduction to AI and Machine Learning

Prof. Andrew D. Bagdanov
andrew.bagdanov AT unifi.it

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze

25 January 2020

Outline

Introduction

Overview

- I will be your lecturer for two classes on very different topics:
  - 25/01/2020 (Today!): Introduction to AI and Machine Learning
  - 07/02/2020: Introduction to Human Computer Interaction.
- Both of these courses are designed to give a broad overview of two vast areas of research and practice.
- You will not become experts, but you should master some of the fundamentals.
- And, perhaps more importantly, you will be versed in the key themes and critical design issues in both areas.

Lecture presentations

- You can will find all class lecture presentations at this site:

  http://micc.unifi.it/bagdanov/i4.0/

- Published here will also be links to Colaboratory Notebooks and any supplementary material.

Defining Artificial Intelligence

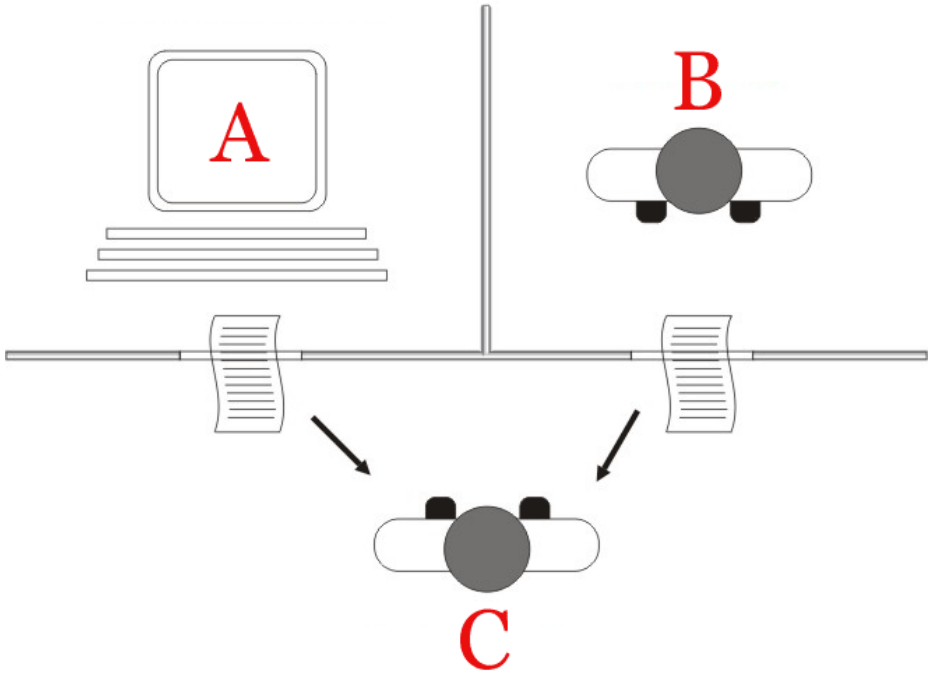
- Thinking Humanly:
  "The study of mental faculties through the use of computational models." – Charniak+McDermott, 1985.
- Thinking Rationally:
  "The branch of computer science that is concerned with the automation of intelligent behavior." – Luger+Stubblefield, 1993.
- Acting Humanly:
  "The study of how to make computers do things at which, at the moment, people are better." – Rich+Knight, 1991.
- Acting Rationally:
  "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning. . . " – Bellman, 1978.

Acting Humanly: The Turing Test

- Turing (1950): Computing machinery and intelligence:

"Can machines think?" $\longrightarrow$ "Can machines behave intelligently?"

- Operational test for intelligent behavior: the Imitation Game:



Acting Humanly: The Turing Test

- Turing predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes.
- Anticipated all major arguments against AI for 50 years.
- Suggested major components of AI: knowledge, reasoning, language understanding, learning.

Problem: Turing test is not reproducible, constructive, or amenable to mathematical analysis.

"It seems probable that once the machine thinking method had started, it would not take long to outstrip our feeble powers. They would be able to converse with each other to sharpen their wits. At some stage therefore, we should have to expect the machines to take control." – Alan Turing, 1951.

Thinking humanly: Cognitive Science

- 1960s cognitive revolution: information-processing psychology replaced prevailing orthodoxy of behaviorism.
- Requires scientific theories of internal activities of the brain:
  - What level of abstraction? "Knowledge" or "circuits"?
  - How to validate? Requires:
    1. Predicting and testing behavior of human subjects (top-down); or
    2. Direct identification from neurological data (bottom-up).
- Both approaches (roughly, Cognitive Science and Cognitive Neuroscience) are now distinct from AI.
- Steven Pinker on Cognitive Science

Thinking rationally: Laws of Thought

Normative (or prescriptive) rather than descriptive:

- Aristotle: what are correct arguments/thought processes?
- Several Greek schools developed various forms of logic: notation and rules of derivation for thoughts.
- This may or may not have proceeded to the idea of mechanization.
- Direct line through mathematics and philosophy to modern AI.

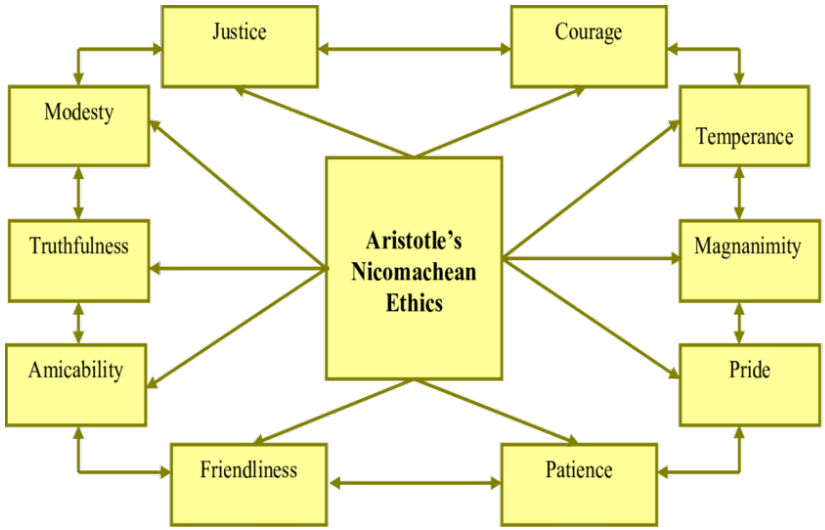
Problems:

1. Not all intelligent behavior is mediated by logical deliberation.
2. What is the purpose of thinking? What thoughts should I have?

Acting rationally

Rational behavior: doing the right thing.

- The right thing: that which is expected to maximize goal achievement, given the available information.
- Doesn't necessarily involve thinking – e.g., blinking reflex – but thinking should be in the service of rational action.
- Aristotle (Nicomachean Ethics): *Every art and every inquiry, and similarly every action and pursuit, is thought to aim at some good.*



Source: Using Accounting Reform to Stimulate Sustainability Practices in Higher Education, 2011.

A first formalism: Rational Agents

Definition (Rational Agents)

An agent is an entity that perceives and acts. This course is about designing rational agents. Abstractly, an agent is a function from percept histories to actions:

$$f : \mathcal{P}^* \to \mathcal{A}$$

For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance.

Caveat: computational limitations make perfect rationality unachievable; so we design best program for given machine resources.

The devil is in the details

- This mathematical formalism doesn't even hint at a recipe for actually building artificially intelligent systems.
- For now we will use an operational definition of AI

Definition (Artificial Intelligence)

Artificial Intelligence refers to the design and implementation of algorithms, applications, and systems that perform tasks normally thought to require *human intelligence* to perform.

Classical roots of AI

- Philosophy: logic, methods of reasoning, mind as physical system, foundations of learning, language, rationality.
- Mathematics: formal representation and proof, algorithms, computation, (un)decidability, (in)tractability, probability.
- Psychology: adaptation, phenomena of perception and motor control, experimental techniques (psychophysics, etc).
- Linguistics: knowledge representation, grammars.
- Neuroscience: physical substrate for mental activity.
- Control theory: homeostatic systems, stability, simple optimal agent designs.

A prehistoric timeline

1943	McCulloch & Pitts: Boolean circuit model of brain.
1950	Turing's "Computing Machinery and Intelligence."
1952–69	Look, Ma, no hands!
1950s	Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine.
1956	Dartmouth meeting: "Artificial Intelligence" adopted.
1965	Robinson's complete algorithm for logical reasoning.
1966–74	AI discovers computational complexity. Neural network research almost disappears.
1969–79	Early development of knowledge-based systems.
1980–88	Expert systems industry booms.
1988–93	Expert systems industry busts: "AI Winter."
1985–95	Neural networks return to popularity.
1988–	Resurgence of probabilistic and decision-theoretic methods. Rapid increase in technical depth of mainstream AI. "Nouvelle AI": ALife, GAs, soft computing.

Early successes

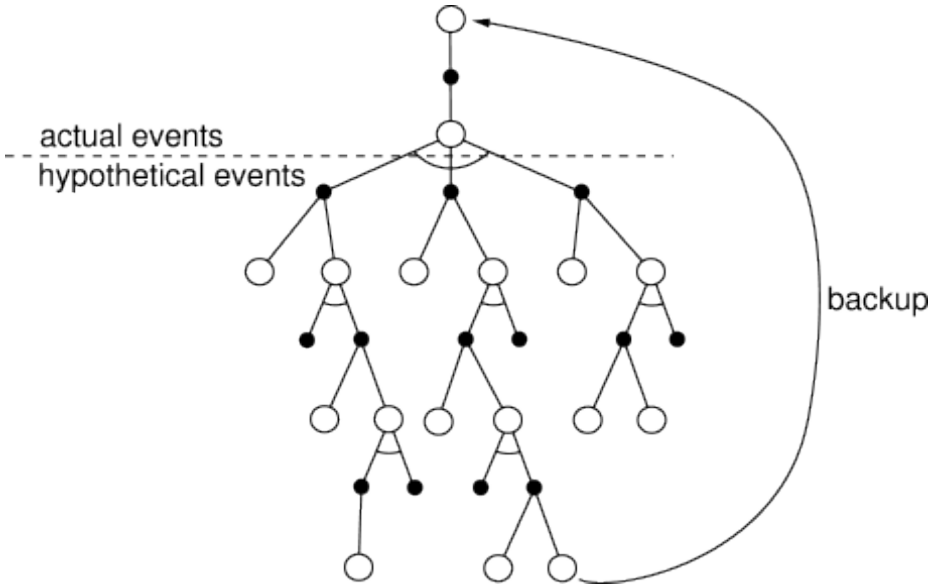
Game playing

- Some of the earliest work in applied AI involved games.
- Arthur Samuel built a system to play checkers in the mid-1950s
- This seminal work defined much of the foundations for classical, search-based AI.
- Game AI has continued to be a benchmark for progress in AI.

Early successes (continued)

Theorem proving

- In 1955, Allen Newell and Herbert A. Simon created the Logic Theorist.
- The program would eventually prove 38 of the first 52 theorems in Russell and Whitehead's *Principia Mathematica*
- It would even find *new and more elegant proofs* for some.



Early optimism

The Dartmouth Workshop

- The Dartmouth Summer Research Project on Artificial Intelligence was a 1956 summer workshop widely considered to be the founding event of artificial intelligence as a field.
- The workshop hosted then (and soon to be) luminaries of the field: Marvin Minsky, John McCarthy, Claude Shannon, Oliver Selfridge, Allen Newell, Herbert Simon, John Nash.
- The organizers thought the general question of artificial intelligence could be resolved (or at least significant progress made on it) over the course of one summer.

Early optimism (continued)

"We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves."

The First AI Winter (1974-1980)

Combinatorial explosion

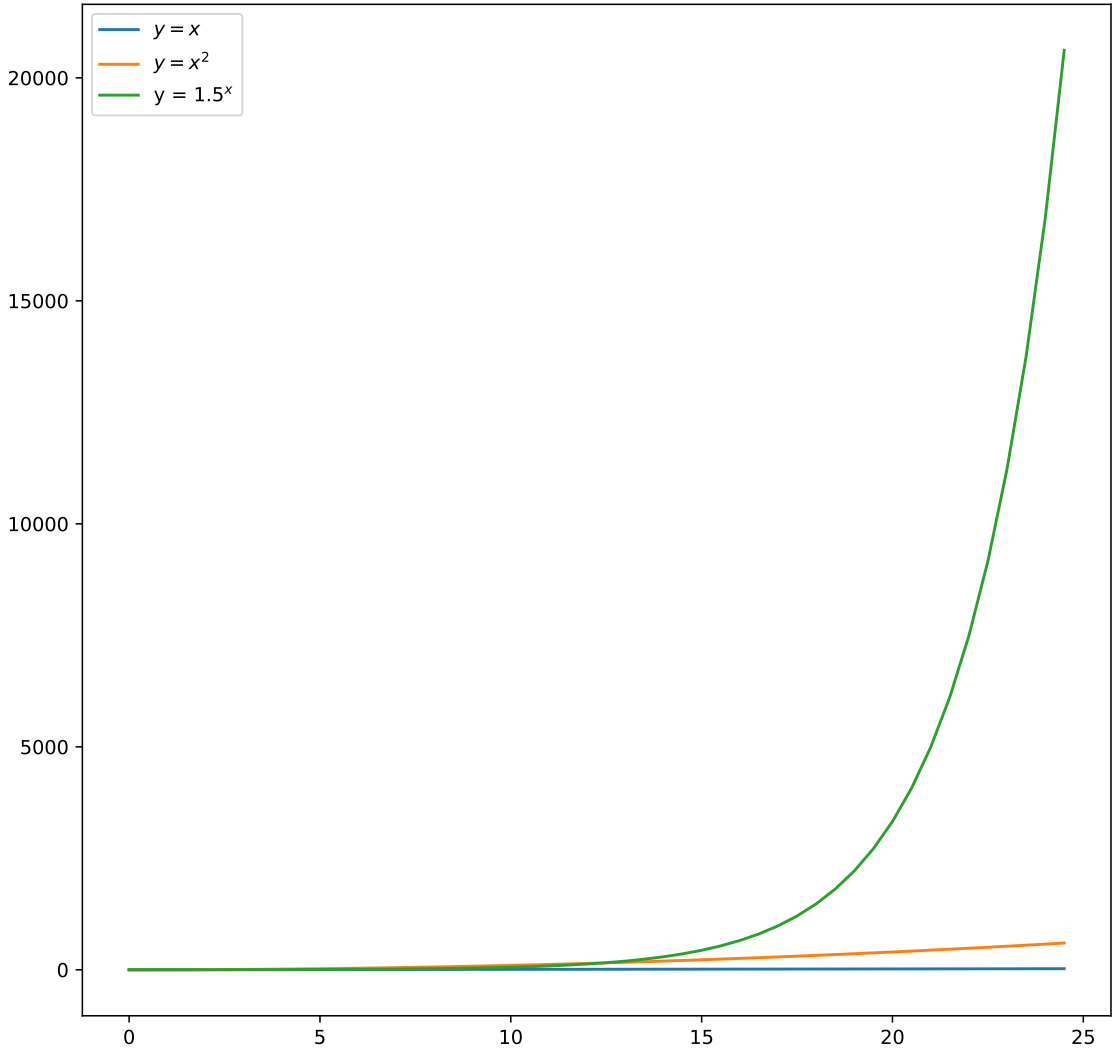
- Most early approaches to AI were based on searching in configuration spaces.
- Even theorem proving is a (very general) type of search in deduction space.
- AI programs could play checkers and prove theorems with relatively few inference steps.
- But going beyond these early successes proved extremely difficult due to the exponential nature of the search space.

Lack of "common knowledge"

- Another difficulty was that solving many types of problems (e.g. recognizing faces or navigating cluttered environments) requires a surprising amount of background knowledge.
- Researchers soon discovered that this was a truly vast amount of information.
- No one in 1970 could build a database so large and no one knew how a program might learn so much information.

The First AI Winter (1974-1980)

- We often don't appreciate what exponential growth really means:



Insurmountable problems (continued)

Limited computing power

- There was not enough memory or processing speed to do anything truly useful.
- In 2011, computer vision applications required 10,000 to 1,000,000 MIPS.
- By comparison, the fastest supercomputer in 1976, the Cray-1 (retailing at $5 million to $8 million), was only capable of around 80 to 130 MIPS, and a typical desktop computer less than 1 MIPS.

Infighting

- The perceptron neural network was introduced in 1958 by Frank Rosenblatt.
- There was an active research program throughout the 1960s but it came to a sudden halt with the publication of Minsky and Papert's 1969 book Perceptrons.
- It suggested that there were severe limitations to what perceptrons could do and that Rosenblatt's predictions had been grossly exaggerated.
- The effect of the book was devastating: *virtually no research at all was done in connectionism for 10 years*.

The 80s Boom

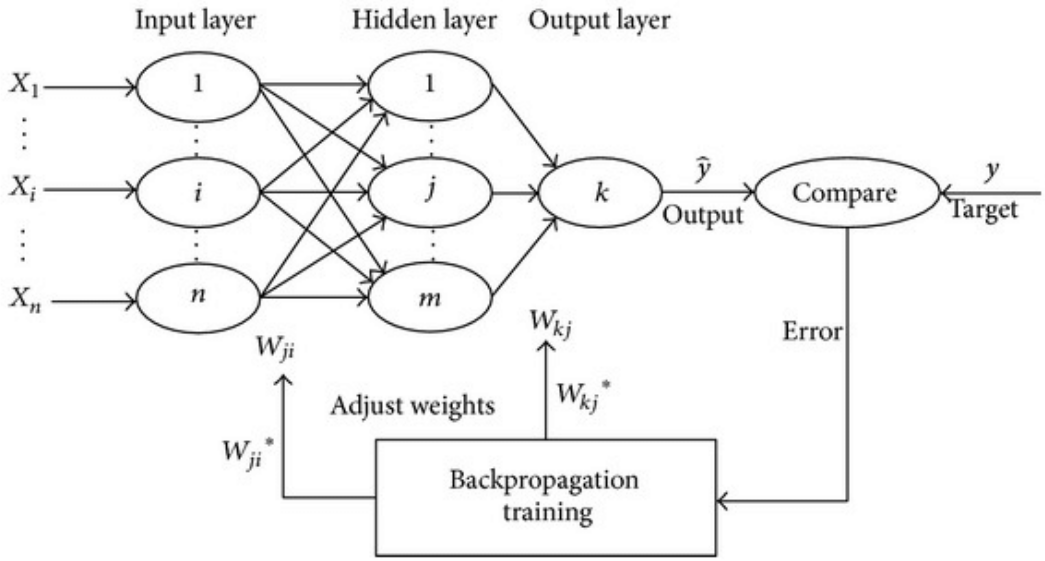
Expert systems

- In the 1980s a form of AI program called an expert system was adopted by corporations around the world.
- Knowledge became the focus of mainstream AI research.
- An expert system is a program that answers questions or solves problems about a specific domain of knowledge and logical rules derived from experts.
- They were part of a new direction in AI research that had been gaining ground throughout the 70s.
- AI researchers were beginning to suspect that intelligence might very well be based on the ability to use large amounts of diverse knowledge in different ways.
- Expert systems restricted themselves to a small domain of specific knowledge (thus avoiding the commonsense knowledge problem).
- All in all, the programs proved to be useful: *something that AI had not been able to achieve up to this point.*

The 80s Boom (continued)

The connectionist revival

- In 1982, John Hopfield proved that a form of neural network (now called a Hopfield net) could learn and process information.
- Around the same time, Geoffrey Hinton and David Rumelhart popularized a method for training neural networks called backpropagation.
- Neural networks would become commercially successful in the 1990s for OCR and speech recognition.



Winter is coming (again): The Bubble Phenomenon

Expansion and crash

- The business community's fascination with AI rose and fell in the 1980s in the classic pattern of an economic bubble.
- The collapse was in the perception of AI by government agencies and investors – the field continued to make advances despite the criticism.
- The first indication of a crash was the sudden collapse of the market for specialized AI hardware in 1987.
- Desktop computers from Apple and IBM were gaining speed and power and were soon more powerful than the more expensive Lisp machines made by Symbolics and others.
- There was no longer a good reason to buy them, and an entire industry worth half a billion dollars was demolished overnight.

Not all bad news

Follow the money (or not)

- In the late 1980s most public funding for AI dried up.
- Despite this, the true believers continued to make steady theoretical and applied progress.
- People like Jürgen Schmidhuber, Yann LeCun, Geoff Hinton, and Yoshua Bengio make significant progress during the Second AI Winter.
- As the community came to grips with the fact that expert systems don't scale very well and are expensive to maintain, neural networks resurfaced as a viable contender for the way forward.
- In particular, the first viable Convolutional Neural Networks (CNNs) were demonstrated and the backpropagation algorithm was proven scalable (and controllable).

The Renaissance: The Right Place and Time

Deep learning

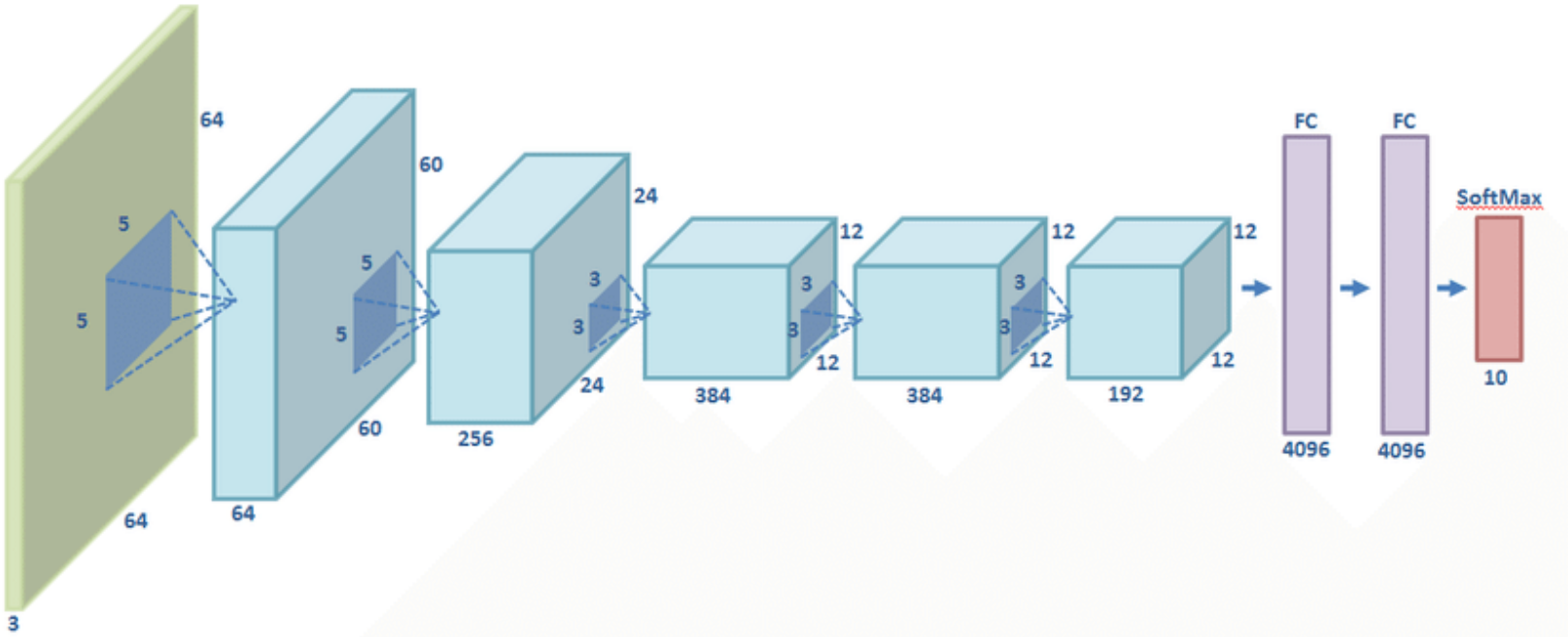
1. Deep Learning – compositional, multi-layer neural networks – has been around since the 1970s.
2. However, we did not understand how to effectively learn the vast number of parameters they have from data.
3. And, computers of the day were not up to the challenge of fitting these models.

ImageNet and GPUs

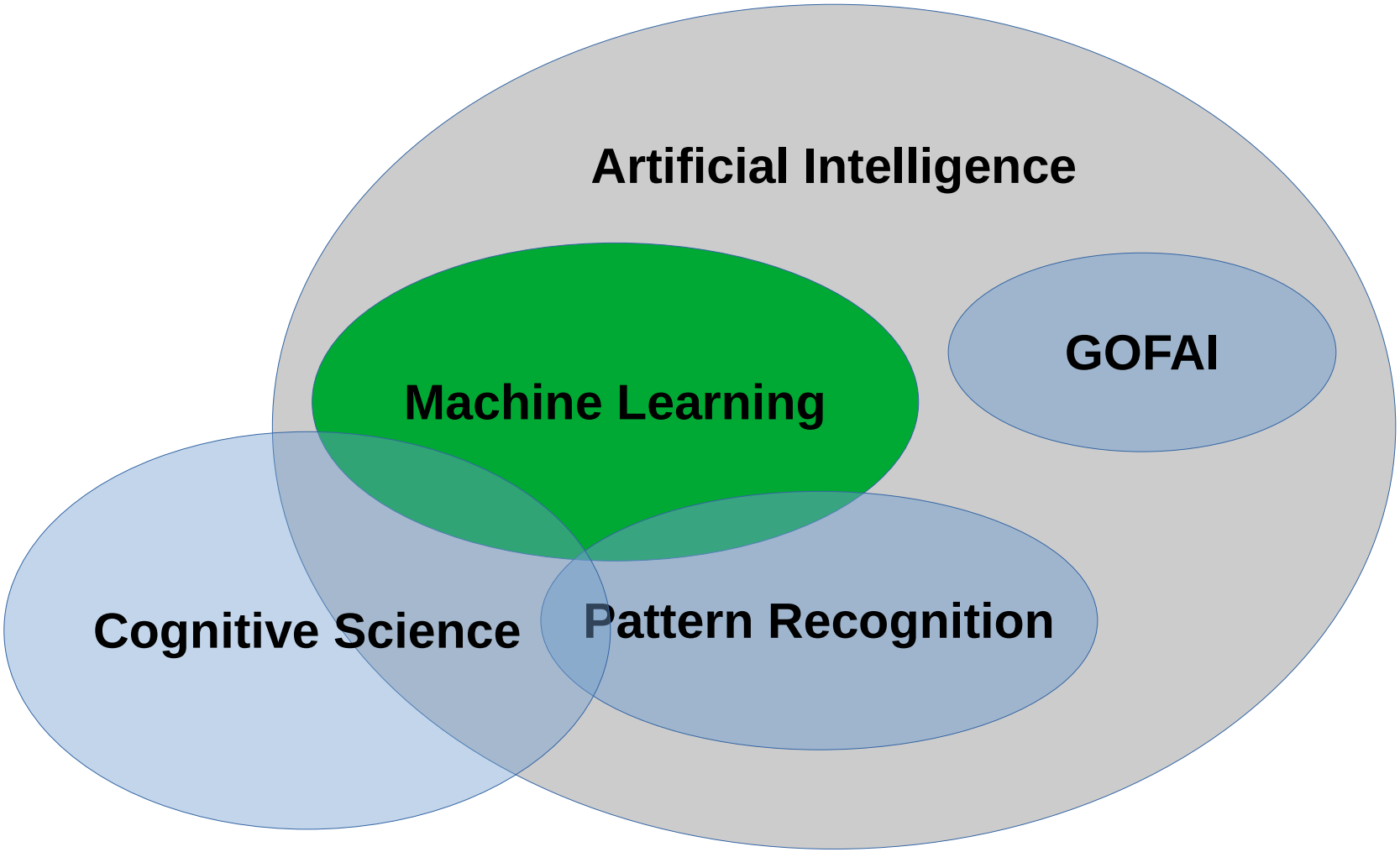
- In the early 2010s the confluence of several technological and theoretical factors combined.
- ImageNet: massive amounts of labeled data made deep learning feasible.
- GPUs: Graphics Processing Units addressed some of the computational issues related to training deep models.

The Shot Heard 'Round the World: AlexNet

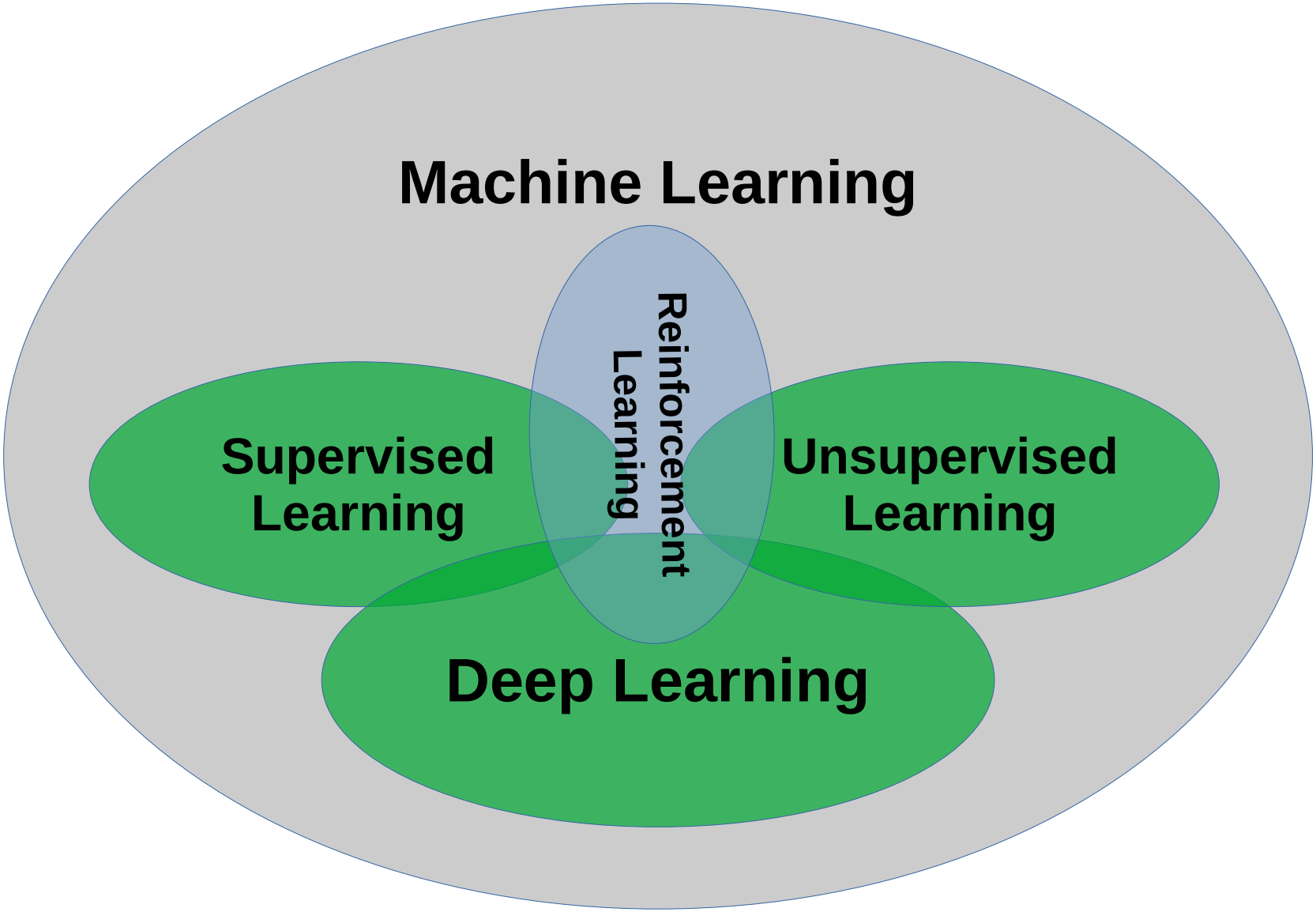
- In 2012 the AlexNet Deep Convolutional Network made history.
- *Everything had changed*: AlexNet surpassed the current state-of-the-art by nearly 20%.



AI, ML, and all that: A global view



AI, ML, and all that: A local view (this course)



Global Objective: Something for everyone

- This course is designed for a broad and diverse audience.
- Some mathematical background is assumed, as well as some exposure to high-level programming (e.g. Python, Java, C/C++, Lisp, heck even Visual Basic).
- [INFORMAL SURVEY]

Global Objective: For the curious

- So everyone is talking about deep learning and General Artificial Intelligence.
- But what is all the hype about?
- This course will give you a broad overview (without the hype) of the fundamental concepts surrounding these recent advances.
- Because it's not all hype. There is an ongoing and sweeping sea change happening today.
- Artificial intelligence is already having significant impact in manufacturing, advertising, smart city management, transportation, entertainment, you name it.
- For the curious: this course should provide you the grounding in essential concepts needed to interpret, understand, and exploit these new developments.

Global Objective: For the studious practitioner

- So everyone is talking about deep learning and General Artificial Intelligence.
- For those of you actually working daily with large (or even massive) amounts of data, what does this mean for you?
- This course will give you hands-on experience with the tools, models, and frameworks used today.
- You will learn how to manipulate data and extrapolate models from it that generalize well to unseen data.

Global Objective: a sign of the times...

- So everyone is talking about deep learning and General Artificial Intelligence.
- Articles and whole courses are appearing daily with tantalizing titles like "Deep Learning Zero to Hero" and "How to learn Deep Learning in One Week!!1!".
- It has become difficult to *sift the wheat from the chaff* because the signal-to-noise ratio is becoming infinitesimal.
- For everyone: this course should ground you in the fundamental concepts and tools needed to discern charlatans from quality sources and to make sense of AI and deep learning advances.

Foundations

Math and Programming

- Mathematical Foundations:
  - Linear algebra has been called the mathematics of the 21st century – and it is essential to understanding all of the models, techniques, and algorithms we will see.
  - Calculus is also central to how we actually learn from data and we will see (from a high level) how learning can be formulated as a optimization problem.

Numerical Programming and Reproducible Science

Working with Tsunamis of Data

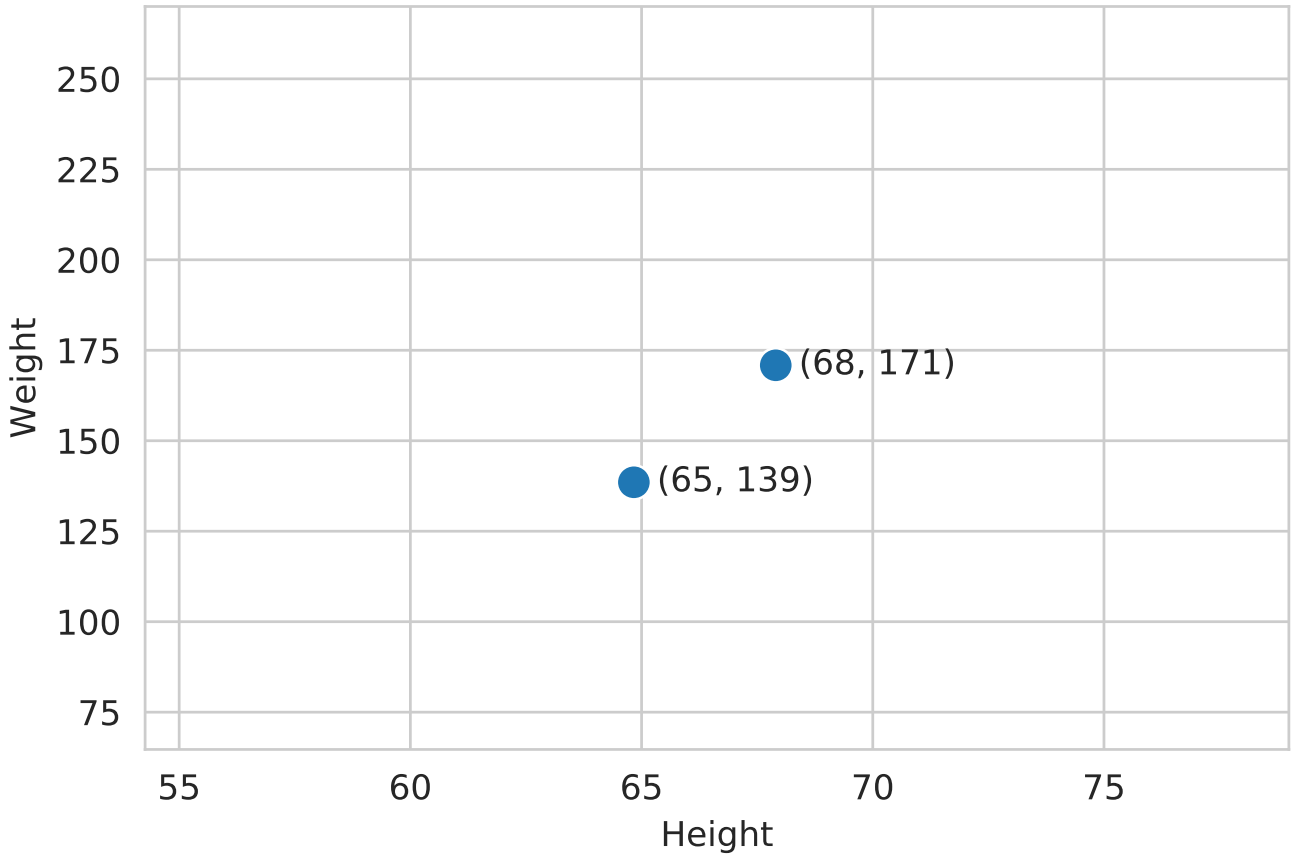
- Visualization:
  - Central to working with massive amounts of data is effective visualization of high dimensional data.
  - We will use techniques like histograms, scatter plots, t-SNE, and others to monitor learning progress and to summarize results.
- Data Science:
  - Also central to working with Big Data is the need to manage data in flexible and abstract ways.
  - We will use Jupyter notebooks (in the form of Google Colaboratory) to organize, document, and guarantee reproducibility.
  - We will use the Pandas library to perform data analysis, to manage data and datasets, and to prepare data for our experiments.

Supervised learning

- Let's say are analyzing the correlation between height and weight.
- (Aside: we will often use synthetic examples of this type to illustrate key concepts and techniques.)
- And let's say that we have only two data points:
$$(67.9, 170.85) \text{ and } (61.9, 122.5).$$
- Ideally, we wish to infer a relation between height and weight that explains the data.
- A good first step us usually to visualize.

Supervised learning (continued)

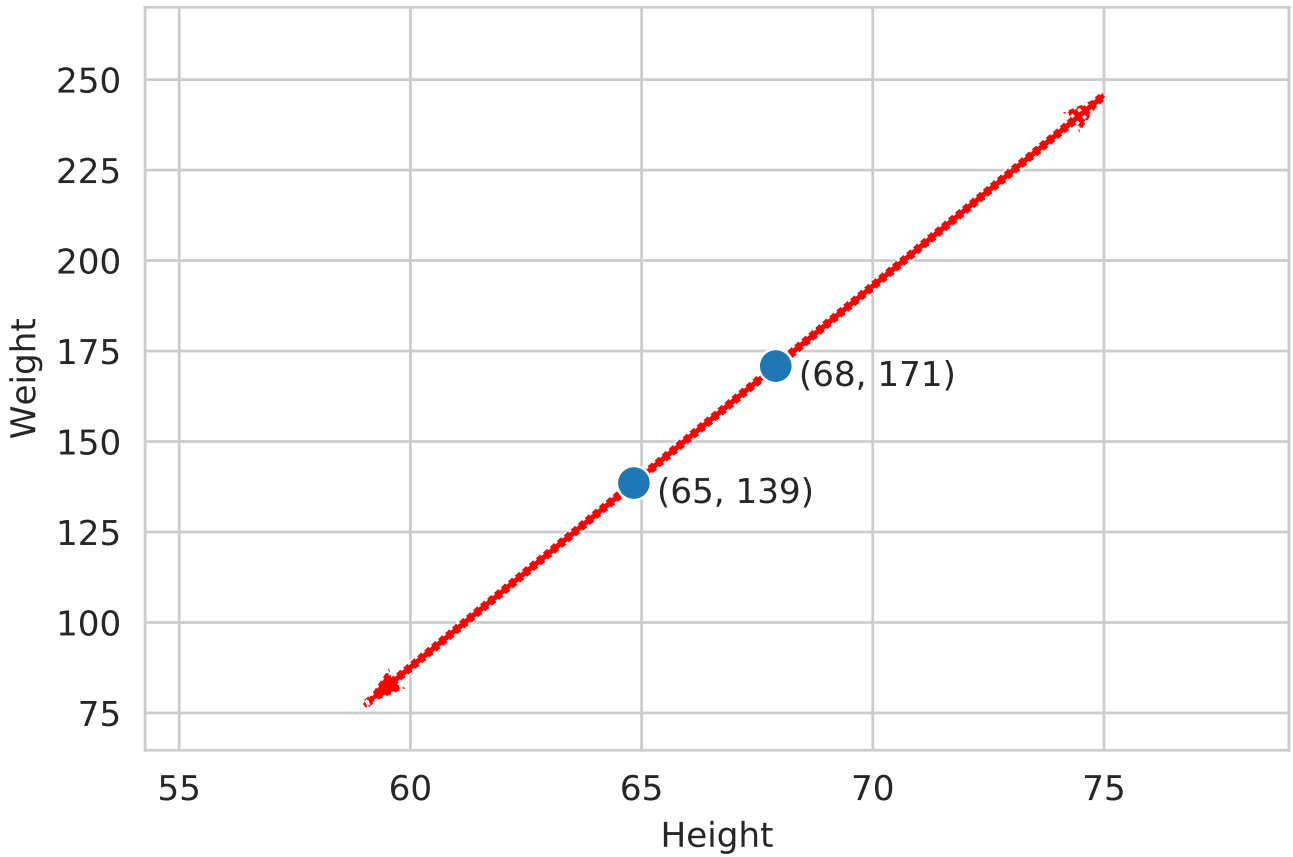
- So, we have a situation like this. . .
- What can we do?



Supervised learning (continued)

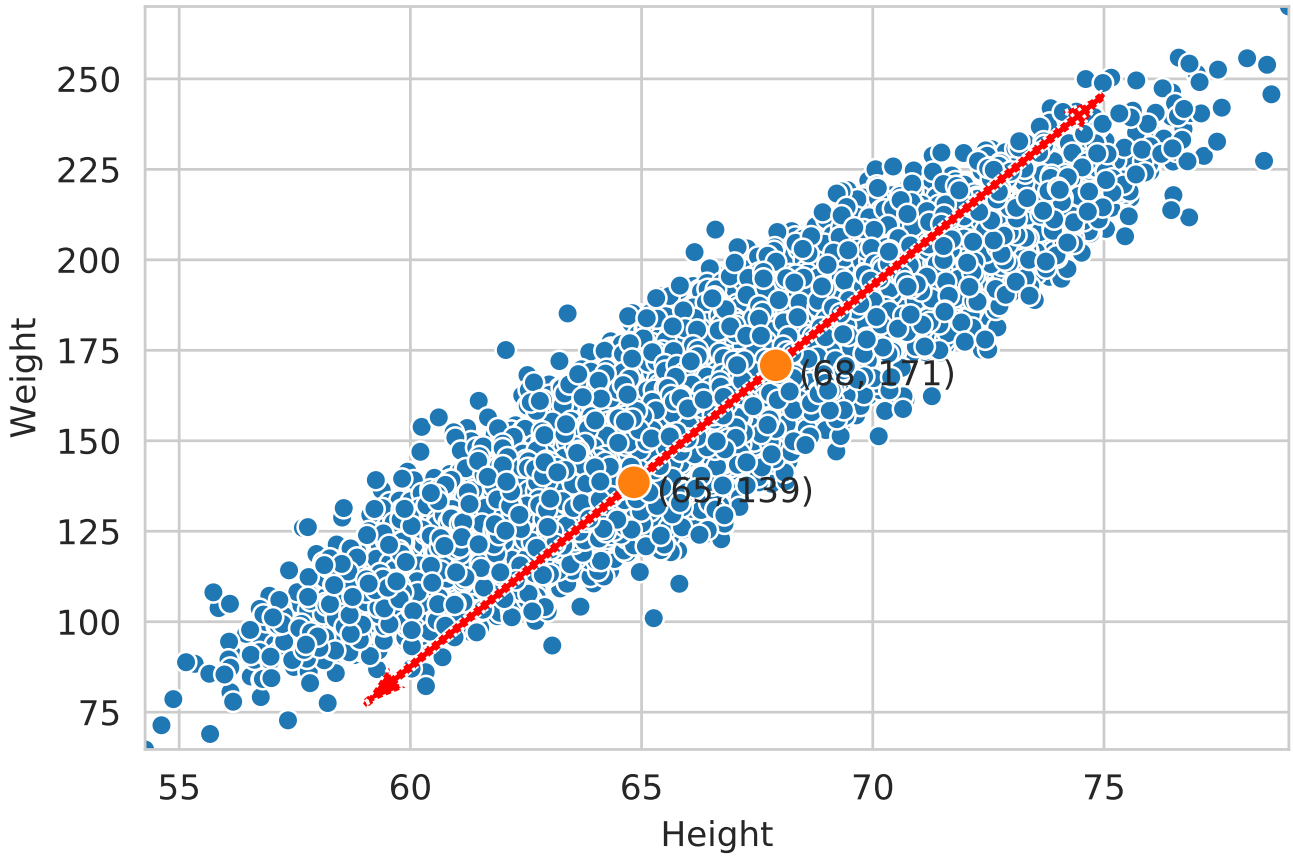
- Well, some grade-school algebra lets us connect the dots:

$$y = 8.013x - 373.247$$



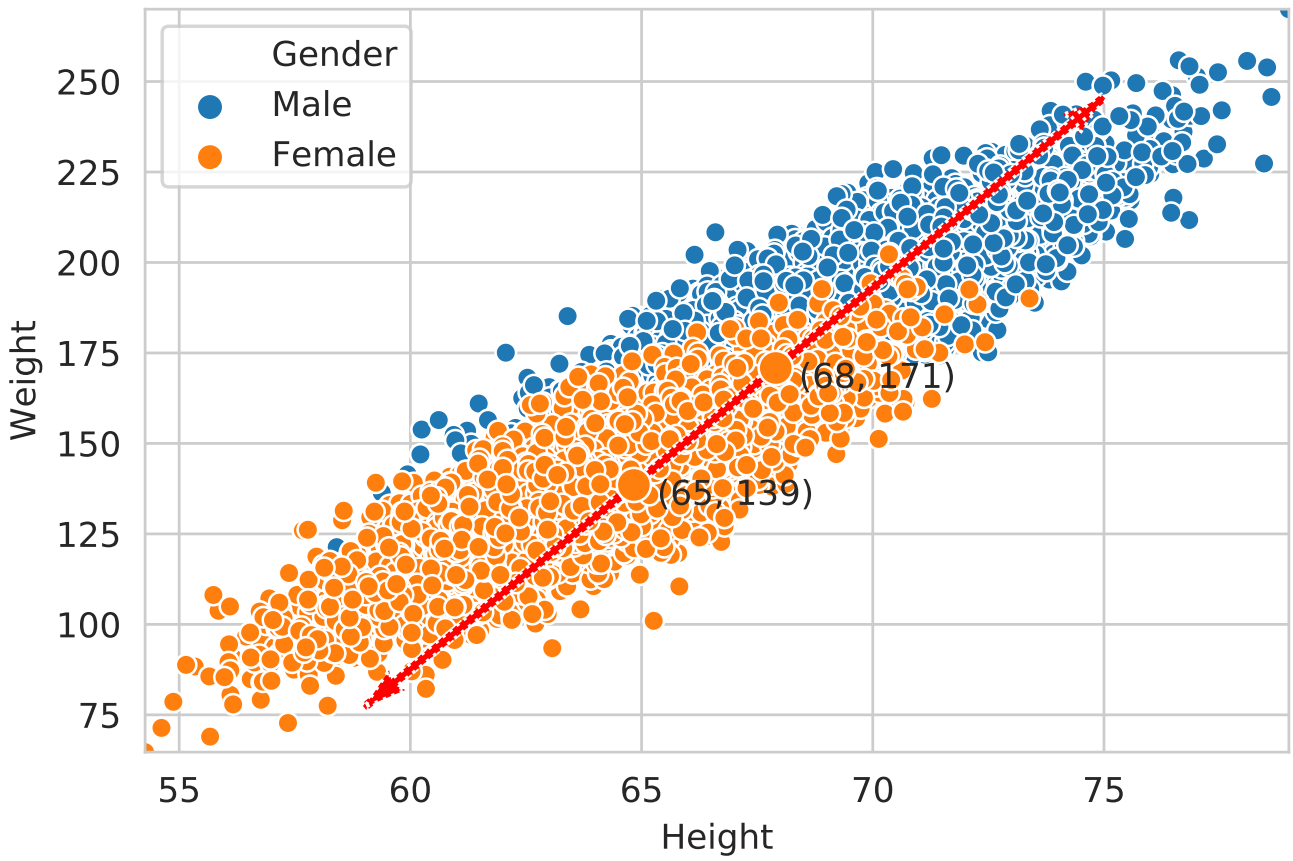
Supervised learning (continued)

- Now lets say that we have a lot more data.
- Does our "model" generalize?



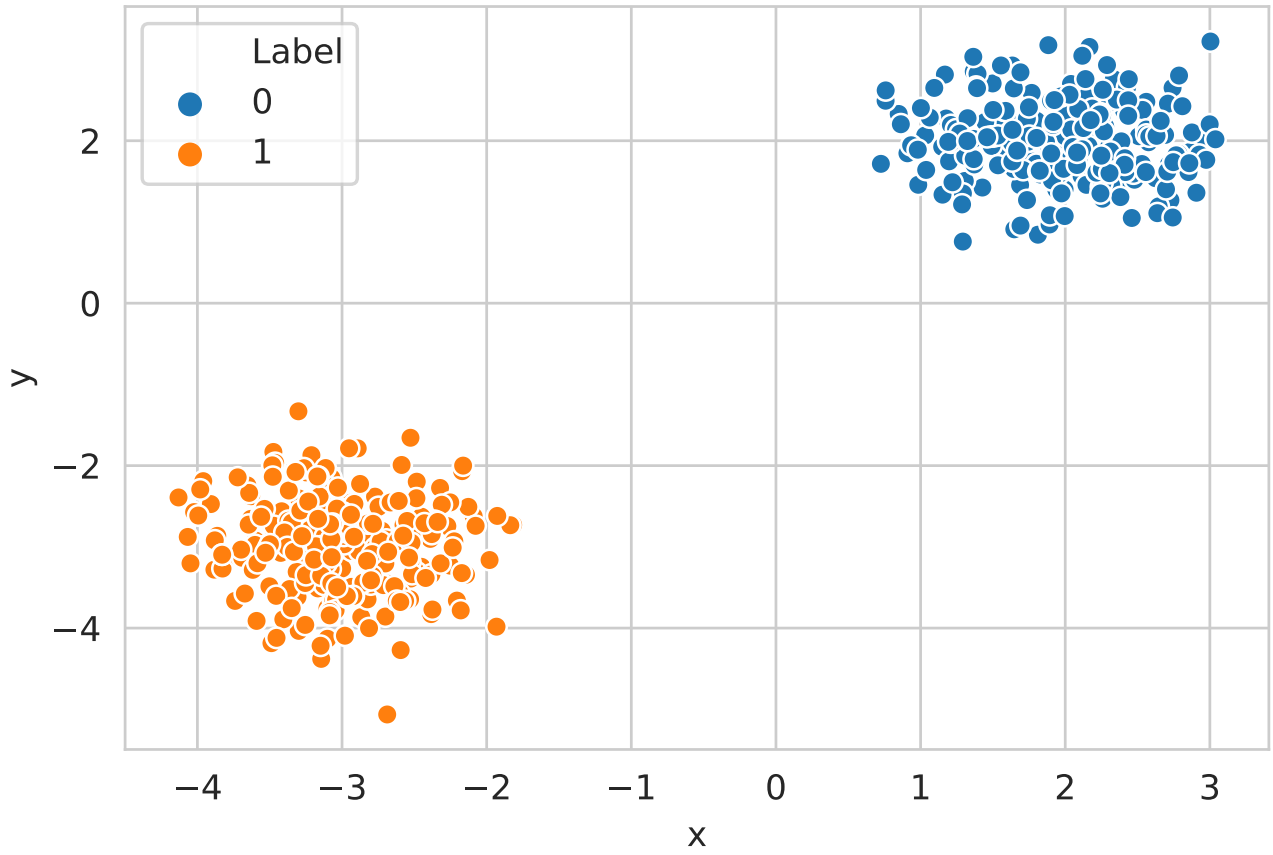
Supervised learning (continued)

- Scratching the surface a bit more, we discover not a single distribution, but rather two.



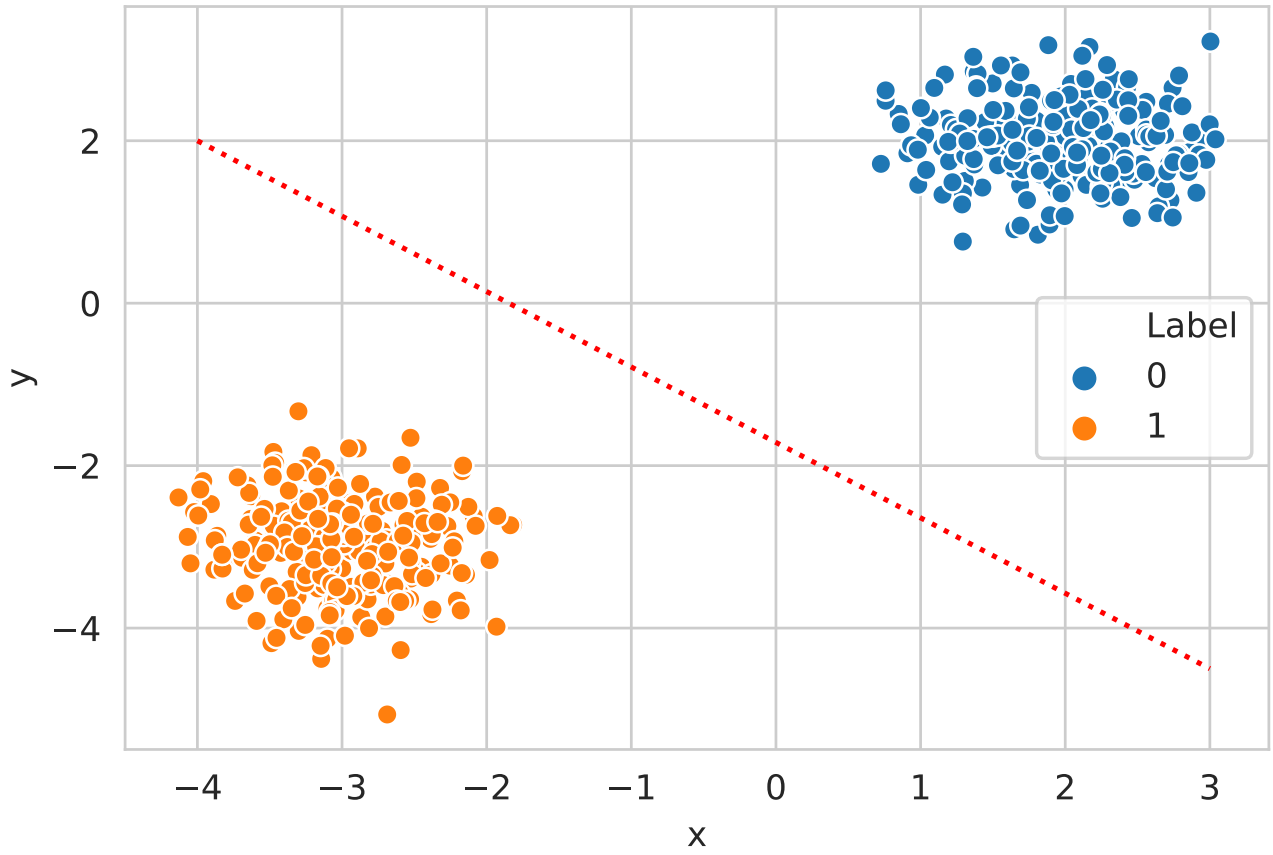
Supervised learning (continued)

- What if our goal is to classify samples into one of two classes?
- We must infer a decision boundary that generalizes to new data.



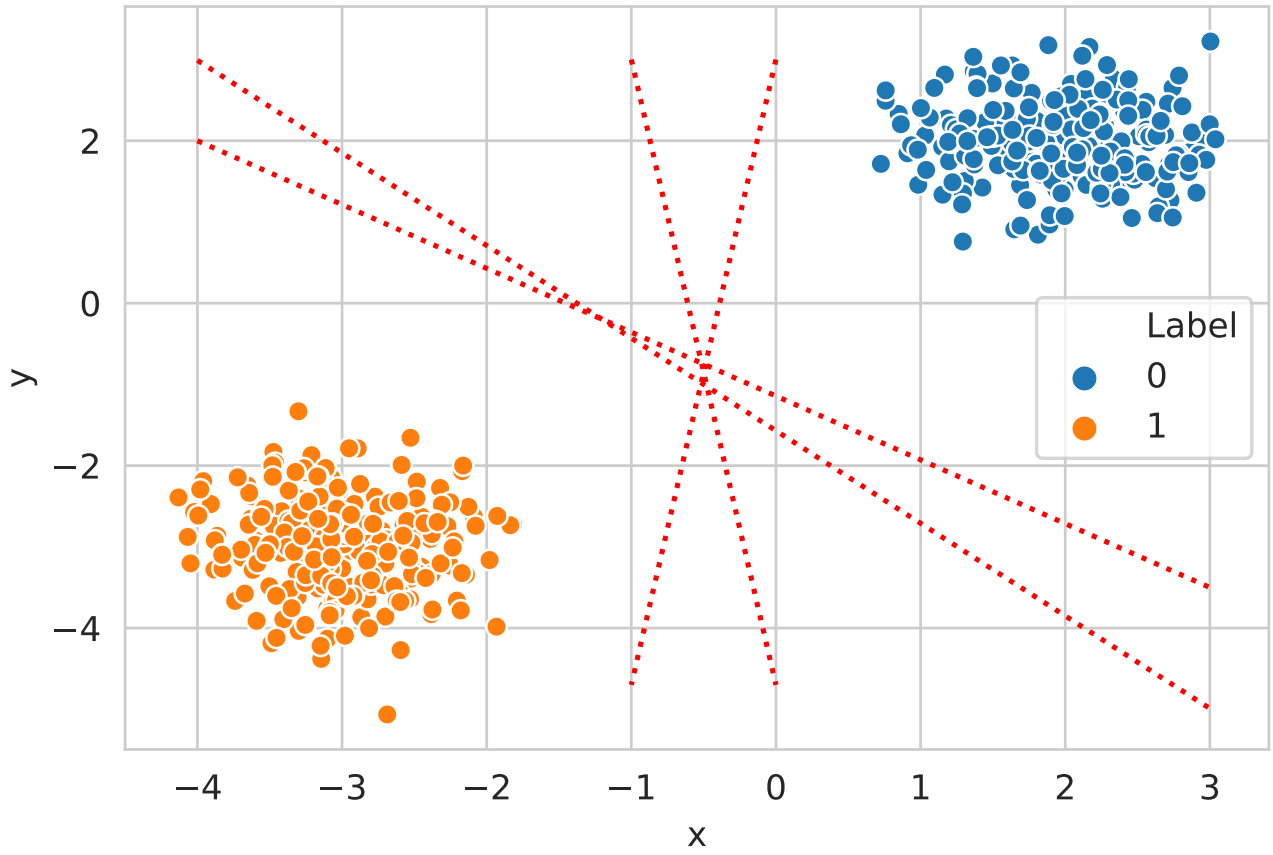
Supervised learning (continued)

- OK, that seems simple.
- But, why should we prefer one solution over another? Or another?



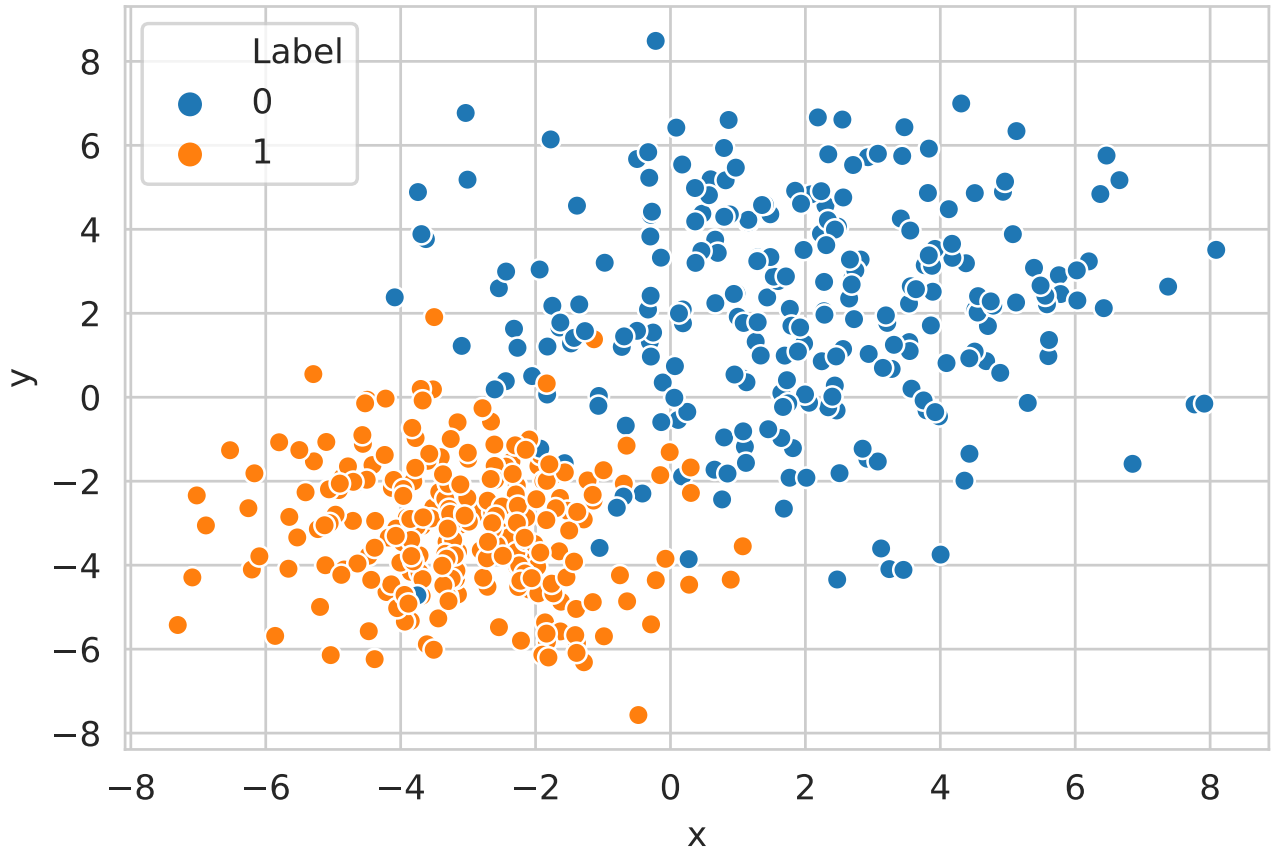
Supervised learning (continued)

- OK, that seems simple.
- But, why should we prefer one solution over another? Or another?



Supervised learning (continued)

- And what the heck do we do here?
- And how should this look in more than two dimensions?



Supervised learning: take home message

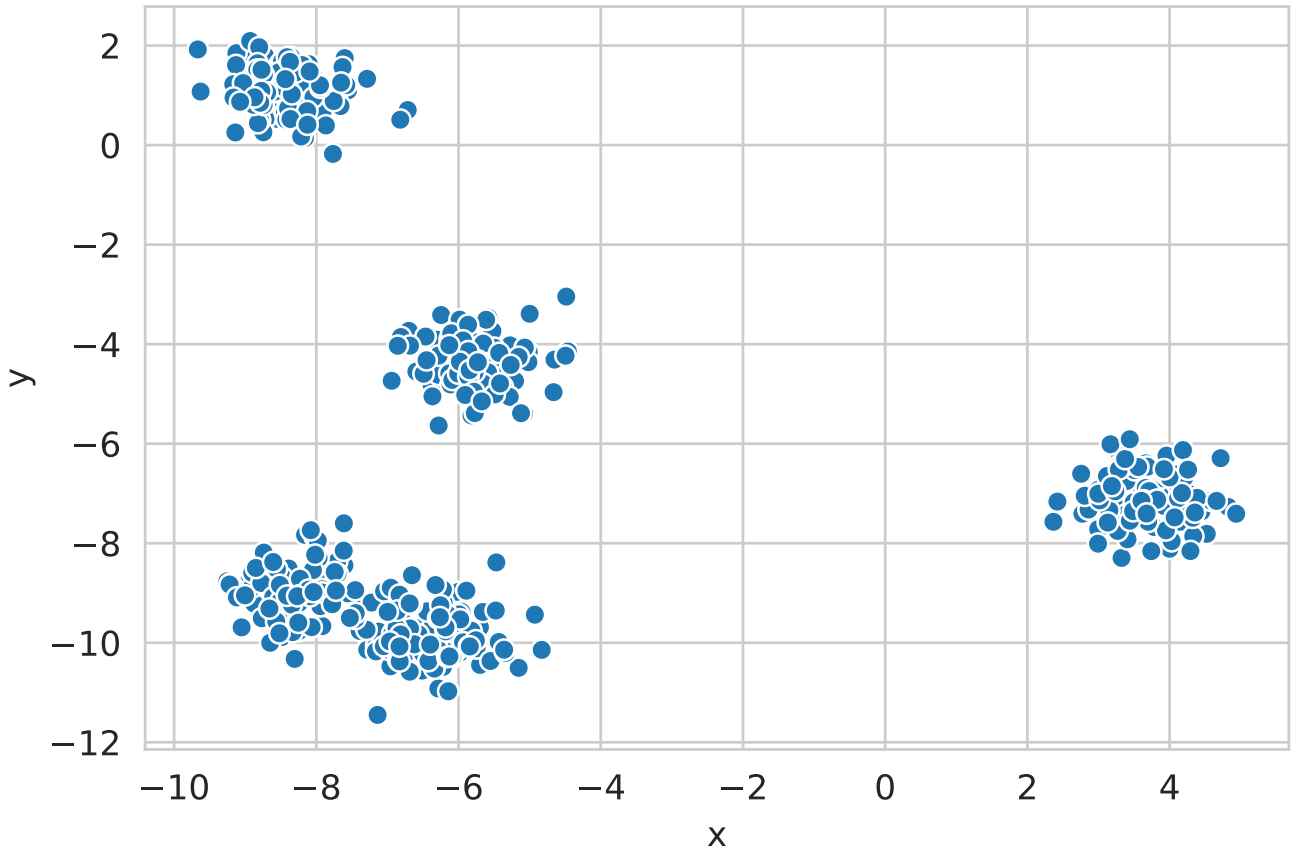
- Supervised learning is about learning from labeled examples – it is sometimes called *learning from a teacher*.
- The goal is to learn a model (i.e. to fit model parameters) that explains the observed data.
- And at the same time is able to generalize to unseen data.
- We will see that there is a delicate balance between fitting the data and guaranteeing generalization to new data – which is the ultimate goal.
- Models we will see: linear discriminants and regression, Support Vector Machines (SVMs), kernel machines, decision trees.

Unsupervised learning: learning without teachers

- Can we learn even without a teacher?
- Well, even very small children are able to learn via exploration of their environment.
- And, after all the amount of unlabeled data vastly outnumbers the available labeled data.

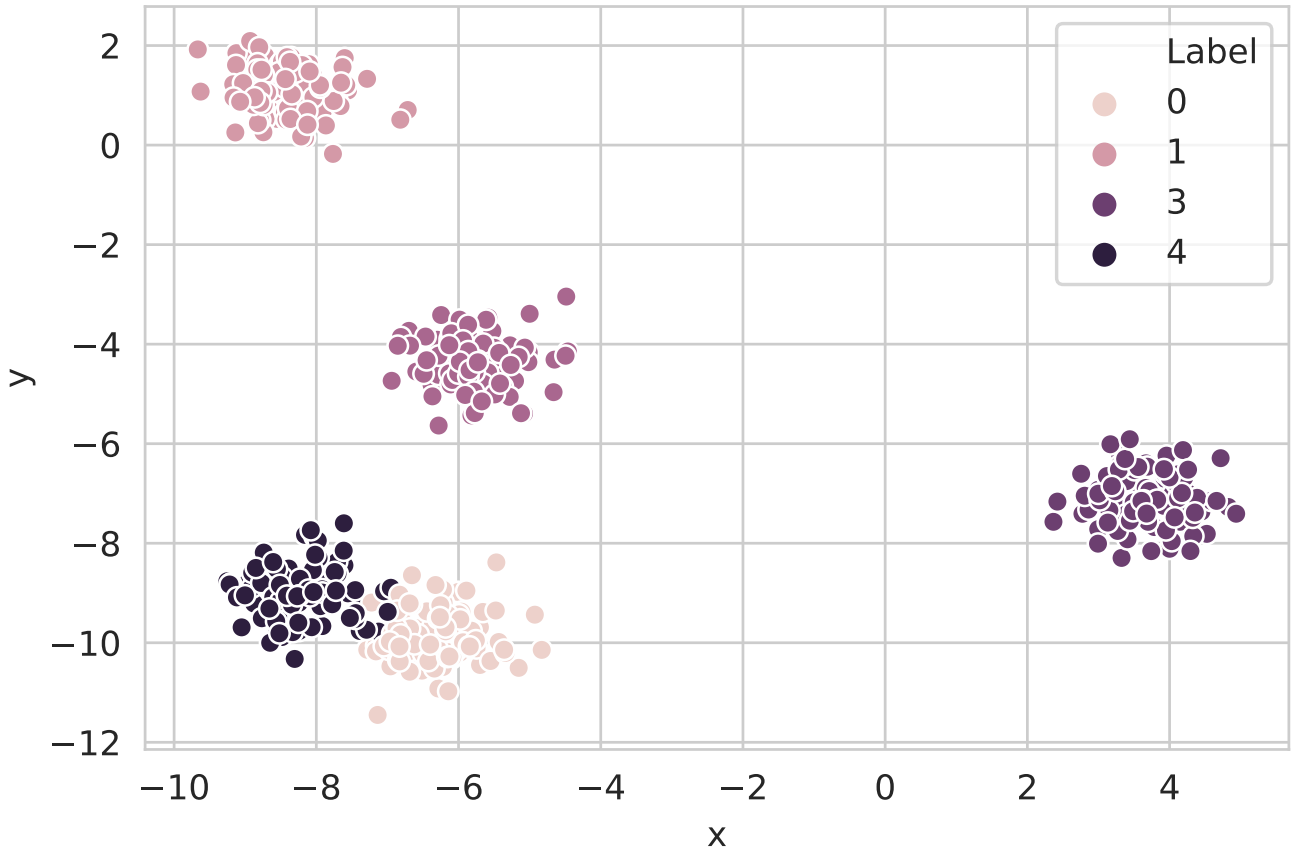
Unsupervised learning: pure data

- Let's say someone gives us some data (in two dimensions).
- Say, something like this:



Unsupervised learning: recovering latent structure

- We would like to learn the structure of the data.
- And recover a hypothesis like this:



Unsupervised learning: take home message

- Again, we would like to learn a hypothesis that generalizes to new data we want to apply the model to.
- Unsupervised learning is about learning from unlabeled data.
- There is actually a spectrum of supervision regimes: unsupervised, semi-supervised, weakly-supervised, self-supervised, fully-supervised...
- Learning from non-fully supervised data is an extremely hot topic in machine learning today.
- Models we will see: K-means clustering, agglomerative clustering, Principal Component Analysis (PCA), t-SNE.

Unsupervised learning: take home message

- A slide borrowed from Yann LeCun:

"Pure" Reinforcement Learning (cherry)
- The machine predicts a scalar reward given once in a while.
- **A few bits for some samples**

Supervised Learning (icing)
- The machine predicts a category or a few numbers for each input
- Predicting human-supplied data
- **10→10,000 bits per sample**

Unsupervised/Predictive Learning (cake)
- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- **Millions of bits per sample**

Who the heck am I?

What I do

- Visual recognition: local pyramidal features, color representations for object recognition, semi-supervised and transductive approaches, action recognition.
- Person re-identification: iterative sparse ranking, semi-supervised approaches to local manifold estimation.
- Multimedia and HCI for cultural heritage: visual profiling of museum visitors, knowledge management for cultural heritage resources, personalizing cultural heritage experiences, human-computer interaction.
- Deep learning: applied and theoretical models for visual recognition, network compression, lifelong learning, reinterpretation of classical approaches in modern learning contexts.
- Other random interests: functional programming languages, operating systems that don't suck, long-distance bicycle touring, Emacs, the Grateful Dead.

Methodology

Vectors and vector spaces

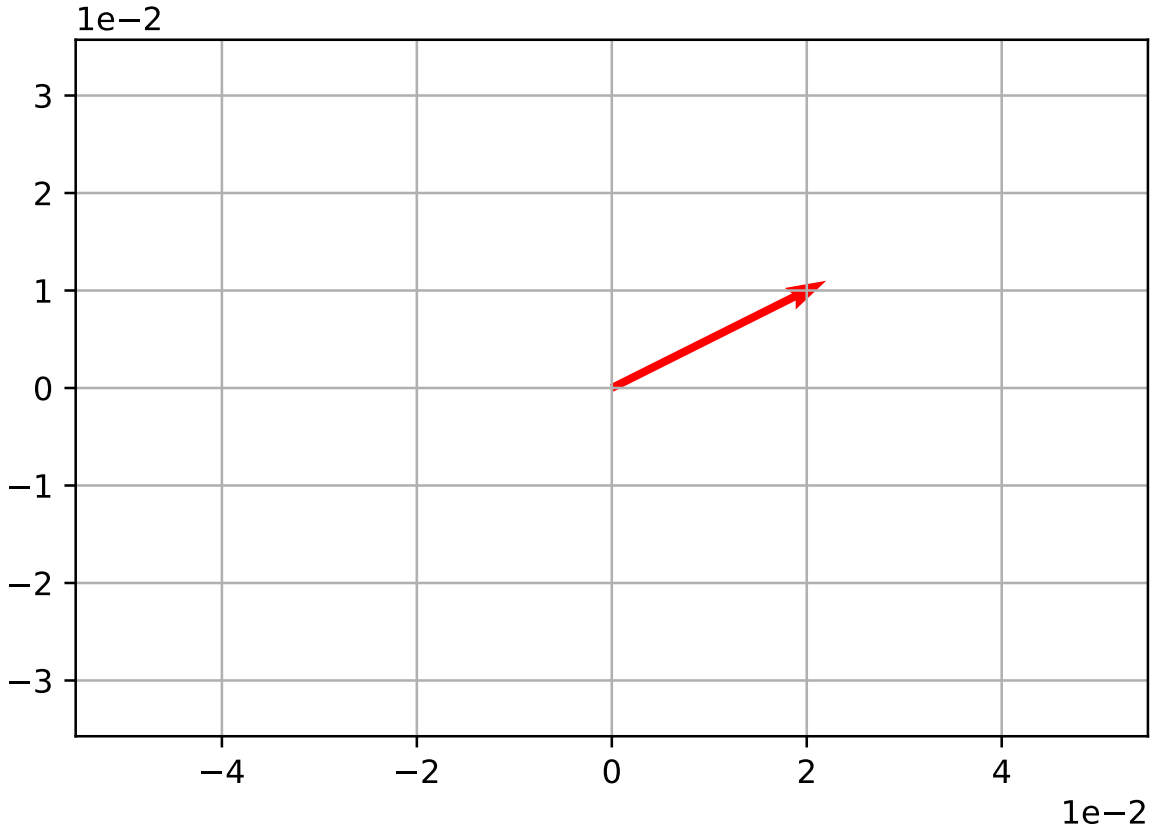
- Vectors and vector spaces are fundamental to linear algebra.
- Vectors describe lines, planes, and hyperplanes in space.
- They allow us to perform calculations that explore relationships in multi-dimensional spaces.
- At its simplest, a vector is a mathematical object that has both magnitude and direction.
- We write vectors using a variety of notations, but we will usually write them like this:

$$\mathbf{v} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

- The boldface symbol lets us know it is a vector.

Vectors and vector spaces (continued)

- What does it mean to have direction and magnitude?
- Well, it helps to look at a visualization (in at most three dimensions):



Vectors and vector spaces (continued)

More formally, we say that $\mathbf{v}$ is a vector in n dimensions (or rather, $\mathbf{v}$ is a vector in the vector space $\mathbb{R}^n$) if:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

for $v_i \in \mathbb{R}$. Note that we use regular symbols (i.e. not boldfaced) to refer to the individual elements of $\mathbf{v}$.

Operations on vectors

Definition (Fundamental vector operations)

- Vector addition: if $\mathbf{u}$ and $\mathbf{v}$ are vectors in $\mathbb{R}^n$, then so is $\mathbf{w} = \mathbf{u} + \mathbf{v}$ (where we define $w_i = u_i + v_i$).
- Scalar multiplication: if $\mathbf{v}$ is a vector in $\mathbb{R}^n$, then so is $\mathbf{w} = c\mathbf{v}$ for any $c \in \mathbb{R}$ (we define $w_i = cv_i$).
- Scalar (dot) product: if $\mathbf{u}$ and $\mathbf{v}$ are vectors in $\mathbb{R}^n$, we define the scalar or dot product as:

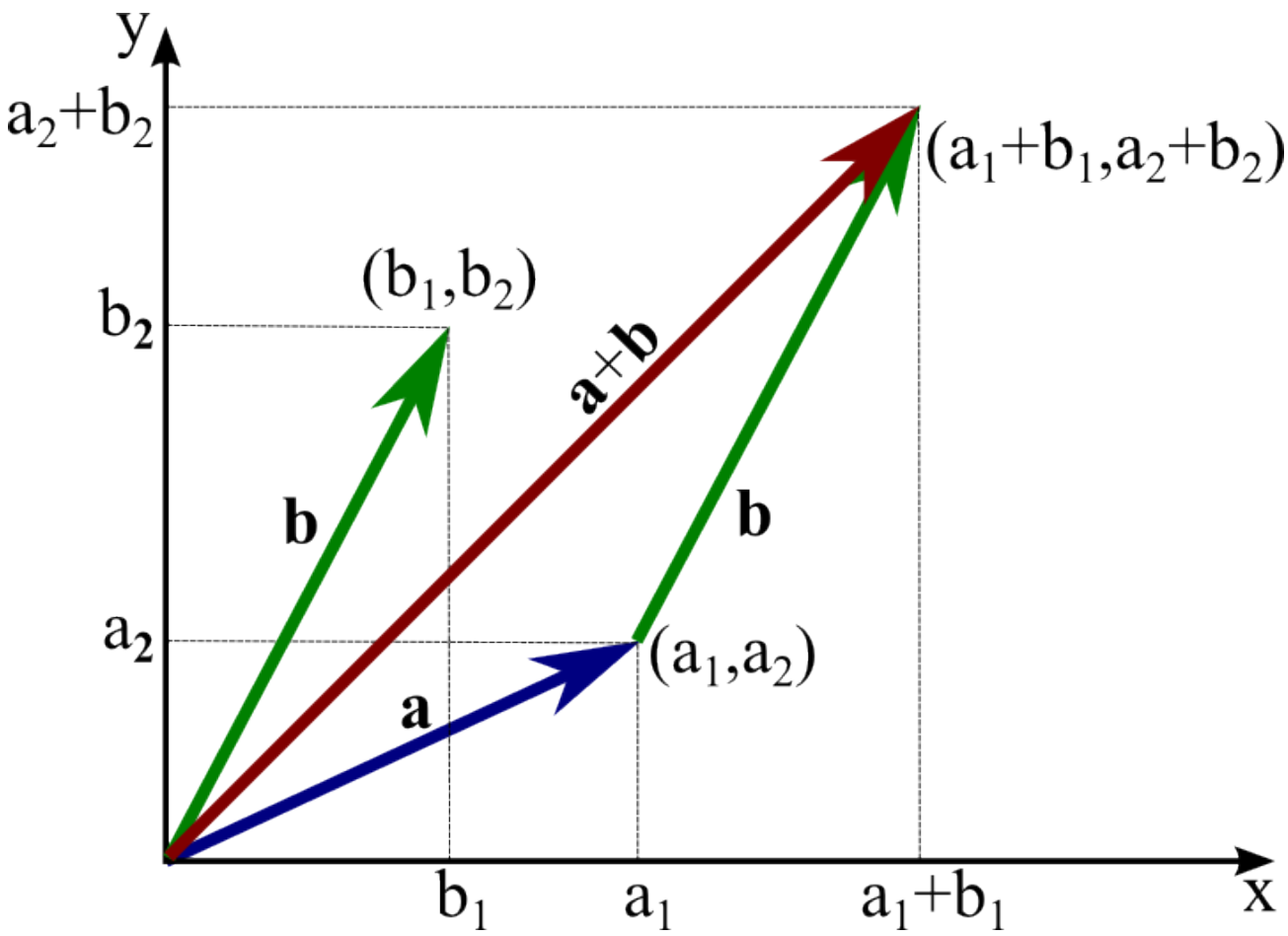
$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

- Vector norm (or magnitude, or length): if $\mathbf{v}$ is a vector in $\mathbb{R}^n$, then we define the norm or length of $\mathbf{v}$ as:

$$||\mathbf{u}|| = \sqrt{\mathbf{u} \cdot \mathbf{u}}$$

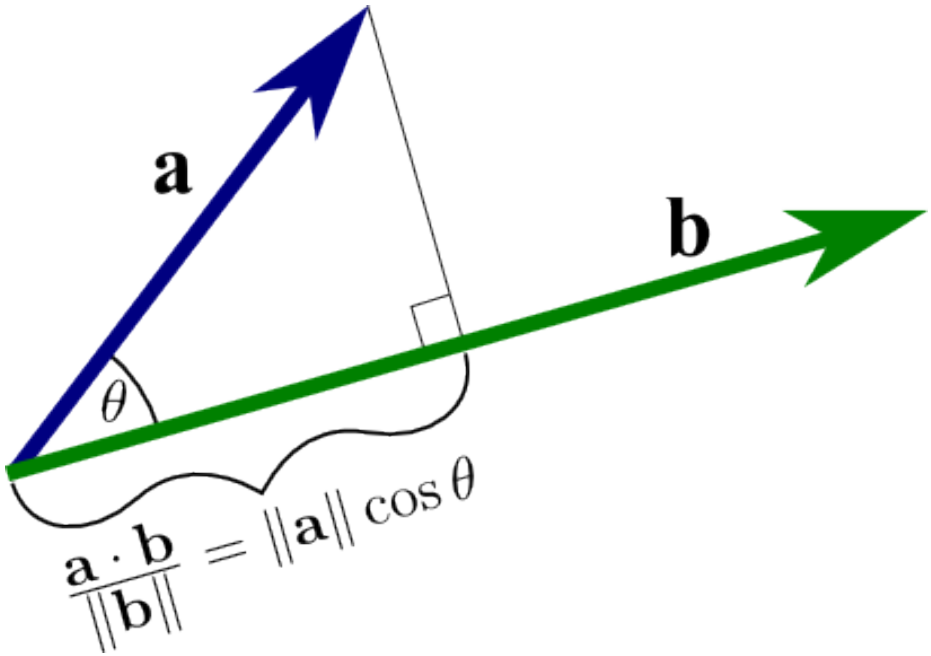
Visualizing vectors (in 2D)

- Vector addition is easy to interpret in 2D:



Visualizing the dot product

- The scalar or dot product is related to the directions and magnitudes of the two vectors:



- In fact, it is easy to recover the cosine between any two vectors.
- Note that these properties generalize to any number of dimensions.
- Question: how can we test it two vectors are perpendicular (orthogonal)?

Matrices: basics

- A matrix arranges numbers into rows and columns, like this:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Note that matrices are generally named as a capital, boldface letter. We refer to the elements of the matrix using the lower case equivalent with a subscript row and column indicator:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}$$

- Here we say that $\mathbf{A}$ is a matrix of size 2×3.
- Equivalently: $\mathbf{A} \in \mathbb{R}^{2\times 3}$.

Matrices: arithmetic operations

- Matrices support common arithmetic operations:
- To add two matrices of the same size together, just add the corresponding elements in each matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix}$$

- Each matrix has two rows of three columns (so we describe them as 2×3 matrices).
- Adding matrices $\mathbf{A} + \mathbf{B}$ results in a new matrix $\mathbf{C}$ where $c_{i,j} = a_{i,j} + b_{i,j}$.
- This *elementwise* definition generalizes to subtraction, multiplication and division.

Matrices: arithmetic operations (continued)

- In the previous examples, we were able to add and subtract the matrices, because the operands (the matrices we are operating on) are conformable for the specific operation (in this case, addition or subtraction).
- To be conformable for addition and subtraction, the operands must have the same number of rows and columns
- There are different conformability requirements for other operations, such as multiplication.

Matrices: unary arithmetic operations

- The negation of a matrix is just a matrix with the sign of each element reversed:

$$C = \begin{bmatrix} -5 & -3 & -1 \\ 1 & 3 & 5 \end{bmatrix}$$

$$-C = \begin{bmatrix} 5 & 3 & 1 \\ -1 & -3 & -5 \end{bmatrix}$$

- The transpose of a matrix switches the orientation of its rows and columns.
- You indicate this with a superscript T, like this:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Matrices: matrix multiplication

- Multiplying matrices is a little more complex than the elementwise arithmetic we have seen so far.
- There are two cases to consider, scalar multiplication (multiplying a matrix by a single number)

$$2 \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix}$$

- And dot product matrix multiplication:

$$\mathbf{AB} = \mathbf{C}, \text{ where } c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

- What can we infer about the conformable sizes of **A** and **B**? What is the size of **C**.

Matrices: multiplication is just dot products

- To multiply two matrices, we are really calculating the dot product of rows and columns.
- We perform this operation by applying the RC rule - always multiplying (dotting) Rows by Columns.
- For this to work, the number of columns in the first matrix must be the same as the number of rows in the second matrix so that the matrices are conformable.
- An example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 9 & 8 \\ 7 & 6 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

Matrices: inverses

- The identity matrix $\mathbf{I}$ is a square matrix with all ones on the diagonal, and zeros everywhere else.
- So, $\mathbf{IA} = \mathbf{BI}$, and $\mathbf{Iv} = \mathbf{v}$.
- The inverse of a square matrix $\mathbf{A}$ is denoted $\mathbf{A}^{-1}$.
- $\mathbf{A}^{-1}$ is the unique (if it exists) matrix such that:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

Matrices: solving systems of equations

- We can now use this to our advantage:

$$\begin{bmatrix} 67.9 & 1.0 \\ 61.9 & 1.0 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 170.85 \\ 122.50 \end{bmatrix}$$

- Multiplying both sides by the inverse:

$$\begin{bmatrix} 67.9 & 1.0 \\ 61.9 & 1.0 \end{bmatrix}^{-1} \begin{bmatrix} 67.9 & 1.0 \\ 61.9 & 1.0 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 67.9 & 1.0 \\ 61.9 & 1.0 \end{bmatrix}^{-1} \begin{bmatrix} 170.85 \\ 122.50 \end{bmatrix}$$

- And we have:

$$\mathbf{I} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 67.9 & 1.0 \\ 61.9 & 1.0 \end{bmatrix}^{-1} \begin{bmatrix} 170.85 \\ 122.50 \end{bmatrix}$$

Matrices: linear versus affine

- Matrix multiplication computes linear transformations of vector spaces.
- We are also interested in affine transformations that don't necessarily preserve the origin:
- An affine transformation is a linear transformation followed by a translation:

$$f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$$

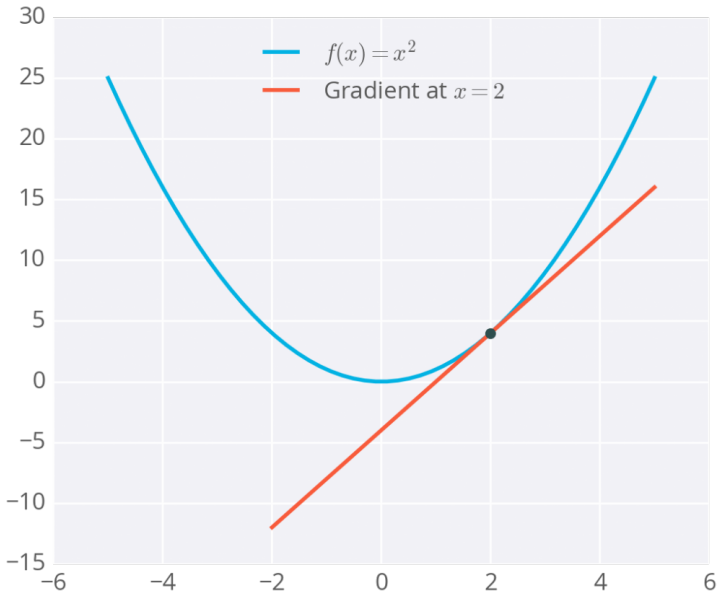
- Note: an affine transformation in n dimensions can be modeled by a linear transformation in $n + 1$ dimensions.

A general structure for dense data

- There is nothing magic about one dimension (vectors) or two dimensions (matrices).
- In fact, the tools we use are completely generic in that we can define dense, homogeneous arrays of numeric data of any dimensionality.
- The generic term for this is a tensor, and all of the math generalizes to arbitrary dimensions.
- Example: a color image is naturally modeled as a tensor in three dimensions (two spatial, one chromatic).
- Example: a batch of b color images of size 32×32 is easily modeled by simply adding a new dimension: $\mathbf{B} \in \mathbb{R}^{b \times 32 \times 32 \times 3}$.

Calculus: An illustrative example

- Let's say we want to find the minimal value of the function $f(x) = x^2$.
- Here's a recipe:
  1. Start with an initial guess x_0.
  2. Take a small step in the direction of steepest descent; call this x_{i+1}.
  3. If $|f(x_{i+1}) - f(x_i)| < \varepsilon$, stop.
  4. Otherwise: repeat from 2.



Calculus: Gradient descent

- Maybe the only thing imprecise about this recipe is the definition of small step in the direction of steepest descent.
- Well, in one variable we know how to do this:

$$x_{i+1} = x_i - \eta \frac{d}{dx} f(x_i)$$

- So the derivative gives us the direction, and the parameter η defines what "small" means.
- This recipe also works in more dimensions:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \eta \nabla_x f(\mathbf{x}_i)$$

- Let's dissect this...

Calculus: Fitting models with gradient descent

- Many of the models we will see have a form like:

$$f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

- That is: function f is parameterized by parameters $\boldsymbol{\theta}$.
- Goal: find a $\boldsymbol{\theta}^*$ that optimize some fitness criterion $\mathcal{L}$ on data $\mathbf{D}$:

$$\boldsymbol{\theta}^* = \arg\min \mathcal{L}(\mathbf{D}, \boldsymbol{\theta})$$

- Example (least squares):

$$\begin{aligned} D &= \{ (x_i, y_i) \mid 1 \leq i \leq n \} \\ \boldsymbol{\theta} &= \begin{bmatrix} m \\ b \end{bmatrix} \\ \mathcal{L}(D, \boldsymbol{\theta}) &= \sum_i ||(mx_i + b) - y_i||_2 \end{aligned}$$

Rules: Overview

- Here I want to give a simple recipe that we will try to follow for most ML problems we will look at.
- These are a loose sort of best practices you can follow when working with data.
- There is, of course, no strict set of rules that you can (or should) blindly follow.
- In fact, one of the overarching objectives of this course is to give you some exposure and experience to a set of tools and techniques.
- Enough so that you can develop your own practices and draw well-founded conclusions about your own learning and data analysis problems.

Rules Step 1a: Data design

- The first step, of course, is to get your hands on some data.
- Depending on the data, you should design a Pandas `Dataframe` to encapsulate it.
- Think about data types – are features continuous, categorical, or discrete.
- Pick good names for the columns in your dataset – you will be using them a lot, `['A', 'B', ..., 'Z']` is probably a bad idea.

Rules Step 1b: Getting a feel for the data

- When you have data in a `DataFrame`, you can start to get a feel for it.
- Look at the descriptive statistics that `describe()` gives you.
- What you are looking for are surprises and insights.
- Are there any undefined values in your data? How will you deal with them?

Rules Step 1c: Visualization

- An important tool for data exploration is visualization.
- This is especially true if you have very many features (high dimensionality).
- Use simple plots, scatter plots, and histograms to get a better picture of the nature and behavior of your data.
- More advanced plotting capabilities can be employed as needed:
  - Seaborn: prettier plots, better Pandas integration
  - Bokeh: interactive plotting widgets, great for exploration.

Rules Step 1d: Normalization and standardization

- Are some (or all) features badly scaled with respect to others?
- Do you have categorical variables that might need to be embedded or mapped to other spaces?
- You might think about standardization or normalization at this point.
- However, the decision about how to preprocess data is often intimately tied to downstream modeling decisions.

Rules Step 1e: Iterate

- Finally, repeat.
- What you discover via visualization and data exploration can often change how you decide to model your data.
- It is most important to ensure you understand your data and have a good data model going forward.
- Take your time.

Rules Step 2a: Decide how to model your problem

- What type of learning problem are you faced with?
- Is it supervised (do you have target values?):
  - Is it a regression (continuous target) problem?
  - Is it a classification (categorical outputs) problem?
  - During exploratory data analysis (step 1) you should have acquired an idea of which features are correlated with targets.
- Is it an unsupervised learning problem (do you only have blobs of data?):
  - In this case during exploratory data analysis you should have acquired an idea if there is latent structure to learn.

Rules Step 2b: Pick a model parameterization

- Depending on which type of learning problem (supervised or unsupervised), you can now think about selecting a model to try.
- Do there appear to be simple and linear correlations between features and targets?
- Or, is the correlation structure not immediately evident (which might indicate that linear models won't work)?
- Whichever model you start with, you should have a good idea of what the model parameters are that will be estimated.
- General advice: start with a simple model and gradually increase complexity.

Rules Step 2c: Understand hyperparameters

- Most models, in addition to the learnable parameters, will have one of more hyperparameters.
- Some of these are architectural choices (e.g. whether to fit both slope AND y-intercept in a regression).
- Some will be continuous parameters that cannot be fit by gradient-based optimization (e.g. regularization wights).
- The important thing here is to be aware of what hyperparameters exist and to pick reasonable defaults.

Rules Step 2d: Understand how to evaluate

- Finally, we need to know how to evaluate the performance of our models.
- For regression, this might be a simple RMS error.
- For classification, you might be interested in accuracy.
- This can be a delicate decision, however.
- Question: let's say you have an unbalanced binary classification problem (one class has 1000x more example than the other). Why might accuracy not be a good choice?

Rules Step 3a: The very least: training/testing

- This might be the easiest, but MOST IMPORTANT step.
- Whenever you are working with machine learning you MUST be sure to work with *independent training and testing* sets.
- These are usually referred to as splits:
  - Training split: a randomly chosen portion (say, 75%) of the data you set aside ONLY for estimating model parameters.
  - Testing split: a portion (the remaining 25%) of the data you use ONLY FOR EVALUATING PERFORMANCE.
- Very important: using independent training/testing splits like this is the *only way to guarantee generalization*.

Rules Step 3a: Even better: training/validation/testing

- If you have enough data, an even better way is to have three splits:
  - Training split: a randomly chosen portion (say, 60%) of the data you set aside ONLY for estimating model parameters.
  - Validation split: a randomly chosen portion (50% of the remaining data) used to monitor learning and to select hyperparameters.
  - Testing split: a portion (the remaining part) of the data you use ONLY FOR EVALUATING PERFORMANCE.
- Later we will see how cross-validation techniques can be used to make the most of available data without violating the independence of train/validation/test splits.

Rules Step 3a: ALWAYS obey this rule

- If you want to draw conclusions about the performance of your models, you must use independent splits.
- *I cannot emphasize this enough*.

Rules Step 4: Fit your model, evaluate, repeat.

- Now we can actually start doing some machine learning.
- Frameworks like sklearn provide tools with a consistent API (e.g. `model.fit()` to estimate parameters).
- Frameworks like sklearn also usually provide most of the evaluation (and splitting) functions you need.
- We usually talk about building a pipeline that, given data and values for hyperparameters:
  1. Fits the model to the training data.
  2. Evaluates the model on the test (or validation) data.
  3. Visualizes model output and/or performance as appropriate.
- Having a pipeline allows us to repeatably perform experiments with different hyperparameters, with different data, etc.

Supervised Learning

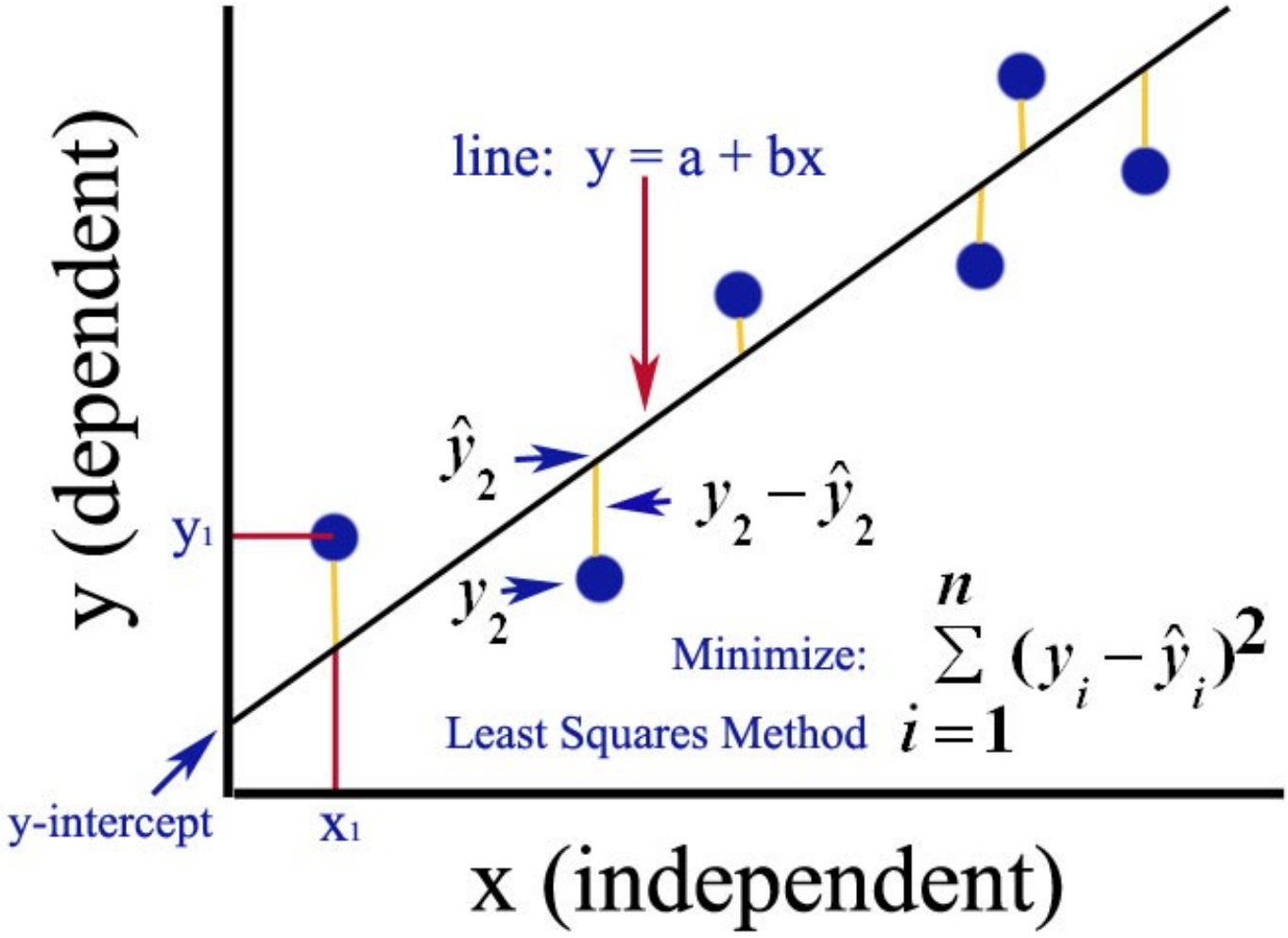
Ordinary Least Squares: The Math

- The Data: matched pairs $\mathbf{x}, y)$ of features $\mathbf{x}$ and targets y.
- The Model:

$$\begin{aligned} \hat{y} \equiv f(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\ \mathcal{L}(D; \mathbf{w}, b) &= \sum_{(\mathrm{x},y)\in D} ||y - f(\mathbf{x}, b)||_2 \end{aligned}$$

- The Parameters: Vector of weights $\mathbf{w}$ and scalar bias b.
- The Hyperparameters: None to speak of.
- Fitting: Efficient, exact, and closed-form solution using psuedo-inverse.

Ordinary Least Squares: The Big Picture

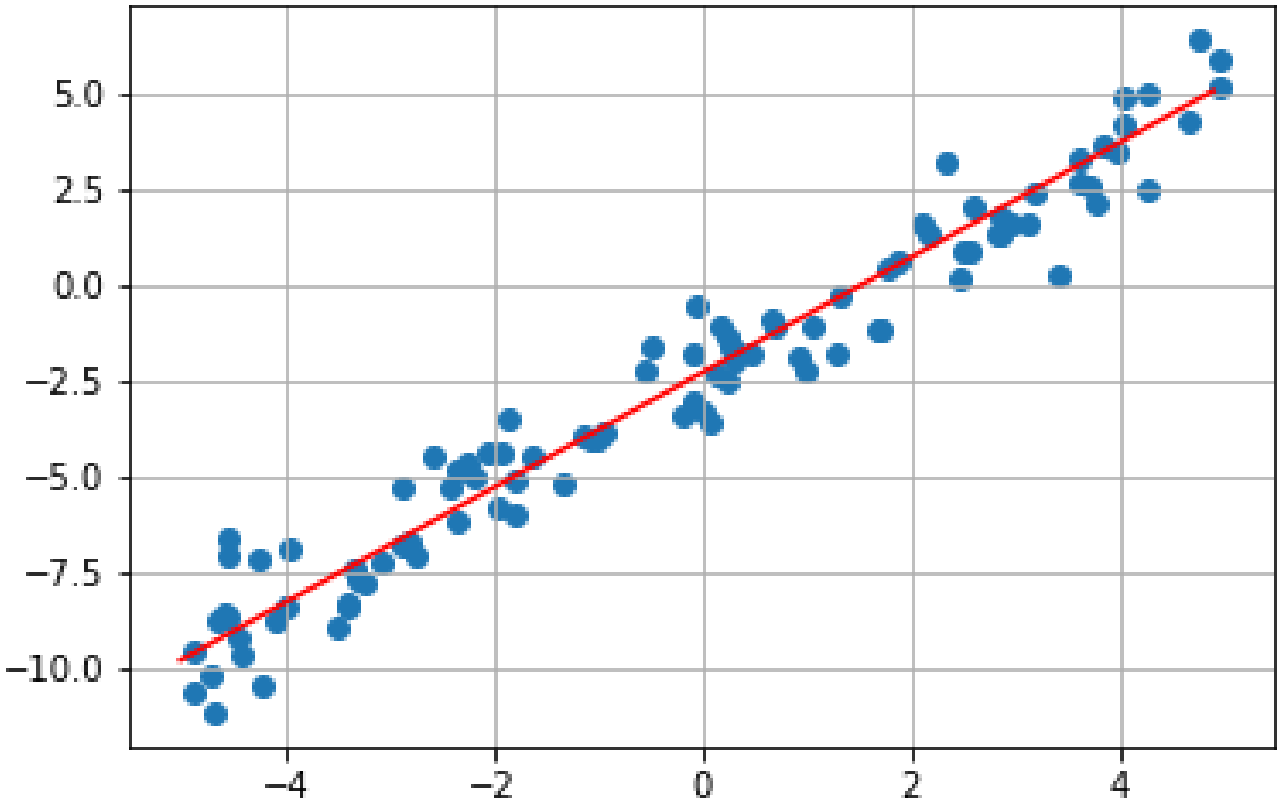


Ordinary Least Squares: Discussion

- Least Squares is simple and effective.
- Having one learned weight for each feature makes it robust to feature scaling.
- Assumes that there is a linear relation between features and target.
- Also assumes the features are independent and thus decorrelated.
- If there are approximate linear relationships between features, the matrix to be inverted can become singular.
- In sklearn: `sklearn.linear_model.LinearRegression`

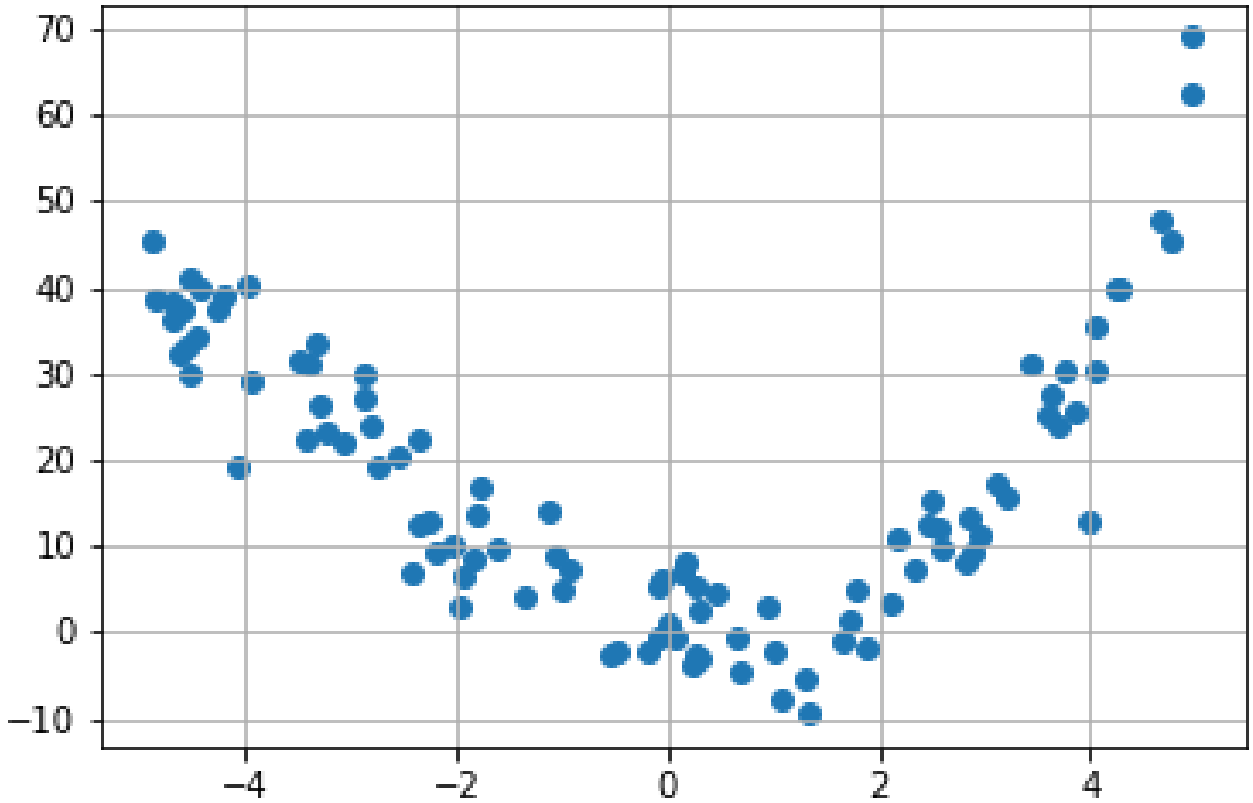
Explicit feature mapping

- We know what to do with this type of estimation problem:



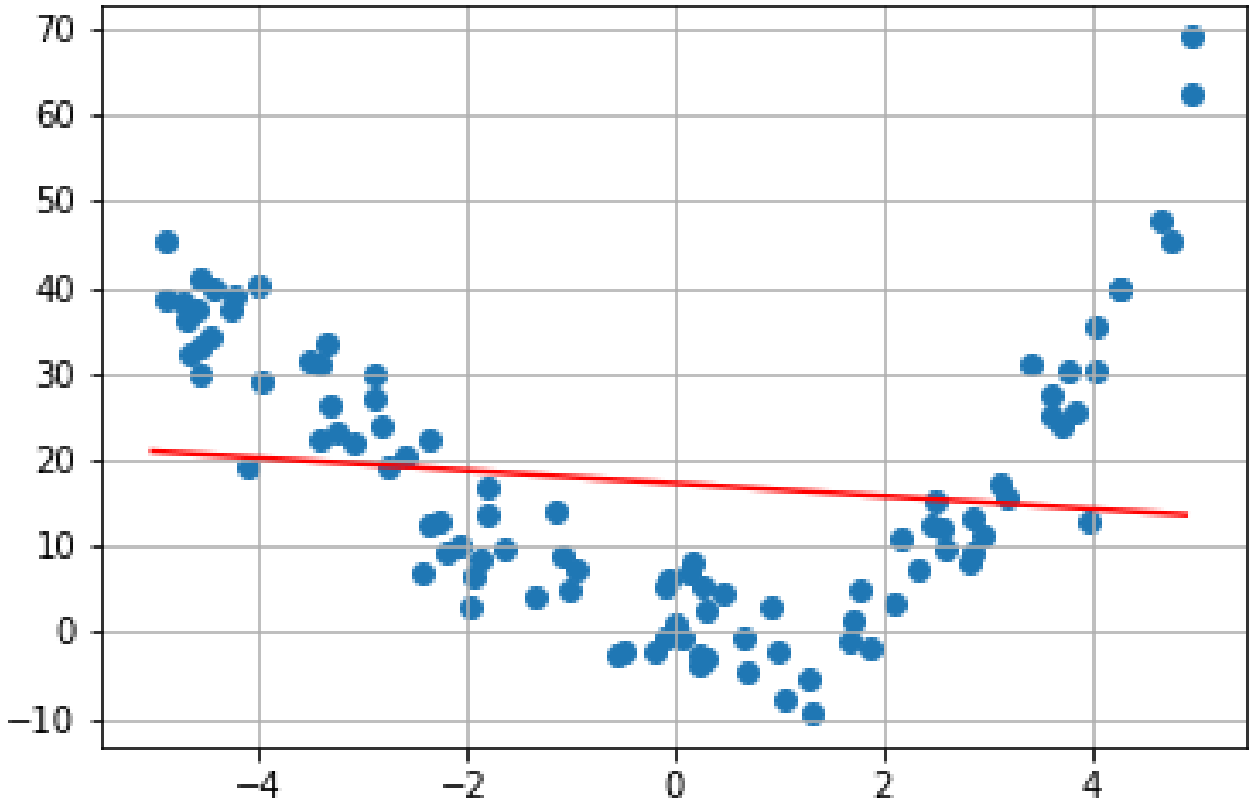
Explicit feature mapping

- What do we do if we have data like this:



Explicit feature mapping

- What do we do if we have data like this:



Explicit feature mapping: How it works

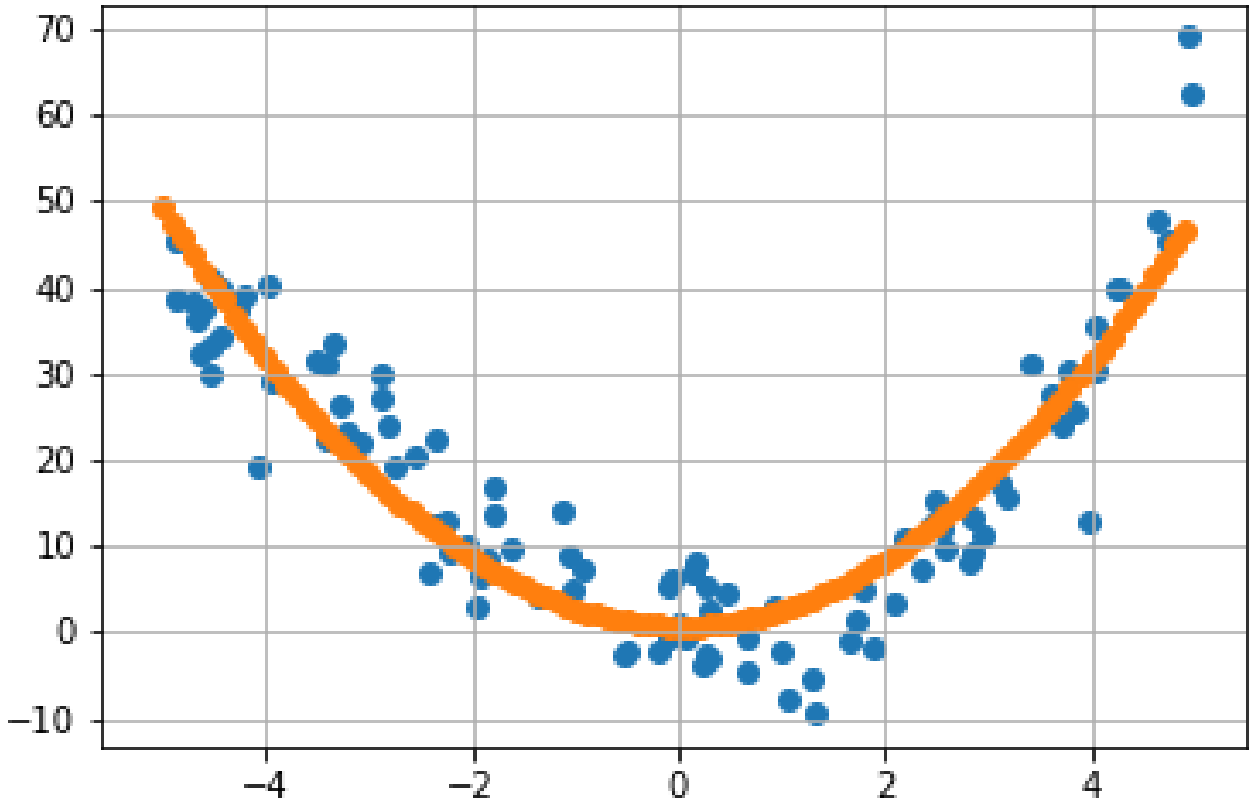
- The general idea is to map our original features into a higher dimensional space.
- For example, we can map linear features (one-dimensional) into a space with a polynomial basis:

$$\begin{aligned} e(x) &= [x\ x^2\ x^3\ \ldots\ x^k]^T \\ f(e(x)) &= \mathbf{w}^T e(x) + b \\ &= w_1 x + w_2 x^2 + \cdots + w_k x^k + b \end{aligned}$$

- So, by fitting a linear model in k features, we are really fitting a polynomial to the data.

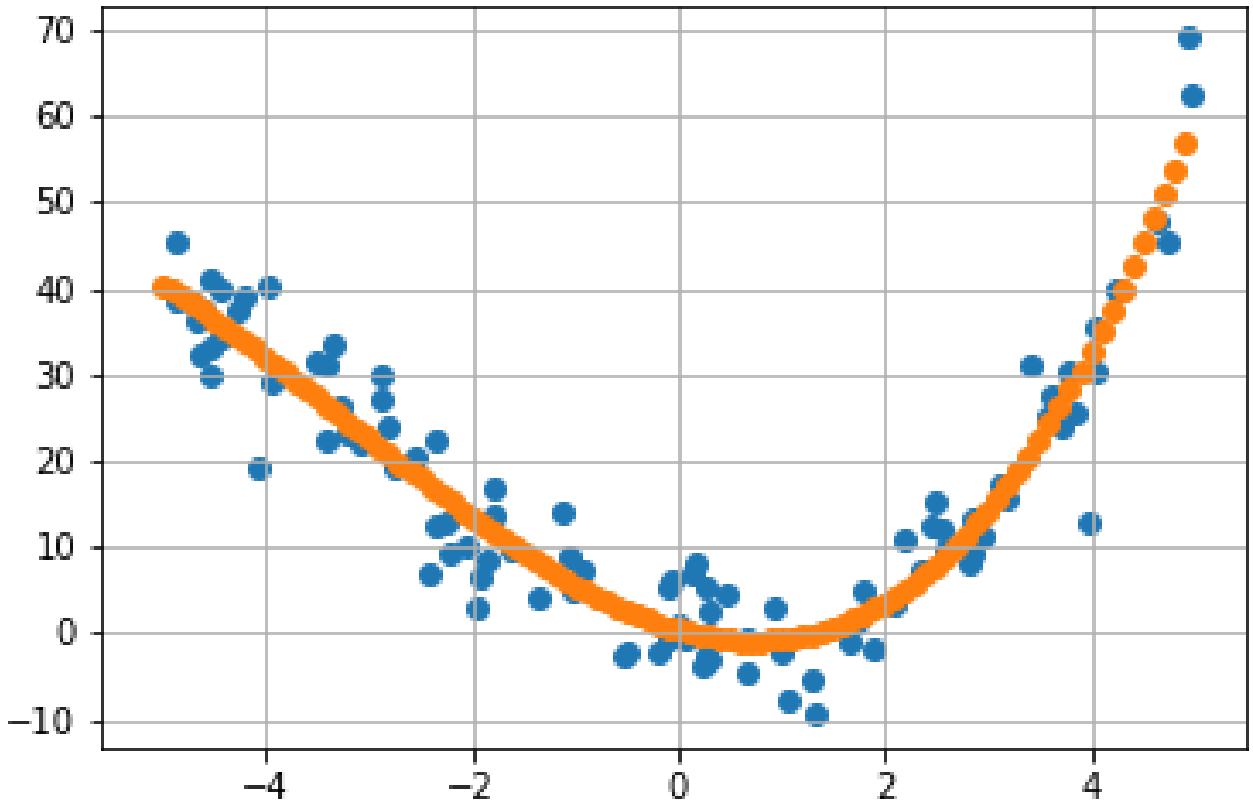
Explicit feature mapping

- Using a degree 2 polynomial embedding:



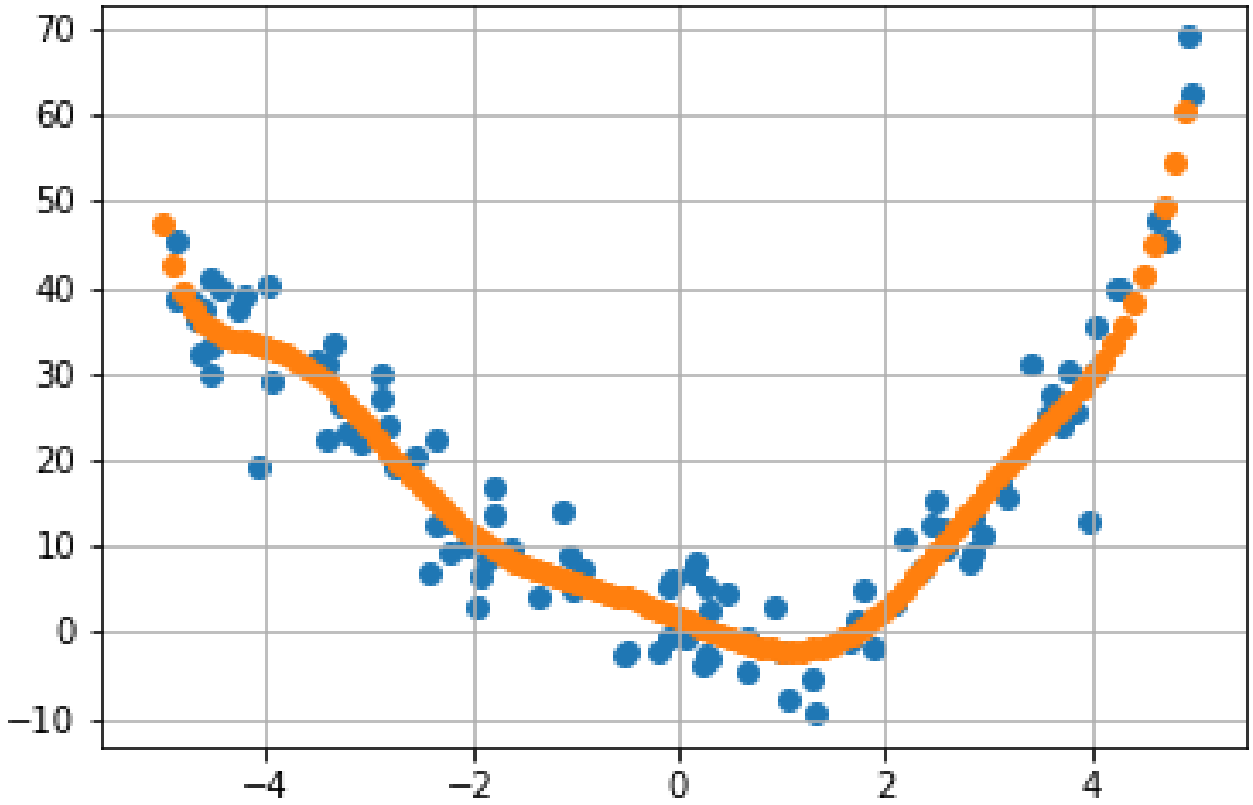
Explicit feature mapping

- Using a degree 3 polynomial embedding (which is correct):



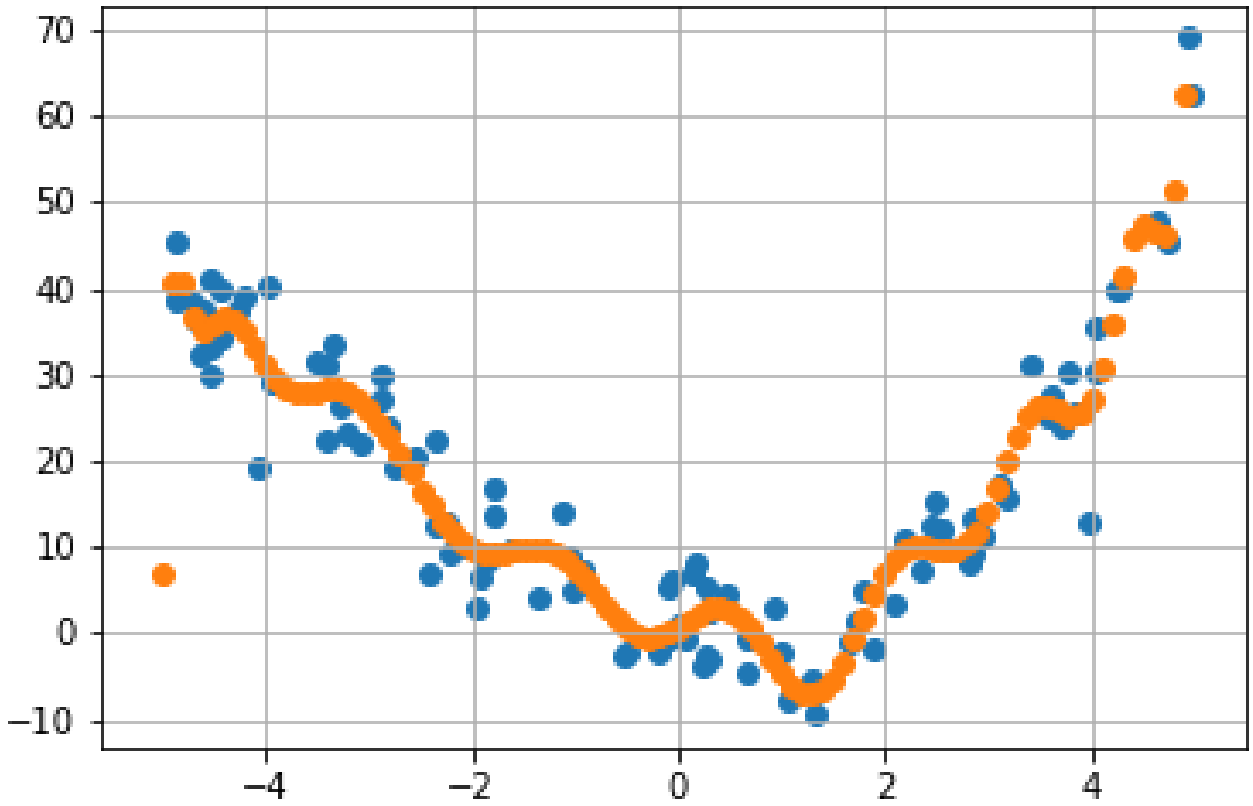
Explicit feature mapping

- Degree 10 – more complex isn't always better:



Explicit feature mapping

- Degree 20 – now we're just fitting noise:



Explicit feature mapping: Discussion

- Mapping features into a higher dimensional space is a powerful tool.
- It allows us to use simple tools (like good old linear regression) to fit non-linear functions to data.
- You must be careful, though, to not increase the power of the representation too much.
- At some point the model will begin capturing the noise and will never generalize.
- In sklearn: `sklearn.preprocessing.PolynomialFeatures`

Classification problems

- Classification problems are usually formulated in terms of a discriminant function and a decision rule.
- The discriminant function tells us something about the similarity of an input to all classes.
- The decision rule tells us how to act on this information.
- Training data: matched pairs $(\mathbf{x}, c_x)$ of features $\mathbf{x}$ and target labels c_x.

KNN: Overview

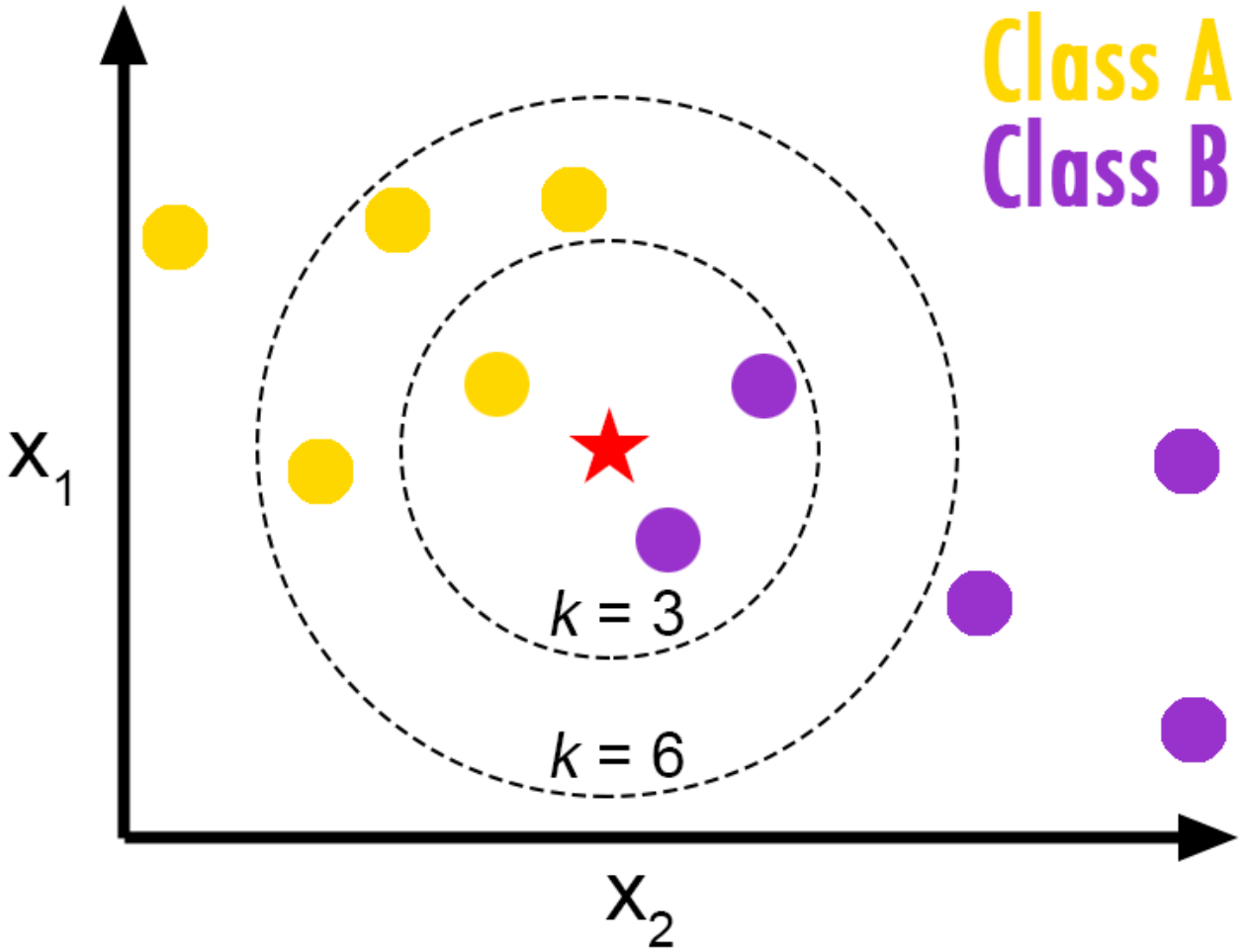
- We will first look at a non-parametric model for classification.
- Non-parametric models are, well, models without any parameters to be learned.
- KNN works by looking at the closest training samples in feature space.
- Its main advantage is that it is simple and intuitive.

KNN: The Algorithm

- The K-nearest Neighbors algorithm doesn't have a fancy mathematical model.
- Given a test sample $\mathbf{x}$ to classify:
  1. Load all training data into memory
  2. For each training example $\mathbf{x}'$ in the data:
     - 2.1 Calculate the distance between $\mathbf{x}$ and $\mathbf{x}'$.
     - 2.2 Add the distance and the index of $\mathbf{x}'$ to an ordered collection (ascending by distance).
  3. Pick the first K entries from the sorted collection.
  4. Get the labels of the selected K entries.
  5. Return the mode of the K selected labels (the most frequent label).

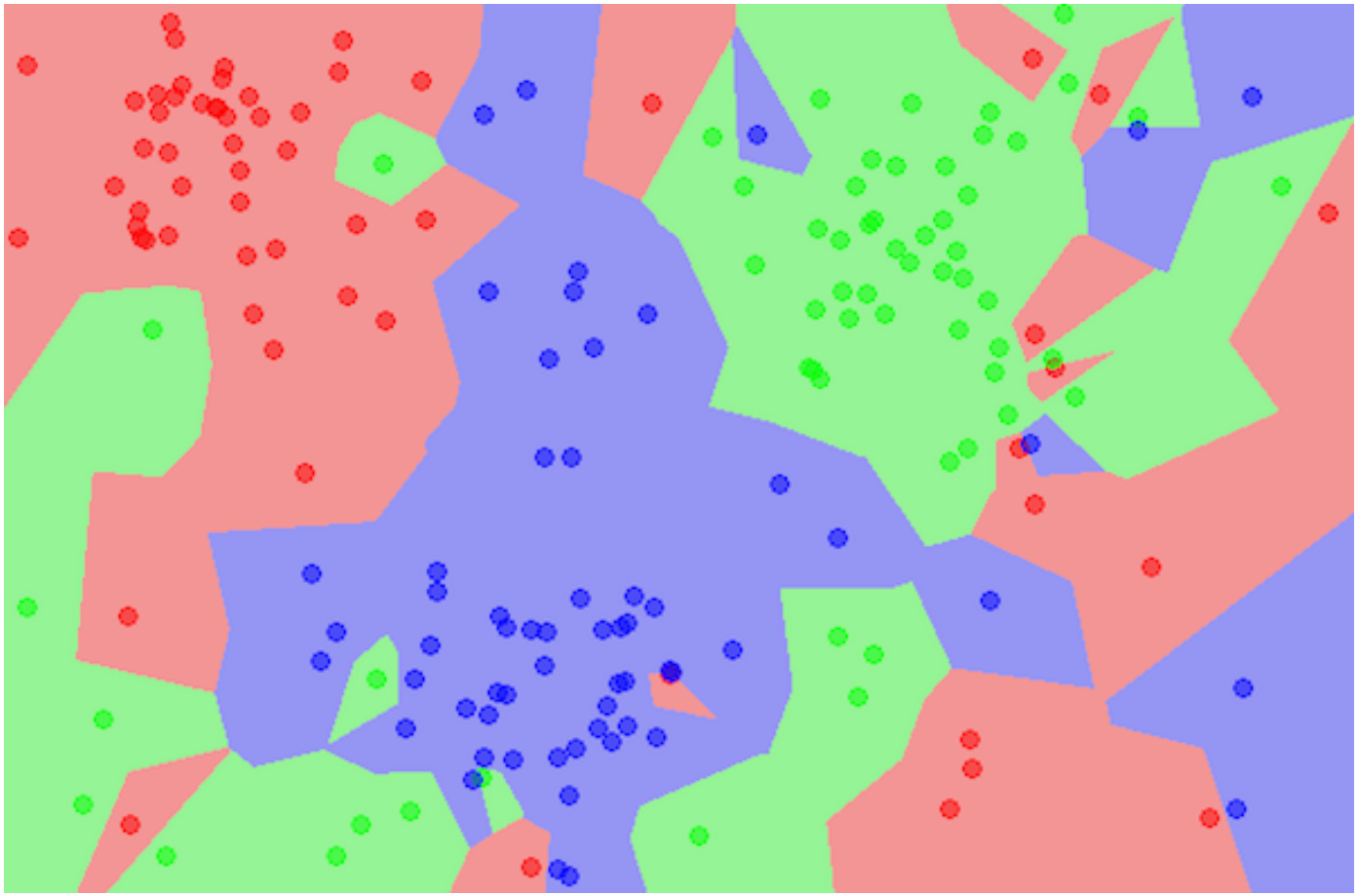
KNN: The Big Picture

- The KNN model is easily visualized:



KNN: A non-linear partitioning of feature space

- KNN scales directly to multi-class problems with complex decision boundaries:



KNN: Discussion

- The KNN algorithm is extremely Simple to implement and works with arbitrary distance metrics.
- It naturally handles multi-class cases, and is optimal in practice with enough representative data.
- Its main disadvantage is that Computation cost is quite high because we need to compute the distance of each test sample to all training samples.
- Plus, we need to keep all training samples on hand, forever.
- In sklearn: sklearn.neighbors.NearestNeighbors

Linear SVMs: Overview

- The Support Vector Machine is one of the most tried and true models for classification.
- The basic theory goes back to the 1950s, but was solidified by Vapnik in the 1990s.
- It addresses many of the problems related to model complexity and overfitting.
- Nice feature: the theory developed by Vapnik lets us extend the SVM to non-linear versions using the kernel trick.

Linear SVMs: The Math

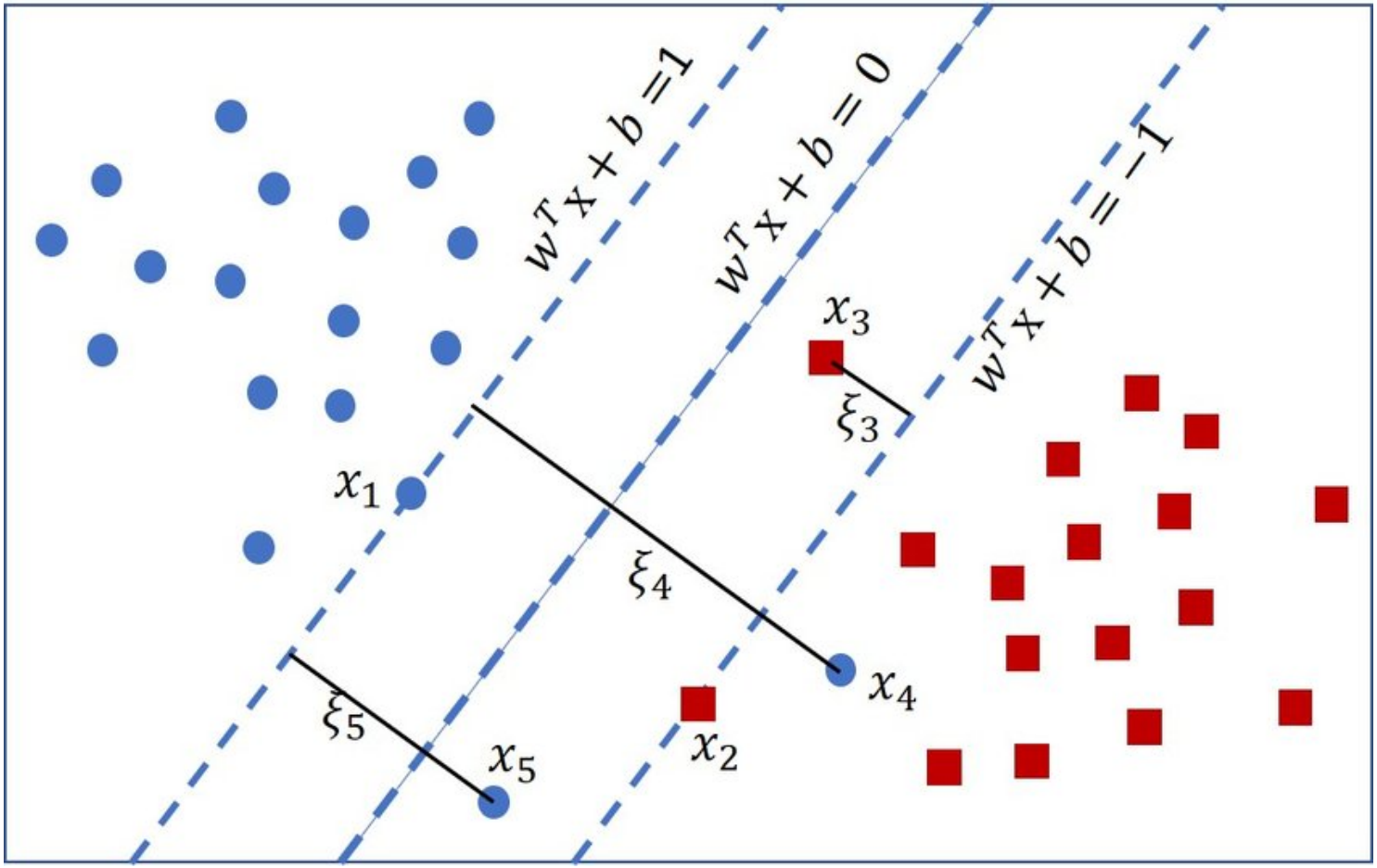
- The Model:

$$
\begin{aligned}
f(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\
\mathcal{L}(D; \mathbf{w}, b) &= \min_{\mathbf{w}} ||\mathbf{w}||_2 + \sum_{(\mathrm{x},y)\in D} C \max(0, 1 - y f(\mathbf{x})) \\
\text{class}(\mathbf{x}) &= \begin{cases} -1 & \text{if } f(\mathbf{x}) \leq 0 \\ +1 & \text{if } f(\mathbf{x}) > 0 \end{cases}
\end{aligned}
$$

- The Parameters: Vector of weights $\mathbf{w}$ and scalar bias b.
- The Hyperparameters: Trade-off parameter C (more if using more "fancy" formulations – should be cross-validated).
- Fitting: Efficient, exact, but iterative solution using convex optimization.

Linear SVMs: The Big Picture

- We are maximizing the margin between classes:



Linear SVMs: Discussion

- SVMs are reliable (convex guarantees global optimum) and robust (theory tells us SVM gives an "optimal" discriminant).
- Still effective in cases where number of dimensions is greater than the number of samples (C parameter).
- A disadvantage is that they do not provide probabilistic estimates of class membership.
- A linear SVM is almost always one of the first things you should try.
- In sklearn: `sklearn.svm.SVC`

Kernel Machines: The Math

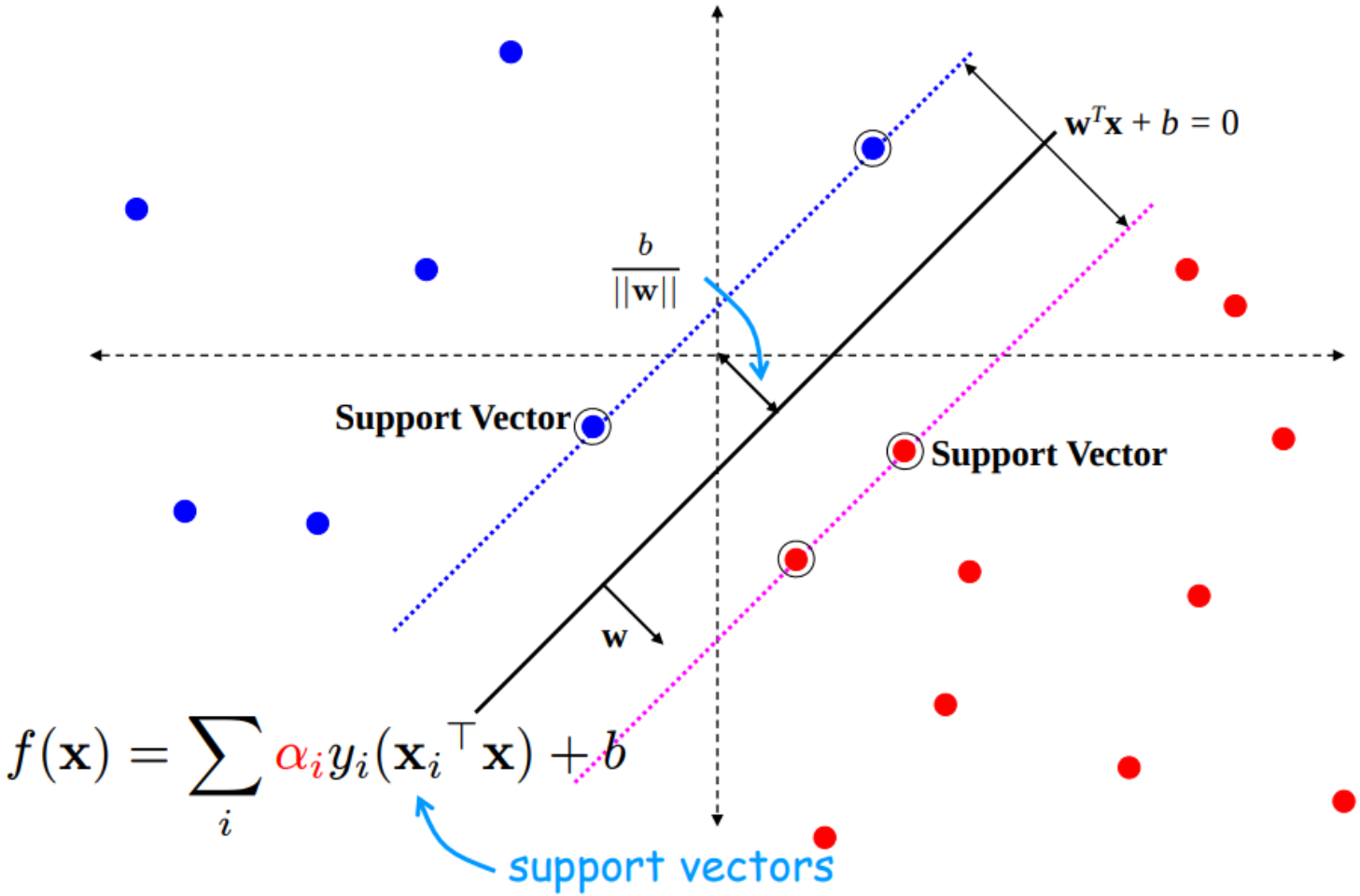
- SVMs can also be formulated in a different way (called the dual formulation:

$$f(\mathbf{x}) = \sum_{(x_i, y_i) \in D} \alpha_i y_i (\mathbf{x}_i^T \mathbf{x}) + b$$

- So, we have as many parameters (α_i) as training samples now.
- At first, this seems absurd – like Nearest Neighbors, we have to keep all our training data around forever.
- In reality, usually only a small fraction of training samples have non-zero α – these are called support vectors.
- The real advantage: formulating the problem in a way that uses only inner products of vectors – this allows us to implicitly change the metric we are using to compare vectors.

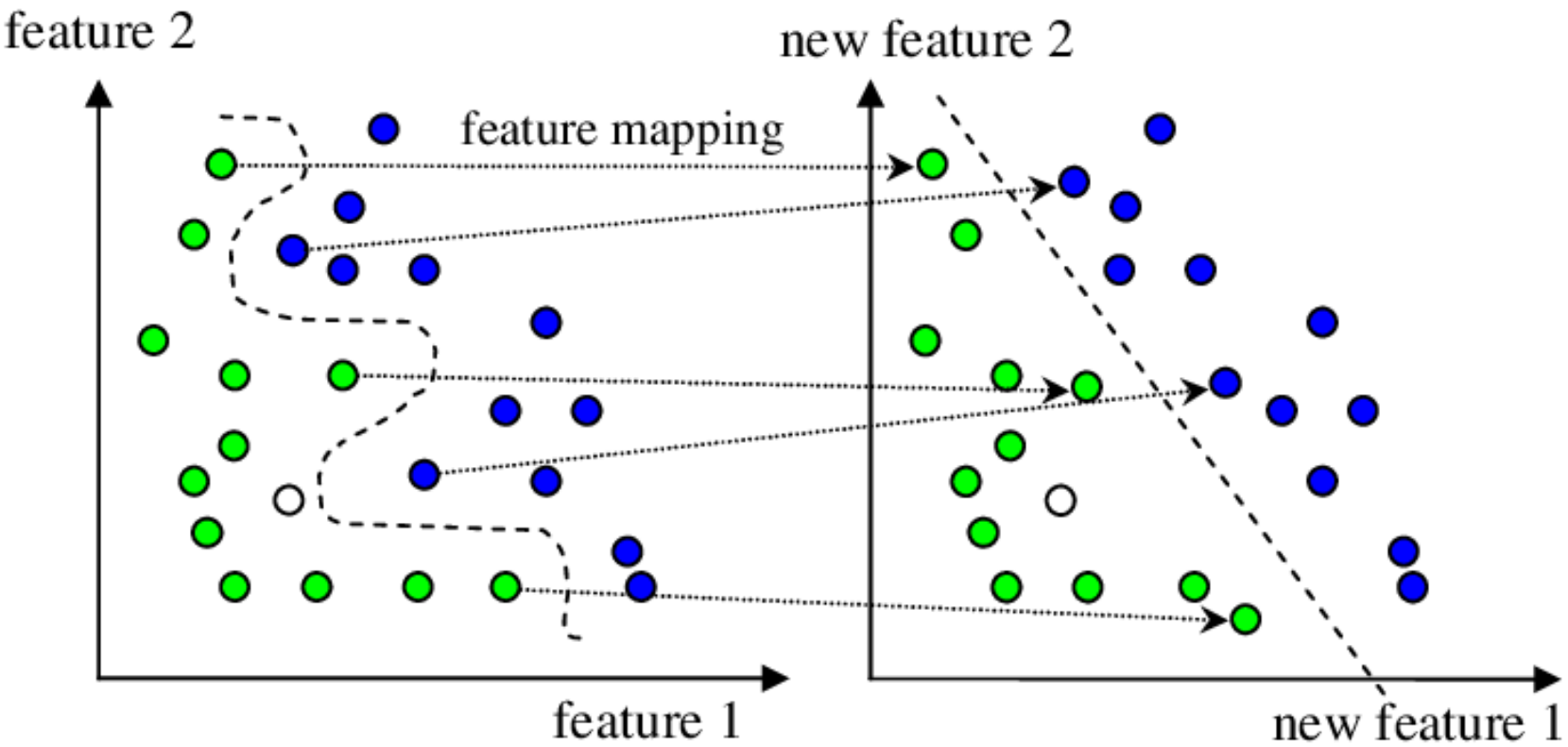
Kernel Machines: The Big Picture

- Margin maximization revisited:



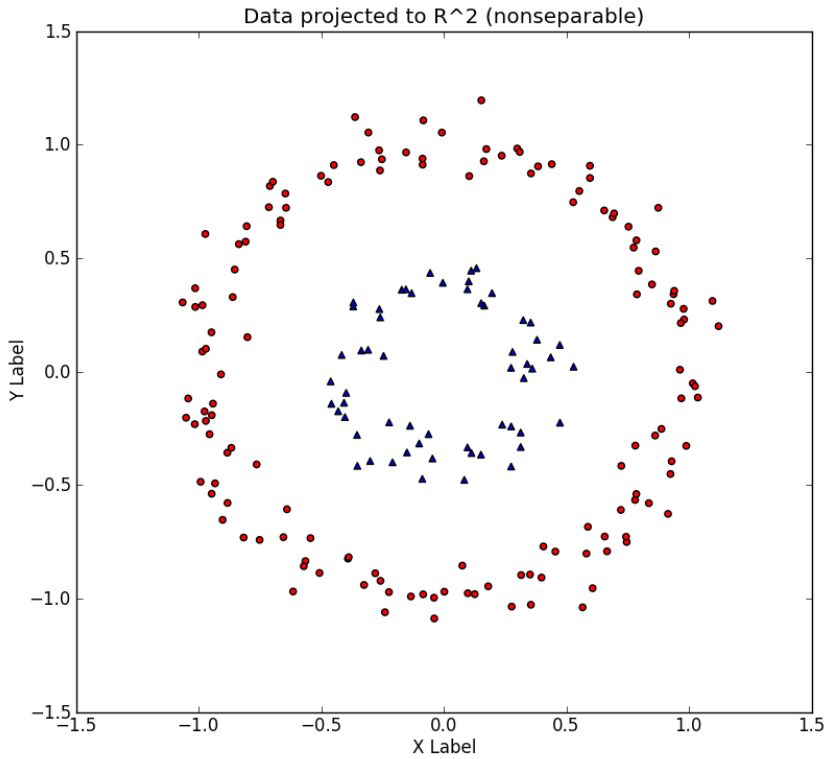
Kernel Machines: A Smaller Picture

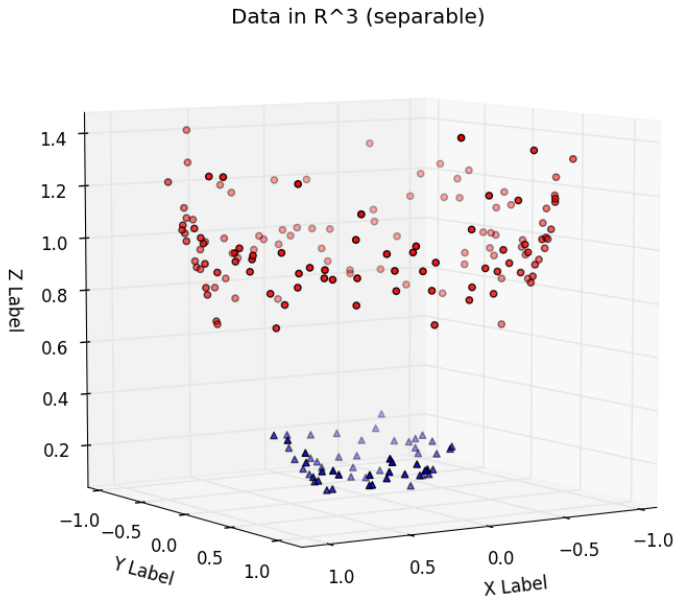
- Duality is key:



Kernel Machines: Explicit feature embeddings (revisited)

- Can think of it as an explicit feature embedding:





Kernel Machines: Discussion

- Using the dual formulation also reformulates the optimization problem in terms of a matrix of inner products between training samples.
- This is called the Kernel Matrix (or Gram matrix) – hence the name.
- The resulting classifiers are implicitly operating in another feature space – one that is of higher, or lower, or even with infinite dimensions.
- Disadvantages: the optimization problem is MUCH more computationally intensive, and introduces more hyperparameters (which depend on the kernel you use).
- In sklearn: `sklearn.svm.SVC`

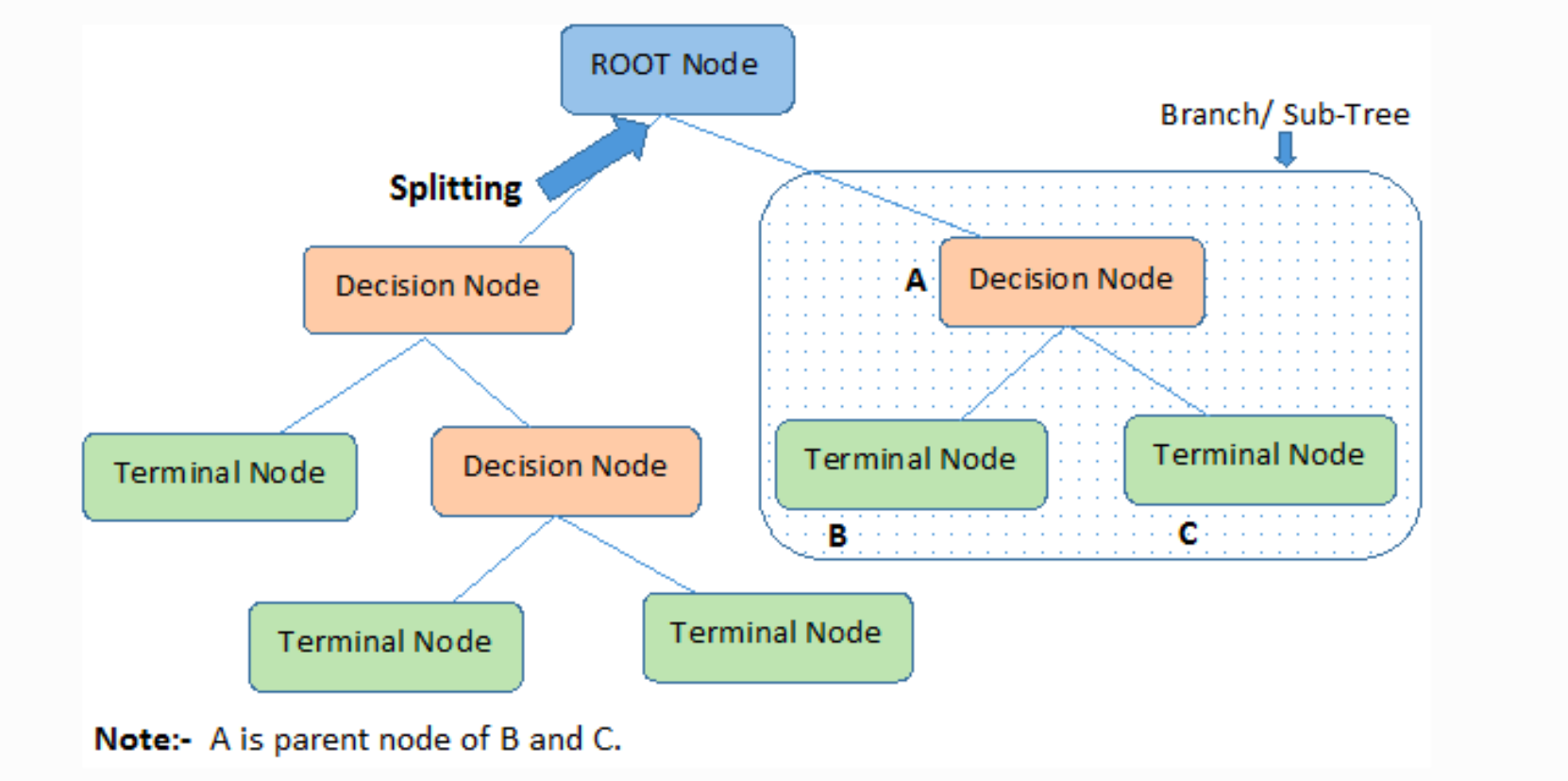
Decision Trees: Overview

- Decision trees are models that partition feature space – much like KNN.
- They are recursive models that learn how to best partition space using training data.
- They are also classical models that have been used for decades in statistics, medicine, biology – you name it.
- They are often preferred in practice because they mirror how humans think about decision making.

Decision Trees: The Algorithm (ID3)

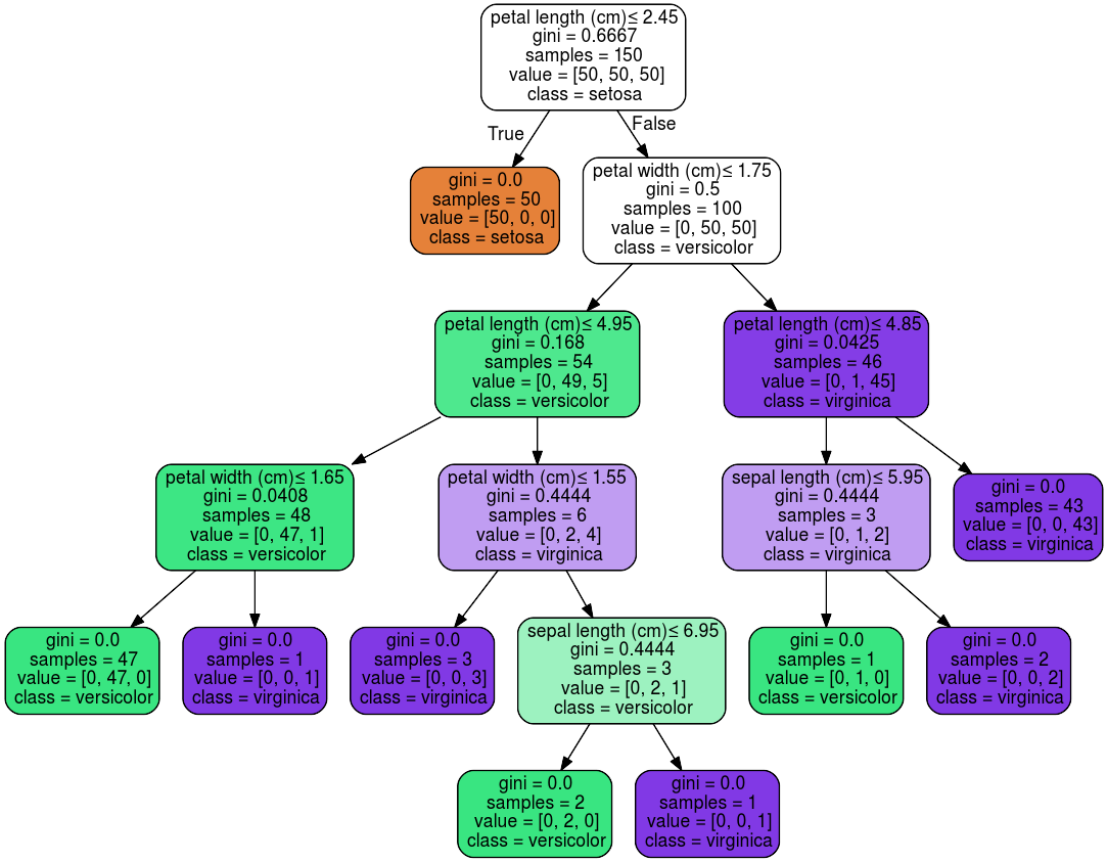
- The algorithm:
  1. Begins with the original set D as the root node.
  2. On each iteration of the algorithm, it iterates through the unused features of the set D and calculates the entropy H of this feature.
  3. It then selects the attribute which has the smallest entropy
  4. The set D is then split using the selected attribute to produce two subsets of the data.
  5. The algorithm then recursively splits each subset, considering only attributes never selected before.
  6. It terminates at nodes with pure (or nearly pure) subsets containing only one class.

Decision Trees: General Idea



Decision Trees: A Parametric View

- Decision trees can also use learned thresholds to split sets:



Decision Trees: Discussion

- Decision trees are nice and – most importantly – readily explainable models.
- There is a huge variety of algorithms: non-parametric, parametric, probabilistic, etc.
- They can also be used for regression.
- The main disadvantage is that they can very easily overfit the available training data (and this not generalize).
- In sklearn: `sklearn.tree.DecisionTreeClassifier`

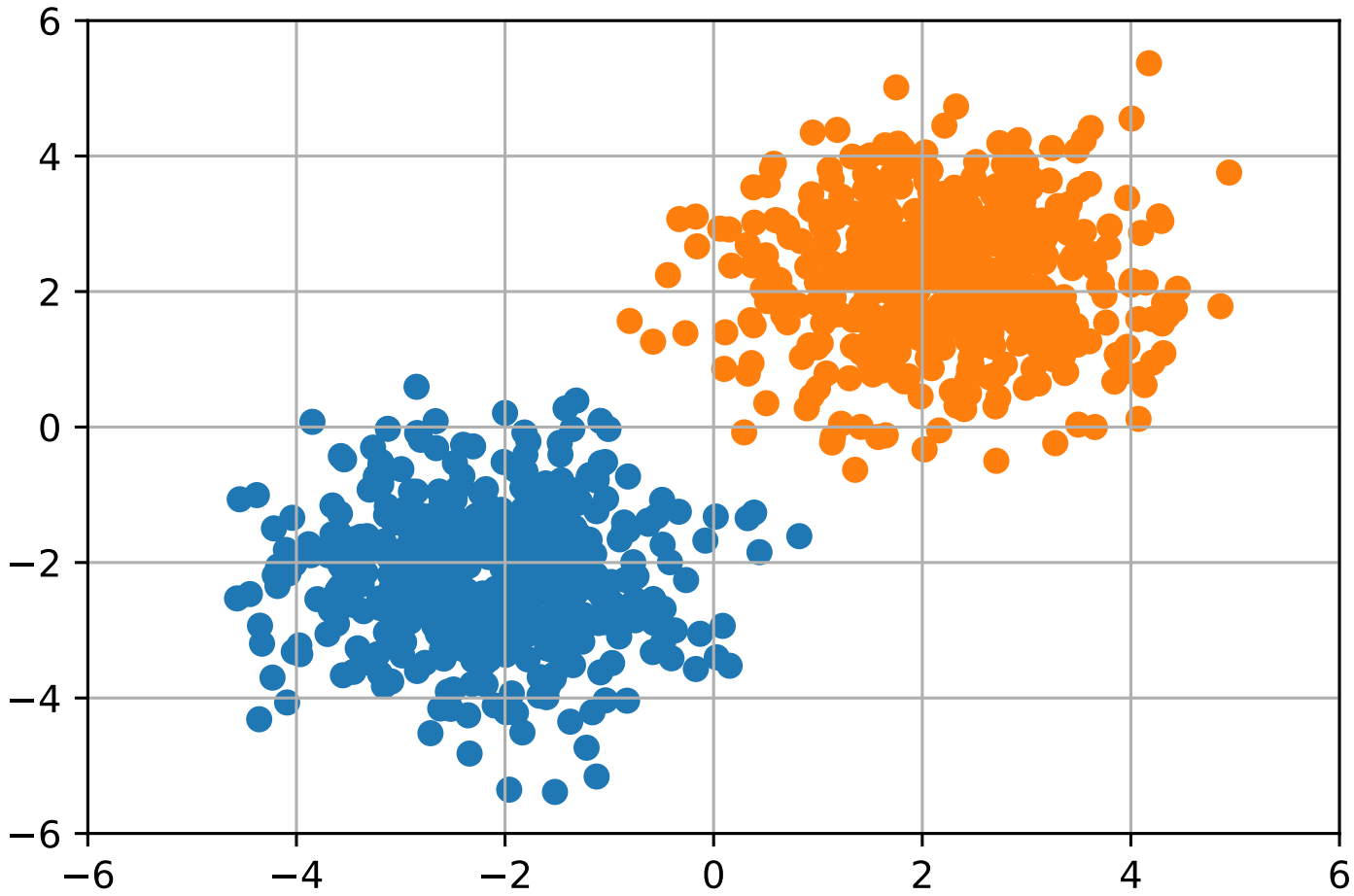
Unsupervised Learning

Why unsupervised learning?

- Unsupervised learning is learning something from data in the absence of labels or target values.
- It is an active area of current research.
- Note that it is often used even in the context of supervised learning problems:
  - For dimensionality reduction: often not all of the input features are needed (or even desirable), and unsupervised techniques can be to reduce the input dimensionality of a problem.
  - For visualization: to get a sense of the data, we may want to visualize it in some meaningful way – this often uses dimensionality reduction to reduce high dimensional features to just two or three.
- In this part of the lecture we will see three useful unsupervised techniques: Principal Component Analysis (PCA), Clustering, and t-Distributed Stochastic Neighbor Embedding (t-SNE).

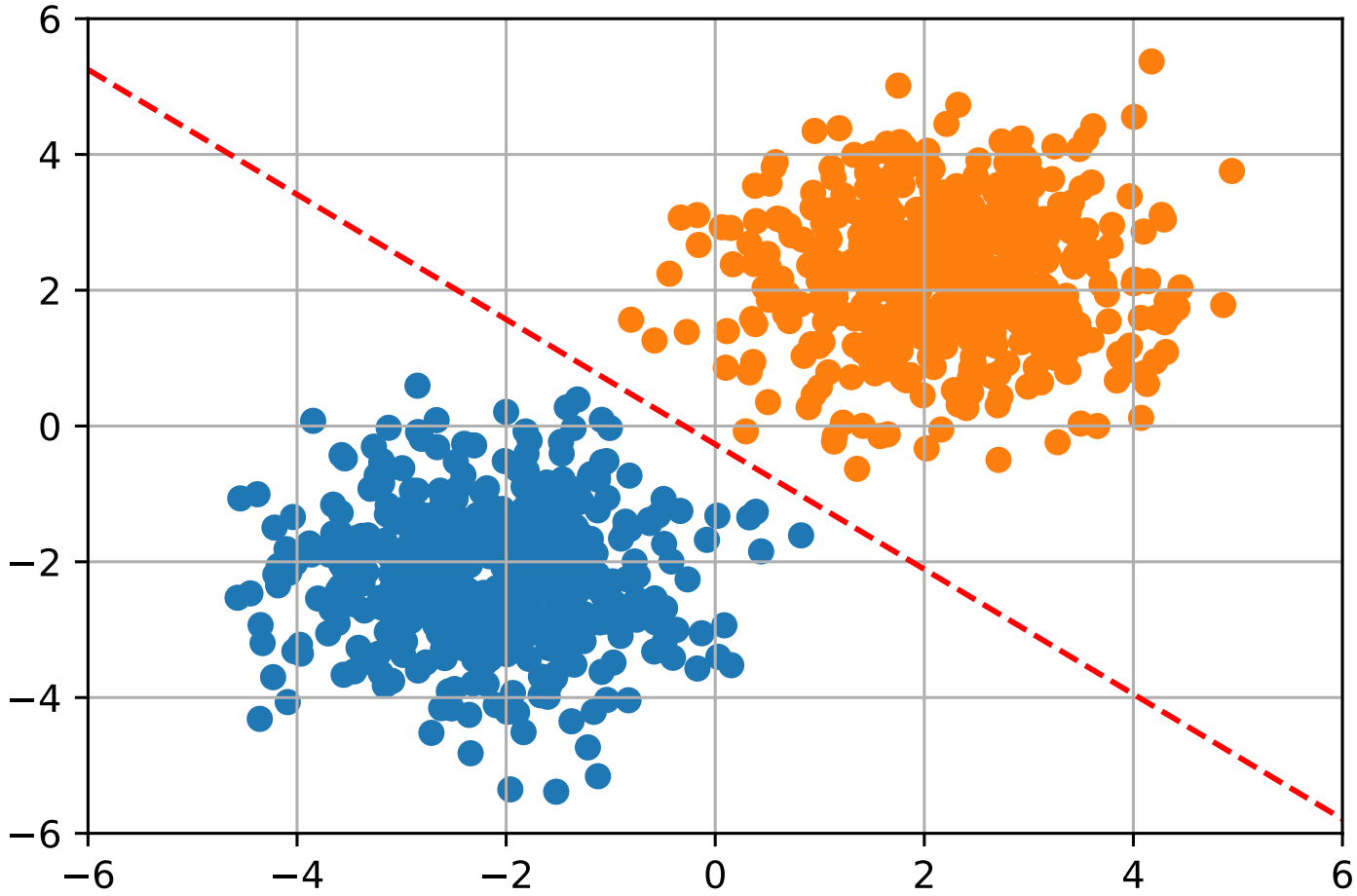
PCA: Motivation

- Say we have a classification problem with data distributed like this:



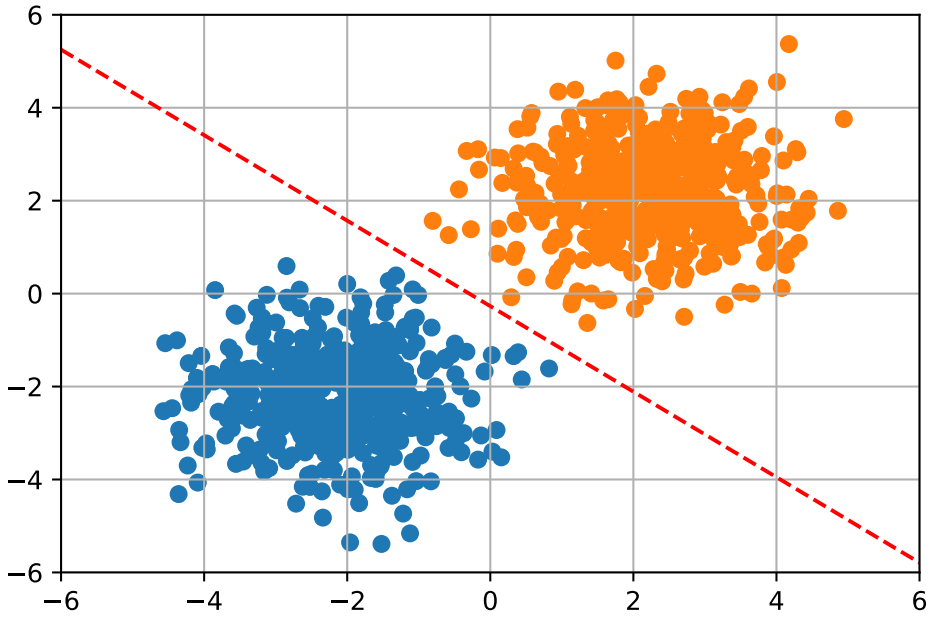
PCA: Motivation (continued)

- We know how to train a classifier for linearly separable classes:



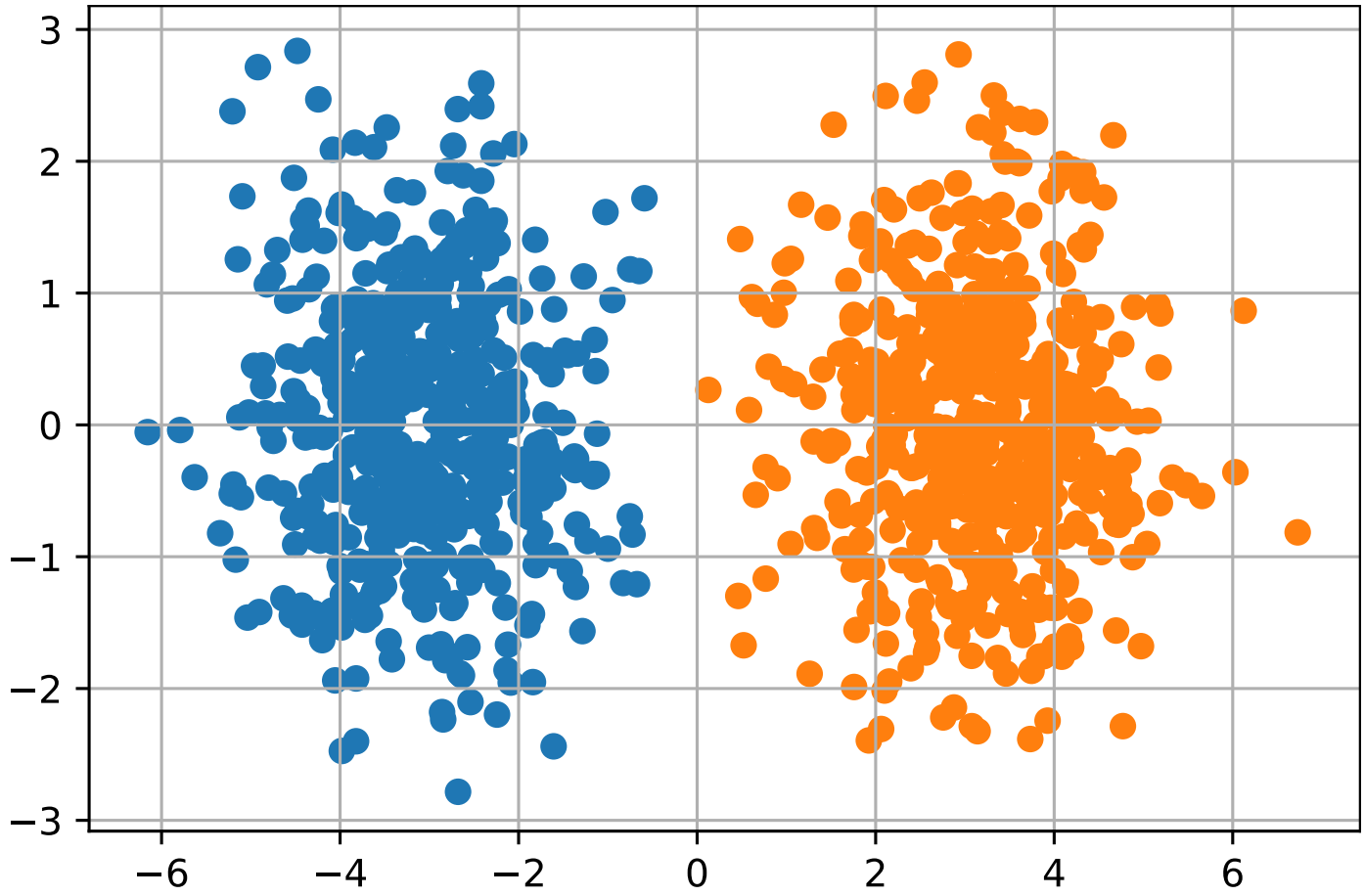
PCA: Motivation (continued)

- But, let's take a closer look at this situation.
- In the original feature space we need both features to define the discriminant dividing the two classes.
- But, maybe if we could somehow transform this space so that the features are decorrelated...



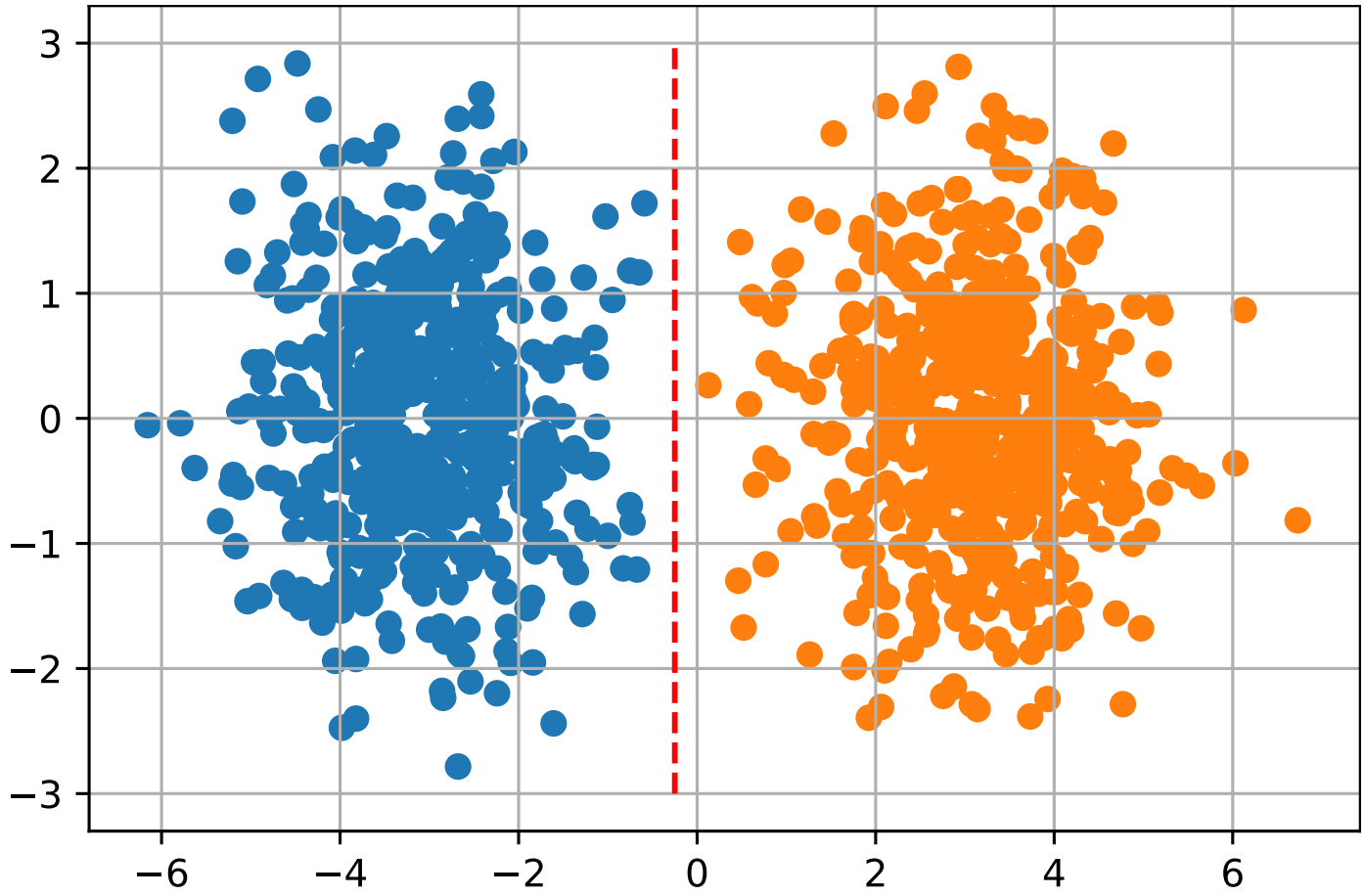
PCA: Motivation (continued)

- What if we rotate the feature space so that the principal data directions are aligned with the axes?



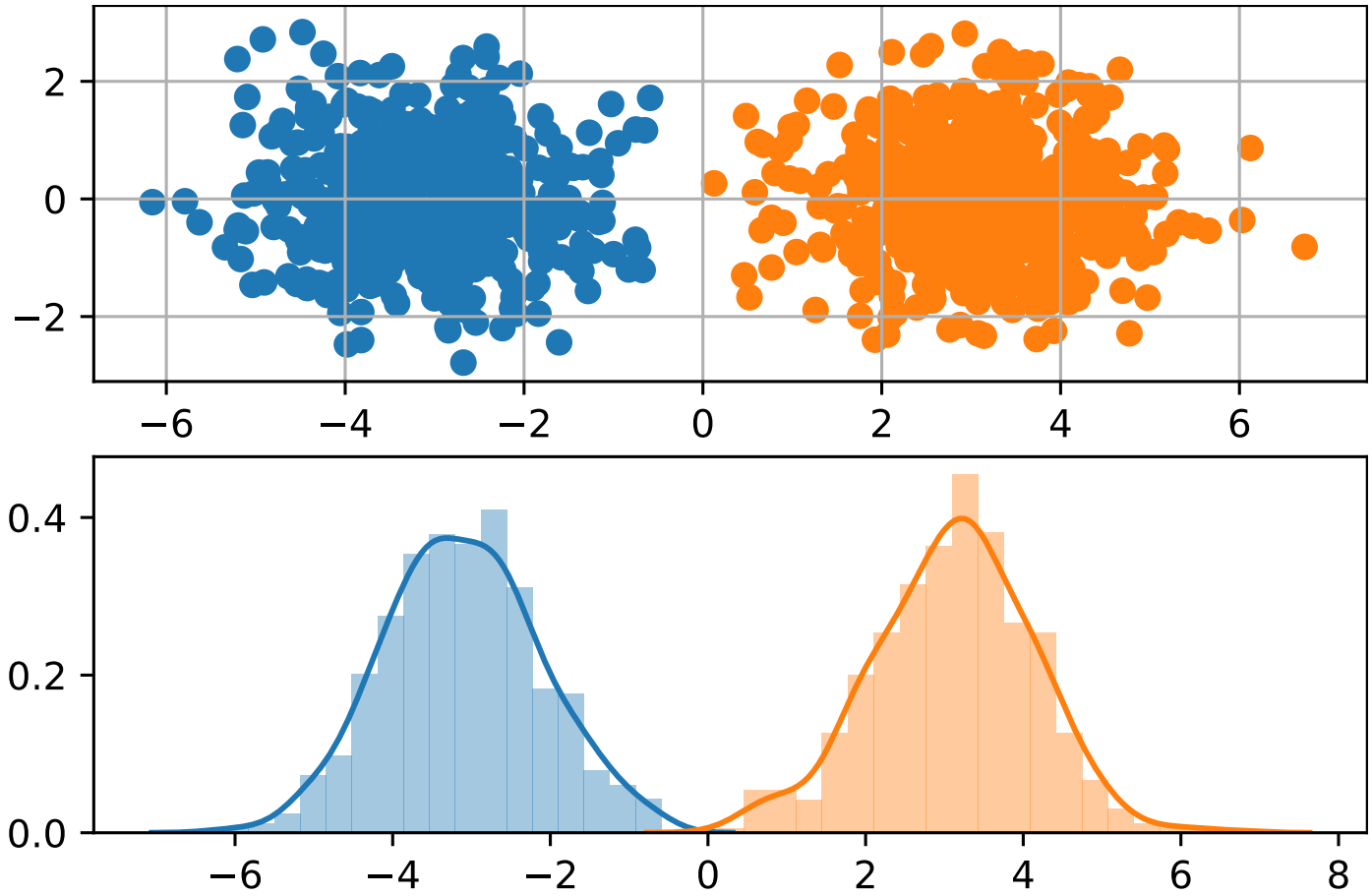
PCA: Motivation (continued)

- Well now we have an easier problem to solve, since the discriminant function needs only one feature:



PCA: Motivation (continued)

- We have turned a two-dimensional problem into a one-dimensional proble.



PCA: The Math

- How do we find these principal directions of the data in feature space?
- How do we even define what a principal direction is?
- Well, one natural way to define it is iteratively:
  1. The first principal component is the *direction in space along which projections have the largest variance*.
  2. The second principal component is the *direction which maximizes variance among all directions orthogonal to the first*.
  3. etc.
- This defines up to D principal components, where D is the original feature dimensionality.
- If we project onto all principal components we are rotating the original features so that in the new axes the features are decorrelated.

PCA: The Math (continued)

- If we project onto the first $d < D$ principal components, we are performing dimensionality reduction.
- How do we do any of this? We use the eigenvectors of the data covariance matrix $\mathbf{\Sigma}$.
- Recall the multivariate Gaussian:

$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}))$$

- $\mathbf{\Sigma}$ this defines the hyperellipsoidal shape of the Gaussian.
- If Σ is diagonal, the features are decorrelated, so if we diagonalize it we are decorrelating the features.
- We do this by finding the eigenvectors of (an estimate of) Σ – the data covariance matrix.

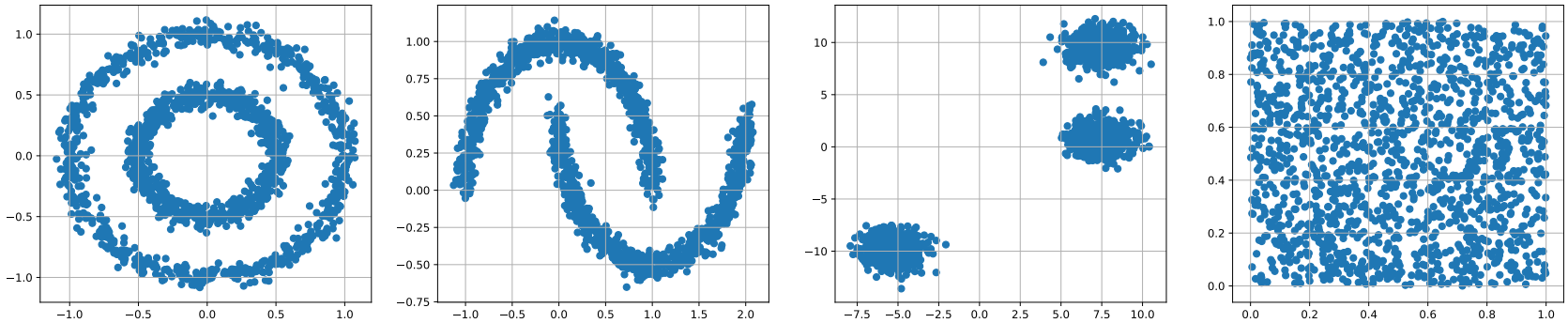
PCA: Analysis

- Principal Component Analysis is used for many purposes:
  - Dimensionality reduction: in high-dimensional features spaces with limited data, PCA can help reduce the problem to a simpler one in fewer, decorrelated dimensions.
  - Feature decorrelation: some models assume feature decorrelation (or are simpler to solve with decorrelated features).
  - Visualization: looking at data projected down to the first two principal dimensions can tell you a lot about the problem.
- In sklearn: sklearn.decomposition.PCA

```
from sklearn.decomposition import PCA
model = PCA()
model.fit(xs)
new_xs = model.transform(xs)
```

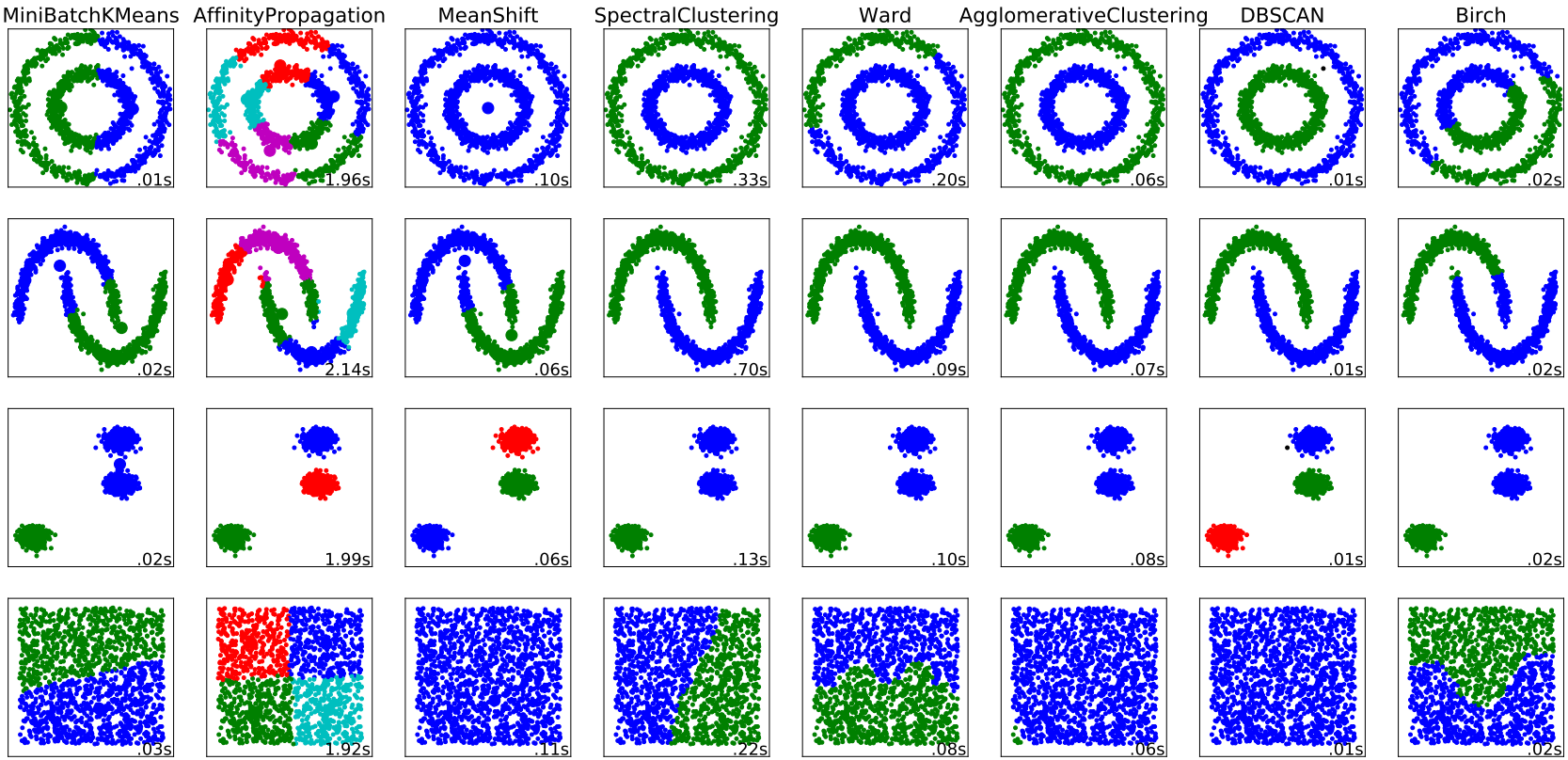
Clustering: Motivation (continued)

- What if you have data with structure, but you don't know what this structure is or have any a priori model for it?



Clustering: Motivation (continued)

- Clustering refers to techniques that learn groups of related data points in feature space:



Clustering: The k-Means Algorithm

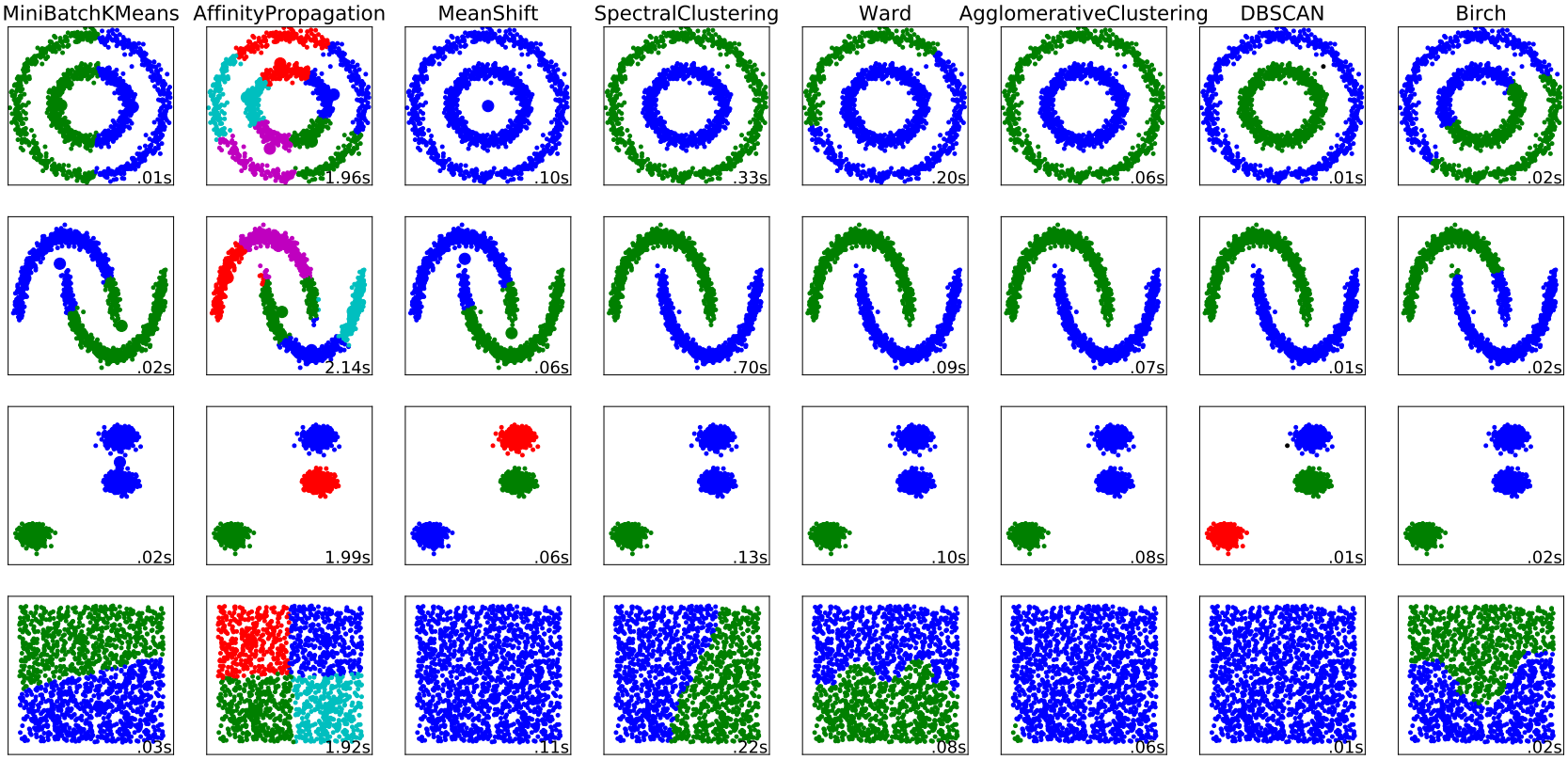
- k-Means is a very simple algorithm that associates each data point with one of k means or centroids in the data space:
- Given an initial set of k means $\mathbf{m}_1^1, \ldots, \mathbf{m}_k^1$, the alternates between two steps:
  1. Assignment: $S_i^t = \{ \mathbf{x}_p \mid ||\mathbf{x}_p - \mathbf{m}_i^t|| \leq ||\mathbf{x}_p - \mathbf{m}_j^t|| \forall j \}$.
  2. Update: $\mathbf{m}_i^{t+1} = \frac{1}{|S_i^t|} \sum_{\mathrm{x}_i \in S_i^t} \mathbf{x}_j$.
- The algorithm converges when the cluster assignments no longer change.
- Note: this algorithm is not guaranteed to find the global optimum clusters.

Clustering: Analysis

- Clustering is a robust technique (in that it never fails).
- Its usefulness depends on the data distribution and which type of clustering you perform.
- In sklearn:
  - `cluster.AffinityPropagation`
  - `cluster.AgglomerativeClustering`
  - `cluster.Birch`
  - `cluster.DBSCAN`
  - `cluster.KMeans`
  - `cluster.MeanShift`
  - `cluster.SpectralClustering`
  - See `sklearn.cluster` for more details.

Clustering: Motivation (continued)

- It's useful to revisit this plot:



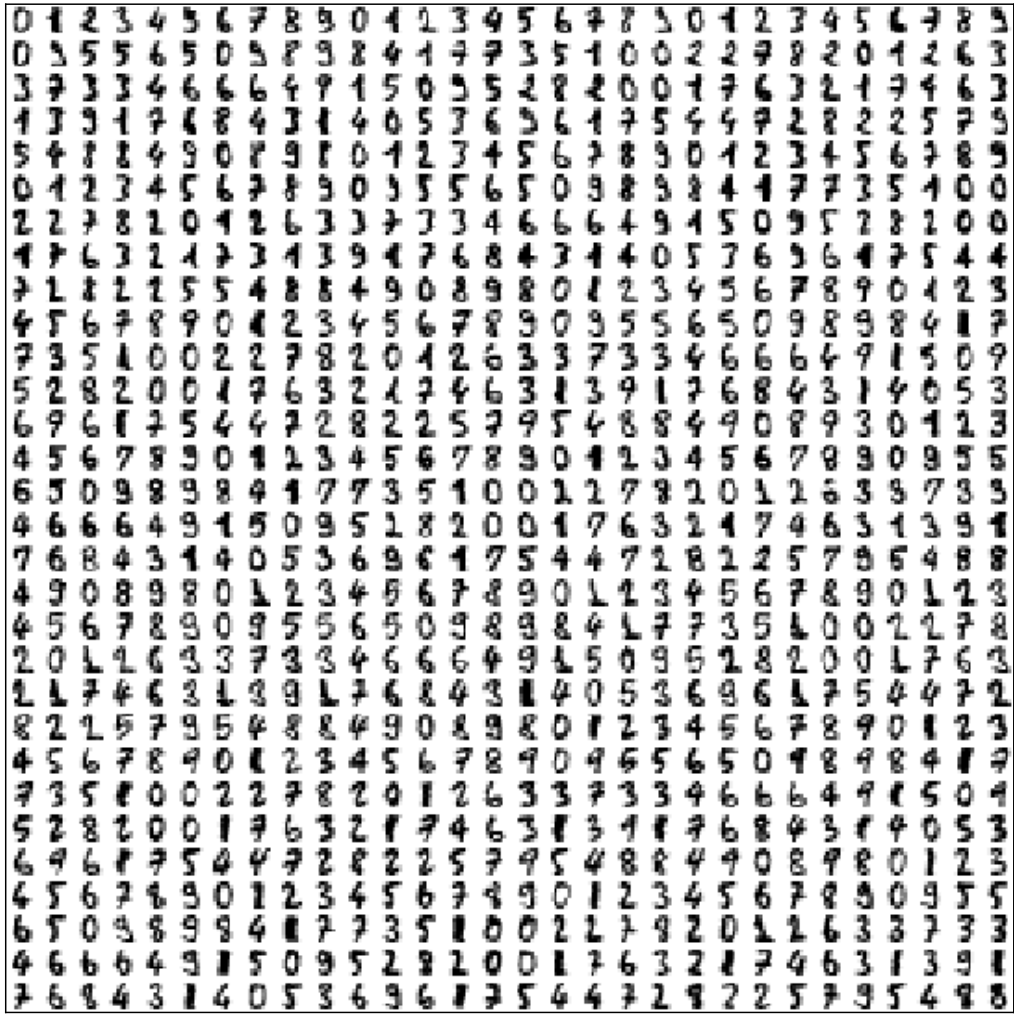
t-SNE: Motivation

- Sometimes the structure of data in high dimensional data is obscured by its very high-dimensional nature.
- Questions like this are related to data design: you often need to discover if the features you have collected are at all related to the questions you want to ask about it.
- In this last section we will look at a very powerful algorithm for uncovering latent structure in high-dimensional data.
- This approach is called the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorith,
- It works by reducing dimensionality (usually to just two dimensions) in a way that preserves the distances between samples.

t-SNE: Motivation (continued)

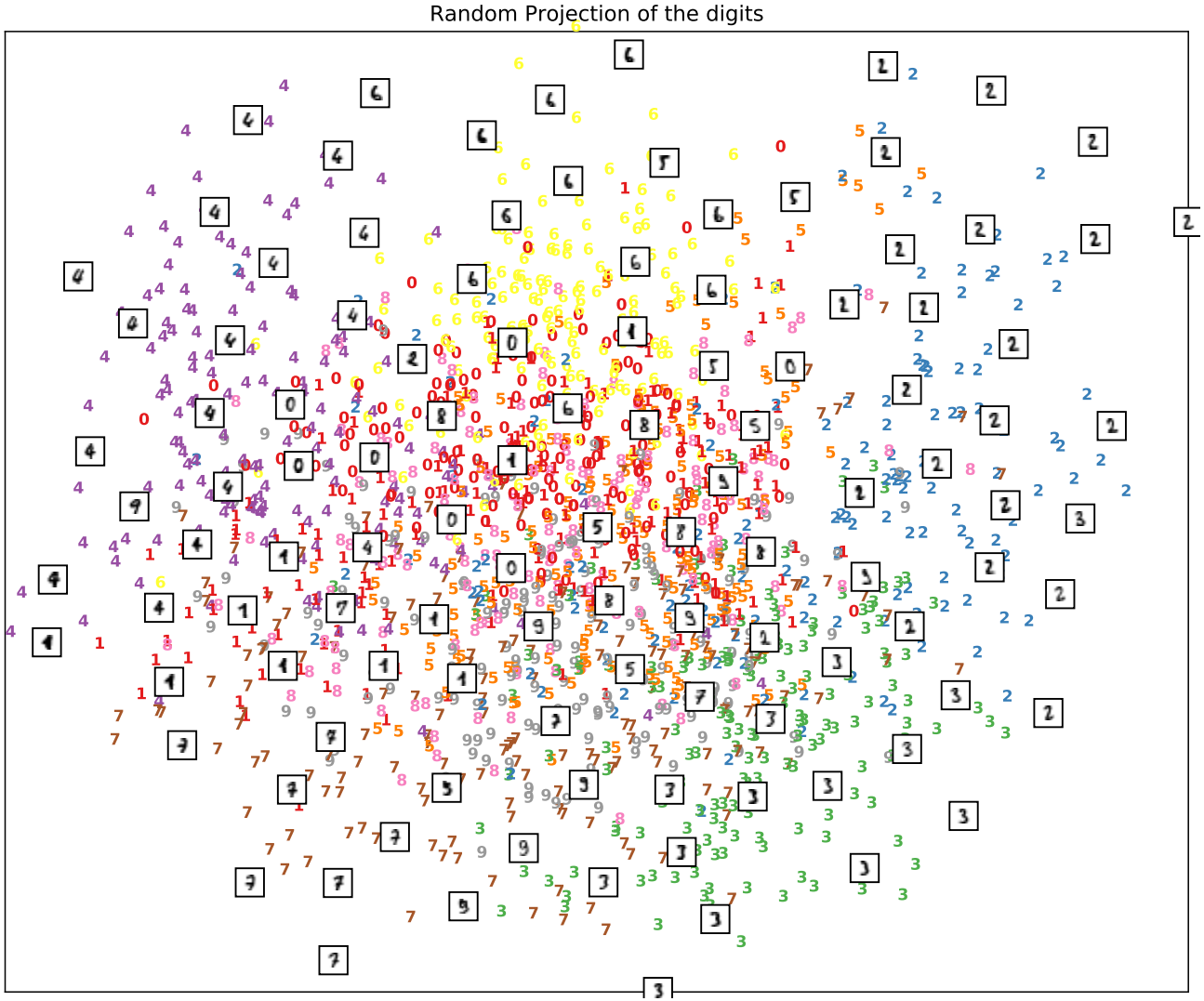
- Even in just 64 dimensions data can be complex:

A selection from the 64-dimensional digits dataset



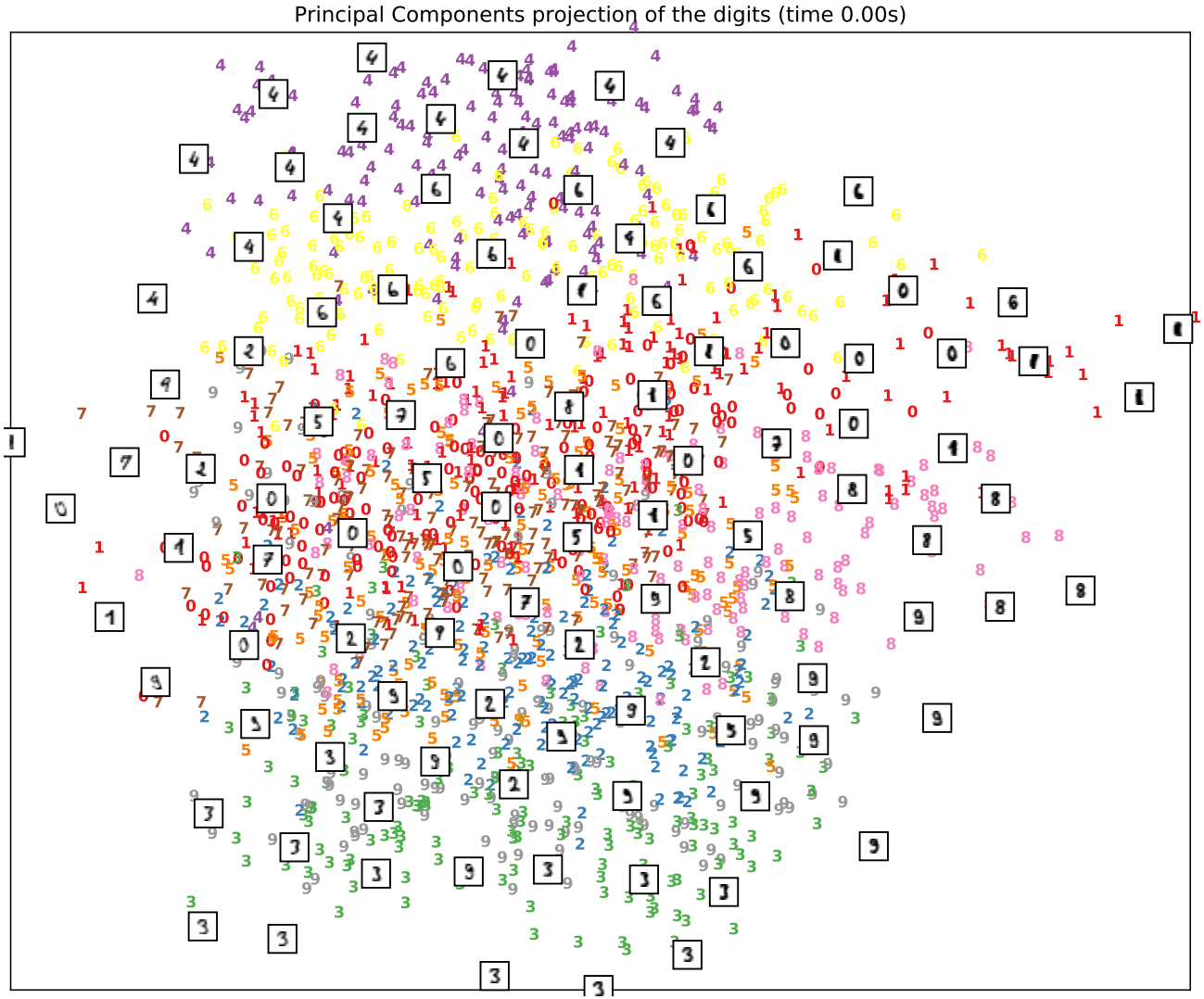
t-SNE: Motivation (continued)

- We can try random projection to two dimensions:



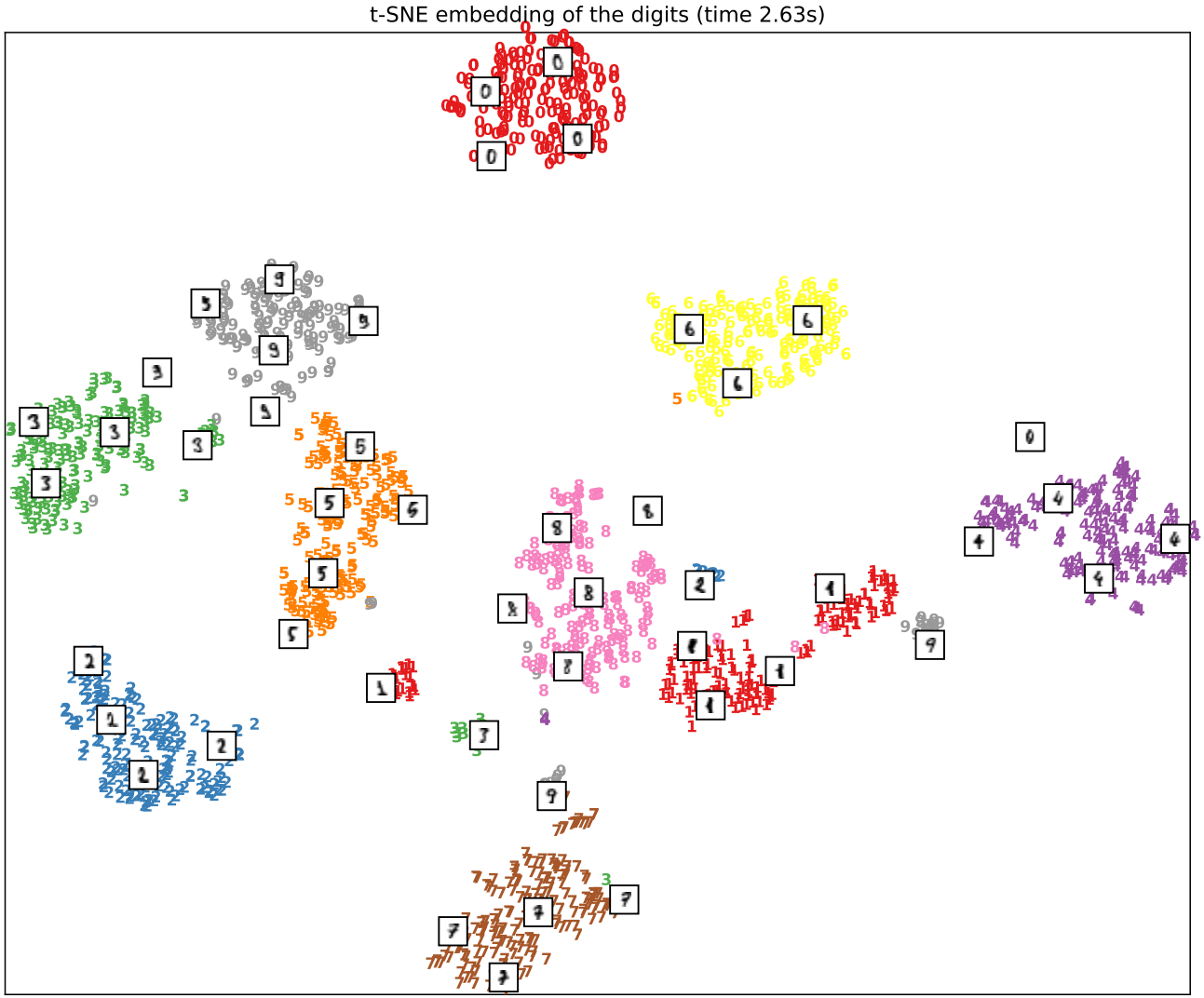
t-SNE: Motivation (continued)

- Or projection onto first two principal components:



t-SNE: Motivation (continued)

- What we want is something like:



t-SNE: The Math

- t-distributed Stochastic Neighbor Embedding (t-SNE) is a algorithm for visualization.
- It is a nonlinear dimensionality reduction for embedding high-dimensional data in a low-dimensional space of two or three dimensions.
- It models each high-dimensional point by a low-dimensional point so that that similar objects map to nearby points and dissimilar objects are to distant points with high probability.
- The short version: points that are close in the high-dimensional space are close in the mapped space.

T-SNE: Analysis

- t-SNE is a very powerful tool for visualization and dimensionality reduction.
- It can give you visual feedback that indicates if, on average, the high-dimensional features represent your problem well (or not).
- Note that t-SNE visualization can sometimes be misleading in that the associations it learns are not always present in the original space.
- In sklearn: `sklearn.manifold.TSNE`

Model Selection

The Math

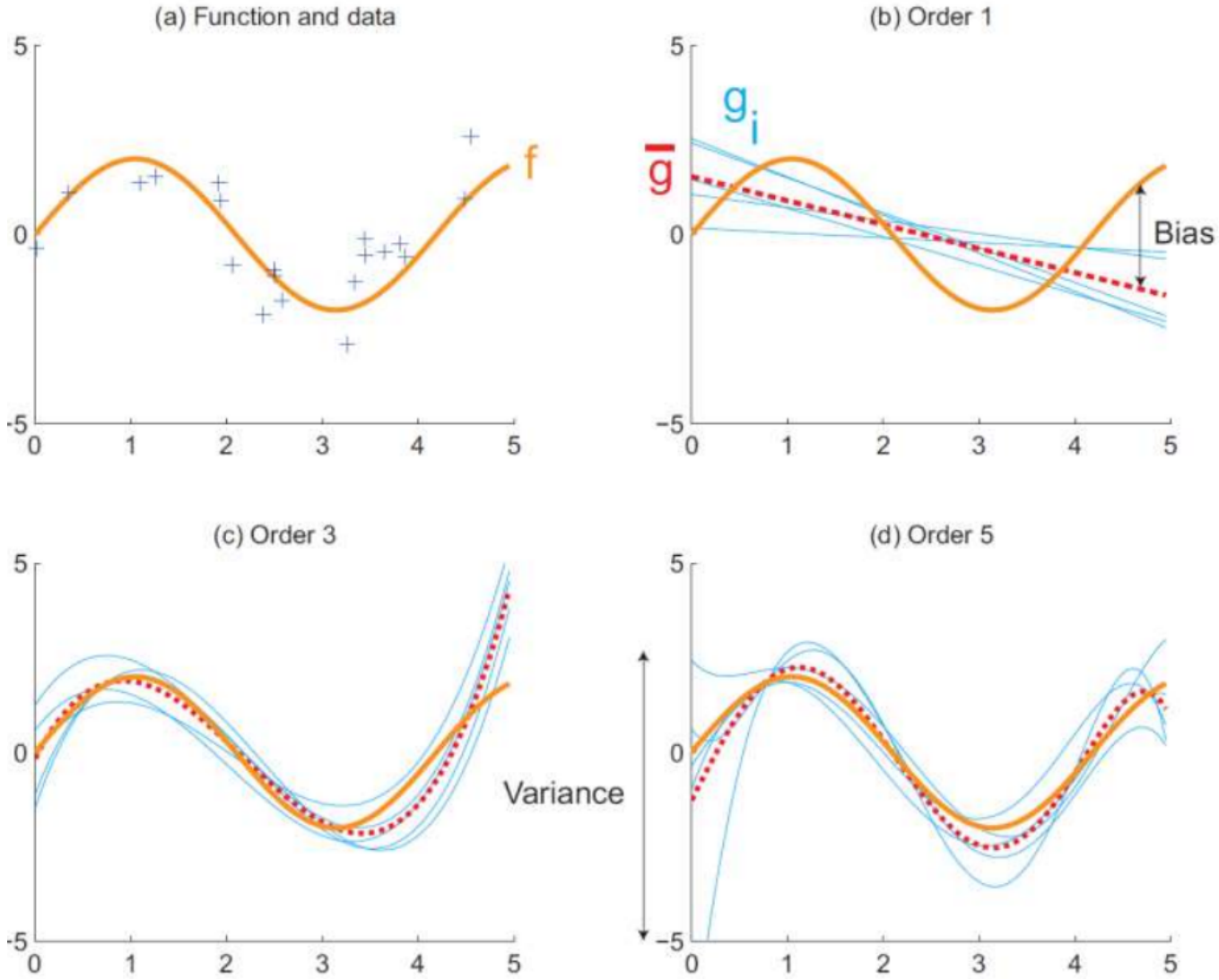
- The biasvariance decomposition is a way of analyzing a model's expected generalization error: the bias, the variance, and the irreducible error resulting from noise in the problem itself.
- Say we estimate a true function $f(x)$ by $y = \hat{f}(x) + \varepsilon$, where ε is the noise with zero-mean and variance σ^2.
- It can be shown that the expected error is equal to:

$$\begin{aligned} E[f(x) - \hat{f}(x) + \varepsilon] &= (\textbf{Bias}[\hat{f}(x)])^2 + \textbf{Var}[\hat{f}(x)] + \sigma^2, \text{ where} \\ \textbf{Bias}[\hat{f}(x)]) &= E[\hat{f}(x)] - E[f(x] \\ \textbf{Var}[\hat{f}(x)] &= E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 \end{aligned}$$

The Main Point

- As we increase model complexity:
  - Bias decreases: a better fit to data.
  - Variance increases: fit model varies more with data.
- Imagine the hierachy of polynomial models:
  1. $f(x) = c$
  2. $f(x) = ax + c$
  3. $f(x) = ax^2 + bx + c$
  4. ...
- As we go up in this hierarchy, model complexity increases and bias decreases.
- But, the model parameters estimated from data will wildly fluctuate with changing data – *even if drawn from the same distribution*.

Visualizing Bias and Variance

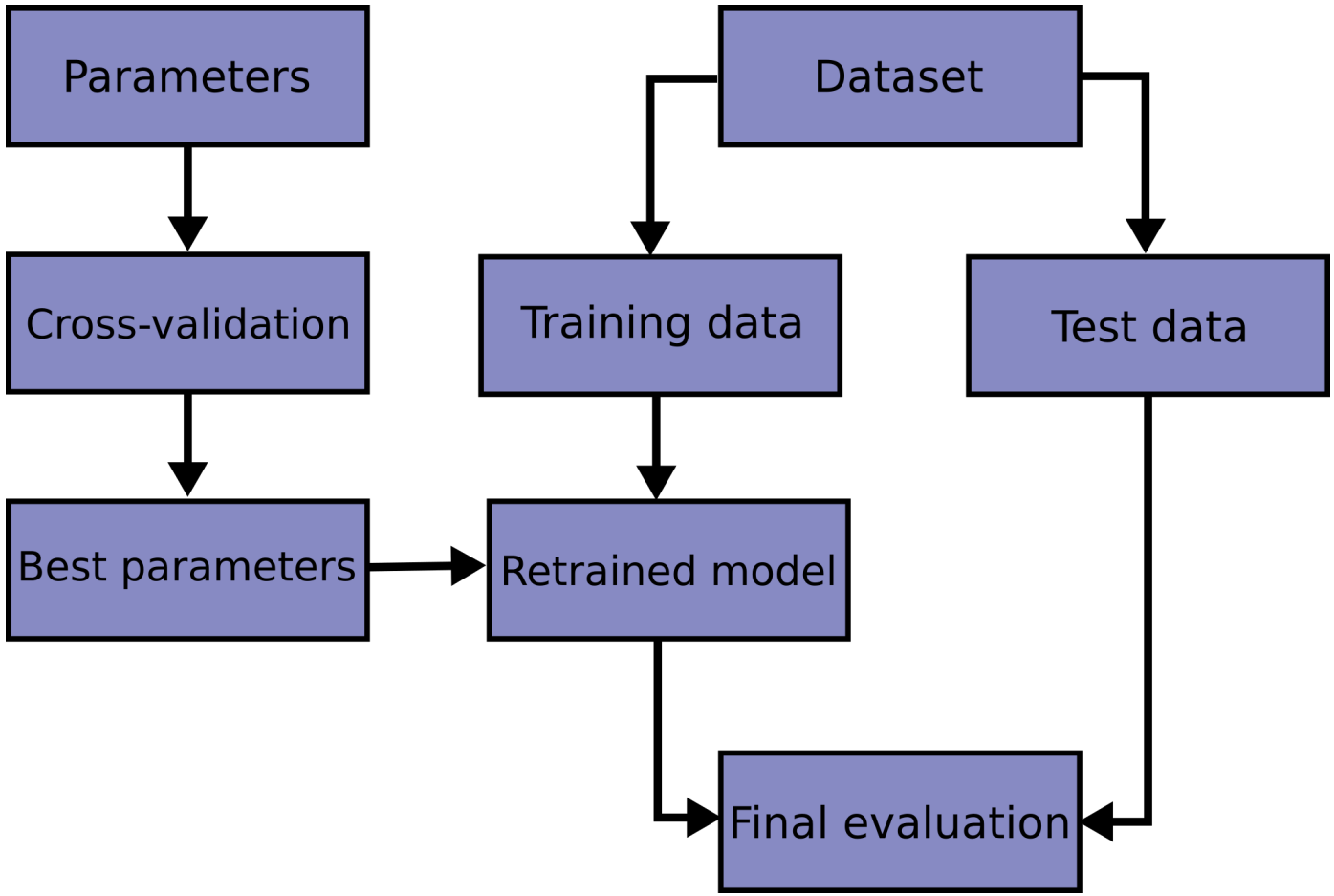


Cross-validation Overview

- Learning the parameters of any model and testing it on the same data is a methodological mistake.
- A model that just memorizes the labels of the training samples would have a perfect score.
- But, of course it would fail miserably to classify any samples not yet seen.
- This is an extreme example of what is called overfitting.
- To avoid it, it is common practice when performing supervised machine learning to hold out part of the available data as a test set (as we have done since the beginning).

A Useful Flowchart

- Here is a flowchart of the cross-validation workflow for training:



Validation Set

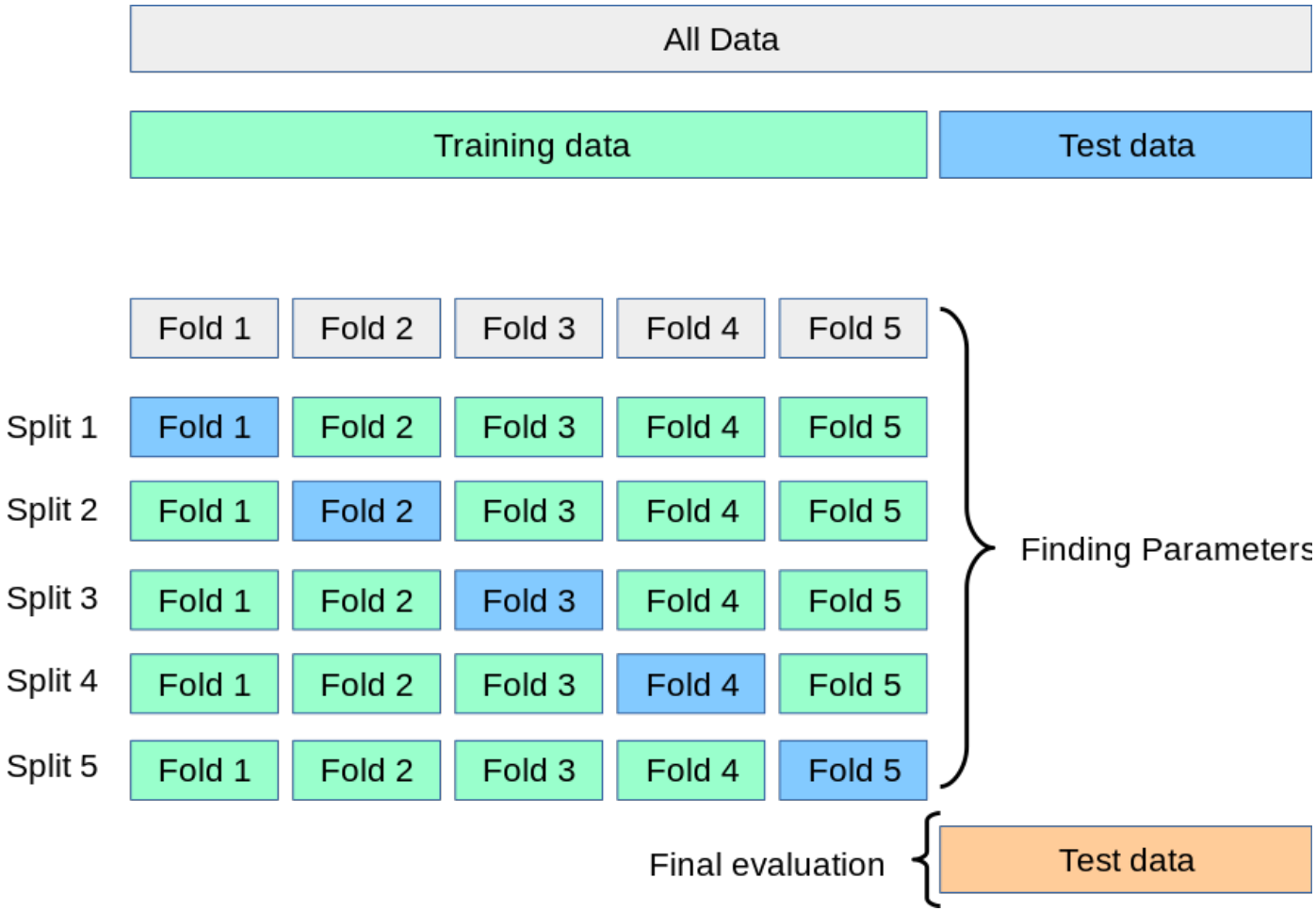
- When evaluating different hyperparameter settings, there is still a risk of overfitting on the test set.
- If we tweak parameters until estimator is optimal, knowledge about the test set can "leak" into the model and evaluation metrics no longer reflect generalization performance.
- To solve this problem, usually another part of the dataset can be held out as a validation set: we train on training set, then evaluate on the validation set, and when the model seems to work well the *final evaluation is done on the test set*.
- However, by partitioning the available data into three sets, we drastically reduce the number of samples used for learning.
- Moreover, the results can depend on a particular random choice for train and validation sets.

Enter, Cross-validation

- A solution to this problem is a procedure called cross-validation.
- A test set should still be held out for final evaluation, but the validation set is no longer needed.
- The basic approach is called k-fold cross-validation: the training set is split into k equally-sized, smaller sets, and the following procedure is followed for each of the k folds:
  1. A model is trained using $k - 1$ of the folds as training data;
  2. The resulting model is validated on the remaining part of the data by computing a performance measure such as accuracy on it.
- Important: the average performance over the k folds gives us a lower bound on the generalization of the model to unseen data.

Cross-validation (continued)

- Here is a diagram explaining the k-fold process:



Cross-validation (continued)

- In sklearn we can easily do k-fold cross-validation using the `sklearn.model_selection.cross_val_score` function:

```
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC
model = LinearSVC(C=100, verbose=3)
scores = cross_val_score(model, X_tr, y_tr, cv=3,
                         verbose=3, n_jobs=4)
```

- Some parameters to pay attention to:
  - `cv`: number of folds to use.
  - `verbose`: logging level – useful to have feedback for long runs.
  - `n_jobs`: number of parallel jobs to use.
  - `scoring`: function to use for scoring (defaults to `model.score()`.

Cross-validation: Analysis

- Cross-validation is a powerful tool for understanding how models (might) generalize.
- As we will see next, it is the basis for hyperparameter evaluation and selection.
- Problem: cross-validation is expensive as multiple models must be fit to multiple splits of data.

Hyperparameter Selection

- Up to now we have used cross-validation only to obtain a more reliable estimate of the performance of our estimator.
- By training multiple times on random train/validation splits we make the most of available data.
- But this still leaves open the question of how to effectively select the hyperparameters of our model.
- Up to now we have used models that have relatively few hyperparameters.
- When we look at deep models based on neural networks, however, there will be significantly more.
- Fortunately, cross-validation also gives us a tool for robustly estimating performance over a grid of hyperparameters.

Hyperparameter Selection: Validation Curves

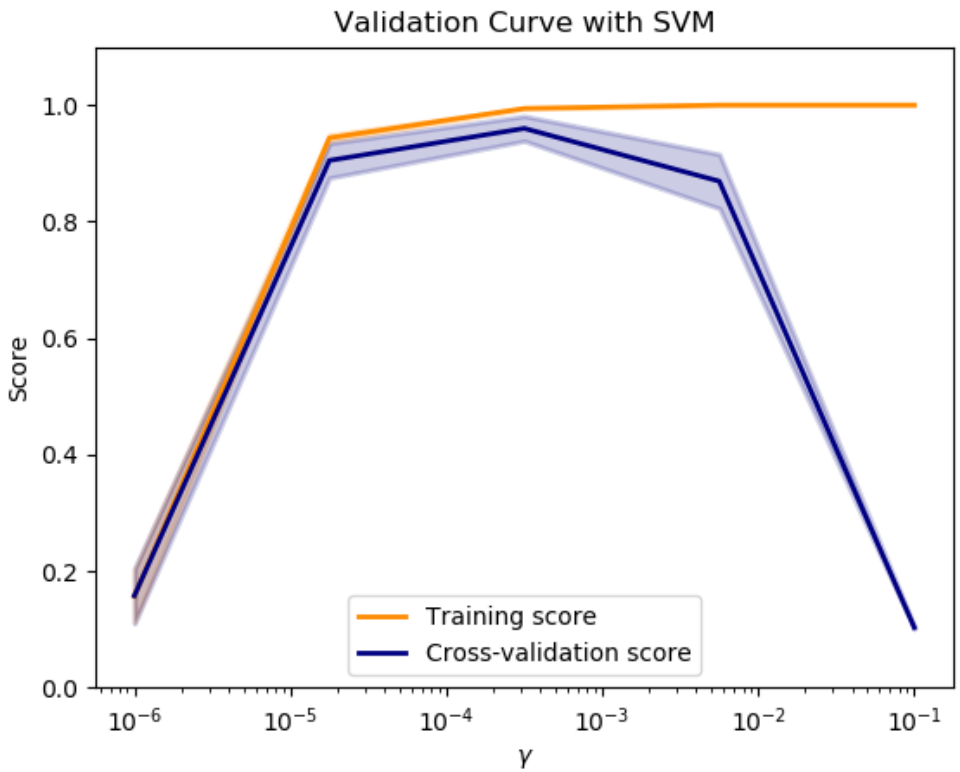
- An excellent way to get an overview of the sensitivity of a model to one hyperparameter is to plot a validation curve.

```
from sklearn.model_selection import validation_curve
(train_scores, val_scores) = validation_curve(
                                LinearSVC(), X_tr, y_tr,
                                "C", [0.1, 1.0, 10, 100, 1000],
                                cv=3)
val_scores

array([[0.85090745, 0.85135743, 0.82478248],
       [0.85180741, 0.84535773, 0.85238524],
       [0.84910754, 0.85300735, 0.83408341],
       [0.83620819, 0.84640768, 0.85358536],
       [0.85525724, 0.86170691, 0.84983498]])
```

Hyperparameter Selection: Validation Curves

- Useful: `validation_curve` returns the cross-validated scores for all folds for all parameters.
- This allows us to make useful plots:



See: *https://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html*

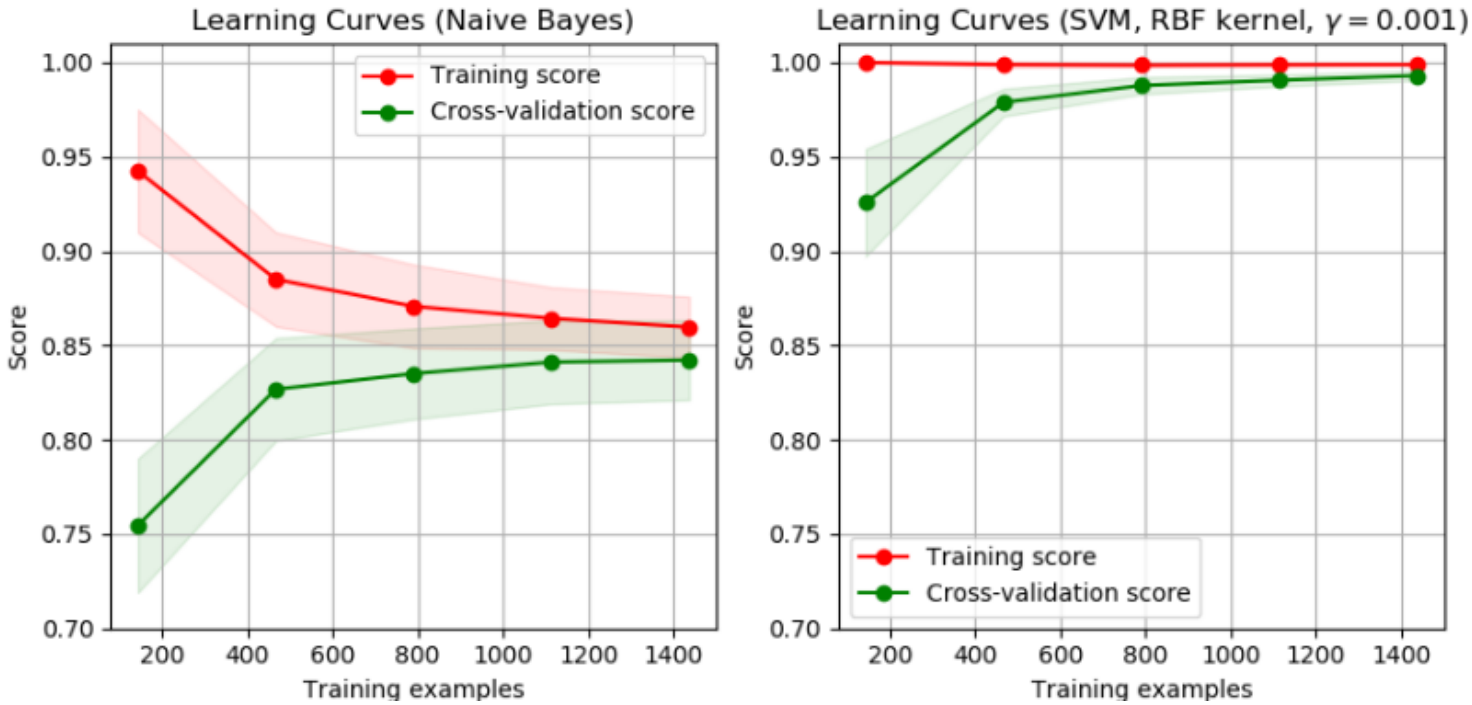
Hyperparameter Selection: Learning Curves

- An important factor in the variance of any model is the size of the training split.
- According to Geoffrey Hinton: "More labeled data is the best possible model regularizer. . . "
- Using `sklearn.model_selection.learning_curve()` we can evaluate model performance as a function of test split size:

```
from sklearn.model_selection import learning_curve
train_sizes, train_scores, test_scores = learning_curve(
                                            LinearSVC(),
                                            X_tr, y_tr, cv=3)
```

Hyperparameter Selection: Learning Curves

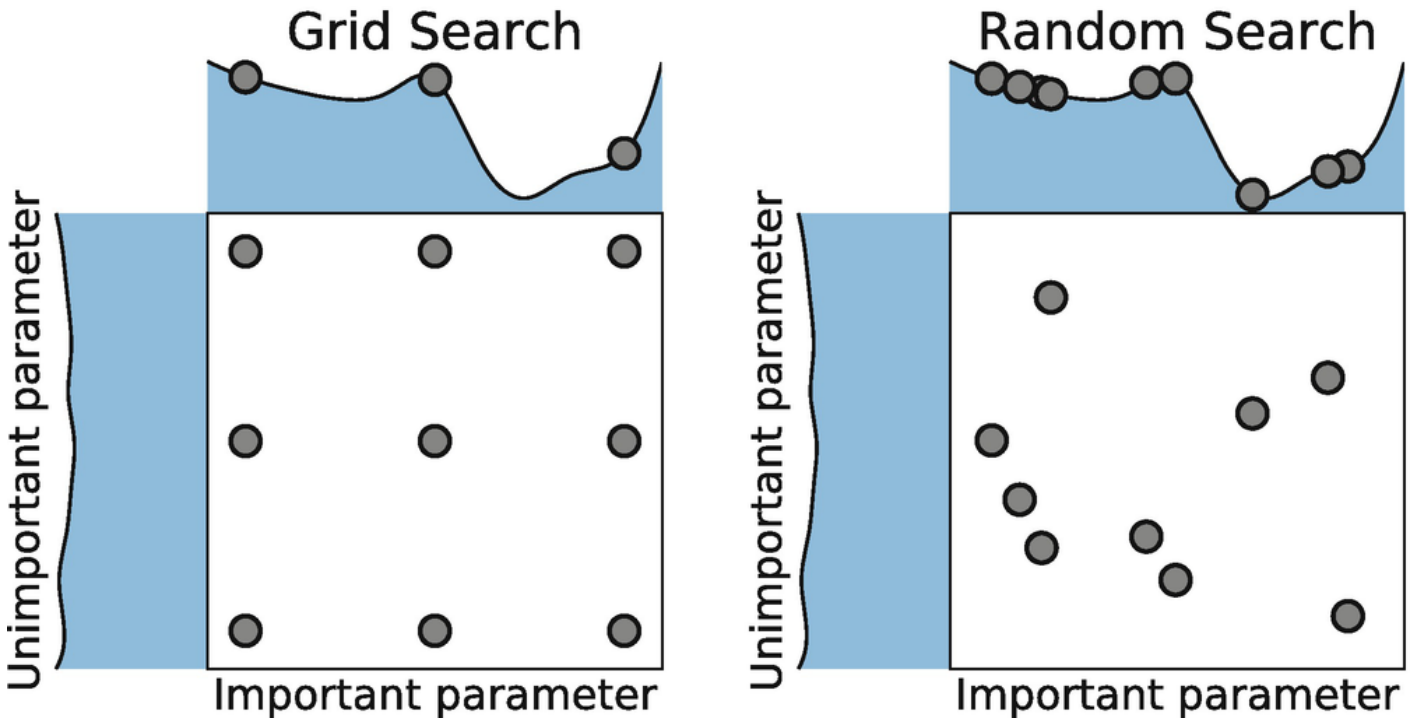
- Again, this returns all scores for all folds for all training set sizes.
- From these we can produce nice plots like:



See: *https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html*

Hyperparameter Selection: Grid Search

- Grid search is an unsophisticated, brute-force technique that works very well in practice.
- There are two main variations: Uniform and Random Grid Search



Hyperparameter Selection: Grid Search (continued)

- The first thing to do is understand which hyperparameters are of interest.
- This almost always requires a detailed perusal of the documentation.
- Consider a linear SVM with hinge loss: the model essentially has only one hyperparameter: the C used to weight model complexity versus empirical loss:

$$
\begin{aligned}
f(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\
\mathcal{L}(D; \mathbf{w}, b) &= \min_{\mathbf{w}} ||\mathbf{w}||_2 + C \sum_{(\mathrm{x},y)\in D} \max(0, 1 - yf(\mathbf{x})) \\
\text{class}(\mathbf{x}) &= \begin{cases} -1 & \text{if } f(\mathbf{x}) \leq 0 \\ +1 & \text{if } f(\mathbf{x}) > 0 \end{cases}
\end{aligned}
$$

Hyperparameter Selection: Grid Search (continued)

- The key class in sklearn is `sklearn.model_selection.GridSearchCV`.
- What must provide to `GridSearchCV` is a grid of parameters to search:

```
from sklearn.model_selection import GridSearchCV
model = LinearSVC(max_iter=2000)
param_grid = {'C': [0.001, 0.1, 1.0, 10, 20, 50, 100, 1000]}
search = GridSearchCV(model, param_grid, cv=3, verbose=3, n_jobs=4)
search.fit(X_tr, y_tr)
test_score = accuracy_score(y_te, search.best_estimator_.predict(X_te))
print(f'Best parameters: {search.best_params_}')
print(f'Best cross-val score: {search.best_score_}')
print(f'Score on test set: {test_score}')

Fitting 3 folds for each of 8 candidates, totalling 24 fits
...
```

Hyperparameter Selection: Grid Search (continued)

- What if we have more hyperparameters?
- For example, in `LinearSVC` we can also choose the type of penalty (L1 or L2).
- Well, we can just add them to the grid:

```
...
param_grid = {'C': [0.001, 0.1, 1.0, 10, 20, 50, 100, 1000],
              'penalty': ['l1', 'l2']}
...

Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

Reflections

Reflections

- We have barely scratched the surface of what we could say about classical machine learning techniques.
- Just an example: we haven't said anything about Bayesian Discriminant methods that return probabilities of class membership.
- Contemporary machine learning is a massive beast, with theoretical, methodological, and practical depth.
- Take home message: applied machine learning is *an experimental science* – to do it, you must approach problems experimentally:
  1. Formulate a hypothesis.
  2. Design and perform an experiment to test it.
  3. Evaluate the results and refine your hypothesis.
  4. Repeat.

Some links

- Here is the Official Python 3 Tutorial if you want to take a deeper look at the Python programming language.
- The The Scikit-learn User Guide is full of useful information and examples.
- Here is a Harvard Article on the History of AI (with many links).

Some references

- The canonical reference for Artificial Intelligence:

  Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach. Pearson Education Limited, 2016.

- The canonical reference for Machine Learning:

  Christopher Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

- A recent book on Deep Learning:

  Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep learning. MIT press, 2016.

- An excellent general book on Artificial General Intelligence:

  Nick Bostrom, Superintelligence. Dunod, 2017.

Laboratory

- The laboratory notebook for today:

`http://bit.ly/I40ML`