

# TDDC17 LE8 HT2025

## Machine Learning I

Fredrik Heintz

Dept. of Computer Science

Linköping University

[fredrik.heintz@liu.se](mailto:fredrik.heintz@liu.se)

@FredrikHeintz

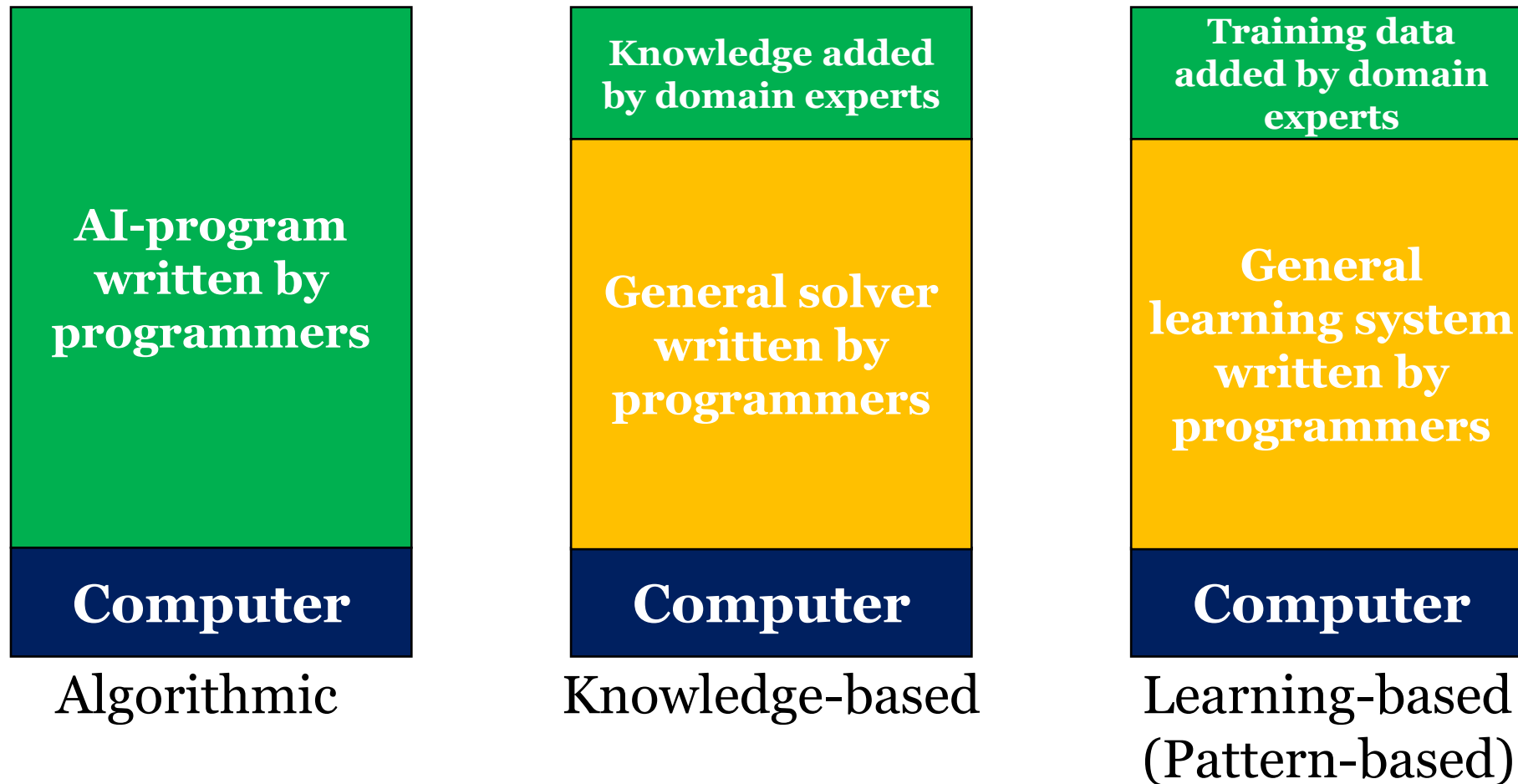
Outline:

- Introduction to machine learning
- Unsupervised learning
- Supervised learning
- Evaluating machine learning methods

# Outline of Machine Learning Lectures

- Introduction to machine learning
  - Supervised learning, unsupervised learning, reinforcement learning
- Deep Learning
- Reinforcement learning

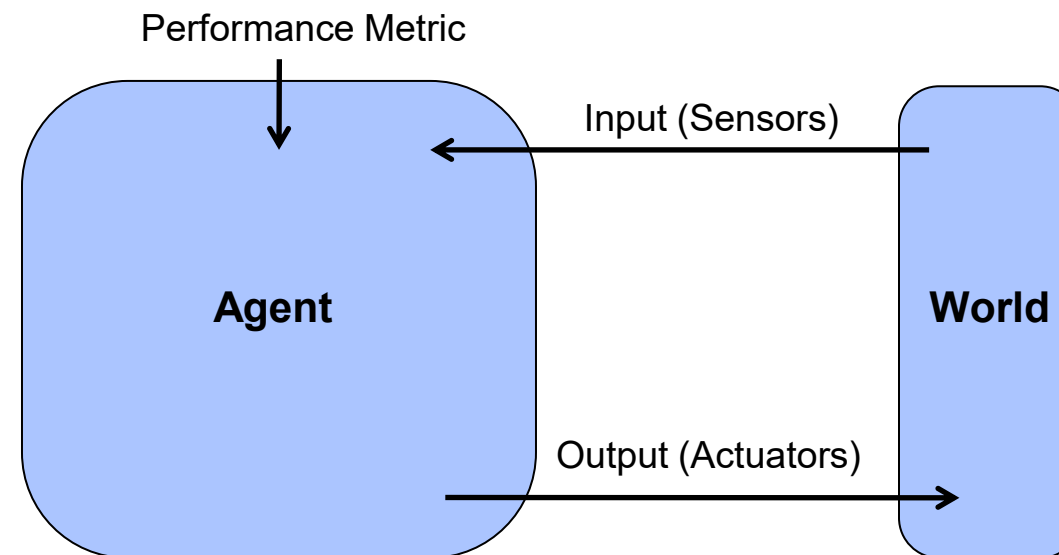
# Algorithmic, Knowledge-Based and Learning-Based AI



# To Define Machine Learning

*Given a task, mathematically encoded via some performance metric, a machine can improve its performance by learning from experience (data)*

From the agent perspective:



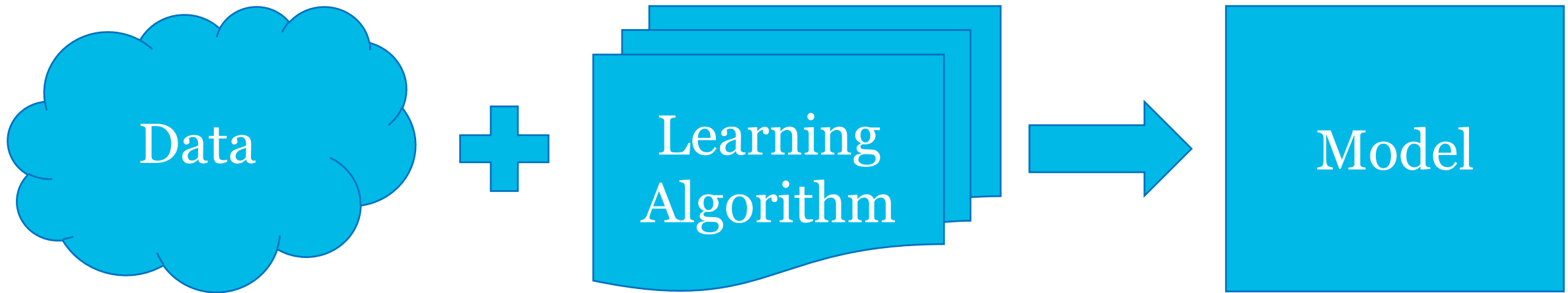
# To Define Machine Learning

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .
- Suppose your email program watches which emails you do or do not mark as spam and based on that learns how to better filter spam.
  - Experience  $E$  is Watching you label emails as spam or not spam.
  - Task  $T$  is Classifying emails as spam or not spam.
  - Performance  $P$  is The number (or fraction) of emails correctly classified as spam/not spam.

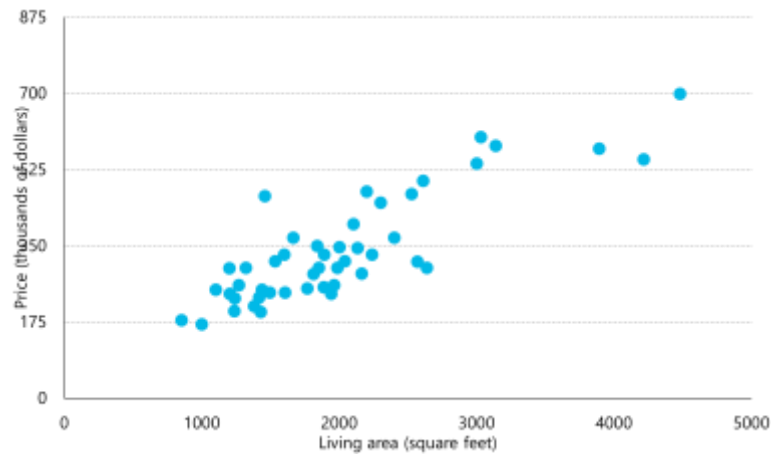
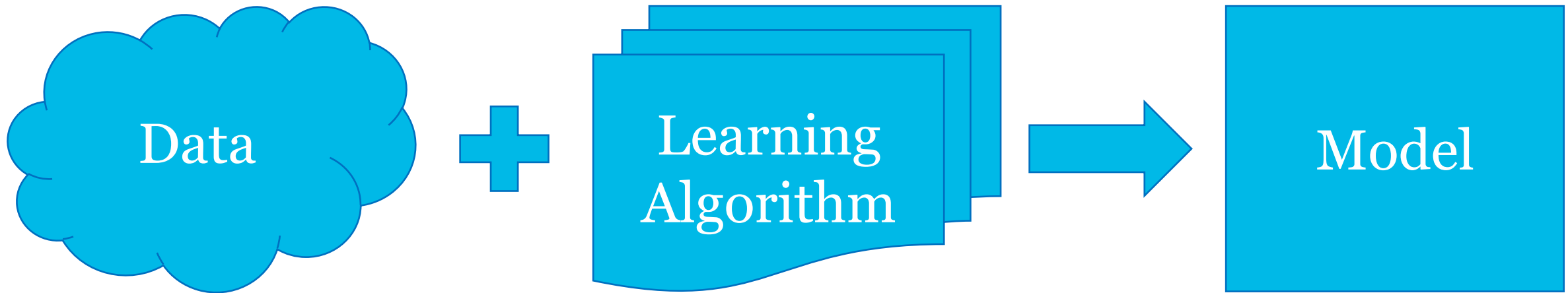
# Machine Learning

- Machine Learning is a branch of artificial intelligence that provides the computer system the ability to progressively learn and improve its performance on handling various tasks without being explicitly programmed to perform all the task.
- Another definition of Machine Learning explains it as: the process of trying to deduce unknown values from known values.

# Machine Learning

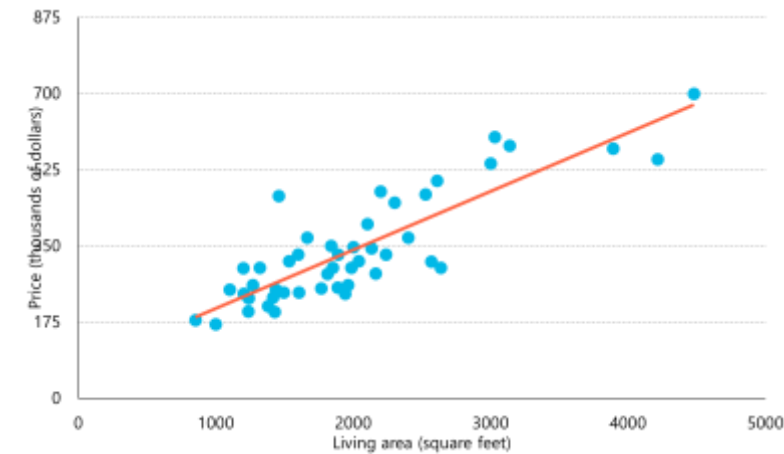


# Machine Learning



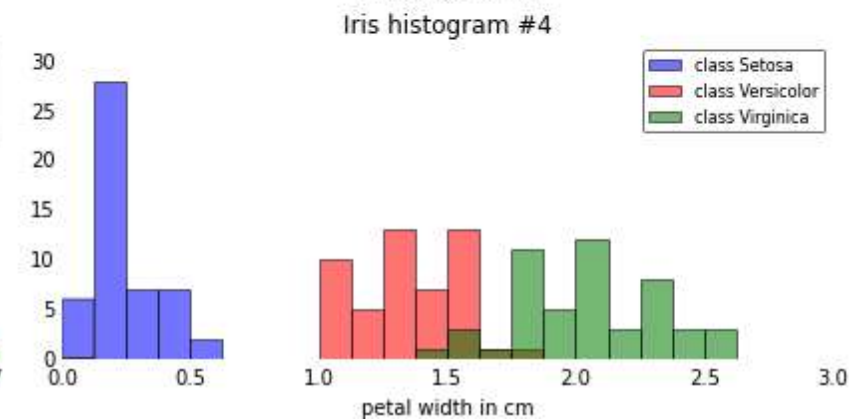
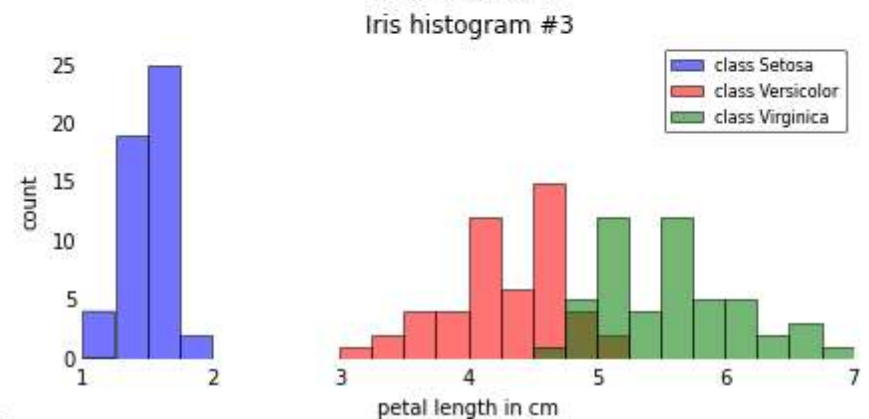
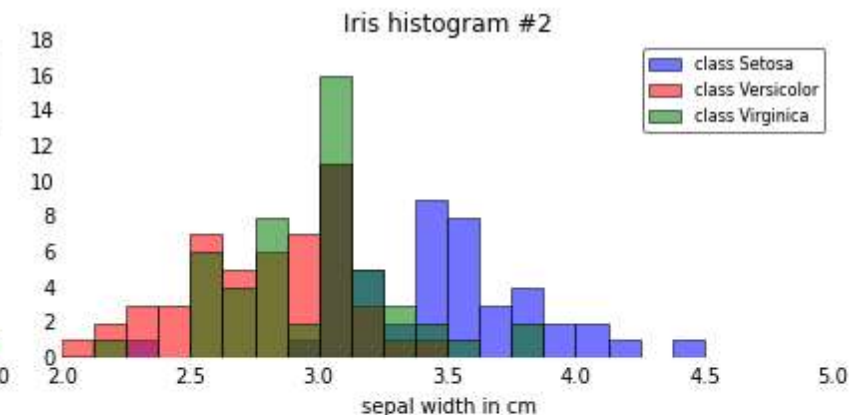
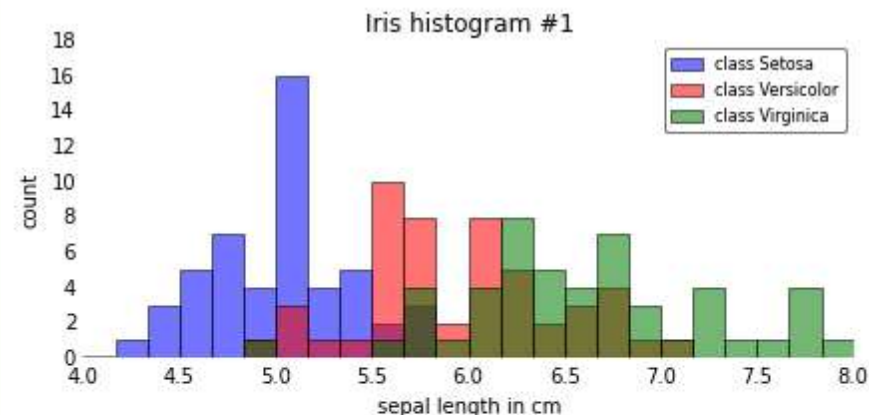
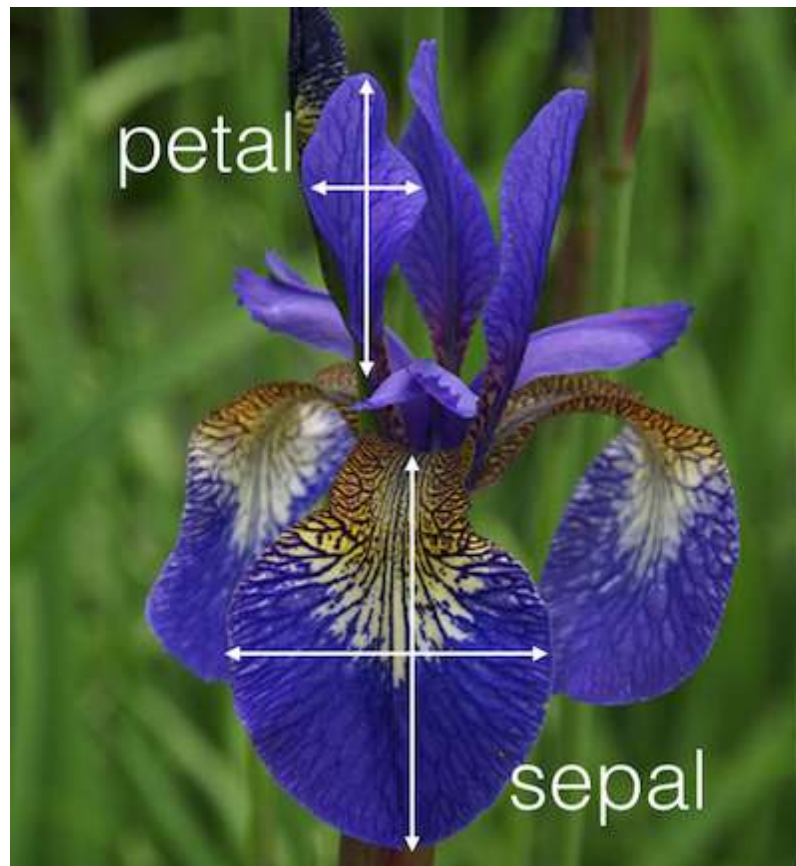
Linear regression

$$\hat{y} = x\theta$$

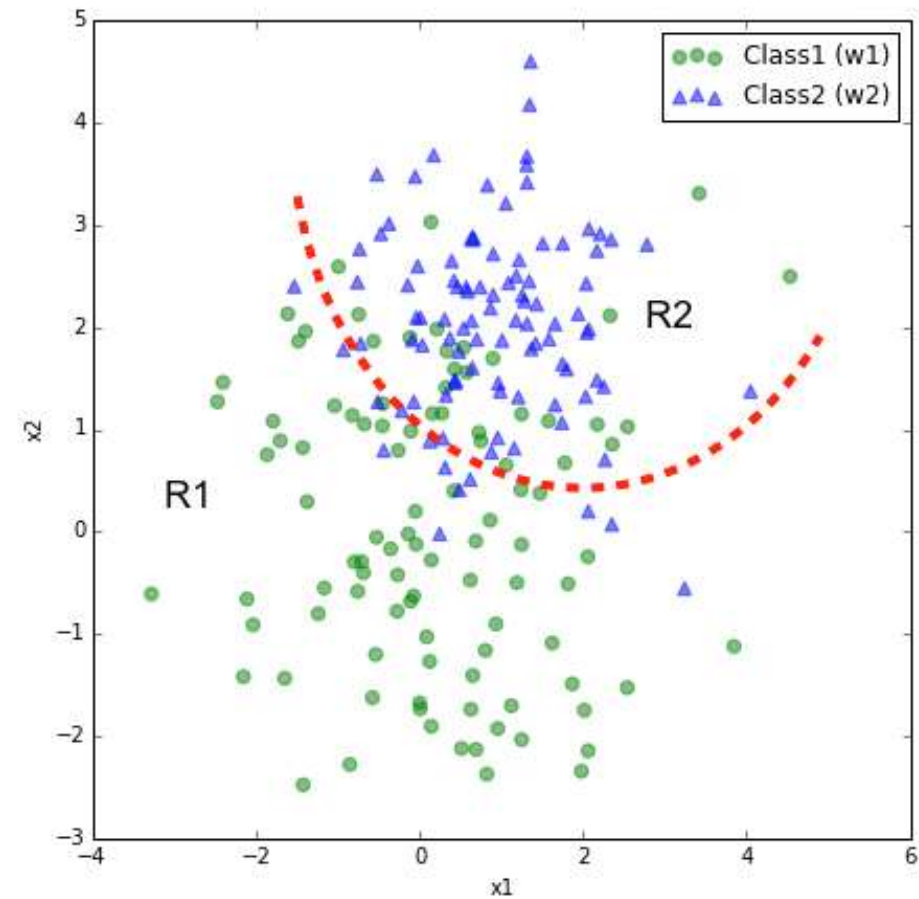
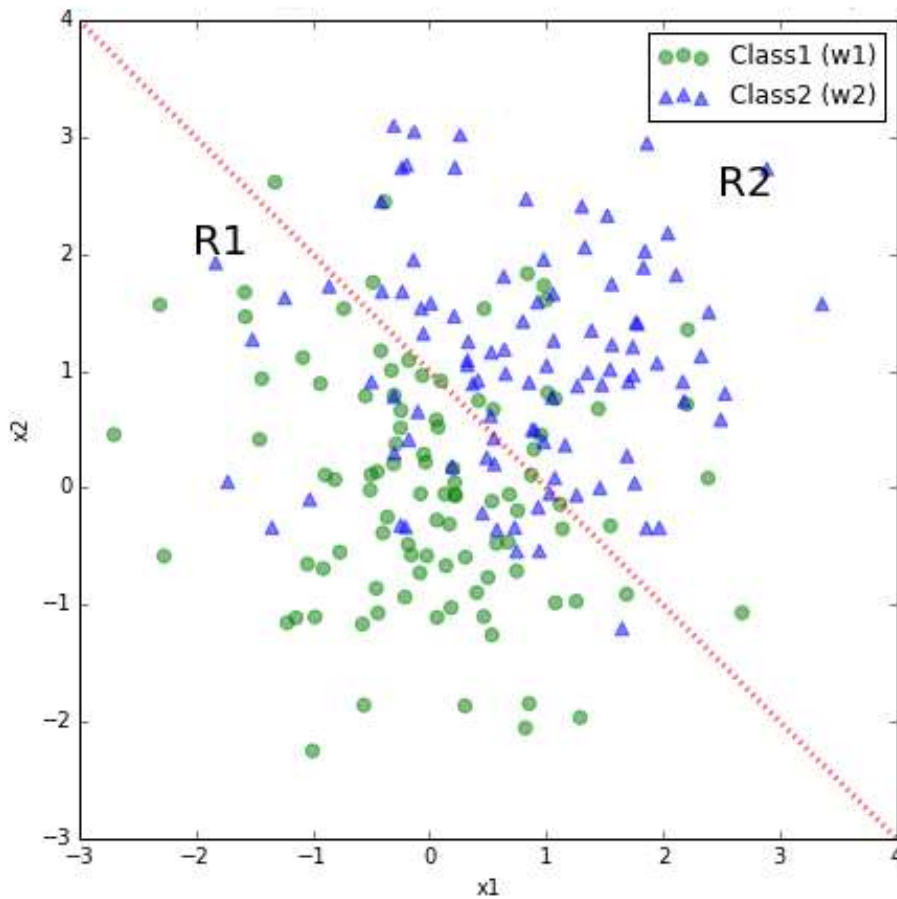




# The Importance of Feature Selection

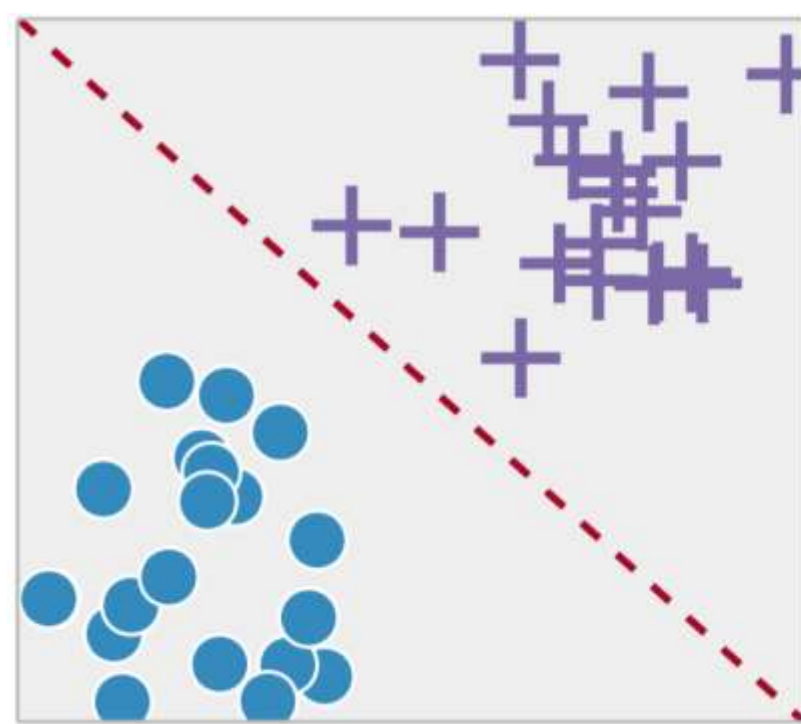


# Classification

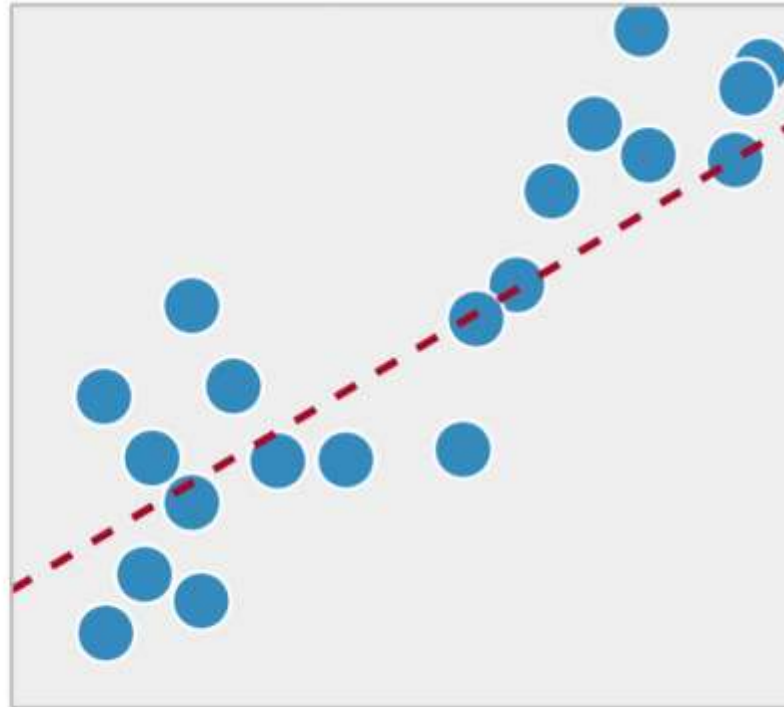


# Model Types

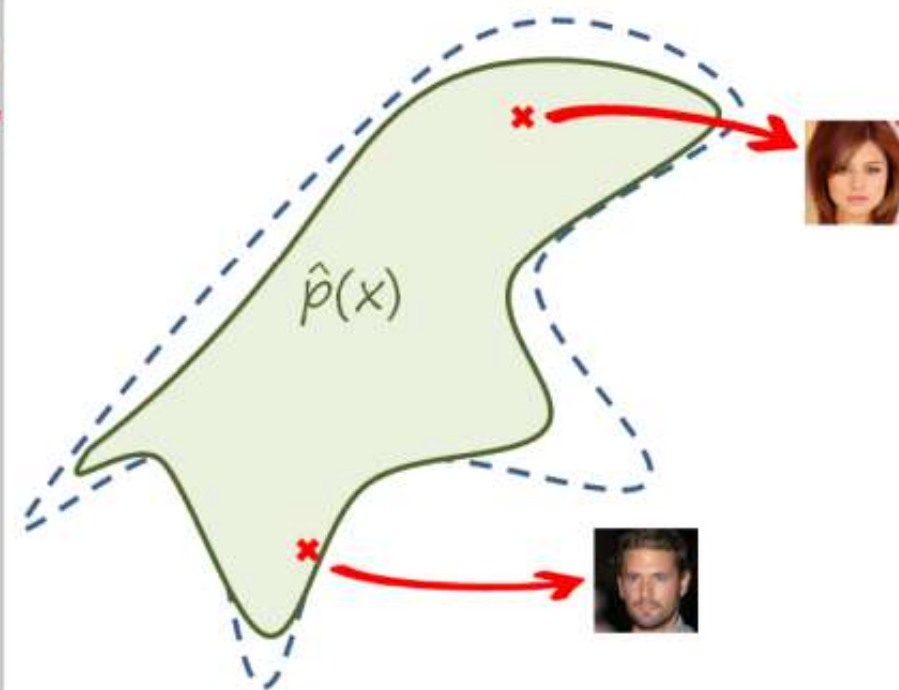
## Classification



## Regression

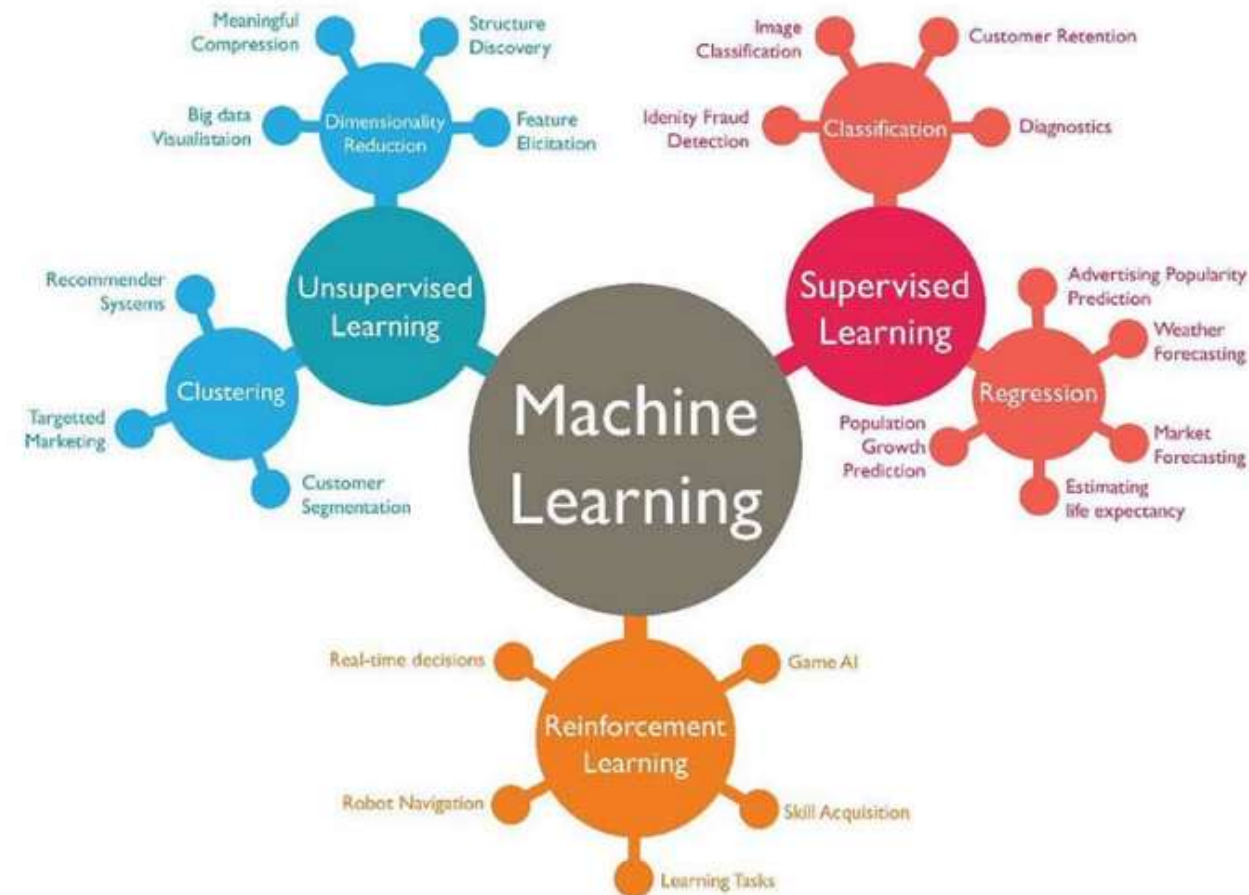


## Generative



# Types of Machine Learning

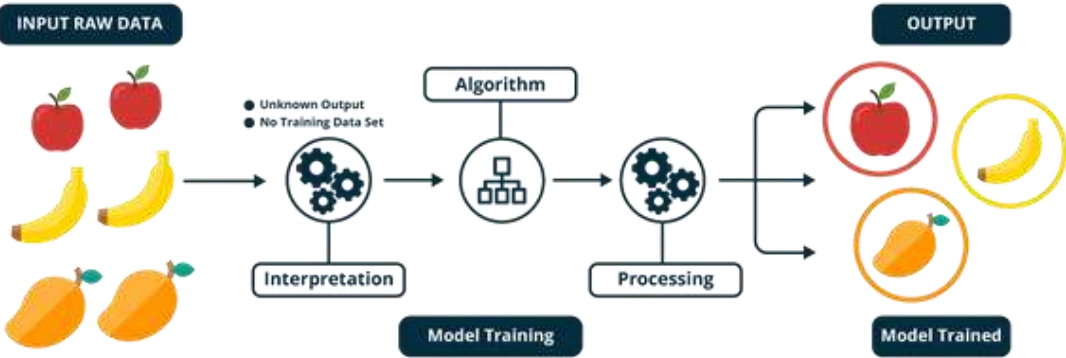
- Supervised learning
  - Given input-output examples  $f(X)=Y$ , learn the function  $f()$ .
- Unsupervised learning
  - Given input examples, find patterns such as clusters
- Reinforcement learning
  - Select and execute an action, get feedback, update policy (what action to do in which state).



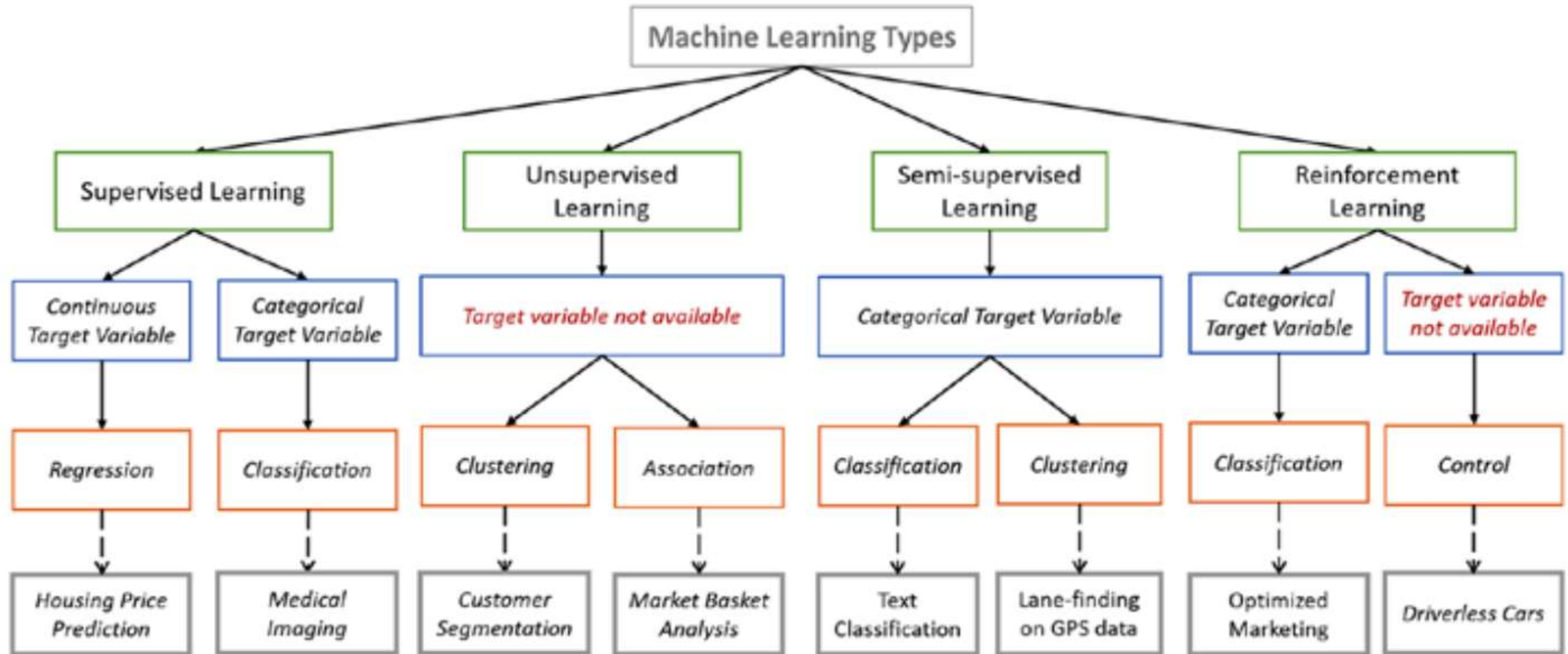


# Supervised learning

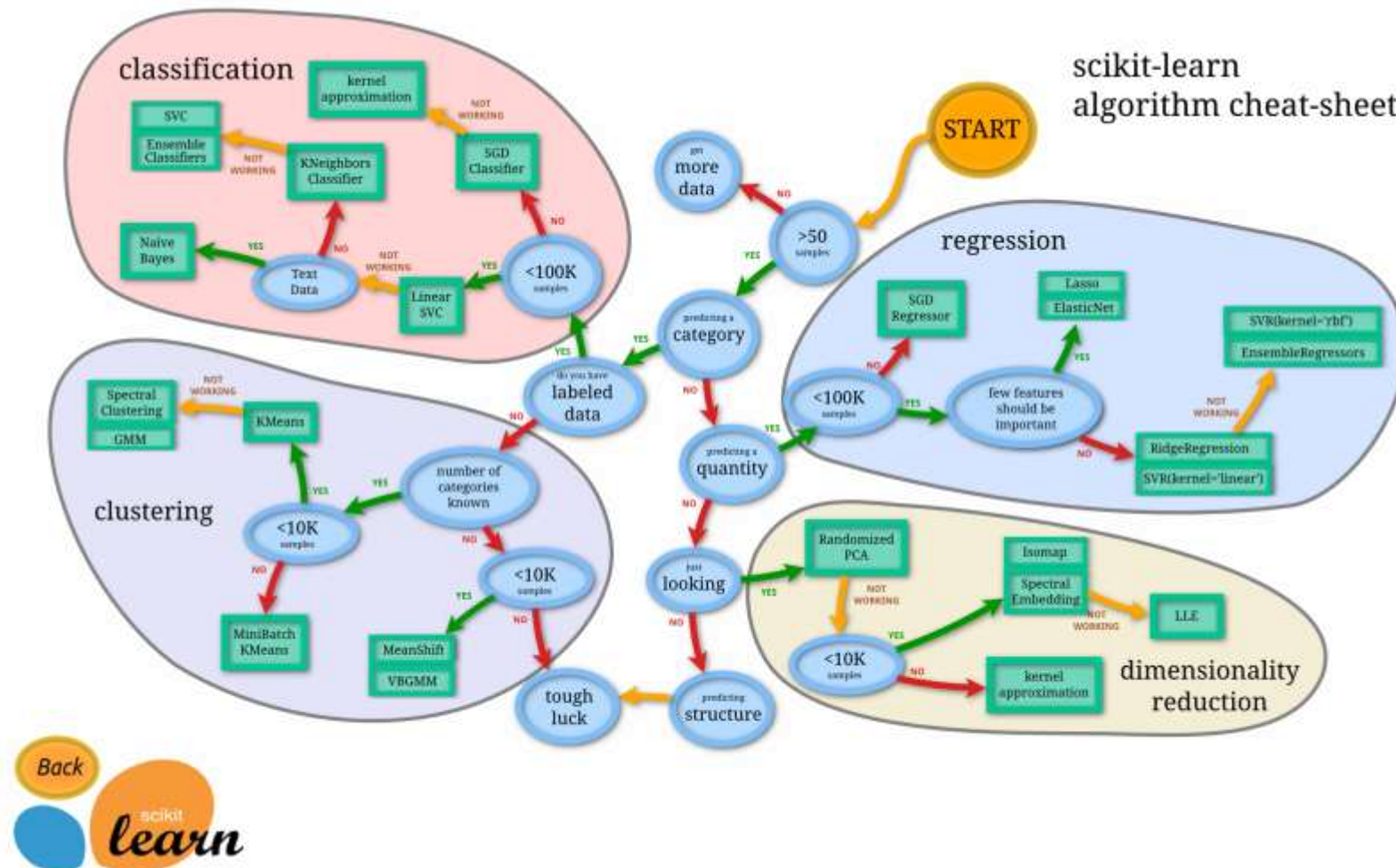
T. Mitchell, M. Jordan:  
“Most of the recent progress in machine learning involves mapping from a set of inputs to a set of outputs.”



INPUT X	OUTPUT Y	APPLICATION
Voice recording	Transcript	Speech recognition
Historical market data	Future market data	Trading bots
Photograph	Caption	Image tagging
Drug chemical properties	Treatment efficacy	Pharma R&D
Store transaction details	Is the transaction fraudulent?	Fraud detection
Recipe ingredients	Customer reviews	Food recommendations
Purchase histories	Future purchase behavior	Customer retention
Car locations and speed	Traffic flow	Traffic lights
Faces	Names	Face recognition



# Map over Methods



# Unsupervised Learning



# Unsupervised Learning at a Glance

## In **unsupervised learning**

- Neither a correct answer/output, nor a reward is given
- Task is to **find some structure** in the data
- Performance metric is some **reconstruction** error of patterns compared to the input data distribution

## Examples:

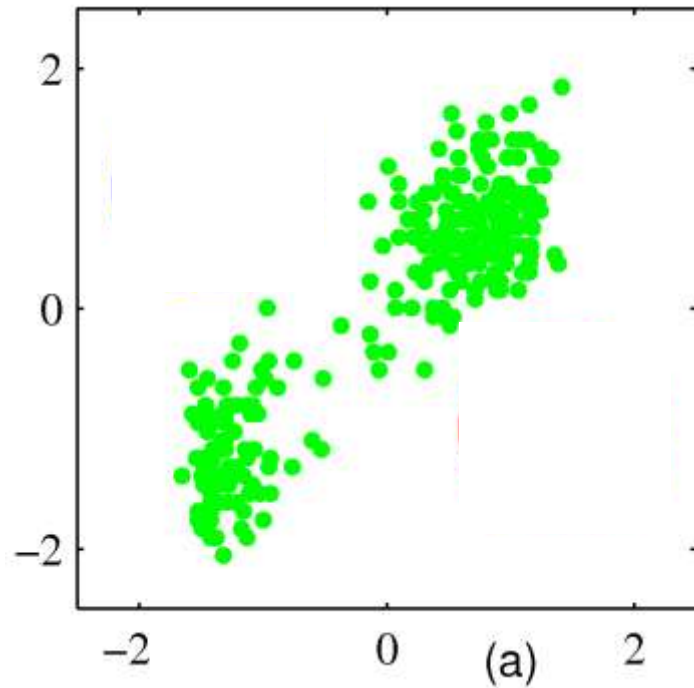
- **Clustering** – When the data distribution is confined to lie in a small number of “clusters” we can find these and use them instead of the original representation, e.g. bigger recommender system (news, ads, etc.)
- **Dimensionality Reduction** – Finding a suitable lower dimensional representation while preserving as much information as possible, e.g. image/video compression

Recent trend: Found structure can be used to **generate new data (content)**!

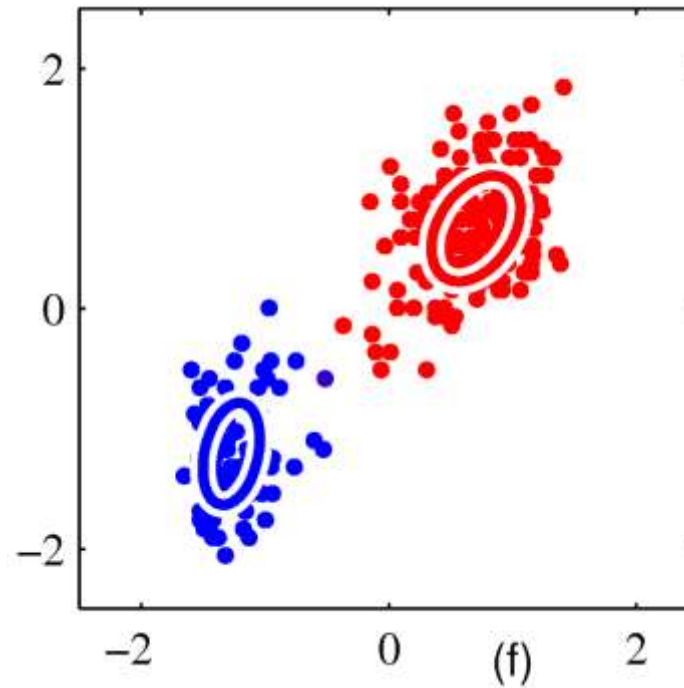
# Unsupervised Learning at a glance II

- Not directly applicable to the agent perspective as there is no clear way to encode a goal or behavior
- However, the techniques can be useful as a **preprocessing step** in other learning approaches
  - If fewer dimensions or a few clusters can accurately describe the data, big **computational wins** can be made
- It is also frequently used for **visualization** as smaller representations are easier to visualize on a computer screen
- To keep this brief, we will not go into any further detail on unsupervised learning

# Unsupervised Learning Example: Clustering - Continuous Data



Two-dimensional continuous input



(Bishop, 2006)

# Unsupervised example

- Original faces were down sampled to save space but still remain majority features.



# Supervised Learning

# Formalizing Supervised Learning

**Remember**, in Supervised Learning:

- Given tuples of **training data** consisting of  $(\mathbf{x}, y)$  pairs
- The objective is to learn to **predict** the **output**  $y'$  for a new input  $\mathbf{x}'$

Formalized as **searching** for approximation to **unknown function**  $y = f(\mathbf{x})$ , given  $N$  examples of  $\mathbf{x}$  and  $y$ :  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$

Two major classes of supervised learning

- **Classification** – Output are **discrete** category labels
  - Example: Detecting disease,  $y = \text{“healthy”}$  or  $\text{“ill”}$
- **Regression** – Output are **numeric** values
  - Example: Predicting temperature,  $y = 15.3$  degrees

In either case, input data  $\mathbf{x}_i$  could be **vector valued** and **discrete, continuous** or **mixed**.  
Example:  $\mathbf{x}_1 = (12.5, \text{“cat”}, \text{true})$ .

# Linear regression with one variable

Living area (x)	Price (y)
2 104	399 900
1 600	329 900
2 400	369 000
1 416	232 000
...	...

$N$  training instances of the form  $(x, y)$

# Linear regression with one variable

- Modelling assumption

The relation between housing area and price can be described in terms of an affine function.

affine = linear function + intercept (bias)

- Learning task

Find the 'best' affine function – the function that minimizes the total distance to the data points.

distance measure: mean squared error



# Linear regression with one variable

A diagram illustrating the linear regression equation  $\hat{y} = xm + b$ . The equation is centered, with labels and leader lines pointing to its components: "input value" points to  $x$ , "slope" points to  $m$ , "bias" points to  $b$ , and "predicted output" points to  $\hat{y}$ . The word "predicted" is positioned above "output" and is underlined.

$$\text{predicted output } \hat{y} = xm + b \text{ bias}$$

input value

slope

# Linear regression with several variables

**$X$**

Living area ( $x_1$ )	# bedrooms ( $x_2$ )	Price ( $y$ )
2 104	3	399 900
1 600	3	329 900
2 400	3	369 000
1 416	2	232 000
...		...

**$y$**

training set = design matrix  $X$ , target vector  $y$

# Linear regression with several variables

The diagram shows the equation  $\hat{y} = Xw + b$  with three annotations: "input matrix (N-by-F)" pointing to  $X$ , "predicted output" pointing to  $\hat{y}$ , and "weight vector (F-by-1)" pointing to  $w$ . The term  $b$  is labeled as "bias".

input matrix (N-by-F)

predicted output  $\hat{y} = Xw + b$  bias

weight vector (F-by-1)

N = number of training examples, F = number of features (independent variables)

# Learning problem

- Choose parameters such that the total distance between the corresponding hyperplane and the data points is minimal.  
as measured by mean squared error
- This problem has an exact solution that can be found using the method of least squares.
- An inexact (numerical) but more general method to solve the problem is to use gradient descent.

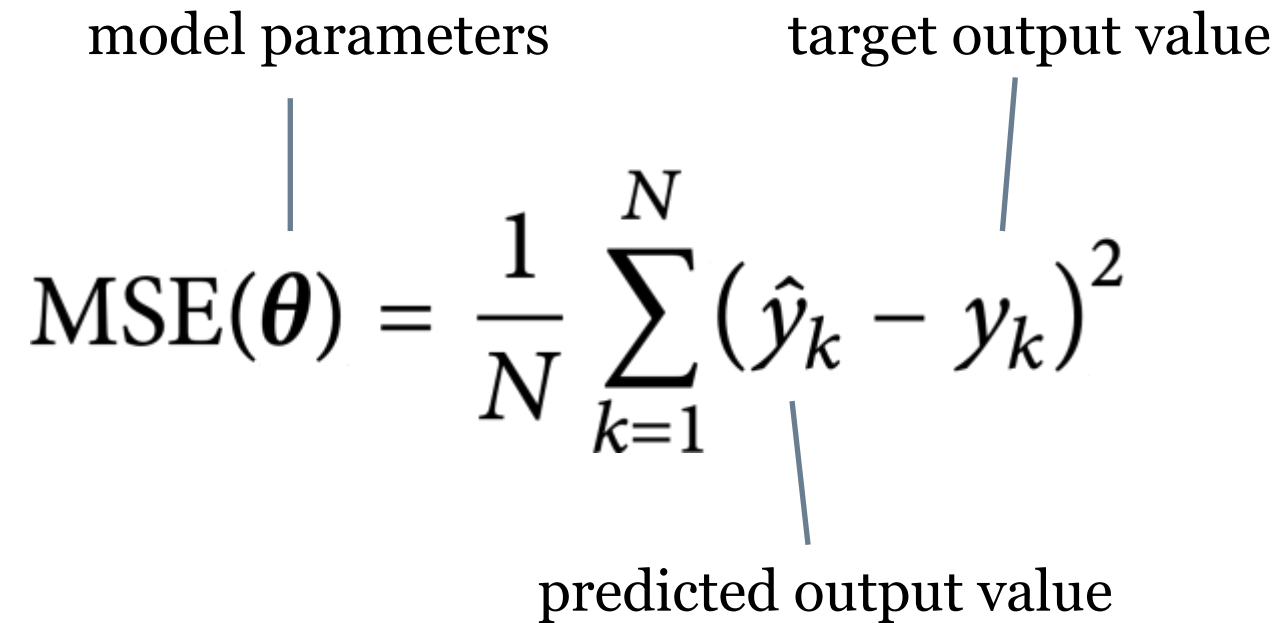
# Mean squared error (L2)

model parameters

target output value

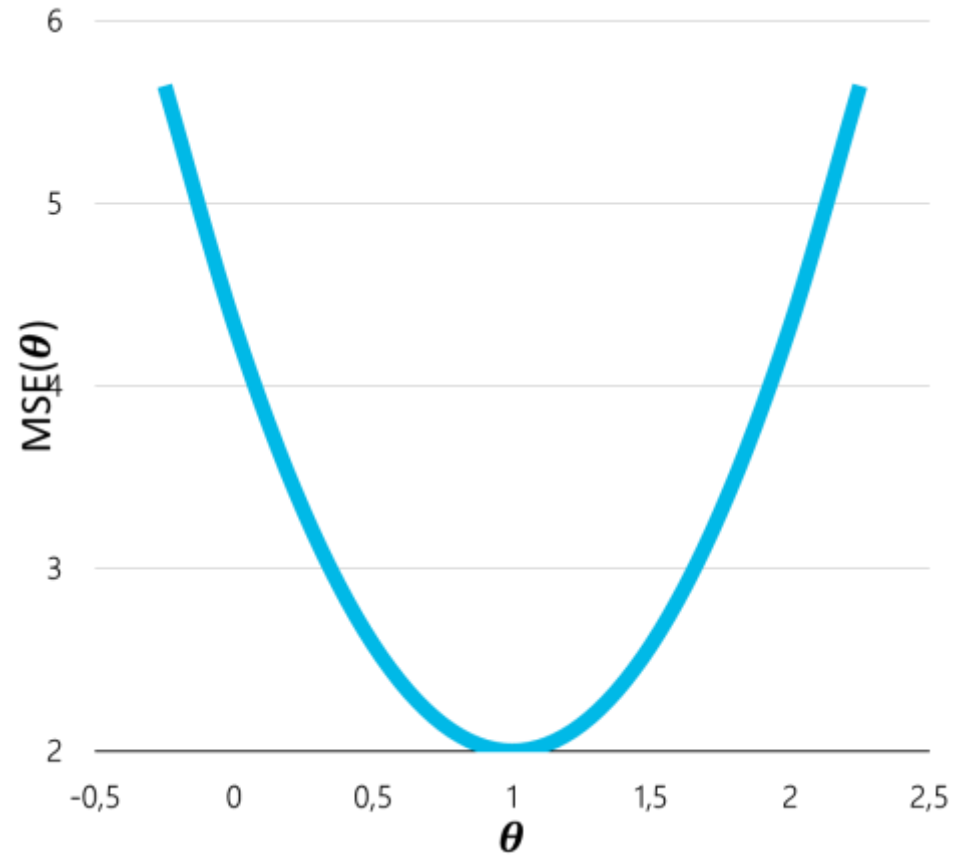
$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^N (\hat{y}_k - y_k)^2$$

predicted output value

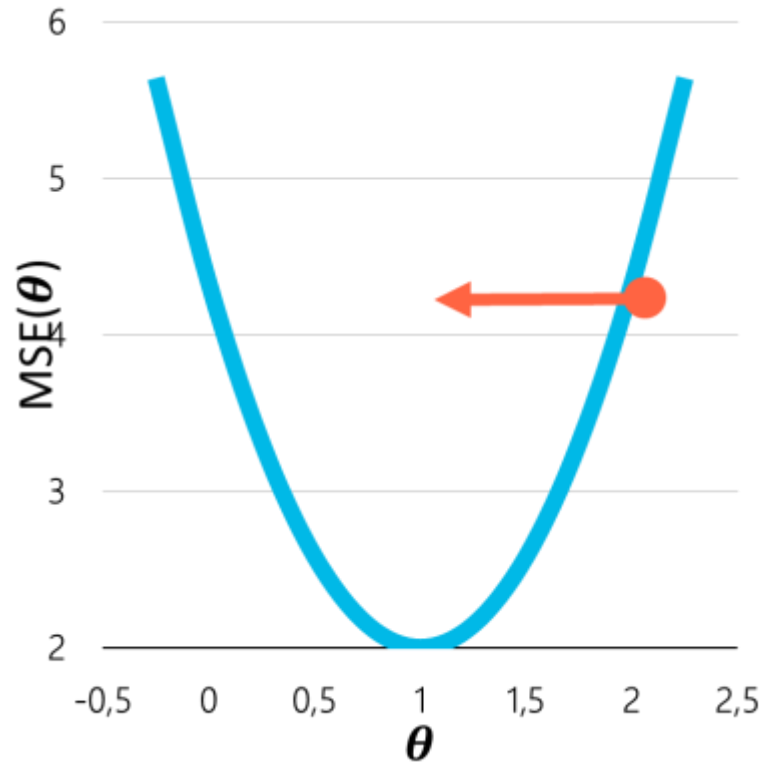


Later, it will be convenient to divide by  $2N$  instead of by  $N$ .

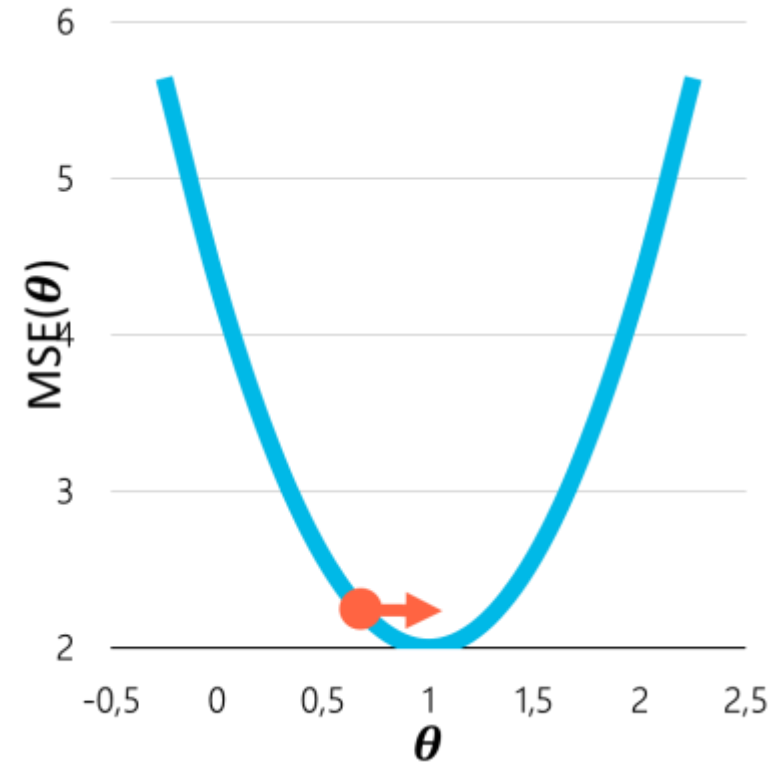
# Mean squared error (L2)



# Gradient descent: Intuition

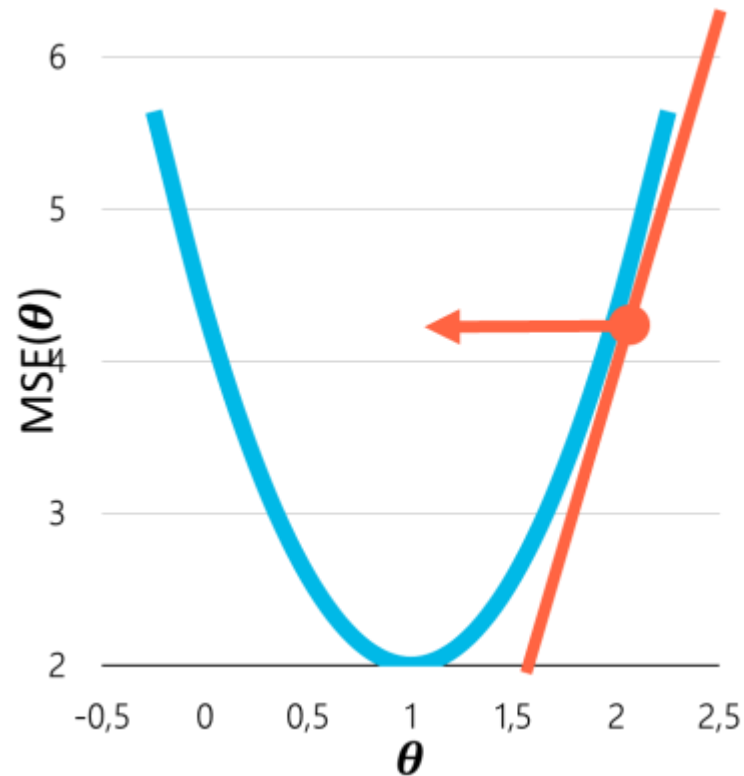


$\theta := \theta - \text{large value}$

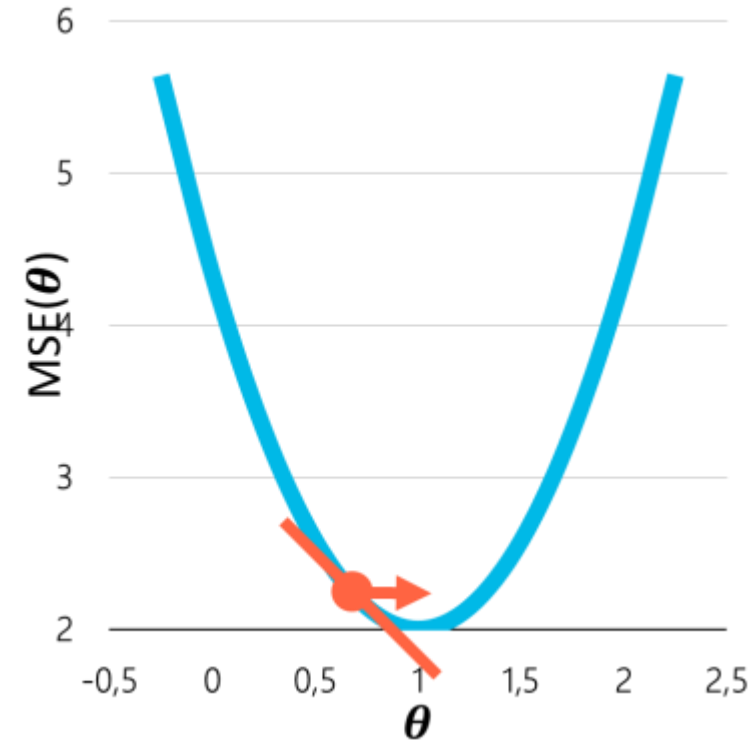


$\theta := \theta + \text{small value}$

# Gradient descent: Intuition



$$\theta := \theta - \nabla MSE(\theta)$$



$$\theta := \theta - \nabla MSE(\theta)$$



# Gradient descent

‘Follow the gradient into the valley of error.’

- Step 0: Start with an arbitrary value for the parameters  $\theta$ .
- Step 1: Compute the gradient of the loss function,  $\nabla L(\theta)$ .
- Step 2: Update the parameters  $\theta$  as follows:  $\theta := \theta - \alpha \nabla L(\theta)$

The parameter  $\alpha$  is the learning rate.

- Repeat step 1–2 until the error is sufficiently low.

# Gradient update rule for linear regression

Substituting L2 loss into the generic update rule, we get

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\alpha}{N} \sum_{i=1}^N \mathbf{x}_i^\top (\mathbf{x}_i \boldsymbol{\theta} - y_i)$$

This can be more succinctly expressed as

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\alpha}{N} \mathbf{X}^\top (\mathbf{X} \boldsymbol{\theta} - \mathbf{y})$$

# Stochastic gradient descent

- For large training sets, even taking a single step in the gradient descent algorithm will take a lot of time.
- The idea behind stochastic gradient descent (SGD) is to estimate the gradient by computing it on a small set of samples.

```
def minibatches(x, y, batch_size):  
    random_indices = np.random.permutation(np.arange(x.shape[0]))  
    for i in range(0, x.shape[0] - batch_size + 1, batch_size):  
        batch_indices = random_indices[i:i+batch_size]  
        yield x[batch_indices], y[batch_indices]
```

# The unreasonable effectiveness of SGD

- Stochastic gradient descent (SGD) provides a very general framework for pushing model parameters towards small loss.
- SGD and its variants are probably the most widely used optimization algorithms for deep learning.
- What is crucial for the applicability of SGD is that both the model function and the loss function are differentiable.

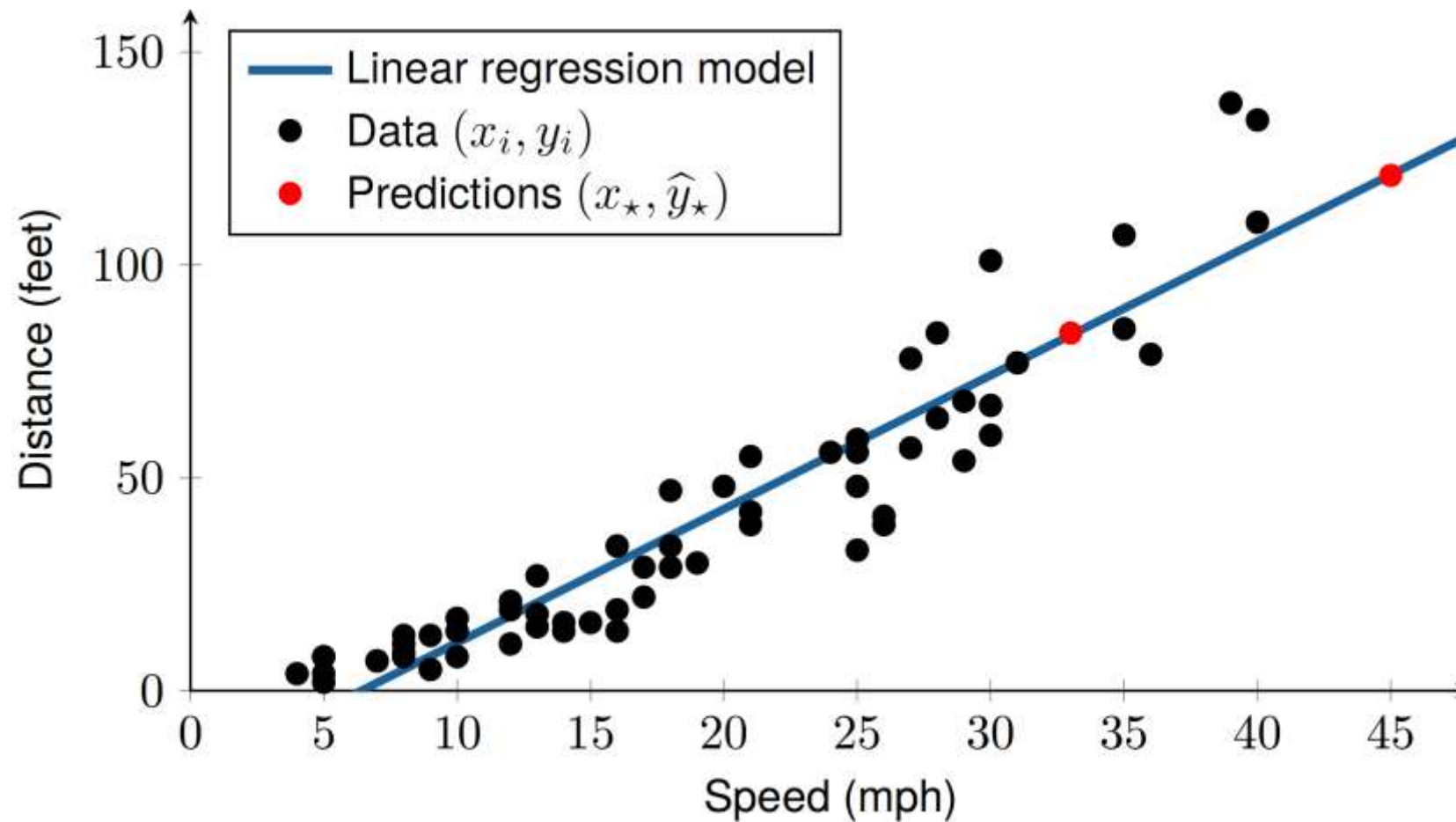
One can sometimes use sub-gradient methods.

# The unreasonable effectiveness of SGD

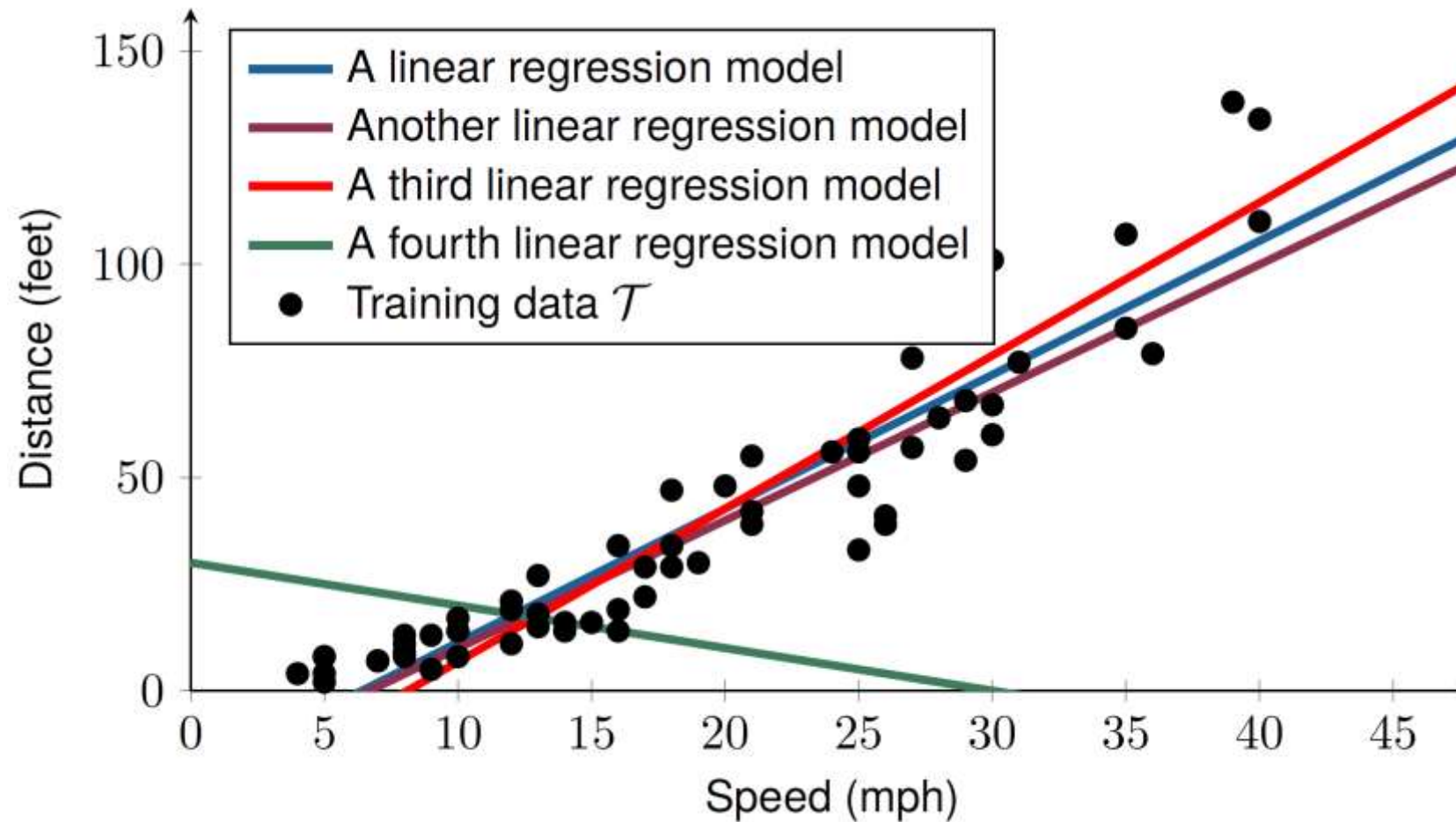
- In the case of convex loss functions such as mean squared error, gradient descent is guaranteed to find an optimal solution.
- Most loss functions are not convex, and there is no guarantee that gradient descent will arrive at even a local minimum.
- In practice however, neural networks work surprisingly well when trained with stochastic gradient descent.

momentum, parameter initialisation, regularisation, ...

# Linear Regression



# Linear Regression



# Linear Regression

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma_{\varepsilon}^2 \mathbf{I}).$$

Assumptions (for now):

1.  $\mathbf{y}$  – observed **random** variable.
2.  $\boldsymbol{\theta}$  – unknown **deterministic** variable.
3.  $\mathbf{X}$  – known **deterministic** variable.
4.  $\boldsymbol{\varepsilon}$  – unknown **random** variable.
5.  $\sigma_{\varepsilon}$  – unknown/known **deterministic** variable.

The least squares problem

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2$$

is solved by the **normal equations**

$$\mathbf{X}^T \mathbf{X} \hat{\boldsymbol{\theta}} = \mathbf{X}^T \mathbf{y}.$$



# Linear Regression

**The linear regression model**

$$y = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p + \varepsilon$$

+

**Maximum likelihood**

$$\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2) \text{ iid}$$

**Our first  
learning tool**

# Linear Regression Example

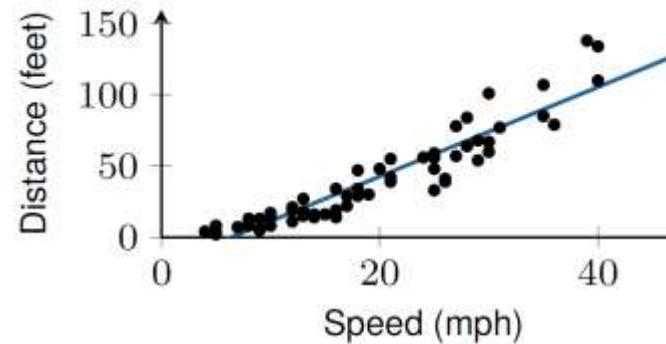
- $x$  = Speed
- $y$  = Distance

$$y = \theta_0 + \theta_1 x + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 1 & 4 \\ 1 & 5 \\ 1 & 5 \\ 1 & 5 \\ 1 & 5 \\ 1 & 5 \\ 1 & 7 \\ 1 & 7 \\ 1 & 7 \\ 1 & 8 \\ \vdots & \vdots \\ 1 & 39 \\ 1 & 39 \\ 1 & 40 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 4 \\ 2 \\ 4 \\ 8 \\ 8 \\ 7 \\ 7 \\ 7 \\ 8 \\ \vdots \\ 138 \\ 110 \\ 134 \end{bmatrix}$$

The normal equations  $\Rightarrow \hat{\theta} = \begin{bmatrix} -20.1 \\ 3.1 \end{bmatrix}$



Use the model for predictions!

# Linear Regression Example

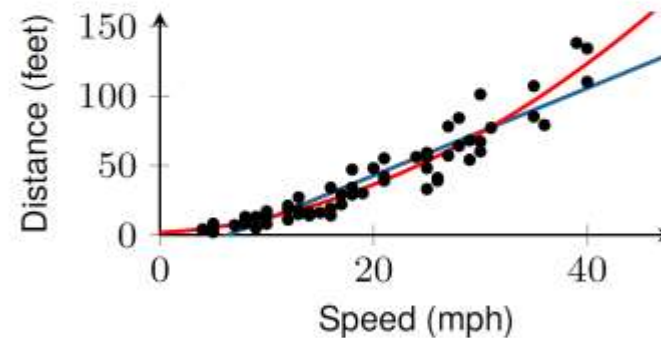
- $x$  = Speed
- $y$  = Distance

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \varepsilon, \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 1 & 4 & 16 \\ 1 & 5 & 25 \\ 1 & 5 & 25 \\ 1 & 5 & 25 \\ 1 & 5 & 25 \\ 1 & 7 & 49 \\ 1 & 7 & 49 \\ 1 & 8 & 64 \\ \vdots & \vdots & \vdots \\ 1 & 39 & 1521 \\ 1 & 39 & 1521 \\ 1 & 40 & 1600 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 4 \\ 2 \\ 4 \\ 8 \\ 8 \\ 7 \\ 7 \\ 8 \\ \vdots \\ 138 \\ 110 \\ 134 \end{bmatrix}$$


The normal equations  $\Rightarrow \hat{\theta} = \begin{bmatrix} 1.58 \\ 0.42 \\ 0.066 \end{bmatrix}$



# Classical Supervised Learning in Practice

Can be seen as **searching** for an approximation to unknown function  $y = f(\mathbf{x})$  given  $N$  examples of  $\mathbf{x}$  and  $y$ :  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$

Want the algorithm to **generalize** from **training** examples to new inputs  $\mathbf{x}'$ , so that  $y' = f(\mathbf{x}')$  is “close” to the correct answer

- 
1. An **input “feature” vector**  $\mathbf{x}_i$  of examples is constructed by **mathematically encoding** relevant problem data
    - Examples of such  $(\mathbf{x}_i, y_i)$  make up the **training set**
  2. A *model (or hypothesis)* for  $f(x)$  is selected with some parameters
  3. A *loss function* is selected that defines “closeness” to correct answers
  4. The model is **trained** on the examples by searching for its **parameters** that minimize loss on the training set (i.e. are “close” to unknown  $f(x)$ )

# Feature Vector Construction

Want to learn  $f(\mathbf{x}) = y$  given  $N$  examples of  $\mathbf{x}$  and  $y$ :  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$

Most standard algorithms work on **real number variables**

- If inputs  $\mathbf{x}$  or outputs  $y$  contain categorical values like “book” or “car”, we need to encode them with numbers
  - With only two classes we get  $y$  in  $\{0,1\}$ , called **binary classification**
  - Classification into multiple classes can be reduced to a sequence of binary one-vs-all classifiers
- The variables may also be structured as **text**, **audio**, **image** or **video** data

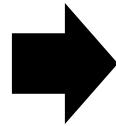
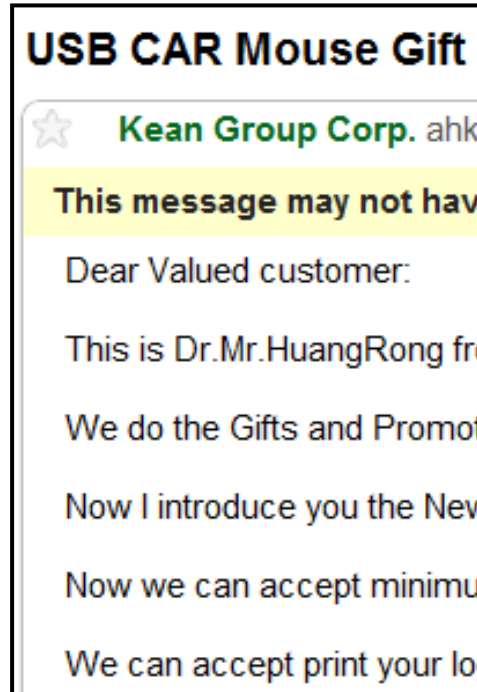
Finding a suitable feature representation **can be non-trivial**, but there are standard approaches for the common domains

- With sufficient data, features can also be learned (*deep learning*, later...)

# Feature Vector Example for Text - Bag of Words

One of the early successes of ML was **learning spam filters**

Spam classification example:



Each mail is an input, some mails are flagged as spam or not spam to create training examples.

**Bag of Words Feature Vector:**

Encode the existence of a fixed set of relevant **key words** in each mail as the feature vector.

Feature	Exists?
"Customer"	1 (Yes)
"Dollar"	0 (No)
"Fund"	0
"Accept"	1
"Bank"	0
....	...

$\mathbf{x}_i = \text{words}_i =$

$y_i = 1$  (spam) or  $0$  (not spam)

Simply learn  $f(\mathbf{x})=y$  using suitable classifier!

# Selecting Models: Linear Regression Example

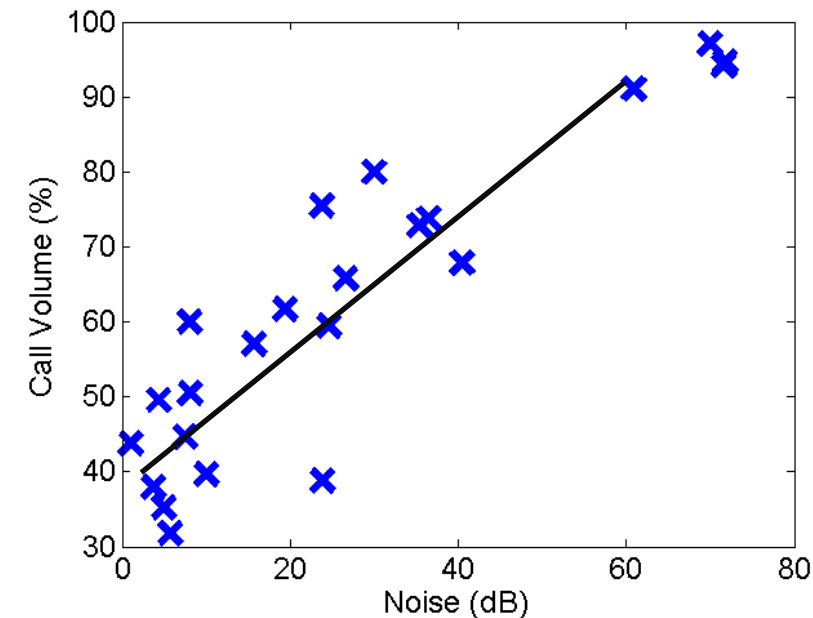
- I. Construct a **feature vector**  $\mathbf{x}_i$  to be used with examples of  $y_i$
- II. Select a model and **train** it on examples (search for a good approximation to the unknown function)

Fictional example: Smartphone app that learns desired ring volume based on examples of volume and background noise level

Feature vector  $\mathbf{x}_i = (\text{Noise dB})$ ,  $y_i = (\text{Volume } \%)$

- Select the family of **linear** functions:  $y_i = w_1 \cdot x_i + w_0$
- **Train** the algorithm by searching for a line that **fits the data well**

...but how does "training" really work?



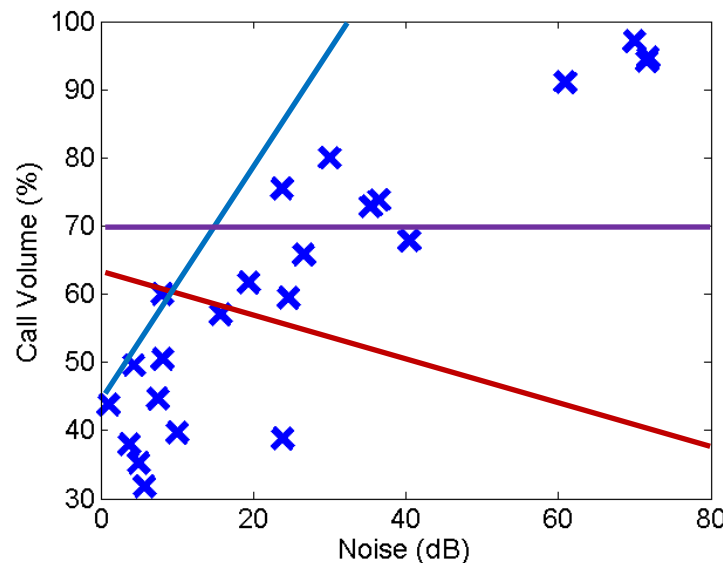
# Training a Learning Algorithm

Feature vector  $\mathbf{x}_i$  = (Noise in dB), outputs  $y_i$  = (Volume %)

- Recap: Want to find approximation  $h(x)$  to the unknown function  $f(x)$
- As an example, let it to be the family of **linear** functions:

$$y_i = w_1 \cdot x_i + w_0$$

- The model  $h_{\mathbf{w}}(x)$  has two **parameters**:  $\mathbf{w} = (w_1, w_0)$  (line slope and offset)
- How do we find parameters that result in a **good** approximation  $h$ ?



Three poor  
linear  
hypotheses



# Training a Learning Algorithm – Loss Functions

How do we find parameters  $\mathbf{w}$  that result in a **good** approximation  $h_{\mathbf{w}}(x)$ ?

- Need a performance metric for function approximations of unknown  $f(x)$ 
  - **Loss functions**  $L(f(x), h_{\mathbf{w}}(x))$
- Minimize deviation against the  $N$  example data points from  $f(x)$ 
  - For **regression** one common choice is a **sum square loss** function:

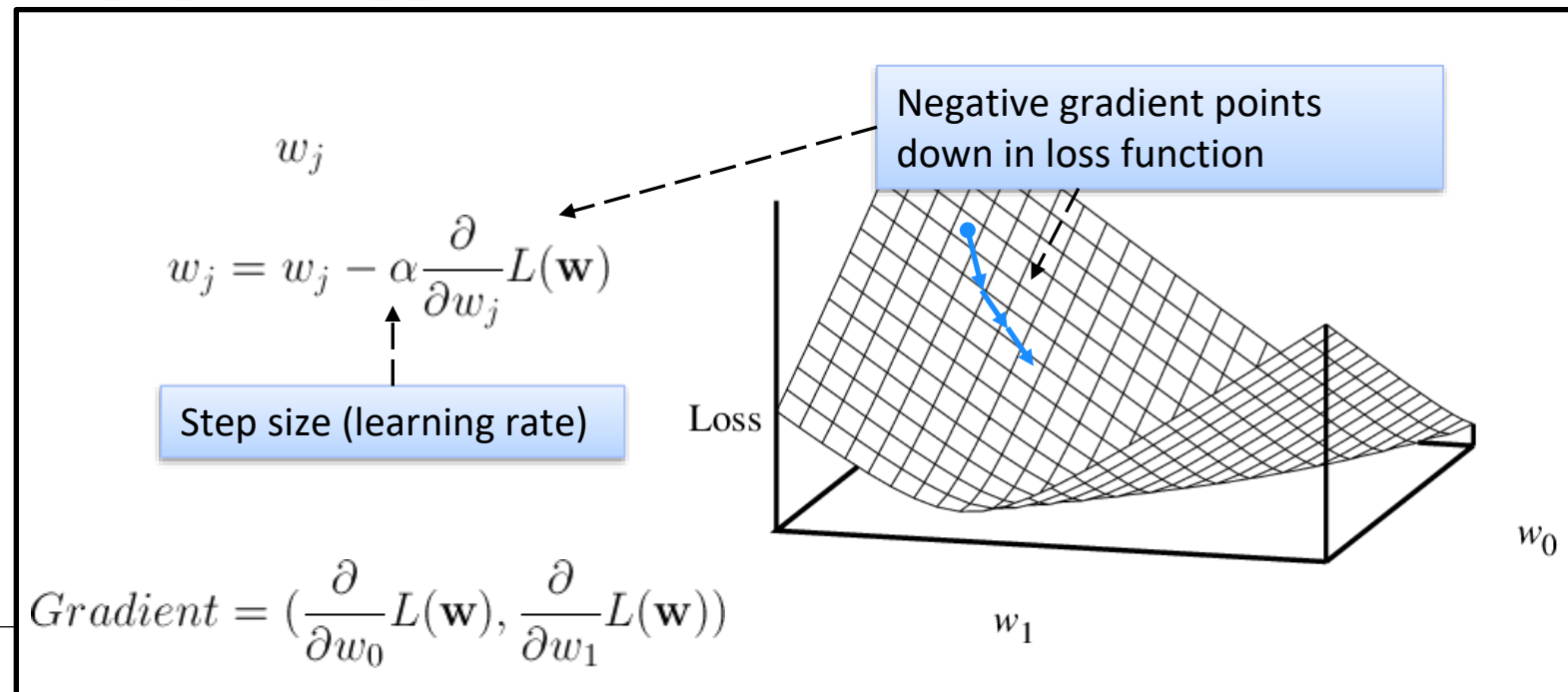
$$L(f(x), h_{\mathbf{w}}(x)) = (f(x) - h_{\mathbf{w}}(x))^2 = \sum_{i=1}^N (y_i - h_{\mathbf{w}}(x_i))^2$$

- Why square loss? Negative difference is as bad as positive
- Search in continuous domains like  $\mathbf{w}$  is known as **optimization**
  - (if unfamiliar, see Ch4.2 Local Search in Continuous Spaces in course book AI: A Modern Approach)

# Training a Learning Algorithm – Optimization

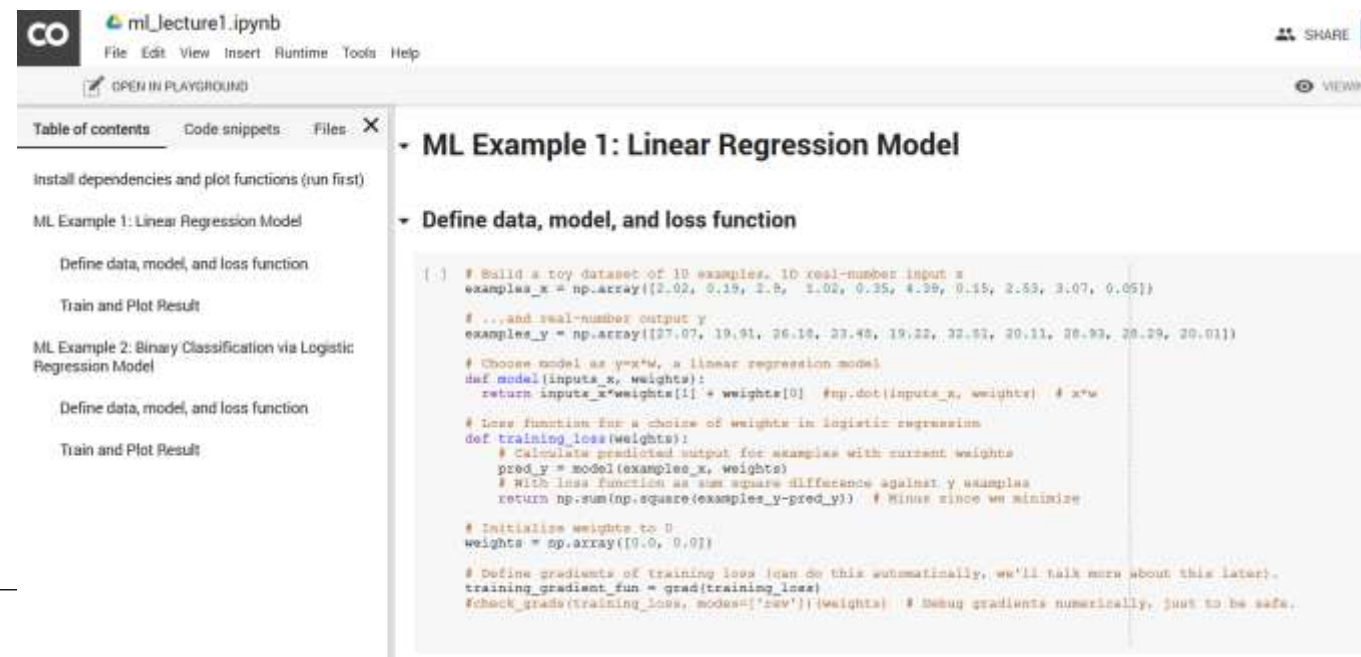
How do we find parameters  $\mathbf{w}$  that minimize the loss?

- Optimization approaches iteratively move in the **direction that decreases** the loss function  $L(\mathbf{w})$
- Simple and popular approach: **gradient descent**



# Worked Example – Linear Regression

- Google Colab at: <http://bit.ly/2maVQKY>
  - Run top box to install dependencies (30s), then scroll to ML Example 1
- NOTE: Need to be signed in to a Google account. *Might* need to **save** or download workbook to run it.



```
[.] # Build a toy dataset of 10 examples. 10 real-number input x
examples_x = np.array([2.02, 0.19, 2.8, 1.02, 0.35, 4.39, 0.15, 2.53, 3.07, 0.05])

# ...and real-number output y
examples_y = np.array([27.07, 19.91, 26.19, 23.48, 19.22, 32.33, 20.11, 28.93, 28.29, 20.01])

# Choose model as y=x*w, a linear regression model
def model(inputs_x, weights):
    return inputs_x*weights[1] + weights[0] #np.dot(inputs_x, weights) # x*w

# Loss function for a choice of weights in logistic regression
def training_loss(weights):
    # Calculate predicted output for examples with current weights
    pred_y = model(examples_x, weights)
    # With loss function as sum square difference against y examples
    return np.sum(np.square(examples_y-pred_y)) # Minus since we minimize

# Initialize weights to 0
weights = np.array([0.0, 0.0])

# Define gradients of training loop (can do this automatically, we'll talk more about this later).
training_gradient_fun = grad(training_loss)
#check_grads(training_loss, modes=['rev'])(weights) # Check gradients numerically, just to be safe.
```

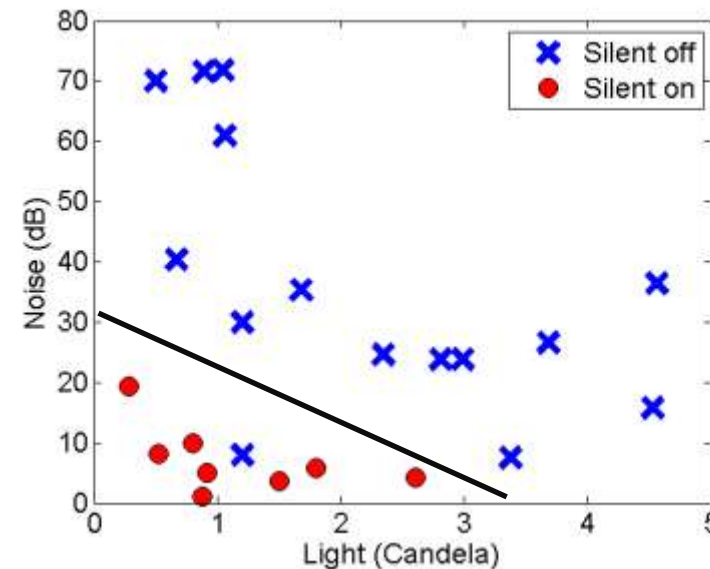
# What about categorical outputs (classification)?

- I. Construct a **feature vector**  $\mathbf{x}_i$  to be used with examples of  $y_i$
- II. Select a model and **train** it on examples (search for a good approximation to the unknown function)

Fictional example: Smartphone app that learns if silent mode should be on/off at different levels of **background noise** and **light**

Feature vector  $\mathbf{x}_i = (\text{Noise}, \text{Light level})$ ,  $y_i = \{\text{"silent on"}, \text{"silent off"}\}$

- Again, can select the family of **linear** functions. However, now outputs  $y$  have to be **transformed** to the interval  $[0,1]$
- Can classify new inputs according to how close output is to 0 or 1.
- For linear models, the decision boundary will still be a straight line.



# Classifier Training – Loss Functions II

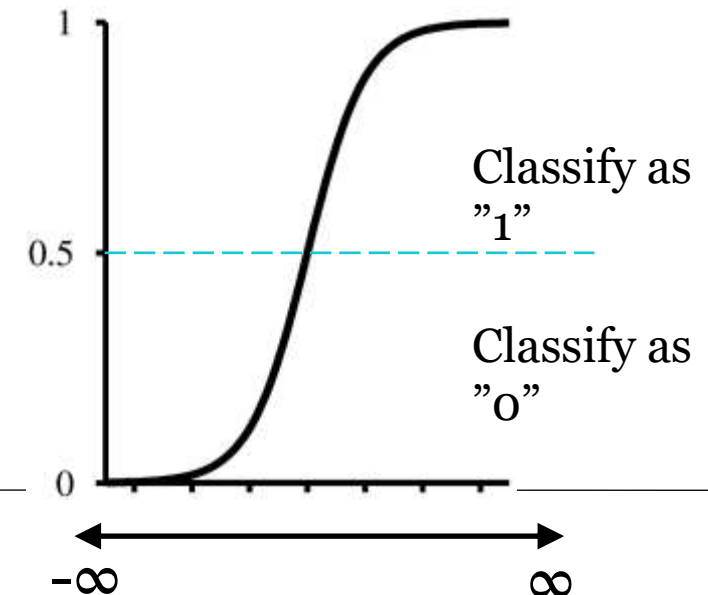
- How to transform standard models to classification?
  - Squared error does not make sense when target output discrete set  $\{0,1\}$
- Could use custom loss functions for classification
  - Minimize number of missclassifications (unsmooth w.r.t. parameter changes)
  - Maximize information gain (used in decision trees, see book)
  - **However**, requires specialized parameter search methods
- Instead: Make outputs **probabilities [0,1]** by **squashing** predicted **numeric outputs** via sigmoid ("S")

Sigmoid functions allow us to use any regression model with binary classification by def.  $\Pr(y="1"|X) = g(\text{model}(x))$

Where  $g$  is **"logistic" sigmoid** :

$$g(x) = \frac{1}{1 + e^{-x}}$$

For >2 classes, use **soft-max** (see book)



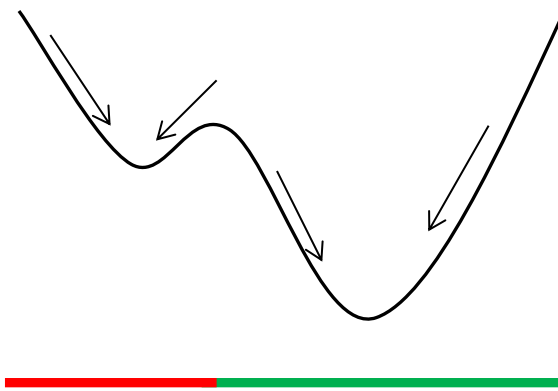
# Worked Example - Binary Classification via Linear Logistic Regression

- Same Google Colab as before: <http://bit.ly/2maVQKY>
  - Run top box to install dependencies (30s), then scroll down to ML Example 2
- NOTE: May need to be signed in to a Google account

# Training a Learning Algorithm – Limitations

## Limitations

- Local optimization of loss is greedy – Gets stuck in *local* minima unless the loss function is **convex** w.r.t.  $\mathbf{w}$ , i.e. there is **only one minima**.
- **Linear models are convex**, however most more advanced models are **vulnerable** to getting stuck in local minima.
- Care should be taken when training such models by using for example **random restarts** and picking the least bad minima.



If we happen to start in red area, optimization will get stuck in a bad local minima!

# Linear Models in Summary

## Advantages

- Linear algorithms are **simple and computationally efficient**
  - For both regression and classification
- Training them is a **convex** optimization problem, i.e. one is guaranteed to find the **best** hypothesis in the space of linear hypothesis
- Can be extended by non-linear feature transformations

## Disadvantages

- The hypothesis space is very restricted, it cannot handle non-linear relations well

Still **widely used** in applications

- Recommender Systems – Initial Netflix Cinematch was a linear regression, before their **\$1 million** competition to improve it. Rather simple and are appropriate for small systems.
- Often a good place to start...
- At the core of many big internet services. Ad systems at Twitter, Facebook, Google etc...



# What about models with uncertainty?

## Supervised Learning:

Mathematically, can be seen as finding an approximation to an unknown function  $y = f(\mathbf{x})$  given  $N$  examples of  $\mathbf{x}$  and  $y$

Two perspectives:

- **Deterministic Models**

- Search for a suitable function  $y = h(\mathbf{x})$
- What we have looked at so far, the most common approach
- Example: In classification something may be either A or B, never in-between, regression gives an exact answer like 15.3

- **Probabilistic Models**

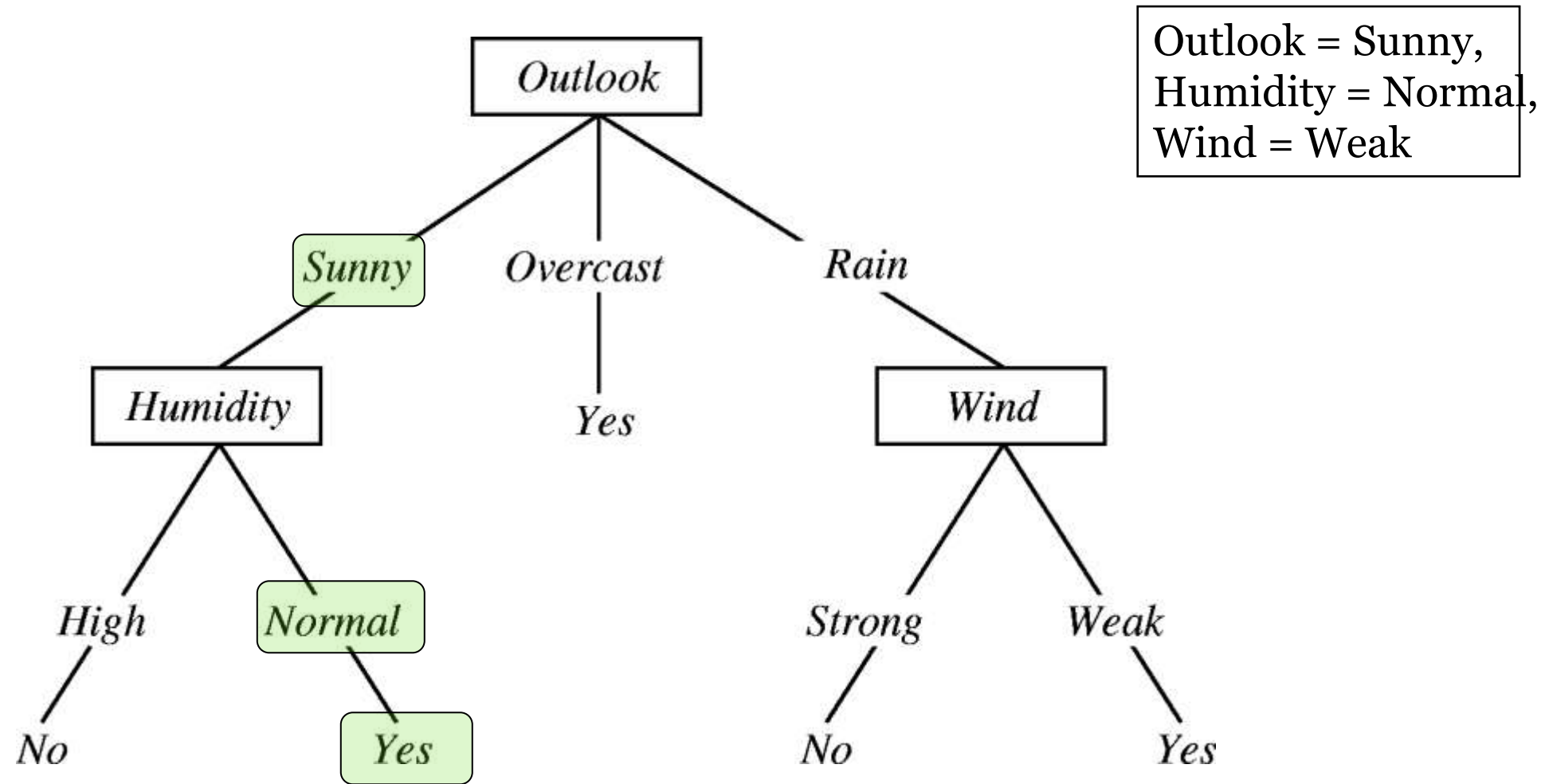
- Search for a suitable **probability distribution** like  $P(Y|X)$
- When we also want to predict the **uncertainty**
- Example:  $P(Y=\text{"Healthy"}|X) = 0.7$  and  $P(Y=\text{"Cancer"}|X) = 0.3$
- In a spam filter we might prefer to get a spam too many than to trash that important mail from your boss...

# Supervised Learning - Decision Tree Learning

# Decision Tree Learning

<b>Day</b>	<b>Outlook</b>	<b>Temp</b>	<b>Humidity</b>	<b>Wind</b>	<b>Play</b>
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes

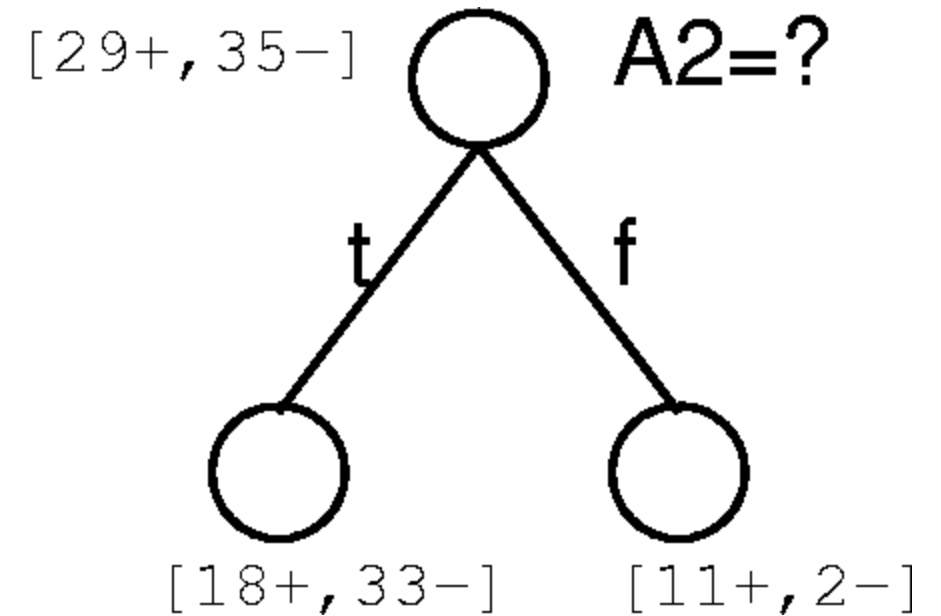
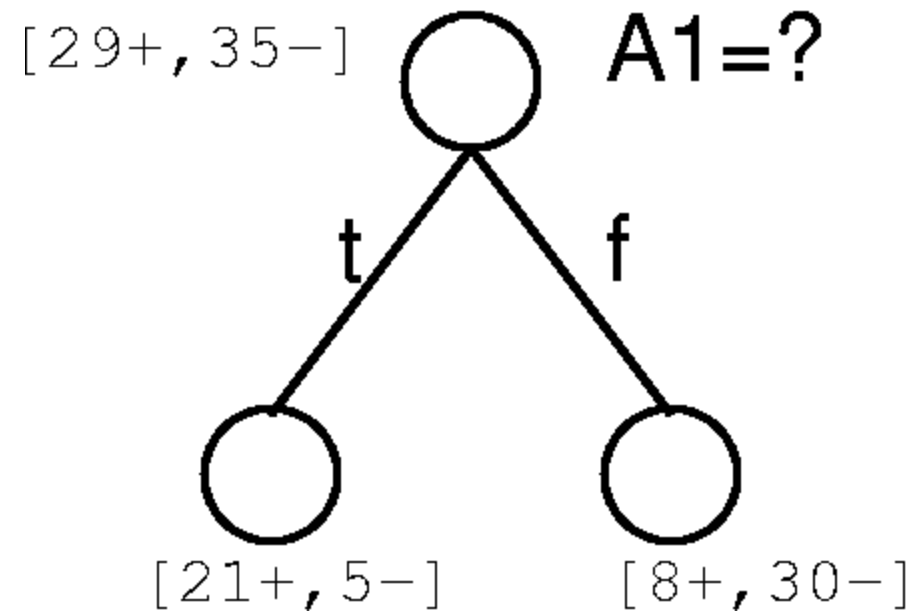
# Example



# Top-Down Induction of DT

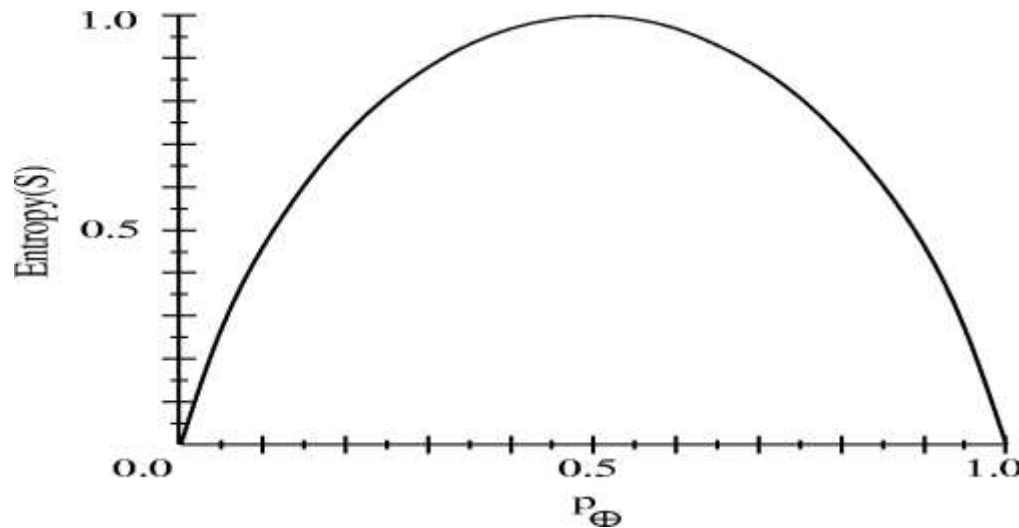
1.  $A :=$  the "best" decision attribute for next *node*
2. Assign  $A$  as decision attribute for *node*
3. For each value of  $A$ , create new descendant of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, then stop, else iterate over the new leaf nodes

# Selecting the "Best" Attribute



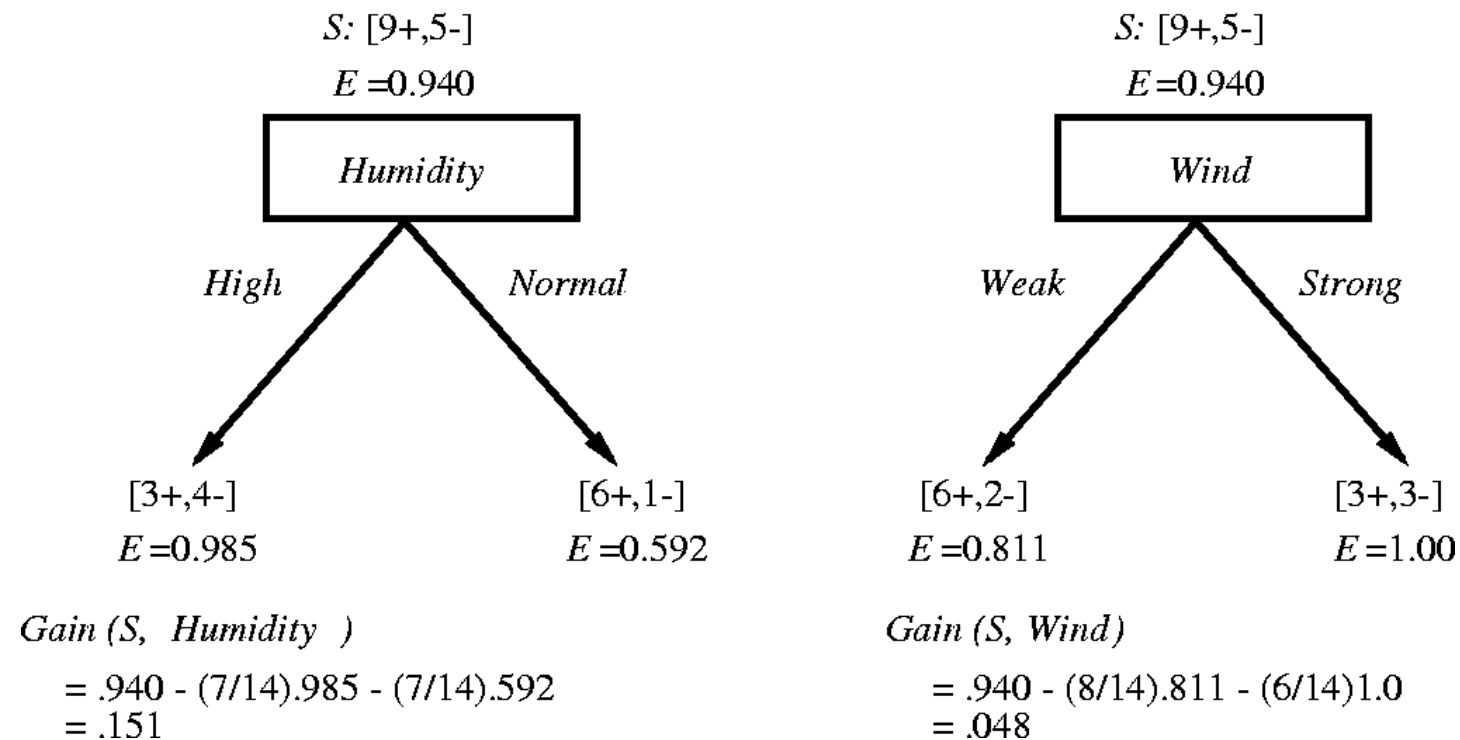
# Information Gain

- Entropy measures the impurity of a sample of training examples  $S$ :  
 $Entropy(S) := \sum_v -p_v \log_2 p_v$
- $Gain(S,A)$  the expected reduction in entropy due to sorting on  $A$ :  
 $Gain(S,A) := Entropy(S) - \sum_{v \in A} (Entropy(S_v) |S_v| / |S|)$



# Selecting the "Best" Attribute

**Which attribute is the best classifier?**







# Problems With ID3

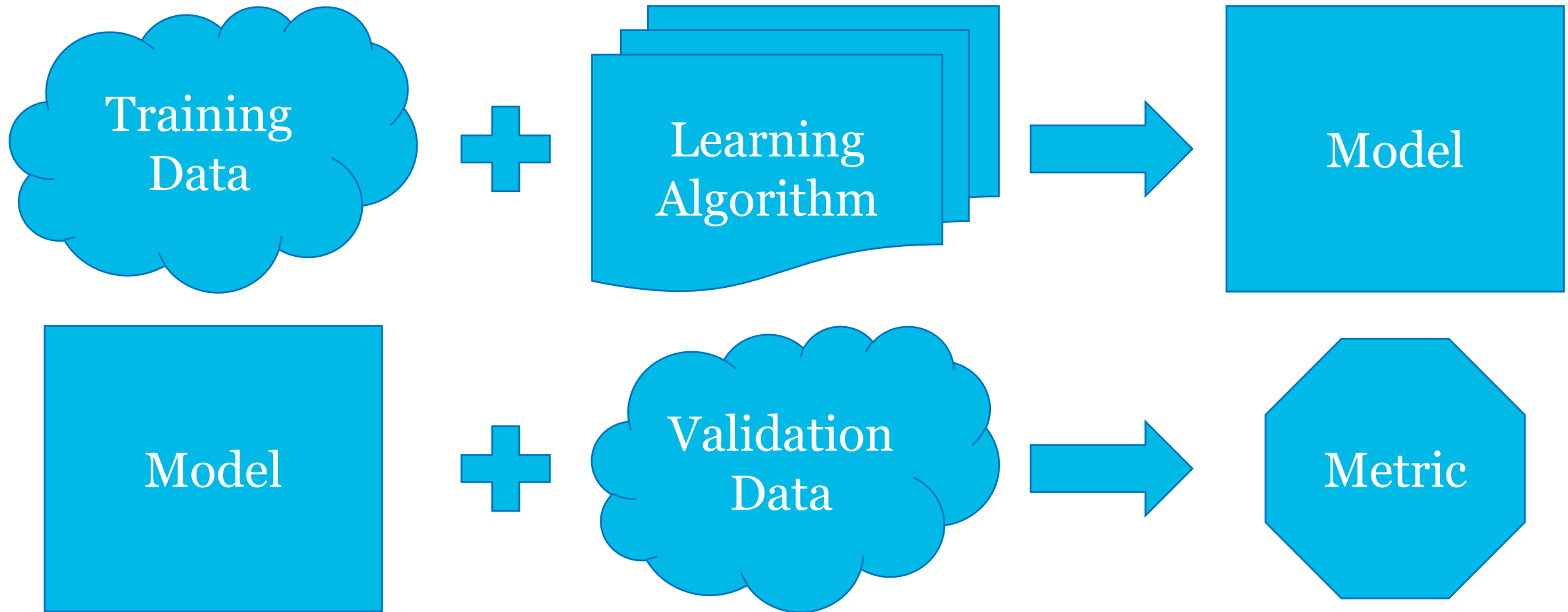
- Overfitting often occur
  - Rules post pruning
    - Convert tree into set of rules, one rule for each path.
    - Prune each rule by removing conditions that result in a reduction of the estimated error.
    - Sort the rules by their estimated accuracy.
- Information gain not always suitable
- Only nominal attributes can be used
- Missing values have to be handled

# When to Consider Decision Trees

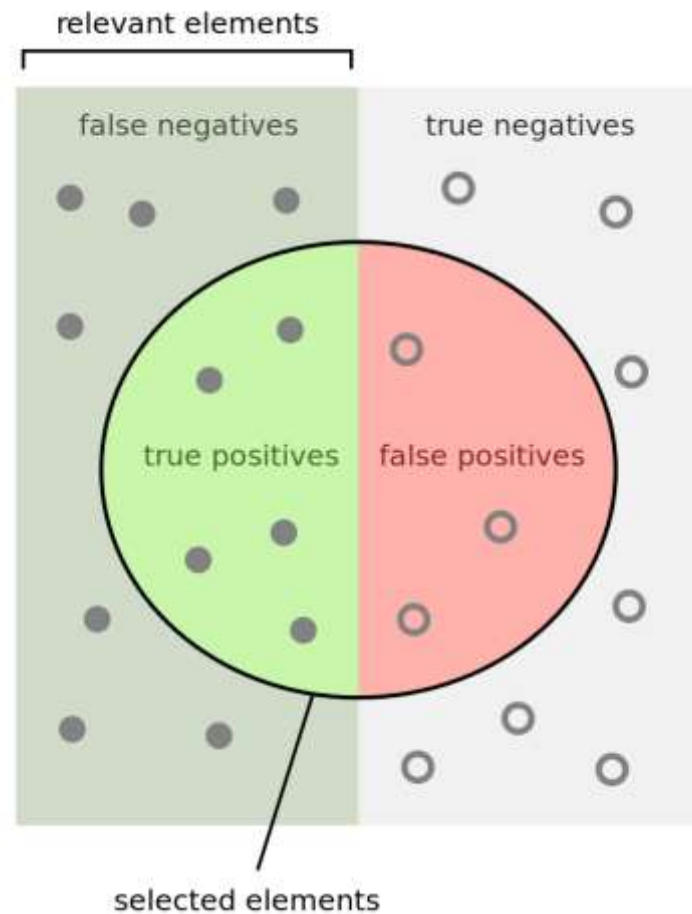
- Instances described by attribute-value pairs
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possible noisy training data

# Evaluating Machine Learning Methods

# Training, Validation, and Test Data



# Precision and Recall



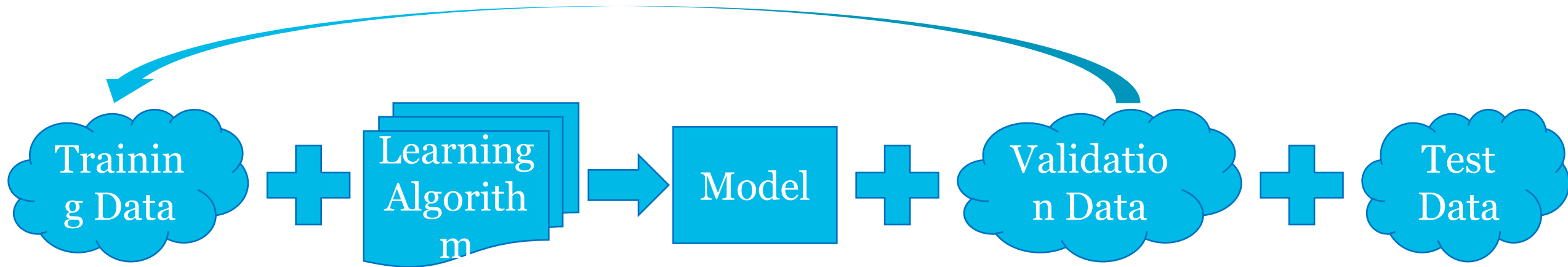
How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

# Machine Learning Process



# Training error and generalization error

- We train a model by minimizing its error on the training data.

optimization

- The training error is different from the generalization error – the expected value of the error on previously unseen inputs.
- We can estimate the generalization error of a model by measuring its test error – the error on a held-out test set.

held-out = not seen during training



# How can we hope to perform well on the test set?

- Assumption 1: The examples in the training set and the test set are mutually independent.
- Assumption 2: The examples in the training set and the test set are identically distributed.

sampled from the same data generating distribution

- Under these assumptions, the expected test error is greater than or equal to the expected training error.

# Underfitting and overfitting

- Underfitting

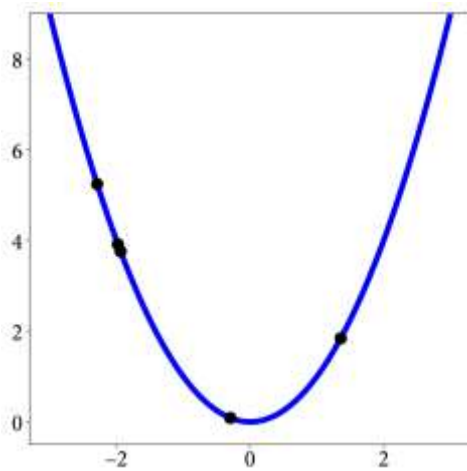
The model is unable to obtain a sufficiently low error on the training set. The model is not expressive enough.

- Overfitting

The gap between the training error and the test error is too large. The model is over-optimized for the training data.

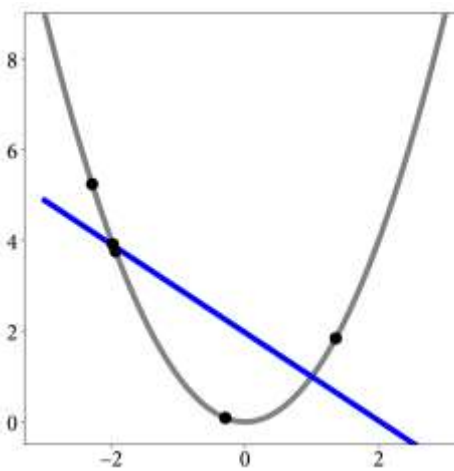
memorises noise

# Underfitting and overfitting



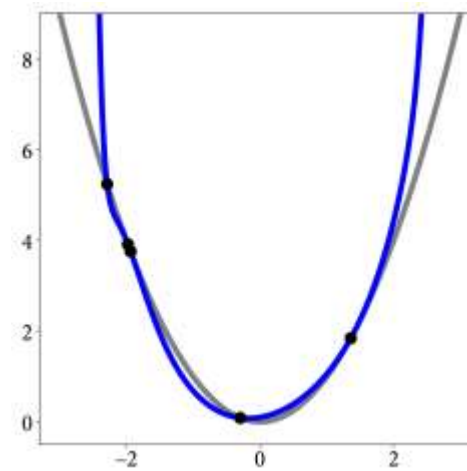
appropriate

polynomial of  
degree 2



underfitting

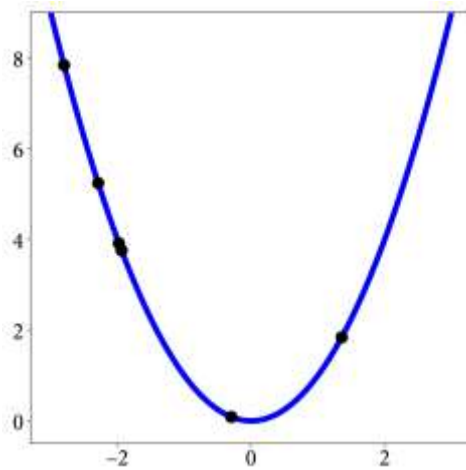
polynomial of  
degree 1



overfitting

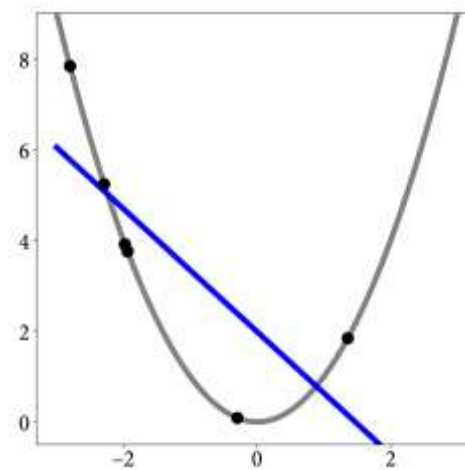
polynomial of  
degree 30

# Underfitting and overfitting



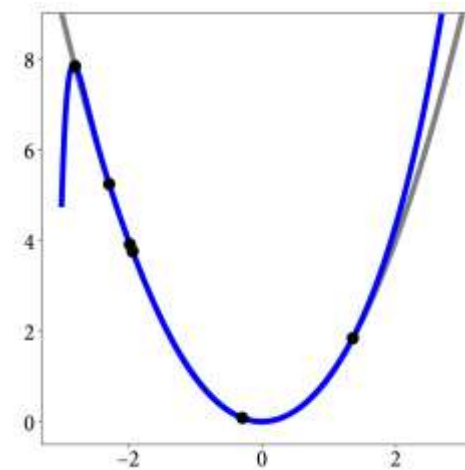
appropriate

polynomial of  
degree 2



underfitting

polynomial of  
degree 1

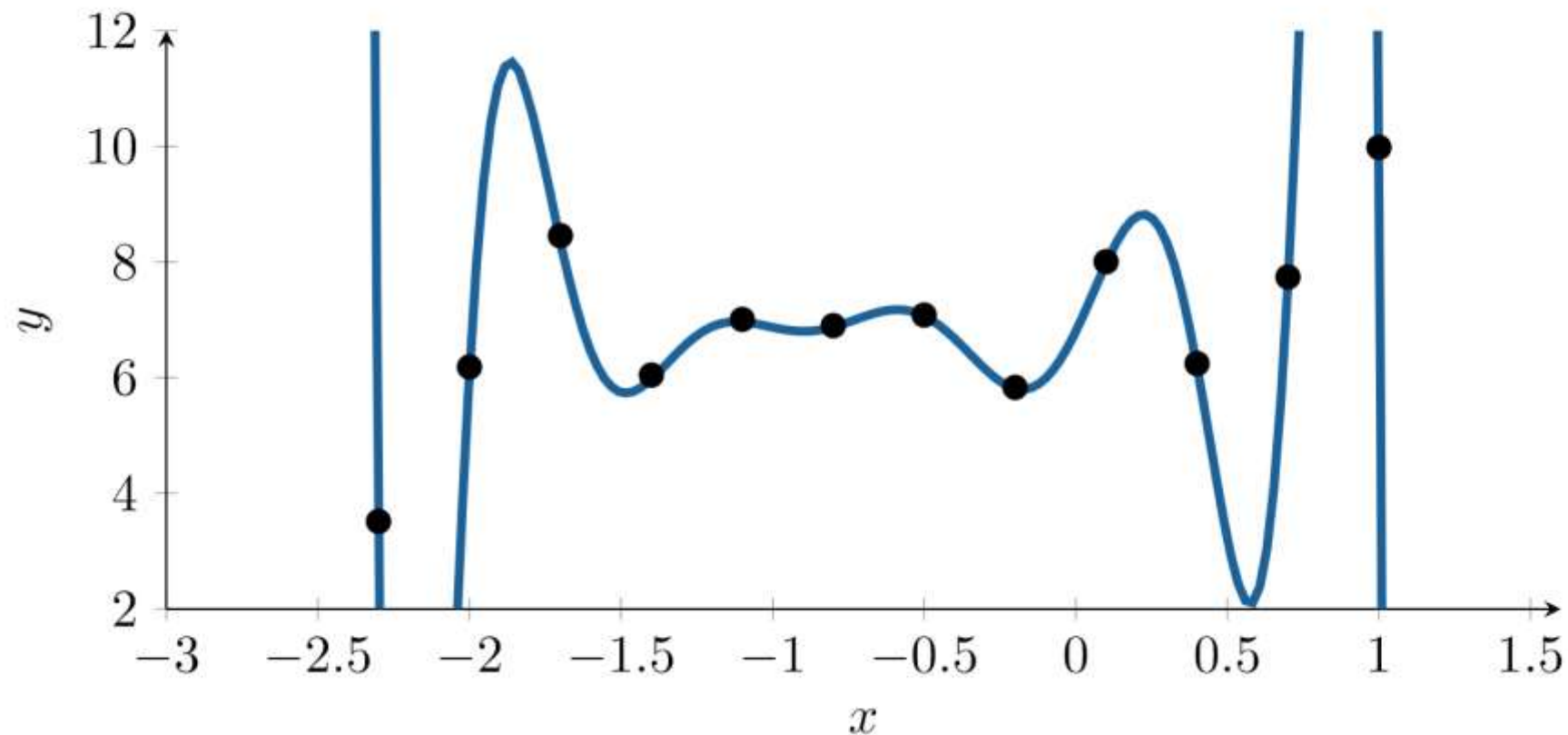


overfitting

polynomial of  
degree 30

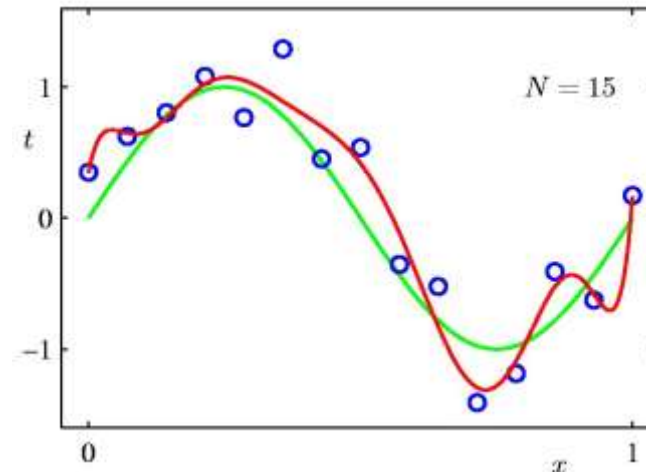
# Machine Learning Pitfall - Overfitting

With a  $p = n - 1$  degree polynomial, we can fit  $n$  data points perfectly.



# Machine Learning Pitfall - Overfitting

- Models can overfit if you have **too many parameters in relation to the training set size**.
- Example: 9th degree polynomial regression model (10 parameters) on 15 data points:



Green: True function (unknown)  
Blue: Training examples (noisy!)  
Red: Trained model

(Bishop, 2006)

- This is **not** a local minima during training, it **is** the best fit possible on the given training examples!
- The trained model captured "**noise**" in data, variations independent of  $f(\mathbf{x})$

# Overfitting – Where Does the Noise Come From?

- Noise are small variations in the data due to **ignored** or **unknown variables**, that cannot be predicted via chosen feature vector  $\mathbf{x}$ 
  - Example: Predict the temperature based on season and time-of-day. What about atmospheric changes like a cold front? As they are **not included** in the model, nor entirely captured by other input features, their variation will show up as **seemingly random** noise for the model!
- With low proportion of examples vs. model parameters, training can also mistake the variation that unmodeled variables cause in  $\mathbf{y}$  as coming from variables  $\mathbf{x}$  that **are** included. This is known as “overfitting”.
  - Since this  $\mathbf{x} \rightarrow \mathbf{y}$  relationship was merely chance, the model will **not generalize** well to future situations
  - It is usually impossible to include all variables affecting the target  $\mathbf{y}$ 's
    - Overfitting is important to guard against!

# Overfitting - Demo

- See the interactive example of ANN training again <http://playground.tensorflow.org/>
  - 2D input  $x \rightarrow$  1D  $y$  (binary classification or regression)

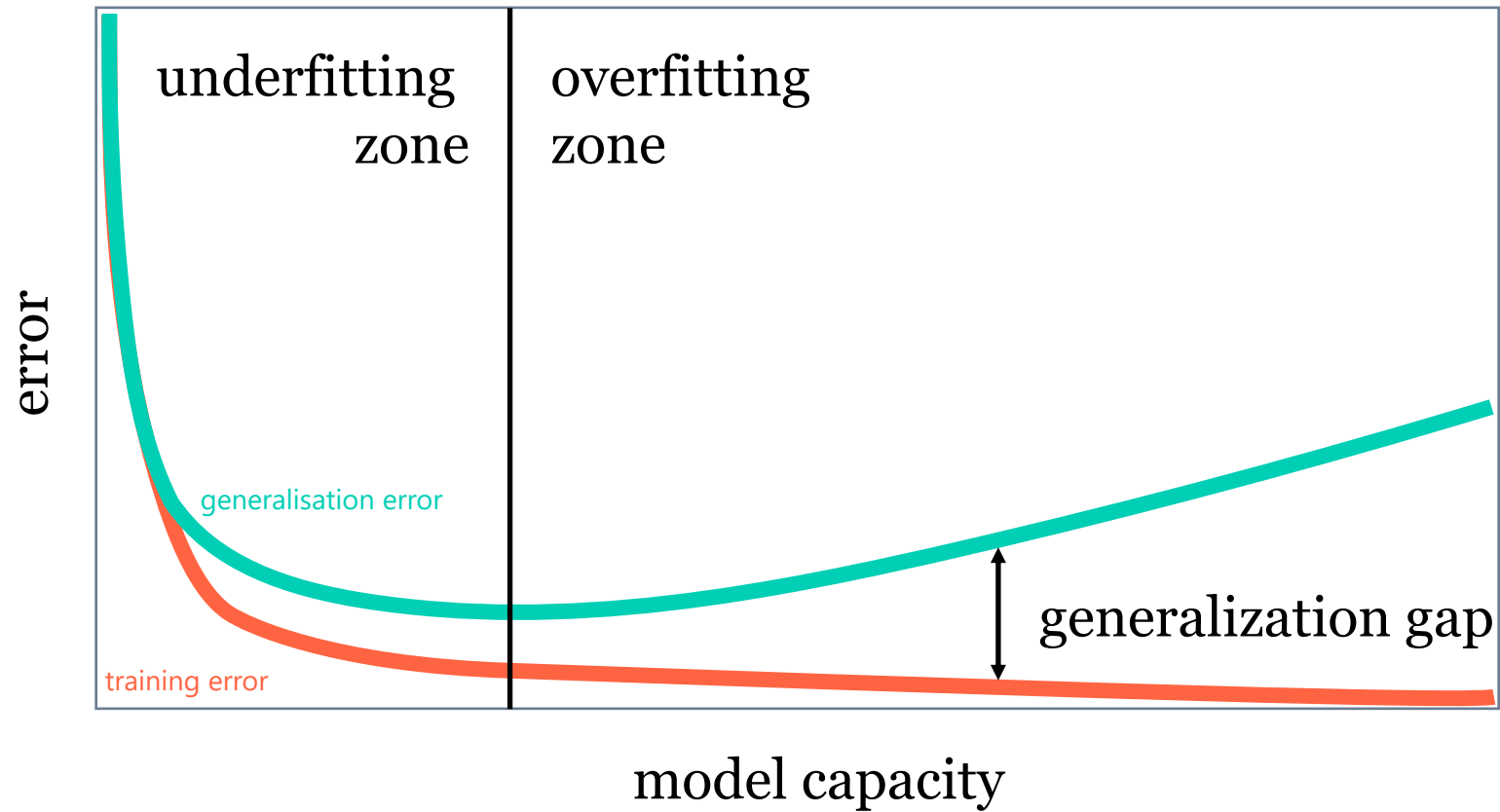
## Exercise:

- Pick the bottom-left data set, two (Gaussian) clusters
- Make a flexible network, e.g. 2 hidden layers w/ 8 neurons each
- Activation "Sigmoid"
- Set "Ratio of training to test data" to 10%
- Max out noise
- Train for a while, can adjust "learning rate" e.g. 0.3
- Compare result to "Show test data"
- How well does this model generalize? Very bad

Up next: How do we fix it?



# Relationship between model capacity and error



Source: GBC Figure 5.3

# 'No free lunch' theorems

- Averaged over all possible data-generating distributions, every learning algorithm has the same generalisation error.

Wolpert (1996)

- This means that there is no universal learning algorithm or absolute best learning algorithm.
- We need to make assumptions about the kinds of data-generating distributions we encounter in practice.

# Regularization

- One way to tailor a learning algorithm to a specific task and to prevent it from overfitting is to use regularization.
- Regularization refers to modifications intended to reduce the generalization error but not the training error of an algorithm.
- A standard example is L2-regularization, where we give preference to parameter vectors with smaller Euclidean norms.

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \rho \boldsymbol{\theta}^\top \boldsymbol{\theta}$$

# Regularization

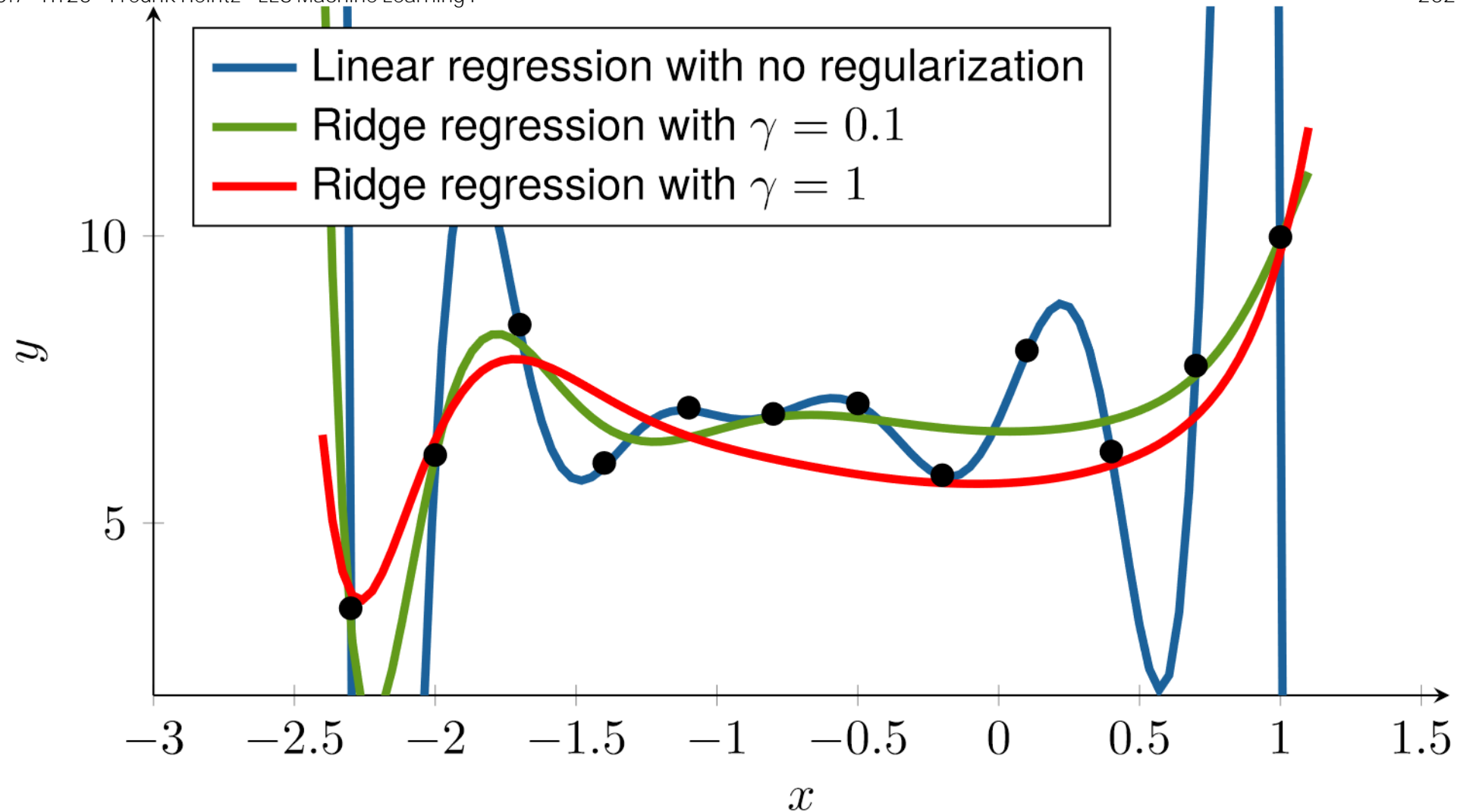
*"Keep  $\theta$  small unless the data really convinces us otherwise"*

Least squares with Ridge regression

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \|\mathbf{X}\theta - \mathbf{y}\|_2^2 + \gamma \|\theta\|_2^2$$

$$\Rightarrow (\mathbf{X}^\top \mathbf{X} + \gamma \mathbf{I}_{p+1}) \hat{\theta} = \mathbf{X}^\top \mathbf{y}$$

$\gamma$  regularization parameter



*Regularization can help us to avoid overfitting!*

# Regularization

## Ridge regression

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \gamma \|\boldsymbol{\theta}\|_2^2$$

(has a closed-form solution for  $\hat{\boldsymbol{\theta}}$ )

## LASSO

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \gamma \|\boldsymbol{\theta}\|_1$$

(lacks a closed-form solution for  $\hat{\boldsymbol{\theta}}$ )

Regularization can be used in many methods, not only linear regression!

# Model Selection – Choosing Between Models

- In conclusion, we want to avoid **unnecessarily complex** models
- This is a fairly general concept throughout science and is often referred to as **Ockham's Razor**:

*“Pluralitas non est ponenda sine necessitate”*

-Willian of Ockham

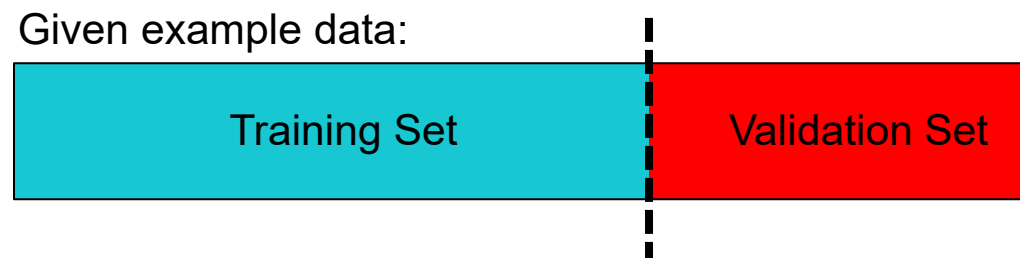
*“Everything should be kept as simple as possible, but no simpler.”*

-Albert Einstein (paraphrased)

- There are several mathematically principled ways to **penalize** model **complexity** during training, e.g. regularization, which we will not cover here.
- A simple approach is to use a separate **validation set** with examples that are **only** used for evaluating models of different complexity.

# Model Selection – Hold-out Validation

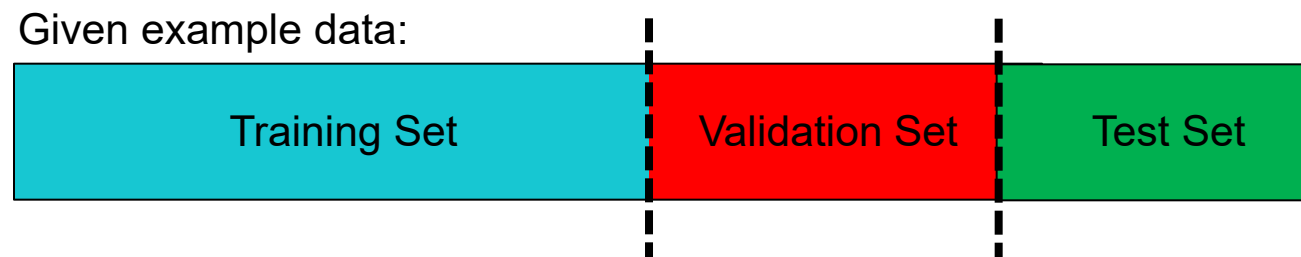
- This is called a **hold-out validation set** as we keep the data away from the training phase
- Measuring performance (loss) on such a validation set is a **better metric of actual generalization** error to unseen examples
- With the validation set we can compare models of **different complexity** to select the one which generalizes best, for model selection.
- Examples could be polynomial models of different order, the number of neurons or layers in an ANN etc.





# Measuring Final Generalization Error

- We have seen that having a validation set will lead to a more accurate estimation of generalization error to use for model selection
- However, by **extensively** using the validation set for model selection we can also contaminate it (overfitting model against the data in the validation set)
- To combat this one usually sets aside a separate **test set**
- This test set is **not** used during training or model selection
- It is basically locked away in a safe and only brought out in the end to get a fair estimate of final generalization error



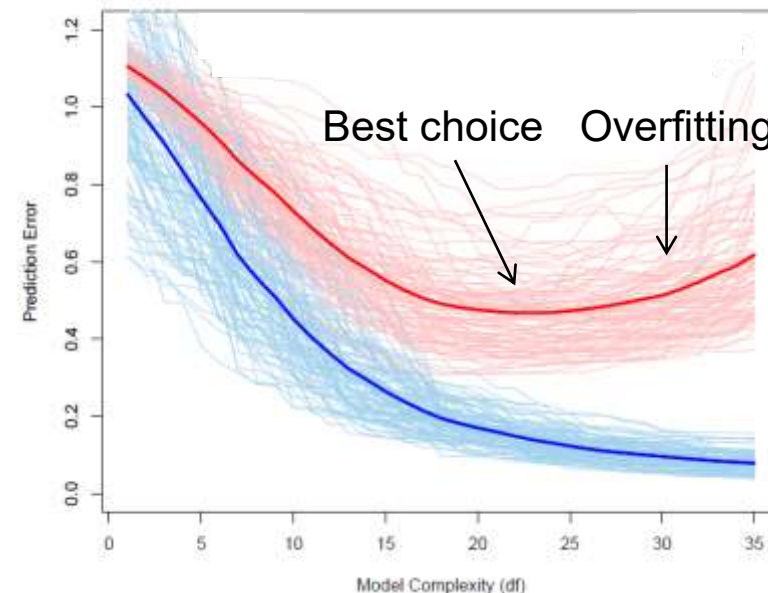
# Model Selection - Selection Strategy

- As the number of parameters increases, the size of the hypothesis space also increases, allowing a **better fit to training data**
- However, at some point it is **sufficiently flexible** to capture the underlying patterns. Any more will just capture noise, leading to **worse generalization to new examples!**

Example: Prediction error vs. model complexity over many (simulated) data sets. (Hastie et al., 2009)

Red: Validation set (generalization) error  
Blue: Training set error

- Do we need to train and test many models of different complexity?
  - Various tricks to avoid this



# Hyperparameters and validation sets

- A setting of a machine learning algorithm that is not adapted by the algorithm itself is called a hyperparameter.

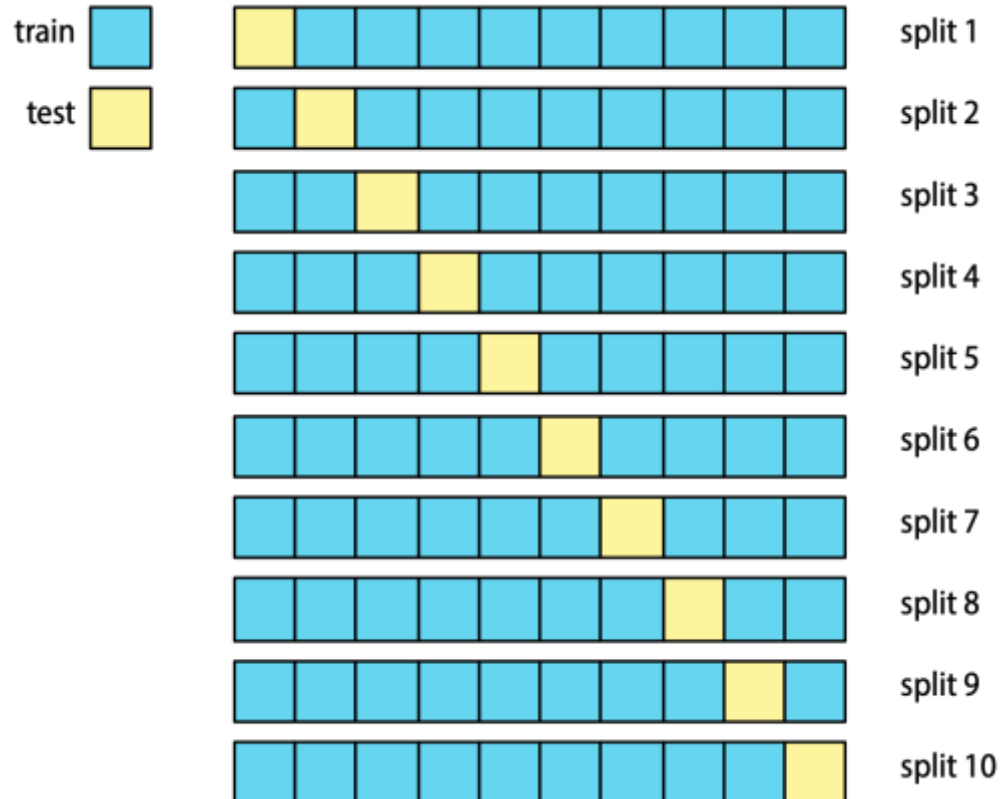
typical example: learning rate

- Some settings need to be hyperparameters because adapting them during training would lead to overfitting.

such as parameters related to the model's capacity

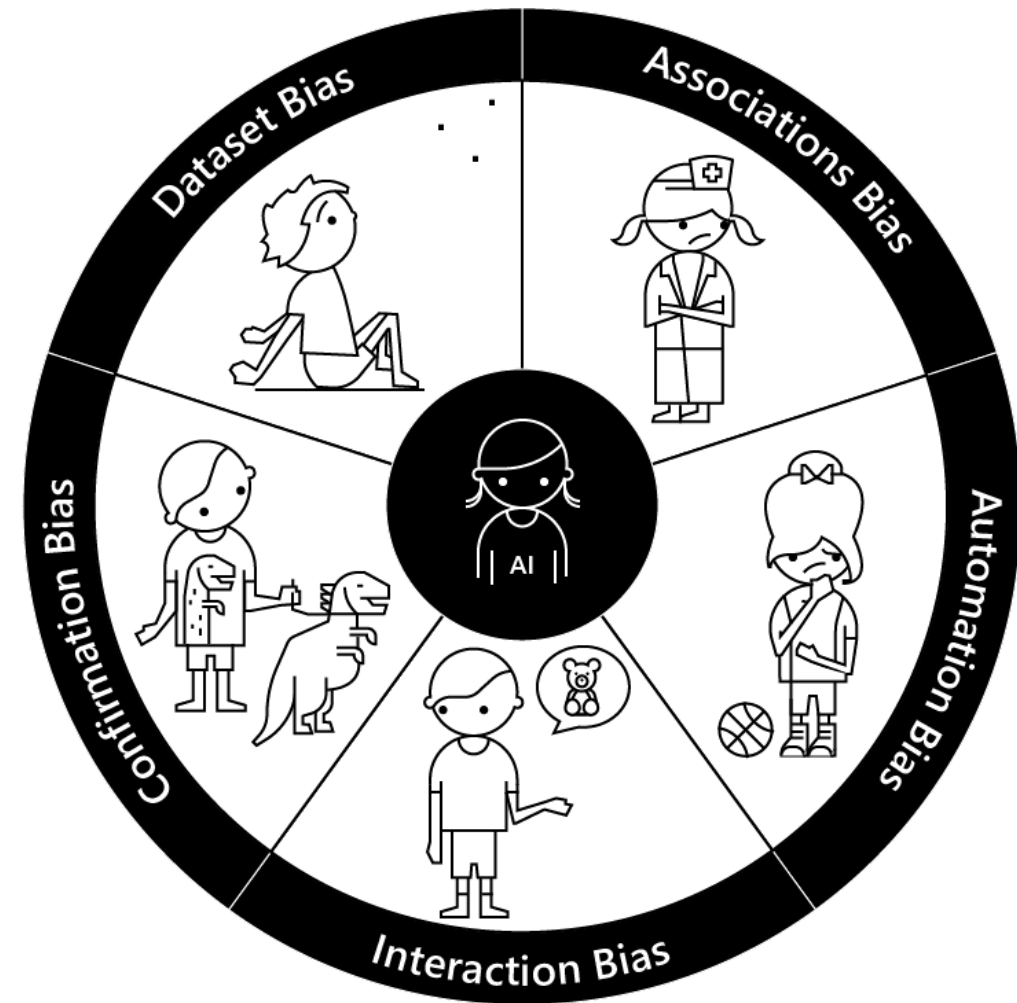
- To tune hyperparameters, we need a separate validation set, or need to use cross-validation.

# Cross-validation

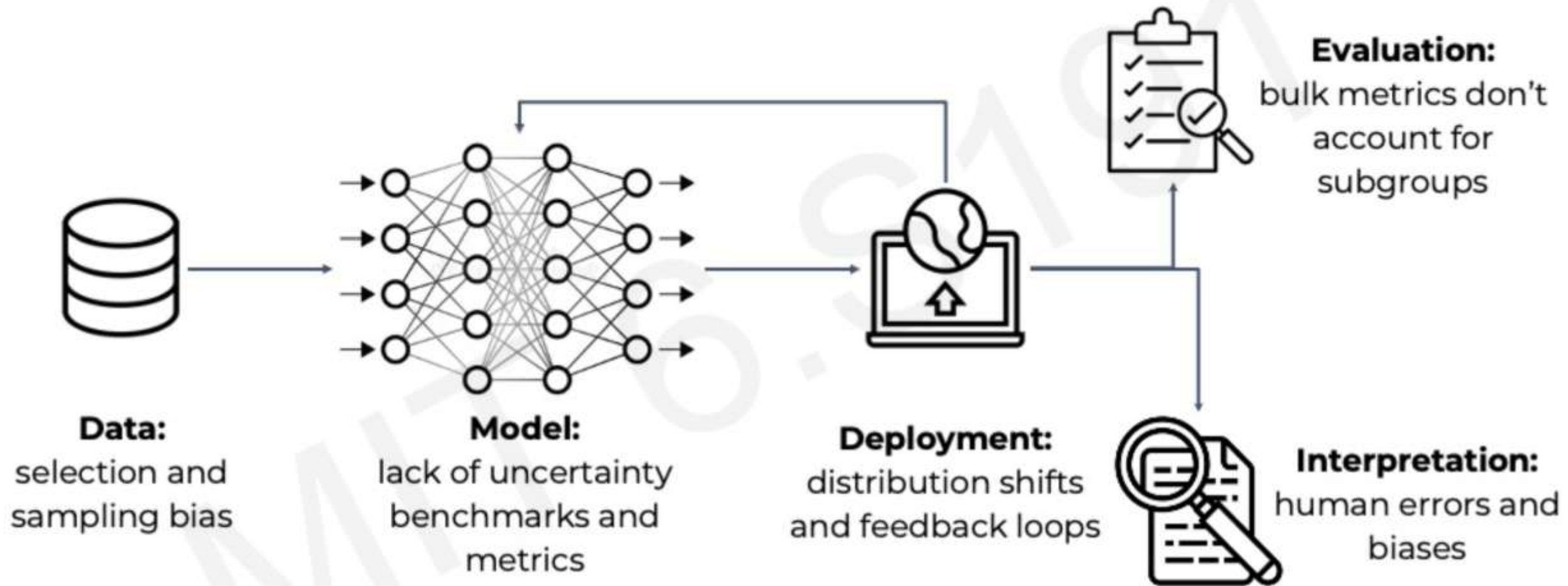


# Bias

- **Dataset bias** – When the data used to train machine learning models doesn't represent the diversity of the customer base.
- **Association bias** – When the data used to train a model reinforces and multiplies a cultural bias.
- **Automation bias** – When automated decisions override social and cultural considerations.
- **Interaction bias** – When humans tamper with AI and create biased results.
- **Confirmation bias** – When oversimplified personalization makes biased assumptions for a group or an individual.



# Bias in the AI Lifecycle



# Cited figures from...

C M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Hastie, T., Tibshirani, R. and Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second Edition, Springer, 2009.

Which are two good (but fairly advanced) books on the topic.

TDDC17 AI LE8 HT2025:  
Introduction to machine learning  
Unsupervised learning  
Supervised learning  
Evaluating machine learning methods

[www.ida.liu.se/~TDDC17](http://www.ida.liu.se/~TDDC17)