# Introduction to Machine Learning and Kernel Machines

Michael Fairbank
m.fairbank@essex.ac.uk

School of Computer Science and Electronic Engineering
University of Essex (UK)

Essex Big Data & Analytics Summer School 2022
26 July 2022

# Welcome

About me:

- Senior Lecturer at Essex University
- Main research area is Machine Learning:
  - Neural Networks (Deep Learning)
  - Reinforcement Learning.
- Delivering courses in Week 2 on TensorFlow and Recurrent Neural Networks

About this course:

- These slides were written by Prof. Luca Citi (Essex University).
- He is on paternity leave this year.

# Welcome

About today's course:

- A gentle introduction to Machine Learning.
  - Learn about what methods are available
  - Learn about what issues are with overfitting and underfitting.
  - Introduce to regression, linear classifiers and SVMs (Kernel) machines
- Introduce using Scikit-Learn (Sklearn) and Pandas
  - How to prepare data using pandas.
  - How to create a classifier or regessor using sklearn.
  - How to validate how well your system works (metrics)

# Outline

# Algorithms VS Machine learning

- We can often solve a problem on a computer using an algorithm.

**Example: Find the mean of a set of numbers (e.g., grades)**

```
sum = 0.;
for (i=1; i<=N; ++i)
    sum += grade[i];
mean = sum / N;
```

**Example: Compute the parity of a binary string**

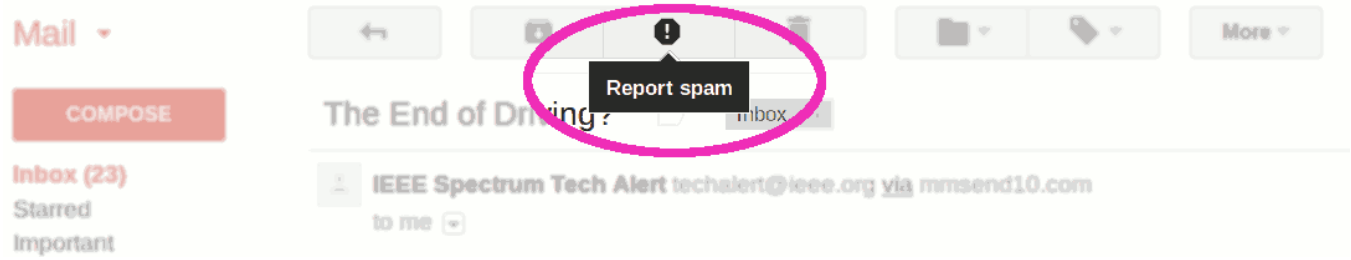
```
parity = 1;
for (i=1; i<= N; ++i)
    parity = parity XOR bit[i];
```

# Algorithms VS Machine learning

- For some problems we don't have an algorithm
  - Example: Recognize a face
  - Example: Perform Optical Character Recognition (OCR)
  - Example: Tell spam from legitimate emails
  - Example: Group products frequently bought together
- Why?
  - Humans are often very good at performing a task unconsciously but are unable to explain how. If we are not able to explain our expertise, we cannot write a computer program!
  - The task may change with time or situation (e.g., blocking spam is an arms race); we need a flexible framework to perform the task.

# Algorithms VS Machine learning

- We would like the computer (machine) to extract automatically the algorithm for these tasks.
- How?
  - We collect a set of examples



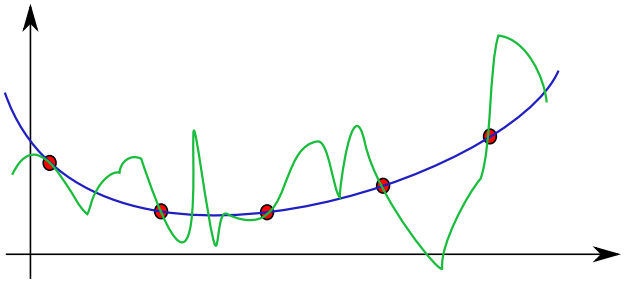

  - We build computer algorithms able to learn the task-related algorithm from the data

    What we lack in knowledge, we make up for using data.

  - On the other hand, data by itself is not enough:
    we still need the expert's knowledge to design the ML system and provide the so-called prior knowledge.

# Importance of prior knowledge in ML

- Prior knowledge: information about the problem available in addition to the training data
- Why? without prior knowledge, learning a ML model from finite (and incomplete) data is an ill-posed problem (no unique solution)



Without prior knowledge both solutions (and infinitely many more) are possible and equally "legitimate"

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

Should ML learn OR or XOR? Without prior knowledge, both are equally "legitimate" solutions

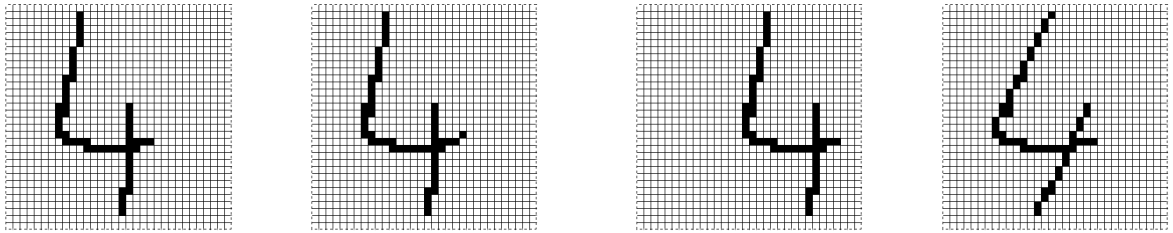
- The no free lunch theorem states that, if we average over all possible problems, no learning algorithm is better than another one (not even random guessing) on unseen data

# Importance of prior knowledge in ML

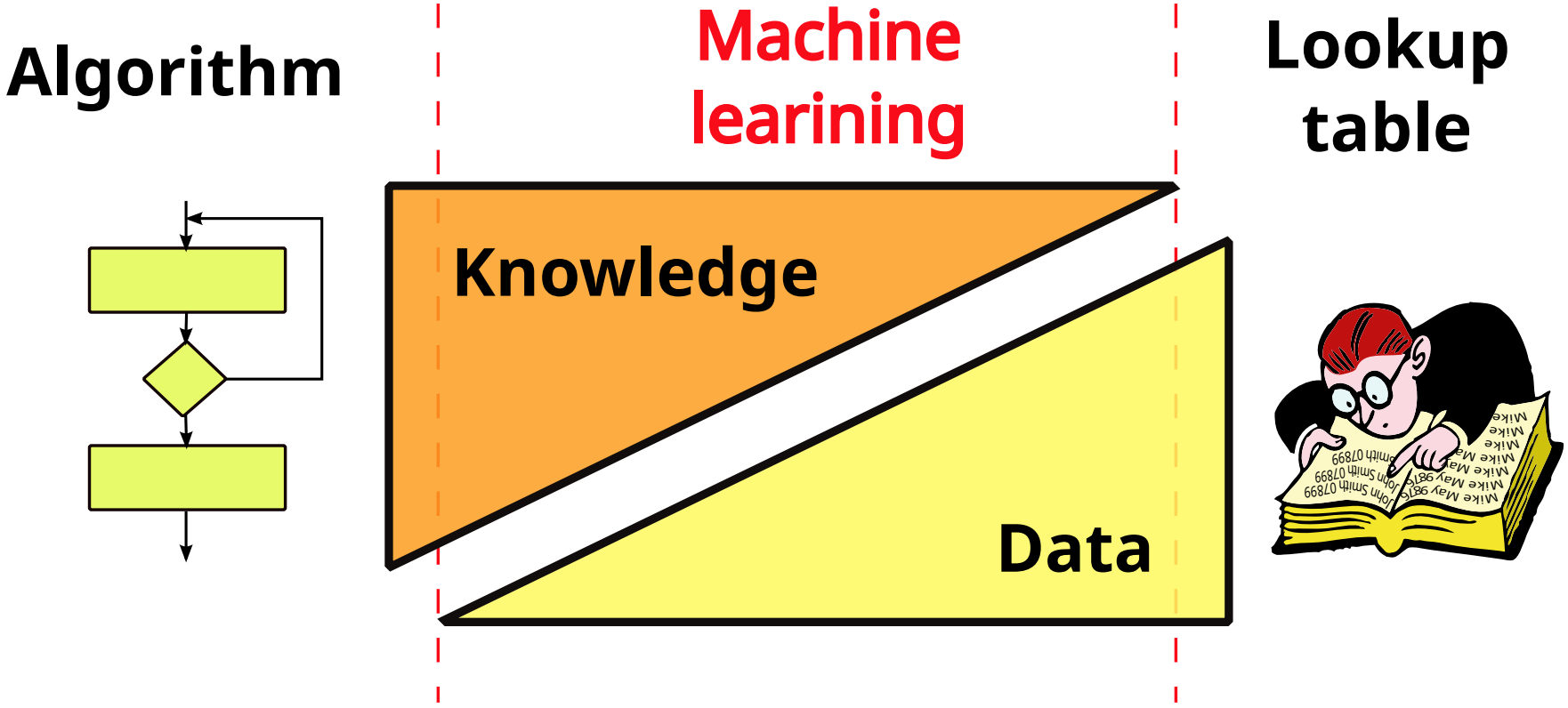
- Listing all possible inputs is often impossible or unfeasible:
  - Even a simple $\mathbb{R} \to \mathbb{R}$ function would require a table with an (uncountable) infinite number of entries.
  - OCR on a small (32x32) B/W image would require $2^{32 \cdot 32} \sim 1.8 \cdot 10^{308}$ labelled sample images
- By exploiting prior information about a specific problem one can improve the performance by favouring solutions that are known to perform better in similar situations. For example:
  - General smoothness assumption, often small changes in the input are unlikely to cause big changes in the output

    (e.g., in OCR a single pixel is unlikely to change a 4 to a 6)
  - Transformation-invariance, often the output can be assumed to be invariant to some transformations of the input pattern

    (e.g., shear and translation in OCR, pitch and speed in speech recognition)



# Machine Learning: data & prior knowledge



# Assumptions for using Machine Learning

- We believe that there is a process that explains the data we observe.
- We try to learn a general model from limited amount of data and prior knowledge.
- Even if we may not be able to identify the process completely, we might be able to construct a good and useful approximation.
- The learned model should be able to make sufficiently-accurate predictions in previously-unseen cases

Learn a simple model with good GENERALIZATION abilities

# Generalization in Machine Learning

- Generalization is a key concept in ML
  - It is the ability of our ML model to make sufficiently-accurate predictions in previously-unseen cases.
- Often we train our ML models to learn a set of data but it merely memorises it and does not generalise well:
  - This is called Overfitting
- Often the opposite happens: we train our ML models to learn a set of data but it can't recall all of the training examples with sufficient accuracy:
  - This is called Underfitting

# Key Elements of a Machine Learner

- Learning is often defined as improving performance at some task.

  So there must be:
  - a task
  - an associated performance (+) or loss (-) measure
- Extracting information / knowledge (learning) from data.

  So there must be:
  - a set of examples $\mathcal{X}$ (labelled or unlabelled)
  - a representation format for examples
- Using prior knowledge ($\rightarrow$ inductive bias).

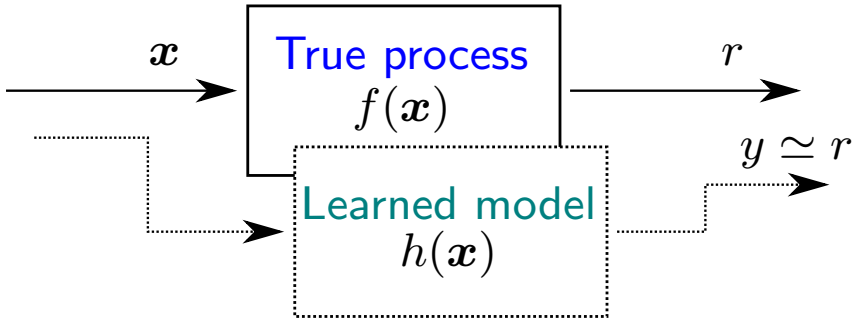
  So there must be:
  - a set of (smoothness, invariance, ...) assumptions about the domain in which learning takes place
  - a family of possible models from which the one produced is selected (often called hypothesis class $\mathcal{H}$)

Learning can then be viewed as a search of the hypothesis class for the model $h \in \mathcal{H}$ maximizing the performance measure.

# Outline

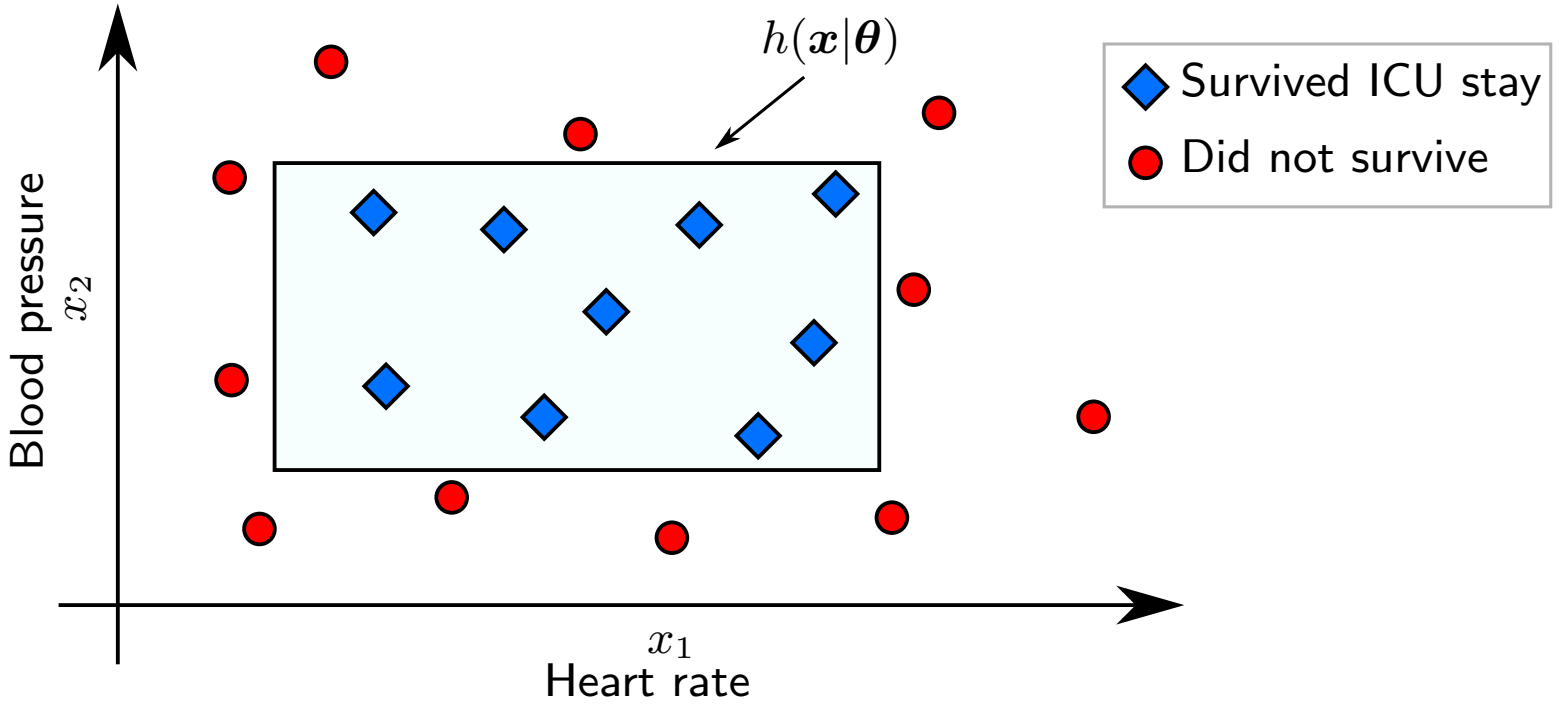
# Supervised learning

- A teacher provides a label/target for each example in the training set: $\mathcal{X} = \{(\boldsymbol{x}^t, r^t)\}_{t=1}^N$.
- By using the labelled examples, we learn a model which provides outputs $(y)$ that are close enough to the observed $(r)$ ones on the training set.




- We expect our model outputs to be close to the true process outputs on unseen data.

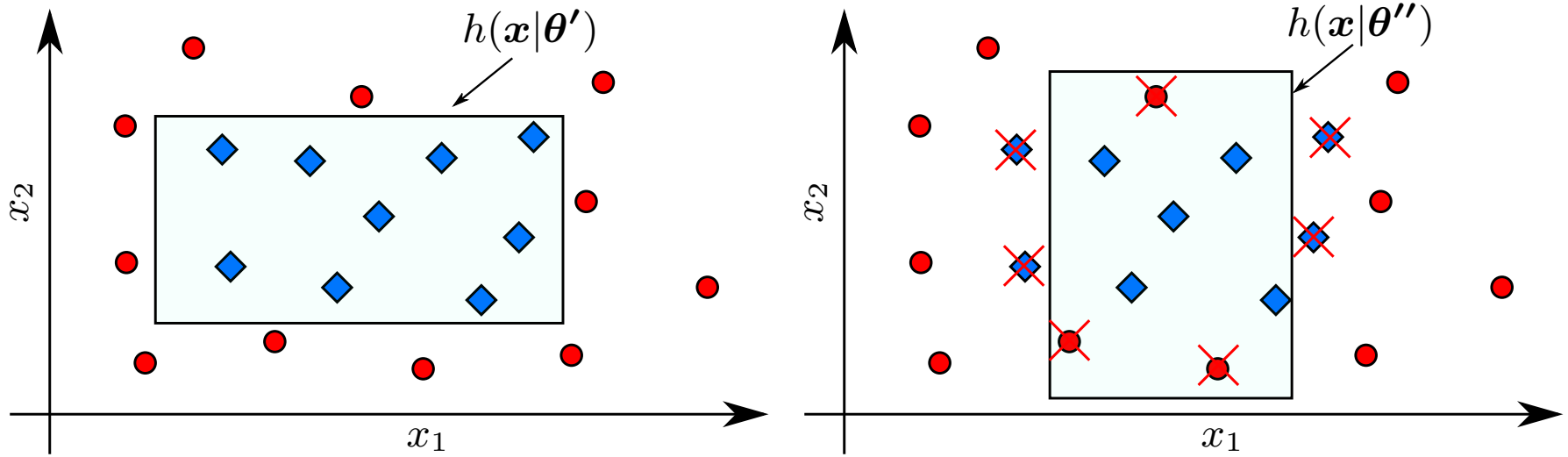
# Classification



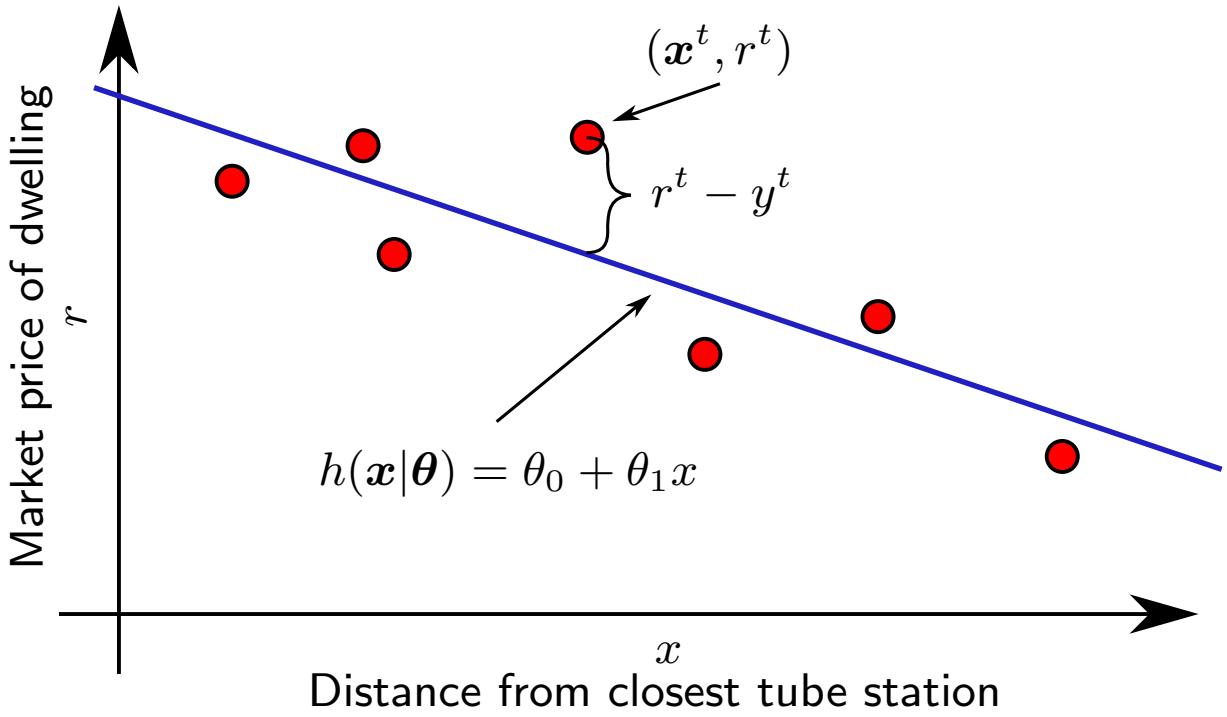

- Training set (set of examples that are used to fit the model): $\mathcal{X} = \{(\boldsymbol{x}^t, r^t)\}_{t=1}^N$, for example with $\boldsymbol{x}^t \in \mathbb{R}^2$ and $r^t \in \{0, 1\}$
- Hypothesis class; e.g., set of rectangles defined by $\mathcal{H} = \{h(\boldsymbol{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta} \in \mathbb{R}^4}$ with $h(\boldsymbol{x}|\boldsymbol{\theta}) = \begin{cases} 1 & \text{if } \theta_1 < x_1 < \theta_2 \text{ and } \theta_3 < x_2 < \theta_4 \\ 0 & \text{otherwise} \end{cases}$

# Classification

- Loss function (what we pay for the error between observed data and model output); e.g., 0-1 loss: $L(r, y) = \begin{cases} 1 & \text{if } r \neq y \\ 0 & \text{otherwise} \end{cases}$
- Empirical risk minimization; find optimal $\boldsymbol{\theta}$: $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{t=1}^{N} L(r^t, h(\boldsymbol{x}^t|\boldsymbol{\theta}))$



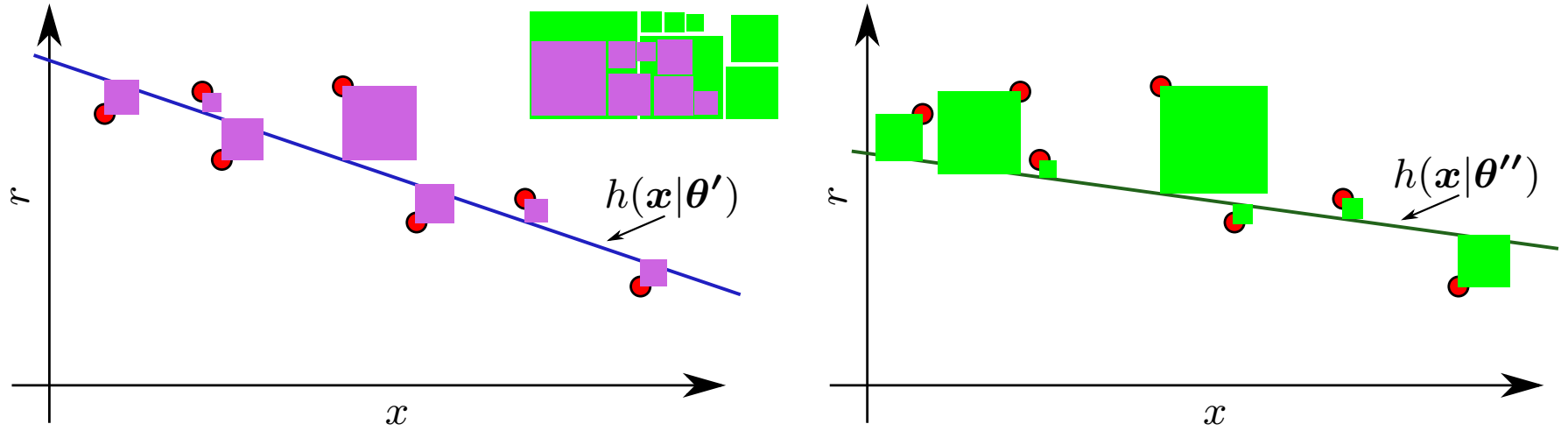
# Regression




- Training set (set of examples that are used to fit the model): $\mathcal{X} = \{(\boldsymbol{x}^t, r^t)\}_{t=1}^N$, for example with $\boldsymbol{x}^t \in \mathbb{R}$ and $r^t \in \mathbb{R}$
- Hypothesis class; e.g., set of linear functions $\mathcal{H} = \{h(\boldsymbol{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta} \in \mathbb{R}^2}$ with $h(\boldsymbol{x}|\boldsymbol{\theta}) = \theta_0 + \theta_1 x$

# Regression

- Loss function (what we pay for the error between observed data and model output); e.g., quadratic loss: $L(r, y) = (r - y)^2$
- Empirical risk minimization; find optimal $\boldsymbol{\theta}$: $\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{t=1}^{N} L(r^t, h(\boldsymbol{x}^t|\boldsymbol{\theta}))$

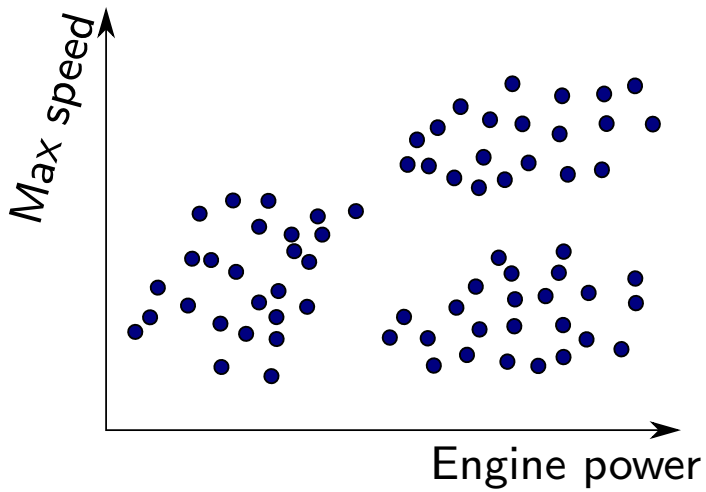


# Unsupervised learning

- We don't have labels for the examples in the training set: $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$.
- The system forms clusters, finds patterns and hidden structure in unlabeled data.
- There is no explicit error or reward to evaluate a potential solution.

# Clustering

Given a set of (unlabelled) data points...




form groups (clusters) so that:

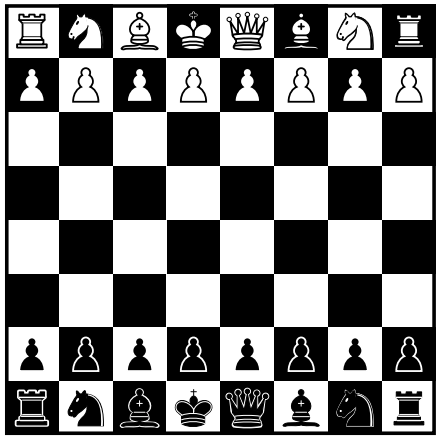
- points within each cluster are similar to each other
- points from different clusters are different

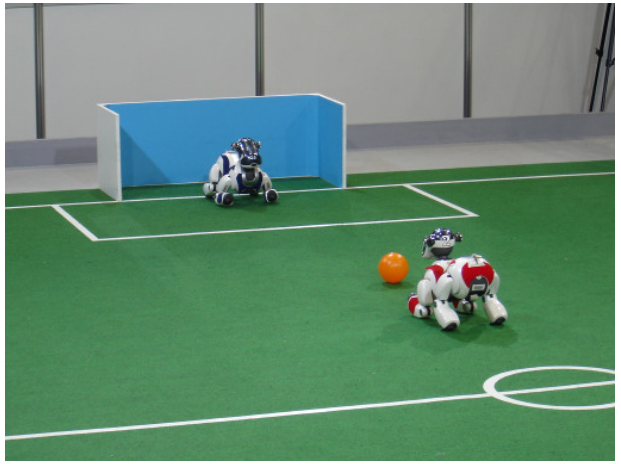
This requires the definition of a meaningful similarity (or distance) measure.

Examples: MRI image segmentation, customer segmentation, clustering documents, clustering DNA sequences, ...

# Reinforcement learning

Think of a machine that learns to play the following games:





- We get feedback/reward only after a complete sequence of actions/decisions
- There is no explicit (not even retrospective) scoring of individual actions/decisions
- The goodness of each action depends on the actions that follow (sometimes on the opponent's/environment's response to our action)

# References

**Core material reading:**[1]

Alpaydin 1.1, 1.2.1, 1.2.2, 1.2.3, 1.2.4, 1.2.5
2.1(~~Version space~~), 2.5, 2.6, 2.7, 2.8

**Further interesting reading material:**

Wikipedia:

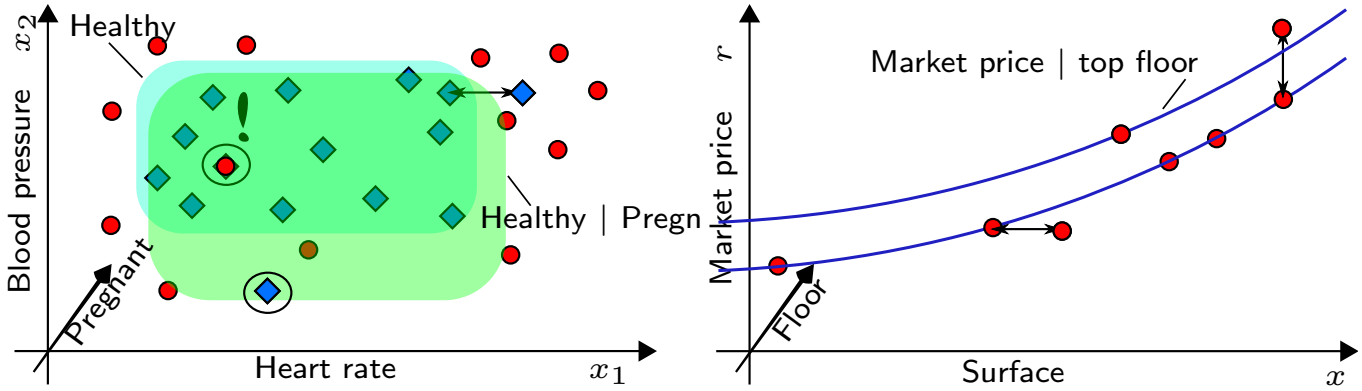
`https://en.wikipedia.org/wiki/Machine_learning`
`https://en.wikipedia.org/wiki/Prior_knowledge_for_pattern_recognition`
`https://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization`

[1] Textbook: Introduction to Machine Learning, Ethem Alpaydin, 2020

# Outline

# Noise in the data

- In machine learning, data typically contain errors
- Possible causes:
  - measurement errors, human mistakes and other sources of imprecision in recording the input attributes;
  - measurement errors, errors of expert judgement in classifying training examples and other sources of imprecision affecting the target value (teacher noise);
  - effect of additional attributes that affect the label of an instance but are unavailable to the machine learner (neglected/hidden/unobservable); these can be modelled as source of random error
- We refer to all of these as noise

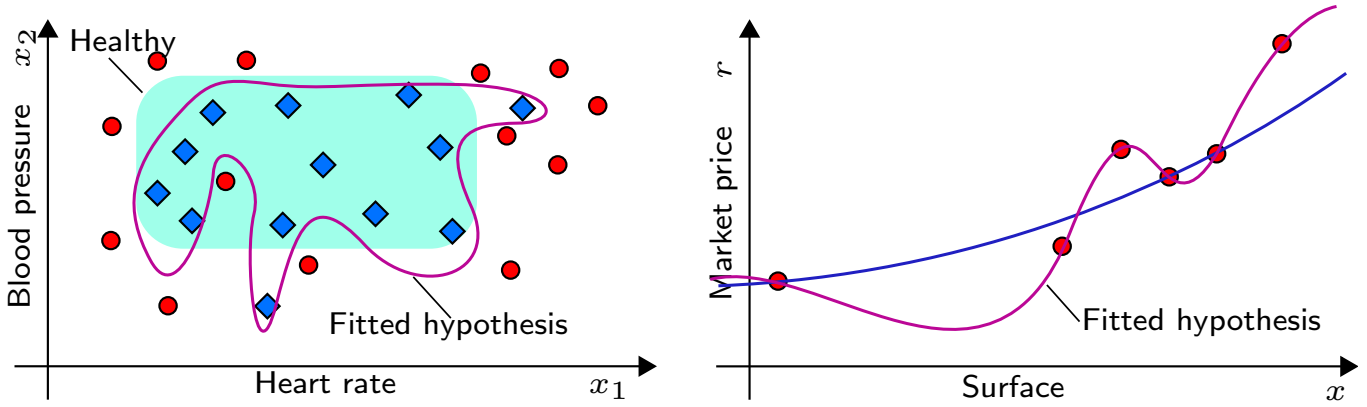


# Consequences of noise

The main consequence of noise in the training data is overfitting that occurs when the fitted hypothesis reflects the random error instead of the underlying relationship between the features and the target.

In turn, overfitting causes:

- Solutions that are more complex than necessary and hard to interpret; e.g., noise-induced growth of DTs
- Solutions that don't generalize, i.e., poor accuracy on new unseen data



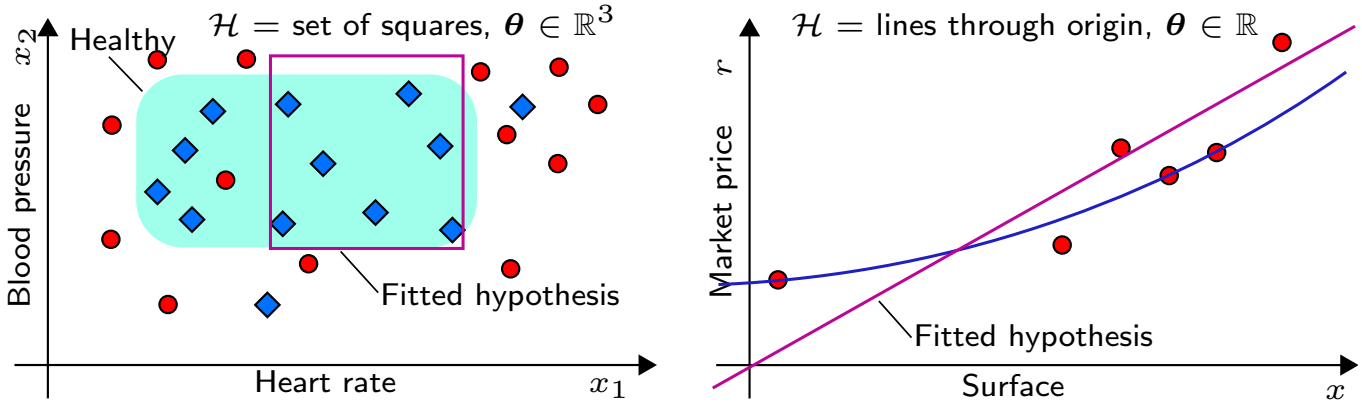
# Alleviate effect of noise

To alleviate these harmful effects of noise, we have to prevent overfitting.

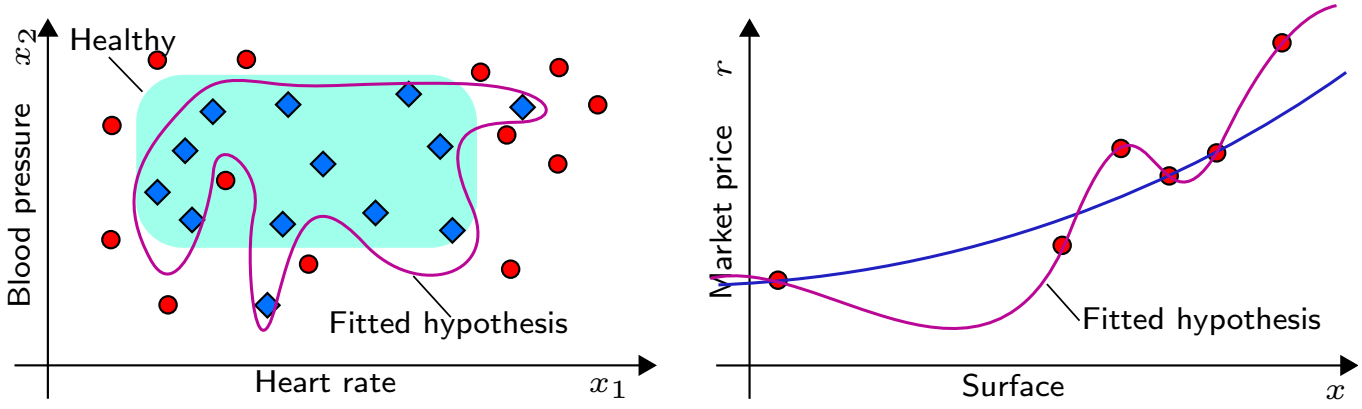
The common approach is to simplify induced hypotheses, e.g.:

- Use Linear Regression instead of non-linear regression
- Using Naïve Bayes instead of full Bayes classifiers
- Pruning decision trees (DTs)
- In general, use models requiring a smaller number of parameters that need be estimated from the data

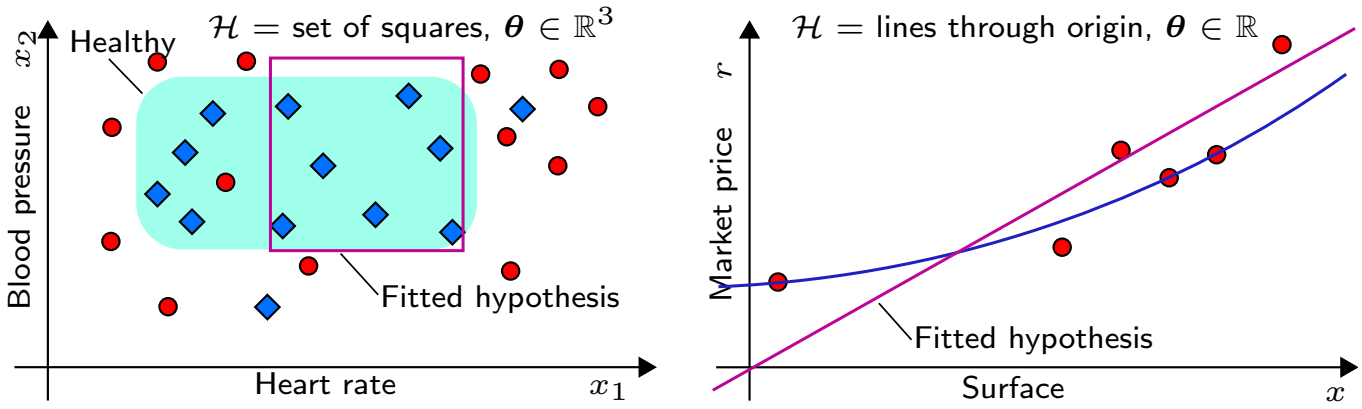
On the other hand, if the hypothesis is too simple, we incur in the opposite type of error: underfitting



# Overfitting VS Underfitting




Q1: How can we make sure we choose the hypothesis of "the right size", not too simple and not too complex?



# Errors in predictions

After analysing possible sources of error in the training data, we are interested in the errors we make when predicting unseen examples.

Possible sources of error:

- Same errors as in the training data (after all, we made the assumption that training data and test data are drawn from the same population)
- Errors due to a model that underfits the training data
- Errors due to a model that overfits the training data

Q2: Can we quantify the errors we make when predicting unseen examples?

In the following, we do this for the case of linear regression but the same considerations hold for any ML algorithm

Nice interactive tutorial on the bias-variance trade-off for classification at: `http://scott.fortmann-roe.com/docs/BiasVariance.html`

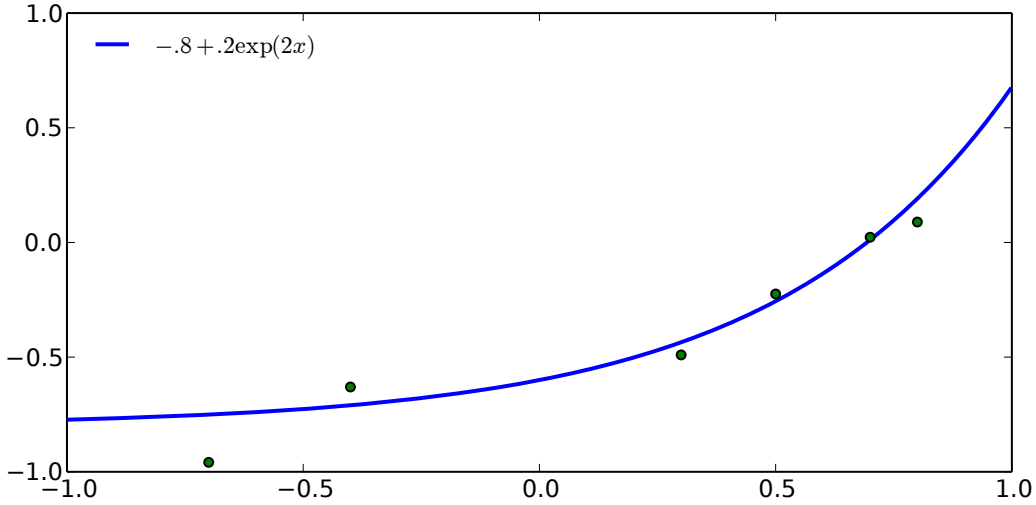
# Example: linear regression

Assumption: all sources of noise on the training data (errors on features, errors on targets, errors due to unobserved features) can be adequately modelled as an additive zero-mean uncorrelated observation noise

| $x$ | $\dots$ | $x_{\mathrm{M}}$ | $r$ |
|---|---|---|---|
| $0.4$ | $\dots$ | $0.8$ | $0.5$ |
| $0.1$ | $\dots$ | $0.5$ | $0.3$ |
| $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| $\boldsymbol{x}^{(t)}$ | | | $r^{(t)} \leftarrow f(\boldsymbol{x}^{(t)}) + n^{(t)}$ |
| $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |

[2]

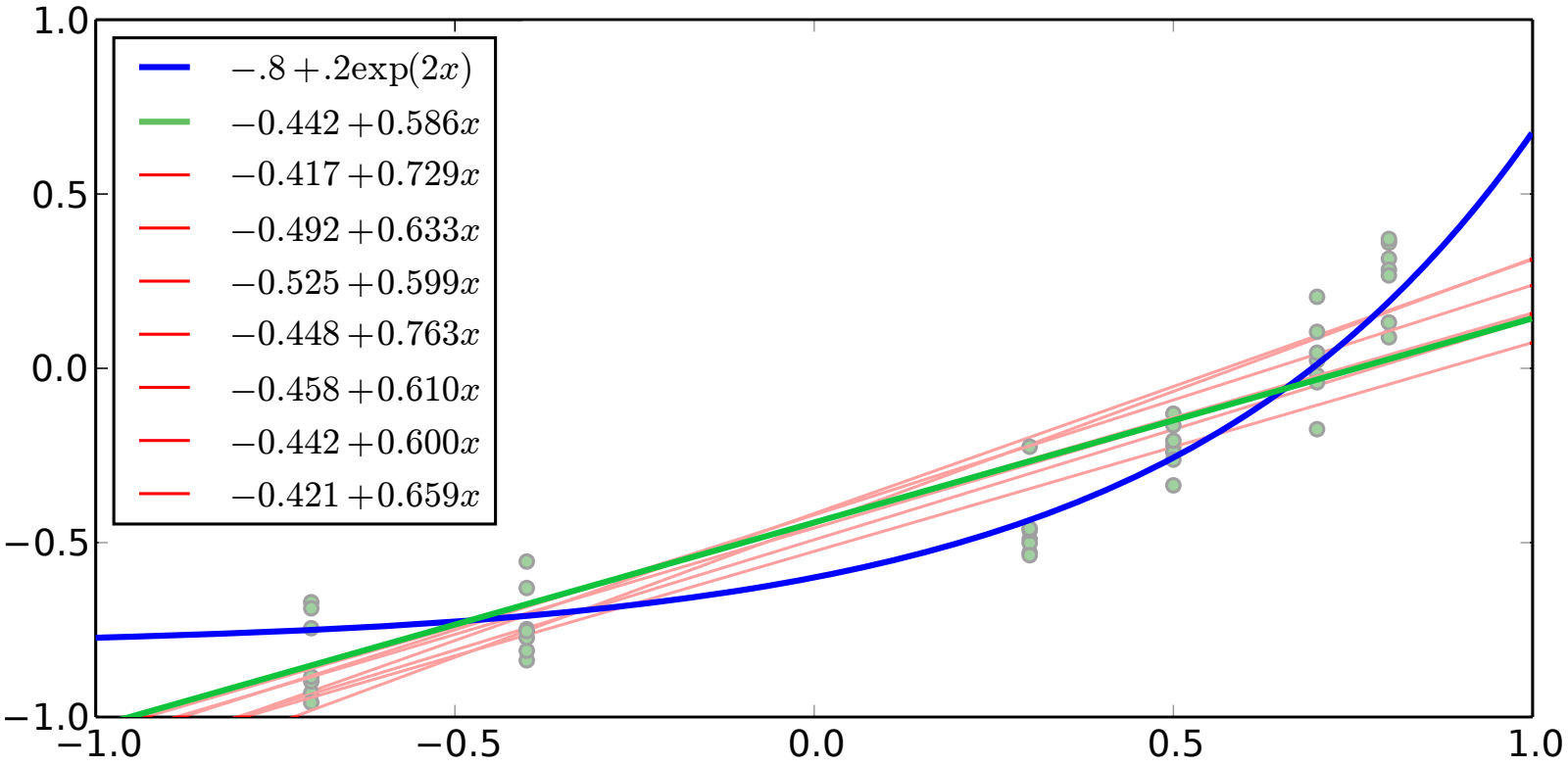
We further simplify by considering a single feature.
Let $f(x) = 0.2\exp(2x) - 0.8$




[2] As in the following we will use the superscript to indicate powers, we use $\boldsymbol{x}^{(t)}$ and $r^{(t)}$ to indicate the features and target of the $t$-th training instance

# Fitting a straight line

Set of linear functions: $\mathcal{H} = \{h(\boldsymbol{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}\in\mathbb{R}^2}$ with $h(\boldsymbol{x}|\boldsymbol{\theta}) = \theta_0 + \theta_1 x$

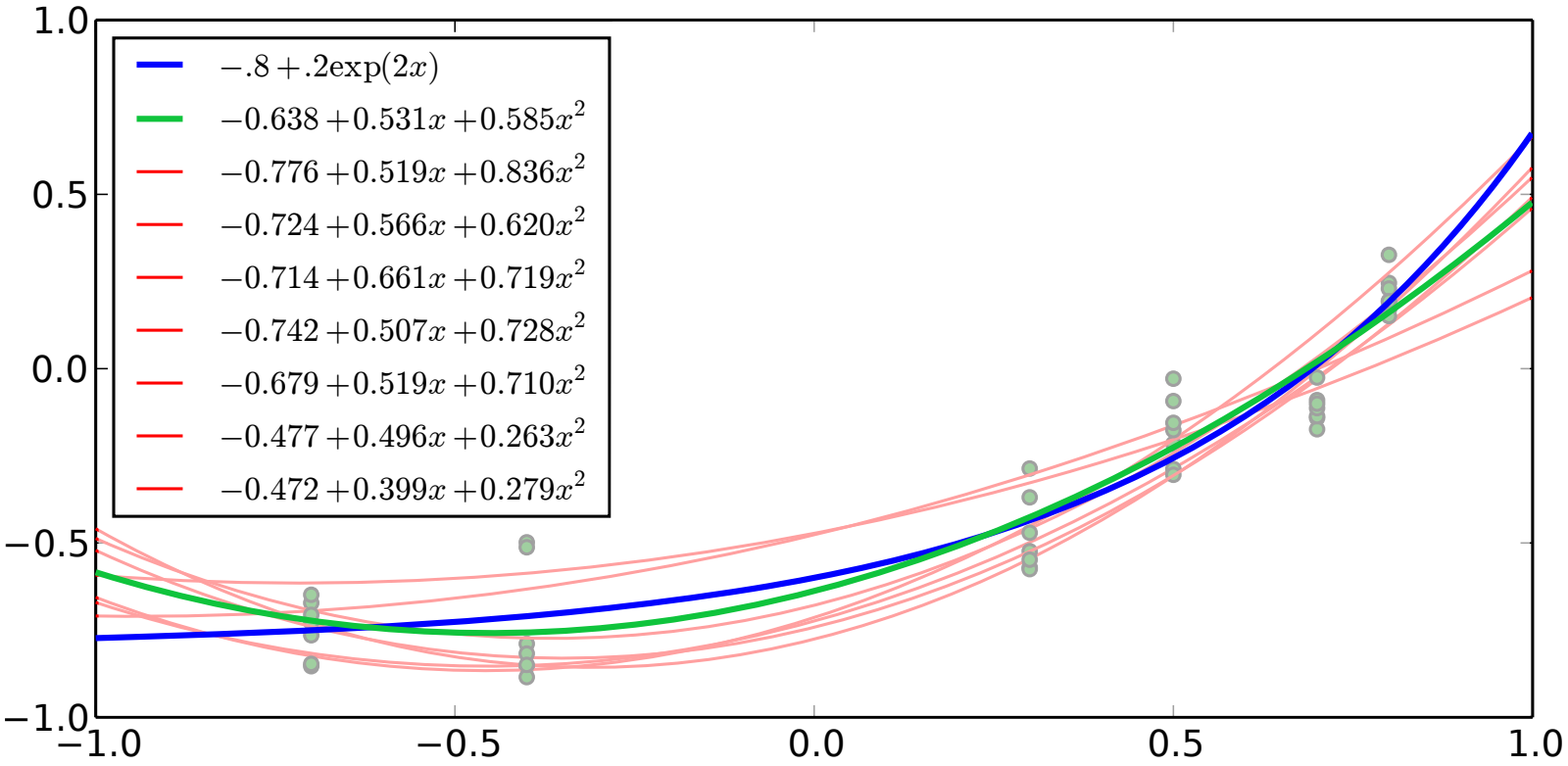



Training using a different training set gives slightly different solutions: overfitting

If we collect (infinitely) many training sets and take the average of all solutions, we still make an error (e.g., $\bar{h}(0) = -0.442 \neq -0.6 = f(0)$): underfitting

# Fitting a quadratic polynomial

Set of quadratic polynomials: $\mathcal{H} = \{h(\boldsymbol{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}\in\mathbb{R}^3}$ with $h(\boldsymbol{x}|\boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2$

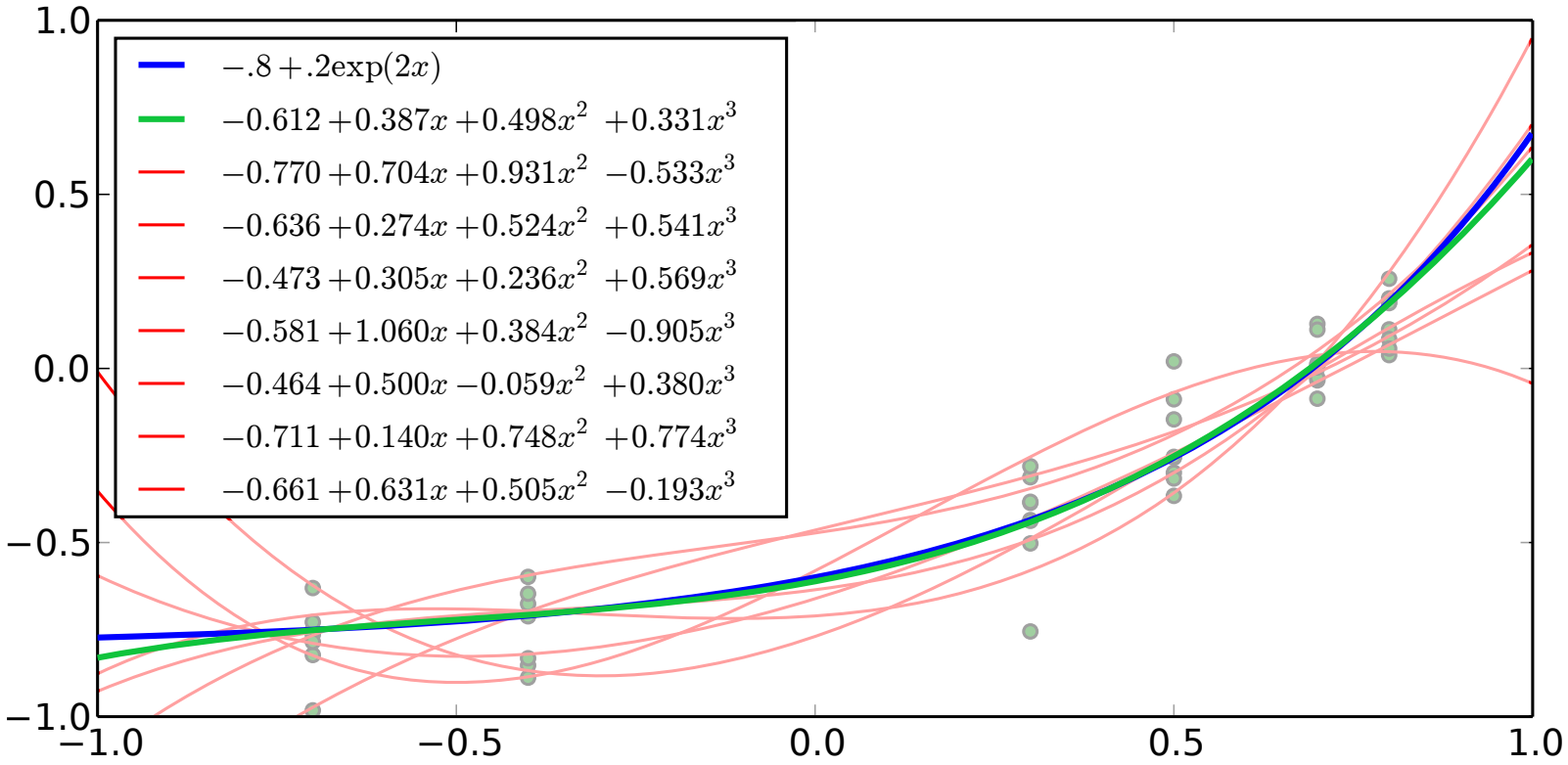



It seems like increasing the degree of the polynomial (i.e. the complexity of the model) makes the problem of overfitting worse

OTOH, there is less underfitting now: e.g., $\bar{h}(0)$ is closer to $f(0)$ than before

# Fitting a cubic polynomial

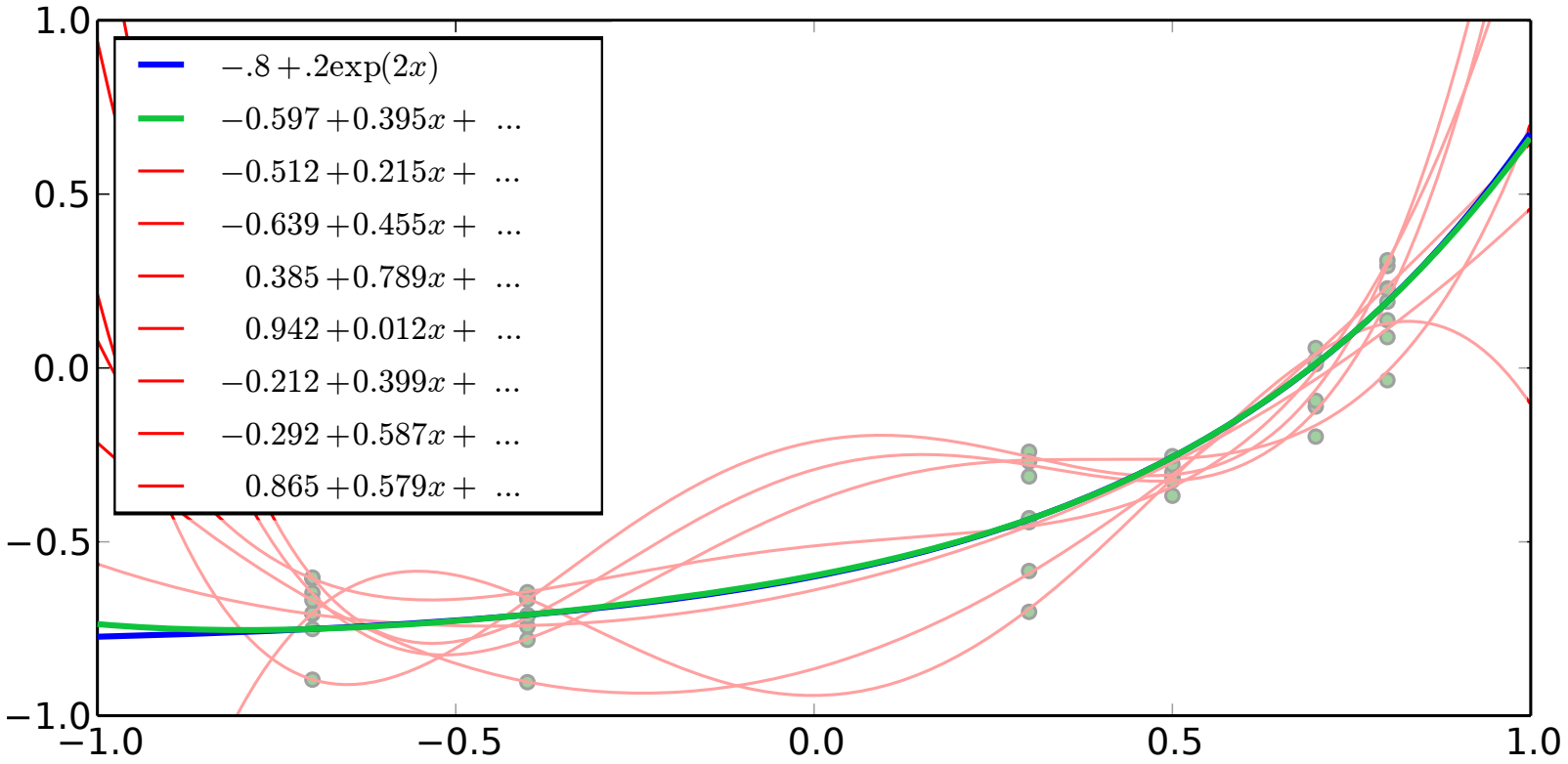
Set of cubic polynomials: $\mathcal{H} = \{h(\boldsymbol{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}\in\mathbb{R}^4}$ with $h(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{i=0}^{3} \theta_i x^i$




... also in this case ...

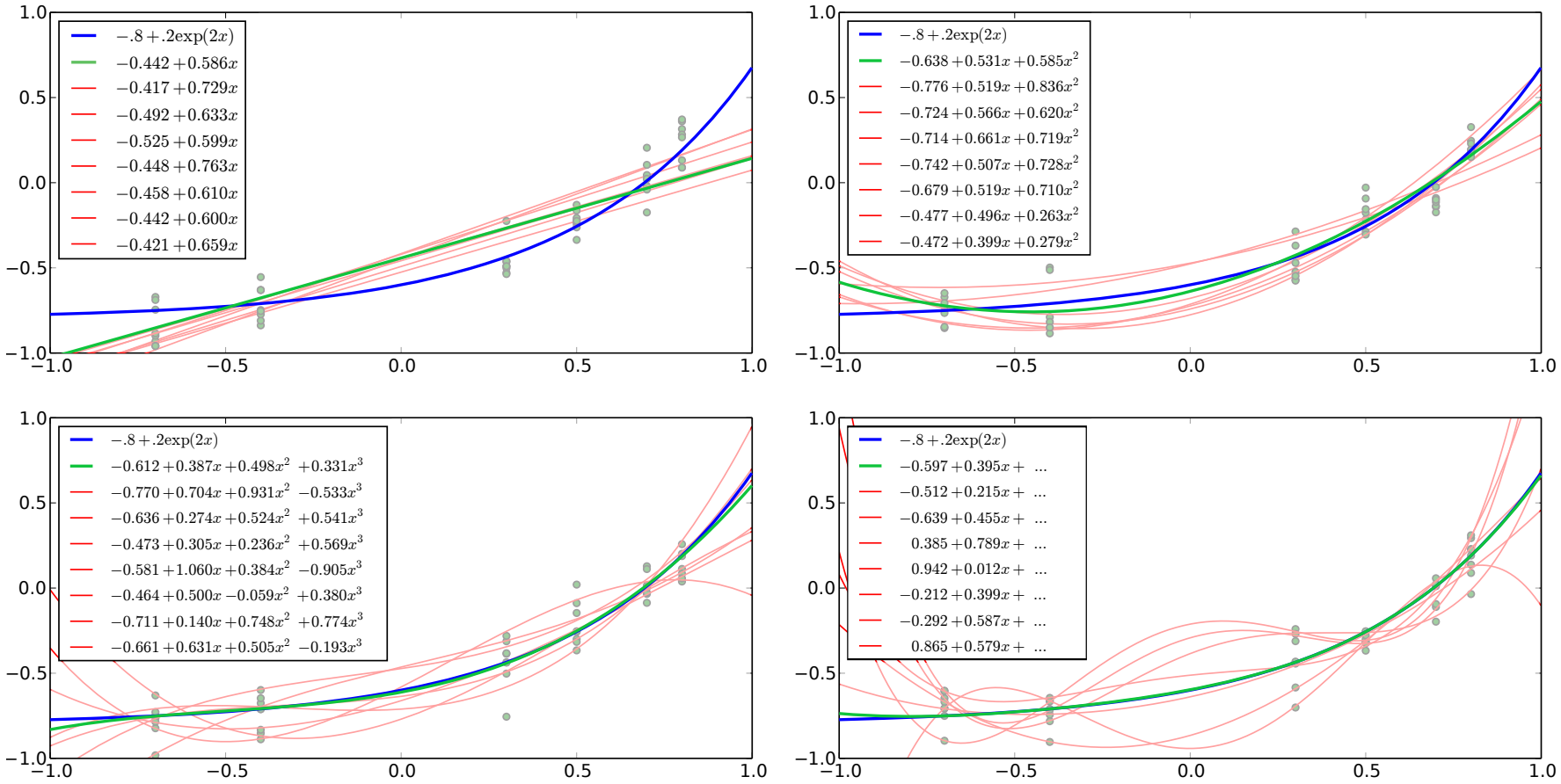
# Fitting a 4th order polynomial

Set of 4th order polynomials: $\mathcal{H} = \{h(\boldsymbol{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}\in\mathbb{R}^5}$ with $h(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{i=0}^{4} \theta_i x^i$




... and in this case.

# General rule




In general, increasing complexity reduces bias (i.e. the error related to underfitting) and increases variance (i.e. the error related to overfitting).

# Expected squared prediction error

Let's suppose, for simplicity, that we are interested in the prediction error for a fixed value of $x$: $x^{\text{test}} = \tilde{x}$ (e.g., error in the prediction of the market price of a 100 m$^2$ apartment).[3]

We repeat the following experiment:

- Randomly sample a training set $\mathcal{X}$
- Fit a model $h(x)$
- Randomly draw a test point $x^{\text{test}} = \tilde{x}$ from the dataset, and record its target $r^{\text{test}}$
- Compute the squared prediction error $(r^{\text{test}} - h(x^{\text{test}}))^2$

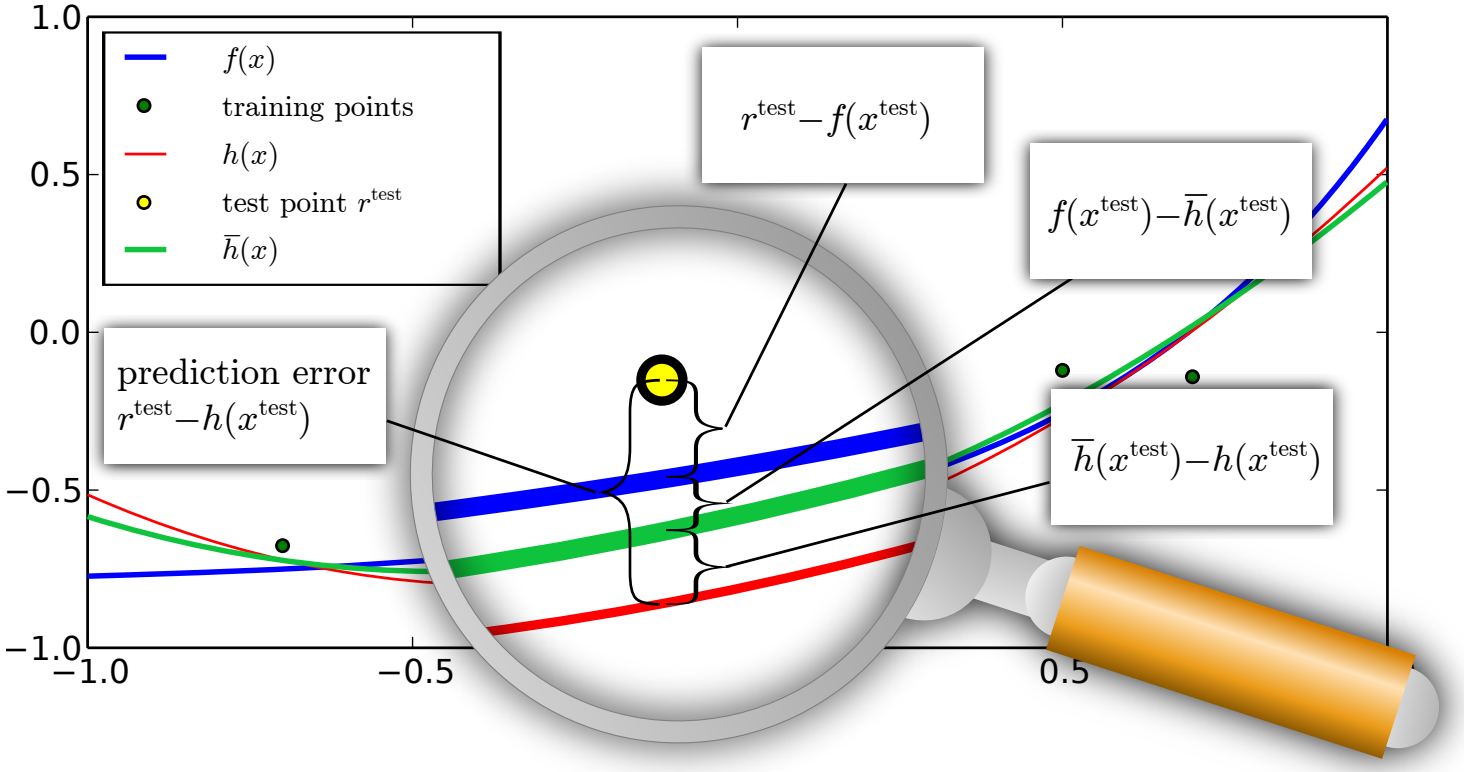
What is the squared prediction error that we should expect on average?

We can define it using the "expected value" operator.

[3] Assuming a fixed value of $x^{\text{test}}$ makes the working easier because we can work with the probability distribution of $r^{\text{test}}$ (given $x^{test}$) rather than the joint distribution of $(x^{\text{test}}, r^{\text{test}})$ but the result we will obtain is general.

# Decomposing the prediction error

From a training set..., we fit a model $h(x)$. If we are given a new test data point $(x^{\text{test}}, r^{\text{test}})$, what prediction error $r^{\text{test}} - h(x^{\text{test}})$ should we expect?

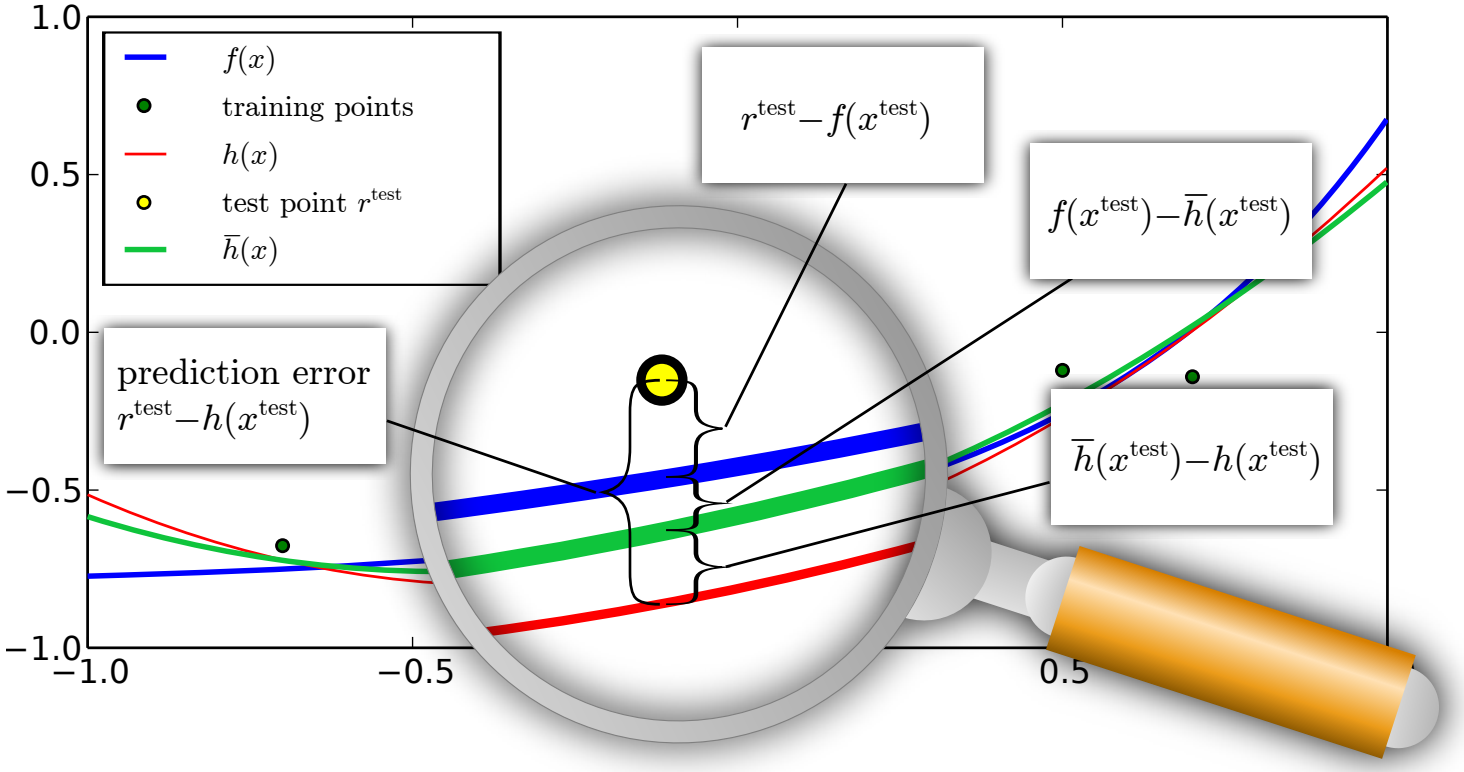



Considering the (unknown) functions $f(x)$ and $\bar{h}(x)$..., the prediction error is

$$r^{\text{test}} - h(x^{\text{test}}) = [r^{\text{test}} - f(x^{\text{test}})] + [f(x^{\text{test}}) - \bar{h}(x^{\text{test}})] + [\bar{h}(x^{\text{test}}) - h(x^{\text{test}})]$$

# Decomposing the prediction error

From a training set..., we fit a model $h(x)$. If we are given a new test data point $(x^{\text{test}}, r^{\text{test}})$, what prediction error $r^{\text{test}} - h(x^{\text{test}})$ should we expect?




Considering the (unknown) functions $f(x)$ and $\bar{h}(x)$..., the prediction error is

$$r^{\text{test}} - h(x^{\text{test}}) = [r^{\text{test}} - f(x^{\text{test}})] + [f(x^{\text{test}}) - \bar{h}(x^{\text{test}})] + [\bar{h}(x^{\text{test}}) - h(x^{\text{test}})]$$

# Decomposing the squared prediction error (maths as ref.)

After renaming $r^{\text{test}} = r$ (for simplicity of notation) and using $x^{\text{test}} = \tilde{x}$, the expected squared prediction error can be written as:[4]

$$\mathbb{E}\left[\{r - h(\tilde{x})\}^2\right] = \mathbb{E}\left[\left\{[r - f(\tilde{x})] + [f(\tilde{x}) - \bar{h}(\tilde{x})] + [\bar{h}(\tilde{x}) - h(\tilde{x})]\right\}^2\right] =$$

$$\mathbb{E}\left[\{r - f(\tilde{x})\}^2\right] + \quad \leftarrow \text{noise contribution}$$

$$\mathbb{E}\left[\{f(\tilde{x}) - \bar{h}(\tilde{x})\}^2\right] + \quad \leftarrow \text{bias squared}$$

$$\mathbb{E}\left[\{\bar{h}(\tilde{x}) - h(\tilde{x})\}^2\right] + \quad \leftarrow \text{variance of the model}$$

$$2\,\mathbb{E}\left[\{r - f(\tilde{x})\}\{f(\tilde{x}) - \bar{h}(\tilde{x})\}\right] + \quad \leftarrow = 2\,\mathbb{E}\left[...\right]\{...\} = \{f(\tilde{x}) - f(\tilde{x})\}\{...\} = 0$$

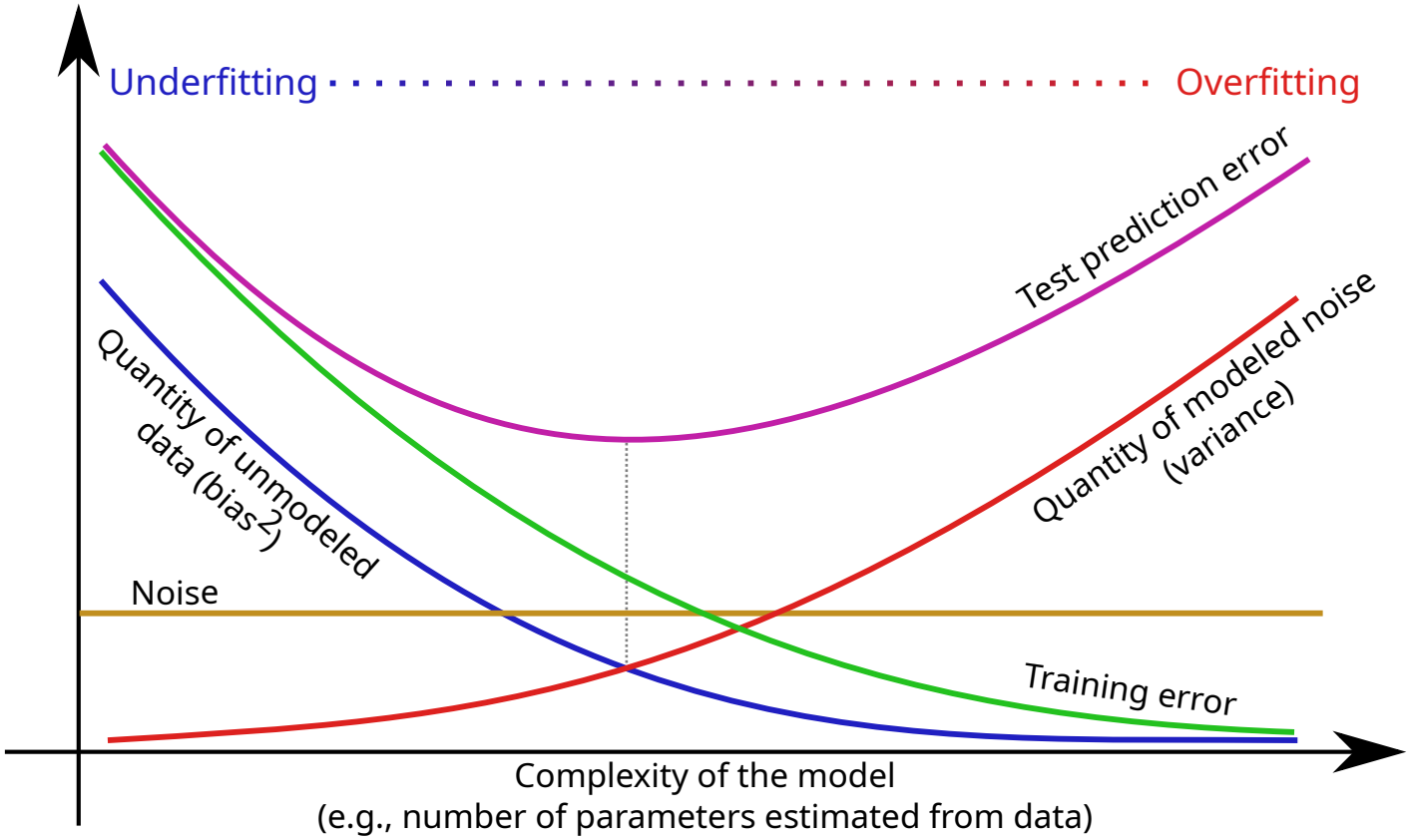
$$2\,\mathbb{E}\left[\{f(\tilde{x}) - \bar{h}(\tilde{x})\}\{\bar{h}(\tilde{x}) - h(\tilde{x})\}\right] + \quad \leftarrow = 2\{...\}\,\mathbb{E}\left[...\right] = 2\{...\}\{\bar{h}(\tilde{x}) - \bar{h}(\tilde{x})\} = 0$$

$$2\,\mathbb{E}\left[\{\bar{h}(\tilde{x}) - h(\tilde{x})\}\{r - f(\tilde{x})\}\right] = \quad \leftarrow = 2\,\mathbb{E}\left[...\right]\mathbb{E}\left[...\right] = 0 \cdot 0 \text{ because indep 0-avg}$$

$$\underbrace{\mathbb{E}\left[\{r - f(\tilde{x})\}^2\right]}_{\text{noise}} + \underbrace{\{f(\tilde{x}) - \bar{h}(\tilde{x})\}^2}_{\text{bias}^2} + \underbrace{\mathbb{E}\left[\{\bar{h}(\tilde{x}) - h(\tilde{x})\}^2\right]}_{\text{variance}}$$

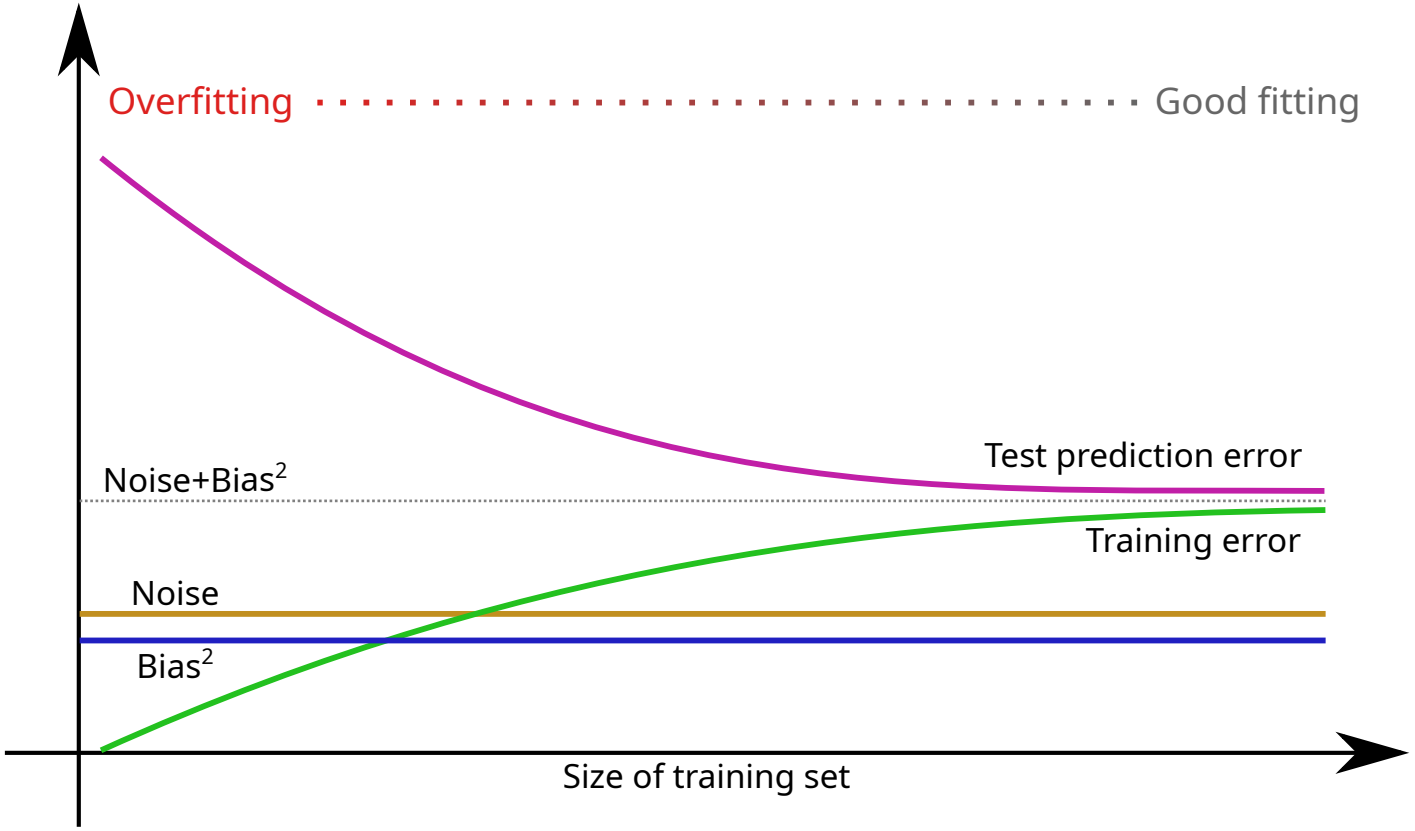
[4] Some of the terms involved are deterministic (in our setting), random because they depend on $\mathcal{X}$, or random because they depend on $r$

# Bias and variance as a function of model complexity



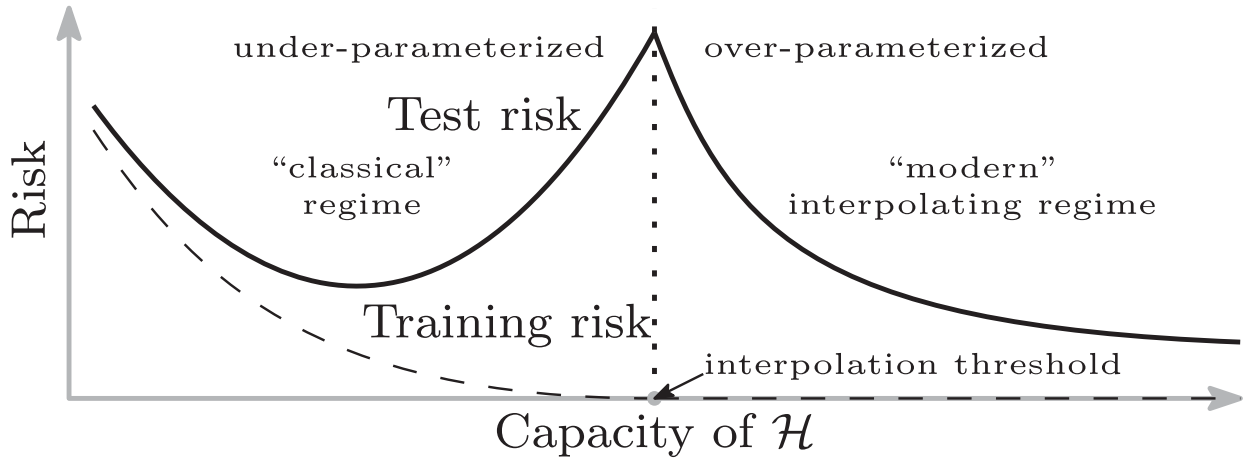

E.g. This diagram shows us that if we choose a more complex ML model, then we get more overfitting, and lower training error; but worse error on test (out-of-sample) data.

# Bias and variance as a function of training set size




E.g. This diagram shows us that if we have access to more training data, then performance on the test set will improve.

# Double descent in modern machine learning




Interesting and surprising recent results showing that:

- The "classical regime" holds on the left of the interpolation threshold (complexity just enough to interpolate data $\rightarrow$ zero training error)
- Past that point, we are in the "modern regime" where the test error decreases again, possibly due to implicit inductive bias of optimisers (e.g. SGD finds "shallow" as opposed to "sharp" minima)

Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off." https://dx.doi.org/10.1073/pnas.1903070116

# References

**Core material reading:** Textbook: Introduction to Machine Learning, Ethem Alpaydin, 2020

Alpaydin 2.4, 4.7

**Further interesting reading material:**

Wikipedia:

`https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff`
`https://en.wikipedia.org/wiki/Tikhonov_regularization`
`https://en.wikipedia.org/wiki/Least_squares#Lasso_method`

# Outline

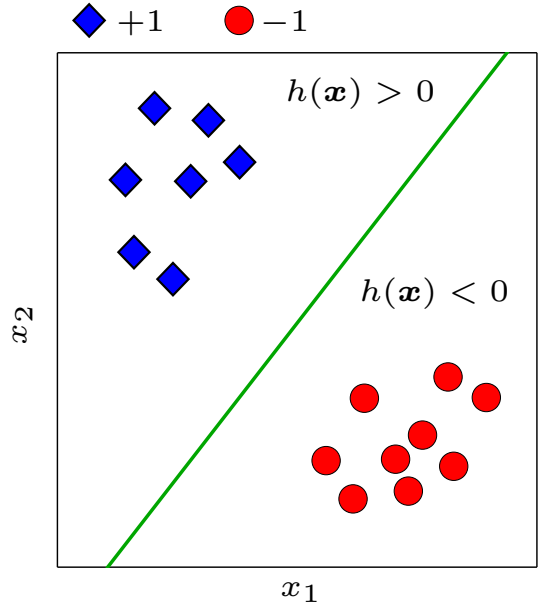
# Linear classifier

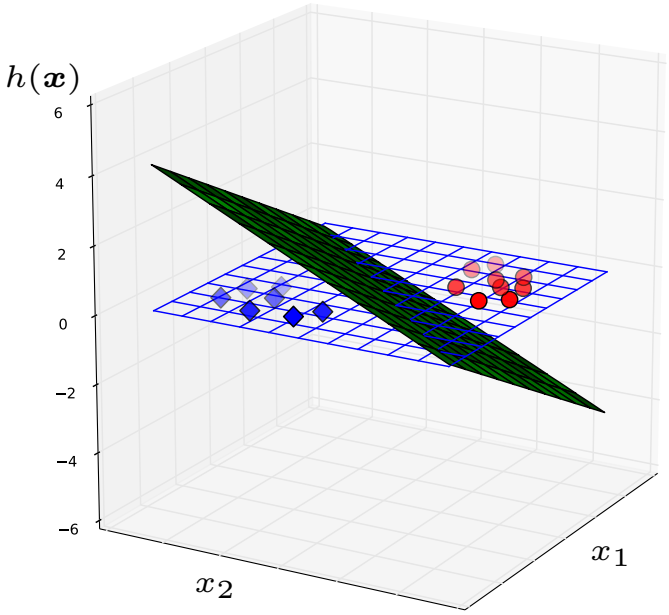
Let's consider a binary classification problem where the two classes are encoded as $\{+1, -1\}$.

A linear classifier tries to identify which class an example belongs to by making a decision based on the value of a linear combination of the features:

$h(\boldsymbol{x}|\boldsymbol{w}, b) = \sum_{i=1}^{P} w_i x_i - b = \langle \boldsymbol{w}, \boldsymbol{x} \rangle - b$ where $[\boldsymbol{w}, b]$ is a vector of parameters ($\boldsymbol{\theta}$)

with the predicted class being simply: $\text{sign}(h(\boldsymbol{x}|\boldsymbol{w}, b))$



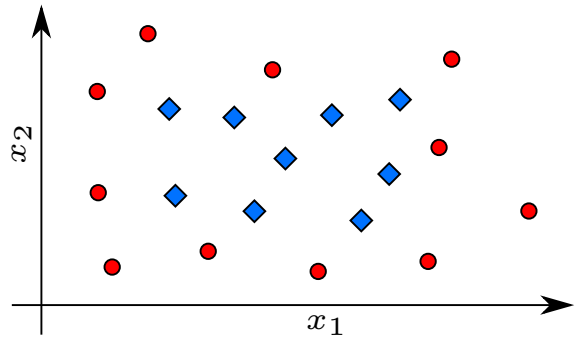
# Linear classifier

Pros:

- easy(-ier) to train than other classifiers
- less prone to overfitting
- assuming normalised features[5], the square of the weight $w_i^2$ indicates the relative importance of the feature $x_i$;
- the sign of $w_i$ indicates whether the feature $x_i$ has a positive or negative effect on the class $+1$;

Cons:

- prone to underfitting for non linearly separable problems.




[5]When using most classifiers the features should always be normalised to: 0 mean and range included in the interval $[-1, 1]$ or unitary standard deviation

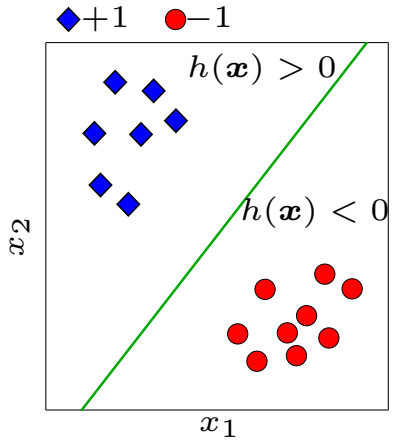
# Fitting the model

Given a training set $\mathcal{X} = \{(\boldsymbol{x}^t, r^t)\}_{t=1}^N$ with $r^t \in \{+1, -1\}$ ... we want to find the optimal linear classifier $h(\boldsymbol{x}|\boldsymbol{w}, b)$. How?

Ideally we would like[6]:

- $h(\boldsymbol{x}) > 0$ for all training examples belonging to the class $r = +1$
- $h(\boldsymbol{x}) < 0$ for all training examples belonging to the class $r = -1$

This is equivalent to: $r^t\, h(\boldsymbol{x}^t) > 0$ for all $t = 1..N$



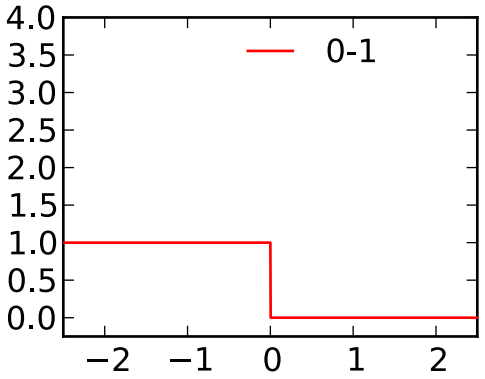

[6] To simplify the notation, we will often write $h(\boldsymbol{x}|\boldsymbol{w}, b)$ as $h(\boldsymbol{x})$ when unambiguous.

# Fitting the model: 0-1 loss

So, aiming for: $r^t\, h(\boldsymbol{x}^t) > 0$ for as many examples $t \in \{1..N\}$ as possible seems a good choice.

We can do so by defining the function:

$$\ell_{01}(z) = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$




using:

$$L(r, y) = \ell_{01}(ry) \text{ where } y = h(\boldsymbol{x})$$

and then minimizing the empirical risk:

$$R_{\text{em}}(\boldsymbol{w}, b|\mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N} L(r^t, h(\boldsymbol{x}^t|\boldsymbol{w}, b)) = \frac{1}{N}\sum_{t=1}^{N} \ell_{01}(r^t\, h(\boldsymbol{x}^t|\boldsymbol{w}, b))$$

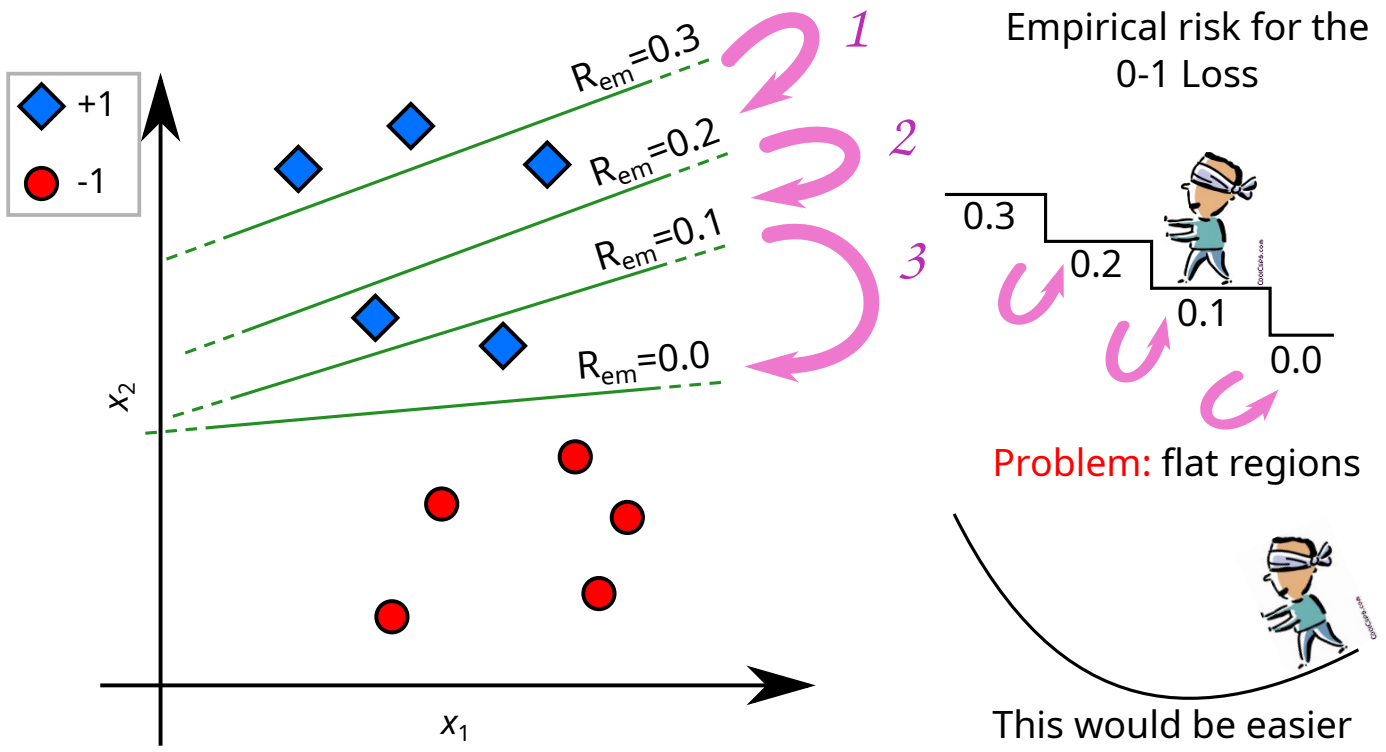
to find the optimal $[\boldsymbol{w}, b]$:

$$[\hat{\boldsymbol{w}}, \hat{b}] = \arg\min_{[\boldsymbol{w},b]} R_{\text{em}}(\boldsymbol{w}, b|\mathcal{X})$$

This finds the best separation line according to the 0-1 loss, i.e., by minimizing the fraction of misclassifications in the training set.

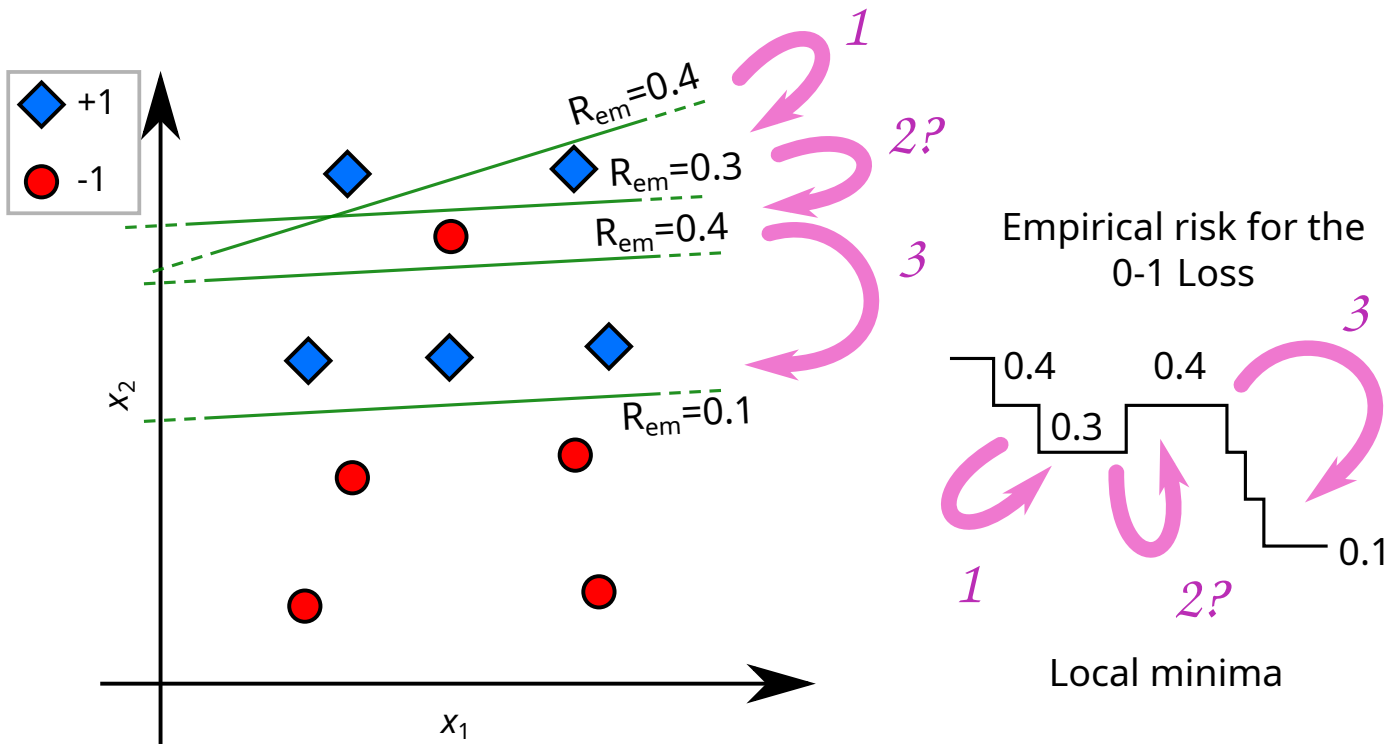
# Issues of the 0-1 loss (1)

Gradient is zero almost everywhere: finding the optimum in hard (practically unfeasible for most problems)



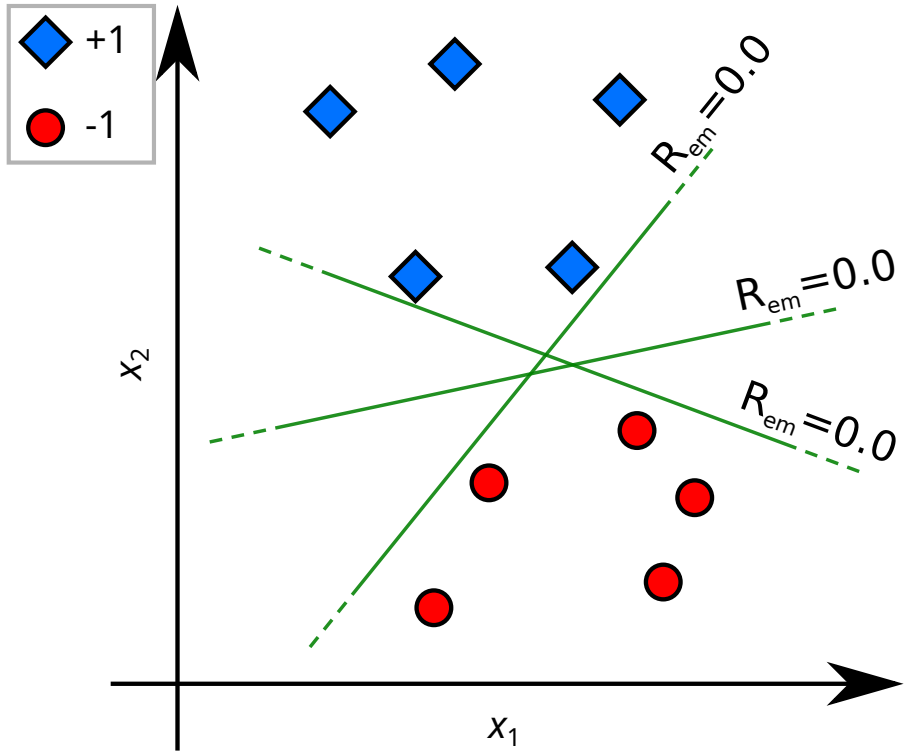
# Issues of the 0-1 loss (2)

Presence of (non-global) local minima



# Issues of the 0-1 loss (3)

Among all solution minimising the risk, some are clearly better (more robust) than others. Minimizing using the 0-1 loss doesn't do anything to promote those solutions.



# Outline

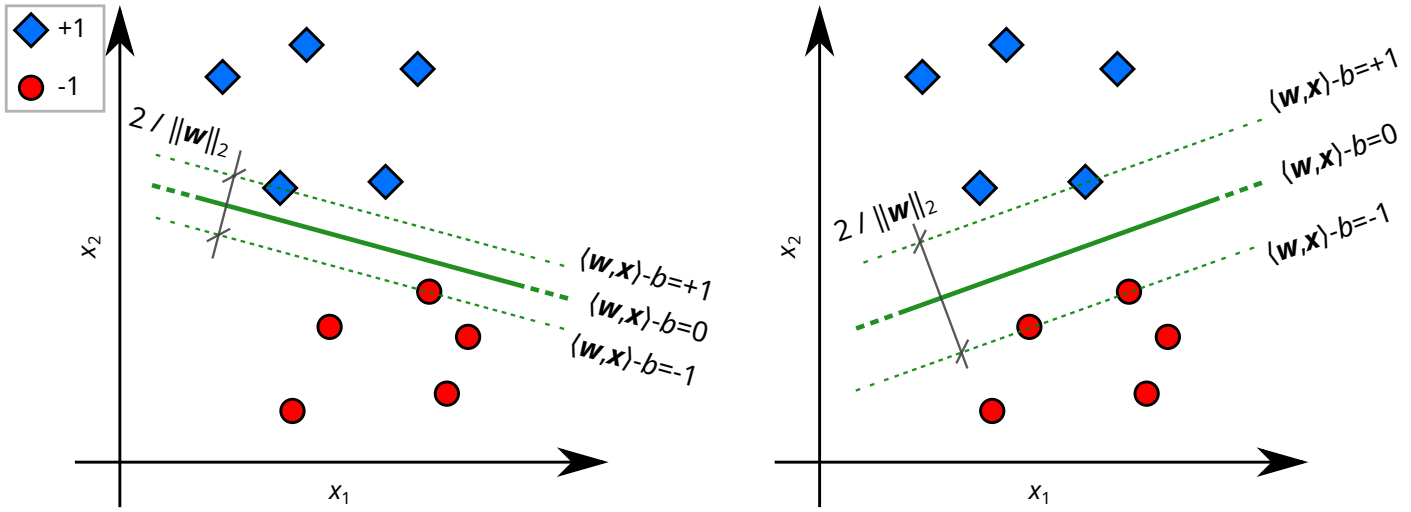
# Margin

In the linearly separable case, intuitively, the best solution is the one leading to the largest separation between the two classes.

We consider all linear classifiers $h(\boldsymbol{x}|\boldsymbol{w}, b) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle - b$ such that:

$$r^t \, h(\boldsymbol{x}^t) \geq 1 \text{ for all } t = 1..N$$

The separation between the two classes, i.e. the distance in the direction orthogonal to the hyperplane $h(\boldsymbol{x}) = 0$ between the closest two points of either class, is at least: $2/\|\boldsymbol{w}\|_2$.

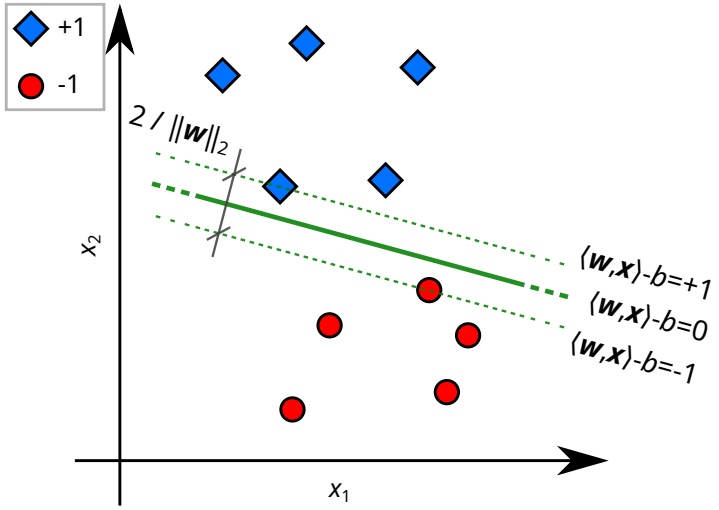


# Margin

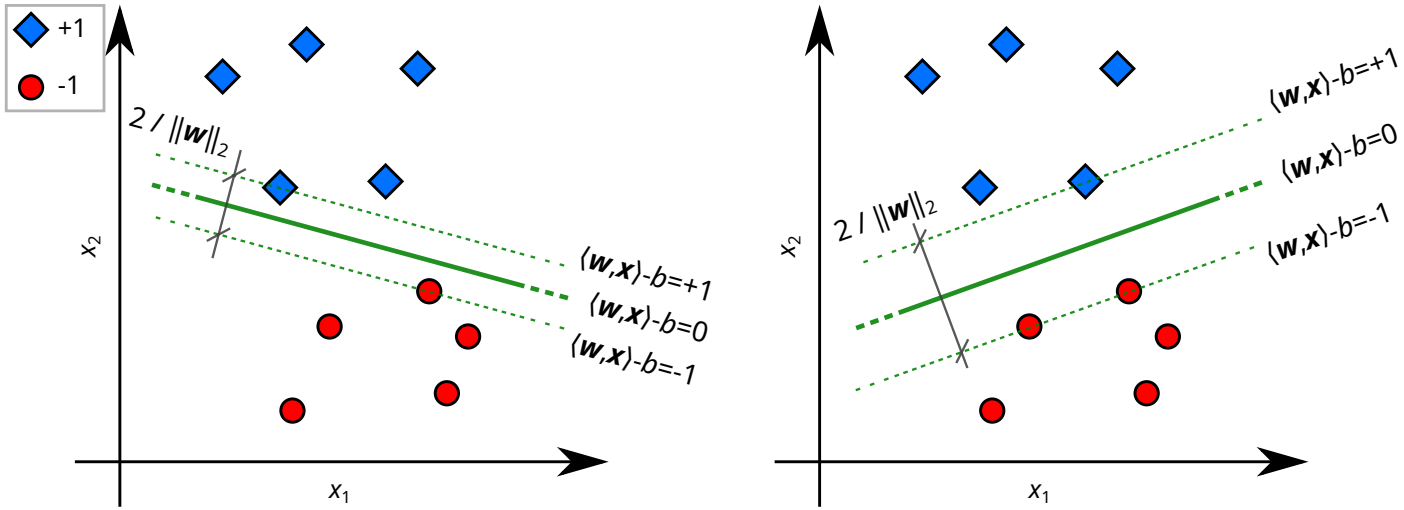
The separation between the two classes, i.e. the distance in the direction orthogonal to the hyperplane $h(\boldsymbol{x}) = 0$ between the closest two points of either class, is at least: $2/\|\boldsymbol{w}\|_2$.

Note that $\|\boldsymbol{w}\|_2$ means the euclidean length of $\boldsymbol{w}$

We can see this because subtracting $\langle \boldsymbol{w}, \boldsymbol{x}_1 \rangle - b = -1$ from $\langle \boldsymbol{w}, \boldsymbol{x}_2 \rangle - b = 1$ gives: $\langle \boldsymbol{w}, \boldsymbol{x}_2 - \boldsymbol{x}_1 \rangle = 2$; and so the component of the displacement vector from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ which is parallel to $\boldsymbol{w}$ (i.e. which is perpendicular to the green separating plane) must have length $2/\|\boldsymbol{w}\|_2$.



# Hard-margin Support Vector Machine




Maximizing the margin $2/\|\boldsymbol{w}\|_2$ is equiv. to minimizing $\frac{1}{2}\|\boldsymbol{w}\|_2$ or $\frac{1}{2}\|\boldsymbol{w}\|_2^2$.
The optimization problem:

$$[\hat{\boldsymbol{w}}, \hat{b}] = \arg\min_{[\boldsymbol{w},b]} \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

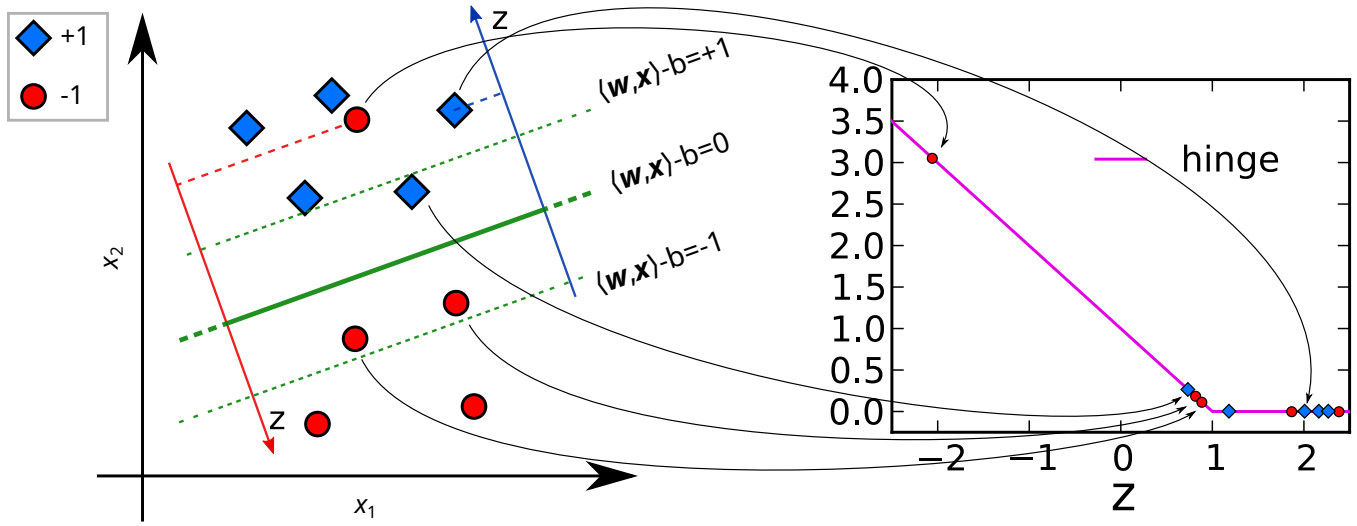
subject to:

$$r^t \left(\langle \boldsymbol{w}, \boldsymbol{x}^t \rangle - b\right) \geq 1 \text{ for all } t = 1..N$$

defines a Hard-margin Linear Support Vector Machine

# Non-linearly separable classes

We have seen the optimal solution in the linearly-separable case.




But if the two classes are not linearly separable, there will be no solution satisfying the constraints: $r^t \left(\langle \boldsymbol{w}, \boldsymbol{x}^t \rangle - b\right) \geq 1$ for all $t = 1..N$

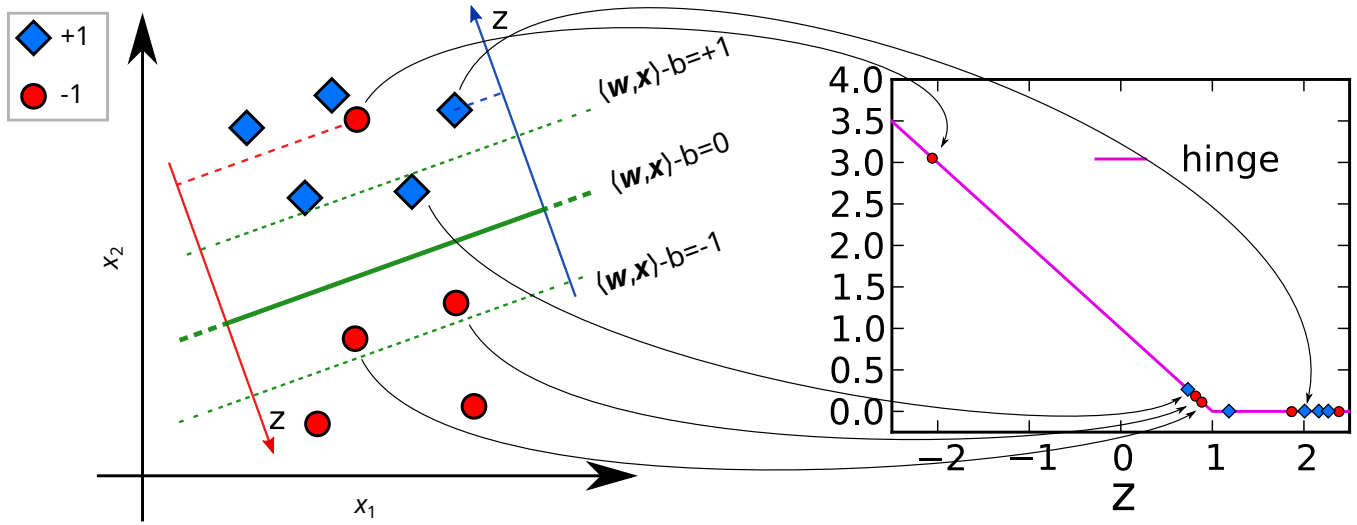
Therefore we introduce a soft margin where points are allowed to violate the constraint but pay a price ("loss") proportional to their distance from the margin boundary: $C\, \ell_{\mathrm{h}}(r\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle - b\right))$ where $\ell_{\mathrm{h}}$ is the hinge loss:

$$\ell_{\mathrm{h}}(z) = \begin{cases} 1 - z & \text{if } z < 1 \\ 0 & \text{otherwise} \end{cases} = \max\{0, 1 - z\}$$

while $C$ determines the trade-off between margin maximization and error minimization.

# Non-linearly separable classes

For each tentative solution $[\boldsymbol{w}, b]$ ...




the hinge loss is zero for points that are correctly classified with sufficient margin, while it is positive for misclassified points or points with insufficient margin.

Intuitively, maximizing the margin (like in the hard-margin case) while minimizing the average loss seems a good strategy.

In fact, the optimization problem:

$$[\hat{\boldsymbol{w}}, \hat{b}] = \arg\min_{[\boldsymbol{w},b]} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{t=1}^{N} \ell_{\mathrm{h}}(r^t\,(\langle \boldsymbol{w}, \boldsymbol{x}^t\rangle - b))$$

defines a Soft-margin Linear Support Vector Machine

# Soft-margin Linear Support Vector Machine

The Soft-margin Linear SVM is defined by the following optimisation goal:

$$[\hat{\boldsymbol{w}}, \hat{b}] = \arg\min_{[\boldsymbol{w},b]} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{t=1}^{N} \ell_{\mathrm{h}}(r^t\left(\langle \boldsymbol{w}, \boldsymbol{x}^t\rangle - b\right))$$

Note that there are two terms here to minimise:

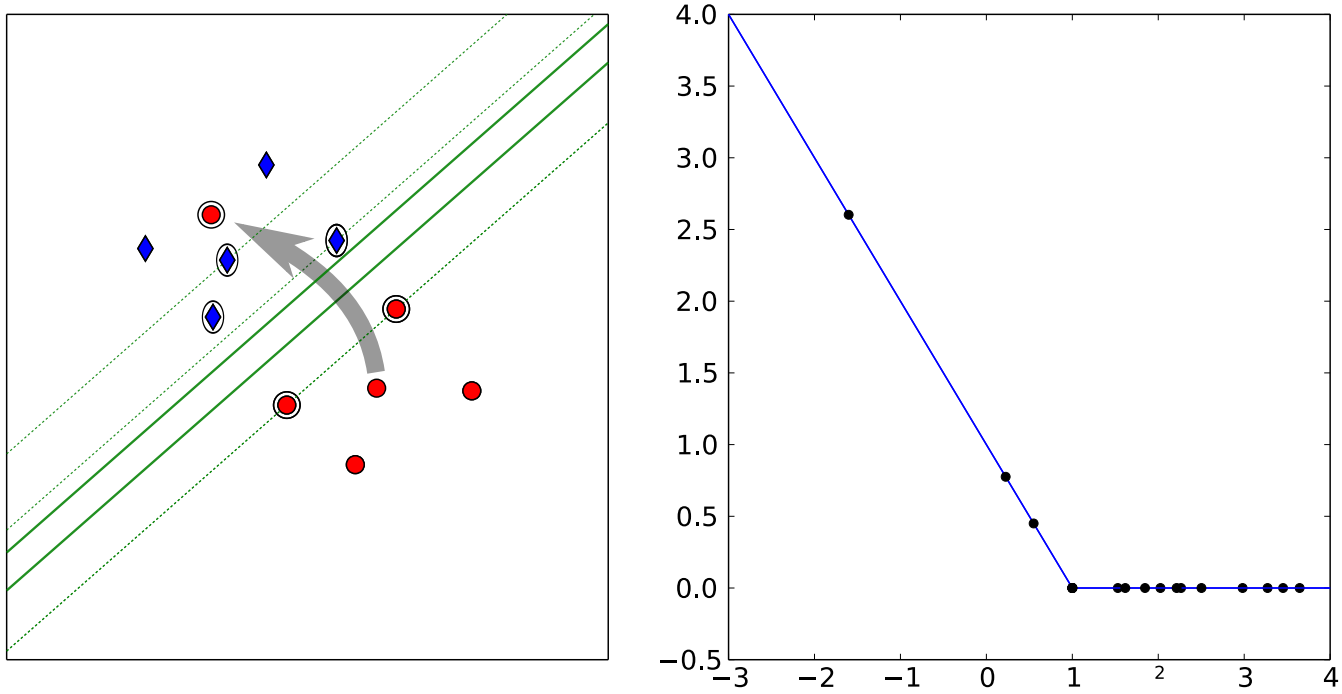
- The $\|\boldsymbol{w}\|_2^2$ term. This is asking the ML system to produce maximum margin between the two different classes of points
- The $C\sum_{t=1}^{N} \ell_{\mathrm{h}}(r^t\left(\langle \boldsymbol{w}, \boldsymbol{x}^t\rangle - b\right))$ term. This asking the ML system not to mis-classify any training point.

Hence by adjusting the hyper-parameter $C$, we can choose how important fitting every single training point is (variance) versus keeing a nice clean simple model with maximum margin (bias).

- If $C$ is too large, then we may get overfitting.
- If $C$ is too small, then we may get underfitting.

# Support vectors

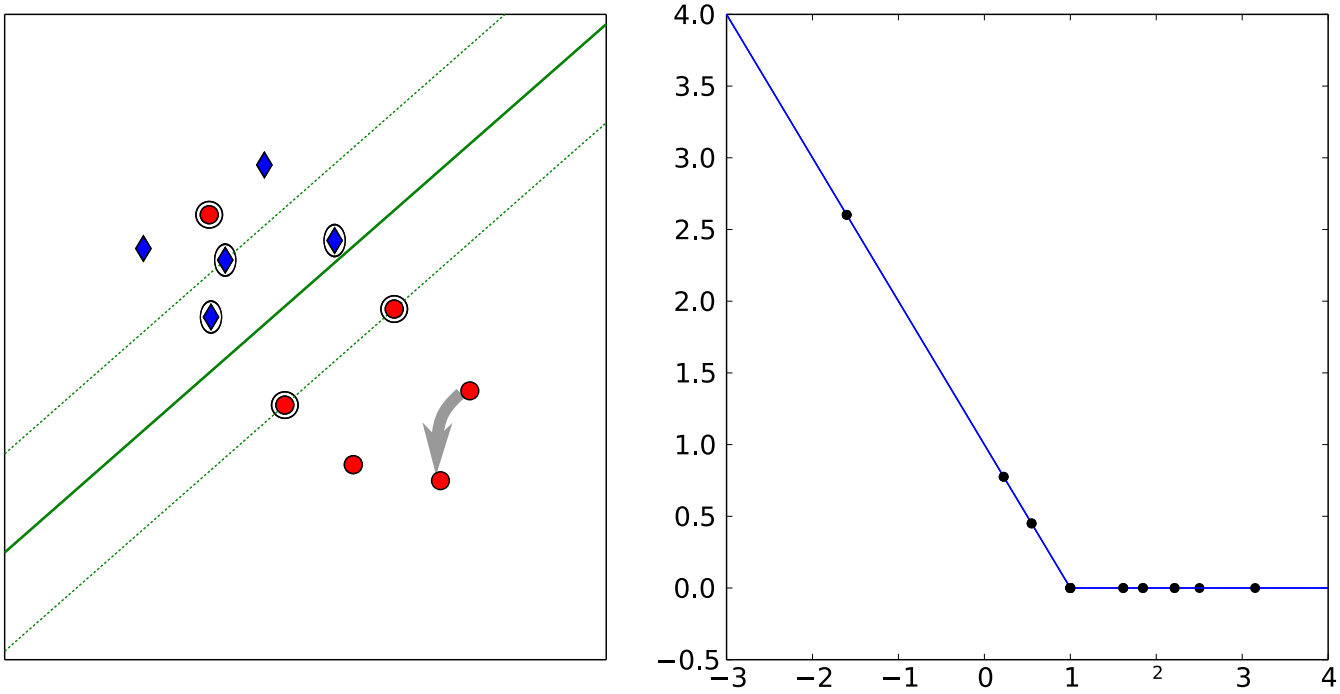
Why the name Support Vector Machine?
Let's see what happens when we move some of the training examples ($C$ large):




- If the classes are separable all points have 0 loss
- If they become non-separable the margin gets larger and some points have non-zero loss

# Support vectors

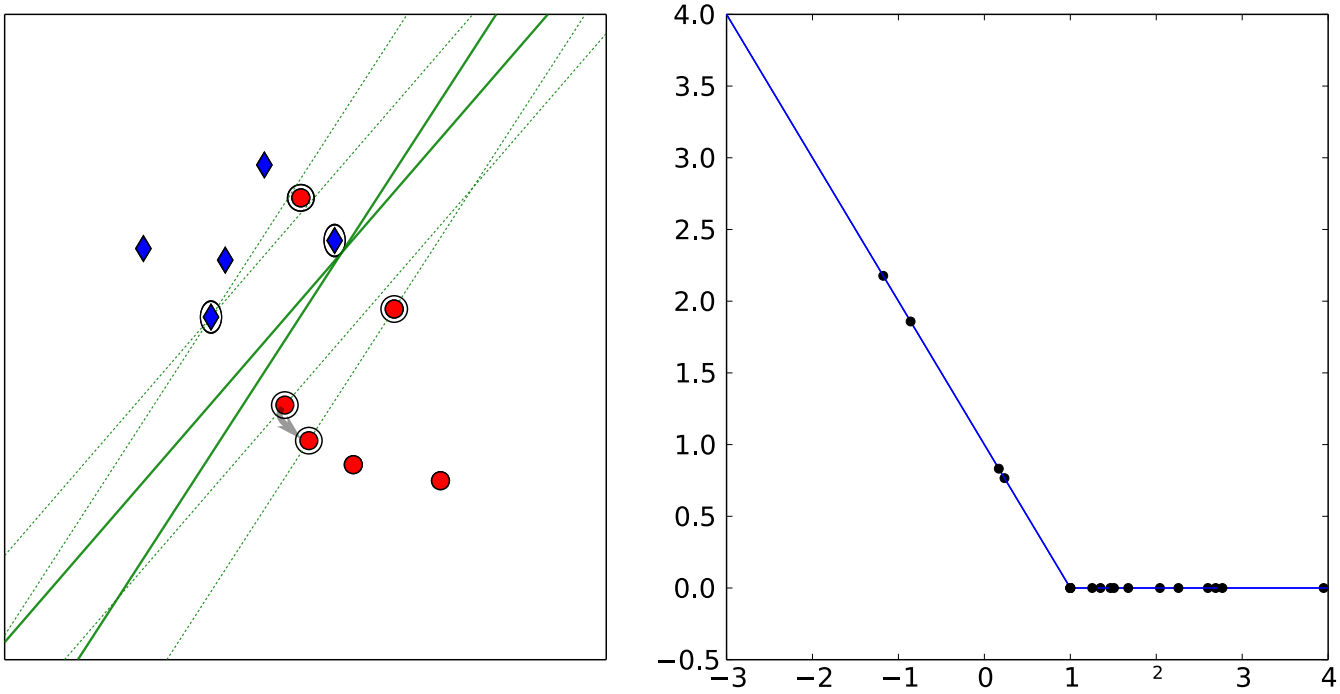
Why the name Support Vector Machine?
Let's see what happens when we move some of the training examples ($C$ large):




- If point moves within the correct side of the margin (with 0 loss) nothing changes

# Support vectors

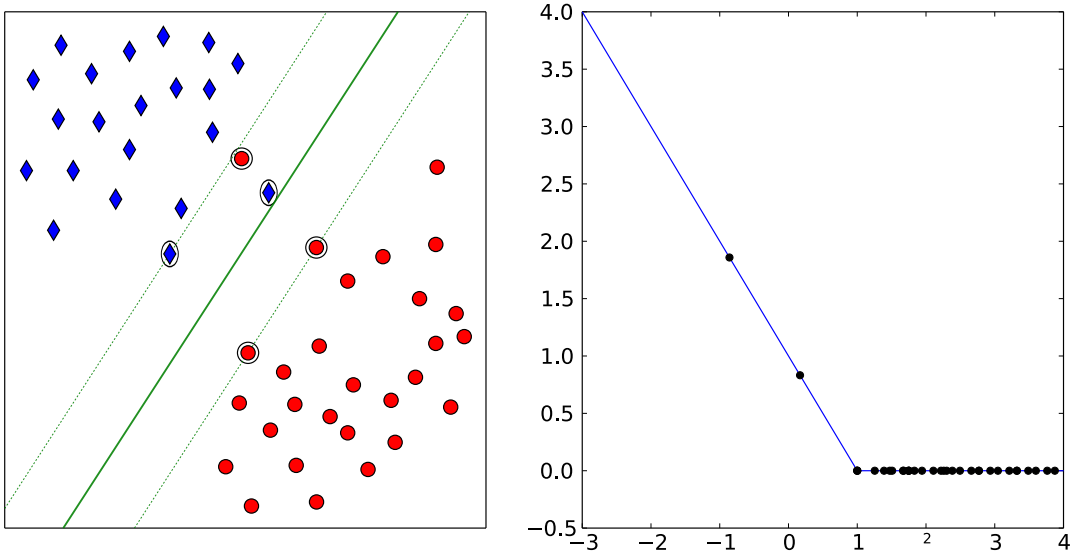
Why the name Support Vector Machine?
Let's see what happens when we move some of the training examples ($C$ large):




- If a misclassified point moves, the decision hyperplane and the margin change
- If a point on the boundary of the margin zone changes, the decision hyperplane and the margin change

# Support vectors

Apparently, the decision hyperplane depends only on a subset of the training points . . .




those for which $r^t \left( \langle \boldsymbol{w}, \boldsymbol{x}^t \rangle - b \right) \leq 1$, i.e.:

- misclassified training points
- training points that are correctly classified with insufficient margin (including points lying exactly on the margin boundary)

These points are called support vectors.

# Support vectors

Actually, thanks to the Representer theorem we can say more:

- the optimal $\hat{\boldsymbol{w}}$ is a linear combination of support vectors

$$\hat{\boldsymbol{w}} = \sum_{t=1}^{N} \alpha_t \, r^t \boldsymbol{x}^t$$

  where $\alpha_t = 0$ if $t \notin \mathcal{S}$, the set of indices of support vectors

- and, therefore, the optimal decision function is a weighted sum of the inner product between the instance to be classified and the support vectors

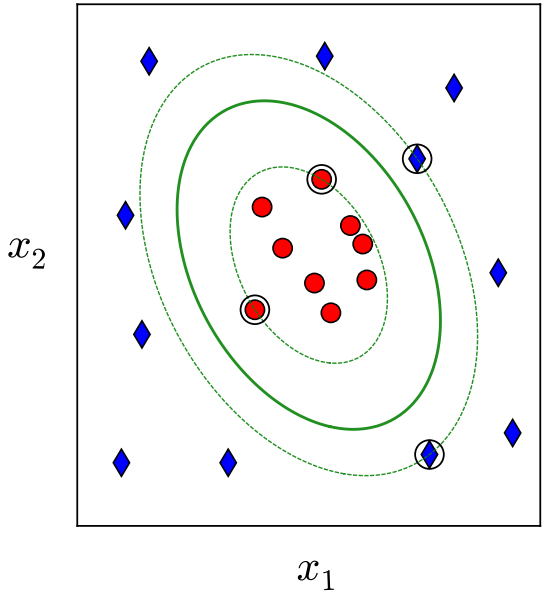
$$\hat{h}(\boldsymbol{x}) = \langle \hat{\boldsymbol{w}}, \boldsymbol{x} \rangle - \hat{b} = \sum_{t=1}^{N} \alpha_t \, r^t \langle \boldsymbol{x}^t, \boldsymbol{x} \rangle - \hat{b} = \sum_{t \in \mathcal{S}} \alpha_t \, r^t \langle \boldsymbol{x}^t, \boldsymbol{x} \rangle - \hat{b}$$

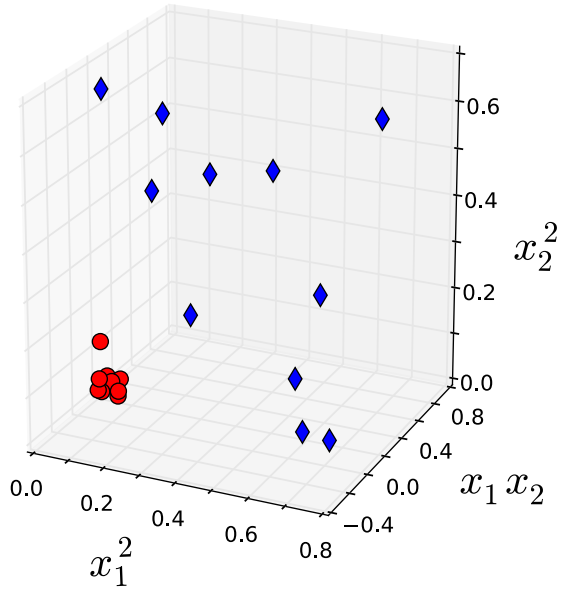
The fact that the decision function only depends on these training instances, explains the name "support vectors".

# Nonlinear SVMs

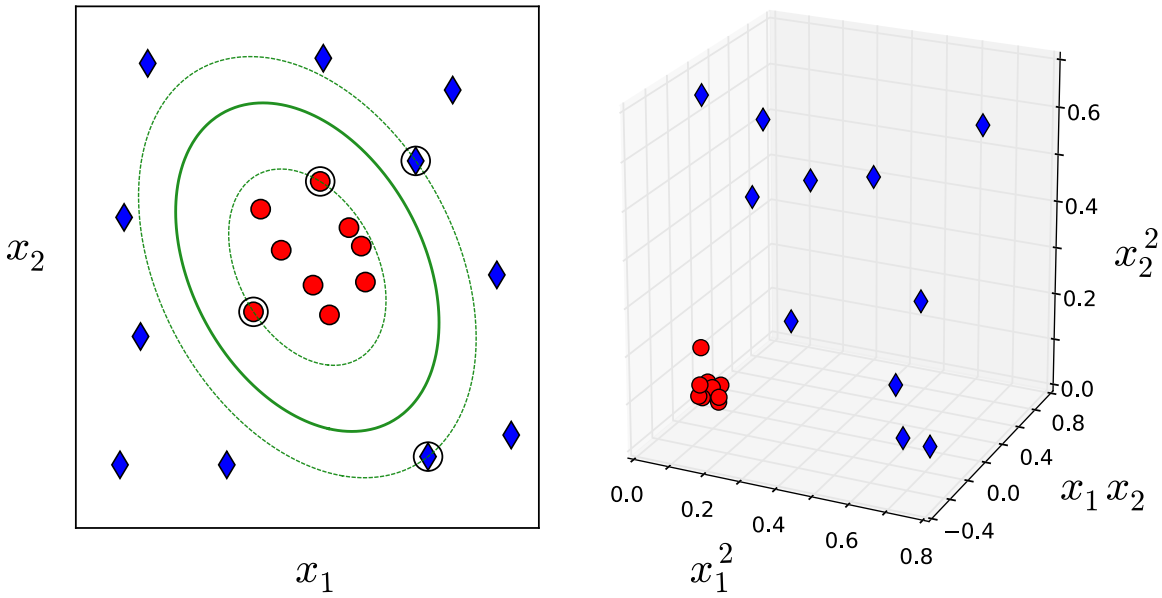
We can "create new features" by computing nonlinear functions of the original features (e.g., powers in polynomial regression)

$$\phi(\boldsymbol{x}) = \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_2 \end{bmatrix}$$



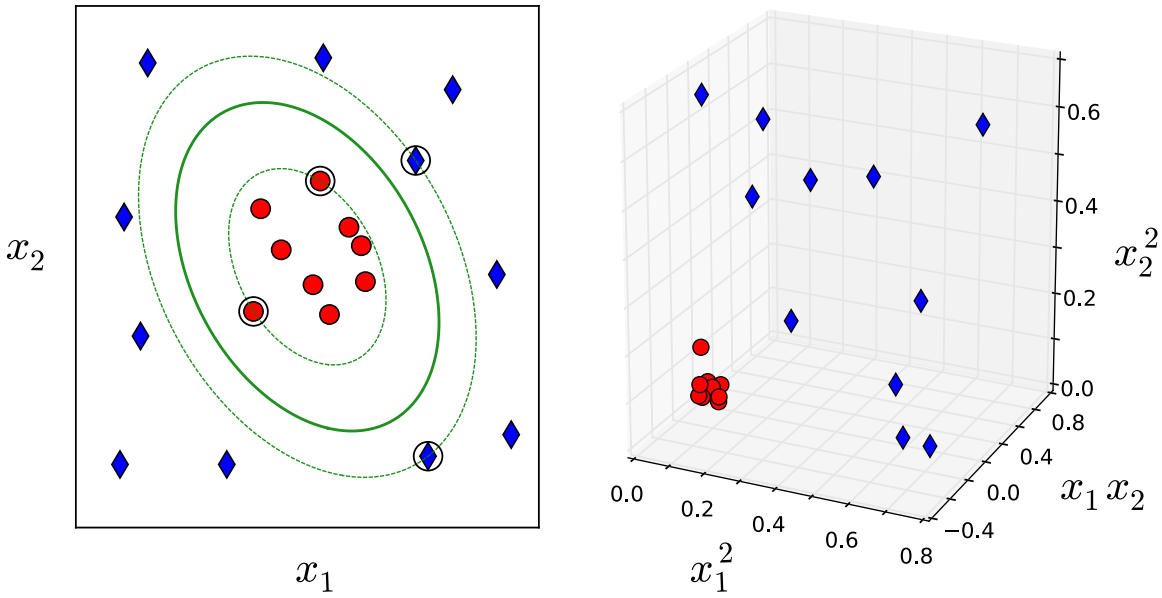
# Nonlinear SVMs: Mapped Feature Space




Notice that the left diagram is not linearly separable. But all the red points seem to lie in a circle, $(x_1)^2 + (x_2)^2 < k^2$. (c.f. "$x^2 + y^2 = r^2$")

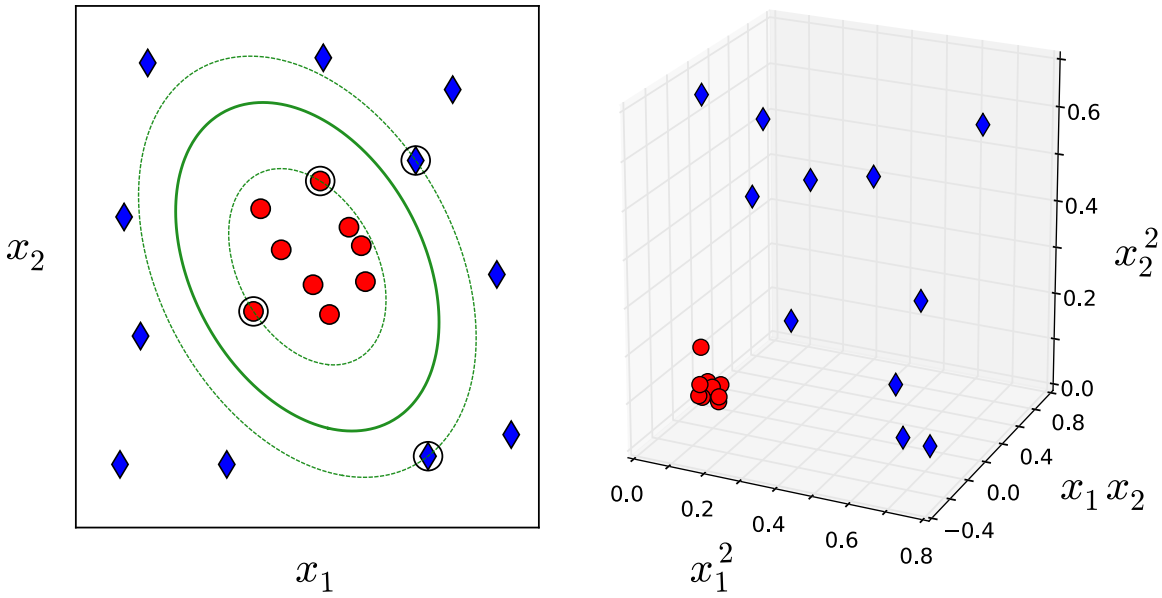
So if instead of having input features to our SVM as $(x_1, x_2)$, instead we allow our input features to include $((x_1)^2, (x_2)^2)$, then the problem becomes trivially easy to find a linear separation.

# Nonlinear SVMs: Mapped Feature Space




By choosing new mapped features $\phi(\boldsymbol{x}) = \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_2 \end{bmatrix}$, we project our training points to the new 3d diagram on the right. In this diagram (The mapped feature space), it's quite easy to find a separating plane that isolates the red dots from the blue ones.

# Nonlinear SVMs: Mapped Feature Space




In general, we can introduce a function $\phi(\boldsymbol{x})$ that maps the original features into a new set of mapped features.

We can then train a SVM using the elements of the new feature vector $\phi(\boldsymbol{x})$ as features and build a linear classifier in the mapped feature space.

This would correspond to a non-linear classifier in the original feature space.

# Nonlinear SVMs (maths given as reference)

We can now take our original Soft-margin Linear SVM

$$[\hat{\boldsymbol{w}}, \hat{b}] = \arg\min_{[\boldsymbol{w},b]} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{t=1}^{N} \ell_{\mathrm{h}}(r^t\, h(\boldsymbol{x}^t|\boldsymbol{w}, b))$$

and replace the original feature vector $\boldsymbol{x}$ with the nonlinear mapping $\phi(\boldsymbol{x})$, using:

$$\boldsymbol{w} = \sum_{t=1}^{N} \alpha_t\, r^t \phi(\boldsymbol{x}^t) \quad \begin{matrix}\rightarrow \\ \searrow\end{matrix} \quad \|\boldsymbol{w}\|_2^2 = \langle \boldsymbol{w}, \boldsymbol{w}\rangle = \sum_{t=1}^{N}\sum_{s=1}^{N} \alpha_t\, \alpha_s\, r^t r^s \langle \phi(\boldsymbol{x}^t), \phi(\boldsymbol{x}^s)\rangle$$

$$h(\boldsymbol{x}|\boldsymbol{w}, b) = \langle \boldsymbol{w}, \phi(\boldsymbol{x})\rangle - b = \sum_{t=1}^{N} \alpha_t\, r^t \langle \phi(\boldsymbol{x}^t), \phi(\boldsymbol{x})\rangle - b$$

Notice that $\phi(\boldsymbol{x})$ is never required explicitly but only as argument of the inner product. Therefore, given a feature mapping $\phi(\boldsymbol{x})$, we define the corresponding Kernel as:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z})\rangle$$

Finally, we can restate the optimization problem as finding the optimal $[\hat{\boldsymbol{\alpha}}, \hat{b}]$ such that:

$$[\hat{\boldsymbol{\alpha}}, \hat{b}] = \arg\min_{[\boldsymbol{\alpha},b]} \frac{1}{2}\sum_{t=1}^{N}\sum_{s=1}^{N} \alpha_t\, \alpha_s\, r^t r^s K(\boldsymbol{x}^t, \boldsymbol{x}^s) + C\sum_{t=1}^{N} \ell_{\mathrm{h}}\left(r^t\left(\sum_{s=1}^{N} \alpha_s\, r^s K(\boldsymbol{x}^s, \boldsymbol{x}^t) - b\right)\right)$$

Although one could in principle solve this problem directly, there are more efficient approaches that recast it as a constrained optimization problem that can efficiently solved in the dual space.

# Kernels

In the previous slides we introduced the Kernel function:

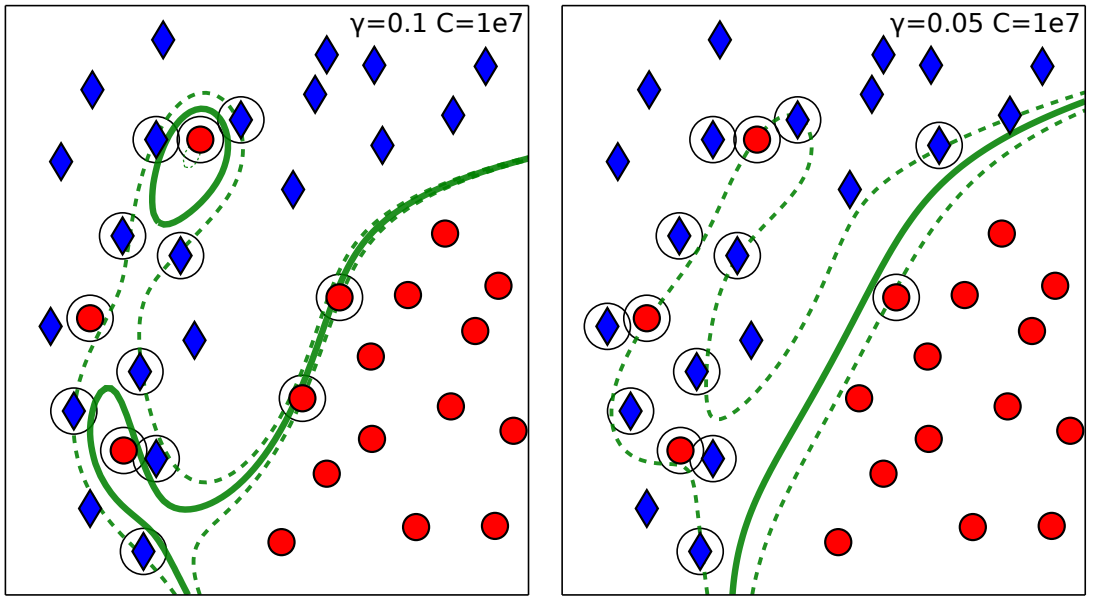
- The kernel trick allowed the use of SVMs as nonlinear classifiers.
- Like the inner product, the kernel represents a kind of similarity function.
- Given $\phi$, the corresponding kernel $K(\boldsymbol{x}, \boldsymbol{z})$ can be easily computed as the inner product between $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{z})$.
- Often $K(\boldsymbol{x}, \boldsymbol{z})$ is cheaper to compute than $\phi$; as the algorithm depends on $K(\boldsymbol{x}, \boldsymbol{z})$ (and only implicitly on $\phi$), we can train and use a SVM in the high dimensional feature space given by $\phi$ without having to explicitly compute it.
  In fact, using kernels, the decision function is simply:

$$h(\boldsymbol{x}) = \sum_{s \in \mathcal{S}} \alpha_s \, r^s K(\boldsymbol{x}^s, \boldsymbol{x}) - b$$

# Kernels

Examples of widely used kernels:

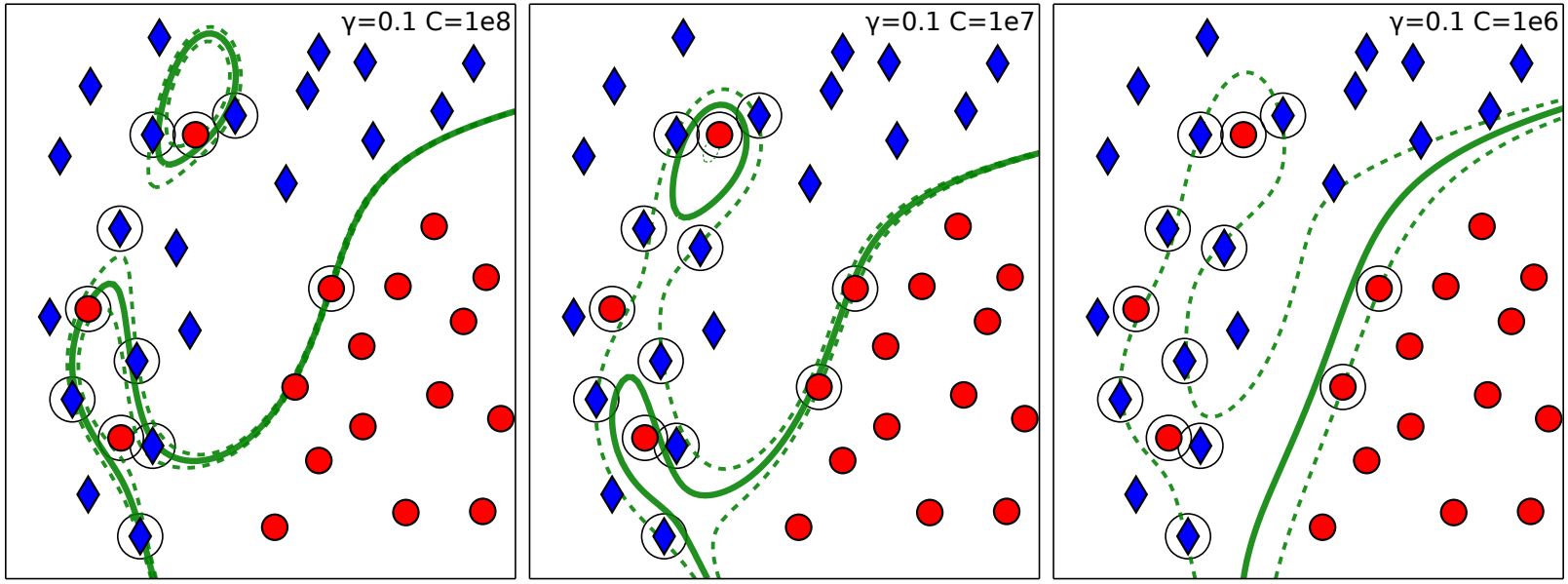
- Linear: $K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle$
- Polynomial (homogeneous)[7]: $K(\boldsymbol{x}, \boldsymbol{z}) = (\langle \boldsymbol{x}, \boldsymbol{z} \rangle)^d$
- Polynomial (inhomogeneous): $K(\boldsymbol{x}, \boldsymbol{z}) = (\langle \boldsymbol{x}, \boldsymbol{z} \rangle + \gamma)^d$
- Gaussian radial basis function: $K(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\gamma \left\| \boldsymbol{x} - \boldsymbol{z} \right\|^2)$, $\gamma > 0$.




[7] Our example of slide 67 corresponds to a quadratic homogeneous polynomial kernel

# Regularisation

The parameter $C$ (and also the kernel's parameters $\gamma$, $d$, ...) tunes the bias (underfitting) VS variance (overfitting) trade-off.
Example:




Often their optimal values need to be estimated through cross-validation.

# References

**Core material reading:** Textbook: Introduction to Machine Learning, Ethem Alpaydin, 2020

Alpaydin 4.8
13.1, 13.2, 13.3, 13.4, 13.5, 13.6

**Further interesting reading material:**

Wikipedia:

`https://en.wikipedia.org/wiki/Support_vector_machine`

`https://en.wikipedia.org/wiki/Kernel_method`