Introduction to Machine Learning

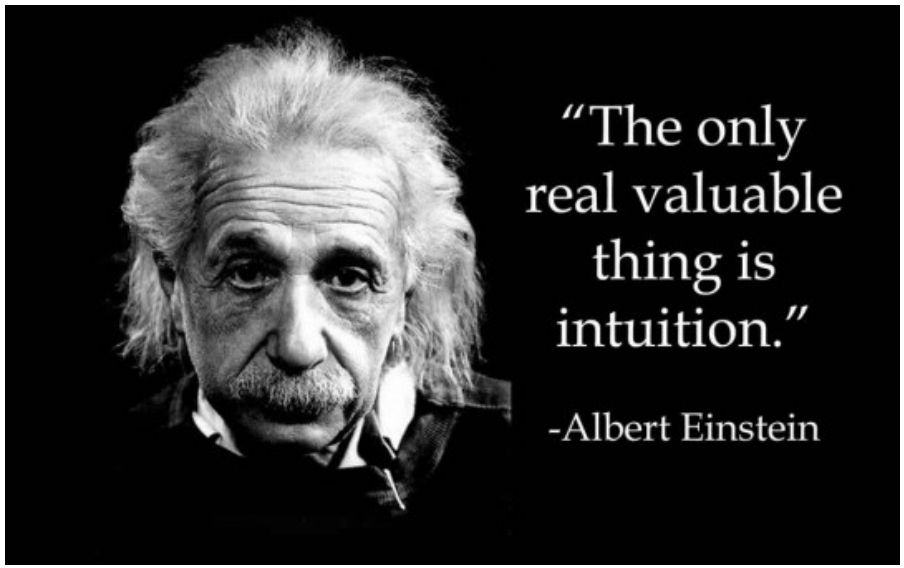
Part 1

Tomer Galanti & Sagie Benaim

March 6, 2019

Today's Lecture

- A bird's eye view of machine learning.
- A machine learning toy example.
- A machine learning toolbox.



Definitions

- Computer science: ”input-output” - what are the sufficient/required conditions to translate the given input to the desired output.
- Basic algorithms: running a predefined program. Not adapting to the input over time. Never makes a mistake.
- Experience: data-driven task, thus statistics, probability.
- Machine learning: computational methods using experience to improve performance, e.g., to make accurate predictions.

Examples

Basic algorithms: graph algorithms, data structures, combinatorial algorithms, string manipulations, cryptography, etc'.

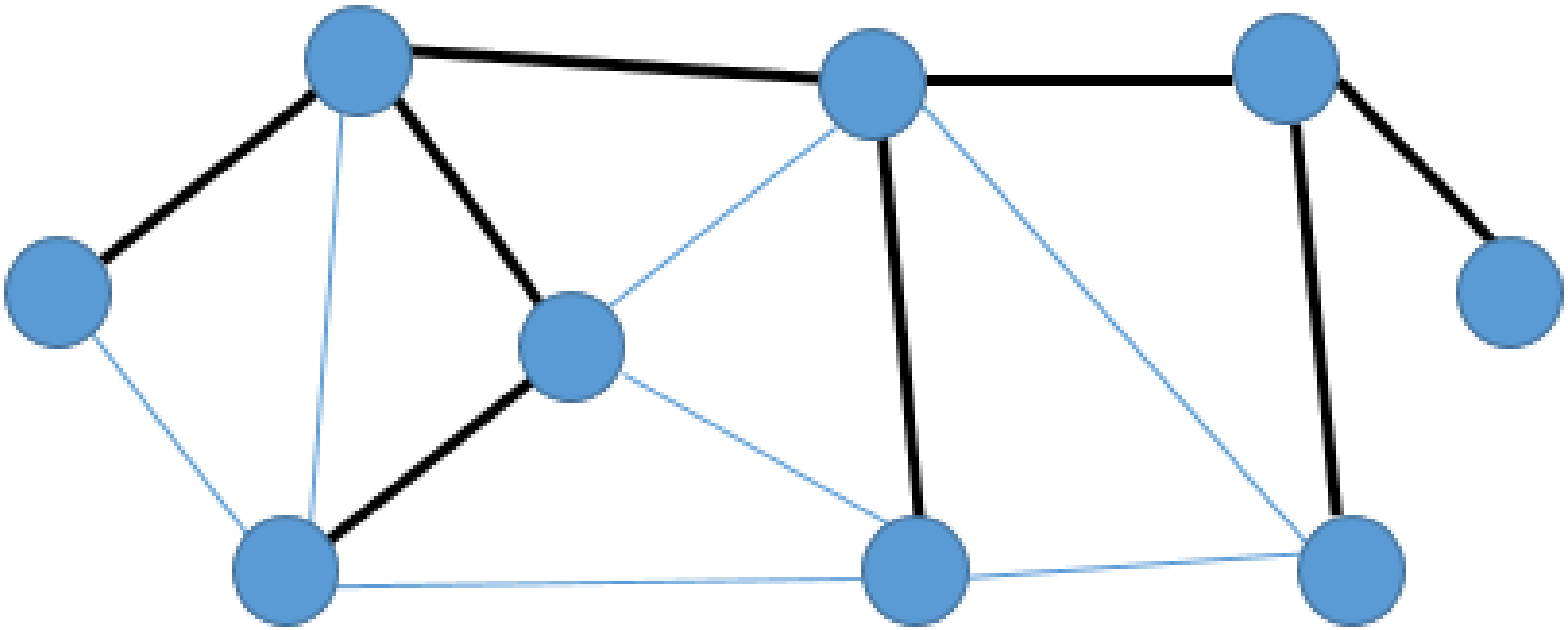


Figure: Spanning tree.

Examples

Machine learning tasks:

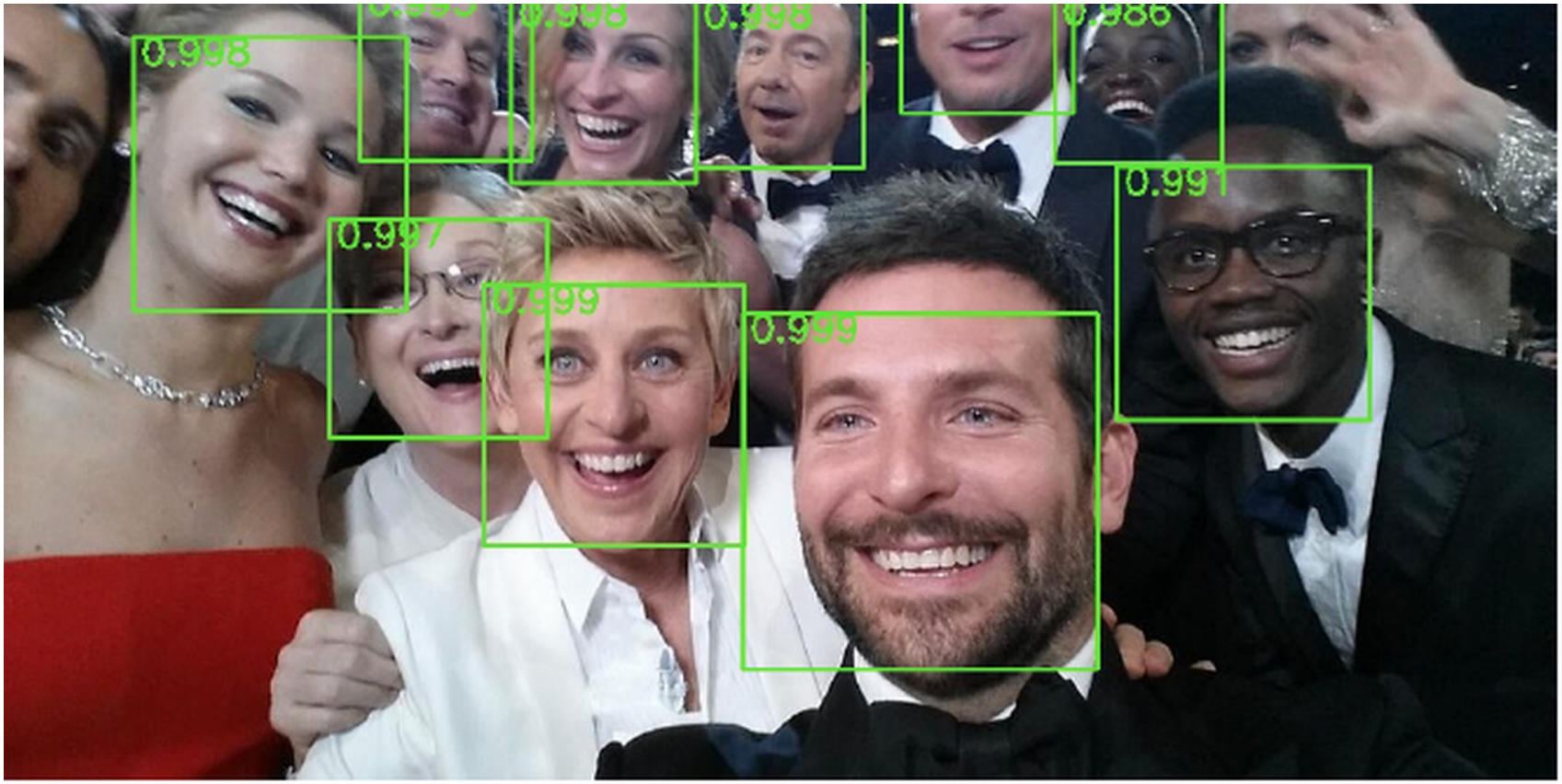
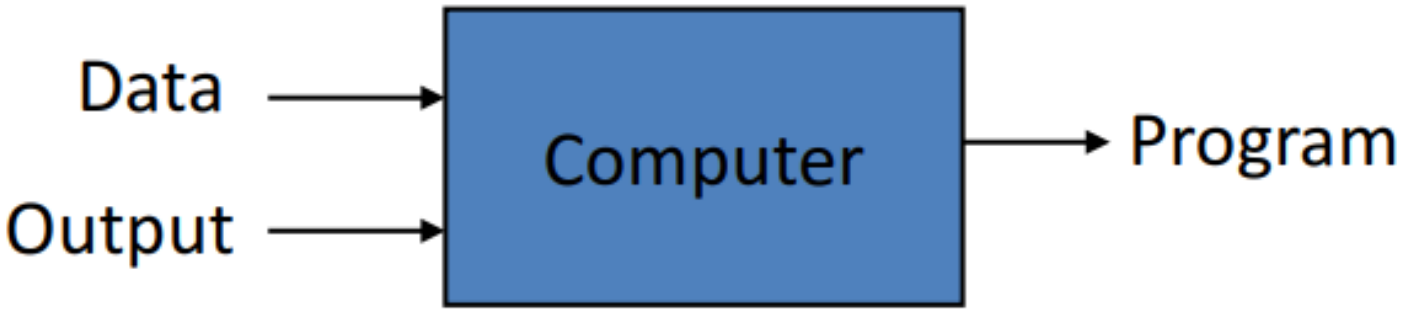


Figure: Image Recognition.

Traditional programming vs. Machine learning

Traditional Programming

Machine Learning



Some Broad Areas of Machine Learning

- Classification: a category to each object (text classification, image detection, speech recognition, etc').
- Regression: predict a real value for each object (price, quality of an image, rating, stock values, etc').
- Clustering: divide data into homogenous groups (analysis of large datasets).
- Density estimation: estimating the probability density function of the distribution from which the data has been sampled.
- Generation: modeling the distribution from which the data has been sampled.

Supervised Learning

Supervised learning is the machine learning task of learning a function $y : x \mapsto y(x)$ that maps an input to an output based on example input-output pairs $(x_i, y(x_i))$.

Supervised Learning

Unsupervised learning is a branch of machine learning that learns from test data that has not been labeled, classified or categorized.

Unsupervised learning identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data.

Some Broad Areas of Machine Learning

Classification:

- Supervised learning.
- Domain: instances with discrete labels.
  - Emails: spam/not spam.
  - Patients: ill/healthy.
  - Credit card: legit/fraud.
  - Images: day/night.
- Input: training data.
  - Examples from the domain.
  - Selected randomly.
- Output: classifier (linear classifier, decision tree, neural network, etc').
  - Classifies new objects.

Some Broad Areas of Machine Learning

Regression:

- Supervised learning.
- Domain: instances with continuous labels.
  - Movies rating.
  - House prices.
  - Stock values.
- Input: training data.
  - Examples from the domain.
  - Selected randomly.
- Output: predictor.
  - Predicts the value of new instances.

Some Broad Areas of Machine Learning

Clustering:

- Unsupervised learning.
- Domain: instances without labels.
  - Document classification.
  - User modeling.
- Input: training data.
- Output: clustering - partition of the space.
  - Somewhat fuzzy goal.

When do we need ML?

- Some tasks are simply too difficult to program.
- Machine learning gives a conceptual alternative to programming.
- What is ML: Arthur Samuel (1959): ”Field of study that gives computers the ability to learn without being explicitly programmed”.

Intro to ML

Let us illustrate the machine learning protocol using a simple toy example.

Intro to ML

Suppose we want to program a function $y : \mathbb{R}^n \rightarrow \{0, 1\}$ that classifies between images of cats and dogs. Each image is encoded as a vector of dimension n.

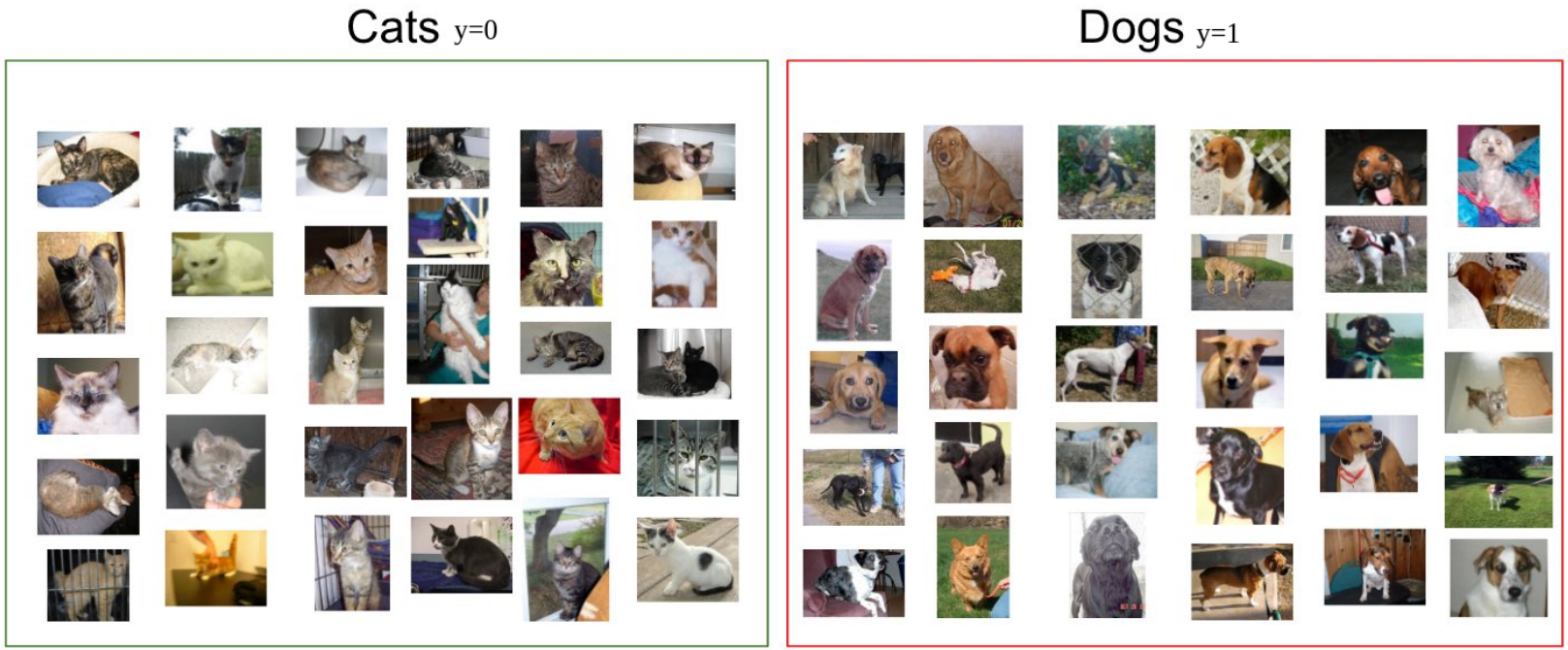


Figure: A classifier for images of cats and dogs. $y(x) = 0$ if x is an image of a cat and $y(x) = 1$ otherwise.

Unfortunately, we do not have a formal definition of ”cat vs. dog”. Therefore, this **target function**, y, is considered unknown and cannot be programmed explicitly.

This is the typical situation when considering applying machine learning.

Instead of programming y explicitly, we are given a **training set** of m labeled images:

$$S = \{(x_i, y(x_i))\}_{i=1}^{m} \tag{1}$$

Illustratively:

$$S = \left\{ \left(\text{  }, 0 \right), \left(\text{  }, 1 \right), \left(\text{  }, 0 \right), \ldots \right\}$$

Intro to ML

In machine learning, we assume that the instances in the domain are samples of a distribution D (distribution of images of cats and dogs).

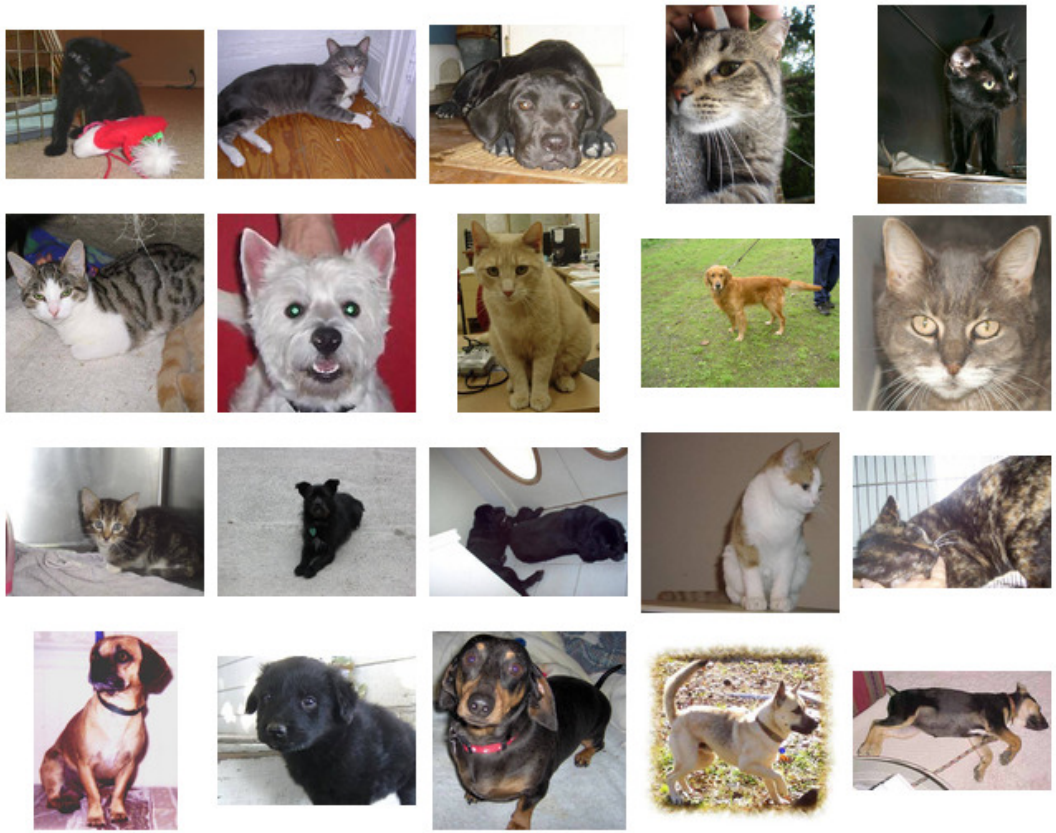


Figure: The distribution of images of cats and dogs.

In machine learning, we assume that the instances in the domain are samples of a distribution D (distribution of images of cats and dogs).

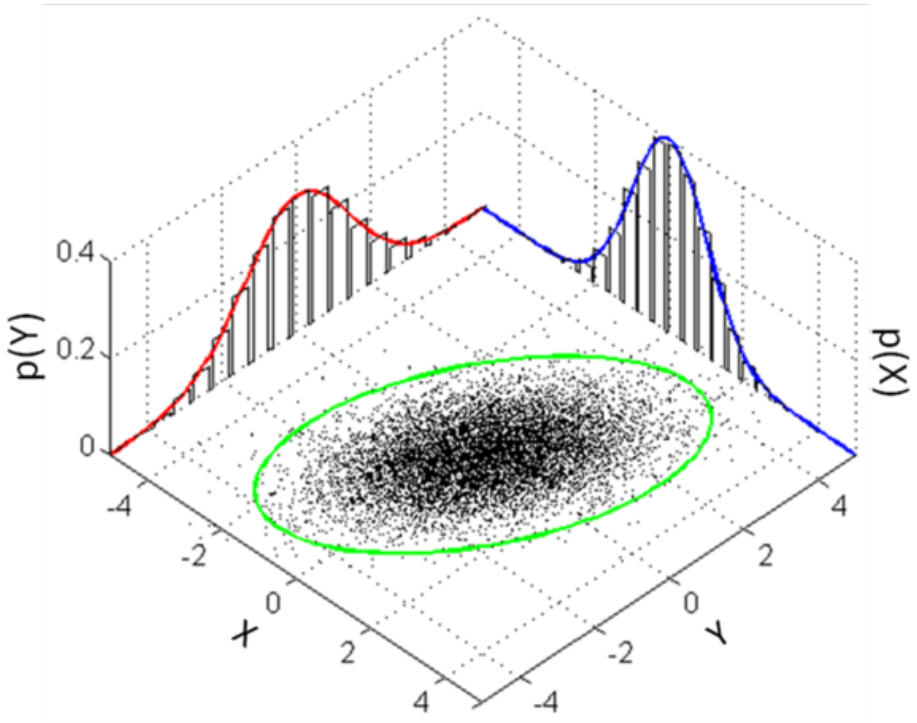
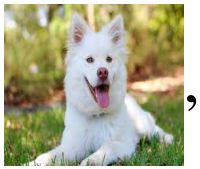


Figure: A Multivariate Normal distribution over $\mathbb{R}^2$.

Training set: $S = \{(x_i, y(x_i))\}_{i=1}^{m}$ of m labeled samples.

We assume that the instances $x_i \sim D$ are
identically independently distributed (**i.i.d** for short).

$$S = \left\{ \left(\text{  }, 0 \right), \left(\text{  }, 1 \right), \left(\text{  }, 0 \right), \ldots \right\}$$

We assume that the instances $x_i \sim D$ are identically independently distributed (i.i.d for short):

- Identically distributed: all of the samples are taken from D.
- Independently distributed: $x_i \sim D$ is independent of $x_j \sim D$.

Anyone knows why do we need the two assumptions?

Why do we need i.i.dness?

Identically distributed:

- D = the instances domain (images of cats and dogs).
- All of the samples are from the same domain.
- We do not have samples from a domain of horse images in our training data if our task is to classify between images of cats and dogs.

Why do we need i.i.dness?

Indepedently distributed:

- It is unrealistic to say that the second image in the training set depends on the tenth image of the training set.
- The i'th label $y(x_i)$ is independent of the j'th instance x_j.

Intro to ML

- There is an unknown target function y (cats vs. dogs) that we would like to program.
- There is a distribution D of images of cats and dogs.
- We are given a training set: $S = \{(x_i, y(x_i))\}_{i=1}^{m}$.
- x_i are i.i.d according to D.

Any questions so far?

So now we have a much more relaxed version of this problem.

Instead of computing y from scratch, we have some guidance that tells us what are the values of y on a finite set of examples $x_1, \ldots, x_m$.

What can we expect to do in this case? Can we compute y perfectly?

No! We cannot compute y perfectly.

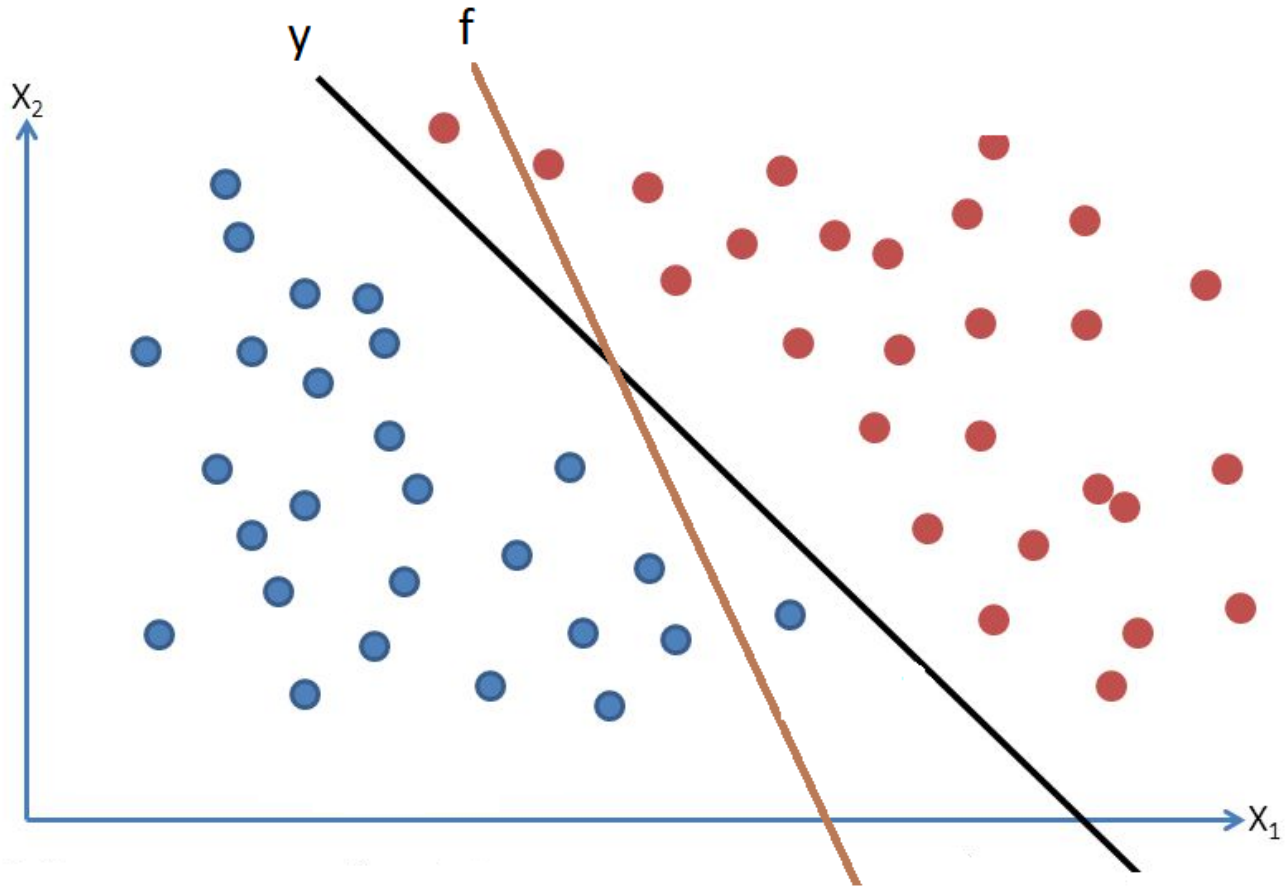


Figure: Both the black and brown separators are consistent with the seen data, but they classify differently.

No! We cannot compute y perfectly. A-priori, y could be any of the two separators. Given the data, we could return any of the two separators as our output function. One of them makes a mistake.

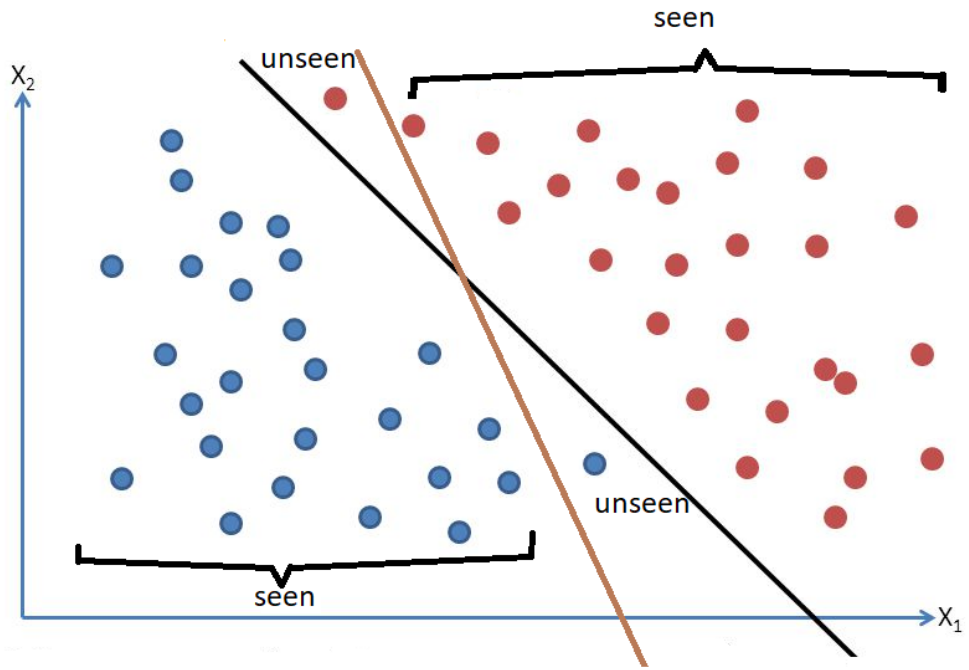


Figure: Both the black and brown separators are consistent with the seen data, but they classify differently.

Therefore, instead of trying to compute y perfectly;
Our goal is to take a training set S as input and to return a function f that approximates y.

We will have some error in approximating y.

Intro to ML

Our goal is to take a training set S as input and to return a function f that approximates y.

What do we mean by approximation?

First, we need to define a **loss function** $x \mapsto \ell(f(x), y(x)) \in [0, \infty)$ (per-sample error) to measure the error of a candidate function f on a specific instance x.

In machine learning there are many kinds of loss functions. In our case, we will employ the **zero-one loss function**.

An indicator if f makes a mistake on x or not,

$$\ell(f(x), y(x)) := \text{Ind}[f(x) \neq y(x)] \tag{2}$$

The error's value is 1 if f mistakes in classifying x correctly and the value is 0 otherwise.

What is the error of f on each instance?

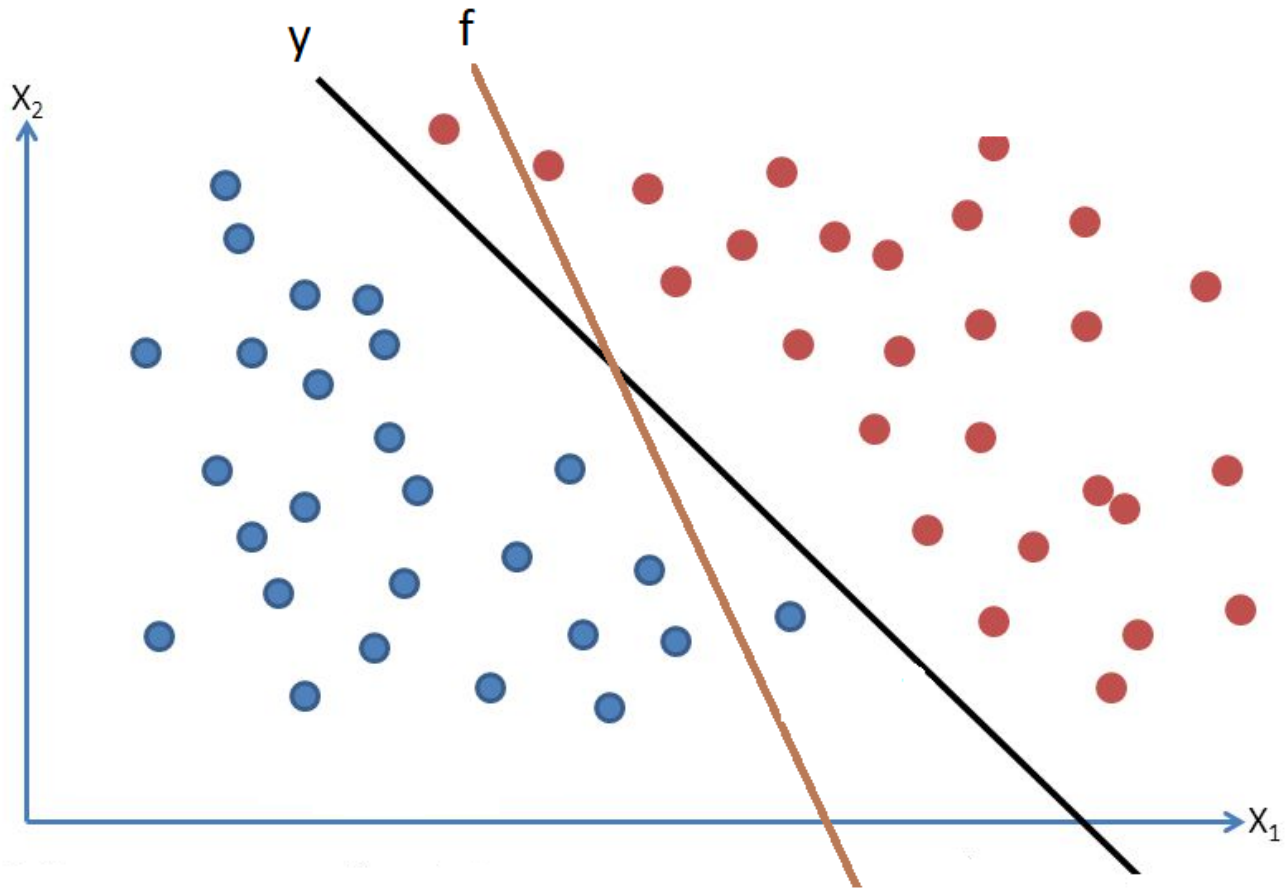


Figure: The black separator is y.

Since we assumed that the instances are distributed by D, a desired output function f of the algorithm is one that has a small error on the population D.

The error on the population D is measured by the **generalization error**,

$$\begin{aligned}\mathcal{L}_D[f, y] &:= \mathbb{E}_{x\sim D}[\ell(f(x), y(x))] \\ &:= \mathbb{E}_{x\sim D}\overbrace{[\mathrm{Ind}[f(x) \neq y(x)]]}^{\text{Per-sample error}} \\ &= \text{"Expected value of the error on a new instance from } D\text{"}\end{aligned} \tag{3}$$

Note: in this case; generalization error = probability f makes a mistake.

What is the generalization error of f?

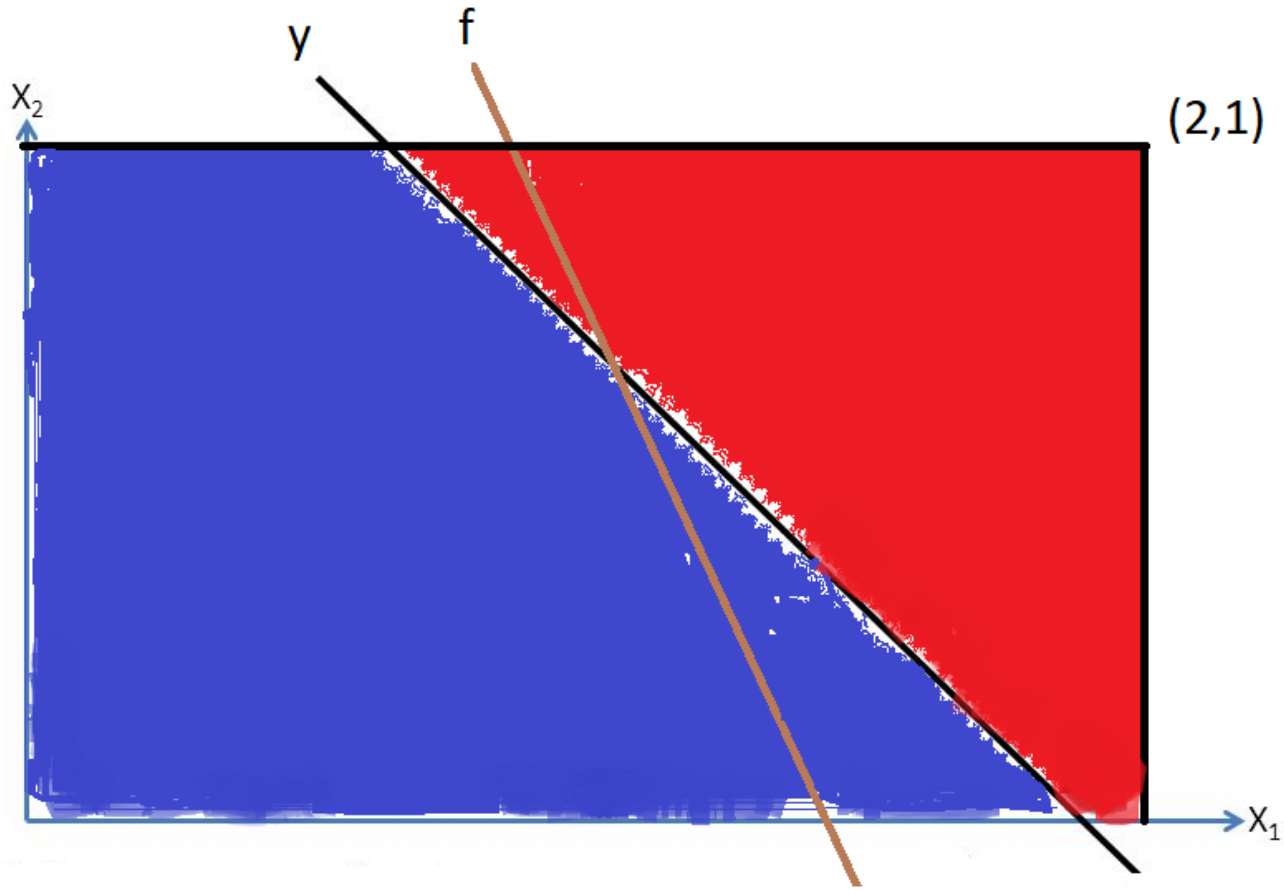


Figure: Assume D is a uniform distribution over the rectangle that is caged within the points $(0,0),(2,0),(0,1),(2,1)$. What is the generalization error of f?

Intro to ML

- There is an unknown target function y (cats vs. dogs) that we would like to learn.
- There is an unknown distribution D of instances.
- We are given a training set: $S = \{(x_i, y(x_i))\}_{i=1}^{m}$.
- x_i are i.i.d according to D.
- We would like to return a function f that has a low generalization error: $\mathcal{L}_D[f, y]$.

Any questions so far?

The central challenge in machine learning is that the algorithm's output f must perform well on new, previously unseen instances - not just those on which our model was trained, i.e., to minimize, $\mathcal{L}_D[f, y]$. This ability is called **generalization**.

Typically, when training a machine learning model, we have access to a training set S; we can compute some error measure on the training set

$$\mathcal{L}_S[f, y] := \underbrace{\frac{1}{m}\sum_{i=1}^{m} \overbrace{\text{Ind}[f(x) \neq y(x)]}^{\text{Per-sample error}}}_{\text{Empirical error}} \tag{4}$$

$$= \text{"Average value of the error in the training dataset } S\text{"}$$

called the **empirical error**; and we reduce it. So far, what we have described is simply an optimization problem.

What is the empirical error of f?

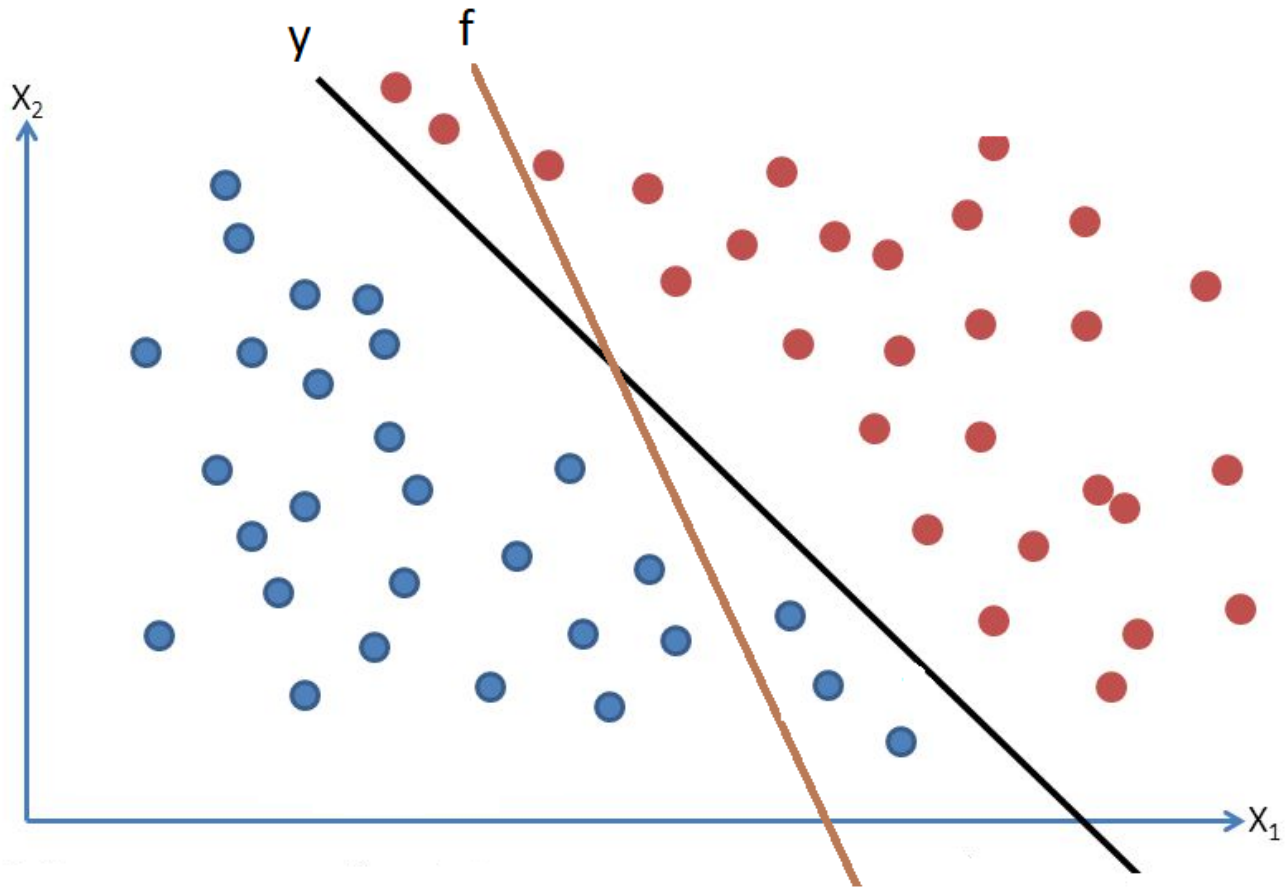


Figure: The dataset consists of the red and blue points.

Capacity, overfitting and underfitting

Machine learning differs from optimization. In ML, we want the generalization error $\mathcal{L}_D[f, y]$, to be low as well. The generalization error is defined as the expected value of the error on a new input.

Capacity, overfitting and underfitting

In general, minimizing the empirical error, $\mathcal{L}_S[f, y]$, does not guarantee minimizing the generalization error, $\mathcal{L}_D[f, y]$.

Capacity, overfitting and underfitting

For example, we can take a classifier f_S that is defined as:

$$f_S(x) = \begin{cases} y(x) & x \in S \\ 1 - y(x) & x \notin S \end{cases} \tag{5}$$

This classifier satisfies: $\mathcal{L}_S[f, y] = 0$ and $\mathcal{L}_D[f, y]$ might be large (equals 1 when D is continuous).

Capacity, overfitting and underfitting

$$f_S(x) = \begin{cases} y(x) & x \in S \\ 1 - y(x) & x \notin S \end{cases} \tag{6}$$

This property is called **overfitting**. The classifier perfectly fits the training data, but does not fit the distribution D. As we can see, an algorithm that takes S and returns f_S does not really learn anything about D from S. This classifier just memorizes all the dataset S.

Capacity, overfitting and underfitting

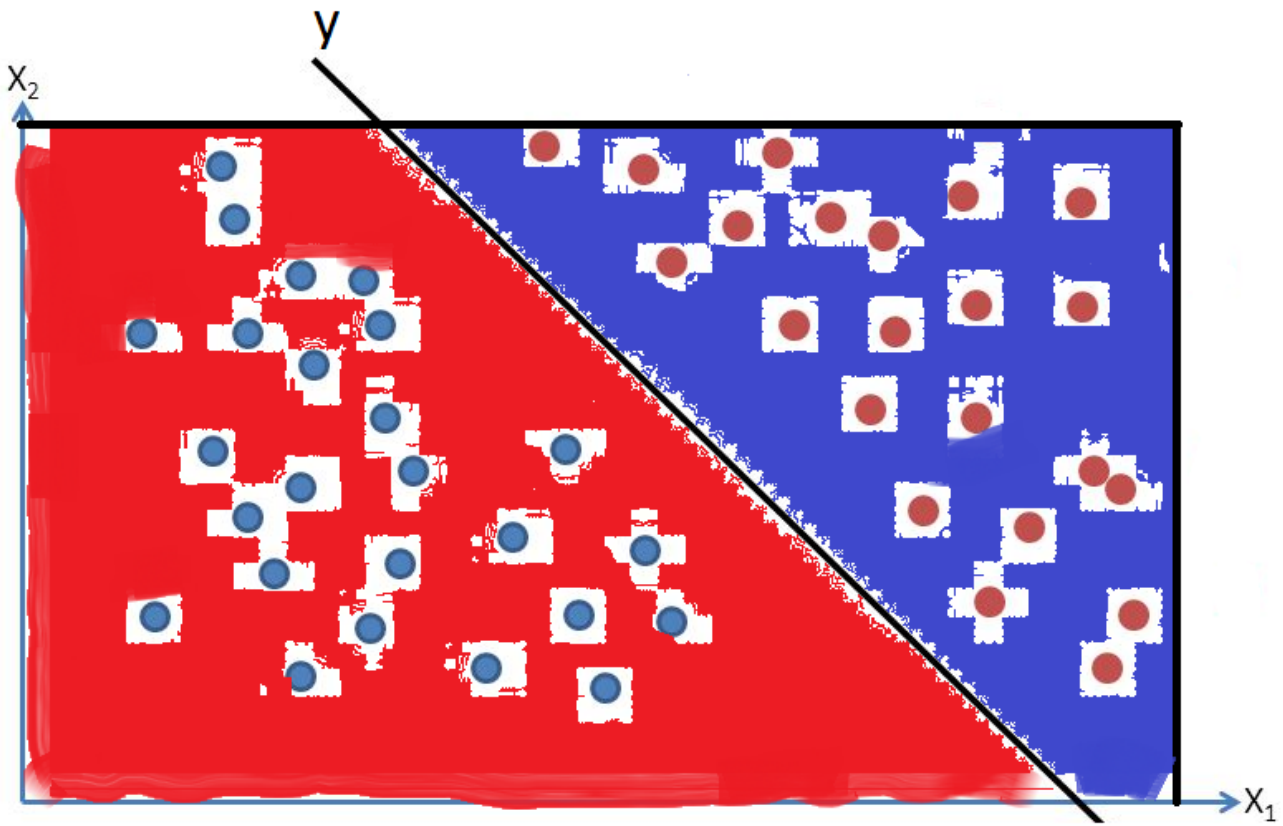


Figure: An illustration of f_S. Red = 1, Blue = 0. y is a linear separator that returns 1 for every x above the black line and 0 for every x under the black line. f_S returns the correct label for every training sample and the wrong label on every instance outside the training dataset.

Capacity, overfitting and underfitting

How do we measure the generalization error?

We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** $S' = \{(\bar{x}_i, y(\bar{x}_i))\}_{i=1}^{k}$ of examples that were collected separately from the training set $S = \{(x_i, y(x_i))\}_{i=1}^{m}$.

Capacity, overfitting and underfitting

The learning procedure:

1. We sample the training set.
2. Choose f to reduce the training error.
3. We sample the test set.
4. We compute the error on the test set.

Capacity, overfitting and underfitting

What's so special about the test set?

- The test set is a fresh new set of samples.
- The test set is not "ours".
- We cannot train on the test set.
- We get the test set $S' = \{(\bar{x}_i, y(\bar{x}_i))\}_{i=1}^{k}$ to **only** estimate the performance on D.

Capacity, overfitting and underfitting

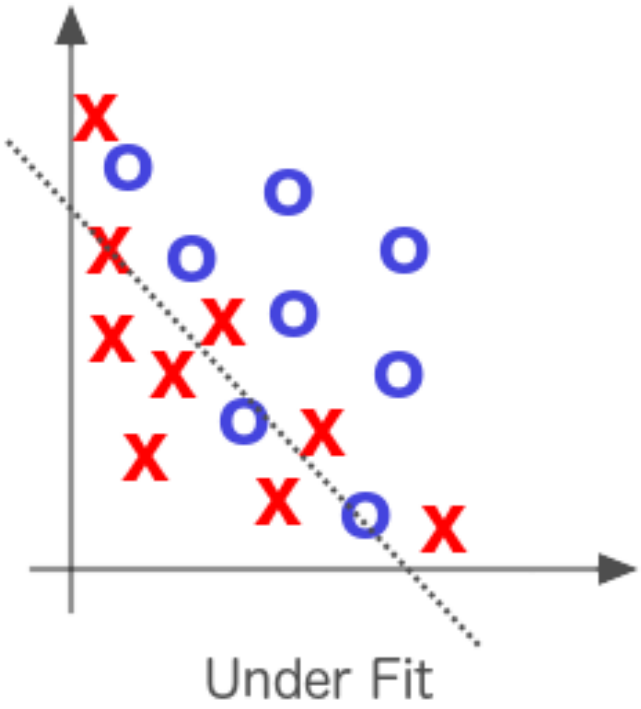
To conclude: the two factors determining how well a machine learning algorithm will perform are:

1. Make the training error small.
2. Make the gap between training and generalization error small.

These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**.

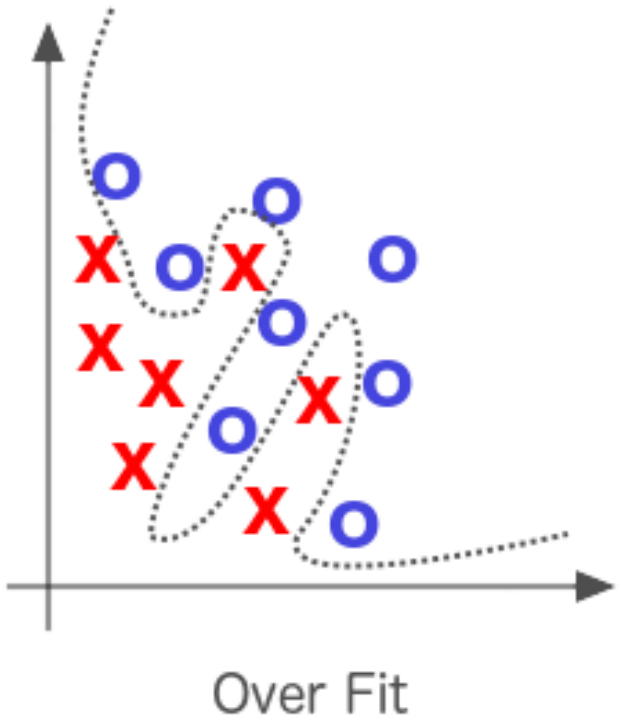
Capacity, overfitting and underfitting

Underfitting: occurs when the algorithm is unable to obtain a sufficiently low error value on the training set $\mathcal{L}_S[f, y]$.



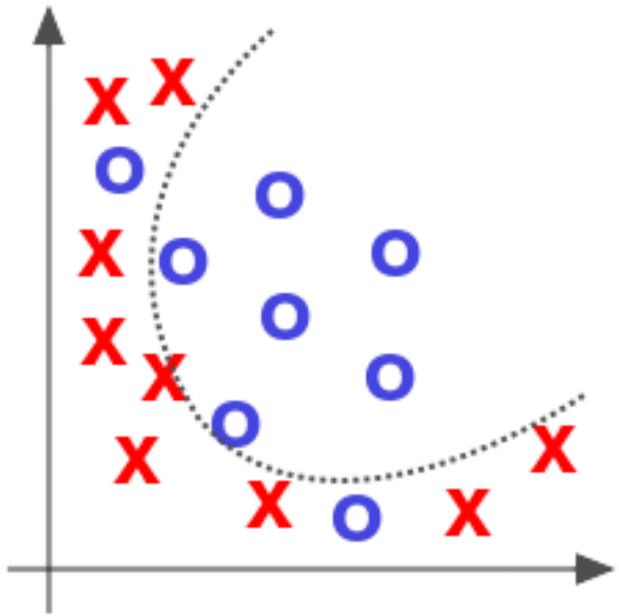
Capacity, overfitting and underfitting

Overfitting: occurs when the gap between the training error $\mathcal{L}_S[f, y]$ and generalization error $\mathcal{L}_D[f, y]$ is too large.



Capacity, overfitting and underfitting

Fitting: occurs when there is no underfitting and overfitting.



Appropriate

Capacity, overfitting and underfitting

We can control whether an algorithm is more likely to overfit or underfit by altering its **capacity**.

Informally, the capacity of an algorithm is its ability to fit a wide variety of functions. Algorithms with low capacity may struggle to fit the training set. Algorithms with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

Capacity, overfitting and underfitting

One way to control the capacity of a learning algorithm is by choosing its **hypothesis space** $\mathcal{H}$, the set of functions that the learning algorithm is allowed to select as being the solution.

For example, we can take $\mathcal{H} = \{\text{sign}(\langle w, x \rangle) | w \in \mathbb{R}^n\}$ to be our hypothesis class. If we add functions of the form $\text{sign}(g(x))$, where g is a polynomial of x with degree $\leq d$, we increase the capacity of the algorithm.

Capacity, overfitting and underfitting

Informal, two very simple (and important) observations.

For $\mathcal{H}_1 \subset \mathcal{H}_2$:

$$\min_{f \in \mathcal{H}_2} \text{train-error}(f) \leq \min_{f \in \mathcal{H}_1} \text{train-error}(f) \tag{7}$$

and

$$\max_{f \in \mathcal{H}_2} \left| \text{train-error}(f) - \text{test-error}(f) \right| \geq \max_{f \in \mathcal{H}_1} \left| \text{train-error}(f) - \text{test-error}(f) \right| \tag{8}$$

(7) = a larger hypothesis class provides better fitting on the training data.
(8) = a larger hypothesis class provides a larger gap between train and generalization errors.

Capacity, overfitting and underfitting

Formal, two very simple (and important) observations.

For $\mathcal{H}_1 \subset \mathcal{H}_2$:

$$\min_{f \in \mathcal{H}_2} \mathcal{L}_S[f, y] \leq \min_{f \in \mathcal{H}_1} \mathcal{L}_S[f, y] \tag{9}$$

and

$$\max_{f \in \mathcal{H}_2} \left| \mathcal{L}_S[f, y] - \mathcal{L}_D[f, y] \right| \geq \max_{f \in \mathcal{H}_1} \left| \mathcal{L}_S[f, y] - \mathcal{L}_D[f, y] \right| \tag{10}$$

Capacity, overfitting and underfitting

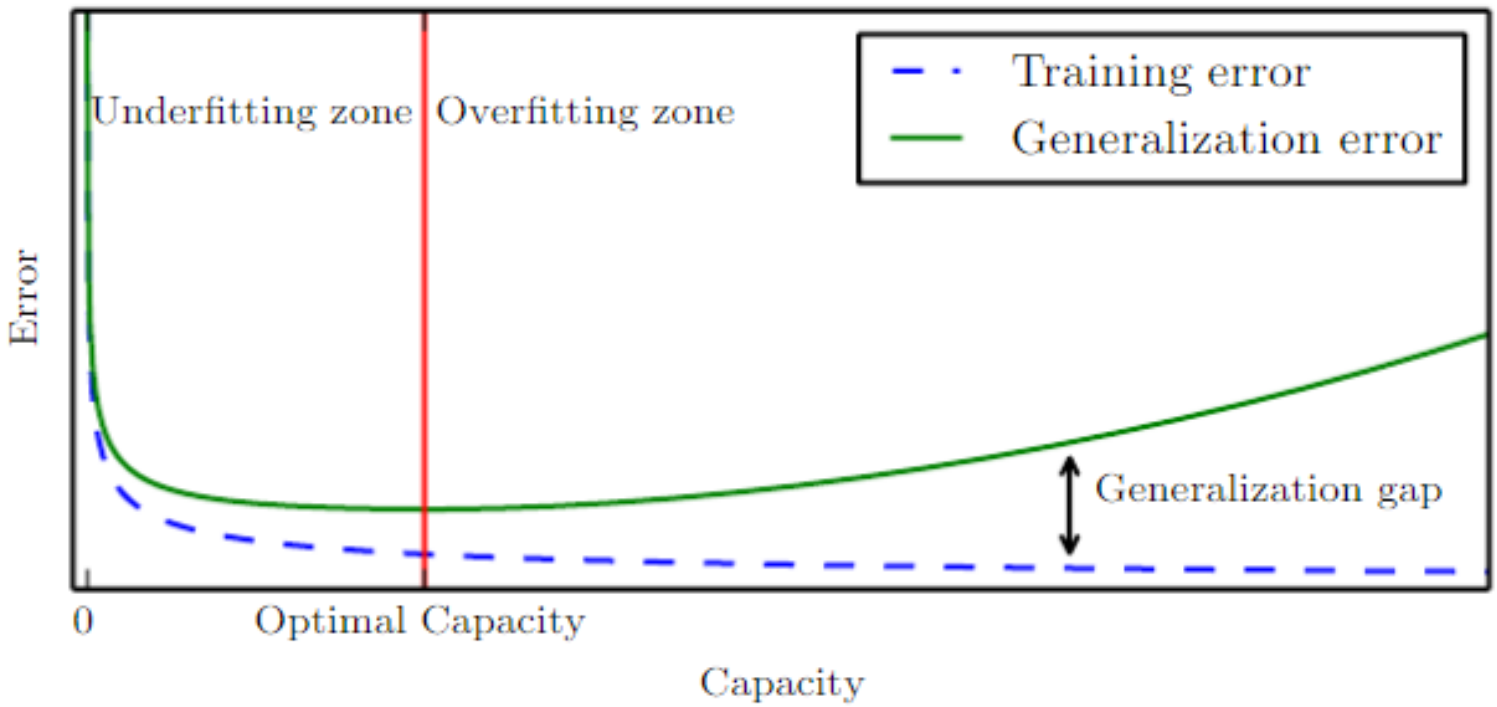
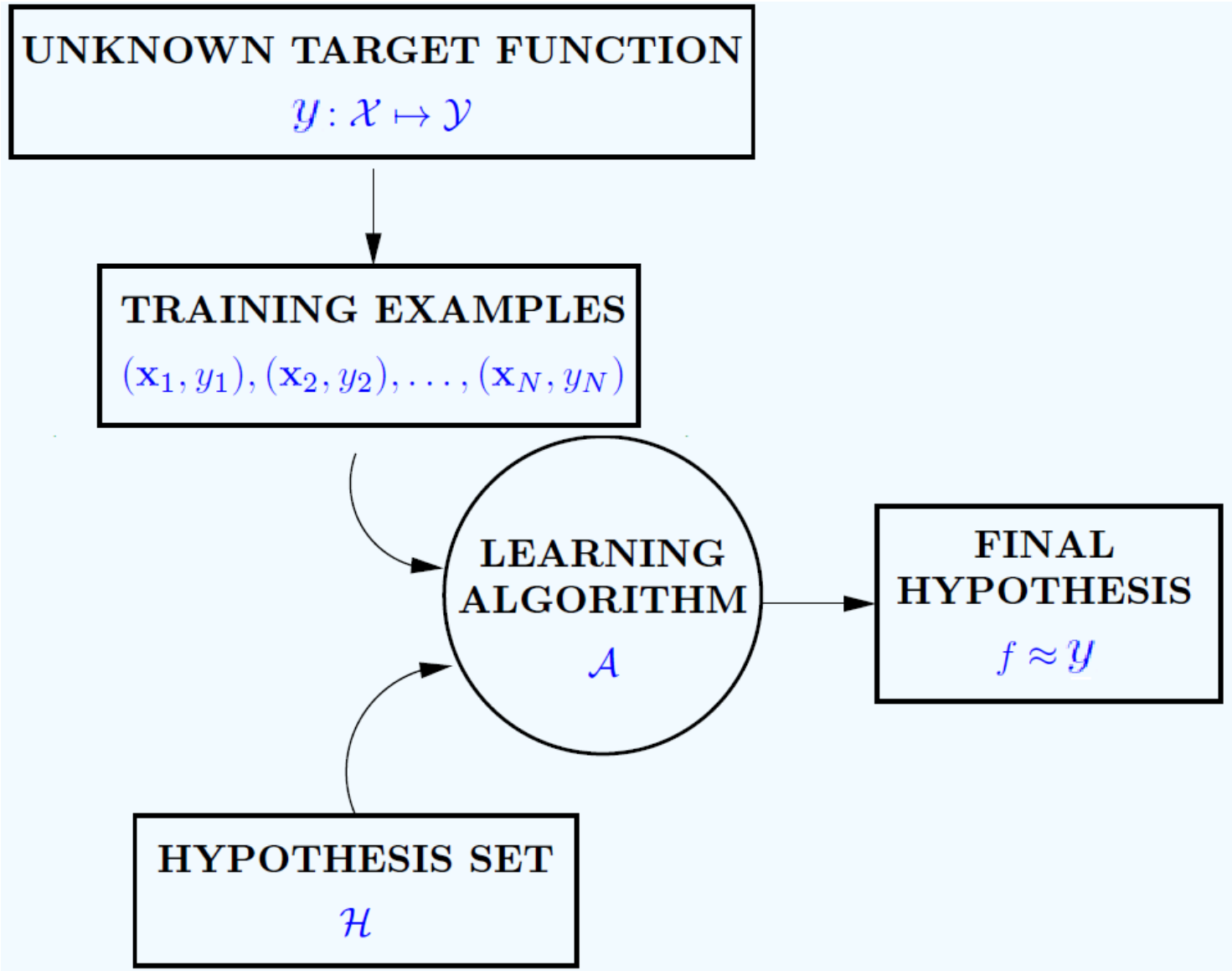
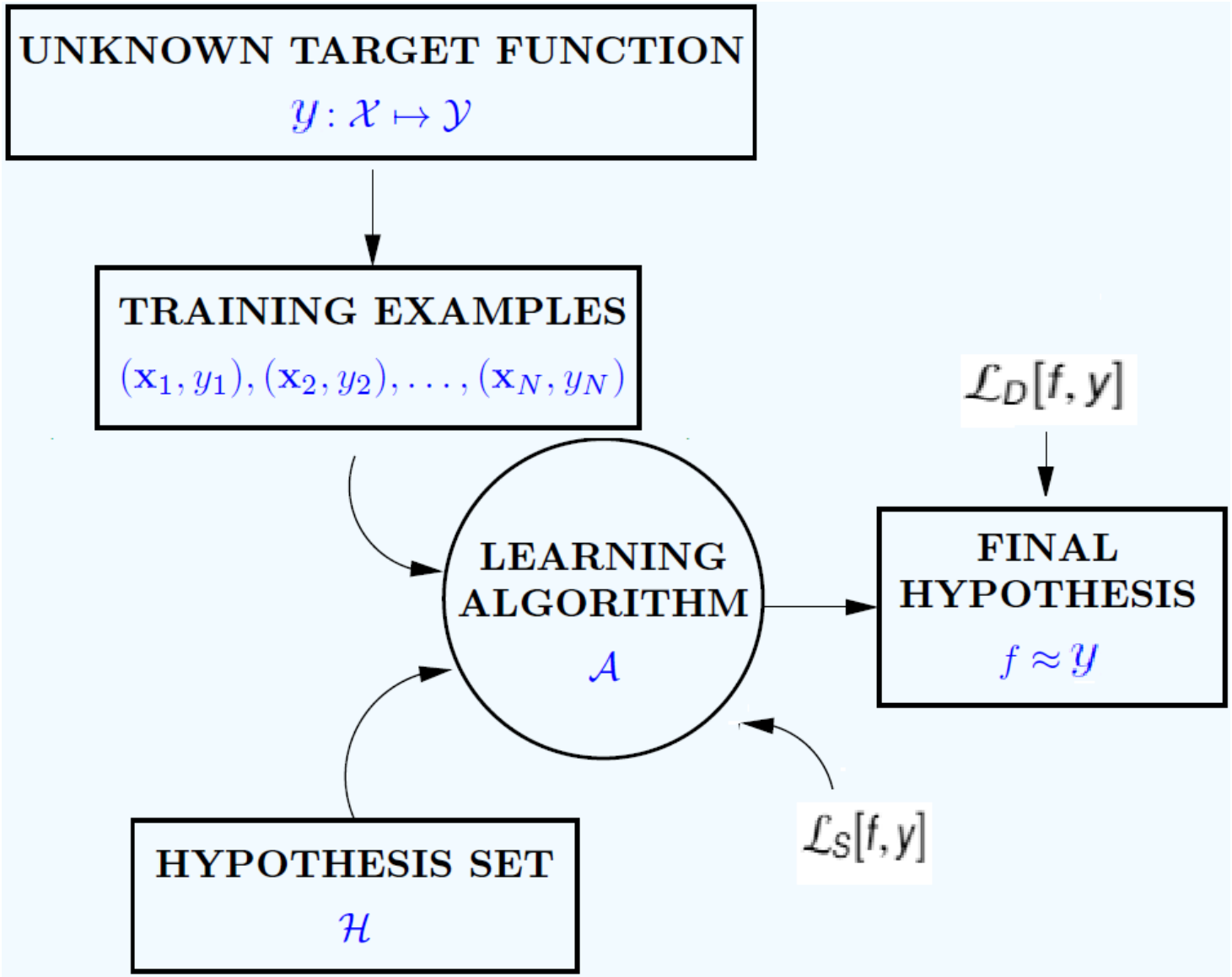


Figure: Typical relationship between capacity and error. At the left end of the graph, training error and generalization error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity.

ML diagram


UNKNOWN TARGET FUNCTION
$y : \mathcal{X} \mapsto \mathcal{Y}$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$f \approx y$
HYPOTHESIS SET
$\mathcal{H}$

ML diagram


UNKNOWN TARGET FUNCTION
$y : \mathcal{X} \mapsto \mathcal{Y}$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
$\mathcal{L}_D[f, y]$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$f \approx y$
HYPOTHESIS SET
$\mathcal{H}$
$\mathcal{L}_S[f, y]$