Machine Learning Basics

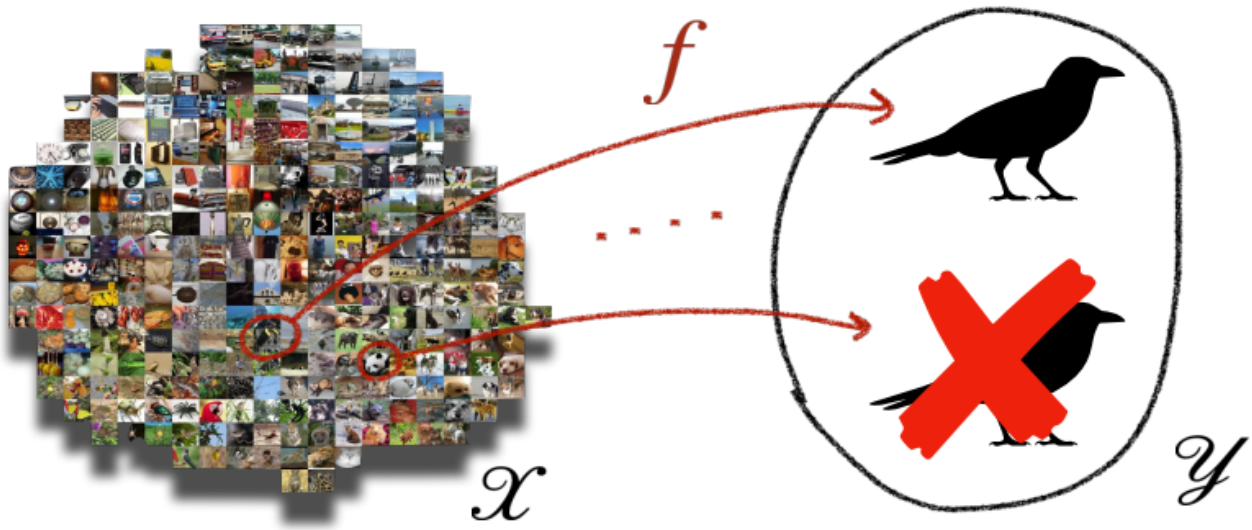
There are many problems one may come across when trying to determine whether or not a picture is of a bird...

#Tasks

A #task represents the type of prediction being made to solve a problem on some data. We can identify a #task with the set of functions that can potentially solve it. In general, it consists of functions assigning each input $x \in X$ an output $y \in Y$. $f : X \to Y$ $F_{task} \subset Y^X$
The nature of X, Y, and F_{task} depends on the type of #task .

#Classification

Find a function $f \in Y^X$ assigning each input $x \in X$ a discrete label. $f(x) \in Y = \{c_1, \ldots, c_k\}$



#Regression

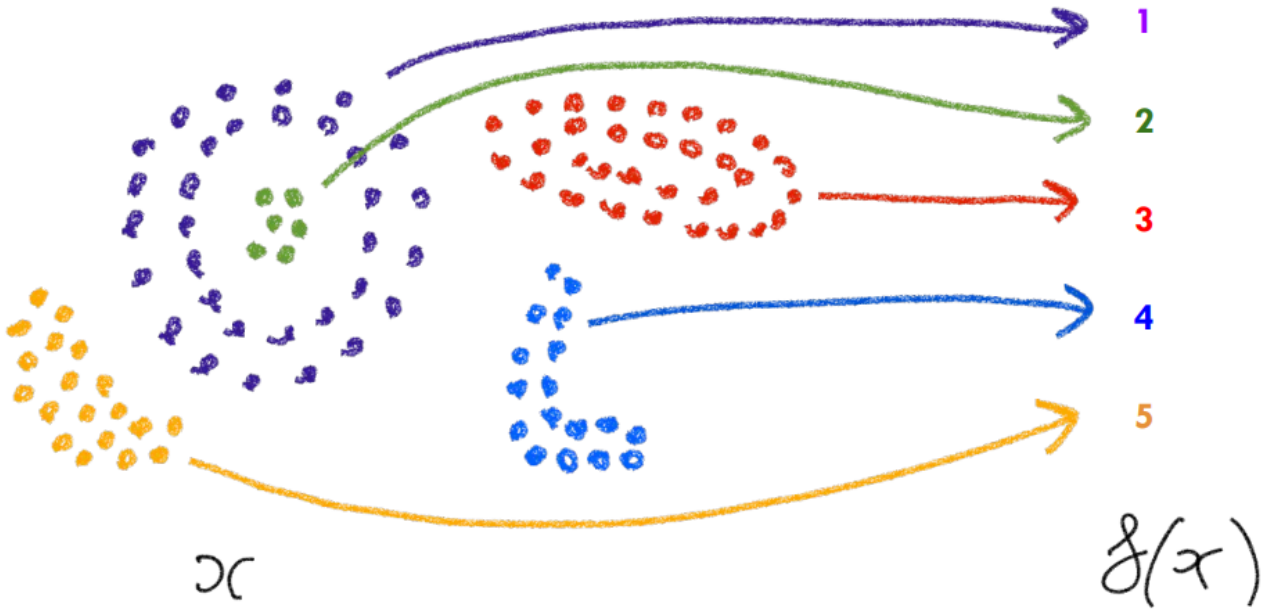
Find a function $f(x) \in Y$ assigning each input a continuous label.

#Density-Estimation

Find a probability distribution $f \in \Delta(X)$ that fits the data $x \in X$

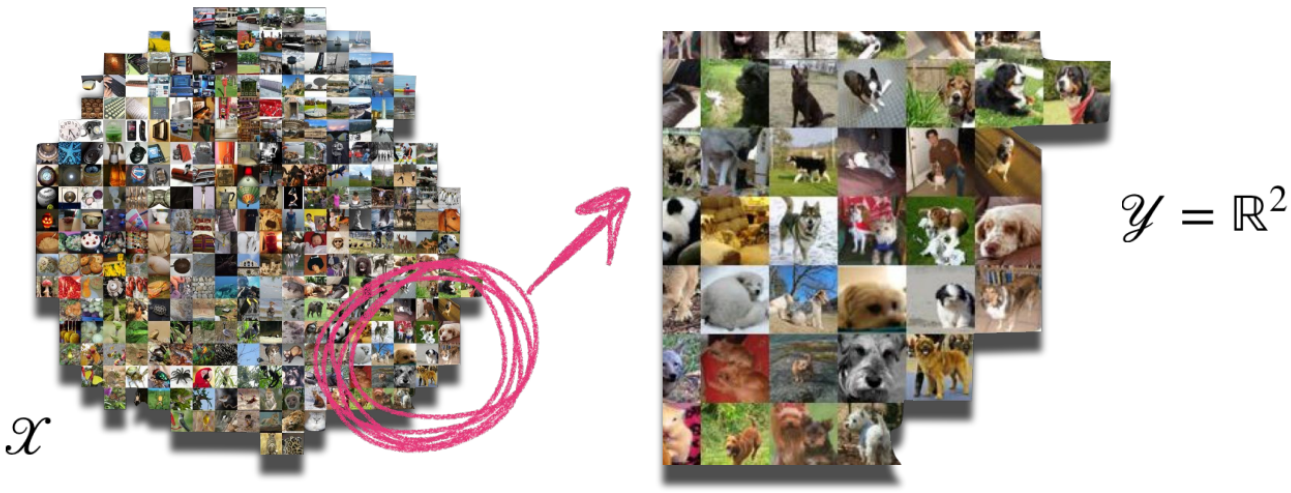
#Clustering

Find a function $f \in \mathbb{N}^x$ that assigns each input $x \in X$ a #cluster index $f(x) \in \mathbb{N}$. All points mapped to the same index form a #cluster .



#Dimensionality-reduction

Find a function $f \in Y^X$ mapping each (high-dimensional) input $x \in X$to a lower dimensional "embedding" $f(x) \in Y$, where $dim(Y) \ll dim(X)$.



> **Note**
>
> #Dimensionality reduction may be used to inspect a classification #algorithm .

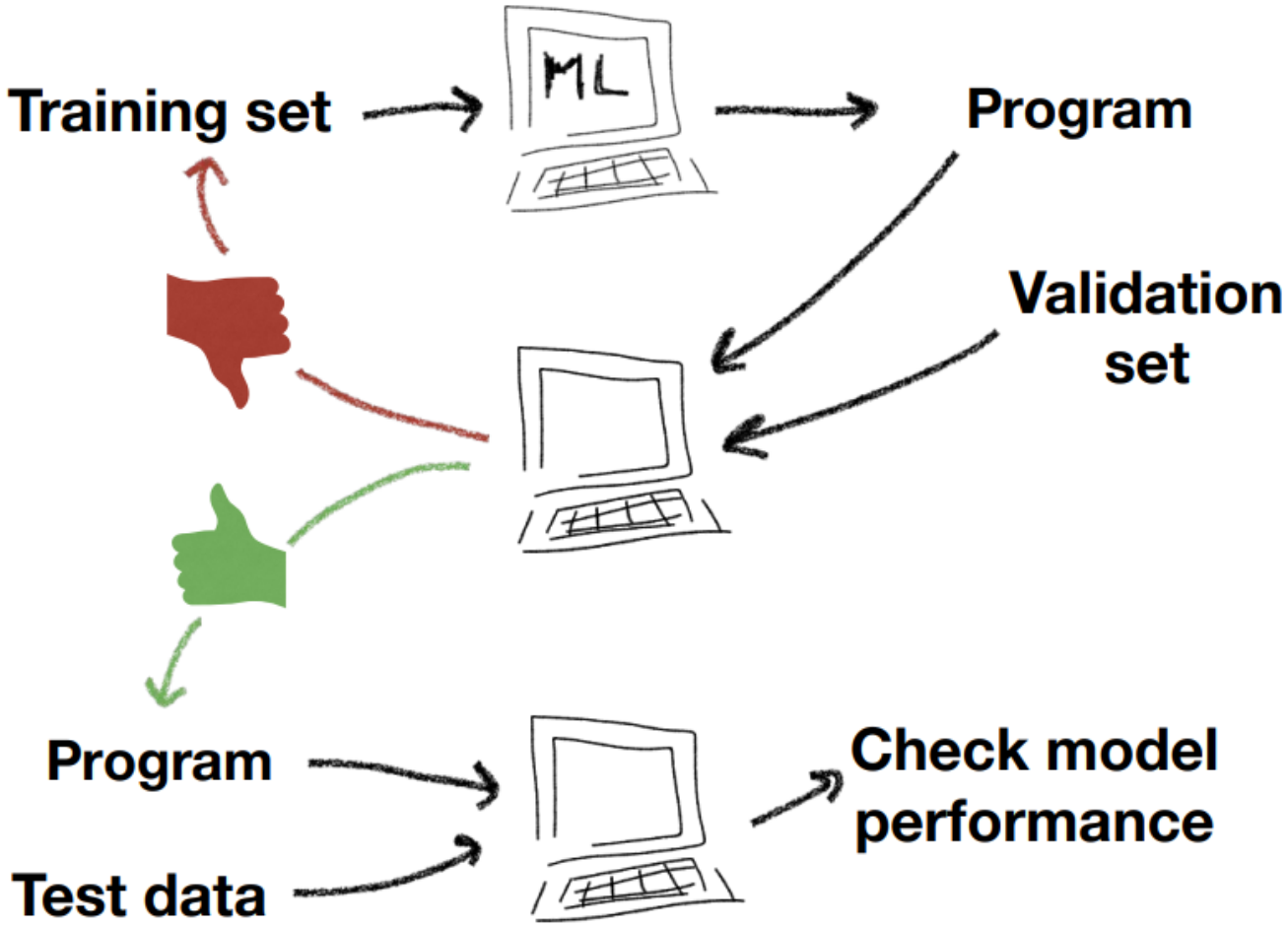
#Data

Data is information about the problem we're trying to solve in the form of a distribution p_{data}.

#Classification and #Regression : $p_{data} \in \Delta(X \times Y)$

#Density-Estimation , #Clustering , and #Dimensionality-reduction : $p_{data} \in \Delta(X)$

The data distribution p_{data} is generally unknown, but we can take samples from it to create the training, validation, and testing sets.[1]



When using a *probabilistic model* of learning, this probability distribution over example/label pairs is called the data generating distribution.

Training set design

The failure of a machine learning #algorithm is often caused by a bad selection of training samples. For example, we might introduce unwanted correlations from which the #algorithm derives wrong conclusions.

> **Example**
>
> If we're trying to make a model that distinguishes between pictures of dogs and pictures of cats, and all of the pictures of dogs are taken on sunny days while all the pictures of cats are taken on cloudy days then the #algorithm is more likely to learn the difference between pictures taken on sunny days and pictures taken on cloudy days.

Models and Hypothesis Space

A model is a program that can solve our problem, or the implementation of a function $f \in F_{task}$ that can be easily computed. A set of models forms a hypothesis space $H \subset F_{task}$. The learning #algorithm seeks a solution within the hypothesis space.

A model may be represented by the expression $f_w(x) = \sum_{j=0}^{M} w_j x^j$ and the hypothesis space may then be represented by $H_M = \{f_w : w \in \mathbb{R}^M\}$, where w_j is the jth parameter and M is a fixed value in $\mathbb{N}$ representing the degree of the function.

Polynomial Curve Fitting Objectives

The ideal target

We want to minimise a generalising error function $E(f; p_{data})$ which determines how well a solution $f \in F_{task}$ fits some given data and guides the selection of the best solution in F_{task}.
$f^\star \in arg\,min_{f \in F_{task}}\, E(f; p_{data}) \mid f \in F_{task}$

Problem

The search space is too large and we don't know p_{data}[1]

1. the distribution of the data regarding the problem we're solving ↩

The feasible target

We need to restrict our focus to finding functions that can be implemented and evaluated easily. We define a model hypothesis space $H \subset F_{task}$ and seek a solution within that space.
$f_H^\star \in arg\,min_{f \in H}\, E(f; p_{data})$

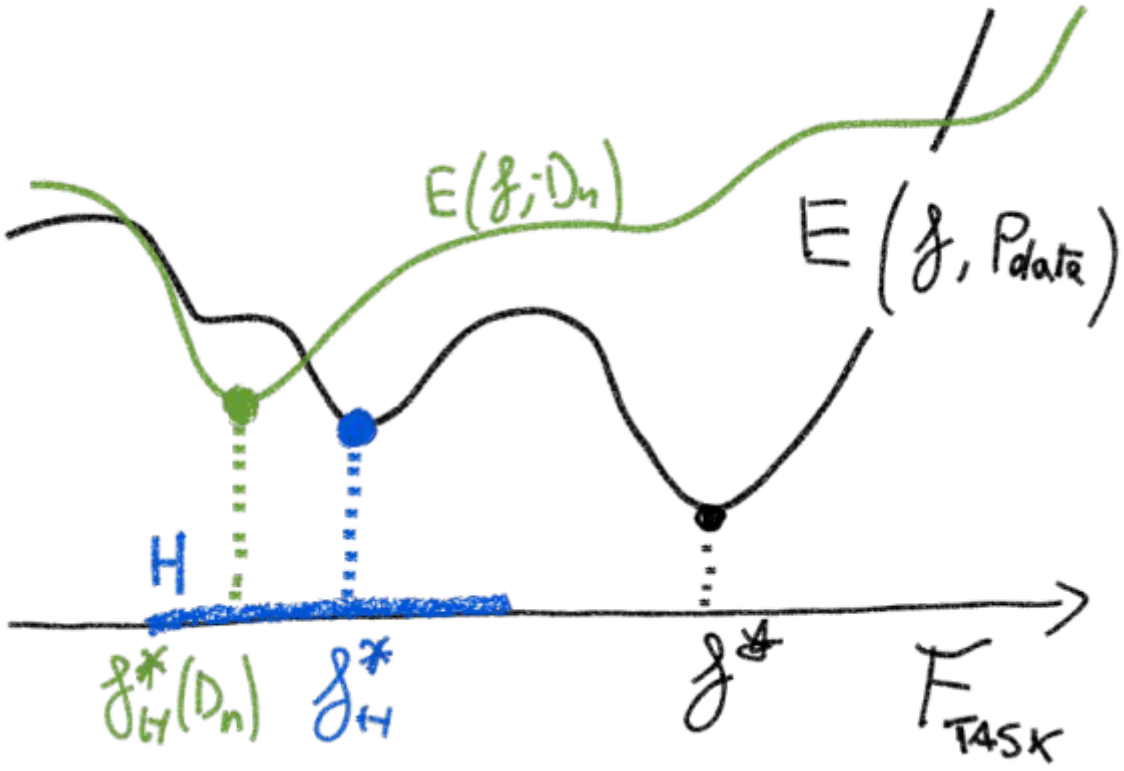
Problem

This cannot be computed exactly, as p_{data} is unknown

The actual target

We need to work on a data sample, i.e. a training set $D_n = \{z_1, \ldots, z_n\}$ where $z_i = (x_i, y_i) \in X \times Y$ and $z_i \sim p_{data}$.
$f_H^\star(D_n) \in arg\,min_{f \in H}\, E(f; D_n)$

> **Note**
>
> We're minimising the error using the training data, so we're not necessarily also minimising the error for the overall data distribution[1].
>
> 1. i.e. the testing data ↩



Error functions

Typically generalisation and training error functions can be written in terms of a pointwise loss $l(f; z)$ measuring the error incurred by f on the training example z. $E(f; p_{data}) = \mathbb{E}_{z \sim p_{data}}[l(f; z)]$[2] $E(f; D_n) = \frac{1}{n} \sum_{i=1}^{n} l(f; z_i)$[3]

Example

Objective:

$$f^{\star}_{H_M}(D_n) \in arg\,min_{f \in H_M} E(f; D_n)$$

equivalent to $f_{w^\star}$ where

$$w^\star \in arg\,min_{w \in \mathbb{R}^M} \frac{1}{n} \sum_{i=1}^{n} [f_w(x_i) - y_i]^2$$

[1]

---

1. requires solving a linear system of equations ↩

Learning #algorithms will search for the optimal value, but may end up at a different result as they are susceptible to finding local minima. This can lead to #underfitting or #overfitting.

#Underfitting

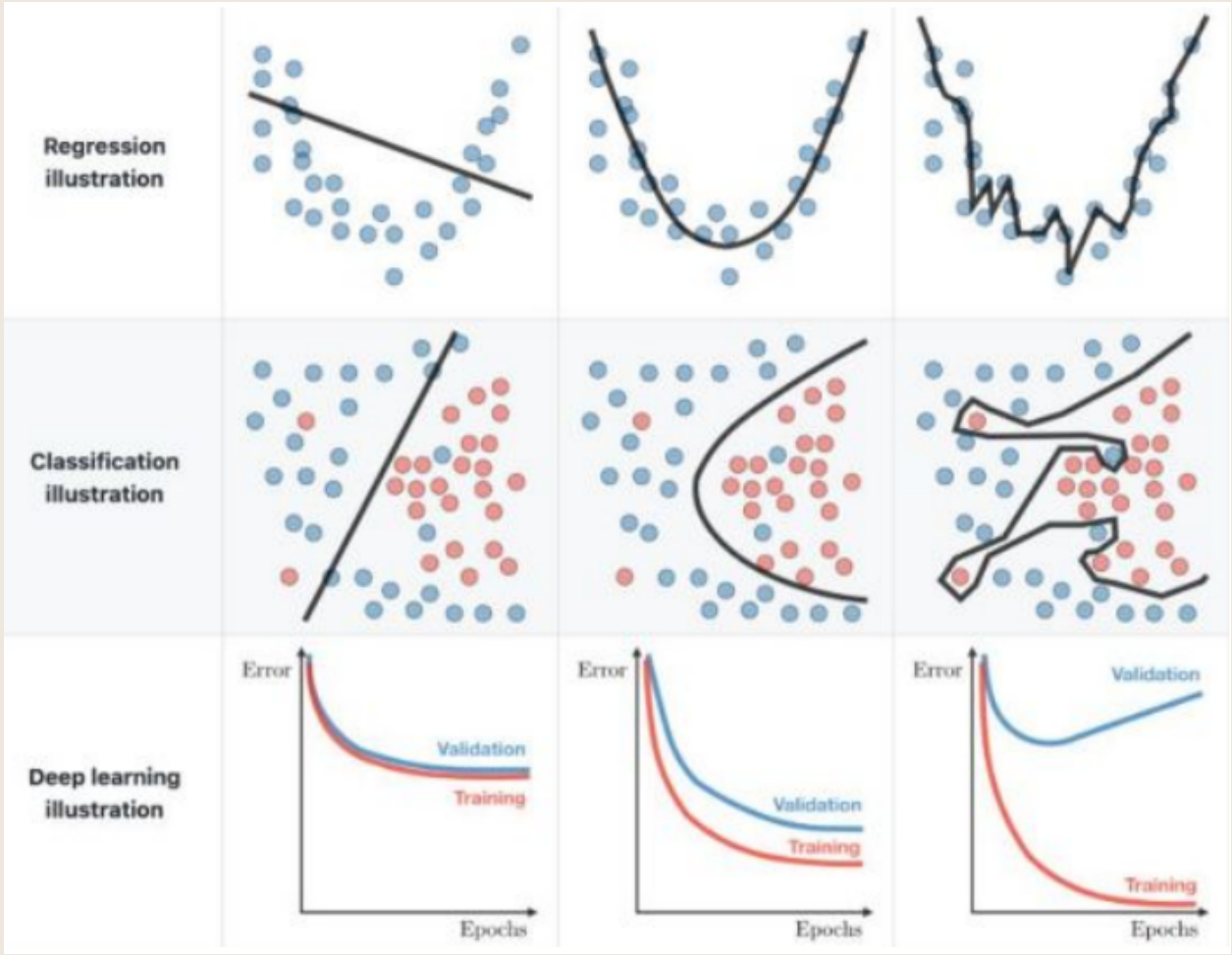
Underfitting may occur when the model hasn't had enough experience with the dataset to properly minimise error and allow effective generalisation. In this case, error on the training data would be roughly the same as the error on any other relevant data, but this error value would render the model practically unusable in any real-world scenario.

#Overfitting

Overfitting occurs when the model adapts to the training data to the point where it is only good for that data, as if it had learnt it "off-by-heart". This would lead to extremely low error on the training data, but extremely high error on any other data due to a lack of generalisability.

Example

In the below image underfitting is shown in the left-hand column, and overfitting is shown in the right-hand column. The middle column is a representation of what we may hope for in a near-perfect scenario.



Approximation Error

It is likely that some error will be induced by the choice of hypothesis space, as the random nature of its selection may exclude the absolute minimum of the function.

Irreducible Error

It's worth noting that it's impossible to reduce error to zero due to randomness or variability in the system we're trying to predict. These are unobservable deterministic factors, but they could be interpreted as inherent variability.

Improving #Generalisation

- Avoid obtaining minimum training error
- Reduce model capacity
- Change the objective with a regularisation term
- Inject noise into the learning #algorithm

- Stop the learning #algorithm before convergence
- Increase the amount of data[4]
- Add more training samples
- Augment the training set with transformations[5]
- Combine predictions from multiple, uncorrelated models[6]

#Regularisation

Definition

Modification of the training error function with a term $\Omega(f)$ that typically penalises complex solutions.

$$E_{reg}(f; D_n) = E(f; D_n) + \lambda_n \Omega(f)$$

[1]

---

1. λ_n is a trade-off parameter ↩

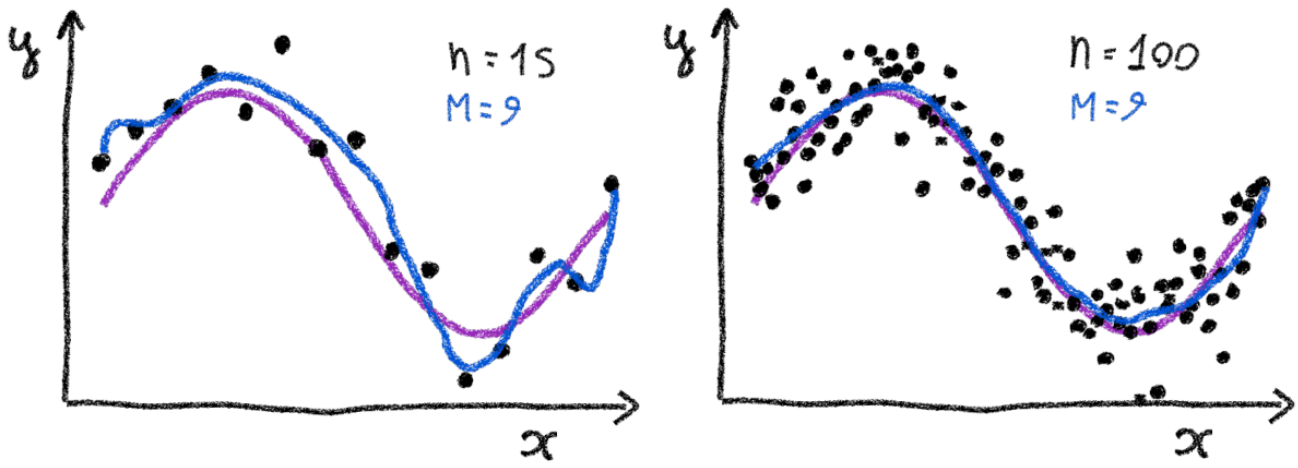
Example

We can regularise by penalising polynomials with large coefficients

$$E_{reg}(f_w; D_n) = \frac{1}{n} \sum_{i=1}^{n} [f_w(x_i) - y_i]^2 + \frac{\lambda}{n} ||w||^2$$

More data

Increasing the amount of (good) data we feed our #algorithm can only improve its performance. In fact, as seen in the example, it can be a good way of forcing our model to fit the curve better even if the degree of the curve we're creating is suboptimal.



Formally, $E(f; D_n) \to E(f; p_{data})$ as $n \to \infty$, that is "as the amount of data increases, the error on the training set tends towards the error the model would get on the actual data".

#Assumptions

Some machine learning approaches make strong #assumptions, which if correct can lead to better performance, but if incorrect could lead to complete failure of the model. Other approaches don't take into account any kind of #assumption, meaning they can learn from more varied data but are more prone to overfitting and generally require more training data.

Knowing the model beforehand can drastically improve not only the learning speed, but also the required number of examples to attain an acceptable level of accuracy. It's important to ensure the correctness of any #assumptions made, however, as incorrect assumptions could have the exact inverse effect.

#Bias

#Bias is a measure of how strong the assumptions on a model are. Low-bias classifiers (such as #K-NN or decision trees) make minimal/no #assumptions about the data, whereas high-bias classifiers make strong #assumptions about the data.

#Bootstrap Aggregation

> #Bootstrap Sampling
>
> Given a set D containing N training examples, create D' by drawing N samples at random with replacement from D.

#Bagging

- Create k #bootstrap datasets D_i

- Train distinct classifier on each D_i
- Classify new example by majority vote/average

---

1. It's important that we use the same distribution to generate all of these sets, otherwise we risk testing the model on data it's seen a disproportionately small or large amount of compared to its importance in the real world. ↩
2. pointwise loss based on a training example similar to the original data distribution ↩
3. average pointwise loss using the training data ↩
4. painful but worth it ↩
5. rotated/cropped versions of images already in the training set for an object recognition model ↩
6. ensembling ↩