Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

CAIDAS

WüNLP

ALGORITHMS IN AI & DATA SCIENCE 1 (AKIDS 1)

Intro to Machine Learning

Prof. Dr. Goran Glavaš

29.1.2024

Content

- Intro to ML
- Supervised ML
- Linear Regression

What is Machine Learning?

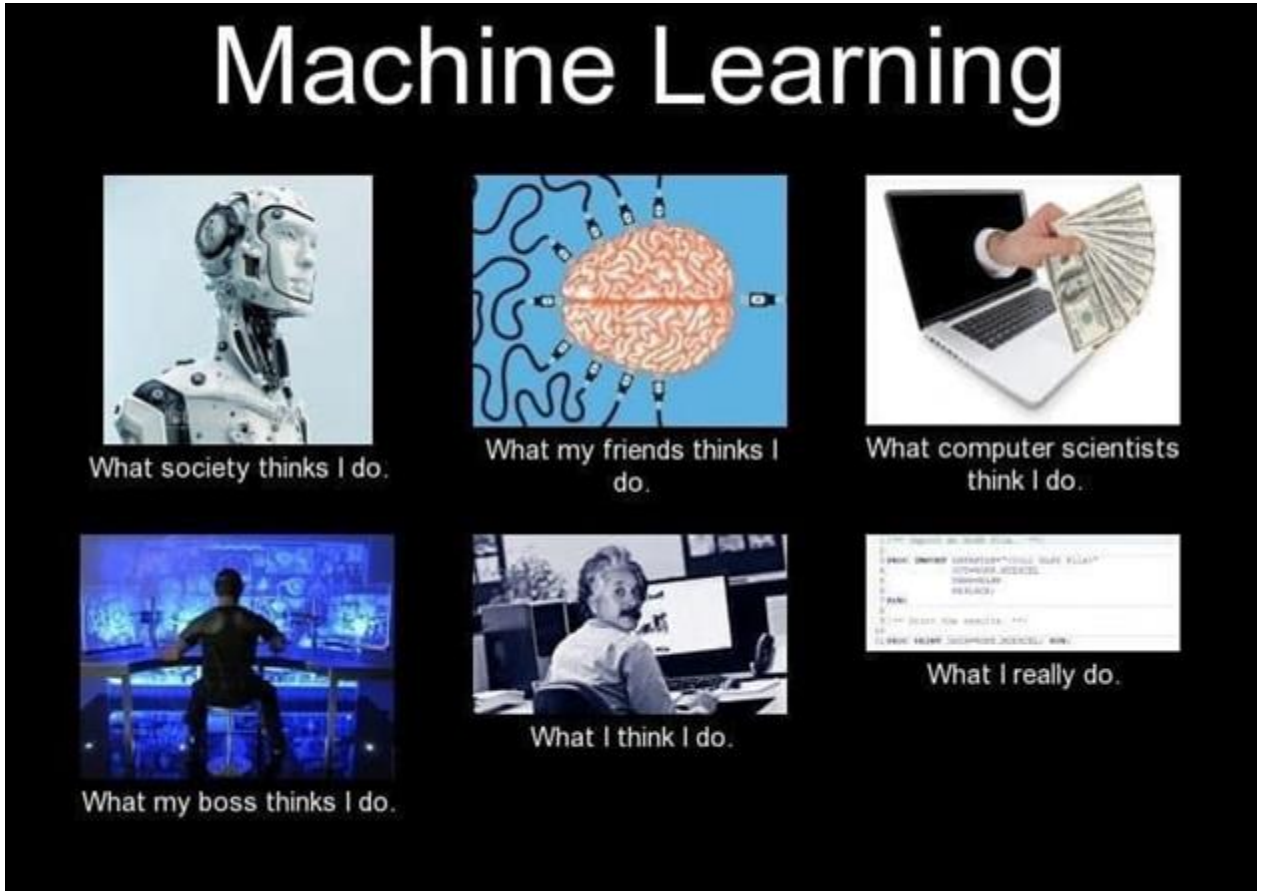
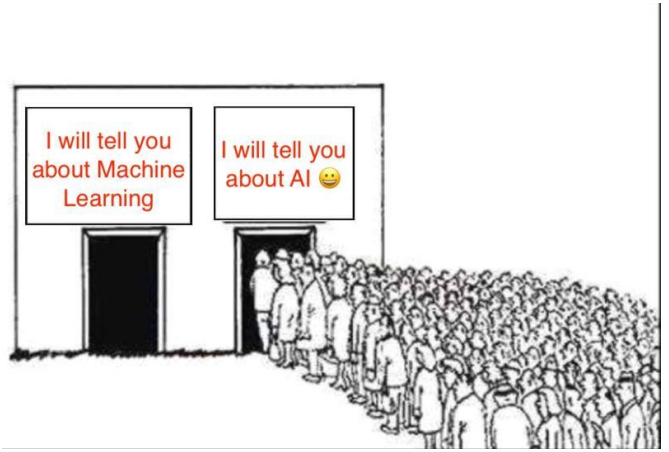
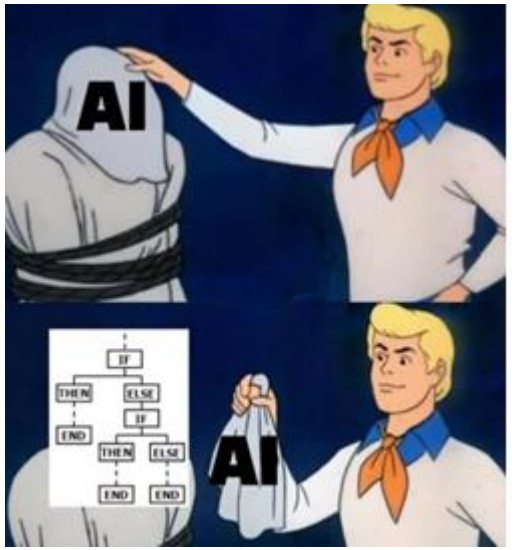


Image from: https://becominghuman.ai/how-to-make-machines-learn-f0c32cf84e28

Machine Learning

- **Machine learning** – or **learning from data** is the beating heart of modern **AI**
  - (Un)supervised learning
  - Reinforcement learning
  - Representation (self-supervised) learning
  - Deep Learning
  - Bayesian Learning
  - Transfer Learning
  - …
- Successful AI that's not ML driven is very rare, and effectively limited to **rules**
  - Not suited for tackling **complex problems** „in the wild", that is, in any domain





Images from: https://tinyurl.com/4c86ts2f

Machine learning

Machine Learning

Machine learning denotes the multitude of algorithms for (semi-)automatic extraction of new and useful knowledge from arbitrary **collections of data** (aka **datasets**). This knowledge is typically captured in the form of rules, patterns, or **models**.

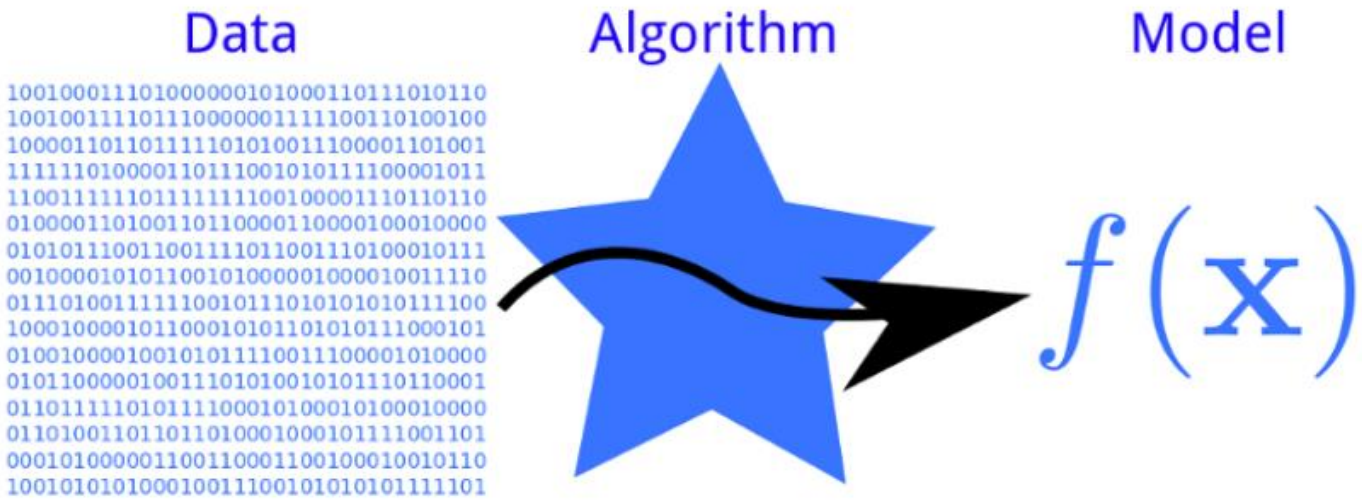


Image from: https://tinyurl.com/mpd39647

Why Machine Learning?

- We normally solve **(computational) problems** with **algorithms**

> A process or set of rules to be followed in calculations or other **problem-solving operations**, especially by a computer.
>
> **Oxford dictionary**

> A finite sequence of rigorous instructions, typically used to **solve a class of specific problems** or to perform a computation.
>
> **Wikipedia**

Why Machine Learning?

- Write an algorithm (in pseudocode) for the following problems...

Image Classification

Given an **arbitrary image**, determine which object, from a set of objects C of interest (e.g., O = {*cat*, *dog*, *chicken*}) is on the image.

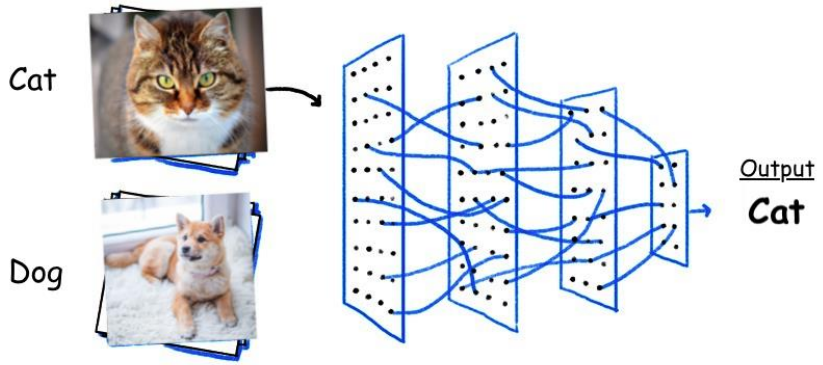


Image from: https://tinyurl.com/yhtnxm3x

Sentiment Analysis

Given an **arbitrary product review** (natural language text), determine whether it expresses positive or negative sentiment towards the product.

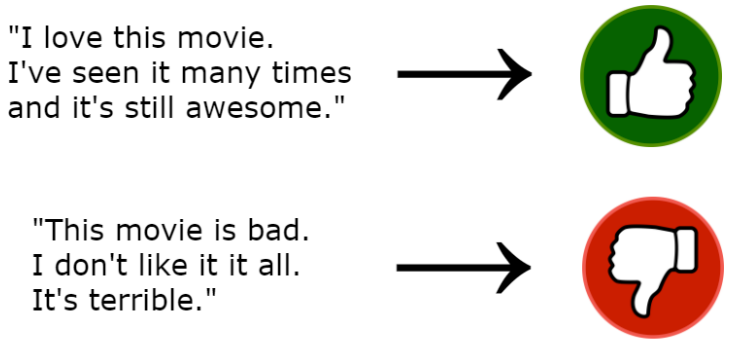


Image from: https://cfml.se/blog/sentiment_classification

AI-Complete Problems

- **AI-Complete Problems:** Problems that seem to require „human-like" intelligence, not solvable in classic „algorithmic" way

- Classic AI Approach: **Search**
  - Humans know how to define and tackle the problem
  - This knowledge is „codifiable" into a set of instructions
  - Machines solve the problems more efficiently

- Modern AI Approach: **Learning**
  - There is no human knowledge about the process / domain / solution or
  - Humans don't know how to explain the solution to the problem (e.g., speech recognition)
  - **Humans, however, typically solve these problems easily!**

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Image from https://en.wikipedia.org/wiki/Sudoku_solving_algorithms

VS.

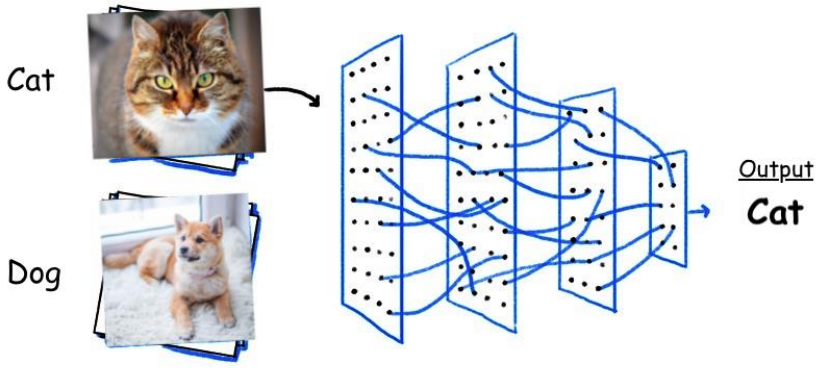


Image from: https://tinyurl.com/yhtnxm3x

ML Paradigms

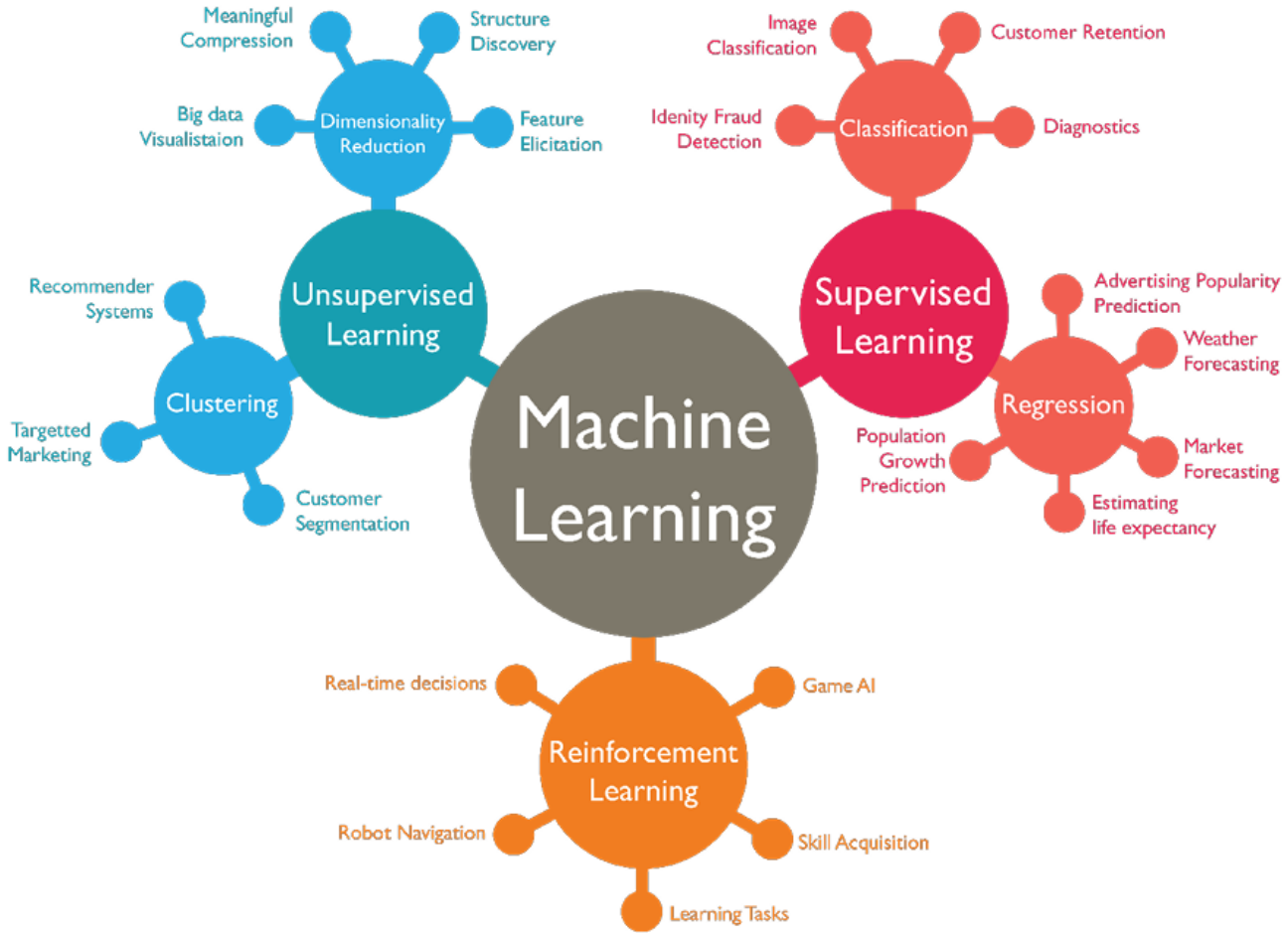


Image from: https://vitalflux.com/great-mind-maps-for-learning-machine-learning/

ML Paradigms

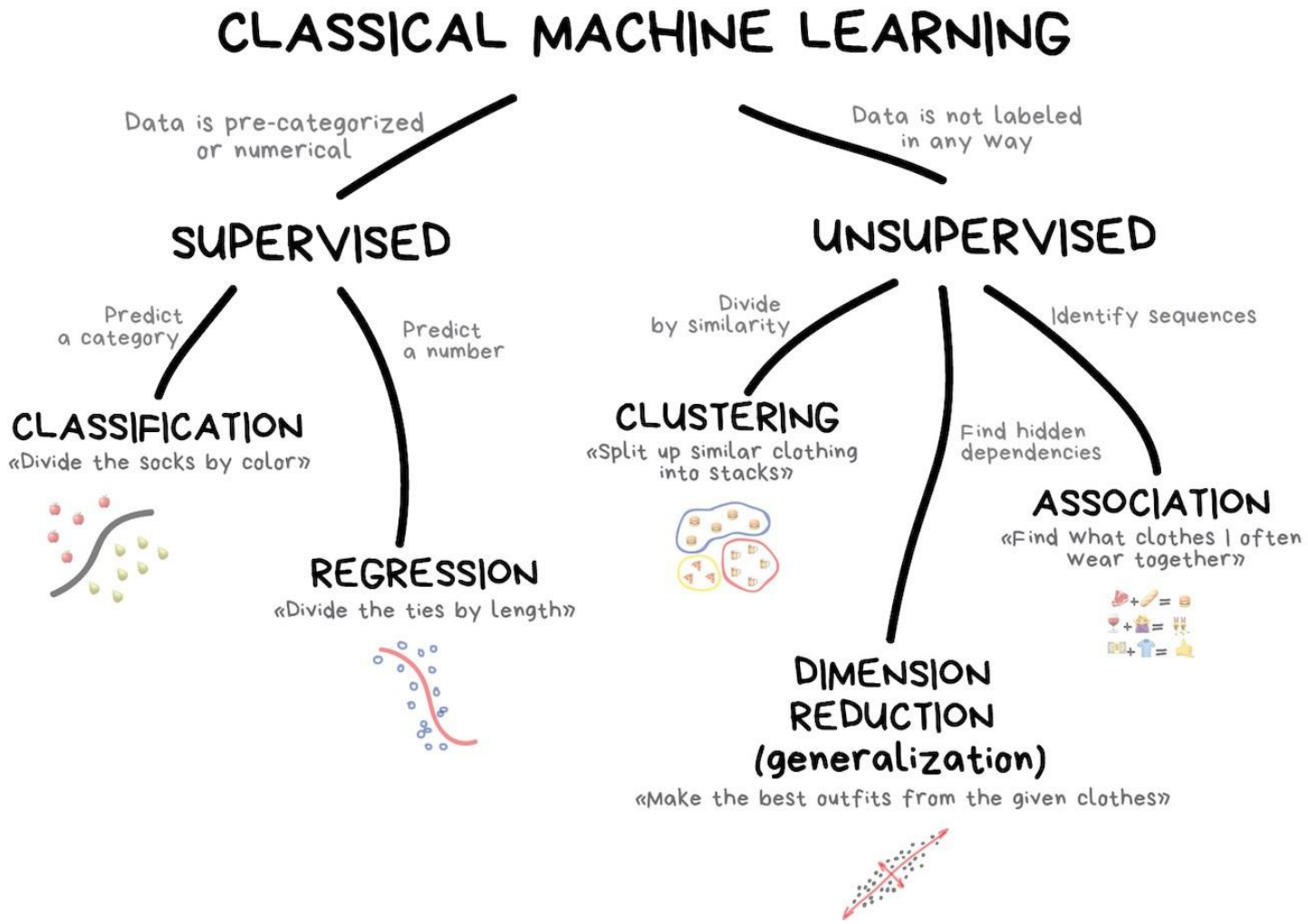


Image from: https://vitalflux.com/great-mind-maps-for-learning-machine-learning/

Machine learning paradigms

1. **Supervised ML**: we have labeled data
   - **Classification**: output is a **discrete label** (but no ordering between the labels)
   - **Regression**: output is a score on a scale (integer or real number)
2. **Unsupervised ML**: we have no labels
   - **Clustering**: grouping of data examples (aka instances)
   - **Outlier detection**: Finding data points/instances that are in some aspect significantly different from most other instances
   - **Dimensionality reduction**: compress the data in lower-dimensional representations in order to find hidden regularities
3. **Reinforcement learning**: a form of indirect supervision
   - Cumulative reward/punishment, known only after several consecutive decisions

Why Machine Learning?

- **Alternative**: **rule (knowledge)-based systems**
  - Sometimes we don't even know how to define the rules for a problem
    - E.g., how much word overlap we need to to treat texts as similar?
  - Even when we can think of some meaningful rules, we just need too many rules to cover all the cases
    - There are many exceptions that need to be handled
    - What about exceptions in exceptions?
- We need expert knowledge
  - If processing natural language, perhaps a **linguist**
- Rules are difficult to...
  - Design: rules interact in unpredictable ways
  - Maintain: adding new rules can easily break everything
  - Adopt to new domains: we need to modify/add rules
- Some tasks are inherently subjective: it is difficult to model subjectivity with rules (rules are strict)

Machine learning paradigm

- For AI-Complete problems it is often **easier** to manually label the data and show for concrete what the solution should look like
- **Learning from data**: „let the data speak”
  - **Supervised learning**: **train** a ML model to associate different inputs to different labels/classes
  - **Unsupervised learning**: find regularities between instances

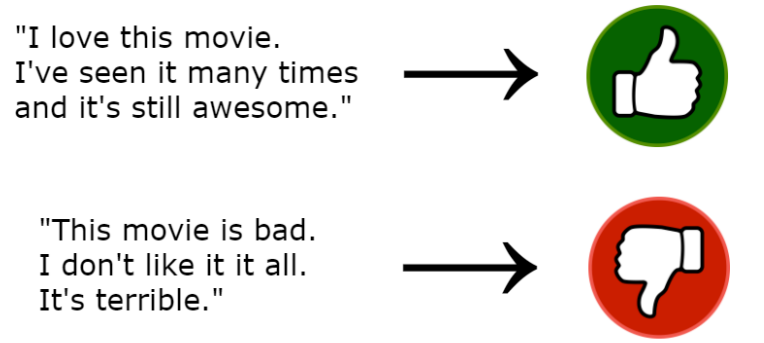


Image from: https://ctml.se/blog/sentiment_classification

ML Paradigm: Disadvantages

- Data labeling can be expensive, especially if large amounts of data are required and/or expert (linguistic) knowledge
  - E.g., Annotating syntactic of natural language sentences
- Data labeling can be tedious (= slow, error-prone)
- Sometimes, it requires quite of lot of training/coaching and many discussion rounds to settle the annotation disagreements between annotators
- Unlike the rules, **ML models** are difficult to interpret (typically it's just a bunch of parameters/numbers) – a „**black box**"
- It is difficult to perform "small tweaks" on the system: we can't add a couple of rules to fix something to make the user happy
  - Hybrid rules-ML approaches are, of course, possible and sometimes make sense

Content

- Intro to ML
- Supervised ML
- Linear Regression

ML Cookbook

- Steps in **creating an ML solution** to a problem

1. **Data preprocessing** and analysis (e.g., cleaning)
2. **Data annotation** (typically manual)
3. **Feature extraction** (aka „how to represent instances”)
4. **Model selection** (choose the right **model** and **learning algorithm)**
5. **Training/learning** (estimate **model’s parameters** from the data)
6. **Evaluation** (estimate how good/accurate your model is)
7. **Error analysis** (when does the model make wrong predictions and why?)
8. **Deployment/usage** (move the model „to production” to make predictions)

Space of Examples

- We typically operate in (vector) **spaces** of examples in which **individual examples** (aka **instances**) are concrete points
  - We have kind of already seen this in numerical optimization

Space of Examples in ML

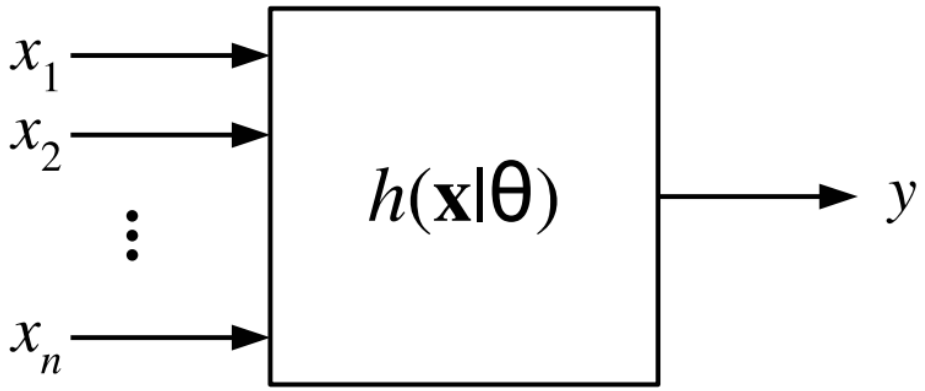
In **machine learning**, individual examples (or instances) $\mathbf{x} = [x_1, x_2, ..., x_n]$ are **points** in a space **X**, consisting of values for **features** $x_1, x_2, ..., x_n$. The space **X** is the determined (i.e., spanned) by the domains of the features: $\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_n$. The domains of different features can be **discrete** (the so-called categorical or multinomial features) or **continuous.**

- $[x_1, x_2, ..., x_n]$ is a **feature vector** of the example/instance

Supervised ML

- **Input**: example represented by the **feature vector**: **x** = **[**x_1, x_2, ..., x_n**]**
- **Output**: the **label** y assigned to the example
  - y is a **discrete class** (in classification problems) or a **score** (in regression problems)

- A machine learning **model** *h* maps an input **[**x_1, x_2, ..., x_n**]** to a label y:
- The model has a set of *k* **parameters θ = [**θ_1, θ_2, ..., θ_k**]**: $y = h(\mathbf{x}|\boldsymbol{\theta})$

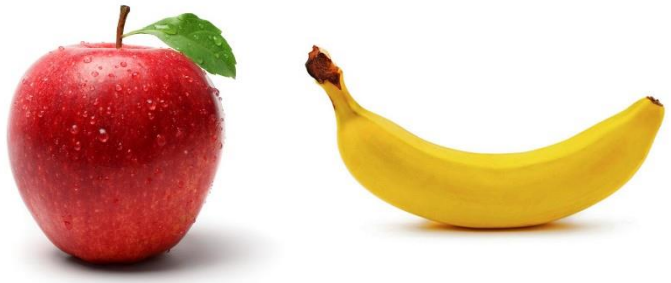


Supervised ML: Toy Example

- You want to **learn** a **classifier** that can differentiate between an apple and a banana

- **Instance/example**: some *concrete* apple or some *concrete* banana.
  - Feature vector **x** = [x_1, x_2, x_3, x_4, ...]

    x_1: length of the fruit
    x_2: circumference
    x_3: weight
    x_4: color
    …



- **Label**: y ∈ { c_1 = apple, c_2 = banana}

Classification tasks

- **Binary classification**: just **two** classes (yes/no, 0/1)

$$h : \mathcal{X} \to \{0, 1\}$$

- **Multi-class classification**: an instance belongs to strictly **one** of *K* classes

$$h : \mathcal{X} \to \mathcal{Y}, \quad \mathcal{Y} = \{1, \ldots, K\}, \quad K > 2$$

- **Multi-label classification**: an instance can belong to **two or more classes**

$$h : \mathcal{X} \to \wp(\mathcal{Y})$$

Classification tasks

- Many **good** classification models are **binary classifiers**
  - Can „out of the box” be applied only to binary classification problems
  - E.g., logistic regression, support vector machines (SVM)

- How to use those models in **multi-class classification** too?
  - **Change the model**: invent a multi-class version of it
    - E.g., softmax regression is a **multi-class version** of logistic regression
  - Train and combine **multiple binary classifiers**: two main schemes
    - **One-vs-rest** (or **one-vs-all**): one binary classifier per class
    - **One-vs-one**: one binary classifier for each pair of classes

Classification tasks

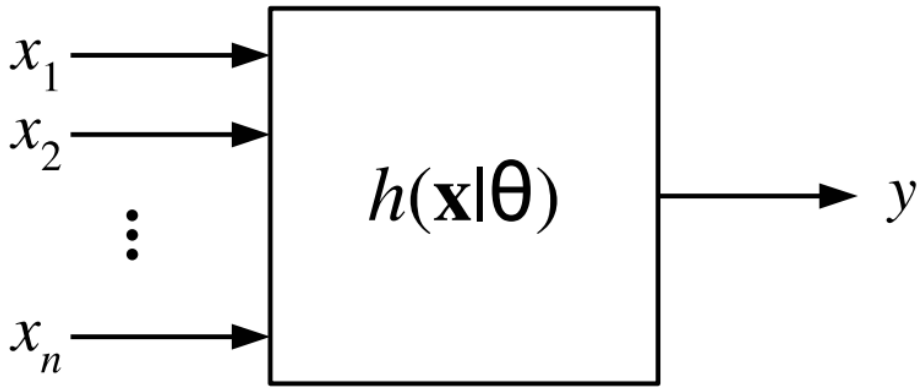
- **Binary classifiers** typically produce some kind of score indicating probability or **confidence** that instance **x** belongs to class y, `conf(x, y)`
- **Multi-class classification schemes** for binary classifiers leverage these confidence scores
  - Multi-class problem: Y – set of N classes, y_1, y_2, …, y_N
- **One-vs-rest:** one **binary classifier** for each class y_i
  - **Training**: instances of y_i assigned class y = 1, instances of all other classes y = 0
  - **Prediction** (aka **inference**):
    - After training we have N binary classifiers C = {c_1, c_2, …, c_n}, one for each class y_1, y_2, …, y_N
    - Make prediction with each classifier c_i and get its **confidence score** for the new instance **x** (i.e., confidence that **x** belongs to y_i)
    - Assign the class y_i the classifier c_i of which was the most confident
- **Q**: how would you do it (implement prediction) with **one-vs-one**?

```
one_vs_all_rest(x, C, Y)
  best = -inf
  cls = null
  for i in 1 to len(C):
    cf = C[i].conf(x, Y[i])
    if cf > best
      best = cf
      cls = Y[i]
  return cls
```

```
one_vs_one_pred(x, C, Y):
  ...
  Q: how many classifiers in C?
  Q: how to get one score for
      each class?
```

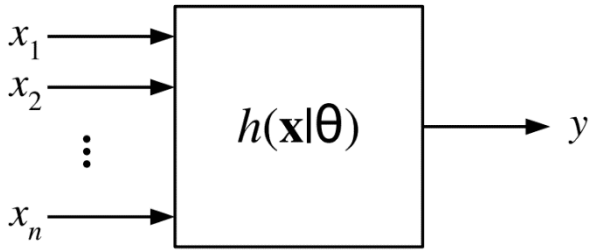
Supervised ML

- How do we choose the function *h*?
  - How do we find good values for its parameters **θ**?
- We have the **annotated dataset** D = {(**x**, $y)_i$}, i in {1, 2, …, N}
  - N examples with correct labels (classes or scores)
  - N inputs for *h* for which we know the output we would like *h* to give



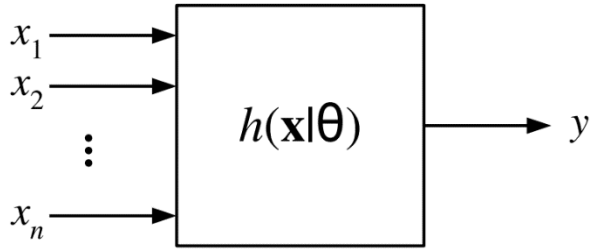
Supervised ML

- We need to „find” h using D = $\{(\mathbf{x}, y)_i\}$, i in {1, 2, ..., N}
- We need to start from **some set of candidate functions** for h
  - Simplification: let’s say we only have one feature – $\mathbf{x} = [x]$
  - E.g., a 3rd degree polynomial function: $h(\mathbf{x}|\boldsymbol{\theta}) = x^3\theta_3 + x^2\theta_2 + x\theta_1 + \theta_0$
- **Model** = set of candidate functions h (up to the parameter values)
- **Model selection** = choosing the model
- **Training/learning:** finding the optimal values of the parameters $\boldsymbol{\theta}$ using the annotated data D = $\{(\mathbf{x}, y)_i\}$
  - Optimization problem!
  - **How to optimize $\boldsymbol{\theta}$ using D?**
  - What is the **function** that we should optimize?



Supervised ML

- **Training/learning:** finding the optimal values of the parameters **θ** using the annotated data D = $\{(\mathbf{x}, y)_i\}$
  - **How to optimize θ** using **D**?
  - What is the **function** that we should optimize?



- We want *h* to make correct predictions
  - We know correct labels y for inputs **x** in the training set D
  - For some concrete values of parameters **θ**, we quantify how much $h(\mathbf{x}|\boldsymbol{\theta})$ differs from y
- **Empirical error** quantifies how much predictions of *h* (for some concrete parameter values **θ**) deviate from true labels for annotated examples from D

$$E(h|D) = \frac{1}{N}\sum_{i=1}^{N} L(h(\boldsymbol{x}_i|\boldsymbol{\theta}), y_i)$$

- L is a **loss function** – a concrete function that quantifies the difference between the prediction and true label of an individual example **x**

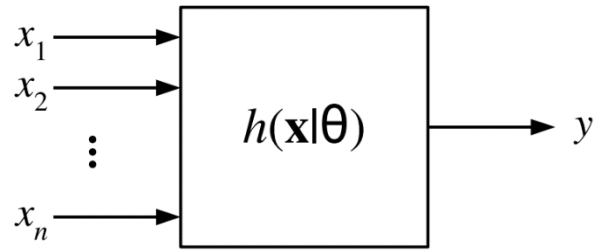
Supervised ML

$E(h|D) = \frac{1}{N}\sum_{i=1}^{N} L(h(\boldsymbol{x}_i|\boldsymbol{\theta}), y_i)$

- L is a **loss function** – a concrete function that quantifies the difference between the prediction and true label of an individual example **x**

- **Example loss functions**
  - **0-1 loss** (classification): L = 1 **if** $h(\mathbf{x}|\boldsymbol{\theta}) \neq y$ **else** 0 (not differentiable)
  - **Absolute loss** (classification and regression): $|h(\mathbf{x}|\boldsymbol{\theta}) - y|$ (not differentiable)
  - **Quadratic loss** (regression): $(h(\mathbf{x}|\boldsymbol{\theta}) - y)^2$
  - **Cross-entropy loss** (classification): $-y \ln h(\mathbf{x}|\boldsymbol{\theta}) - (1-y) \ln(1-h(\mathbf{x}|\boldsymbol{\theta}))$

- **Some loss functions are** not differentiable
  - Thus cannot be used in gradient-based numerical optimization



Supervised ML

Three components of a supervised **machine learning algorithm**

1. Model: a set of functions among which we're looking for the best

$$H = \{ h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

- **hypothesis** = a concrete function obtained for some values $\boldsymbol{\theta}$
- Model is a set of hypothesis

2. Loss function L: used to compute the **empirical error** E on a dataset $D = \{(\mathbf{x}, y)_i\}$

$$E(h|D) = \frac{1}{N}\sum_{i=1}^{N} L(\, h(\boldsymbol{x}_i|\boldsymbol{\theta}), y_i)$$

3. Optimization procedure: procedure or algorithm with which we find the hypothesis h^* from the model H that **minimizes** the empirical error

- Equivalent to finding parameters $\boldsymbol{\theta}^*$ that minimize E

$$h^* = \text{argmin}_{h \in H}\, E(h|D)$$

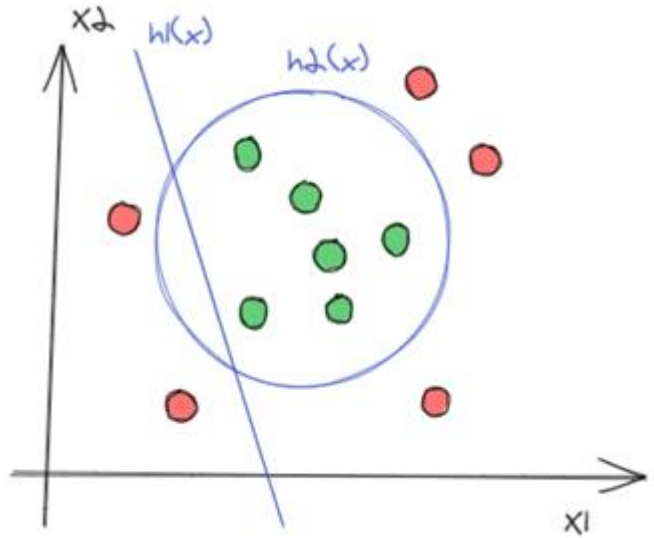
$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\, E(h|D)$$

Model Complexity

- Ideally, in the set of hypotheses (i.e., model) $H = \{ h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$ we can find at least one h for which the empirical error $E(h|D)$ is **0**

- If there is no such hypothesis, then the model is of **insufficient complexity** for the data

- **Example**: Binary classification dataset, instances $\mathbf{x} = [x_1, x_2]$
  - **Model 1** (lines): $h_1(\mathbf{x}|\boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 x_2$
  - **Model 2** (circles): $h_2(\mathbf{x}|\boldsymbol{\theta}) = (x_1 - \theta_1)^2 + (x_2 - \theta_2)^2 - \theta_3^2$
    - **Note**: Model 2 has more parameters than Model 1
  - No hypothesis from Model 1 can successfully divide the examples of the two classes (green from red)



Model Selection

- In a supervised ML problem, we only have the data $D = \{(\mathbf{x}, y)_i\}$
  - In principle, we don't know the **complexity** of the **actual function that** generated the data (i.e., maps **x** into y)
  - Data is **always noisy**

- **Model selection** = find the model of an appropriate complexity
  - Most **model families** have some values that determine the model **complexity**
  - These values are called **hyperparameters**
  - Setting values for (all) hyperparameters gives a concrete model
  - Once we have a concrete model, we find the optimal hypothesis (optimal values of actual parameters **θ**) via **training**

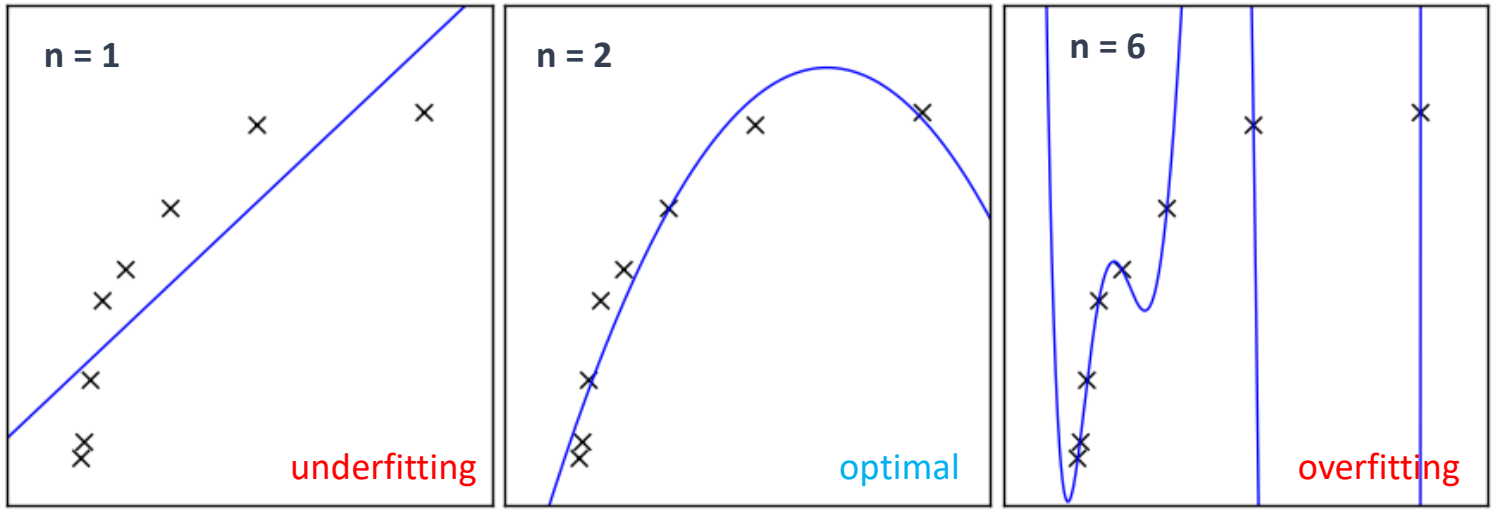
Model Selection: Example

- **Model family**: polynomial functions (of a single variable)
  - $H = \{h(\mathbf{x}|\boldsymbol{\theta})\} = \theta_n x^n + \theta_{n-1} x^{n-1} + \ldots + \theta_1 x + \theta_0$
  - **Hyperparameter**: the degree of the polynomial n
  - Different choice of n, different number of actual parameters: $\boldsymbol{\theta} = \{\theta_0, \theta_1, \ldots, \theta_n\}$

  - Each concrete value of n instantiates one model
    - For $n = 1$, the model is $h(x|\boldsymbol{\theta})\} = \theta_1 x + \theta_0$
    - For $n = 2$, the model $h(x|\boldsymbol{\theta})\} = \theta_2 x^2 + \theta_1 x + \theta_0$
    - ...
- Hyperparameter values need to be **fixed** before training
  - Training then finds the optimal values of model parameters
  - Bigger n (hyperparam. value), **more parameters**, the **more complex** the model

Model Selection

- Toy examples: single feature, **x** = x
- Let's assume our data was generated with a 2nd degree polynom (and then some noise was added to the labels)



- Optimal hyperparameter value would be n = 2
  - Model with three parameters: $h(x|\boldsymbol{\theta})\} = \theta_2 x^2 + \theta_1 x + \theta_0$

Model Selection

- In a supervised ML problem, we only have the data $D = \{(\mathbf{x}, y)_i\}$
  - In principle, we don't know the **complexity** of the **actual function that** generated the data (i.e., maps **x** into y)
  - Data is **always noisy**
- **Q:** How do we find the optimal values for hyperparameters then?
  - Answer: **search**, we try different values and see which „works best"
  - For that we need another **annotated dataset**, **different from training set**
- **Validation (or development) data**
  - An annotated dataset we use for model selection

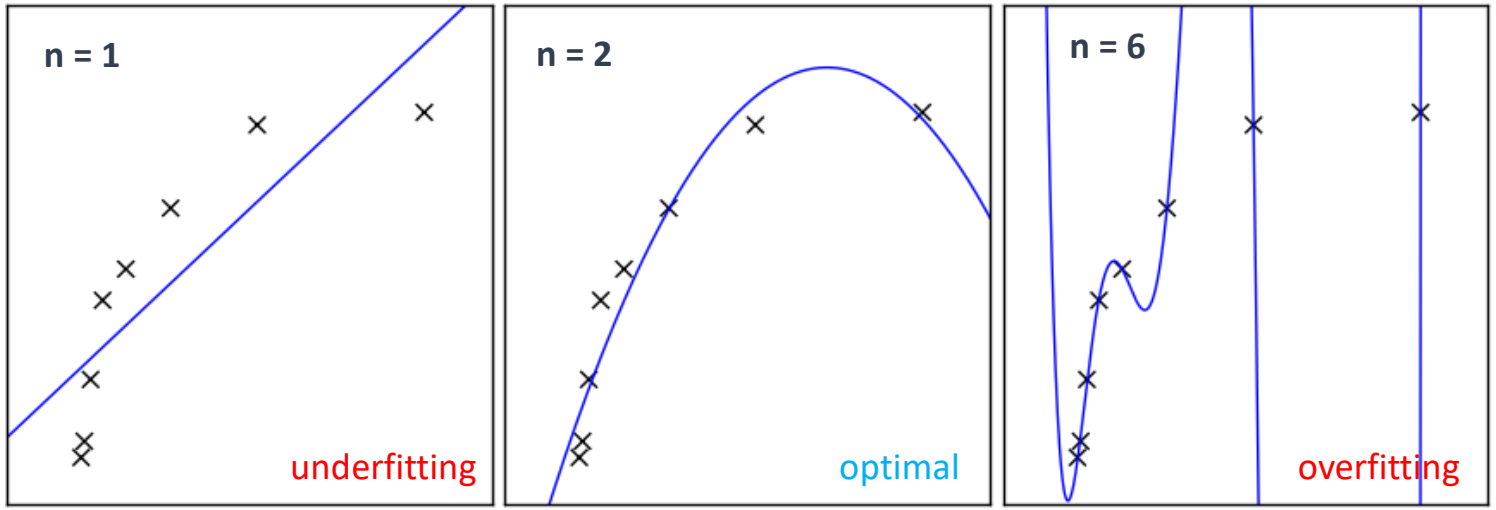
Model Selection: Cross-Validation

- Two non-overlapping annotated datasets
  - Training dataset: $D_{tr} = \{(\mathbf{x}, y)_i\}$
  - Validation dataset: $D_{val} = \{(\mathbf{x}, y)_j\}$

- **Model selection algorithm**:
  1. Define the set of hyperparameter values (models of different complexity) to examine
  2. For each hyperparameter value:
     - **Train the model** on the training set D_{tr}: $h^* = \text{argmin}_{h \in H} E(h|D)$ (or $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} E(h|D)$)
     - Measure the **empirical error** of the trained model on the validation set D_{val}: $E(h^*|D_{val})$
  3. Select the model that corresponds to the hyperparameter value for which the smallest empirical error $E(h^*|D_{val})$ on the validation data was observed

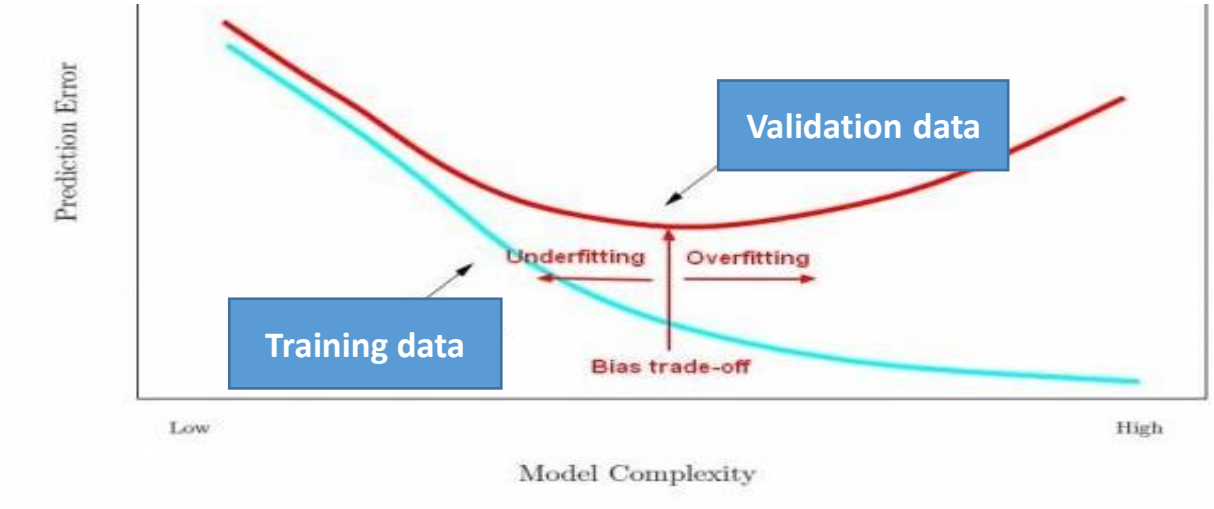
Overfitting and Underfitting

- **Overfitting**: model too complex for the data (too many parameters)
  - Too expressive model will learn the noise rather than the underlying function
- **Underfitting**: model complexity too low, insufficient to model the data
  - No matter which parameters we find for the model, it won't work well
  - „No line can match a parabole"



Overfitting and Underfitting

- **Q**: How to recognize underfitting and overfitting?
  - **Underfitting → large** empirical error on both D_{tr} and D_{val}
  - **Overfitting → small** empirical error on D_{tr} but **large** on D_{val}



- **Optimal model:** the one with smallest empirical error on D_{val}
  - Model that **generalizes** best (to „unseen examples")

Supervised ML: **Evaluation**

- **Q: What do we report as model performance**
  - Different tasks have different evaluation metrics
  - But on which data should we measure model's performance?

- An annotated dataset must not inform anything about the model
  - **Not** D_{val} because we determined the optimal hyperparameter values using it (i.e., did the model selection on it)
  - **Not** D_{tr} because we trained the model on it (determined the optimal parameter values using it)

- **Test set**: third annotated dataset D_{test}, not overlapping with D_{tr} and D_{val}
  - We measure the **empirical error** (or some highly correlated performance measure) on the D_{test} and report that as a **fair estimate of model's performance**

Content

- Intro to ML
- Supervised ML
- Linear Regression

Linear Regression

- **Linear Regression** is arguably the simplest supervised ML models
  - In statistics called „ordinary least squares", or just „regression"
  - Model output is a linear combination of input features

$$h(\mathbf{x} = [x_1, x_2, \ldots, x_n] \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

- Linear regression **assumes** that
  - y is **linearly** dependent on each feature x_i
  - Features mutually **independent**
    - Value of x_i has no effect on value of x_j and vice versa
  - y values represent a **normal (Gaussian) distribution** over **x**
- Linear regression is a poor model choice if data D does not satisfy these assumptions

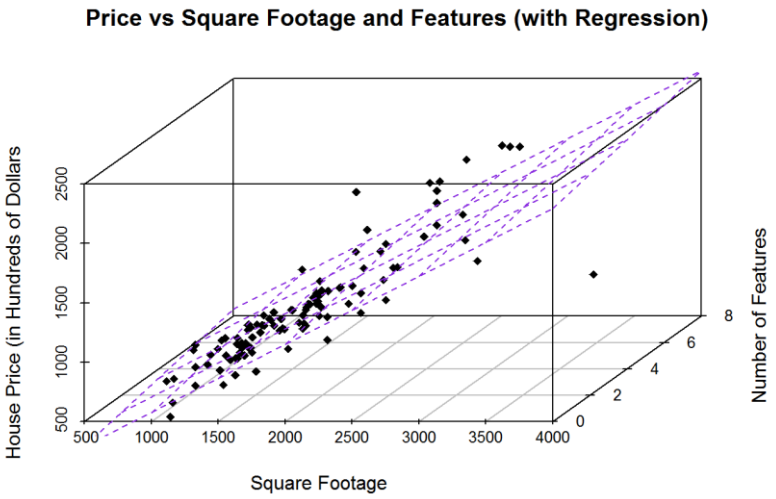


Image from: https://rpubs.com/svoboa/64900

Linear Regression

- Every machine learning algorithm has three components:
  - **Model**, **loss function**, and **optimization procedure**

- **Loss function** for linear regression: **squared error**

$$L(y, h(\mathbf{x})) = (y - h(\mathbf{x}|\boldsymbol{\theta}))^2$$

- **Empirical error** (average L on a dataset D)**: mean square error (MSE)**

$$E(h|D) = \frac{1}{2} \sum_{i=1}^{N} (y_i - h(\mathbf{x_i}|\boldsymbol{\theta}))^2$$

- **Optimization procedure**: minimize MSE on training data, $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} E(h|D_{tr})$
  - **Q**: How do we find a minimum of a function?
  - **Solve the equation**: $\nabla_{\boldsymbol{\theta}} \left[\frac{1}{2} \sum_{i=1}^{n} (y_i - h(\mathbf{x_i}|\boldsymbol{\theta}))^2 \right] = 0$

Linear Regression

- **Optimization procedure**: minimize MSE on training data, $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} E(h \mid D_{tr})$
  - **Solve the equation**: $\nabla_{\boldsymbol{\theta}} \left[\frac{1}{2} \sum_{i=1}^{n} (y_i - h(\mathbf{x}_i \mid \boldsymbol{\theta}))^2 \right] = 0$

- Luckily, this equation actually has a **closed form solution**
  - It means there is a **formula** by which we can **directly compute** $\boldsymbol{\theta}^*$
  - For many other models and their loss functions, this is not the case, and we have to resort to numerical optimization (typically gradient-based)

- Let us stack all the examples of the training set D_{tr} in a **matrix** $\mathbf{X}$ and corresponding labels in the same order in a **vector** $\mathbf{y}$

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \cdots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,n} \\ & \vdots & \ddots & \vdots \\ 1 & xN,_1 & \cdots & x_{N,n} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix}$$

Solution is then computed as:

$$\boldsymbol{\theta}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Questions?


Awọn ibeere?
ਸਵਾਲ?
Küsimusi?
Turite klausimų?
Pitanja?
Sorusu olan?
Dúvidas?
質問は？
有问题吗？
Fragen?
¿Preguntas?
Questions?
Domande?
Frågor?
Pytania?
Ερωτήσεις;
Vragen?
Πιτάπαρ?
Porandukuéra?
أسئلة؟