

TDDC17 LE5 HT2023

Machine Learning I

Fredrik Heintz

Dept. of Computer Science

Linköping University

fredrik.heintz@liu.se

@FredrikHeintz

Outline:

- **Introduction to machine learning**
- **Unsupervised learning**
- **Supervised learning**

Outline of Machine Learning Lectures

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

What is Machine Learning about?

- To enable machines to *learn and adapt* without explicitly programming them
- Our only frame of reference for learning is from biology
 - ...but brains are hideously complex, the result of ages of evolution
- Like much of AI, Machine Learning mainly takes an **engineering approach**¹
 - Remember, humanity didn't master flight by just imitating birds! ☺



¹. Although there is occasional biological inspiration

Theoretical Foundations

Mathematical foundations borrowing from several areas

- Statistics (theories of how to **learn from data**)
- Optimization (how to **solve** such learning problems)
- Computer Science (efficient **algorithms** for this)

This intro lecture will focus more on **intuitions** than mathematical details

ML also **overlaps** with multiple areas of engineering, e.g.

- Computer vision
- Natural language processing (e.g. machine translation)
- Robotics, signal processing and control theory

...but traditionally differs by focusing more on **data-driven** models and AI

Why Machine Learning

- Difficulty in **manually programming** agents for every possible situation
- The world is ever **changing**, if an agent cannot adapt, it will fail
- Many argue learning is required for Artificial **General** Intelligence (AGI)
- We are still far from human-level general learning ability...
 - ...but the algorithms we have so far have shown themselves to be useful in a wide range of applications!
 - Using just data, recent “deep learning” approaches can come *near* human performance on many problems, but *near* may not always be sufficient

When Is Machine Learning Useful Today?

- While **not as data-efficient** as human learning, once an AI is “good enough”, it can be cheaply duplicated
- Computers work **24/7** and you can usually **scale** throughput by piling on more of them

Software Agents (Apps and web services)

- Companies collect ever more data and processing power is cheap (“*Big data*”)
- Can **let an AI learn how to improve business**, e.g. smarter product recommendations, search engine results, ad serving, to decision support
- Can **sell services that traditionally required human work**, e.g. translation, image categorization, mail filtering, content generation...?

Hardware Agents (Robotics)

- Although data is more extensive, many capabilities that humans take for granted like **locomotion, grasping, recognizing objects, speech** have turned out to be **difficult to manually construct rules** for.

Example - Google Deepmind's Go Agent

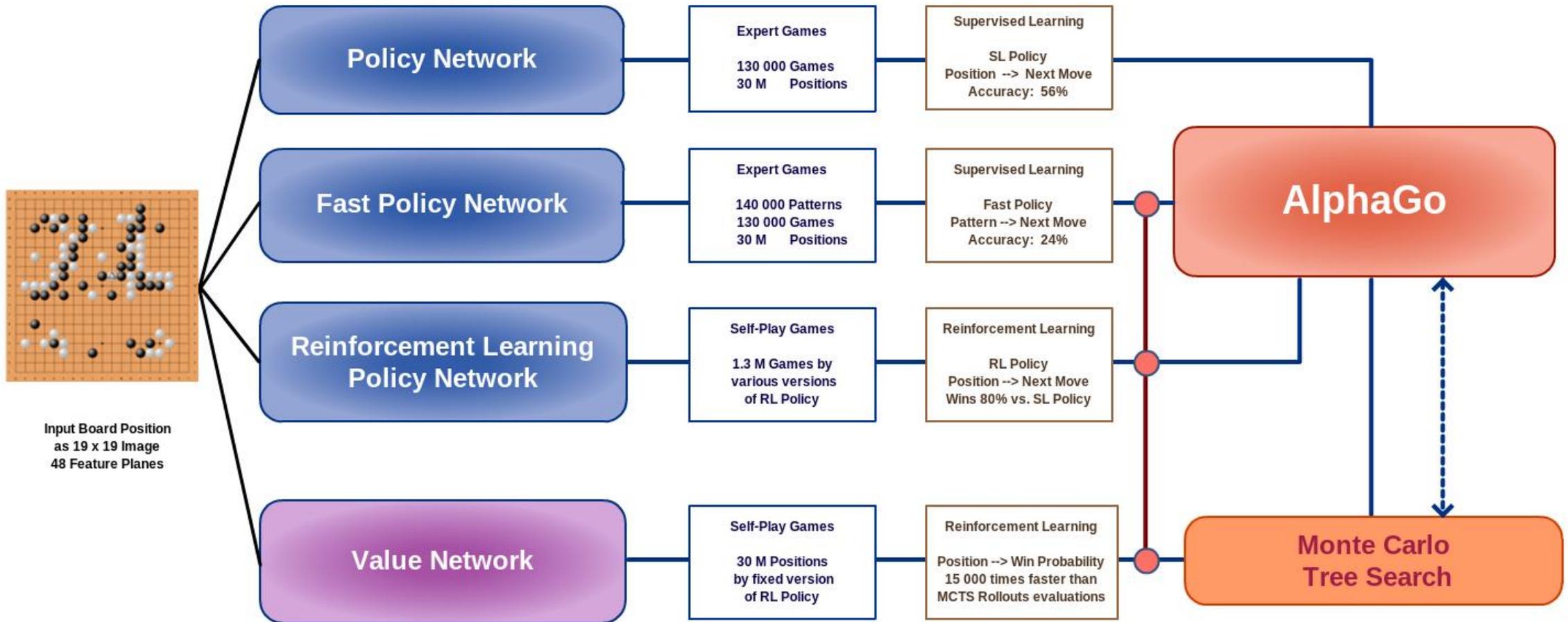
However, in **narrow applications** machine learning can rival or beat human performance.



The image shows a screenshot of a news article from the journal Nature. The article is titled "Google AI algorithm masters ancient game of Go" and is written by Elizabeth Gibney, dated 27 January 2016. The article's subtitle is "Deep-learning software defeats human professional for first time." Below the text, there is a video player with the title "The computer that mastered Go". The video player shows a Go board with black and white stones, and a hand is visible in the bottom right corner, reaching into a bowl of stones. The video player has a progress bar at the bottom, showing a duration of 07:51 and a current time of 00:00.

AlphaGo Overview

based on: Silver, D. et al. Nature Vol 529, 2016
 copyright: Bob van den Hoek, 2016

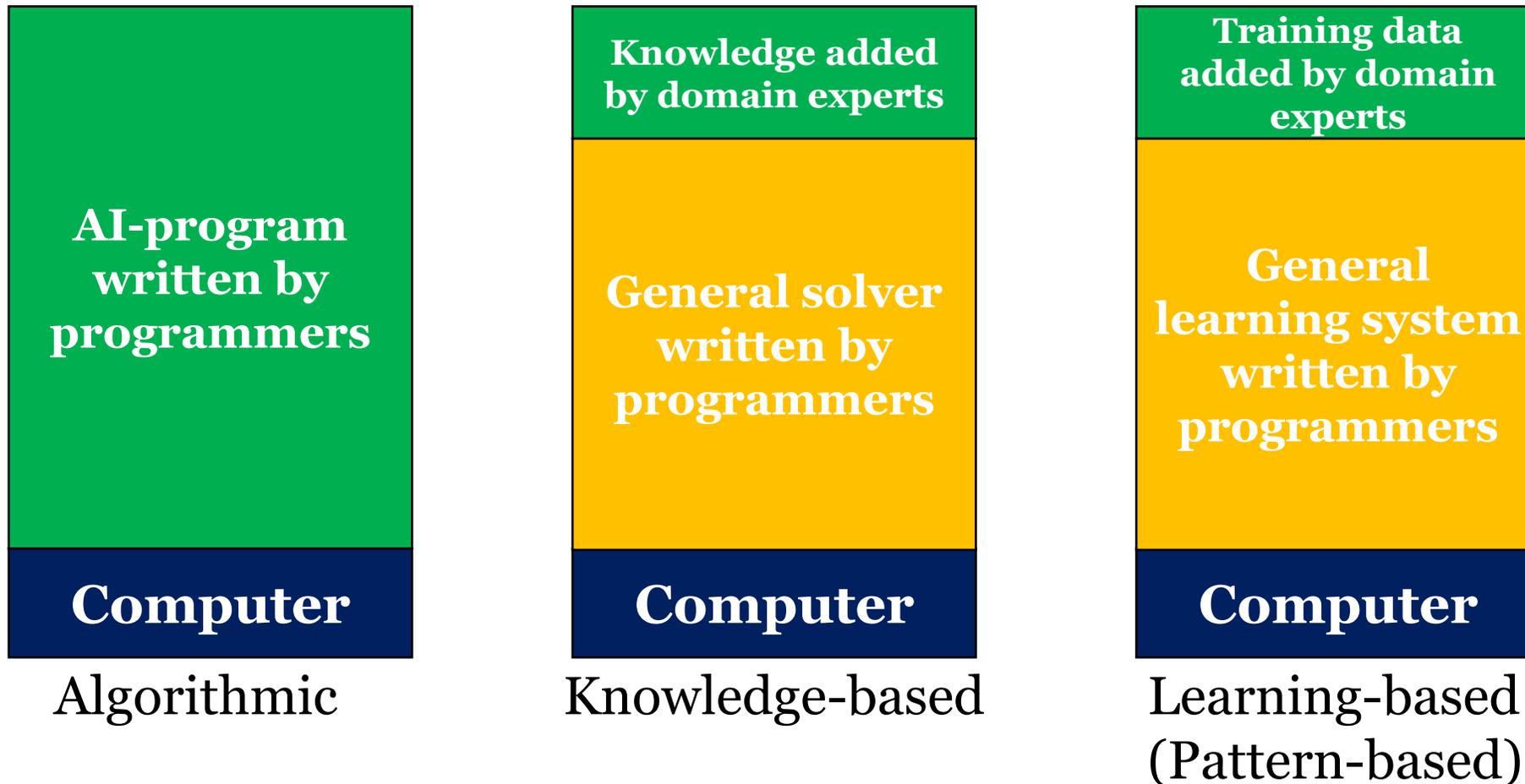


Example - Stanford Helicopter Acrobatics

However, in **narrow applications** machine learning can even rival or beat human performance. This work was done over a decade old but still astonishing.



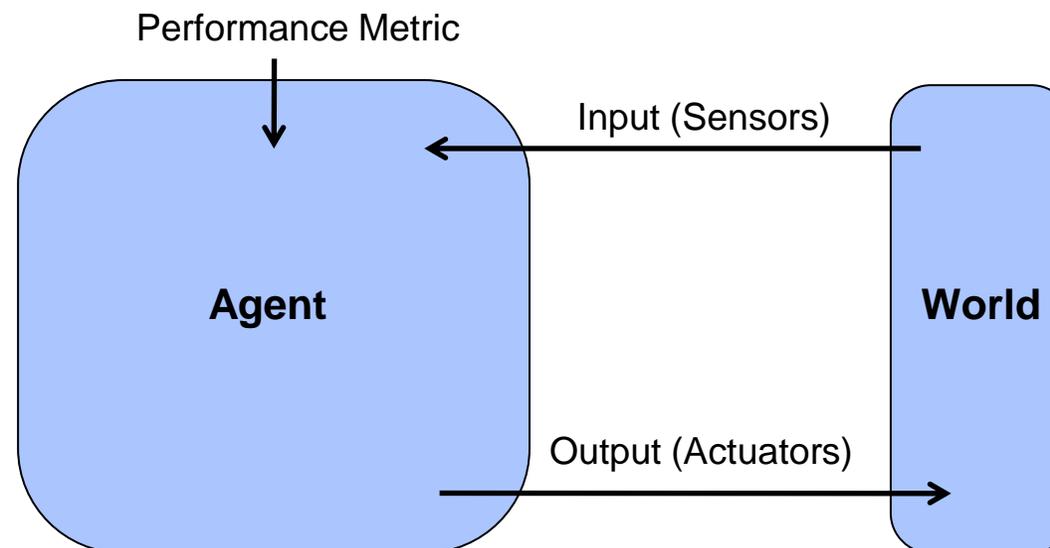
Algorithmic, Knowledge-Based and Learning-Based AI



To Define Machine Learning

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

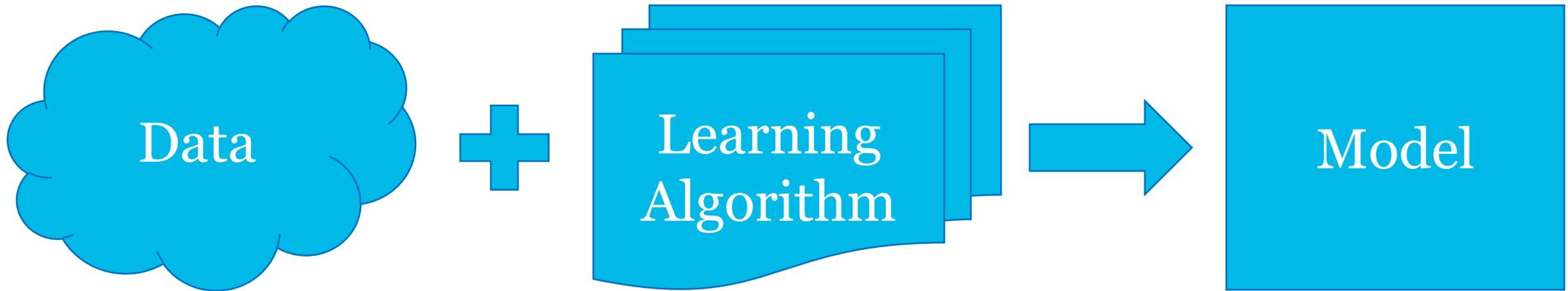
From the agent perspective:



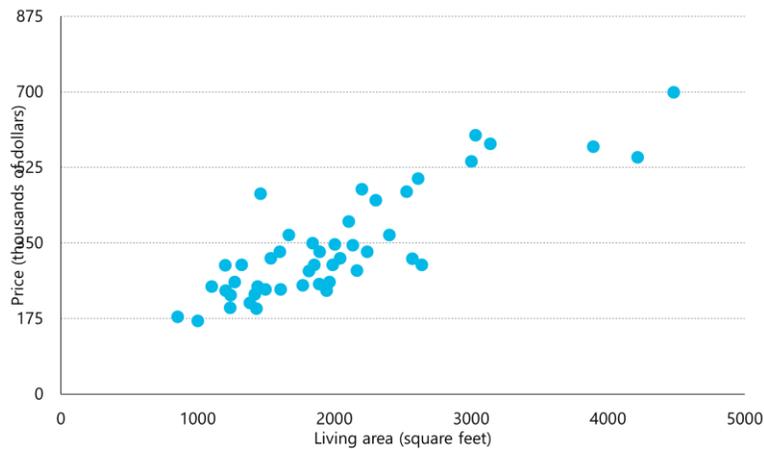
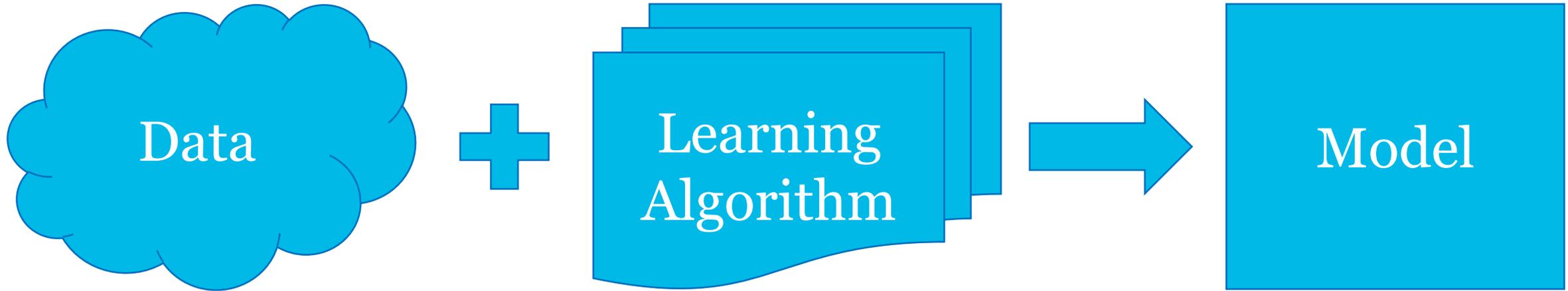
To Define Machine Learning

- 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

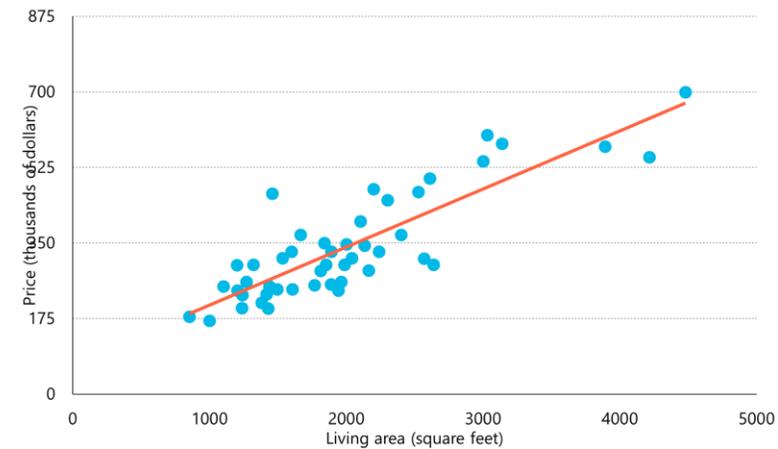


Machine Learning



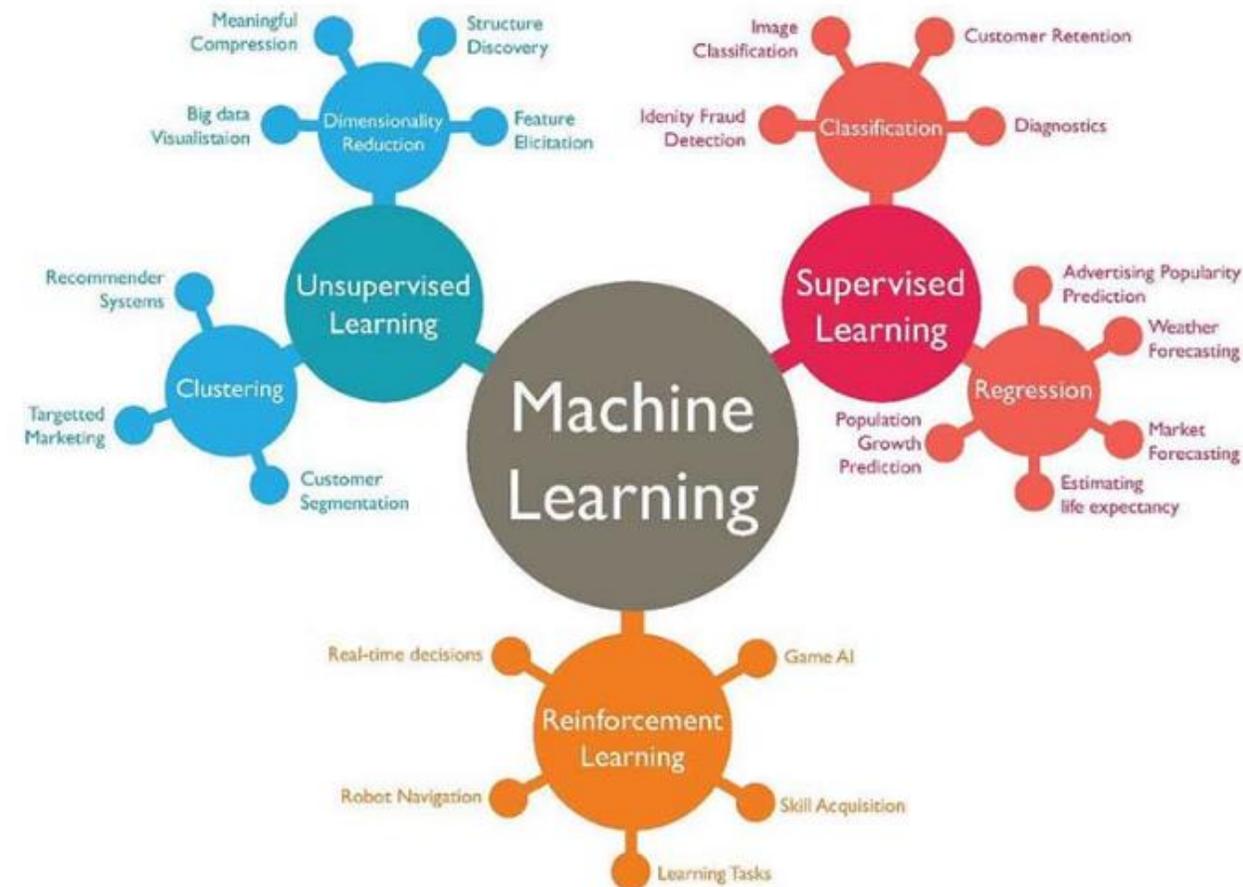
Linear regression

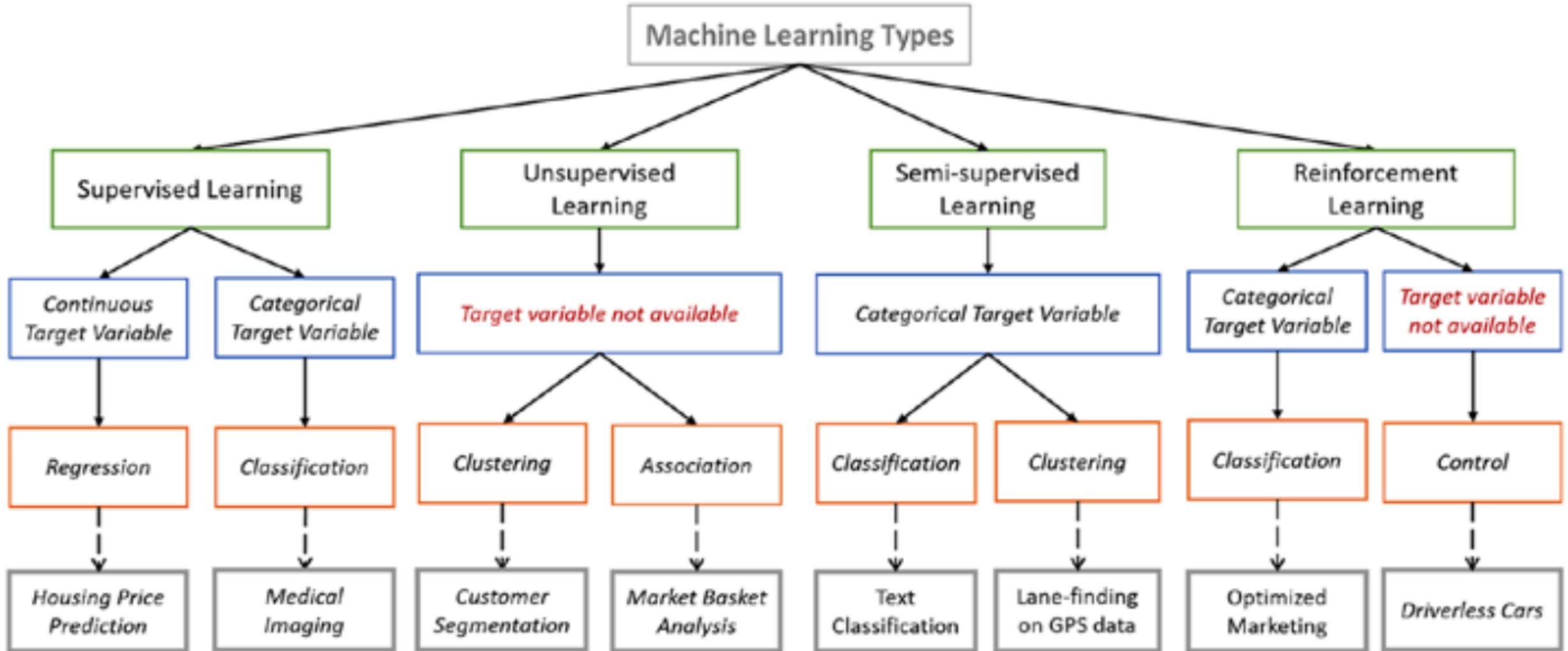
$$\hat{y} = x\theta$$

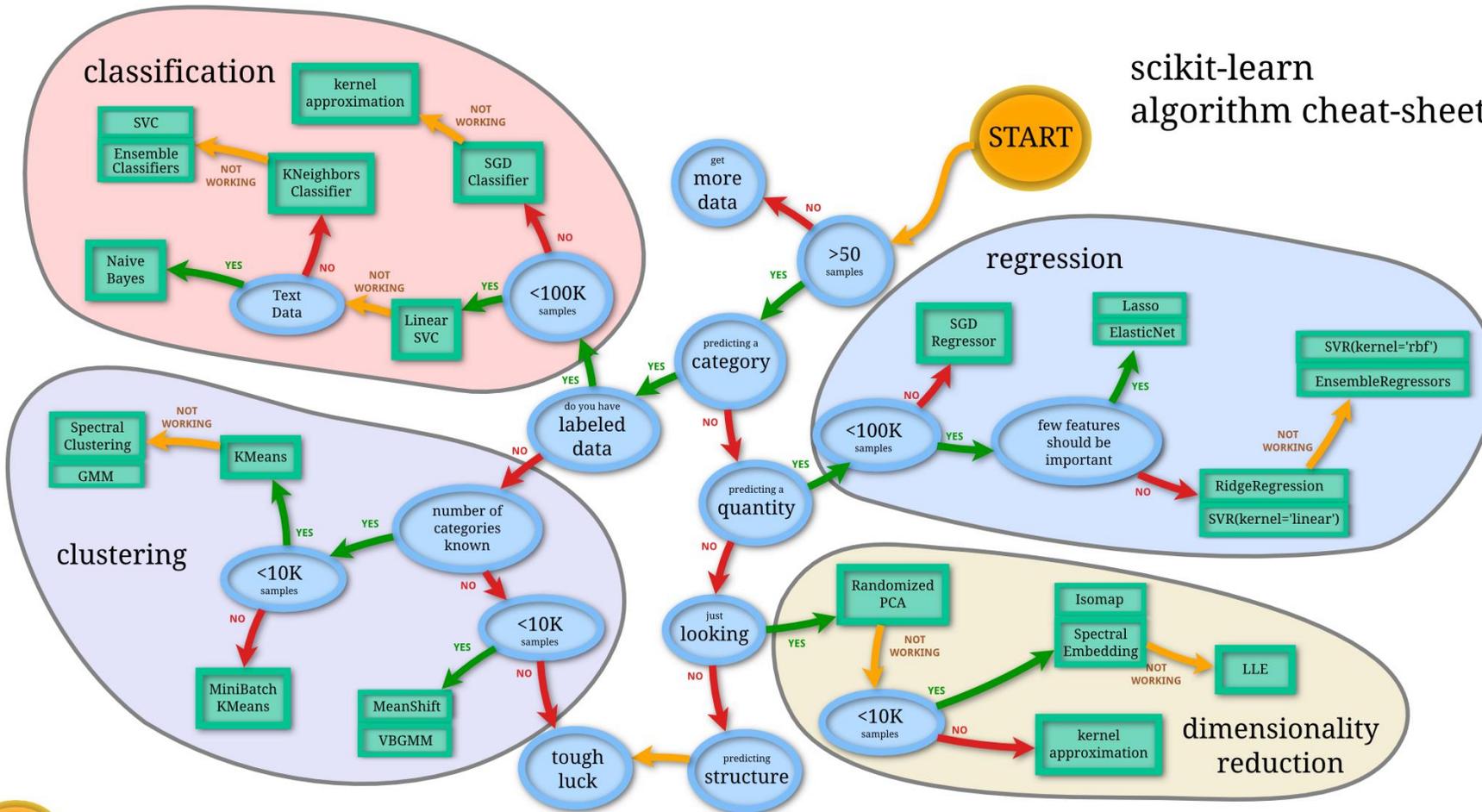


The Three Main Types of Machine Learning

- 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).







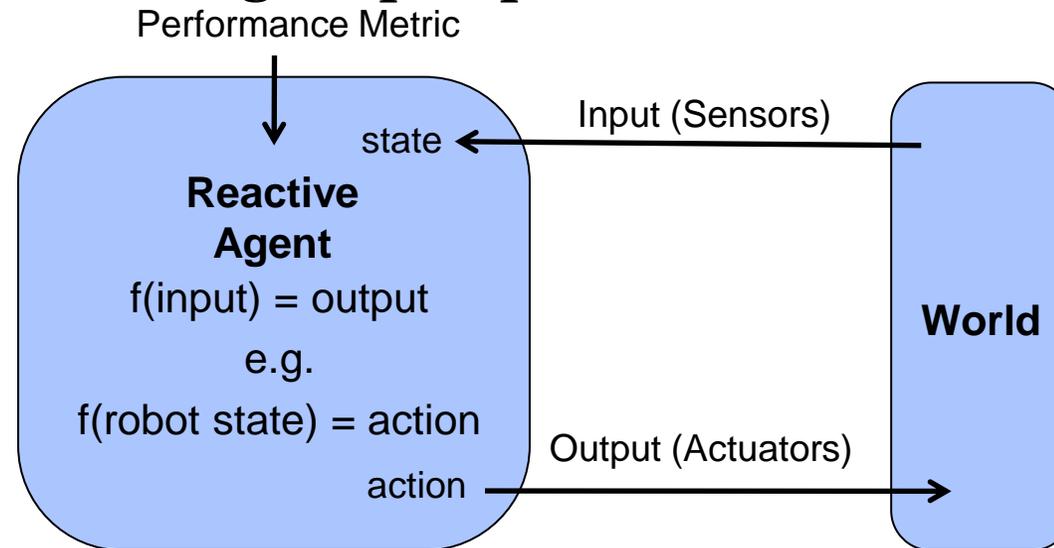
Supervised Learning at a Glance

In supervised learning

- Agent has to learn from **examples of *correct*** behavior
- Formally, **learn an unknown function $f(\mathbf{x}) = \mathbf{y}$** given examples of (\mathbf{x}, \mathbf{y})
- Performance metric: **Loss** (difference) between learned function and correct examples
- Typically classified into:
 - Regression: Predict continuous valued output
 - Classification: Discrete valued output

Supervised Learning - Agent Perspective

Representation from agent perspective:



...but it can also be used as a component in other architectures

- Supervised Learning is surprisingly powerful and ubiquitous
- Some real world examples
 - Spam filter: $f(\text{mail}) = \text{spam?}$
 - Graphics upscaling: $f(\text{pixels}) = \text{pixels}$

Supervised Learning of "Super Resolution"

- Learn $y=f(x)$ from examples (x,y) , ...
 - x = "low-res image", y = "high-res image" (real numbers)
 - Given a new low-res image x' below, predict y' :



- Similar technique ships with NVIDIA graphics cards Deep Learning Super Sampling (DLSS)

Reinforcement Learning at a Glance

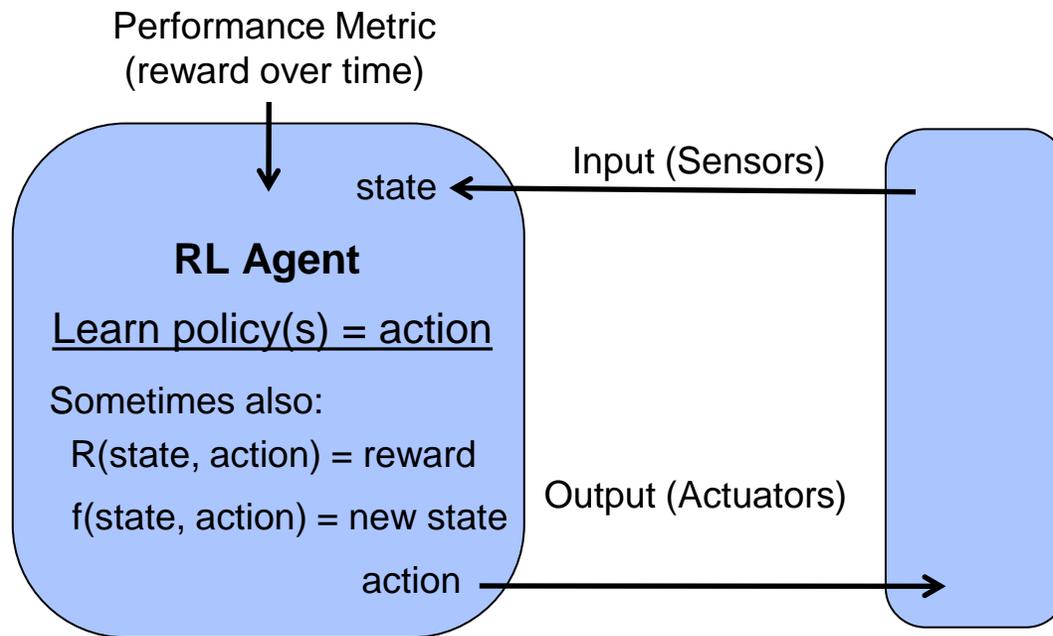
In **reinforcement learning**

- World may have **state** (e.g. position in maze) and be **unknown** (how does an action change the state)
- In each step the agent is only given **current state** and **reward** instead of examples of correct behavior
- Performance metric is sum of **rewards over time**
- Combines learning with a **planning** problem
 - Agent has to **plan a sequence of actions** for good performance
- The agent can **even learn on its own** if the reward signal can be mathematically defined

Reinforcement Learning at a Glance II

RL is based on a utility (reward) maximizing agent framework

- Agent learns *policy* (plan function) to maximize reward over time
- Either learn intermediate models of the effect of actions (next state, reward) from state s , or use *model-free* approaches



Real world examples – Robot Behavior, Game Playing (AlphaGo...)

Supervised vs. Reinforcement Learning for Robot Behavior

- Learning to flip pancakes, "supervised" and reinforcement learning (reward not shown).



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:

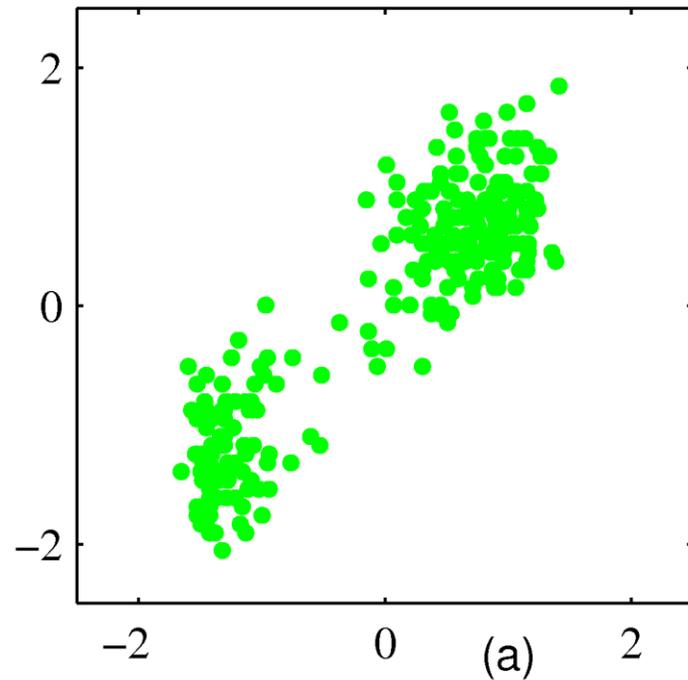
- **Clustering** – When the data distribution is confined to lie in a small number of “clusters” we can find these and use them instead of the original representation, e.g. bigger recommender system (news, ads, etc.)
- **Dimensionality Reduction** – Finding a suitable lower dimensional representation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

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

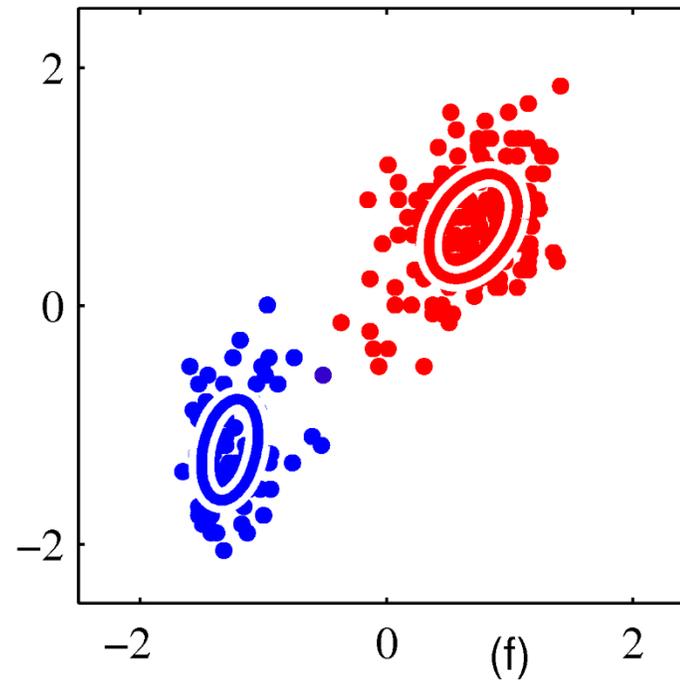
Unsupervised Learning at a glance II

- 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.



(Deep) Unsupervised Learning – Do AI's dream?

- Generative model ("Dream up" new data) fed e.g. images...
- Can we use them to e.g. fill in scenery in a movie scene?



(Karras et al, 2018) <https://youtu.be/G06dEcZ-QTg>

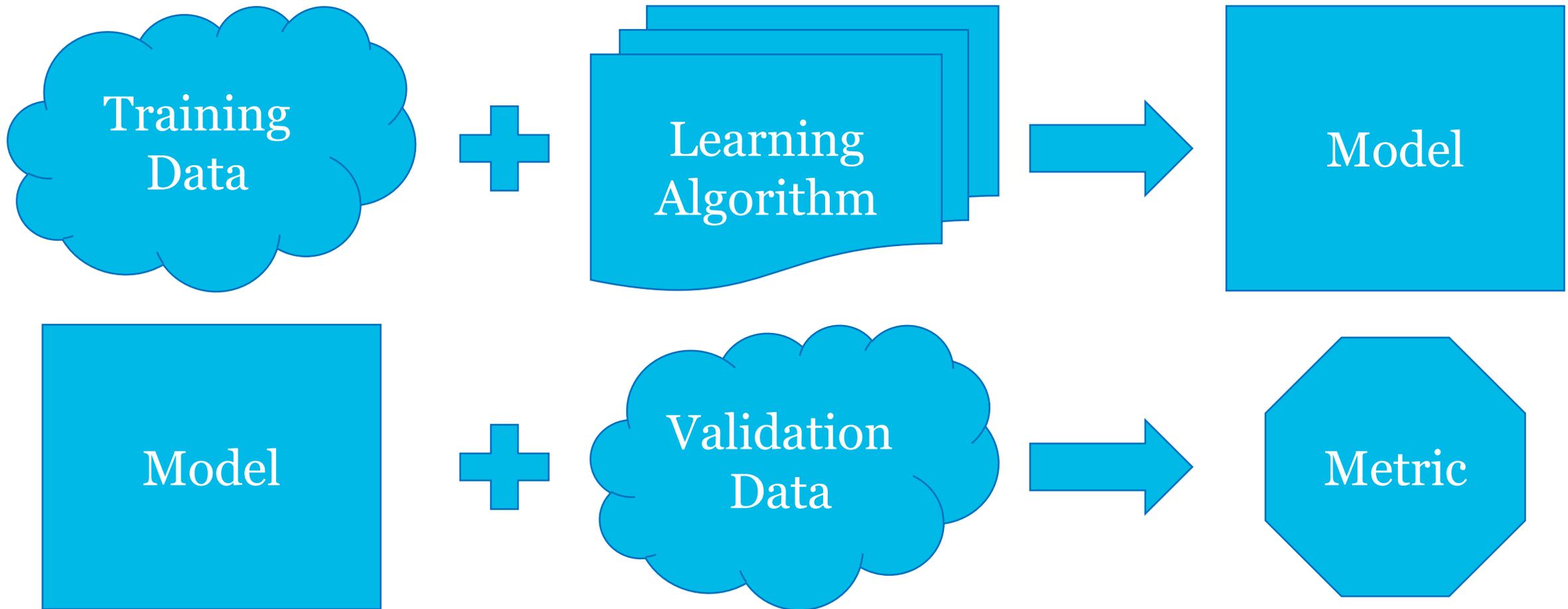
(Deep) Unsupervised Learning – Do AI's Dream?

- Generative model based on text-image data
- Future applications in content generation?

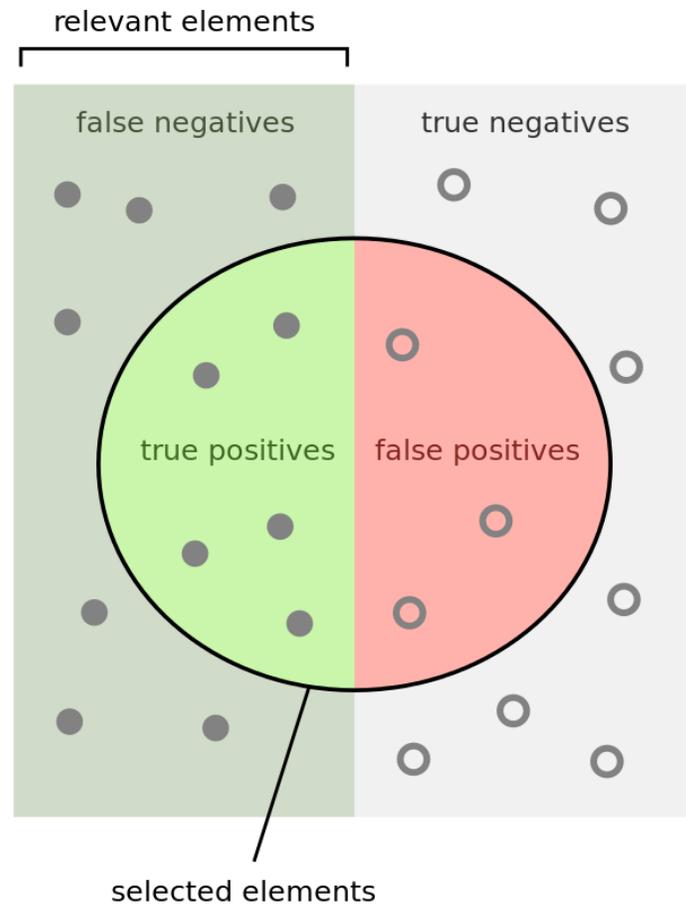
Teddy bears mixing sparkling chemicals as mad scientists in the style of steampunk



Training, Validation, and Test Data



Precision and Recall



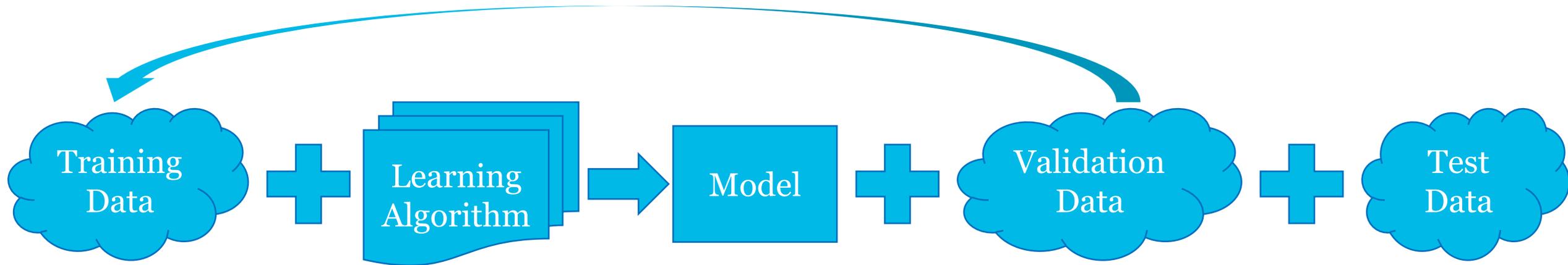
How many selected items are relevant?

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

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

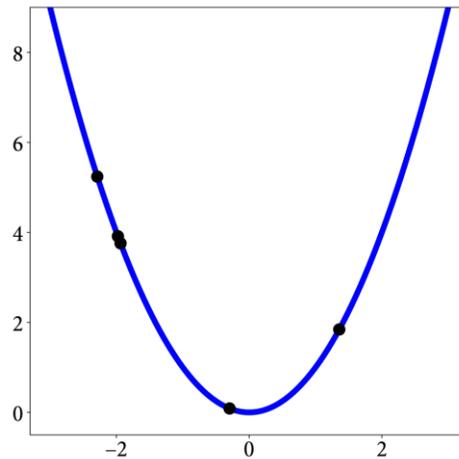
The model is unable to obtain a sufficiently low error on the training set. The model is not expressive enough.

- Overfitting

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

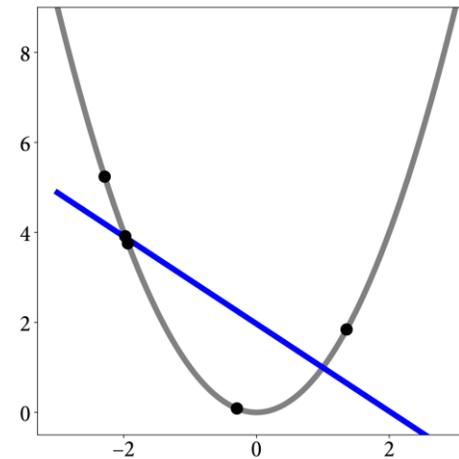
memorises noise

Underfitting and overfitting



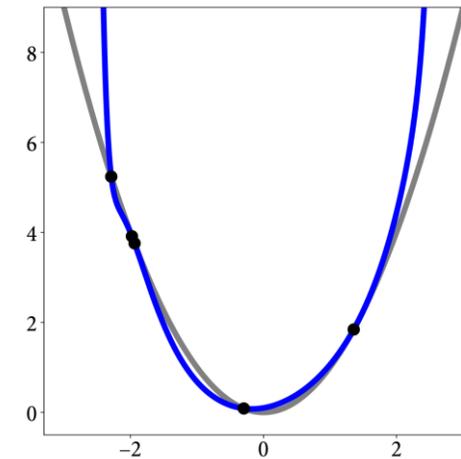
appropriate

polynomial of
degree 2



underfitting

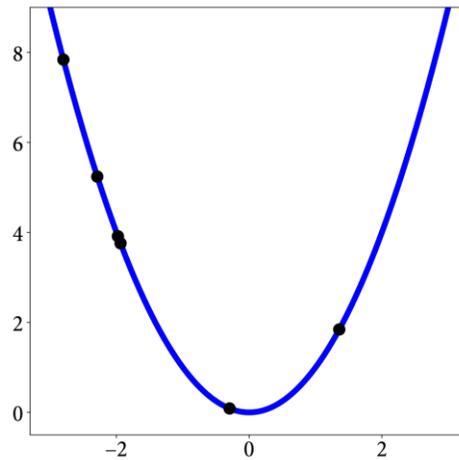
polynomial of
degree 1



overfitting

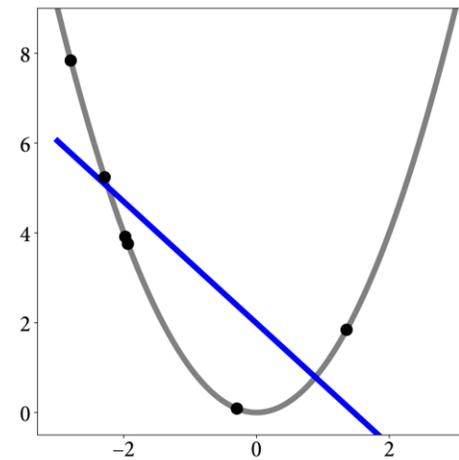
polynomial of
degree 30

Underfitting and overfitting



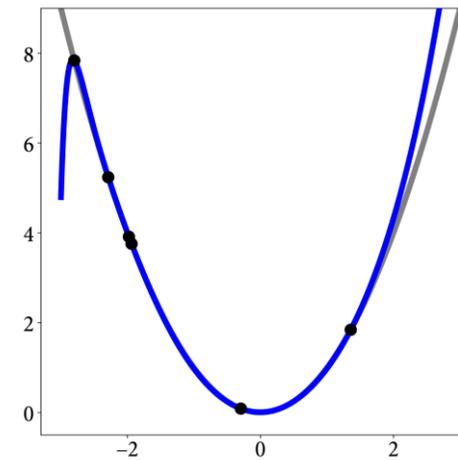
appropriate

polynomial of
degree 2



underfitting

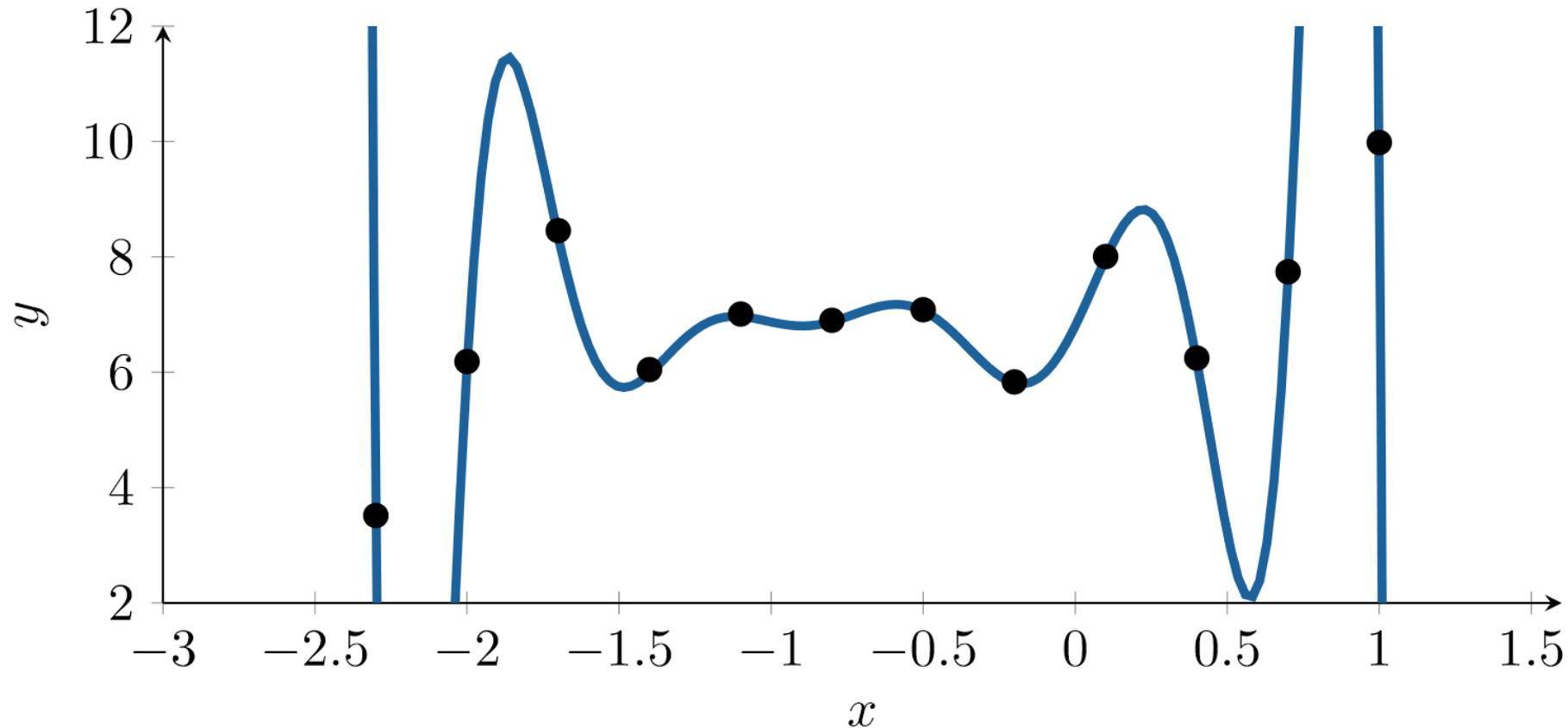
polynomial of
degree 1



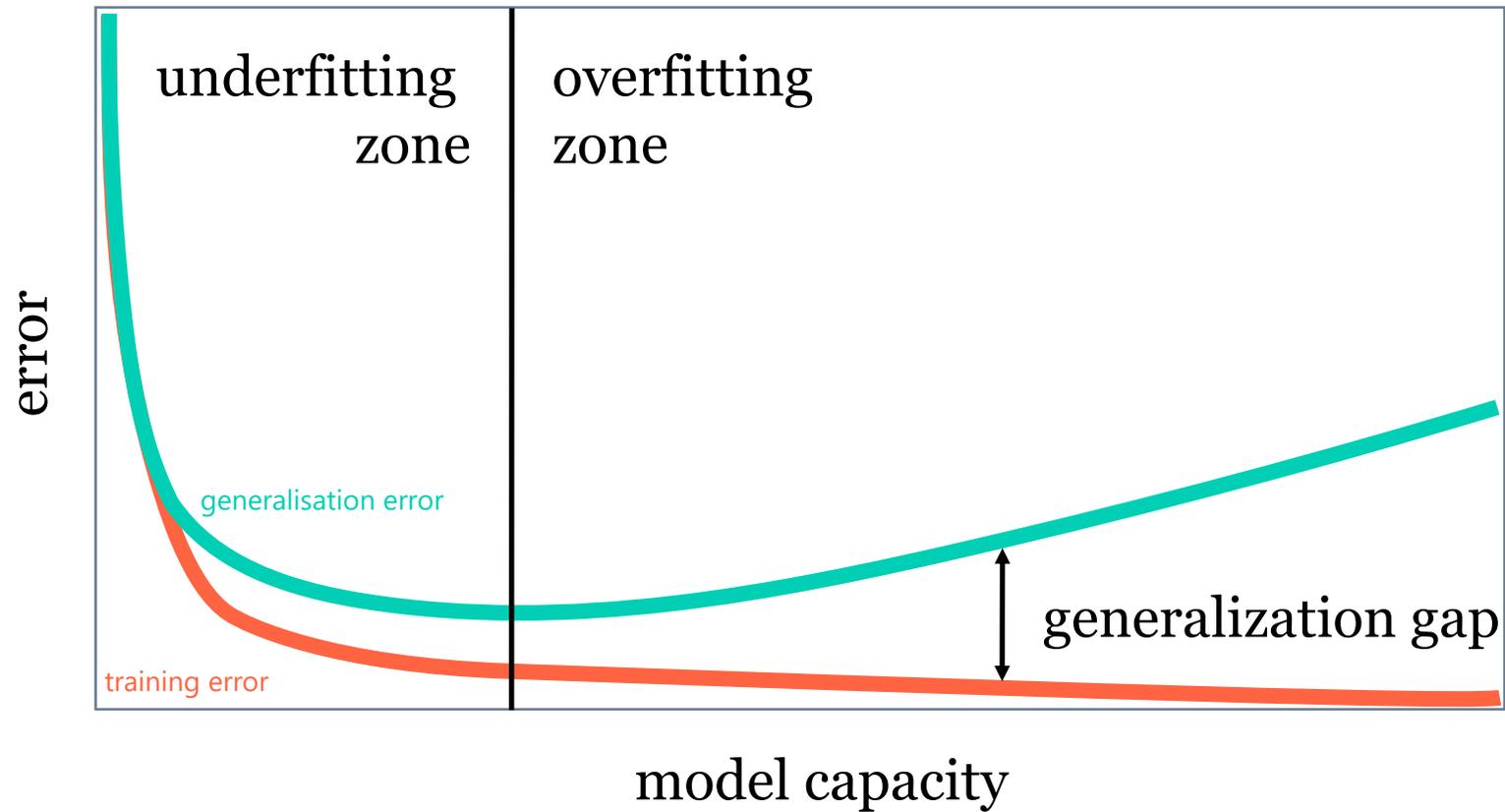
overfitting

polynomial of
degree 30

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



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)

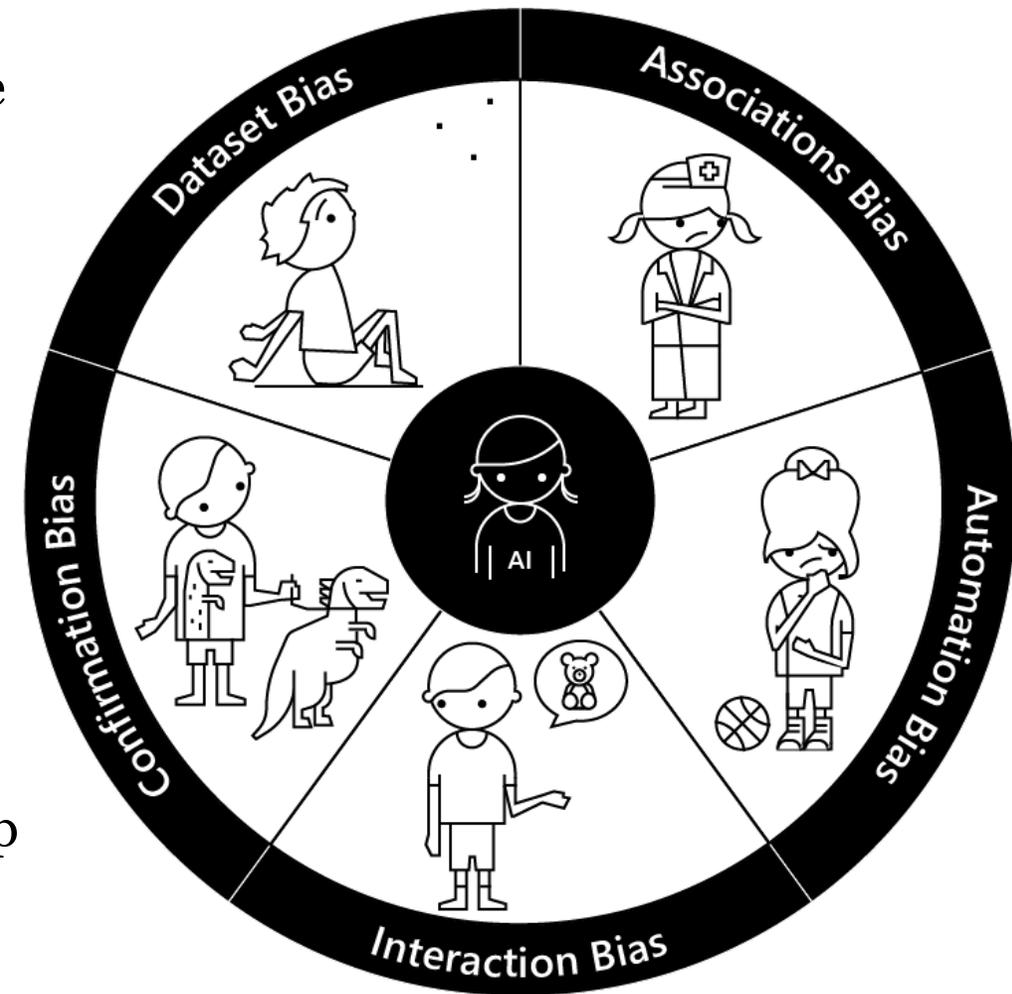
- This means that there is no universal learning algorithm or absolute best learning algorithm.
- We need to make assumptions about the kinds of data-generating distributions we encounter in practice.

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.

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.



Supervised Learning

Outline of Supervised Learning

Today we will focus on **Supervised Learning**

- Definition
- Fundamentals: Features, Models, Loss (or cost) Functions, Training
- Linear Models
- Neural Networks
 - Trend: Deep Learning (more in next ML lecture)
- Pitfalls and Limitations

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 “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: $\mathbf{x}_1 = (12.5, \text{“cat”}, \text{true})$.

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

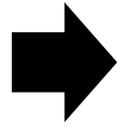
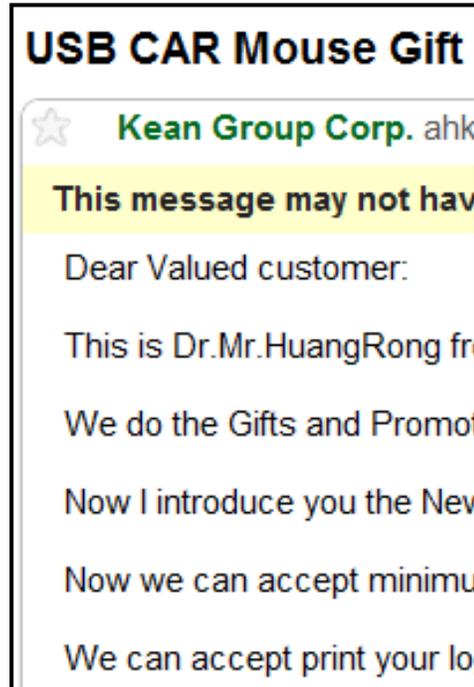
Finding a suitable feature representation **can be non-trivial**, but there are standard approaches for the common domains

- 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
....	...

$x_i = \text{words}_i =$

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

Simply learn $f(x)=y$ using suitable classifier!

Selecting Models: Linear Regression Example

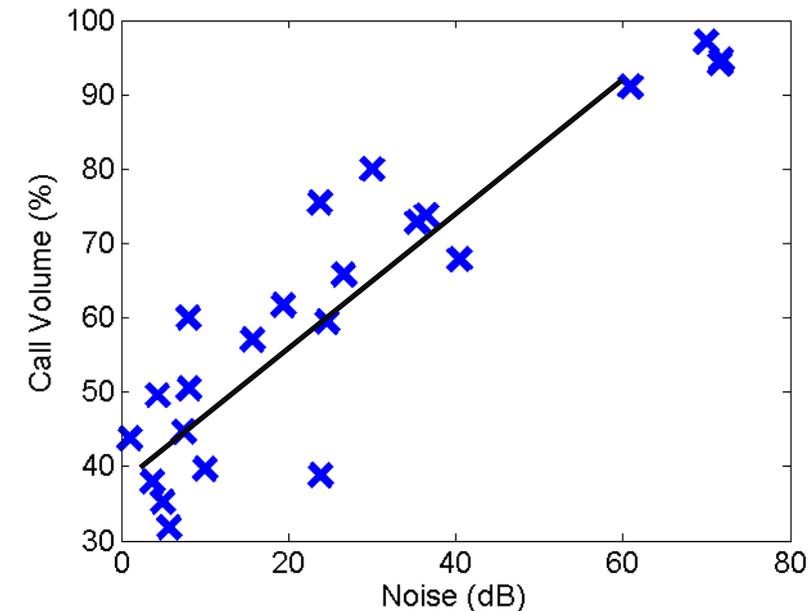
- I. Construct a **feature vector** \mathbf{x}_i to be used with examples of y_i
- II. Select a model and **train** it on examples (search for a good approximation to the unknown function)

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?



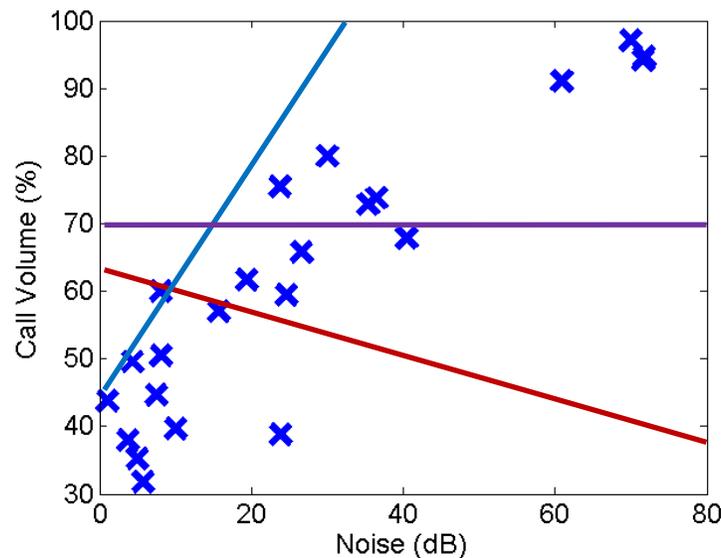
Training a Learning Algorithm

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

- 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:

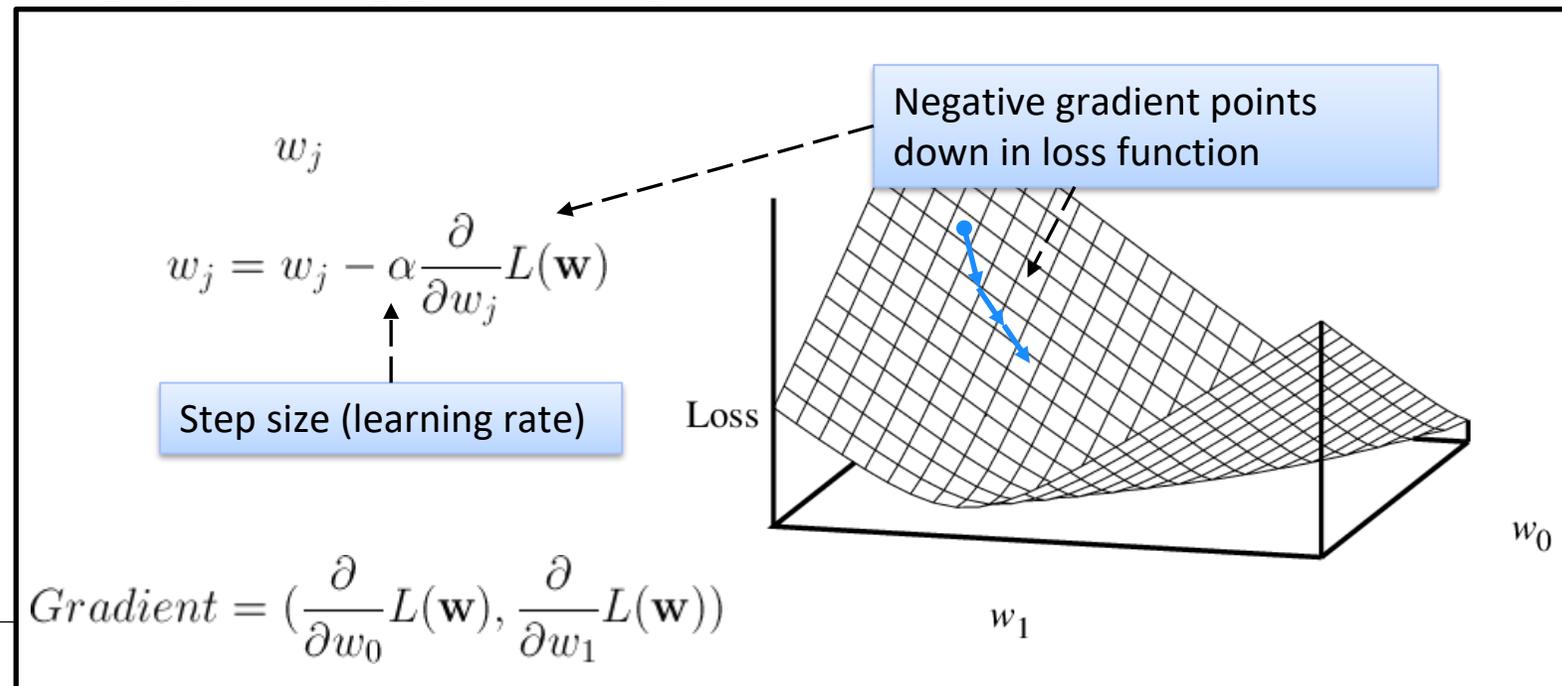
$$L(f(x), h_{\mathbf{w}}(x)) = (f(x) - h_{\mathbf{w}}(x))^2 = \sum_{i=1}^N (y_i - h_{\mathbf{w}}(x_i))^2$$

- 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

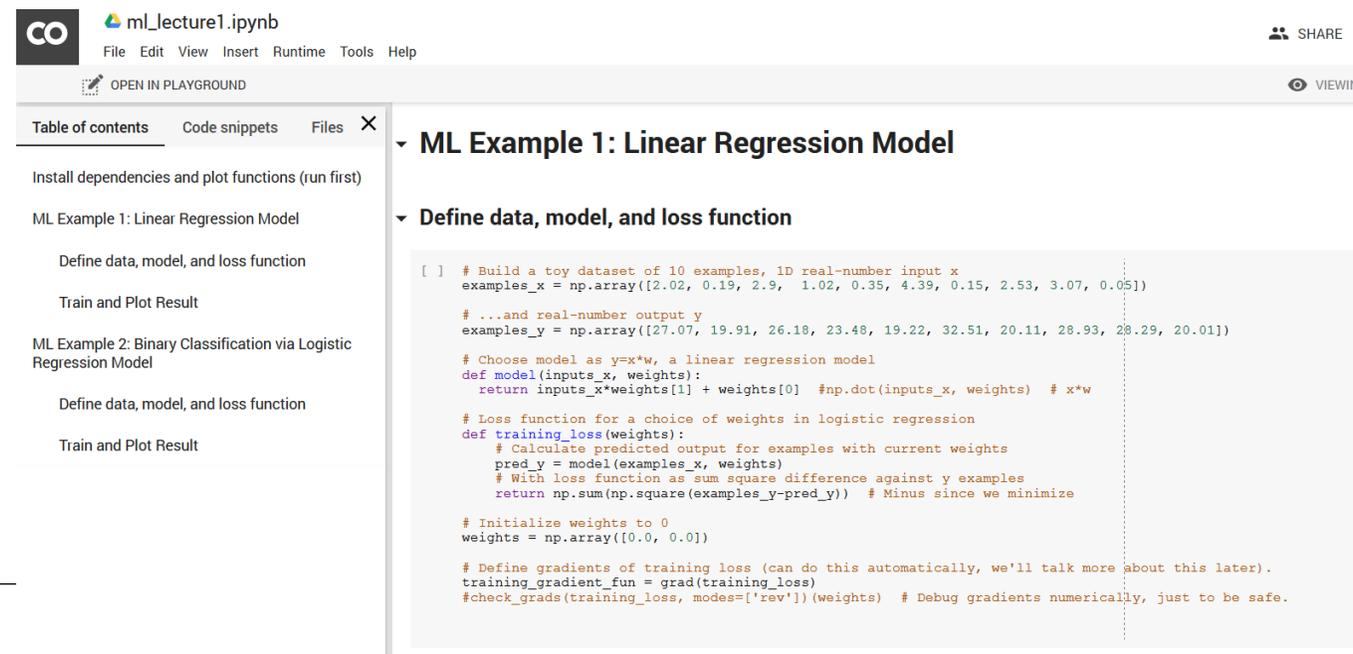
How do we find parameters \mathbf{w} that minimize the loss?

- 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, 1D real-number input x
examples_x = np.array([2.02, 0.19, 2.9, 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, 26.18, 23.48, 19.22, 32.51, 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 loss (can do this automatically, we'll talk more about this later).
training_gradient_fun = grad(training_loss)
#check_grads(training_loss, modes=['rev'])(weights) # Debug gradients numerically, just to be safe.
```

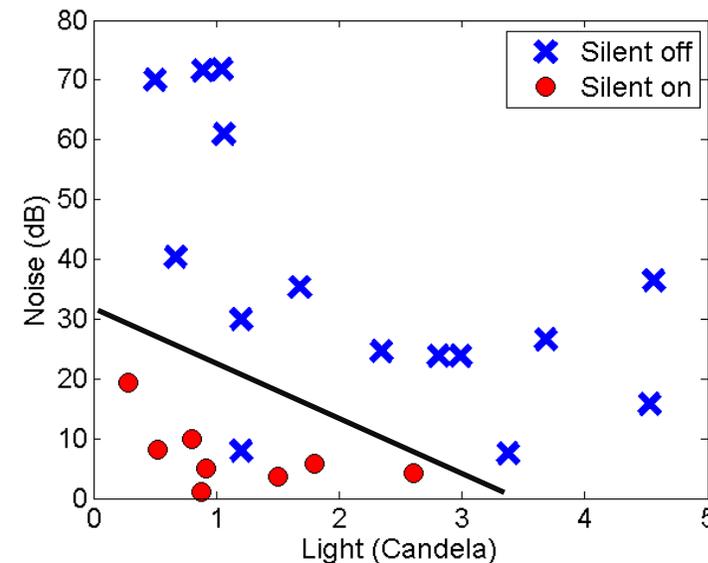
What about categorical outputs (classification)?

- 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 $\mathbf{x}_i = (\text{Noise}, \text{Light level})$, $y_i = \{\text{“silent on”, “silent off”}\}$

- 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

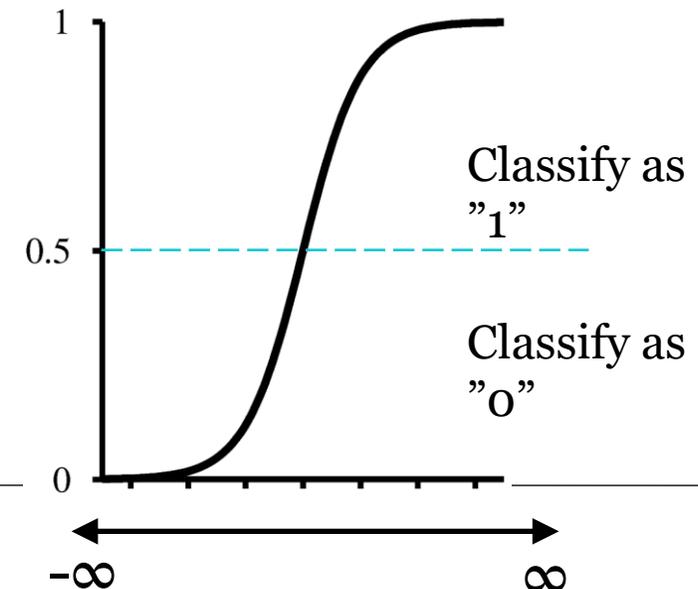
- How to transform standard models to classification?
 - Squared error does not make sense when target output discrete set $\{0,1\}$
- Could use custom loss functions for classification
 - Minimize number of missclassifications (unsmooth w.r.t. parameter changes)
 - Maximize information gain (used in 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 do 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)



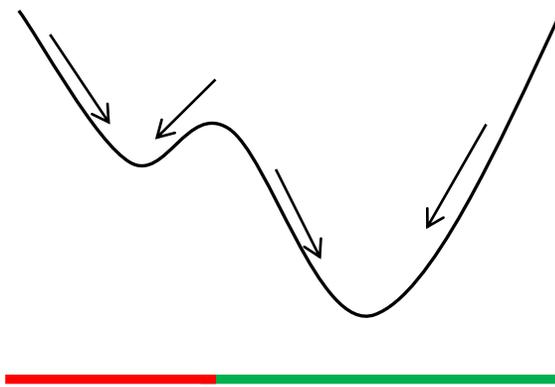
Worked Example - Binary Classification via Linear Logistic Regression

- 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...

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 LE5 HT2023:
Introduction to machine learning
Unsupervised learning
Supervised learning

www.ida.liu.se/~TDDC17